EMOSY: An SNMP Protocol Object Generator for the Protocol Independent MIB

Shyhtsun F. Wu *

Computer Science Department
Columbia University
New York, NY 10027

Subrata Mazumdar, Stephen Brady

Network Management Systems
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

In this paper, we describe a non-adhoc approach to building an SNMP protocol object for the Protocol Independent Management Information Base (PIMIB). In [5], the design of a protocol independent MIB agent is proposed which can support both SNMP and CMIP. Then, in [9], we present the implementation of PIMIB, which provides not only management protocol independence (i.e., SNMP and CMIP) but also network device protocol independence. In practice, for SNMP management applications talking to the PIMIB, an SNMP protocol object with the knowledge of a MIB specification is needed to transform the SNMP requests into the PIMIB generic interface. In [7, 6], MOSY (Managed Object Syntax-compiler Yacc-based) is introduced as a tool to build an SNMP protocol object. However, in the normal building process of an SNMP subagent, a certain amount of knowledge about the implementation of network resources is still necessary for connecting the MIB with the real network environment. Thus, for a specific MIB description, the development process of an SNMP protocol object is adhoc. In order to improve this situation, two new key words **SOURCE** *and* **MAPPING** *representing the information about the network environment are introduced in an extended version of a MIB specification. Also, the EMOSY (Extended MOSY) compiler is implemented to mechanically generate the SNMP protocol agent as well as the PIMIB itself with this extended MIB specification as its input. In other words, we provide a tool that would directly compile a high-level MIB specification into a complete agent implementation. The major advantage of EMOSY is enhancement of agent portability, extensibility, and reusability in management agent development. Furthermore, PIMIB/EMOSY together present a prototype of a new network management agent architecture.*

1 Introduction

From a high level view, a network management system consists of a set of network management applications as well as a group of network devices. One key issue in network management system design is to define the communication path between the management applications and the devices. When the size and the speed of the network grows, it is intractable for network management applications to talk directly to devices because:

- it is unrealistic to build any one application which uses so many different device-specific information access protocols, and,
- since some raw information and events from the devices would be generated very frequently, it is inefficient to transmit every piece of information without certain levels of abstraction.

Therefore, the concept of a management agent with associated management information exchange protocols (CMIP and SNMP) was introduced to resolve the first problem, while the structure of the management information base was developed to abstract the management information in a more efficient way. This consensus view of a network management system is shown in figure 1.

The key component in building such a network management agent is the MIB specification (or description). In practice, this complies with the *Structure of Management Information (SMI)* for SNMP

*Wu is supported by an IBM Fellowship.

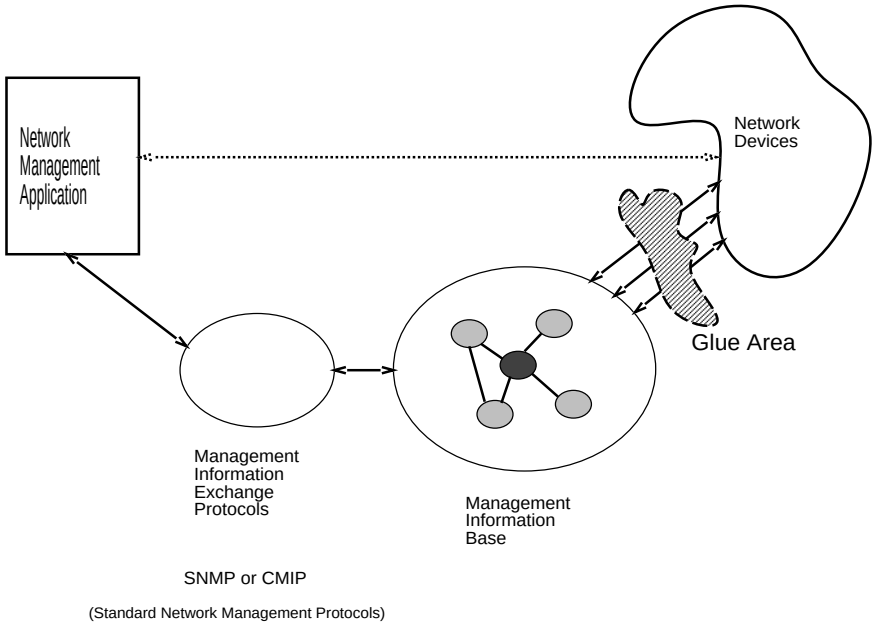

Figure 1: A High-Level View of A Network Management System

[6] or the *Guidelines for the Definition of Managed Objects (GDMO)* for CMIP [3, 2, 4]. The MIB specification is embodied in the implementation of management information exchange protocol handler (or, so called, protocol object or sub-agent) because otherwise the protocol object would have insufficient information to process requests from the applications. Also, the MIB specification presents the schema of the management information base. Thus, it is important for the management application builders and network device builders to reach a consensus on the MIB specification to ensure that information needed by the applications is in the MIB and information in the MIB can be provided by the network devices. Unfortunately, in most existing MIB documents, only the first item, *i.e.,* what information should be in the MIB, is presented. In other words, the problem of *how to get this information* is left open. One reason for this current situation is to provide the portability and interoperability of the MIB specification and network management applications. The "glue" area between the network devices and the MIB is a big problem in building network management systems because people are forced to take an adhoc approach as we will describe later. Very recently, people who realize this problem come out a specification for *Desktop Management Interface (DMI)* [1], which defines a common set of rules for accessing different types of components in a consistent fashion.

In [5], the design of a *protocol independent MIB agent* is proposed which can support both SNMP and CMIP. Then, in [9], we present the implementation of *PIMIB (Protocol Independent Management Information Base)*, which provides not only *management protocol independence* (*i.e.,* SNMP and CMIP) but also *network device protocol independence*. In practice, for SNMP management applications accessing the PIMIB, an *SNMP protocol object* with knowledge of a MIB specification is needed to transform the SNMP requests into the PIMIB generic interface. Usually, to build an SNMP protocol object, *MOSY (Managed Object Syntax-compiler Yacc-based)* is used. However, because the information about "how to get" is not in a MIB specification and the MOSY takes the specification as its only input, a certain amount of source code hacking is still necessary for connecting the MIB with the actual network devices. In the next section, we describe the shortcomings of this adhoc development process for an SNMP protocol object. Then, we propose a remedy, *EMOSY (Extended MOSY)*, to the problem we observe. Finally, we evaluate our approach and experience.

2 Motivation

A MIB specification provides information for the management protocol objects to process requests from management applications and it also defines the schema of the management information base. For example, in figure 2, an SNMP MIB variable, **coreAdapterState**, which is the second variable in the table entry **coreAdapterStatusEntry**, is defined as an integer with object identity **coreAdapterStatusEntry.2**. The information is enough for the SNMP protocol object to correctly identify the object instance for a request. For instance, if a **GET** request is received for **coreAdapterState.101**, then the protocol object should find out the **coreAdapterStatusEntry** instance with the **coreAdapterStatusLabel** value equal to "101". Then, the MIB builder has to provide pieces of source code for accessing each defined variable simply because the information about "how to get" is not available in the specification.

Therefore, a typical SNMP MIB building process would contain two different phases:

1. Given an MIB specification, we use a tool (e.g. MOSY) to mechanically generate the code for processing the request object identity as well as finding the instance.

2. Writing pieces of code for accessing the information from the network devices, and linking them to the generated code in step 1.

Obviously, comparing to the first step, the second step is very adhoc because we have to not only build those pieces of code for network devices but also link them to the *right* places. In practice, we might reuse the pieces of device code, but we must link them manually.

In a management agent development process, it is usual that, after testing a network management application, we realize that some information is missing from the MIB. This implies that we need to modify the MIB specification, and reimplement the SNMP protocol object with the adhoc approach. Certainly, the development process is slow if the changes to the MIB specification happen frequently. Furthermore, if we would like to port the MIB to other network platforms, we still have to redo the step 2 again, which restricts the MIB portability.

Our major motivation is to reduce the amount of time we spend in this lengthy development process. This goal can be achieved if the adhoc step 2 can be done in a mechanical way. However, mechanical transformation is impossible with the curent approach to define the MIB. In other words, we need to specify not only "what is the information" but also "how to get the information" in our MIB description.

3 EMOSY

In this section, a non-adhoc approach to building a network management agent is described. First, we introduce two new keywords: **SOURCE** and **MAPPING**, which give the information about "how to get" in the MIB specification. Then, the PIMIB generated by the EMOSY compiler is shown. Next, the registration tree and mechanism is discussed. Furthermore, the *object creation module* is presented. Finally, we illustrate the EMOSY/PIMIB architecture.

3.1 Extended MIB Specification

In order to put information about "how to get" in the MIB specification, we introduce two new key words: **SOURCE** and **MAPPING**:

SOURCE: SOURCE specifies the device protocol for accessing the information. It takes some parameters, and the number of parameters depends on the value of the first parameter. For example, if we have only one parameter with value "LOCALVALUE," then the value would directly and locally come from a *CoreObject* as described in [9]. If the first parameter is "CDSVALUE" and the second one is "[ADAPTER::Connect_State]," then the value is from the *control data segment (CDS)*, which is a piece of shared memory updated periodically by the network controller. The string "[ADAPTER::Connect_State]" represents the offset in the CDS.

MAPPING: Sometimes, the value stored in the CDS can not be directly used by the applications. (Keep in mind that the CDS might have already existed in a platform before we would like to port the management agent.) Therefore, we need to specify how to map the values from the CDS value to the value specified in the MIB.

After introducing SOURCE and MAPPING, we can get a new MIB specification for the **coreAdapterState** as shown in figure 3.

```
coreAdapterStatusEntry OBJECT-TYPE
    SYNTAX  COREAdapterStatusEntry
    ACCESS  not-accessible
    STATUS  mandatory
    DESCRIPTION
            ‘‘Entries in the Adapter Status table are indexed by label.’’
    INDEX { coreAdapterStatusLabel }
    ::= { coreAdapterStatusTable 1 }

COREAdapterStatusEntry ::=
    SEQUENCE
    {
        coreAdapterStatusLabel          INTEGER,
        coreAdapterState                INTEGER,
        coreAdapterAdminState           INTEGER,
        coreAdapterUpTime               TimeTicks,
        coreAdapterReceiverInError      TimeTicks,
        coreAdapterRedThresholdExceeded  TimeTicks
    }

coreAdapterState OBJECT-TYPE
    SYNTAX INTEGER
    {
        coreAdapterRunning (1),
        coreAdapterSuspended (2),
        coreAdapterFailed (3),
        coreAdapterUninitialized (4)
    }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "This variable indicates the current operational state of
             the adapter."
    ::= { coreAdapterStatusEntry 2 }
```

Figure 2: An Example of SNMP MIB Description

```
coreAdapterState OBJECT-TYPE
    SYNTAX  INTEGER
    {
        coreAdapterRunning (1),
        coreAdapterSuspended (2),
        coreAdapterFailed (3),
        coreAdapterUninitialized (4)
    }
    ACCESS  read-only
    STATUS  mandatory
    DESCRIPTION
            "This variable indicates the current operational state of
             the adapter."
    SOURCE  { CDSVALUE [ADAPTER::Connect_State] }
    MAPPING
    {
        ["coreAdapterUninitialized" "NO_CONNECT"],
        ["coreAdapterRunning" "CONNECT"],
        ["coreAdapterFailed" "CONNECT_BROKEN"]
    }
    ::= { coreAdapterStatusEntry 2 }
```

Figure 3: An Example of SOURCE and MAPPING

3.2 Generated PIMIB

Given the extended MIB specification, it is not surprising that we can generate the whole SNMP protocol object as well as the PIMIB mechanically. The EMOSY compiler would parse the specification and check the information source. Then, EMOSY would search in the *network device information protocol object library* to see whether that information source is available or not. If it is available, then EMOSY would link mechanically the device protocol object code to the access method in the corresponding *CoreAttribute* [9] presenting that SNMP variable (e.g. **coreAdapterState**). For example, in the MIB specification, the first parameter in the SOURCE of **coreAdapterState** is "CDSVALUE," and therefore, in the generated PIMIB code, the type of the attribute is "cds_direct_attribute," which contains the code directly accessing the information from the CDS segment. Furthermore, in the same MIB, the SOURCE of **coreAdapterStatusLabel** is "LOCAL," which means the attribute type is "core_attribute." The PIMIB code generated by EMOSY for the **coreAdapterStatusEntry** consists of three parts:

- attribute identifier definition (figure 4);
- attribute type and creation definition (figure 5);
- class definition (figure 6),

and all these three parts are contained in the file **Gclass_coreAdapterStatusEntry.hh**. Since, the class "core_adapter_status_entry" inherits from "static_object," which inherits from "core_object," the generated class inherits the PIMIB generic interface (figure 7). Finally, the relationship between the *CoreObject* and the *CoreAttribute* is depicted in figure 8.

3.3 Registration Tree

Registration tree, which represents the containment hierarchy, is constructed for processing SNMP requests such as **GET**, **GETNEXT**, and **SET**. Each leaf as depicted in figure 9 would own a hash table to contain object instances under that leaf. For example, **coreAdapterStatusEntry** is a leaf, and all the instances (*e.g.,* **coreAdapterStatusEntry.101**) would be kept in its hash table. When processing a SNMP query like **GET coreAdapterState.101** (which is **coreAdapterStatusEntry.2.101** or **1.3.6.1.4.1.2.2.9.3.6.1.2.101**), we will perform the following steps:

```
#ifndef _PIMIB_Gclass_coreAdapterStatusEntry_hh_
#define _PIMIB_Gclass_coreAdapterStatusEntry_hh_
// Gclass_coreAdapterStatusEntry.hh
// -- note:    automatic Generated File -- do not edit.
// -- time:    Fri Sep 25 16:58:16 1992.
// -- project: protocol independent MIB.
// -- version: emosy version 2.1.
// -- group:   Network Management Systems.
#include <class_staticObject.h>
#include <adapter.hh>
#include <link.hh>
#include <interface_isode.h>
#include <coreMIB_defs.hh>
#define OI_coreAdapterStatusLabel          "1.3.6.1.4.1.2.2.9.3.6.1.1"
#define OI_coreAdapterState                "1.3.6.1.4.1.2.2.9.3.6.1.2"
#define OI_coreAdapterAdminState           "1.3.6.1.4.1.2.2.9.3.6.1.3"
#define OI_coreAdapterUpTime               "1.3.6.1.4.1.2.2.9.3.6.1.4"
#define OI_coreAdapterReceiverInError      "1.3.6.1.4.1.2.2.9.3.6.1.5"
#define OI_coreAdapterRedThresholdExceeded "1.3.6.1.4.1.2.2.9.3.6.1.6"
int MF_coreAdapterStatusEntry_coreAdapterState(int);   // MAPPING
```

Figure 4: The Generated PIMIB Code: Part I, Object Identifier Definition

1. use **coreAdapterStatusEntry** (or **1.3.6.1.4.1.2.2.9.3.6.1**) to get to the leaf.
2. use the value **101** as the hash value to get the object instance.
3. use the attribute identity **2** to get the attribute value in that instance.

Furthermore, a *ClassName* table is generated by EMOSY to transform from a class name to the address of its corresponding leaf in the registration tree. This classname table is useful in doing SNMP registration process [9]:

1. The *StaticObject* instance representing the table **coreAdapterStatusEntry.101** has no knowledge about the address of the leaf **coreAdapterStatusEntry**. However, it knows that it is belonging to the class "coreAdapterStatusEntry."
2. The registration process takes the string "coreAdapterStatusEntry" as input and performs a table lookup at the *ClassName* table.
3. If the leaf **coreAdapterStatusEntry** exists, then the registration process would update the hash table and insert an entry for the table instance **coreAdapterStatusEntry.101**.

The leaf **coreAdapterStatusEntry** is actually a "class" object because it contains information shared by all the instances in that class. And the class describing all the leaves in the registration tree is a "meta-class." Unfortunately, our implementation language, C++ [8], does not support this language feature, and thus, we have to build this *ClassName* table to simulate the functions of "class object" and "meta-class."

3.4 Object Creation Module

In certain situations, the number of object instances in the MIB is unknown until run-time or even dynamic. For example, the number of virtual connections in a network is usually not constant. Or, at run time, a new link-adapter could be added to the switch. Therefore, we need a *object creation module* to create *CoreObject* instances dynamically.

The difficulty in *CoreObject* instance creation is that the mapping between network devices and MIB object classes is not one to one (1:1) but one to many (1:m). For example, if a new link adapter is inserted into the managed node, we have to create an object instance for each of the following three MIB classes: **coreAdapterConfigEntry**, **coreAdapterStatusEntry**, and **coreAdapterStatisticsEntry**. Therefore, EMOSY finds mapping re-

```
core_attribute coreAdapterStatusEntryList[] =
{
    // coreAapterStatusEntry 1
    (core_attribute&) core_attribute
        ("coreAdapterStatusLabel", OI_coreAdapterStatusLabel,
         (mapping_ptr_func) NULL, READONLY, (int) 0),
    // coredapterStatusEntry 2
    (core_attribute&) cds_direct_attribute
        ("coreAdapterState", OI_coreAdapterState,
         "ADAPTER", MF_coreAdapterStatusEntry_coreAdapterState,
         READONLY, (int) 0),
    // coreAdapterStatusEntry 3
    (core_attribute&) core_attribute
        ("coreAdapterAdminState", OI_coreAdapterAdminState,
         (mapping_ptr_func) NULL, READWRITE, (int) 0),
    // coreAdapterStatusEntry 4
    (core_attribute&) counter_attribute
        ("coreAdapterUpTime", OI_coreAdapterUpTime,
         (mapping_ptr_func) NULL, READONLY, (int) 0),
    // coreAdapterStatusEntry 5
    (core_attribute&) cds_direct_attribute
        ("coreAdapterReceiverInError", OI_coreAdapterReceiverInError,
         "LINK_ADAPTER", (mapping_ptr_func) NULL,
         READONLY, (int) 0),
    // coreAdapterStatusEntry 6
    (core_attribute&) cds_direct_attribute
        ("coreAdapterRedThresholdExceeded",
         OI_coreAdapterRedThresholdExceeded,
         "LINK_ADAPTER", (mapping_ptr_func) NULL,
         READONLY, (int) 0),
    // coreAdapterStatusEntry 0
    (core_attribute&) core_attribute
        ("END_OF_LIST", (OI_class *) NULL,
         (mapping_ptr_func) NULL,
         NOTACCESSIBLE, (int) 0),
};
```

Figure 5: The Generated PIMIB Code: Part II, Attribute Type Definition

```
struct INDEX_DESC
coreAdapterStatusEntryOffsetTable[] =
{
    {"coreAdapterState",
     (((int) &LINK_ADAPTER::Connect_State) - 1)},
    {"coreAdapterReceiverInError",
     (((int) &LINK_ADAPTER::Adpt_Counters) - 1) +
     (((int) &LNK_ADAPTER_COUNTERS::G_Receive_Sig) - 1)},
    {"plaAdapterRedThresholdExceeded",
     (((int) &LINK_ADAPTER::Adpt_Counters) - 1) +
     (((int) &LNK_ADAPTER_COUNTERS::Red_Pkt_Thres_Exceed) - 1)},
    {(char *) NULL, 0},
};
class coreAdapterStatusEntry : virtual public static_object
{
  private:
  protected:
  public:
    virtual char *className() { return(STRDUP("coreAdapterStatusEntry")); }
    coreAdapterStatusEntry
    (core_object *_parent, core_attribute **_key,
     int _indexCount, core_attribute *_pa_list);
    coreAdapterStatusEntry(core_object *_parent, core_attribute **_keyArray,
                           int _indexCount, core_attribute *_pa_list,
                           struct INDEX_DESC *_off_set);
    ~coreAdapterStatusEntry();
};
#endif _PIMIB_Gclass_coreAdapterStatusEntry_hh_
```

Figure 6: The Generated PIMIB Code: Part III, Class Definition

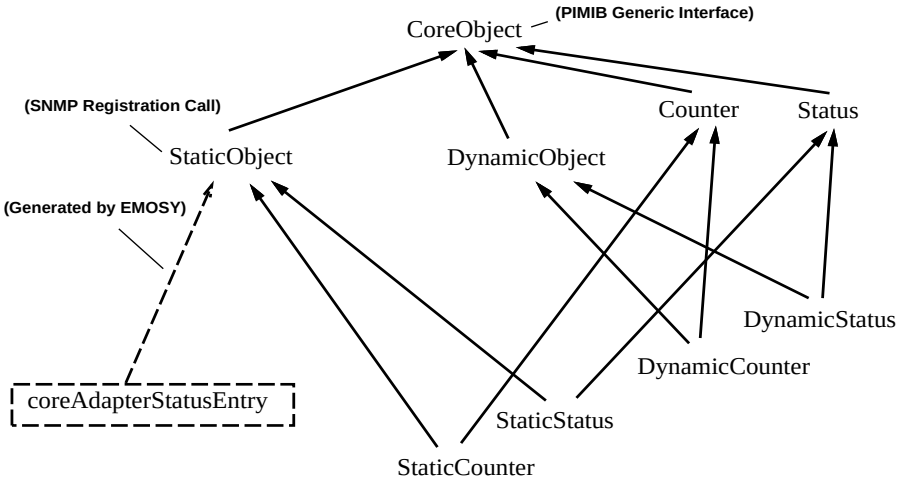

Figure 7: The Inheritance Hierarchy in PIMIB Object World

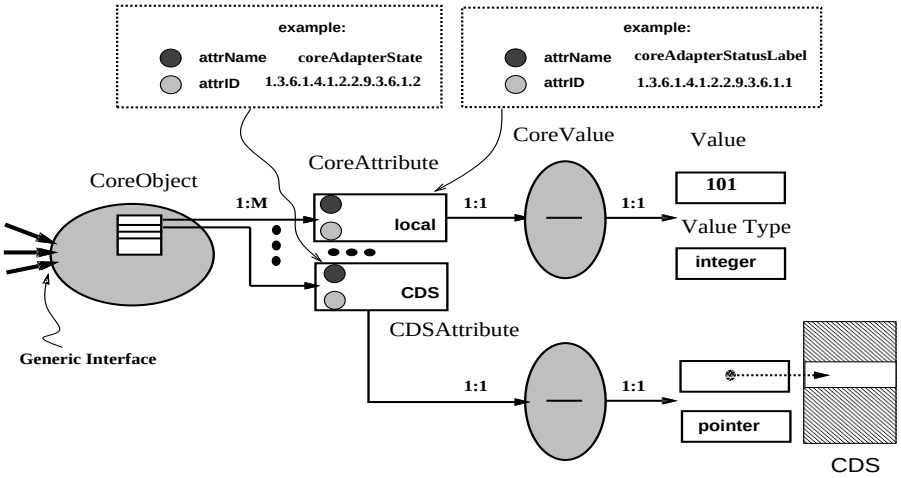

Figure 8: The Relationship between *CoreObject* and *CoreAttribute*

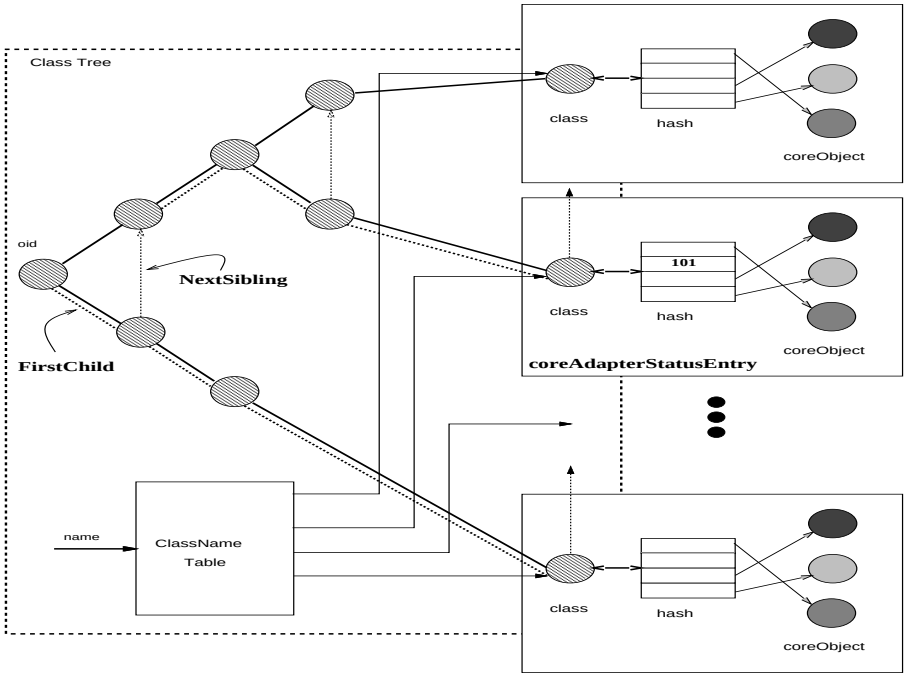

Figure 9: The Class Registration Tree from EMOSY

lations and encodes these relations into the object creation module that can be used in run time.

4 The EMOSY/PIMIB Architecture

In this section, a big picture for the integrated architecture of EMOSY and PIMIB is provided. From the bottom part of figure 10, a SNMP MIB description is extended with two key words **SOURCE** and **MAPPING**. Then, with another input, *network device information protocol object library*, the EMOSY module produces the modules of code that would be useful for the SMUX subagent as well as the PIMIB (protocol independent management information base). At run time, when the information about the existence of object instances is available, some *CoreObjects* are created in the PIMIB. In this creation process, the SNMP registration process is invoked. After receiving the registration request, the SMUX subagent searches through the *registration tree* from EMOSY and adds the new *CoreObject* instances to the hash table of one particular leaf in the tree. Furthermore, at the time a *CoreObject* is constructed, the *CoreAttributes* in this *CoreObject* are binded to certain device-dependent protocol objects and the binding relations are defined in the extended MIB specification (*i.e.,* **SOURCE** and **MAPPING**).

4.1 EMOSY as a Software Tool

In this paper, we present a software tool EMOSY, which would take an extended version of SNMP MIB specification as its input, to speed up the development process for the SNMP protocol object. Because of EMOSY's mechanical way to produce code for the SNMP protocol object directly from the specification, we have achieved the following:

Portability: Portability is improved in the sense that we only have to modify the extended MIB specification (*i.e.,* different SOURCE or MAPPING) but not the source code if the device information protocol object has already existed.

Extensibility: Changes to the MIB specification would not imply many changes the source code. The only thing we need to take care is to provide the SOURCE and MAPPING information when a new variable is added.

Reusability: For every device information protocol we develop, we can keep it in the *network device information protocol object library* and we can reuse the library later. Furthermore, the DMI interface [1] could also be included in the library as one particular protocol to access the device information.

4.2 EMOSY/PIMIB as a Network Management Agent Architecture

Usually, a SNMP MIB specification is only used in building the SNMP protocol processing module, *i.e.,* the software for agent, subagent, and applications. However, this specification is not utilized in other parts of a network management agent, and, this results an adhoc development process as well as an adhoc implementation.

As indicated in [9], the knowledge about the MIB specification should be known by the people who builds the MIB or the program who generates the MIB. In this paper, we show how an extended MIB specification can be used not only to build the management information base but also to link various kinds of network devices to the MIB in a systematic way. The result we get from this approach is an *object-oriented network management agent* because every information entity from the subagents to the network devices is modeled as an *object*. Furthermore, we classify the network device information protocols in the "glue" area. And, we feel this is an attractive approach to solve the adhoc implementation problem especially after we watch the efforts people made in developing the DMI specification [1].

5 Restriction and Future Works

One restriction about our approach is that, in a SNMP table entry, two variables belonging to the same column must share the same SOURCE and MAPPING information. This implies that, for instance, if **coreAdapterState.101** is from *CDS* with an offset *[ADAPTER::Connect_State]*, then **coreAdapterState.99** should have the same source information. However, this does not imply that they would access the same value because the label values "101" and "99" would be the first index to search in control data segment. Then, we would use the same offset to retrieve the information.

Since the goal of PIMIB is to support both CMIP and SNMP, while EMOSY is only for SNMP, one major future work would be to build a similar tool for CMIP.

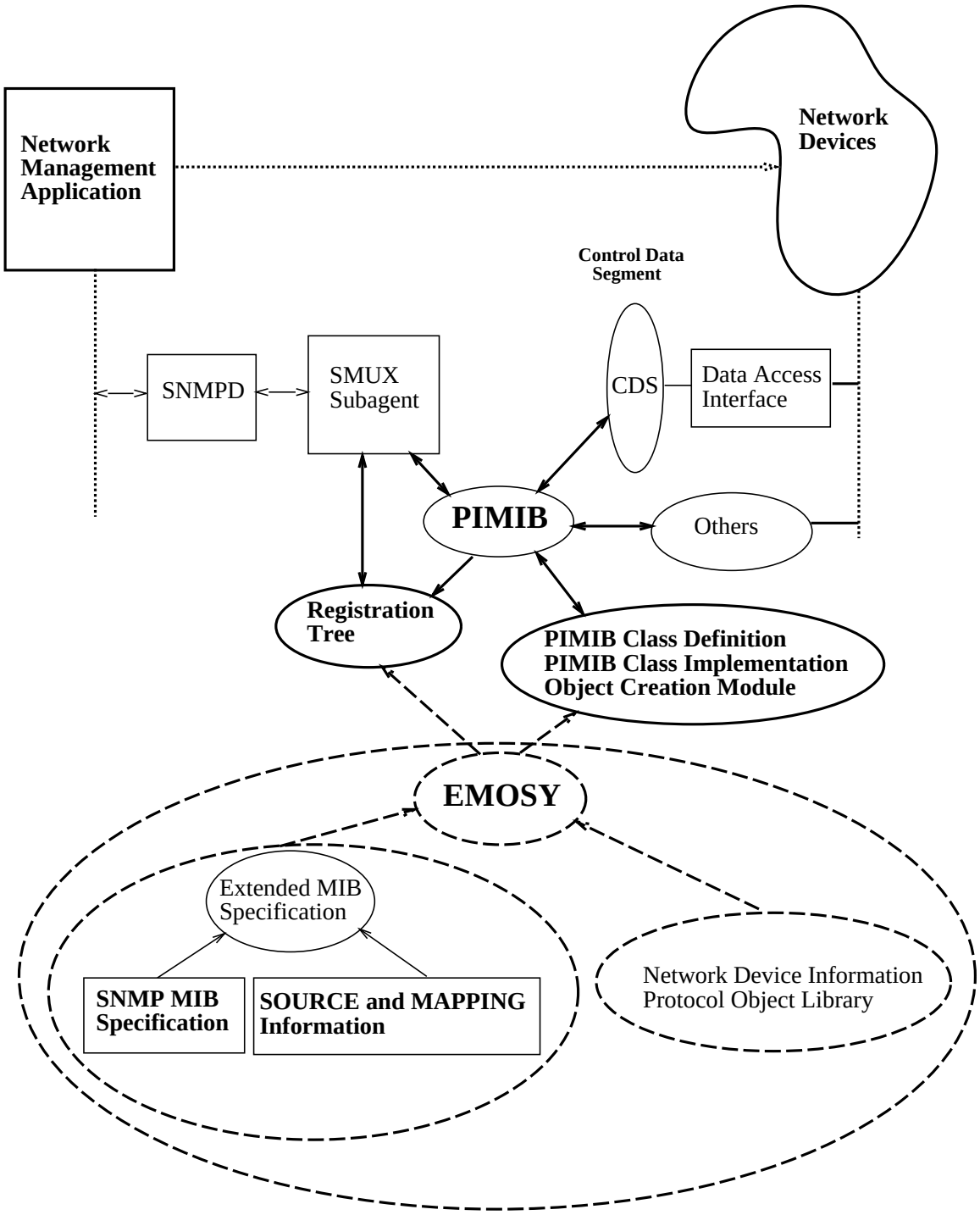

Figure 10: The Architecture of EMOSY/PIMIB

Acknowledgements

Pao-Chi Chang and Dave Levine had provided many valuable suggestions.

References

[1] Desktop Management Task Force. Desktop Management Interface Specification. Draft 3.8.2, March 5 1993.

[2] J. Embry, P. Manson, and D. Milham. Interoperable Network Management. In I. Krishman and W. Zimmer, editors, *Second Integrated Network Management Symposium*, pages 29–44, Washington, D.C., April 1991.

[3] International Standards Organization - ISO. Information Technology - Open System Interconnection - Structure of Management Information. ISO/IEC 10165-4.

[4] S. Mark Klerer. The OSI Management Architecture: an Overview. *IEEE Network*, 2(2):20–29, 1988.

[5] Subrata Mazumdar, Stephen Brady, and David Levine. Design of Protocol Independent Management Agent to Support SNMP and CMIP Queries. In *Third Integrated Network Management Symposium*, San Fransico, California, April 1993.

[6] M. Rose. Network Management is Simple. In I. Krishman and W. Zimmer, editors, *Second Integrated Network Management Symposium*, pages 9–25, Washington, D.C., April 1991.

[7] M. Rose. *The Simple Book*. Prentice Hall, 1991.

[8] Bjarne Stroustrup. *The C++ Programming Language, second edition*. Addison Wesley, 1991.

[9] Shyhtsun F. Wu, Subrata Mazumdar, Stephen Brady, and David Levine. On Implementing a Protocol Independent MIB. In *Second IEEE Network Management and Control Workshop*, Tarrytown, New York, September 1993.