A graphical notation for specifying MIB views

Arne Øslebø
Department of telematics
Norwegian University of Science and Technology
N-7491 Trondheim, Norway
Email: Arne.Oesleboe@item.ntnu.no

Abstract

SNMPv3 supports authentication, confidentiality and access control. Access control is based on MIB views, which makes it possible to have fine grained control over who accesses what in the MIB. Different users can be given different access rights. However, as the number of users increase, it becomes very difficult to keep control over all the different access rights. MIB View Modeling Language is a graphical language that provides a tool for easy specification of MIB views. It is a hierarchical language that provides mechanisms for supporting a large number of different users and MIB views. An algorithm, MVML Translation Algorithm, translates the graphical diagrams into MIB values that can be used to configure the managed entities.

1 Introduction

Simple Network Management Protocol (SNMP) is the most commonly used protocol for management of network devices in TCP/IP networks. The first version of SNMP was introduced in 1990. Today virtually all network devices support it, and it is also used by many printers, applications and work stations. It is however, almost exclusively used for monitoring devices and rarely used for configuration. The reason for this is that the first version did not have sufficient security functionality.

The second version of SNMP (SNMPv2) was supposed to address shortcomings of the first version, especially security issues. Unfortunately this effort failed, and instead SNMPv2c was released which kept the same weak security method used in SNMPv1.

SNMPv3 is the first SNMP version that supports strong security. The use of strong security and access control makes it possible to use SNMP for configuring devices in a secure way. The full SNMPv3 standard can be found in [6], [5], [7], [8], [4] and [3].

One reason for the popularity of SNMP is its simplicity. SNMPv3 implements strong security and this adds complexity. The access control in SNMP consists of a number of tables that together decides who has access to what. The problem is that as the number of users increase, it quickly becomes difficult to manage all the entries in these tables. Little work have been done to make this easier. There are some commercial products, for example [2], that provides a graphical user interface for manipulating the tables. This however is not enough since the user still has to manually keep control over all the entries in the tables.

This paper describes a new approach for making access control in SNMPv3 simpler to use. The MIB View Modeling Language (MVML) is a graphical language for specifying access rights in SNMPv3. It is designed with ease of use as the primary goal. MVML distances the users from the access control tables in SNMPv3 and hence makes it easier to use.

MVML can be used to directly configure the access control in managed entities[1]. This is done by MVML Translation Algorithm (MTA) which translates the graphical notation into MIB[2] values that are used to configure the managed entities. See Figure 1.

MVML is also suited to be used as a tool for documentation. Developers of MIBs that have advanced access control needs, can use MVML as a visual aid to document a recommended configuration of access rights.

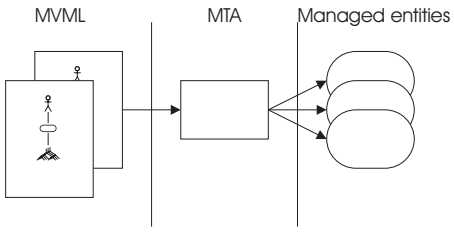

Figure 1: MVML and MTA

2 SNMPv3 Security

The security mechanisms in SNMPv3 are divided into two main parts, User-based Security Model (USM) and View-based Access Control Model (VACM). USM takes care of authentication, timeliness and privacy. VACM assumes authentication has already been performed and concentrates solely on access control.

2.1 USM

USM has support for multiple authentication and privacy protocols. Common for all the protocols is that they all use secret keys. A managed entity needs secret keys for all users accessing it. These keys can be changed remotely through SNMP. Although there is a different key for each managed entity, a user only has to remember one password. This password, together with the engine ID[3] of the managed entity, is used to derive a unique secret key for each entity the user has access to.

USM defines a new MIB, snmpUsmMIB. This MIB defines a table, usmUserTable, that contains information about all the users. When a new user is created, a new entry in this table will be added. All new users are cloned from an existing user.

For a detailed description of USM see [4].

[1] SNMPv3 uses the word entity for both agent and manager.

[2] Management Information Base. Collection of management information in a managed entity

[3] unique ID for a SNMP entity within an administrative domain.

2.2 VACM

VACM takes care of the access control. Access control in SNMP means controlling the managed objects that users can access. It is based on MIB views. A MIB view is a subset of the entire MIB in an entity. It is possible to have a different MIB view for each user, however it is more common to create groups with different access rights and then make users with the same access rights members of these groups. There can be separate MIB views for the different message types[4], security levels[5] and security models[6].

MIB views are built using a list of MIB subtrees. Each of these subtrees can either be included or excluded from the view. When a managed entity processes a set, get or notify operation it always checks with VACM to see if the operation is permitted. VACM checks the subtrees, and if the object in question is included in the MIB subtrees, the operation is permitted.

VACM defines a new MIB, snmpVacmMIB, that has four tables that stores all information needed for access control. The tables are vacmContextTable, vacmSecurityToGroupTable, vacmAccessTable and vacmMIBViews. Table vacmContextTable is a read only table that contains locally available contexts. Tables 1, 2 and 3 show the most relevant fields of the three latter tables.

Object	**Description**
vacmSecurityModel	security model used.
vacmSecurityName	security name that is security model independent
vacmGroupName	name of group this entry belongs to.

Table 1: vacmSecurityToGroupTable. Maps a security model and security name to a group name.

Object	**Description**
vacmAccessContextPrefix	name of collection of management information
vacmAccessSecurityModel	security model used
vacmAccessSecurityLevel	security level used
vacmAccessContextMatch	specifies how ContextPrefix should be matched (exact or prefix)
vacmAccessReadViewName	name of read view
vacmAccessWriteViewName	name of write view
vacmAccessNotifyViewName	name of notify view

Table 2: vacmAccessTable. Maps a group name, security model, security level, context and message type into a MIB view. Also uses vacmGroupName as index.

For a detailed description of VACM see [3].

[4]read, set and notify.

[5]no authentication and no privacy, authentication and no privacy, authentication and privacy.

[6]SNMPv1, SNMPv2c and USM

Object	Description
vacmViewTreeFamilyViewName	name for a family of subtrees.
vacmViewTreeFamilySubtree	portion of the MIB tree that ViewName refers to.
vacmViewTreeFamilyMask	used to control which elements of the Subtree should be regarded as relevant when determining which view an OID is in. Each bit in the mask corresponds to an element. A 1 indicates exact match and a 0 indicates a wild card.
vacmViewTreeFamilyType	type of view (include or exclude).

Table 3: vacmViewTreeFamilyTable. Contains the MIB views and is used to determine if an object is allowed to be accessed.

3 MVML Notation

Most of the symbols defined in MVML have attributes. The syntax of the attributes is specified using ASN.1 notation [1]. Attributes whose names are *emphasized* are optional attributes. The rest are mandatory.

MIB views in MVML are drawn inside one or more diagrams and they are drawn using a tree structure. If a node has any children, the children can be drawn either under the parent node in the same diagram or to avoid cluttering, in a new diagram associated with the parent node. If a new diagram is being used, this diagram should have a tree structure that starts with the symbol of the parent node as the root node. All MVML symbols except scalar object can have a diagram associated with it.

Figure 2 shows all the symbols and relations defined in MVML.

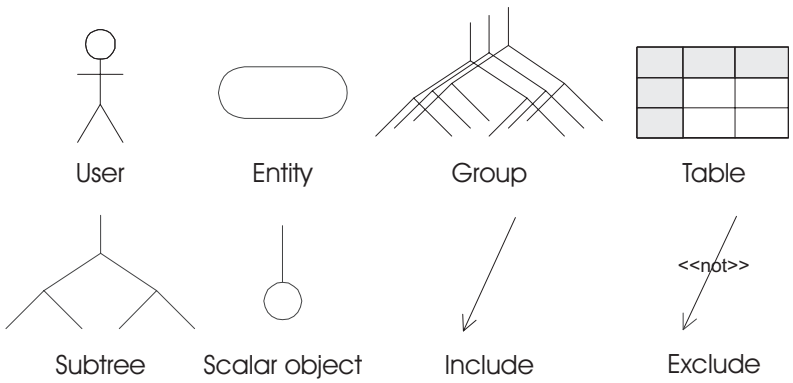

Figure 2: MVML symbols

3.1 Main diagrams

There can be multiple diagrams at the top level. Each diagram specifies the operations, security models and security levels the MIB view is associated with.

Attributes

DiagramName name of diagram. No formal meaning.

```
DiagramName ::= PrintableString
```

AccessType type of access for the MIB view.

```
AccessType ::= "Read" | "Write"| "Notify" | "All" | AccessType "," AccessType
```

SecurityModel security model of the MIB view.

```
SecurityModel ::= Model | Model "," SecurityModel
Model ::= INTEGER { any(0), SNMPv1(1), SNMPv2c(2), USM(3) }
```

SecurityLevel security level of the MIB view.

```
SecurityLevel ::= Level | Level "," SecurityLevel
Level ::= INTEGER { any(0), noAuthNoPriv(1), authNoPriv(2), authPriv(3)
}
```

ContextPrefix context prefix of the MIB view.

```
ContextPrefix ::= PrintableString
```

ContextMatch context match of the MIB view.

```
ContextMatch ::= "Exact" | "Prefix"
```

3.2 User

The user symbol represents one or more users who can access a managed entity. If one user is related to another user, the first user will include or exclude the access rights of the second user in its own access rights.

Attributes

UserName name of user(s). No formal meaning.

```
UserName ::= PrintableString
```

LoginName login name for user(s) when connecting to an entity.

```
LoginName ::= PrintableString | PrintableString "," LoginName
```

CloneFrom name of template user. Only used when the user(s) is first created.

```
CloneFrom ::= PrintableString
```

Password initial password. Only used when the user is first created. User can change this after the user has been created. Should be one password for each LoginName.

```
Password ::= PrintableString
```

3.3 Entity

The entity symbol represents one or more managed SNMP entities and specifies which IP addresses to be used for table, subtree and scalar object symbols. It does not represent the entire MIB of the managed entity. In a tree belonging to an entity symbol, there can not exist any other entity symbols. There has to be at least one entity in a tree belonging to a user.

Attributes

EntityName name of entity. No formal meaning.

```
EntityName ::= PrintableString
```

EntityAdr IP address of the entities

```
EntityAdr ::= IMPLICIT OCTET STRING (SIZE (4)) | IMPLICIT OCTET STRING
(SIZE (4)) "," EntityAdr
```

3.4 Group

The group symbol represents a collection of one or more other elements. There can be multiple instances of the same group in a diagram, but only one instance can have any children. This is where the content of the group is defined. The other instances only refer to this definition.

Attributes

GroupName name of group. No formal meaning.

```
GroupName ::= PrintableString
```

3.5 Table

The table symbol represents a SNMP table. Accessible columns in a table are specified using subtree symbols related to the table. Each subtree symbol specifies one column that should be included. No subtree symbols defaults to full access. If scalar object symbols are used instead of subtree symbols, access to a specific cell in the table is granted. To give access to all elements in the row no scalar object symbols are necessary, but then the attribute index has to be set.

Attributes

TableName name of table. No formal meaning.

```
TableName ::= PrintableString
```

TableOID Object Identifier for the table.

```
TableOID ::= OBJECT IDENTIFIER
```

TableIndex index used to specify a specific row in a table.

```
TableIndex ::= OBJECT IDENTIFIER
```

3.6 Subtree

A subtree symbol represents a subtree node in a MIB. Drawing the nodes belonging to the subtree is optional.

Attributes

SubTreeName name of subtree. No formal meaning.

```
SubTreeName ::= PrintableString
```

SubTreeOID Object Identifier for the subtree.

```
SubTreeOID ::= OBJECT IDENTIFIER
```

3.7 Scalar object

A scalar object symbol represents a scalar object in a MIB.

Attributes

ScalarObjectName name of scalar object. No formal meaning.

```
ScalarObjectName ::= PrintableString
```

ScalarObjectOID Object Identifier for the scalar object

```
ScalarObjectOID ::= OBJECT IDENTIFIER
```

3.8 Include

The include relation is used to show that a node should be included in the access rights.

3.9 Exclude

The exclude relation is used to show that a node should be excluded from the access rights.

4 Hierarchy

MVML uses a tree structure to specify MIB views. It is therefore possible to get ambiguities where access to MIB objects are granted in one place and then denied in another. This is resolved by stating that rules at a higher level in the hierarchy take precedence over rules defined at a lower level. If there are ambiguities at the same level, exclude should take precedence over include.

Subtree, table and scalar object symbols only have an object identifier that specifies the location in a MIB. An entity symbol is used to specify the IP address. Subtree, table and scalar object symbols that are not related to any entity, will be included or excluded from all entities on the same level and lower levels in the hierarchy.

5 Unified Modeling Language

The Unified Modeling Language (UML) [9] is a standardized notation designed primarily for describing object oriented software designs. It is also possible to draw MVML diagrams using UML. The MVML diagrams are drawn inside UML class diagrams. All MVML symbols are stereotypes of a UML package. The include relation is a normal UML dependency arrow and exclude is a stereotype of an UML dependency arrow.

The advantage of using UML is that there are commercial modeling tools available that can be used to draw the MVML diagrams.

6 Example

Figure 3 and Figure 4 shows an example of how MVML can be used to specify the access rights of four users. This example tries to show some of the intricacies of MVML and is not meant as an example in efficient use of the language.

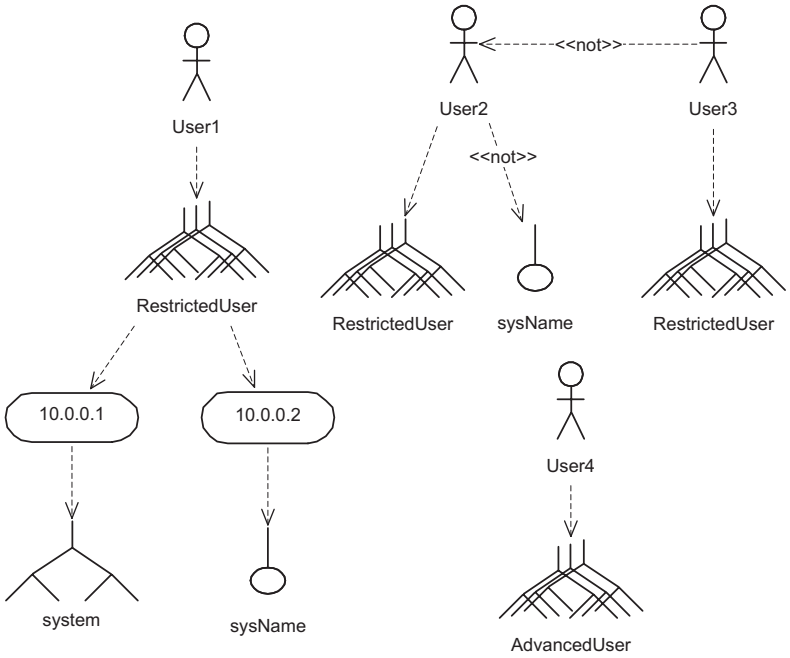

Figure 3: Main diagram

This example has two managed entities, 10.0.0.1 and 10.0.0.2. Both have a very simplified MIB. The only available managed objects are a subset from the iso.org.dod.internet.mgmt.mib-2.system group. Table 4 shows the access matrix for the users. This table also shows the available managed objects in each entity. A shorthand notation shown in parentheses will be used to represent the entities and managed objects.

Entity	MIB object	User1	User2	User3	User4
10.0.0.1 (A)	sysDescr (x_1)	•	•	-	•
	sysUpTime (x_2)	•	•	-	•
	sysContact (x_3)	•	•	-	•
	sysName (x_4)	•	-	•	-
10.0.0.2 (B)	sysName (x_4)	•	-	•	•
	sysORUptime (x_5)	-	-	-	•
	sysORId (x_6)	-	-	-	•
	sysORDescr (x_7)	-	-	-	•

Table 4: Access matrix

Each user will have two sets, A_i and B_i where i is the user number, which represents the accessible managed objects in entity 10.0.0.1 and 10.0.0.2 respectively. A bottom up approach is used to determine the contents of A and B. On each level in the hierarchy, the included managed objects are determined first and then the ones that are excluded are removed.

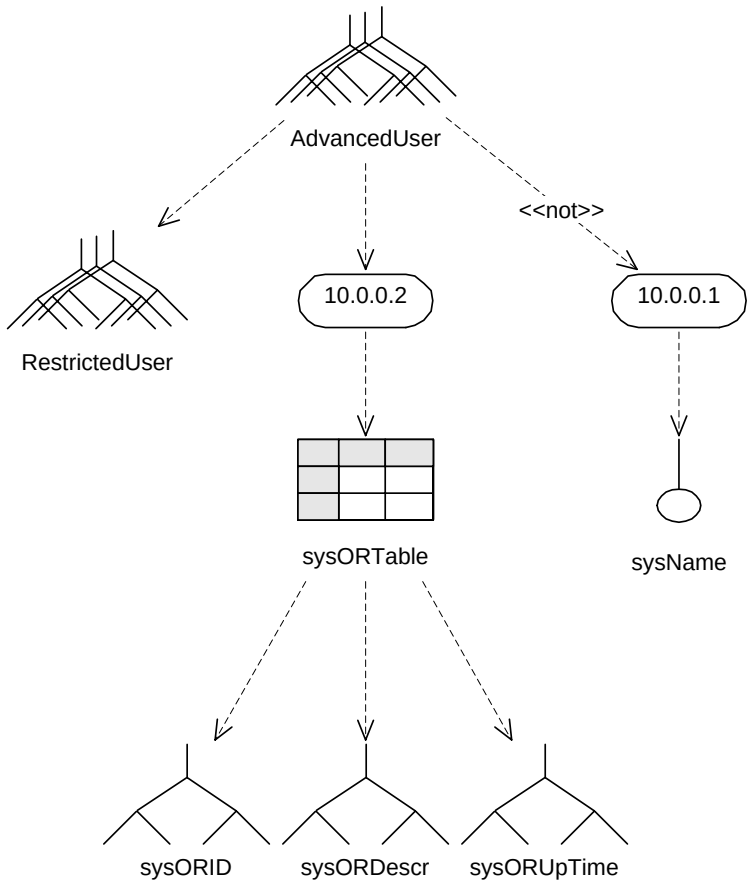

Figure 4: Diagram for AdvancedUser

6.1 User1

It is straight forward to find the access rights of User1. User1 includes the access rights of group RestrictedUser. RestrictedUser includes access to the entire system subtree for entity 10.0.0.1 and for entity 10.0.0.2 it includes access to sysName.

$A_1 = \{x_1, x_2, x_3, x_4\}$

$B_1 = \{x_4\}$

This corresponds with the access rights given in the access matrix.

6.2 User2

User2 first includes the access rights from the group RestrictedUser. The group RestrictedUser refers to the definition given under User1. User2 then excludes sysName from the access rights. The scalar object sysName is not associated with any entity and is therefor excluded from both A and B.

$A_2 = \{x_1, x_2, x_3, x_4\} - \{x_4\} = \{x_1, x_2, x_3\}$

$B_2 = \{x_4\} - \{x_4\} = \{\}$

6.3 User3

User3 includes the access rights from group RestrictedUser and excludes the rights of User2.

$A_3 = \{x_1, x_2, x_3, x_4\} - \{x_1, x_2, x_3\} = \{x_4\}$

$B_3 = \{x_4\} - \{\} = \{x_4\}$

6.4 User4

User4 includes the group AdvancedUser. The content of AdvancedUser is drawn in a separate diagram. See Figure 4. This group includes RestrictedUser and also adds access to sysORIndex, sysORID and sysORDescr for entity 10.0.0.2. AdvancedUser also excludes sysName for entity 10.0.0.1.

$A_4 = \{x_1, x_2, x_3, x_4\} - \{x_4\} = \{x_1, x_2, x_3\}$

$B_4 = \{x_4\} + \{x_5, x_6, x_7\} = \{x_4, x_5, x_6, x_7\}$

The table symbol for sysORTable is not necessary in this example but it helps illustrating that sysORIndex, sysORID and sysORDescr are columns in a table.

7 MVML Translation Algorithm

MVML Translation Algorithm (MTA) takes MVML diagrams as input and translates them into MIB values that can be used for configuring the managed entities. This description assumes that all configuration of access control is done by MTA. To be able to configure the entities remotely, MTA will also need the user name and password of an administrator user at each managed entity. How this user name and password is acquired by MTA is considered an implementation issue. The steps involved in MTA are:

1. Find all users and entities specified in the MVML diagram
2. For each managed entity retrieve existing information from snmpUsmMIB and snmpVacmMIB.
3. For each user find the access list for each access type, security level and security model.
4. For each entity, generate values for snmpUsmMIB and snmpVacmMIB from the access lists.
5. For each entity, compare the new values to the old ones and update the entries in snmpUsmMIB and snmpVacmMIB.

Step 4 in the list is where most of the work is done. MTA should be efficient and produce as few entries in the MIBs as possible. It should also check for possible errors in the MVML diagrams. To be able to do this, MTA also need the MIB definitions as input. For example, if User1 in the previous example only had access to $\{x_1, x_2, x_3\}$ in entity 10.0.0.1 then it would be perfectly legal in MVML to draw scalar object symbols including x_1, x_2 and x_3. However, a more efficient solution would be to include the entire system subtree and then exclude x_4. Using the MIB definitions as input, MTA should optimize situations like this automatically.

Tables 5, 6 and 7 shows the entries in snmpVacmMIB for entity 10.0.0.1 after using MTA on the previous example. User 2 and 4 have the same access list and uses the same entries. The value of vacmViewTreeFamilySubtree should be an OID, but this table uses the symbolic name for clarity. In addition entries in usmUserTable would also have been created.

Object	Entries			
vacmSecurityModel	usm	usm	usm	usm
vacmSecurityName	user1	user2	user3	user4
vacmGroupName	group1	group2/4	group3	group2/4

Table 5: Entries in vacmSecurityToGroupTable

Object	Entries		
vacmGroupName	group1	group2/4	group3
vacmAccessContextPrefix			
vacmAccessSecurityModel	usm	usm	usm
vacmAccessSecurityLevel	authNoPriv	authNoPriv	authNoPriv
vacmAccessContextMatch	exact	exact	exact
vacmAccessReadViewName	view1	view2/4	view3
vacmAccessWriteViewName			
vacmAccessNotifyViewName			

Table 6: Entries in vacmAccessTable

8 Conclusion and future work

The main goal of MVML is to provide an easy to use tool for specifying access rights in SNMPv3. This is achieved by providing a simple graphical notation for specifying MIB views. The entire language only has six symbols and two relations. This makes it an easy language to understand and use. However, the hierarchical nature of the language makes it possible to draw complex and intricate diagrams that can be difficult to understand. To avoid this a methodology for using MVML in a concise and standardized way should be developed.

A more detailed description of the MVML Translation Algorithm is currently under development.

References

[1] Information technology - Abstract Syntax Notation one (ASN.1): Spesification of basic notation, 1997. Recommendation X.680.

[2] AdventNet. Agent Toolkit, 2000.

Object	Entries			
vacmViewTreeFamilyViewName	view1	view2/4	view2/4	view3
vacmViewTreeFamilySubtree	system	system	sysName	sysName
vacmViewTreeFamilyMask	FE	FE	FF	FF
vacmViewTreeFamilyType	include	include	exclude	include

Table 7: Entries in vacmViewTreeFamilyTable

[3] R. Presuhn B. Wijnen and K. McCloghrie. RFC 2575: View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), April 1999. Status: DRAFT STANDARD.

[4] U. Blumenthal and N. Wijnen. RFC 2574: User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3), April 1999. Status: DRAFT STANDARD.

[5] R. Presuhn D.Harrington and B. Wijnen. RFC 2571: An architecture for describing snmp management frameworks, April 1999. Status: DRAFT STANDARD.

[6] D. Partain J. Case, R. Mundy and B. Stewart. RFC 2570: Introduction to version 3 of the internet-standard network management framework, April 1999. Status: DRAFT STANDARD.

[7] R. Presuhn J. Case, D. Harrington and B. Wijnen. RFC 2572: Message processing and dispatching for the Simple Network Management Protocol (SNMP), April 1999. Status: DRAFT STANDARD.

[8] D. Levi and P. Meyer. RFC 2573: SNMP applications, April 1999. Status: DRAFT STANDARD.

[9] Object Management Group. OMG Unified Modeling Language Specification, 1999.

[10] D. Perkins and E. McGinnis. Understanding snmp mibs, 1997.

[11] David Zeltserman. *A Practical Guide to SNMPv3 and Network Management.* P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1999.