Multi-Management: application-centric approach

Alex Aizman
Netro Corporation
3860 North First Street
San Jose, CA 95134
USA
e-mail: alexa@netro-corp.com
phone: (408) 216-1550
fax: (408) 216-1555

Abstract

The pace of evolution in communications and network management and the need to provide new and emerging services gives rise to a whole range of "multi-management" issues. This paper discusses some problems resulting from the lack of compatibility between embedded application and management models. It discusses the requirement to either upgrade existing management technology or support multiple management models and protocols.

Part of the problem may be solved at the embedded software level. This paper presents SNMP Application Framework (SAF), the product specifically tailored to address SNMP-related issues. It shows how to decouple applications from management and support both SNMP and CORBA-based management, and at the same time preserve consistent application design.

Keywords

Embedded application, management agent, management information model, SNMP, CORBA, multi-management.

1. Introduction

It is impossible to provide new and emerging end-user services, like Internet telephony, video-on-demand and content-pushing, without being able to manage heterogeneous networks. There is a direct connection between the explosive growth of the Internet and the need for integrated management. At the same time it is not an easy task to manage, for instance, an ATM network where two different manufacturers supply equipment. The explanation for this is partly in the fact that there are major differences between existing management standards and multi-vendor technologies. There are fundamental differences in their design philosophy, in the way management information is represented, and how managed objects are structured and named [1].

Along with global connectivity, network devices acquire more computing power and memory. Nowadays it becomes viable and often necessary to either interoperate with a “foreign” management application, or to upgrade the management model, or to make a transition from one network management technology to another that is more sophisticated and up-to-date.

This paper is organized as follows. The first part discusses heterogeneous network management problems which every telecommunication OEM faces to a lesser or greater degree. A brief introduction to SNMP is included. It then presents SNMP Application Framework (SAF) [2]. Using a simplified fault management scheme for an example, the paper demonstrates how to implement SNMP tables for two application classes with inheritance. Finally, the *multi-management* potential of the approach is demonstrated using the example of dual SNMP and CORBA-based management.

1.1 Multi-management problems

The management model is typically specified in a number of flat files: SNMP MIBs, CMIP/GDMO MIBs, IDL files, etc. Some commonly used management alternatives are depicted in Figure 1a).

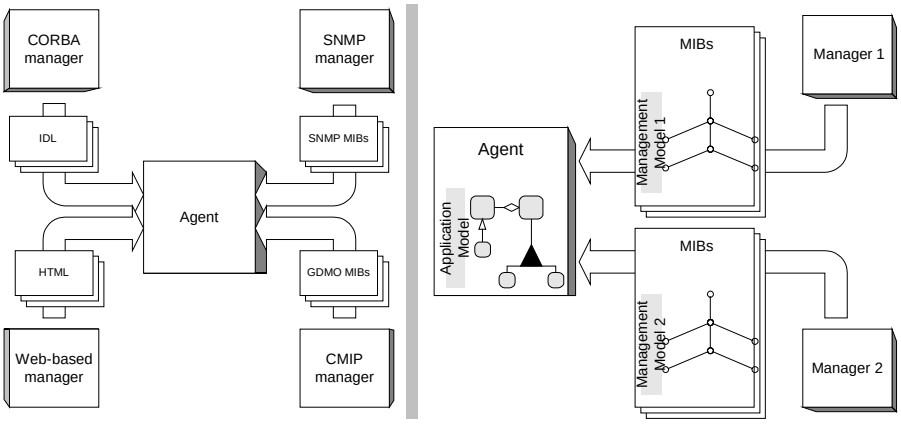

Figure 1: Multi-management: a) multiple management models (left)
b) multiple management specifications (right)

From a manager’s perspective, the agent *implements* the specification. Even if there is only one network management technology, it is often the case that several specifications describe the same managed entity (Figure 1b). One real-life problem that we had to solve at Netro [3] was to support standard ATM (RFC 1695, [4]) and proprietary Cisco MIBs, both of which model an ATM switch, but not quite in the same way. On top of it, there was a Netro proprietary MIB with additional parameters relevant to the point-to-multipoint topology of our system. Did we have to provide 3 different implementations, and in addition correlate between them to monitor and control one ATM switch? The answer is certainly “no.”

Generally speaking, specifications tend to be updated every so often, especially when it concerns worldwide standards and cutting edge technology. SNMP MIBs are revised on a regular basis. What happens when the MIB is outdated; when it is incomplete or ambiguous [5]? When the ATM Forum (or other standards body) has its own view, one that is not completely compliant with the IETF's approach?

Not only multiple management protocols and models (i.e., *multi-management* per se) but also the lack of compatibility between management and application models may adversely impact embedded application development. Many agent implementations incorrectly assume that "the information model used in the protocol is the only one they can work with" [6].

1.2 Alternative approaches

In terms of inter-domain management or system maintenance over time, *multi-management* might create a significant problem for device vendors, equipment manufacturers and the companies that provide network management solutions.

Technically, the problem is addressed by providing additional layers of indirection between the management application and the agent. Figure 2 shows two approaches:

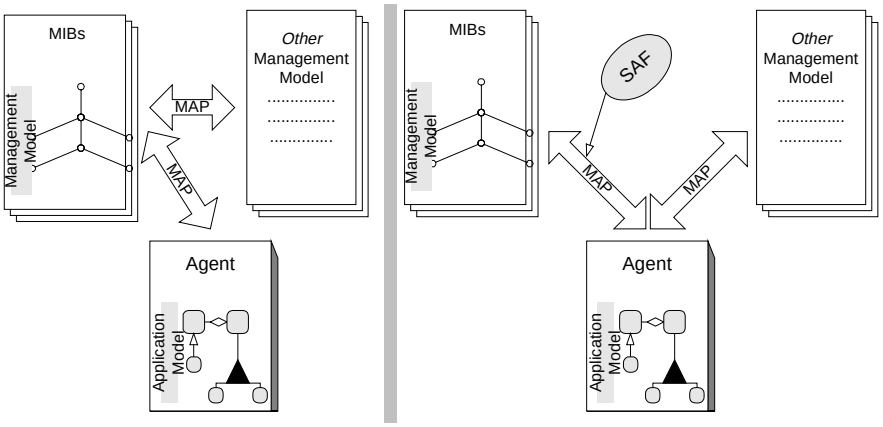

Figure 2: Mapping alternatives: a)MIBs ➪ ANY (left) b)application ➪ ANY (right)

One example of the *MIBs to ANY* approach is an implementation of dual SNMP and CMIP support (see [1], where GDMO MIBs are translated to SNMP MIBs). Another example is SNMP MIB into CORBA IDL conversion [8]. Commercial and non-commercial MIB to HTML converters [9] provide another similar approach. Figure 2a) shows the mapping layer implementation that may require some "cooperation" from the existing software but the overall impact is minimal. The drawback is, in my opinion, that it inadvertently promotes an already implemented management model, with all its restrictions and idiosyncrasies.

This paper takes an *application-centric*, or *application to ANY* approach (Figure 2b). The initial motivation behind this work was to shield developers from the restrictions and particularities of SNMP (i.e., by separating the SNMP tasks from the embedded application). An application model might be specified in OMT diagrams, or UML diagrams, or header files, but possibly (and preferably) not MIBs. The application model, and not the MIBs, is the primary source of mapping.

2. SNMP: Simple Network Management Protocol

SNMP is widely known and used, extensively documented and described (see [5, 6, 7, 10]). SNMP is specified in a series of inter-related standards (for SNMP version 2 specification see [11]); it is widely deployed because of "its low impact and low implementation cost in almost any network device" [6]. The SNMP model "is designed only for protocol and agent efficiency" [6]. To implement an SNMP agent, one usually follows the directions provided by the SNMP agent toolkit vendor and implements access routines (sometimes also called "method routines") to *get* and *set* values of managed attributes (Figure 3).

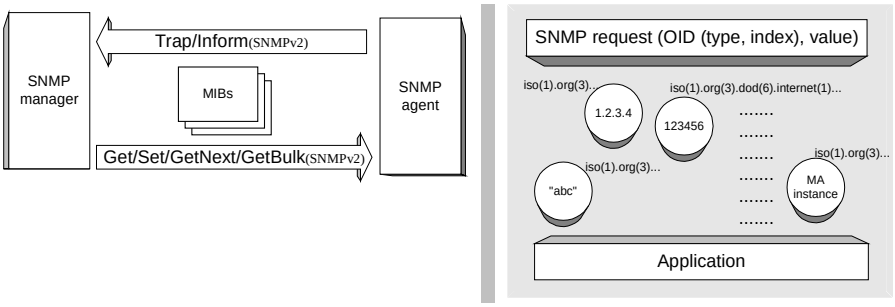

Figure 3: SNMP a) messaging (left) b) information model (right)

A managed object is a software abstraction of "any resource that an organization wishes to monitor and/or control" [7]. Objects like hubs and switches, modems and radios, network interfaces and connections encapsulate their properties and behavior, form application relationships and type hierarchies. At the same time SNMP defines management information as a collection of simple single- or multi-instance variables (Figure 3b). Object technology is an indisputable predominant paradigm for conceptualizing, building and deploying applications. With 32Mb of RAM and a 32bit CPU on the device, one can develop embedded applications using modern object frameworks and off-the-shelf components.

3. SNMP Application Framework

An SNMP Application Framework (SAF) product was completed in October 1997 ([2]). SAF is currently deployed in Netro P-MP (point-to-multipoint) wireless systems ([3]). It serves as a middleware, providing access to management information encapsulated inside application objects.

Agent implementation requires several skills, which in many cases means several people. One person should understand protocol and management information models. Another person codes the embedded system. A third person understands the technology that is being managed. At Netro every developer responsible for a managed component or service combines all these skills. SAF makes it possible and easy, since development is essentially confined to its own application domain.

SAF combines a Perl-based MIB compiler, called a Managed Table GENerator (MTGEN), and a C++ library. MTGEN produces an application layer that incorporates compile-time knowledge of MIB table schemas and indexes, variable types and access modifiers, default values and ranges, textual conventions and traps. Additionally, to support features that are insufficiently covered by SNMP, like object inheritance and object creation [5], MTGEN processes commented keywords and generates "aliases" (see section 3.4) and "object factories".

There is a naming convention between SAF and the application. The MIB variable name (by default), or its *logical* substitution specified in the MIB, must be used to construct instances of variables.

3.1 SAF terms

To eliminate ambiguity of the term "managed object" from here on, MIB variables are called *Managed Attributes* (*MAs*). Following is a brief introduction to other SAF terms (which are discussed in more detail in subsequent sections):

- Application Object (*AO*). In the context of this paper *AO* means a "big" object (as opposed to fine-grained MIB variables). SAF application objects contain managed attributes (*MAs*) and are of type *AoBase*.
- Managed Table (*M-table*) and Managed Row (*M-row*). *M-table* and *M-row* correspond to a MIB table and MIB table rows.
- Managed Attribute Reference (*MaRef*). *MaRef* is a special kind of a managed attribute that does not contain a value of its own and redirects SNMP operations to the "real" *MA* it references.

3.2 SAF architecture

From the perspective of SNMP request resolution, the agent is built of 4 distinct layers shown in Figure 4 below (notice previously introduced *AO, M-table, M-row, MaRef*).

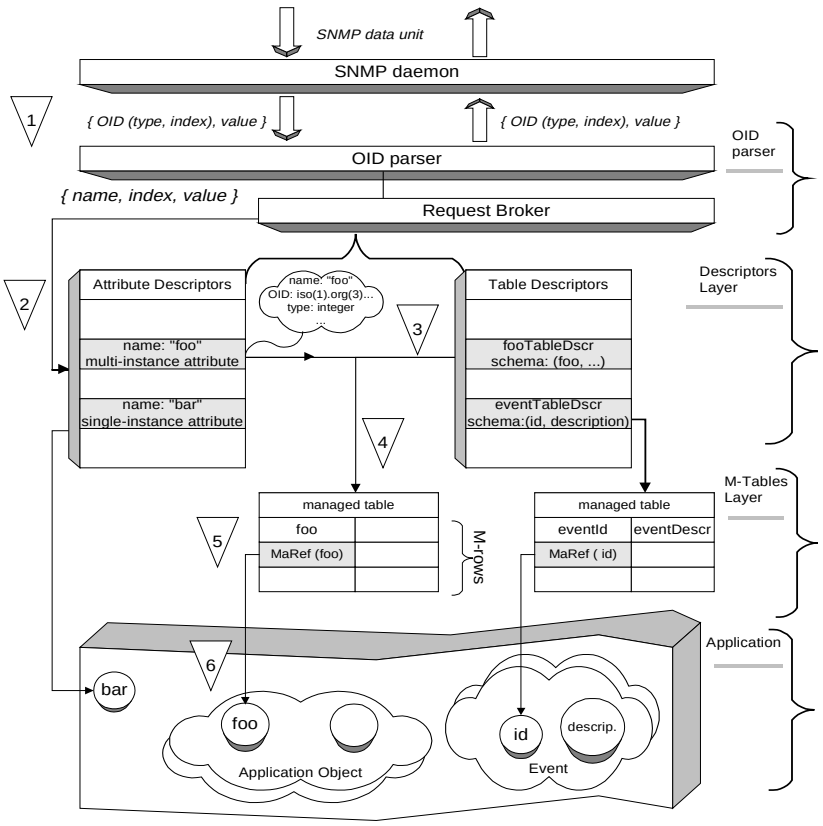

Figure 4: SAF architecture

The 1st layer, *OID parser*, is a thin layer interposed between SNMP implementation and the rest of the Agent. Its task is to translate OID from ASN.1 dotted notation to a pair of objects: *name* and *index*.

The 2nd layer, *Descriptors* is based exclusively on MIB definitions. SAF MIB compiler MTGEN code-generates its entire contents from the MIBs this particular agent is implementing. Once descriptors are created and initialized they stay intact throughout the agent's lifetime.

The 3rd, *Managed tables* or *M-tables* layer facilitates object-relational mapping. Creation and deletion of *AOs* causes ("behind the scenes") creation and deletion of one (in a simple case) or several *M-rows* per *AO*, respectively.

The 4th layer is the application per se. It is the one that performs specific tasks varying from agent to agent, from MIB to MIB, and from device to device.

The *Request Broker* object (Figure 4) owns the result of the MIB compilation: attribute and tables descriptors. To resolve an SNMP request, *Request Broker* performs the following steps (also shown and numbered in Figure 4 above):

Table 1: SNMP request resolution

Step	From		To	defined at
1	OID	➪	(name, index)	compile-time
2	Name	➪	Attribute descriptor	compile-time
3	Attribute descriptor	➪	M-table descriptor	compile-time
4	M-table descriptor	➪	M-table instance	compile-time
5	(M-table, index)	➪	M-row	run-time
6	(M-row, name)	➪	Attribute instance	run-time

Steps 1 through 4 are code-generated, while steps 5 and 6 depend on the application logic of creating and destroying object instances (Table 1).

Now we will take a "real-life" example and walk through the entire development process, from specification to implementation.

3.3 Example: events and alarms

Events report non-exceptional or exceptional developments in the managed system. In our simplified fault management scheme, an event will have only two attributes: *id* and *description*. The application *triggers* an event, which manifests itself by sending SNMP traps to the manager (or managers) and logging to the persistent event log. It may also result in some other application-dependent actions.

Network management emphasizes the role of exceptional (as opposed to informational) events, namely *alarms*. An alarm can be *raised* and *cleared*, it has *criticality* (minor, major), *status* (on, off) and *mask* (enabled, disabled) (Figure 5b).

There are a certain number of documented events and alarms in the managed system. Let's say, the manager wishes to browse registered events and alarms, configure (enable, disable) individual alarms and select active (that is, *raised*) alarms from all available alarms.

3.3.1 Application model

Figure 5a) depicts a typical SNMP view of the managed system. Figure 5b) shows application objects and their attributes. Notice that both the *Event* and the *Alarm* inherit *AoBase* - the base of all application objects. All managed attributes in the system inherit *MaBase*.

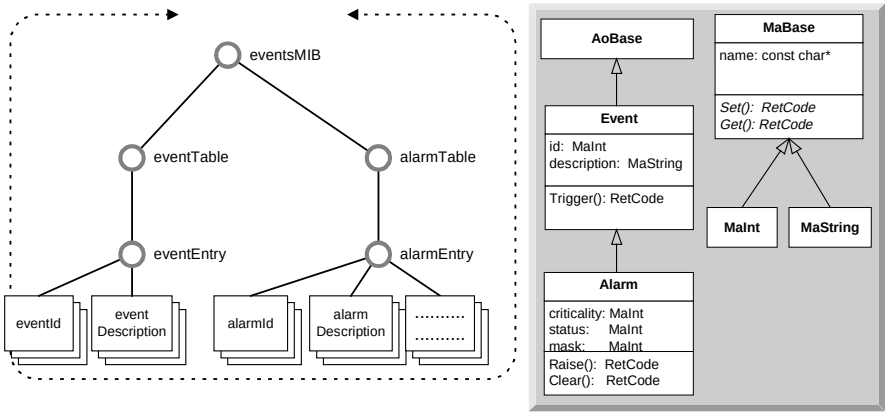

Figure 5: Events and alarms as: (a) eventsMIB (left) (b) static class diagram (right)

The first step is to translate MIB definitions into C++. Figure 6 shows a sample of event definition in the *eventsMIB* and the corresponding MTGEN-generated code:

```
EventEntry OBJECT-TYPE
    …
    INDEX       { eventId }
    ::= { eventTable 1 }
EventEntry ::= SEQUENCE {
    EventId          INTEGER,
    EventDescription String }
eventId OBJECT-TYPE
    SYNTAX           INTEGER
   …
   ::= { eventEntry 1 }
eventDescription OBJECT-TYPE
    SYNTAX           String
   ::= { eventEntry 2 }
```

```
DECLARE_MT_DESCRIPTOR(Event);

MtDscrEvent.Schema().Append(
      "eventId",
      MA_INTEGER_TYPE,
      MA_READONLY);
MtDscrEvent.Schema().Append(
      "eventDescription",
      MA_STRING_TYPE,
      MA_READONLY);

MtDscrEvent.GetSchema().SetIndex(
"eventId");
```

Figure 6: Event in: a) MIB (left) and b) generated C++ (right)

3.3.2 Construction and request resolution

To demonstrate the mechanism by which the construction of *AOs* completes the SNMP request resolution path (Table 1, steps 5 and 6).

```
class Event : public AoBase {
public:
Event(int eid, const char* d) :
  id("eventId", eid, this),
  descr("eventDescription", d,
        this)
…
MaInt      id;
MaString   description;
};
```

```
class Alarm : public Event {
public:
Alarm(int eid, const char* d,
      ALARM_CRITICALITY c, …) :
  Event(eid, d),
  Criticality("alarmCriticality",c,
              this)
…
MaInt    criticality;
};
```

Figure 7: Application object construction: a) event (left) b) alarm (right)

Let's consider the construction of an *Event* (Figure 7a) in more detail:

1) The names "eventId" and "eventDescription" are used to find attribute and table descriptors: *name* ➪ *attribute descriptor* ➪ *M-table descriptor* ➪ *M-table*.
2) The constructor of the managed attribute *Event.id* creates a new *managed row* (*M-row*) and inserts it in the respective *M-table*. This *M-row* will hold a reference to *Event* during the lifetime of this application object.
3) *Event.id* inserts a *reference* (*MaRef*) to itself into the newly created *M-row*. In a regular (that is, not optimized) case the *M-table* does not hold any values; instead it contains rows of references to managed attributes. The reference, a special attribute of type *MaRef*, relays SNMP *Get* and *Set* operations to real managed attributes encapsulated inside *application objects (AoBase*).
4) The *Event.description* constructor finds the newly created *M-row* for *this Event* (notice the parameter *this* in both event members' constructors, Figure 7) and inserts in it a new *MaRef* object: *MaRef (Event.description)*.

3.4 Object-relational mapping

Despite a general rule in SNMP that a MIB table row corresponds to a managed object, in many cases it is not so. Application objects tend to form complex relationships with multiple inheritance, containment and referencing, while an SNMP information model operates in terms of single-instance attributes and simple relational tables. To map tables on objects, SAF uses the mechanism of "multi-naming", or *aliasing*.

Aliases provide multiple SNMP paths to the same objects. Let's consider the following example. An alarm *is an* exceptional event, and therefore it has to be represented (or, more exactly, be visible to SNMP) in both *Event* and *Alarm* tables. Static aliases are specified in MIBs by inserting an ASN.1 comment with the keyword "alias". Figure 8 shows a sample of *eventsMIB* code with an alias for *alarmId*.

```
EventId OBJECT-TYPE
…
DESCRIPTION "Event ID"
::= { eventEntry 1 }
```

```
AlarmId OBJECT-TYPE
…
DESCRIPTION "Alarm ID"
-- alias name:  eventId
::= { alarmEntry 1 }
```

Figure 8: Event ID definition in a) event MIB table (left) and b) alarm MIB table (right)

Using compile-time information, the SAF MIB compiler generates an *alias* object for an *Alarm*. Upon *Alarm* construction, the framework takes care of populating both *Alarm* and *Event M-tables* with references to a single object (Figure 9).

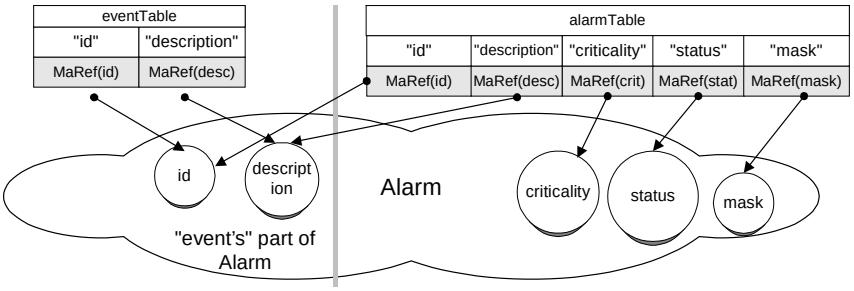

Figure 9: Object-relational mapping: a) eventTable row ➪ Alarm (left)
b) alarmTable row ➪ Alarm (right)

3.5 SNMP queries

One of the intrinsic problems with SNMP is that the manager has to perform numerous fine-grained requests so it can analyze and correlate data. Consider the MIB table of all alarms. In practice there may be dozens of alarms, each one "connected" to a piece of equipment, software module, service or such. Typically all alarms are either *clear*, or a few alarms are *active* (*raised*) and the majority remain *inactive* (*clear*). It is important for the manager to see a subset of all alarms: *active* alarms. Having access to the MIB table of all alarms, the manager is able to select *active* alarms (which in SQL terms would look like: *select * from alarmTable where status = on*). However, to optimize network traffic it would make sense to process "the query" on the server (i.e., agent) side. To accommodate this, we'd have to write a new MIB table, *activeAlarmTable*, which at any given time contains only *active* alarms. The question is; how can it be supported without duplicating data?

One possible approach is to use dynamic (run-time, instead of static compile-time) aliasing. When the "real" alarm is raised (Figure 10), it creates an *ActiveAlarm* object containing references (i.e., objects of class *MaRef)* to "real" alarm attributes, at which point the manager sees the raised alarm in the *activeAlarm* MIB table.

```
SAF_RET_CODE Alarm::Raise () {
   …
   m_active_alarm = new ActiveAlarm(*this);
   …
}
```

Figure 10: Dynamic aliasing

An alternative approach would be to override SAF implementation of an ActiveAlarm *M-table* methods *get* and *get-next*.

4. SNMP/SAF and CORBA

SNMP has a huge installation base and is still proliferating. Quite often dual management support is required for an existing SNMP agent or SNMP-based Element Management System (EMS). The challenge of heterogeneous management in this case boils down to interfacing with an "umbrella" NMSs of upper-tier CLECs

and Integrated Service Providers (ISPs). At the present time, northbound CORBA [12, 13] interfaces are used increasingly for this kind of interoperation. Where the SNMP manager performs several *get* and *set* operations, the CORBA manager acquires the referenced object and executes a single transactional operation on it.

4.1 Example: configuring connections

Let's first consider a simple example of a configuration management, or more specifically, connection configuration management. Being a quite common situation, it becomes more interesting when the network device has capabilities allowing it to configure different kinds of interfaces and provide different types of connections: ATM, Frame Relay, ISDN BRI, etc. Connections may have something in common, for example, every connection may be activated and deactivated (Figure 11):

```
class ConnectionBase : public AoBase, public ConnectionBase_skel {
   …
   SAF_RET_CODE  Activate ()   {return adm_stat.Set(ACTIVE);}
   SAF_RET_CODE  Deactivate () {return adm_stat.Set(NOT_IN_SRVICE);}
   …
   MaInt     adm_stat;
};
```

Figure 11: Connection base class for dual SNMP and CORBA support

To enable a connection, the CORBA manager performs the *connection->Activate()* operation on a CORBA object reference of the "right" connection. ORB supports polymorphic messaging and the call is dispatched and executed on the real target connection. The SNMP manager uses a different language for the same purpose, saying: "*set administrative status of the connection to active*" or something to that effect. This time SAF acts as a "dispatcher" interposed between multiple SNMP tables of connections and the application. The *set* request will reach the "right" *connection* object and, transparently to the application, modify the value of the managed attribute *adm_stat* (Figure 11).

4.2 Example: configuring interfaces

To proceed with the last example, a few introductory words about the company and the product are due. Netro ([3]) manufactures fixed wireless access (FWA)[1] broadband point-to-multipoint (P-MP) communication systems. A system consists of a base station, connected to an ATM network and multiple terminals at customer premises, providing a variety of access services: circuit emulation, frame relay, IP, etc. Figure 12 depicts a typical case when equipment is used to get access to IP networks (in this case, to Internet and Intranet via two different routers). For purposes of discussion, the entire P-MP wireless network is "collapsed" into a "black box" called "Netro", with IP-over-ATM (IPOA) interfaces at the CPE.

[1] Unfortunately, when describing telecommunication equipment configuration, we cannot avoid use of multiple acronyms. Here are some of them. CPE stands for Customer Premises Equipment, IPOA – IP over ATM, PVC - permanent virtual connection.

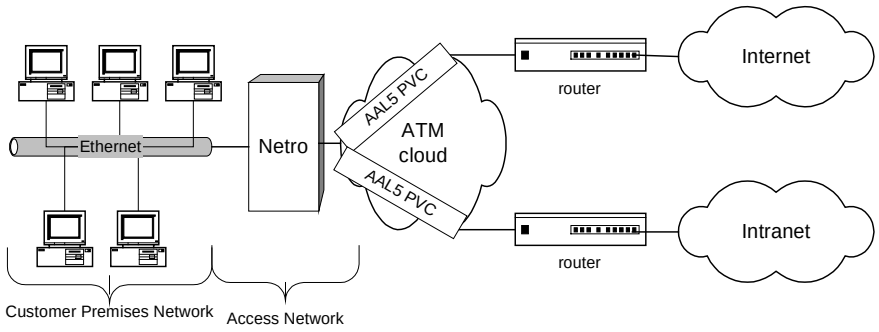

Figure 12: Configuration with IPOA interfaces

Since, generally speaking, this is a case of interworking between connection-oriented and connectionless protocols there should be a way to associate IP flow and a connection. At Netro we implemented just one (simple but useful) scenario when the IP route is one-to-one associated with a point-to-point link between two end points: "Netro" and an ATM-enabled IP router (Figure 12). More specifically, the point-to-point link is the AAL5 PVC that is used to transport IP datagrams according to RFC 1483 ([4]).

To configure the IPOA interface, the NMS has to specify its IP address, the ATM connection between it and a "next hop" RFC 1483 enabled router, and a type of encapsulating IP datagram. Initially we implemented the IPOA application object as shown on the left side of Figure 13.

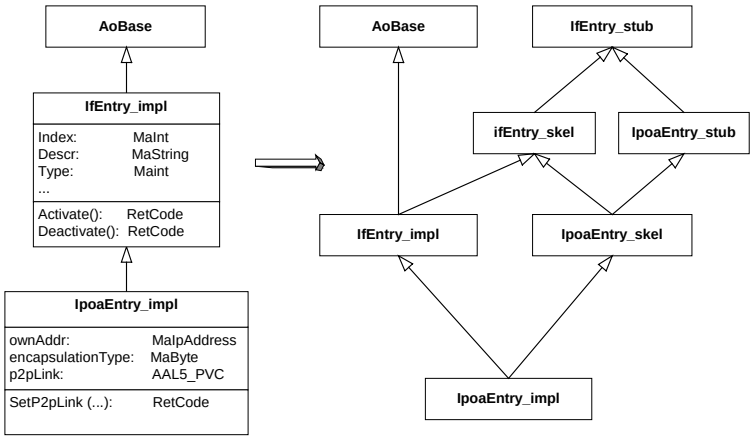

Figure 13: IPOA object: a) SNMP-only support (left) b) SNMP and CORBA (right)

To support SNMP, the IPOA object makes use of framework classes, *AoBase* and *MAs*. Other than that, it is a simple specialization of a MIB-II ifEntry (an entry in the MIB-II table of network interfaces).

Netro equipment, shown as a "Netro" box on Figure 12, is responsible for only one segment of the network, its "access" portion. To provide IP service, the network

management system has to be able to configure all other segments between the end-user and the ISP. Figure 13 demonstrates a transition of IPOA inheritance hierarchy. IPOA implementation becomes a CORBA object implementing the server IDL stub (also called *skeleton*) and exposing methods from the IDL-generated client stub *IpoaEntry_stub*. The new IPOA object facilitates the *application to ANY* approach, as it is shown on Figure 2b.

4.3 Discussion

As it was mentioned earlier (section 1.2), dual CORBA and SNMP support may be achieved by translating SNMP MIBs directly into CORBA IDL files [8]. Table 2 below highlights some of the differences.

Table 2: SNMP ➪ CORBA versus SAF-based application ➪ CORBA

	SNMP ➪ CORBA	*SAF application ➪ CORBA*
Source of IDL interfaces	MIBs	AO class definitions
Objects	Correspond to MIB table rows	1) Application objects 2) M-tables
Inheritance	Not supported	Supported

With SAF, inter-domain management between CORBA and SNMP becomes a technical issue of:

1) selecting a subset of CORBA objects from all application objects and "IDL-izing" their class definitions;

2) generating server IDL stubs and adding the existing implementation code.

That basically concludes the procedure of implementing dual SNMP and CORBA management for a SAF-based agent. Registering *M-tables* with the COS Naming Service [13] allows retrieval of *M-table* object references and subsequently (since *M-table* implements *get* and *get-next* operations), retrieval of *AOs* and their attributes.

The obvious and generic (but not optimized) approach would be to consider all SAF *AOs* to be CORBA objects. However, the overhead of implementing every *AO* as a CORBA object might appear to be too taxing in terms of performance and memory usage. The question of *how* to produce IDL files may be approached in a variety of ways. Often it is sufficient to publish an already existing method of an application object via IDL. In other cases a simple IDL-to-application adaptation is needed. Since the management interface is based on an existing functionality, this adaptation usually boils down to converting input parameters and/or calling several routines that are already implemented.

5. Conclusion

This paper discusses a range of problems arising from the lack of compatibility between embedded application and management models and the need to support multiple management models and protocols. The SNMP Application Framework example shows how to separate “management” and “application” concerns, preserve consistent application design and handle *multi-management* issues. The paper demonstrates one way of facilitating dual management support. In practice the tradeoff of changing the embedded agent has to be weighed against a non-intrusive approach involving CMIP/GDMO Q-adaptation (or CORBA-to-SNMP gateway) and therefore, an additional translation layer, performance deterioration, and inadequate mapping between information models.

References

[1] Mazumdar, S., Brady, S., and Levine, D., "Design of Protocol Independent Management agent to Support SNMP and CMIP Queries", Third International Symposium on Integrated Network Management, San Francisco, CA, 1993.

[2] Aizman, A., “Application framework for rapid agent development”, Third IEEE System Management Workshop, Newport, Rhode Island, April 1998.

[3] Netro corporate site, http://www.netro-corp.com

[4] Internet standards available from http://www.ietf.org

[5] Perkins, D., McGinnis, E. “Understanding SNMP MIBs”, Prentice Hall, 1997.

[6] Bapat, S., "Object-oriented networks: models for architecture, operations and management", Prentice-Hall PTR, 1994.

[7] Stallings, W., "Networking standards: a guide to OSI, ISDN, LAN and MAN standards", Addison-Wesley, 1993.

[8] Mazumdar, S., "Inter-Domain Management between CORBA and SNMP", presented at Seventh IFIP/IEEE International Workshop on Distributed Systems: Operations & Management, L'Aquila, Italy, October, 1996.

[9] The Web-based Management Page: http://www.mindspring.com/~jlindsay/webbased.html

[10] Rose, M. T. “The Simple Book”, Prentice Hall (Second Edition), 1996.

[11] Case J., McCloghrie, K., Rose, M., and Waldbusser, S., “Structure of Management Information for version 2 of the Simple Network management Protocol (SNMPv2)”, RFC 1442, April 1993.

[12] Object Management Group, “The Common Object Request Broker: Architecture and Specification”, Revision 2.0, July 1995.

[13] Object Management Group, “CORBAservices: Common Object Services Specification”, November 1997, http://www.omg.org/corba/sectrans.html