Object-Oriented Network Management Development

As networks continue to proliferate, the need to develop and deploy network management applications has become a critical issue. Two software development tools are described that allow developers to create powerful network management-side applications quickly without necessarily having to be experts on network protocols.

by Peter E. Mellquist and Thomas Murray

The advantages of using an object-oriented design methodology and its associated productivity tools, such as C++,[1] are well entrenched and accepted in today's software engineering practices. One of the areas where the object-oriented paradigm can bring a great deal of productivity is in the development of network management applications.

This article describes two management-side* tools for network management development: SNMP++ and SNMPGen. SNMP++ is a set of C++ classes that provide Simple Network Management Protocol (SNMP) services to a network management application developer.[2] SNMP++ brings the object-oriented advantage to network management programming, and in doing so, makes it much easier to develop powerful, portable, and robust network management applications. SNMPGen is a technology that uses SNMP++ to automatically generate C++ code from the SNMP Management Information Base (MIB) definitions. Fig. 1 shows a conceptual view of these two tools.

SNMP++ was developed by Hewlett-Packard and is a freely available open specification. Anyone can obtain the specification document on the World-Wide Web at URL *http://rosegarden.external.hp.com/snmp++*.

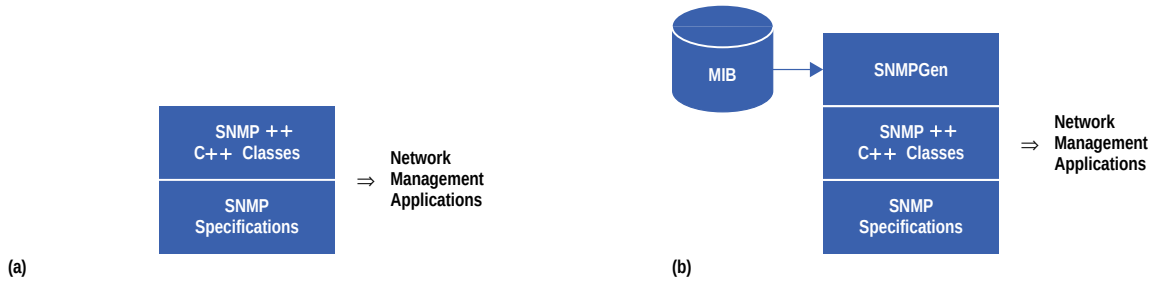

Fig. 1. *A conceptual view of (a) SNMP++ and (b) SNMPGen.*

Benefits of SNMP++

Various SNMP application programming interfaces (APIs) exist that allow the creation of network management applications. The majority of these APIs provide a large library of functions that require the programmer to be familiar with the details of SNMP and SNMP resource management. Most of these APIs are platform-specific, resulting in SNMP code that is specific to a particular operating system or network operating system platform, and thus, not portable.

Application development using a standard set of C++ classes for network management provides many benefits including ease of use, safety, portability, and extensibility.

Ease of Use. SNMP++ is designed so that application programmers do not need to be concerned with low-level SNMP mechanisms. An object-oriented approach to SNMP encapsulates and hides the internal mechanisms of SNMP. SNMP++ provides the following ease-of-use features:

- It provides an interface in which the user does not have to be an SNMP or C++ expert to use its features. For the most part C pointers do not exist in SNMP++, resulting in a simple and straightforward API.
- It provides easy migration to SNMP version 2.[3,4] One of the major goals of SNMP++ was to develop an API that would be scalable to SNMP version 2 with minimal impact on code.

* This is in contrast to agent-side technologies. The manager/agent model, which is used for network management, is analogous to the client/server model for distributed applications.

- It preserves the flexibility of lower-level SNMP programming. A user may want to bypass the object-oriented approach and code directly to low-level SNMP calls.
- It encourages programmers to use the full power of C++.

Programming Safety. Most SNMP APIs require the programmer to manage a variety of resources, such as memory and sockets. Improper allocation or deallocation of these resources can result in corrupted or lost memory. SNMP++ manages these resources automatically. For programming safety SNMP++ addresses the following areas:

- Safe management of SNMP resources. This includes SNMP structures, sessions, and transport layer management. SNMP++ classes are designed as abstract data types providing data hiding and the ability for public member functions to inspect or modify hidden instance variables.
- Built-in error checking and automatic timeout and retry. SNMP++ users do not have to be concerned with providing reliability for an unreliable SNMP transport mechanism. A variety of communications errors can occur, including lost datagrams, duplicated datagrams, and reordered datagrams. SNMP++ addresses each of these possible error conditions and provides the user with transparent reliability.

Portability. Another major goal of SNMP++ was to provide a portable API across a variety of operating systems, network operating systems, and network management platforms (see Fig. 2). Since the internal mechanisms of SNMP++ are hidden, the public interface remains the same across any platform. A programmer who codes to SNMP++ does not have to make changes to move the program to another platform. Another issue in the area of portability is the ability to run across a variety of protocols. SNMP++ currently operates over the Internet Protocol (IP) or Internet Packet Exchange (IPX) protocols.

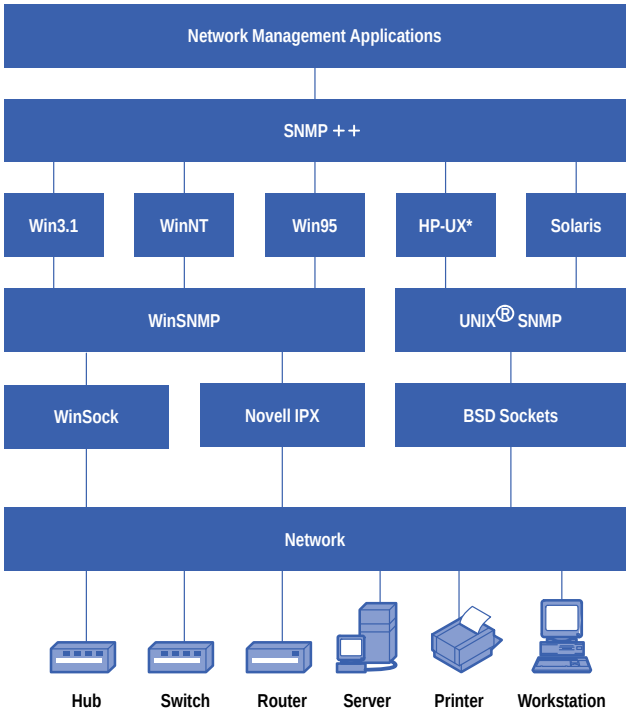

Fig. 2. *An illustration of the different operating systems, network operating systems, and network management platforms to which network management applications based on SNMP++ can be ported.*

Extensibility. Extensibility is not a binary function, but rather one of degree. Extensions to SNMP++ include supporting new operating systems, network operating systems, network management platforms, protocols, SNMP version 2, and other new features. Through C++ class derivation, SNMP++ users can inherit what they like and overload what they wish to redefine.

Overloading SNMP++ Base Classes. The application programmer can subclass the base SNMP++ classes to provide specialized behavior and attributes. The base classes of SNMP++ are meant to be generic and do not contain any vendor-specific data structures or behavior. New attributes can be easily added through C++ subclassing and virtual member function redefinition.

Example

Fig. 3 shows an example of the power and simplicity of SNMP++. This example obtains a system descriptor object from an SNMP Management Information Base (MIB) from the specified managed SNMP agent. Included is all the code needed to create an SNMP++ session, get the system descriptor, and print it out. Retries and timeouts are managed automatically.

```
#include "snmp-pp.h"
#define SYSDESCR "1.3.6.1.2.1.1.1.0"// Object ID for System Descriptor

void get_system_descriptor()
{
  int status                          // return status
  CTarget ctarget((IPAddress) "10.4.8.5"); // SNMP++ v1 target
   Vb vb( SYSDESCR);                  // SNMP++ Variable Binding Object
   Pdu pdu;                           // SNMP++ PDU
// ------ Construct an SNMP++ SNMP Object ------
   Snmp snmp( status);                // Create an SNMP++ session
   if (status != SNMP_CLASS_SUCCESS) {      // check creation status
      cout << snmp.error( status);          // if fail, print error string
      return;  }
//-------- Invoke an SNMP++ Get -----------
   pdu += vb;                         //add the variable binding to the PDU
   if (  (status = snmp.get(  pdu, v1target)) != SNMP_CLASS_SUCCESS)
      cout << snmp.error( status);
   else {
      pdu.get_vb( vb,0);              // extract the variable binding from PDU
                                      // print out the value
      cout << "System Descriptor = "<< vb.get_printable_value(); }
}
```

Fig. 3. *A simple SNMP++ example.*

The SNMP++ calls in this example total only ten lines of code. A CTarget object is created using the IP address of the agent. A variable binding object (Vb) is then created using the object identifier of the MIB object to retrieve the system descriptor (SysDescr). The Vb object is then attached to a PDU (protocol data unit) object. An SNMP object is used to invoke an SNMP Get request. Once retrieved, the response message is printed out. All error handling code is included.

SNMPGen: Machine-Generated SNMP Code

Traditional SNMP development is fairly low-level and redundant. SNMP++ programming requires a thorough understanding of the MIB definition used and the ability to perform manual fabrication of each PDU that is sent and received. When dealing with large amounts of MIB data, manual fabrication becomes tedious and somewhat prone to errors.

Consider, for example, traversing through a column in a MIB table. In SNMP this is performed by loading the object ID of the table column into a PDU and then performing SNMP GETNEXT operations on the table column until the object ID of the variable returned no longer matches the object ID of the table column. Although SNMP++ makes a task like this a little simpler for programmers, there still remains some complexity.

SNMPGen is a set of classes that were created to go beyond the limitations of traditional SNMP development and to further reduce the complexity of developing network management tasks such as the table traversal task mentioned above. These classes allow a higher level of abstraction over SNMP. Rather than deal at the level of PDUs and ASN.1 (Abstract Syntax Notation One) syntaxes, as is typical in SNMP development, SNMPGen offers an abstraction of MIBs and MIB objects. The resulting framework allows the user to write code focused on the manipulation of MIB objects with little concern about the underlying SNMP access details. Additionally, because SNMPGen is based on SNMP++, it can offer all the same advantages as SNMP++.

SNMPGen Classes. A MIB contains managed objects that are represented as tables, scalars, and groups. Each managed object has a name and a unique object identifier and attributes such as syntax and description (see Fig. 4). Tables contain one or more indexes (pointers to other objects) and zero or more managed objects. A scalar represents a single managed object. Groups can contain objects, tables, and other groups.

A MIB also defines the relationships among managed objects. Fig. 5 shows the relationships among SNMPGen classes.

SNMPGen Operations. A set of powerful operations (methods) are available for the various SNMPGen classes. These operations allow the manipulation of objects, columns in tables, entire tables, and groups. Many of the operations are conceptually the same for all of the classes. As a result it takes the same amount of code to get an entire MIB as it does to get a single MIB object. This dramatically reduces the coding effort for writing management applications.

All of the classes have operations to obtain values from an SNMP agent (by performing SNMP GET operations) and operations to send values to an SNMP agent (with SNMP SET operations). Additionally, the classes can have specialized operations that help with manipulating specific types of objects. For example, the CSNMPManagedTable class has operations for manipulating rows and columns in a table.

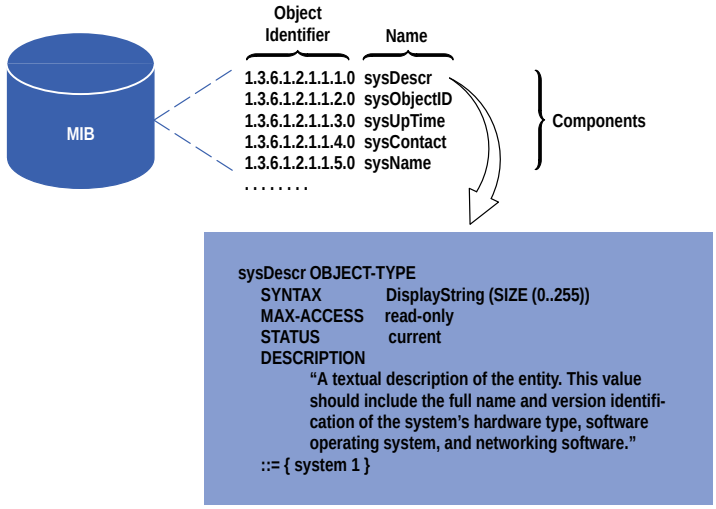

Fig. 4. *Identifiers for the contents of a MIB (Management Information Base) showing object identifiers and the object names. Also shown is the MIB definition of a managed object.*

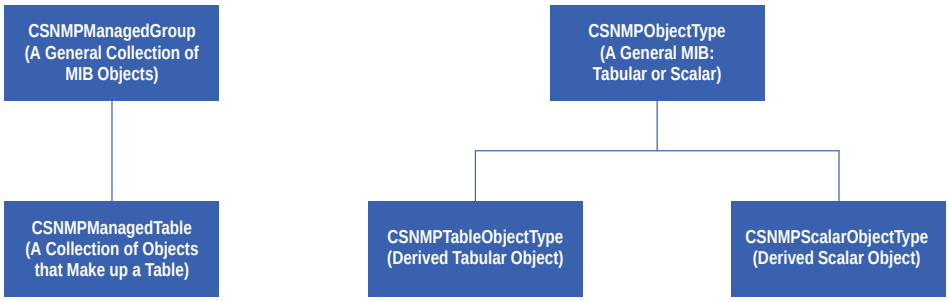

Fig. 5. *The relationships between SNMPGen classes.*

All SNMP considerations are hidden beneath the SNMPGen operations. However, many of the SNMP++ classes are used to manipulate the basic syntax of MIB variables. While nearly eliminating the requirement for detailed SNMP knowledge, this approach provides all the power and flexibility offered by SNMP++.

In addition to the operations provided, the classes can be extended to perform any tasks specialized for a particular application. This can be done simply by deriving new classes from the SNMPGen classes and defining the operations for the new classes.

Mibmgr. SNMPGen classes allow modeling of MIB objects, but they do not give the objects context. For example, instead of having a generic group class, a developer might want a class that represents a group called MIBII* or maybe a system class that represents the system group. A tool called mibmgr was developed to derive classes representing these kinds of specific MIB objects from the classes provided by SNMPGen. The result is a class definition for every object in a MIB. Developers then use these classes in SNMP management applications.

MIB documents tend to be long and somewhat difficult to read. The mibmgr tool was first developed to help the readability of MIBs by displaying them in a simple form. For example, MIBII would be reported as shown in Fig. 6.

This output was created by parsing the output of a MIB compiler tool called MOSY (Managed Object Syntax Compiler yacc-Based), which comes with the SNMP subagent development environment licensed from SNMP Research and maintained by HP's Network System Management Division. MOSY takes a MIB definition as input and produces a compact, easily parsable output file. This output file is then used as input to the mibmgr tool (see Fig. 7).

We quickly realized that this same information could be used in conjunction with the SNMPGen classes to produce C++ header files for SNMP management applications (see Fig. 8). Mibmgr was augmented to produce class definitions for each variable in a MIB. This gives context to the SNMPGen classes by completely defining the object ID space as well as the entire MIB hierarchy.

* MIBII is the second generation of standards for managed SNMP objects as defined by the Internet Engineering Task Force.

```
   Object Name                    Object Type
mgmt
|[1] mib_2
| | [1] system
| | | [1] sysDescr               DisplayString
| | | [2] sysObjectID            ObjectID
| | | [3] sysUpTime              TimeTicks
| | | [4] sysContact             DisplayString
| | | [5] sysName                DisplayString
| | | [6] sysLocation            DisplayString
| | | [7] sysServices            INTEGER
| | [2] interfaces
| | | [1] ifNumber               INTEGER
| | | [2] ifTable                Aggregate
| | | | [1] IfEntry              INDEX "ifIndex"
| | | | | [1] ifIndex            INTEGER
| | | | | [2] ifDesc             DisplayStringr
| | | | | [3] ifType             INTEGER
[...]
```

Fig. 6. *The output produced by the mibmgr for the example MIBII.*

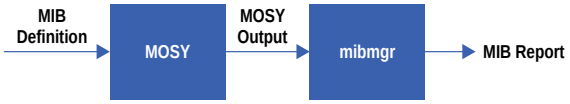

Fig. 7. *The processes involved in creating a MIB report.*

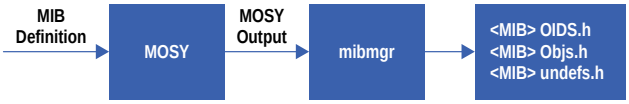

Fig. 8. *The process of automatically generating of C++ header files for network applications using SNMPGen.*

The following code shows an example of a machine-generated scalar object. (Note that mibmgr creates a C++ header file for the entire MIB and only the portion relevant to this example is shown.)

```
#define SsysDescr          "1.3.6.1.2.1.1.1"
 class CsysDescr: public CSNMPScalarObjectType{
   public:
    CsysDescr():
    CSNMPScalarObjectType(SsysDescr,
    sNMP_SYNTAX_OCTETS){};
};
```

Similarly, a machine-generated group would appear as follows:

```
 class Csystem: public CSNMPManagedGroup{
   public:
    Csystem():
    CSNMPManagedGroup(Ssystem){
      AddObject(&sysDescr);
      AddObject(&sysObjectID);
      AddObject(&sysUpTime);
    };
    CsysDescr sysDescr;
    CsysObjectID sysObjectID;
    CsysUpTime sysUpTime; ...
 };
```

This scheme has several benefits. Since the class definitions are produced in C++ header files, any changes to the MIB have little effect on management applications. The C++ header files also allow easy extensions to the classes if needed. Machine generation of class objects greatly reduces the possibility of coding errors and typing mistakes. SNMPGen combined with the machine-generated classes allows the developer to deal only with MIB objects and hierarchies. The use of the machine-generated classes essentially makes a MIB definition the development specification.

```
#include <iostream.h>
#include "mib2objs.h"
main(int argc, char **argv){
int status;                        // result of operation
CsysDescr sysDescr;                // SNMPGen created class from mib2objs.h
int status;                        // result of operation
CsysDescr sysDescr;                // SNMPGen created class from mib2objs.h

if (argc != 2){
cerr << "Usage: " << argv[0] << " <hostname>" << endl;
return(1);   }
IPAddress destination(argv[1]);   // use command line arg as destination
if (!destination.valid()){        // verify we resolved this address
cerr << "Bad hostname: " << argv[1] << endl;
return(1);   }

CTarget target(destination);      // Create an SNMP++ Target
Snmp my_session(status);          // Create an SNMP++ Snmp object

if (status){                      // Ensure the Snmp object is OK
   cerr << "Failed to create SNMP Session" << status << endl;
   return(1);   }

status = sysDescr.SNMPGet(my_session, target);// Access the MIB-II object
   if (status){                                // Check for success
      cerr << "Failed to issue SNMP get: " << my_session.error(status) <<
endl;
      return(1);   }

cout << "SysDescr = " << sysDescr.GetPrintableValue() << endl;  // display
return(0);
}
```

Fig. 9. *A program that uses SNMPGen classes to find a particular host (using a hostname parameter) and returning the MIBII system descriptor.*

Example. Fig. 9 shows a minimal program that takes a hostname as an argument and returns the system descriptor. The example assumes that the mibmgr tool was used to create the header file mib2objs.h from the MIBII specification. A sample output from the example in Fig. 9 is as follows:

```
$ minitest hpisrhx
 SysDescr = HP-UX hpisrhx A.09.01 E 9000/720
 $ minitest hpinddh
 Failed to issue SNMP get: Timed Out While Waiting for Response Pdu
```

The first case worked, and in the second case, no SNMP agent was running on system hpinddh, so the request timed out. The key points to note from this example are:

- Header files include the SNMP++, SNMPGen and its machine-generated files (mib2objs.h includes the SNMP++ and SNMPGen headers), and the standard C++ I/O library.
- SNMP++ target and SNMP classes are required to perform MIB access.
- The machine-generated classes produced by SNMPGen have the same name as defined in the MIB specification, with a C prepended (e.g., CsysDescr).
- Retrieving the object is done using just one operation against the object, which in this case is sysDescr.SNMPGet(). The member function SNMPGet() is inherited from the base class CSNMPObjectType.
- Accessing the retrieved data is also done as a single operation against the object, which in this case is sysDescr.GetPrintableValue(). The member function GetPrintableValue is inherited from the base class CSNMPObjectType.

Practical Uses of SNMPGen

SNMPGen classes combined with the machine-generated classes make it possible for management applications to be developed quickly and efficiently. Engineers writing management applications are frequently not networking experts. The classes allow SNMP management without requiring SNMP expertise. Additionally, when correctly used, the classes can reduce complexity and redundancy in management code.

Several management projects have been based on the SNMPGen classes. The HP ClusterView product, which is used to monitor MC/ServiceGuard high-availability clusters, uses the SNMPGen classes to collect cluster information. The tool was designed and written by engineers with very little networking, SNMP, or even C++ experience. The Web page http://hpgsslhc.cup.hp.com/ClusterView explains this product.

Additional applications include MIB browsing tools and MIB monitoring tools. Since the classes have MIB information embedded within them, they can provide good representations of MIB structures. Additionally, each class can answer questions about MIB object syntax, descriptions, and so on.

Alternatives

Development of network management applications that need to manage network devices or network systems is not an easy task. Often these devices operate in heterogeneous environments where one or more operating systems may be present. This can place additional requirements on a management application. Specifically, an application might need to run on more than one operating system, like the UNIX operating system and Microsoft®Windows.

Before SNMP++, the ability to use a production-quality set of object-oriented SNMP tools did not exist. Without one common set of SNMP libraries, experts are required for each operating system platform and each different SNMP library. Not only does this require additional engineers for development, but testing and maintenance are also adversely affected. SNMP++ allows reused code across any platform that supports ANSI/ISO C++ and supports Berkeley sockets, including all derivatives of the UNIX operating system and Microsoft Windows. Clearly, having to code to and support multiple SNMP APIs is not a viable alternative.

Conclusion

Creating the SNMPGen classes and the mibmgr tool was not a traditional project. It was done as a side job and encountered many obstacles, the foremost being lack of time and resources. The project pioneered the use of C++ development in the division and exposed many issues with the traditional development and integration model. For example, many of the tools had never been used with C++ code before.

The use of SNMP++, SNMPGen, and the machine-generated C++ header files demonstrates the power of object-oriented design and programming. The object-oriented paradigm allows complete freedom for developers. New attributes can be assigned to the class definitions and the classes are fully extensible. The use of these classes greatly reduces development effort. Engineers do not need to know network programming or learn about SNMP to create SNMP-based management applications.

As more and more networked devices and distributed systems are used in today's communication-intensive world, SNMP will play a major role in remote management. SNMP++, SNMPGen, and the classes created with the mibmgr tool will help developers meet the need for network and systems management by enabling rapid development.

References

1. B. Stroustrup, *The C++ Programming Language*, Second Edition, Addison Wesley, 1991.
2. M. Rose, *The Simple Book, An Introduction to Internet Management*, Prentice Hall, 1994.
3. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, *Coexistence Between Version 1 and 2 of the Internet Standard Network Management Framework*, Internet Engineering Task Force Request for Comment 1452, May 3, 1993.
4. J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, Internet Engineering Task Force Request for Comment 1442, May 3, 1993.

▶ Go to Subarticle 11a

▶ Go to Next Article

▶ Go to Journal Home Pag