SNMPv3 can still be simple?

Omar Cherkaoui, Nathalie Rico
Université du Québec à Montréal
Pavillon Président-ennedy local PK4158
201, av. du Président-ennedy
Montréal (Québec) H2X 3Y7 CANADA
cherkaoui.omar@uqam.ca

Ahmed Serhrouchni
École Nationale Supérieure des Télécommunications
46, rue Barrault
Paris 75634 Cedex 13
FRANCE
Ahmed.Serhrouchni@enst.fr

Abstract

The Simple Network Management Protocol (SNMP) was introduced in 1988. The initial version (SNMPv1) is still widely implemented, deployed, and used. SNMPv3 is now in its final stages of standardization. SNMPv3 allows new capabilities for open, interoperable, and secure management on the Internet environment. SNMPv3 builds on SNMPv1 and v2 to include methods for security (authentication, encryption, and privacy), and a new administrative framework. From a practical perspective, the SNMPv3 architecture must allow the evolution of the system to consider different versions of SNMP, different applications, and different security models. We will present an implementation model for SNMPv3 that keeps the simplicity of SNMPv1, while at the same time making it possible to evolve the framework to address different security needs and different configurations. The implementation model is an abstract view of the implementation that represents the building blocks required for any realization of agents, platform, proxies, etc. We explain how the model with a modular approach can allow interoperability with other SNMP applications. The model uses the interfaces as defined by the RFCs to achieve the architectural goals (extensibility, reusability, etc.) and to simplify the module construction. An implementation was done at UQAM and it uses Java to create the modular reusable components.

Key words

Network Management, SNMPv3, Security, Java.

1. Introduction

As the sophistication and complexity of telecommunications networks and services grow, we need to model and build management systems that can evolve easily to manage the different networks, services, and customers.

The Simple Network Management Protocol (SNMP) defines a framework for the management of TCP/IP capable data communications network devices. With the different versions of SNMP (v1, v2, v3), the functions added are becoming more complex: version 3 includes security (authentication, encryption) and a new administration framework. The SNMPv3 architecture

resolves this problem by using modularity to allow the evolution of portions of SNMP without requiring a redesign of the general architecture.

The management platform and agents have to be adapted to the functions required to avoid burdening it with unnecessary components. The agents residing in the equipment must be kept simple and use the minimum amount of resources. Keeping agents and platform as simple as possible is an important goal to allow easy evolution of the management system and to simplify the task of the network administrator.

We will present an implementation model of SNMPv3 that keeps the simplicity of SNMPv1, while at the same time making it possible to evolve the framework to address different security needs and different configurations. We achieve this goal by using a modular approach. The model uses the interfaces as defined by the RFCs to achieve the architectural goals (extensibility, reusability, etc.). The model lends itself to a modular implementation: modules can be easily added and adapted to specific security needs and other functionality required.

The outline is as follows. We will first begin with a presentation of the SNMP protocol and with the different versions of the standard. We will present the objectives of the new architecture and describe the different components of the architecture. We will describe the SNMPv3 message. Then we will explain our SNMPv3 implementation model that uses a modular approach. We will show how modules can be reused when the system is extended and how the model allows handling multiple security models, different SNMP versions, and different applications.

2. SNMP Framework

2.1 Historical perspective

SNMP is a management protocol originally designed for the management of TCP/IP capable data communications network devices. The guiding principle for its design was simplicity, to ensure agent implementations could easily be made available. The Management Information Base (MIB) consists of objects (not in the object-oriented sense) representing variables. The management protocol itself offers a set of operations to retrieve and set values of particular object instances, to traverse tables, and to send simple traps, for example, unsolicited events from the agents to managers. The reader unfamiliar with SNMP concepts is asked to refer to [1] [2].

SNMPv1 became both an open IETF standard and a de facto industry resulting from widespread market acceptance. A broad range of vendors implemented SNMP versions and extended the scope of SNMP in many directions, including network management, system management, application management, manager-to-manager communication, and proxy management of legacy systems. One of the most important weaknesses of SNMPv1 is the lack of adequate mechanisms for securing the management function. This includes authentication and privacy, as well as an administrative framework for authorization and access control. The IETF working group that developed SNMPv2 wanted to include security functionality in the new version. Various proposals were put forward (SNMPv2c, SNMPv2u, SNMPv2*) [3][4][5][6][7][8][9] but none of those were adopted as an IETF standard. The working group was not able to reach agreement on how to define the required security mechanism.

The third version of SNMP [12][13][14][15][16][17] is derived from and builds upon both the first and the second versions. SNMPv3 allows new capabilities for open, interoperable, and secure management on the Internet environment. It supplements the SNMPv2 framework by supporting the following: security (methods for authentication, encryption, privacy, authorization, and access control) and an administrative framework (naming of entities, user names and key management, notification destinations, proxy relationships, remotely configurable via SNMP operations). With SNMPv3, a single protocol entity may provide simultaneous support for multiple security models, as well as multiple authentication and privacy protocols.

2.2 SNMP framework

All versions of SNMP (v1, v2 and v3) share the same basic structure and components, and follow the same architecture. Over time, the framework has evolved [18] and the definition of these architectural components has become richer and more clearly defined, but the fundamental architecture has remained consistent. The SNMP framework emphasizes the use of modularity for the evolution of portions of SNMP without requiring a redesign of the general framework. The framework of the Internet-Standard Management Framework consists of a data definition language, a definition of management information (MIB), a protocol definition, security, and administration.

2.3 Goals of the framework

The objective of the SNMPv3 framework is to allow evolution to realize effective management in a variety of configurations and environments. The SNMPv3 framework builds and extends these architectural principles by building on the basic architectural components, in some cases incorporating them from the SNMPv2 framework by reference, and by using these same layering principles in the definition of new capabilities in the security and administration portion of the architecture. More specifically, the goals are to reuse as much as possible the work in the previous versions (SNMPv2u and SNMPv2*); to address the need for security support; and to make it possible to evolve the framework to support different security models, different features required, and different configurations. Also, the architecture should make it relatively inexpensive to deploy a minimal configuration. This architecture must allow the different modules to be reused independently of the version of the SNMP protocol, depending on the role of the driver (platform, proxy agent), of the different encryption and authentication techniques, and of the different management applications.

As defined by the IETF, the goal of the framework design is to use encapsulation, cohesion, hierarchical rules, and loose coupling to reduce the complexity of design and make the evolution of portions of the framework possible. Encapsulation describes the practice of hiding the details that are used internal to a process. To achieve cohesion, similar functions can be grouped together and their differences ignored, so they can be dealt with as a single entity. It is important that the functions, which are grouped together, are actually similar. Functionality can be grouped into hierarchies where each element in the hierarchy receives general characteristics from its direct superior, and passes on those characteristics to each of its direct subordinates. Coupling describes the amount of interdependence between parts of a system.

3 SNMPv3 entity

The SNMP entity defined by IETF [3] is composed of an SNMP engine and one or more applications. The SNMP engine provides services for sending and receiving messages, authenticating and encrypting messages, and controlling access to the managed objects. An SNMP engine contains a Dispatcher, a Message Processing Subsystem, a Security Subsystem and an Access Control Subsystem. Figure 1 shows the SNMP entity composed of applications such as command generators, notification originators and receivers, and the different modules of the SNMP engine (Dispatcher, Message processing Subsystem, Security Subsystem).

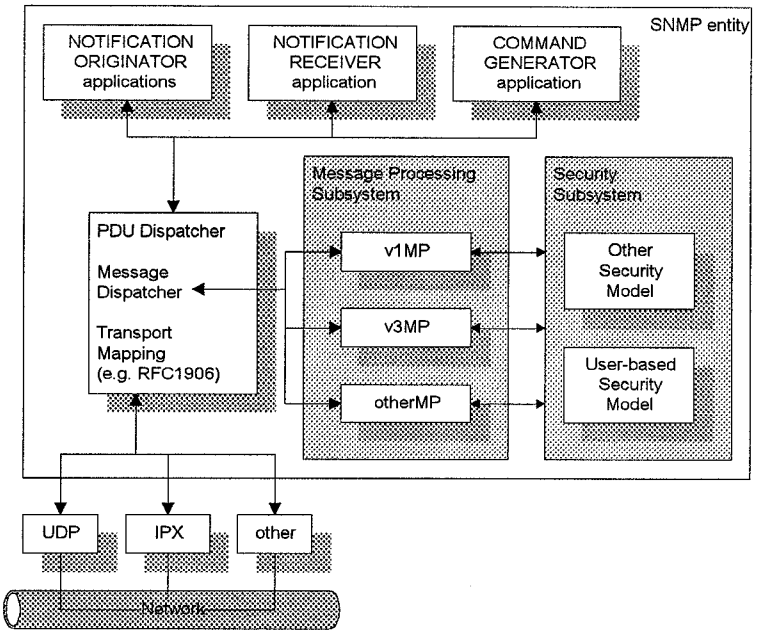

Figure 1: SNMP entity

The Dispatcher allows for concurrent support of multiple versions of SNMP messages. The dispatcher sends and receives messages to/from the network, determines the version of an SNMP message, interacts with the corresponding Message Processing Model, and provides an abstract interface to SNMP applications for delivery of a PDU. There is only one Dispatcher in an SNMP engine.

The Message Processing Subsystem is responsible for preparing messages for sending and extracting data from received messages. It contains multiple Message Processing Models corresponding to the different SNMP versions (SNMPv3, SNMPv1, and SNMPv2c Message Processing Model) or to other Message Processing Models.

The Security Subsystem provides services such as the authentication and privacy of messages and potentially contains multiple Security Models (User-Based Security model or other Security models). A security model defines the threats against which it protects, the service provided and the security protocols used (procedure and MIB data) to provide the service such as authentication and privacy.

The User-Based Security Model (USM) for SNMPv3 [4] defines the elements of procedure for providing SNMP message-level security. USM protects against following primary and secondary threats: modification of information, masquerade, message stream modification, and (optionally) disclosure. The USM uses MD5 and the Secure Hash Algorithm as keyed hashing algorithms for digest computation. This provides data integrity to directly protect against data modification attacks, to indirectly provide data origin authentication, and to defend against masquerade attacks. The USM uses the Data Encryption Standard (DES) in the cipher block chaining mode (CBC) to protect against disclosure. The configuration parameters in the MIB (including key distribution and key management) can be remotely monitored and managed. A single protocol entity may provide simultaneous support for multiple security models, as well as multiple authentication and privacy protocols. All of the protocols used by the USM are based on symmetric cryptography, i.e. private key mechanisms. The SNMPv3 architecture permits the use of public key cryptography, but as of this writing no SNMPv3 security models utilizing public key cryptography have been published.

The Access Control Subsystem provides authorization services by means of one or more Access Control Models. An Access Control Model defines a particular access function to support decisions concerning access rights. An example of Access Control Model is the View-Based Access Control Model. The View-Based Access Control Model defines the elements of procedure for controlling access to management information. It includes a MIB for remotely managing the configuration parameters for the View-Based Access Control Model. The View-Based Access Control Model can simultaneously be associated in a single-engine implementation with multiple Message Processing Models and multiple Security Models. It is possible to have multiple, different Access Control Models active and present simultaneously in a single engine implementation, but in practice it is expected to be very rare.

The Applications Modules are the processes that interact with the SNMP engine using messages that may use formats defined by a protocol, or that may use implementation-specific formats. Applications are developed to achieve certain goals. A proxy application may forward a message from one SNMP engine to another (an SNMP proxy), or from SNMP to another protocol (a foreign proxy). There are several types of applications, such as command generators which monitor and manipulate management data; command responders which provide access to management data; notification originators which initiate asynchronous messages; notification receivers which process asynchronous messages; and proxy forwarders which forward messages between entities. The applications use the services provided by the SNMP Engine.

For example, it is possible to design a platform that can handle each management application and each type of communication protocol (SNMPv1, SNMPv2, and SNMPv3) and a specific security type. The architecture can support older versions of SNMP, as well as future changes to SNMP. The architecture can handle multiple security systems, where a portion may be secure while another portion may be non-secure.

3. SNMPv3 Message

The SNMPv3 message contains fields for global data (such as the SNMP version, the message identifier, the maximum message size, the security model, and the level of security), fields for the security model information, fields for naming scope (context identifier and name), and finally, the PDU. The SNMP version identifies the version of the Message Processing Model in use. SNMP engines use the message identifier (Msg ID) to coordinate the processing of the message by different portions of the framework. MMS (Maximum Message Size) is the maximum message size supported by the sender of the message. Multiple security models may exist concurrently in the SNMP entity. The initial model of the SNMPv3 Security Framework is the User-Based Security Model of the SNMPv3 Security Framework. The LoS (Level of Security) field contains flags to control the processing of the message.

An SNMP context is a collection of management information accessible by an SNMP entity. An item of management information may exist in more than one context. An SNMP entity has access to many contexts. A context name is used to name a context. A scoped PDU is a block of data containing a contextEngineID, a contextName, and a PDU. The context ID defines the engine, which realizes the managed objects referenced in the PDUs. Figure 2 shows the SNMPv3 message format.

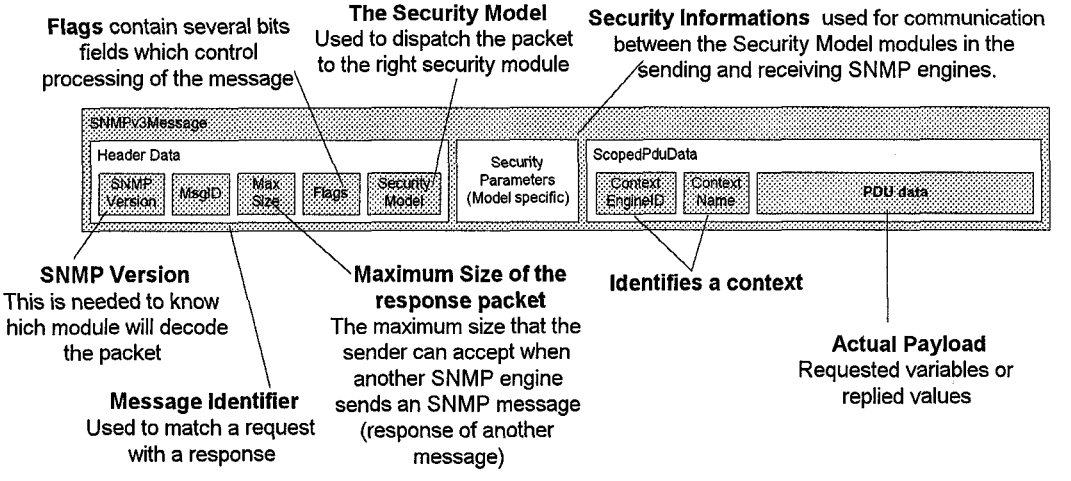

Figure 2: SNMPv3 Message Format

4. Implementation Model of SNMPv3

We will present an implementation model of SNMPv3 that keeps the simplicity of SNMPv1, while at the same time making it possible for the engine to address multiple security needs and different configurations [19]. The modules can be easily added and adapted to the specific

security need and other functionality required. The implementation model has the following characteristics:

- The model follows the framework and the interface definitions in RFC2271;
- Different modules can be dynamically attached and detached through a unique point;
- Classes and hierarchies allow reuse of components.

For the implementation, we chose Java because it is portable, dynamic, object-oriented, multithreaded and simple to use.

4.1 Following RFC2271 interface definitions

One goal of the implementation model is to define objects or modules that can be reused for different scenarios. The difficulty lies in partitioning the functionality between the different modules and to define the interfaces for those different modules in the SNMP Engine to allow individual modules to be replaced or augmented. At the same time, interfaces must not be defined with too many details not to unnecessarily constrain the implementation. Object-oriented methodology is adapted to tackle the interface problem. In order for objects in the model to work together, objects have to know exactly what they can expect from every object they might call upon for a service. The abstract service interfaces between the different subsystems (as defined in the IETF standard) are used in the model and help provide the modular structure that keeps SNMPv3 implementations simple. Primitives specify the service provided and the abstract data elements that are to be passed when the services are invoked.

The implementation model and its decomposition in different modules ease the understanding of the implementation. The decomposition and the interfaces in the model follow the RFC definitions [3][4][5][6][7]. For example, RFC2271 describes an interface in the processing modules that allows the dispatcher to request an SNMP packet. The required parameters are given as input and the SNMP packet is returned.

```
statusInformation =              -- success or errorIndication
     prepareOutgoingMessage(
     IN   transportDomain           -- transport domain to be used
     IN   transportAddress          -- transport address to be used
     IN   messageProcessingModel    -- typically, SNMP version
     IN   securityModel             -- Security Model to use
     IN   securityName              -- on behalf of this principal
     IN   securityLevel             -- Level of Security requested
     IN   contextEngineID           -- data from/at this entity
     IN   contextName               -- data from/in this context
     IN   pduVersion                -- the version of the PDU
     IN   PDU                       -- SNMP Protocol Data Unit
     IN   expectResponse            -- TRUE or FALSE
     IN   sendPduHandle             -- the handle for matching
                                    -- incoming responses
     OUT  destTransportDomain       -- destination transport domain
     OUT  destTransportAddress      -- destination transport address
     OUT  outgoingMessage           -- the message to send
     OUT  outgoingMessageLength     -- its length
          )
```

In the implementation, a java class corresponds to the interface definition:

```
public abstract StatusInformation prepareOutgoingMessage(
        int transportDomain,              // IN  as specified by application
        IpAddress transportAddress,       // IN  as specified by application
        ..........
        IpAddress destTransportAddress[],// OUT destination address
        byte[] outgoingMessage[],         // OUT the message to send
        int outgoingMessageLength[]       // OUT the message length
    );
```

The modules are constructed around Java classes (one module per one Java class). Dynamic construction of Java classes results in the creation of modules. This implementation model which follows closely the standards allows rapid prototyping and easy development of new applications.

4.2 Dynamic attachment and detachment of modules through a unique point

The implementation model uses modules that can be easily added and adapted to the specific functionality required. Only necessary modules are taken to compose the desired implementation: if an implementation does not require security models, the modules are constructed without the security models and are kept as simple as possible.

The proposed model allows to attach and detach the modules through the use of a unique method called "AddModule()" and a unique interface. The SNMP engine is assembled to play the role of agents, platform or proxies. The dynamic construction of modules avoids wasting computer resources (memory, etc.). A SNMPv1 agent can be assembled with only 3 modules, reducing the memory required. A module can be attached or detached during the execution. The class SnmpEngine.java offers the method to add modules. This method is also used during the initialization of the engine to load all modules. To add a new module, a class instance is derived from SnmpModule.java and the object is a parameter passed to the method AddModule().

The implementation model is composed of modules and buses between modules. Those buses can keep a list of active modules and enough information on each module to route the message to the correct module. Figure 3 shows the different buses in the system.

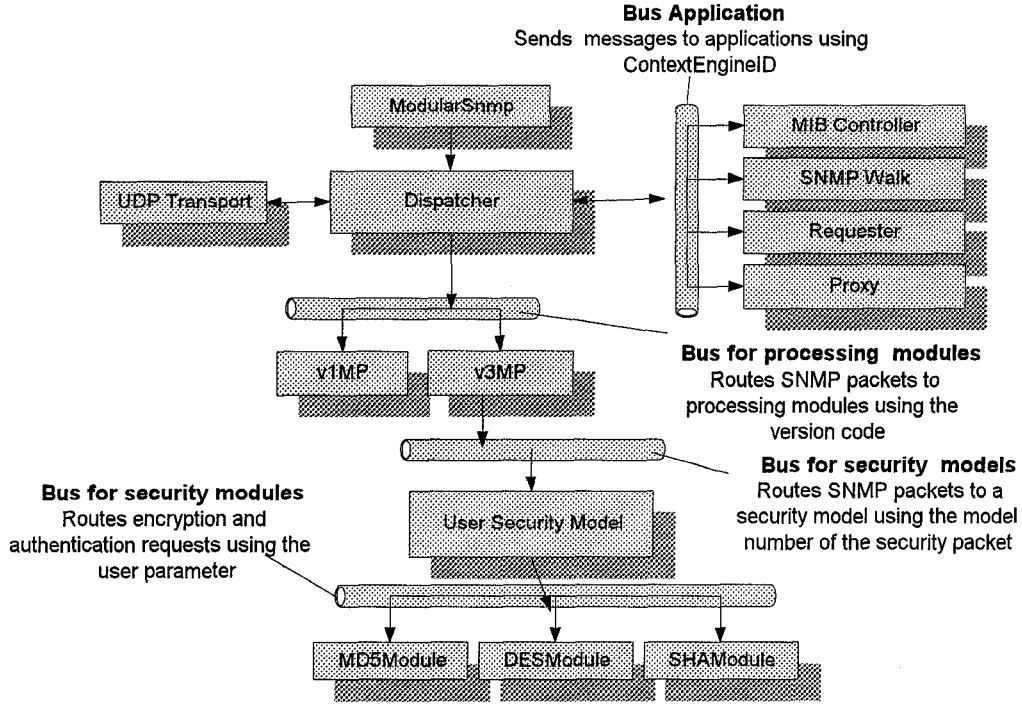

Figure 3: Buses in the model

4.3 Class hierarchy

The classes are defined in a hierarchy to inherit the functionality of other classes and to hide the details of different classes behind a common interface (polymorphism). Figure 4 shows the class hierarchy.

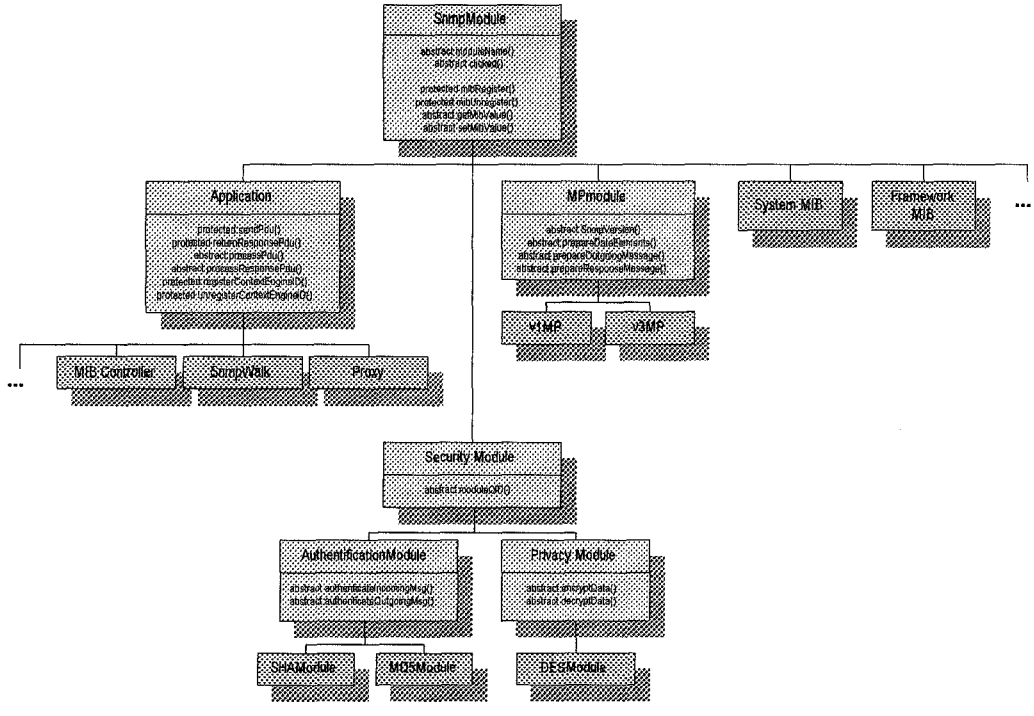

Figure 4: **Class hierarchy**

Abstract classes are used to impose a series of methods for the class that correspond to the implementation. For example, a class for an application must contain methods for receiving the messages, the responses to messages, etc. As a result, a unique and identical interface for all applications is used.

The SNMP engine has an internal MIB. In the implementation model, every module can add variables to the internal MIB of the engine. Only one interface is required to build a MIB. A MIB controller takes the request destined to SNMP engine and sends them to the proper modules. All requests must be kept in a sorted list for all OID (Object identifier) variable. The SNMP request is then broken down in a series of GET and SET that are sent to the different modules. Each module registers its variables with the MIB controller. This list contains all registered OIDs and a pointer to the application that handles the request. For each OID request, the corresponding module is called.

4.4 Modules and buses

The implementation model is composed of the dispatcher, the security modules, the application modules and the transport modules. A class is associated with each module. Figure 5 shows the modules that contain the functions related to the SNMP protocol and the bus that handle the routing of messages to the modules. The buses are used to keep a list of modules and enough information to route the messages to the right module. Each bus in the architecture has a module

for temporary storage. Those information are used to make the link between the modules attached to the bus and the functionality offered by each module. For example, a bus between the dispatcher and the processing modules contains a temporary table that associates the SNMP version to the module that handled the packets. Additionally, modules are attached to the Dispatcher. The dispatcher routes the path of the packets to the appropriate decoding/encoding module. For example, the "Dispatcher" object will dispatch SNMPv1 packets to the correct SNMPv1 Message Processing module for processing. The same process is applied to Security modules and Applications. Dispatchers are also responsible for keeping track of connected modules, so, a 1 to n link exists for dispatcher objects to their modules.

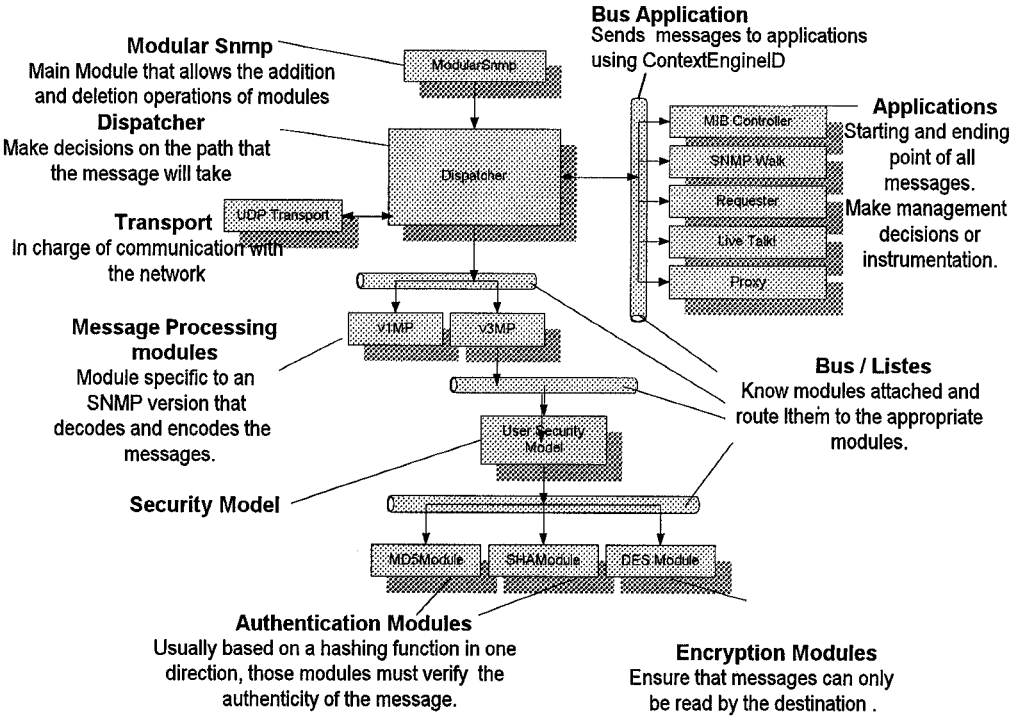

Figure 5 : Implementation model

Modules all have some common methods and attributes; they are grouped in a parent class. This parent class implements two other functions. First, it implements functionality for keeping libraries of modules. Second, it implements a special class, "SNMP Engine", which represents the entire SNMP entity. Its behaves like a master control that handles engine wide operations like adding a new module, stopping/starting the engine and forwarding debugging messages.

Data in transported to and from modules and dispatchers within a "Packet" object that acts as a container for ASN.1 coded SNMP data. The entire system works as a whole to transport information to and from the "UDP Transport" and "Application" modules.

In the proposed implementation model, threads are used to handle faults. A fault in the execution does not block the message handling. If a thread blocks waiting for message in the network, it is mandatory to continue after the software failure. When a packet is received, a new thread is created to handle the packet processing. The previous thread is then released. Many threads can coexist and can result in inconsistent variables. In our implementation, two threads are not allowed to process the same program at the same time.

The implementation covers all main SNMP entity components. The following category of Java classes were implemented: interface and service classes, central module classes, processing module classes, application classes, UDP transport, User security Model classes, Security module classes (DES, MD5, SHA-1, CBC, HMAC) and classes for type definitions.

4.5 Resulting configurations

The implementation model allows different configurations. Elements of an SNMP engine can be reused for the different roles (agent, platform and proxy). The different modules are constructed based on the implementation model to have the required configuration for the platform and the agent.

Figure 6 shows a configuration with a platform and an agent.

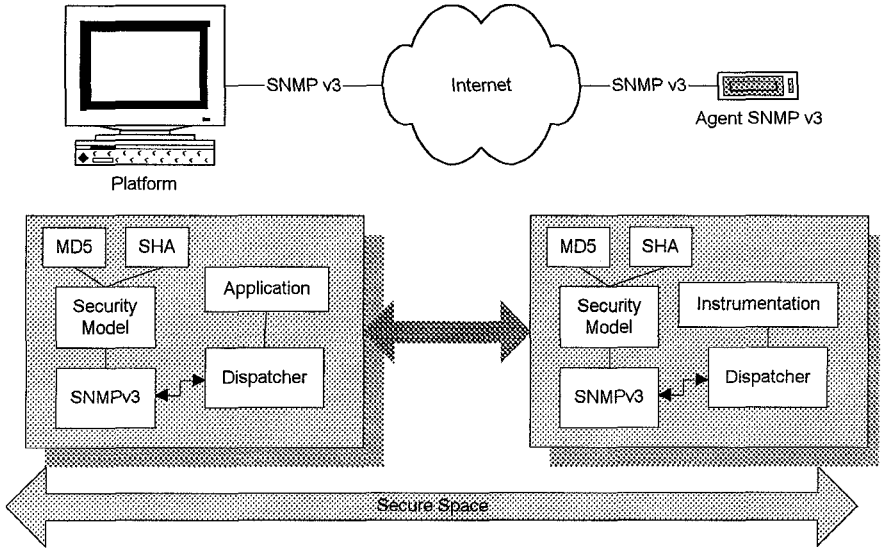

Figure 6: Configuration with a platform and an agent

Figure 7 shows a configuration with a platform and a secure agent exchanging directly in SNMPv3. When versions are different or the security modules (encryption and authentication) are not the same, it is necessary to use a proxy to ensure secure communication between the different security versions.

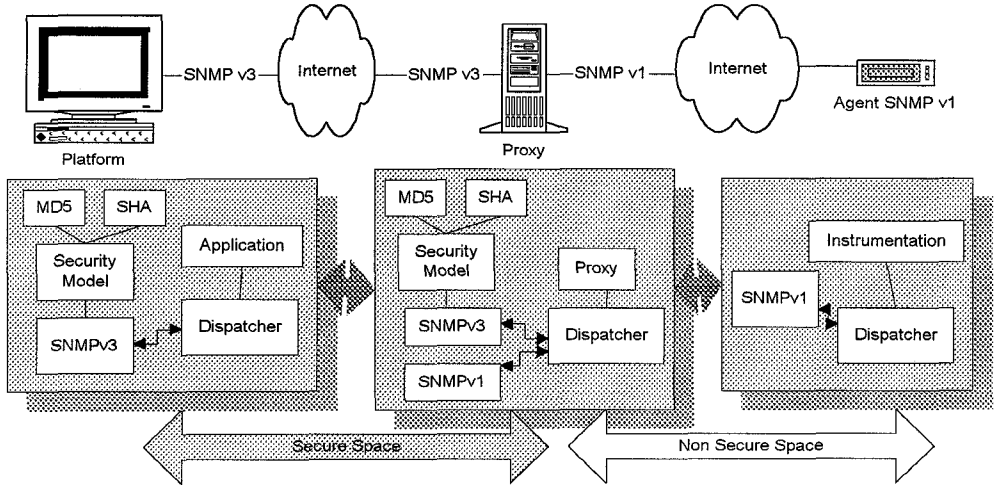

Figure 7: Configuration with a platform and a secure agent

The requirements vary depending on the situation. It is hence important to be able to easily construct the platform/agent/proxy adapted to the specific needs. Using an implementation model based on a modular approach is an important asset.

5. Conclusion

We proposed an implementation model for SNMPv3 that allows the evolution of the system to consider different security models and different configurations while keeping the simplicity of SNMPv1. This work builds upon the existing products and protocols. The modularity of the approach adds considerable flexibility to the solution. At the same time, this modularity help keep SNMPv3 implementations as simple as SNMPv1. In the implementation model, the key is to use the abstract service interfaces between the modules defined in the RFCs. It will allow the evolution of the architecture to encompass different scenarios (two-tier models, three tier models with proxy or delegated agents), different security models and different access control models. This architecture allows to reuse the different modules depending on the version of SNMP, on the security model required (type of encryption and authentication) or on the different management applications. Also, the model allows the interoperability with Corba, JMAPI, and others. The implementation uses Java to create modular reusable components. Java is an open, industry standard suited for defining reusable components. Java is also a portable, object-oriented language that is architecturally neutral.

References

[1] Rose, M., and K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", STD 16, RFC 1155, May 1990.

[2] Case, J., M. Fedor, M. Schoffstall, and J. Davin, The Simple Network Management Protocol", RFC 1157, University of Tennessee at Knoxville, Performance Systems International, Performance International, and the MIT Laboratory for Computer Science, May 1990.

[3] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S., Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996.

[4] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S., Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[5] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Textual Conventions for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1903, January 1996.

[6] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S., Waldbusser, "Conformance Statements for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1904, January 1996.

[7] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S., Waldbusser, "Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1905, January 1996.

[8] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Transport Mappings for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1906, January 1996.

[9] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Management Information Base for Version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1907 January 1996.

[10] The SNMPv2 Working Group, Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management.

[11] McCloghrie, K, Editor, "An Administrative Infrastructure for SNMPv2", RFC 1909, February 1996.

[12] Galvin, J., and McCloghrie, K., "Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1445, Trusted Information Systems, Hughes LAN Systems, April 1993.

[13] Harrington, D, R. Presuhn, and B. Wijnen, "An Architecture for Describing SNMP Management Frameworks, RFC 2271, January, 1998.

[14] Case, J., Harrington, D., Presuhn, R., and B. Wijnen, "Message Processing and Dispatching for the Simple Network Management Protocol (SNMP)", RFC 2272, January 1998.

[15] Levi, D., Meyer, P., and B. Stewart, "SNMPv3 Applications", RFC 2273, January 1998.

[16] Blumenthal, U., and B. Wijnen, "The User-Based Security Model for Version 3 of the Simple Network Management Protocol (SNMPv3)", RFC 2274, January 1998.

[17] Wijnen, B., Presuhn, R., and K. McCloghrie, "View-based Access Control Model for the Simple Network Management Protocol (SNMP)", RFC 2275, January 1998.

[18] Harrington, David, "The evolution of Architectural Concepts in the SNMPv3 Working Group", The Simple Times 5(1), December 1997.

[19] Cherkaoui, O., Saint-Hilaire, Y. and Serhouchni, A., "Towards a modular and interoperable SNMPv3", NOMS'98, New Orleans.