AN SMING-CENTRIC PROXY AGENT FOR INTEGRATED MONITORING AND PROVISIONING

Emmanuel Nataf, Olivier Festor, Guillaume Doyen
The Madynes Research Team
LORIA - University of Nancy 2 - INRIA
615, rue du Jardin Botanique
54602 Villers-lès-Nancy, France
Surname.Name@loria.fr

Abstract:

The combined use of SNMP and policy based frameworks is growing very fast. From both an information model and programmation interface point of view, an integrated view is highly desirable. Several approaches have been proposed so far in the information model sphere. In this paper we present the experience gained in using the SMIng approach for building an integrated management environment that provides seamless integration of both policy provisioning and MIB monitoring. The developed management environment has been deployed for managing an active network called FLAME, dedicated to dynamic IP monitoring.

Keywords: SMIng, management platform, active networks, integration

1. INTRODUCTION

During the last few years, many attention has been brought to novel information modelling techniques, management services and protocols. Results from these efforts are very different in nature mostly because they emerge from different communities with very different requirements (e.g., backward compatibility with a legacy approach or technology conformance). As part of the Operations and Management area of the IETF, a working group was created back in 2000 to address the issue of the evolution of the SNMP Structure of Management Information Version 2 (SMIv2) [11, 12, 10]. Part of this evolution is also concerned with a subset of the policy management framework related to provisioning (COPS-PR [3]) and the associated policy information specification technique (SPPI [9]).

While working on a management framework for an active network environment called FLAME, we faced the need to deal within management applications with both management policies in a provisioning framework and monitoring services relying on SNMP. One of the goals we had in mind, when looking at possible solutions was to provide an unique view of objects to the management applications. Thus, we naturally looked at SMIng *Next Generation Structure of Management Information* which provides a neutral object model and offers mapping facilities to both SNMP-SMI and COPS-PR SPPI.

FLAME is an active network based architecture developed within our research group dedicated to the hosting of IPv6 monitoring and management services. It in-

cludes a BSD native execution environment which can host active applications, a set of servers from which active applications can be downloaded and a management environment dedicated to the configuration and monitoring of the active management framework itself. The management interface of a node is divided into three parts: a command line interface for configuration and activation purpose, an SNMP agent implementing all objects of MIB-II together with objects dedicated to the active node monitoring, and a policy enforcement point (PEP) through which the node configuration is downloaded and activity policies enforced. This environment has been used for various management purposes among which multicast monitoring [13].

In this paper we share the experience gained with SMIng at both management information specification and mapping levels, as well as at a programmatic and engineering level within a management platform. At the information modelling level, we both performed reverse engineering from SPPI and SMIv2 specifications to SMIng classes and we extended the model with specific SMIng objects that were later mapped to SNMP and/or COPS-PR respectively. For the platform part, we describe the design and the development of this SMIng environment and show how it was applied to the FLAME execution environment, used here as the managed environment and not as the management platform.

The remainder of the paper is organised as follows. Section 2 presents the needs of management and provisioning in FLAME. From theses needs, we present an SMIng specification in section 3. Section 4 describes the architecture of the proxy between SMIng objects and management/provisioning information on the network. Section 5 details how we built a proxy agent from SMIng specifications and how it was integrated with the Java JMX environment. The use of the proxy for FLAME is shown in the section 6. Related work and conclusions are given in sections 7 and 8.

2. FLAME MANAGEMENT AND PROVISIONING

As already described in the introduction, FLAME is an active network dedicated to host IPv6 monitoring services. The architecture of a FLAME node is illustrated in Figure 1. It is close to several other active nodes like the ASP EE [2] on which it was initially built.

Basically, a node is divided into three parts: the standard routing engine of the node on which the environment resides, the execution environment (EE) which offers the basic services to active applications (dynamic code loading, access to node resources, naming, . . .). Active applications (AA) are executed on execution environments and use in addition to standard services, dedicated APIs provided by the FLAME environment (e.g., a packet capture API, a multicast routing table manipulation API, . . .). These APIs are also globally named in FLAME and can be dynamically deployed through a FLAME specific management operation within a node. Each element within a FLAME node is under the control of a standard management entity through which configuration and control is performed. Each node today hosts one SNMP agent as well as a PEP for enforcing configuration choices. We use separate technologies for management and provisioning, with SNMP and COPS-PR, because some of their specific properties match well with the FLAME environment. Monitoring information is made of several counters modelling active node activity. The role concept of COPS-PR is a way we use to distinguish some FLAME node (border gateway, community of nodes, . . .). Underlying TCP connections reduce the needed complexity for the critical operation of policy rules dowloading.

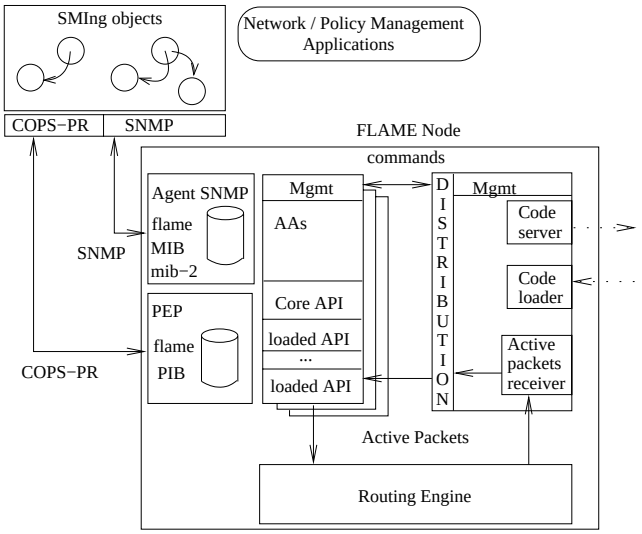


Figure 1. Architecture of a FLAME node

The need of a common information model appears with the fact that several faults can be deduced from network management data and can be reduced by policy-based configuration. For example, some policy rules are listed below:

- authorization policies, e.g., the active application `MRMMonitor` can be launched on the node, if its instantiation parameters are valid (e.g., identity of the code server for the active code, identity of the principal who wants to launch the application);
- obligation policies, e.g., an active application needs to end correctly on 99% of its executions, an active application cannot use more than 20% bandwidth;
- configuration policies, e.g., APIs that are loaded in the FLAME node, . . .

The FLAME SNMP MIB includes MIB-II objects and several FLAME specific ones like: the number of incoming/outgoing octets per active application, the number of active messages processed by an active application, the owner of an active application, the number of shutdowns of an active application, . . .

Many of theses informations could be related as when an active application shows a number of crashes, policy rules should be changed to stop any further launch of the suspicious application. With many such relations, management applications are less complex with an unified interface than with differents protocol APIs. SMIng appears to be one way to seamless integrate SNMP network management and COPS-PR/SPPI policy rules management. To unify the management information model dedicated to FLAME, we have built an SMIng specification for the FLAME specific objects put in the SNMP agent as well as SMIng classes for provisioning policies (upper left part of Figure 1).

3. SMING

SMIng (Structure of Management Information next generation) is a proposal that was submitted to the sming IETF working group [6]. This proposal is not the sole candidate for standardisation (SMI-DS *Data Structure* is an example of another possible approach) but has the advantage of being object-oriented like, and not being bound to any underlying approach while offering compatibility with both SNMP-SMI and SPPI.

While SMIng is described in the related drafts, we shortly present its structure through an example (Figure 2). It contains the definition of management information which model statistical measures of active application executions and together with policy rules used to decide if an active application could be launched.

```
(1)  typedef Counter32 {
(2)    type Unsigned32;
(3)  };
(4)  typedef Counter64 {
(5)    type Unsigned64;
(6)  };
(7)  typedef AAName {
(8)    type OctetString(255);
(9)  };
(10) typdef EEName {
(11)   type OctetString(64);
(12) };
(13) typedef AAVersion {
(14)   type OctetString(8);
(15) };
(16) class AARunning {
(17)   attribute Counter32 run;
(18)   attribute Counter32 crash;
(19)   attribute Counter32 end;
(20) };
(21) class AANetwork {
(22)   attribute Counter64 inOctets;
(23)   attribute Counter64 outOctets;
(24) };
(25) class AAStats {
(26)   attribute AAName name;
(27)   attribute AARunning run;
(28)   attribute AANetwork net;
(29) };
(30) class AAThresold {
(31)   attribute Counter32 crash;
(32)   attribute Counter32 bandwith;
(33) };
(34) class AAInstance{
(35)   attribute AAName name;
(36)   attribute EEName env;
(37)   attribute AAVersion ver;
(38)   attribute AAThresold thr;
(39) };
(40) snmp{
(41)   table ActiveApplStats {
(42)     oid flame-mib.2.3;
(43)     index(1);
(44)     implements AAStats {
(45)       object 2 name;
(46)       object 3 run.run;
(47)       object 4 run.crash;
(48)       object 5 run.end;
(49)       object 6 net.inOctets;
(50)       object 7 net.outOctets;
(51)     };
(52)   };
(53) };
(54) copspr{
(55)   prc AAInstancePolicyRule {
(56)     oid flame-pib.1.4;
(57)     pibindex (1);
(58)     implements AAInstance {
(59)       object 2 name;
(60)       object 3 env;
(61)       object 4 ver;
(62)       object 5 thr.crash;
(63)       object 6 thr.bandwith;
(64)     };
(65)   };
(66) };
```

Figure 2. SMIng management information specification

3.1 Type and class definitions

SMIng is designed for the definition of data interfaces. Thus, there is no procedural or functional statement support but only data oriented specifications. SMIng provides a set of basic data types like `OctetString`, `Unsigned32`, etc. From these types, new ones can be defined through the use of the `typedef` statement. Examples of such `typedef` statements are shown in lines 1 to 15 on Figure 2 (some `Counter` and `Name`).

SMIng provides a `class` statement to define object classes that are composed of `attribute` and `event` statements and can use simple inheritence (not shown in this

paper). Examples of class definitions are shown in lines 16-39 (`AAStats`, `AAInstance` and contained classes `AARunning`, `AANetwork` and `AAThresold`).
Class `AAStats` models statistical counters for executions status and network use of all active applications designed by its name in the `AAName` attribute. Class `AAInstance` models a policy rule for an authorized AA to be instanciated with its obligations (i.e numbers of crashes and network bandwith use).

Class attributes can be either of base types (defined through the `typedef` statement as shown in lines 17-19, 22, 23, 26, 31, 32, 35-37) or of another class (lines 27, 28 and 38).

At this level, one can specify when a new active application, say *aa* is ready to be downloaded by the FLAME node, an instance of the `AAInstance` class is created in all nodes that will use *aa* to enable launches. The name *aa* and other property values are given respectively to the defintion of the class (line 35). The first launch in a node will be followed by the creation of a `AAStats` with the name *aa* and counters start to be updated. Further launches of *aa* will equally update the `AAStats` instance.

A fault state appears when an active application shows a number of shutdowns in the `crash` attribute of `AAStats` greater than the same attribute in the `AAInstance`. This case could be detected by a polling procedure (notification reception through an SMIng event is possible but not used here) and the following action should be the deletion of the `AAInstance` object. Therefore, already launched applications could (maybe correctly) finish but no other launch of this application on this node can be done. On the other hand one can choose to increase the crash number threshold.

3.2 Mapping specification

The core of SMIng is protocol independent and can be used for both network and policy management as well as other domains. As these approaches strongly rely on dedicated information models, albeit sometimes very similar, and specific protocols, SMIng offers a facility to express the specification of a mapping from classes and their attributes to protocol specific information. As an example, we show both a SNMP (lines 40 to 53) and a COPS-PR (lines 54 to 66) mapping specifications. Each mapping specifies which SMI table or SPPI provisioning class of the existing MIB and PIB for FLAME implements some SMIng object class. The `oid` part (lines 42 and 55) gives the global object identifier (other SMIng constructs allow a full definition of the object identifiers for `flame-mib` and `flame-pib`). Following is the index column(s) specification of the table or PRC and the implemented SMIng class (lines 44 and 58). The `object` statement maps a column identifier to an SMIng class attribute. If an attribute is itself a class, a dotted notation is used to go until a simple value attribute is found (lines 46-50 and 62, 63).

4. A SMING-BASED PROXY AGENT ARCHITECTURE

Remember that our goal is to build a set of tools that enable SMIng specifications to be used in the core of management platforms as a unique data model. We therefore need to instantiate SMIng objects in order to use them as programmatic object instances.
We expect management applications to become less complex by hiding specific pro-

tocol dependent operations and as a consequence, to be more homogeneous when processing both policy and network management information.

The architecture that was designed can be seen as an SMIng distributed middleware hosting SMIng object instances. Groups of objects are hosted by several integration agents which are SMIng proxies. This is illustrated in Figure 3.

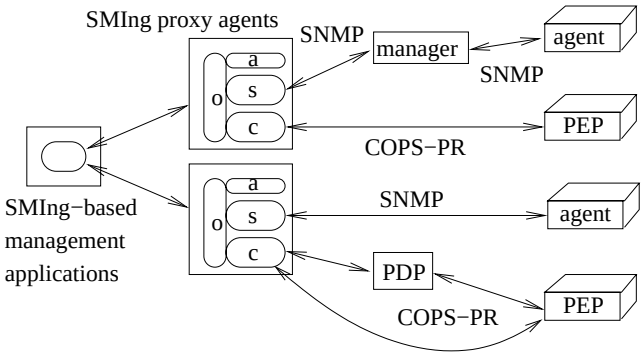


Figure 3. General architecture

An SMIng proxy agent contains an object view (interface **o**) that can be used by object-oriented management applications to access, modify, create or delete managed object instances. These managed objects can then be mapped to underlying resources through two main interfaces:

- interface **s** can be seen as an SNMP manager that has direct access to local or remote SNMP agents. Through this interface, mapping between SMIng and SNMP SMI is automated according to the rules defined in `snmp` statement implementation specifications;
- interface **c** can be seen as a policy decision point (`PDP`) that pushes policy information to policy enforcement points (`PEP`) using the COPS-PR protocol. Here we stick to the mappings defined in `copspr` statement implementation specifications.

A third interface (named **a**) represents access to internal SMIng objects.

5. IMPLEMENTATION

5.1 Generation of object instances

To implement the proxy architecture for SMIng objects, we chose the Java Management eXtension (JMX) framework[1]. Figure 4 contains an illustration of an SMIng proxy agent that includes managed objects taken from the example given in Figure 2.

Only Java objects corresponding to `typedef` values are protocol dependent (`AAName` or `Counter32`). Other SMIng objects are created after the instanciation of contained

[1] http://java.sun.com/products/JavaManagement/

objects (`AAStats` is created after `AARunning`) and linked to each other from the implementation specification. In doing so, the process of object instanciation never leads to "null pointer" troubles.

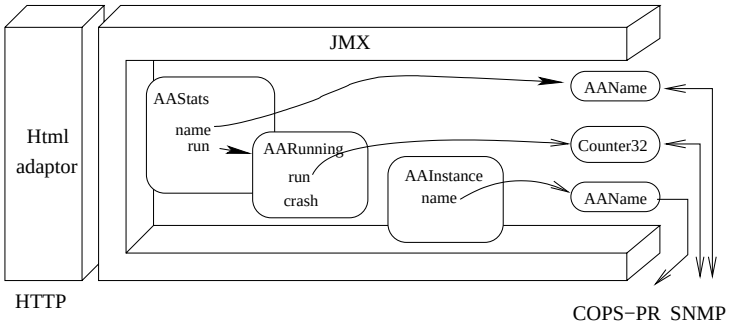
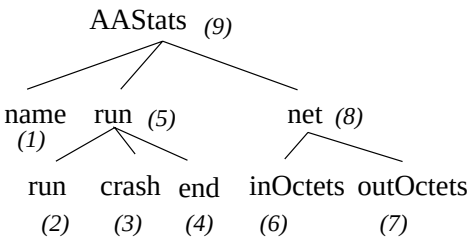


Figure 4. SMIng object instances in a JMX container

Figure 5 shows how the instance creation process is automatically generated from the implementation specification. On the left part of the figure there is a schematic representation of the implementation. It is always a tree structure that is given by our SMIng parser and followed by our proxy agent generator. The root is the name of the class. Intermediate nodes are always attributes that reference a class, while leaf objects are simple types. The right part shows a subset of the generated Java code for this tree. The code of the lines 1 to 6 is for the creation of SMIng `typedef` objects. Parameters of these objects are generated following the object-identifier naming of the implementation (here the `ActiveApplStats` as in Figure 2 line 41) and the remote SNMP agent from which the MIB is translated. Once these leaf objects are created, the generated code creates SMIng object instances (lines 7-9) and sets their attributes through its constructor method invocation The same code is generated for the `AAInstance` class with its attributes.




```
(1) name = new AAName(...);
(2) runrun = new Counter32(...);
(3) runcrash = new Counter32(...);
(4) runend = new Counter32(...);
(5) netinOctets =
              new Counter32(...);
(6) netoutOctets =
              new Counter32(...);
(7) run = new AARunning(runrun,
              runcrash, runend);
(8) net =
        new AANetwork(netinOctets,
                      netoutOctets);
(9) aastat = new AAStats(name,
                         run, net);
```

Figure 5. Implementation scheme generation

This code is generated for each SMIng class implemented directly by an SNMP table. When the processed group is a table, there will be exactly as many object instances created as the number of lines in the table for each SNMP managed object. In

order to do this, we use the `get-next` SNMP request, starting with the table SNMP object identifier and walk through the table in order to get the values we need to create the leaf objects first, and end with containing objects. When SMIng classes are implemented by a scalar group there should be only one object created by the SNMP agent proxy to SMIng and we use one `get` request for all needed values.

When the underlying mapping is COPS-PR, the approach is similar. Mapping to SPPI can even be seen as a subset of the mapping from SMIng to SMIv2 since only tables can be defined using SPPI (no scalar) and these tables all hold an unique and known index. A major difference is that SPPI values in PIB should not be read from the PEP, as it is the case for SMI values from agents MIBs. Instead the creation of such one SMIng instance should be mapped to a provisioning COPS-PR request for the corresponding rule in the PEP (or in a PDP that will forward rules as in Figure 3).

5.2 Naming scheme

Within a JMX container, object instances are named according to a standard naming scheme, very close to the OSI Distinguished Name pattern. The general form of an instance name is:

```
Domain::attr1 = value,attr2 = value ...
```

where `Domain`, each `attr`*i* (attribute) and `value` are character strings. Within our SMIng mapping entity, we use this naming convention to uniquely identify SMIng instances as follows:

- the domain identifier is the SMIng module name where the class is defined;
- attributes and values are defined as:
  - l = *SNMP agent or COPS-PR PEP DNS name or IP address*
  - c = *SMIng class name*
  - p = *position of the class in the containing class*
  - i = *instance number*

Every instance has a name which contains these attributes. Since JMX does not put any constraint on the order of attribute occurrences in a name (the lexicographic ordering is defined as the normal form), they can appear in any order. Figure 6 illustrates the naming with our FLAME example. The position of an instance in a containing instance is built from the containing SMIng class definition and from the instance number of the latter object according to its class identifier. For example the `AAStats` class (Figure 2 lines 25-29) has three attributes and their relative positions are 1, 2 and 3. We chose to give the value “1”to each SMIng object that is not contained in any other SMIng object. So objects referenced by `name`, `run` and `net` attributes have respectively “1.1”, “1.2” and “1.3” for their p naming attribute value. Contained SMIng objects have the same prefix than the containing SMIng one. We chose to keep an SMI object identifier like notation to reduce the length of object name. If the class is not contained by any other class, its name is the class name followed by same value for the position and the instance number. We keep this redundant information to always have the same number of naming attributes. The instance number attribute is given by the proxy agent for each object creation. If an SMIng object is contained by more than one other object, this former should have as many names. In any case, one name provides a way to access to exactly one object instance.

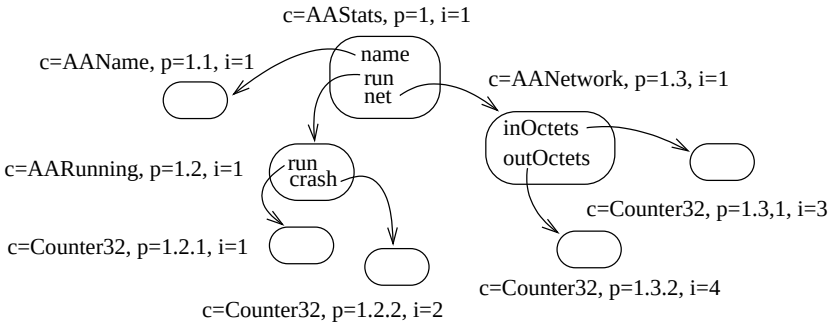


Figure 6. Naming of instances

5.3 Notification support

SMIng allows event definitions in object classes. Such a feature of object instances is only implemented by SNMP notification in the SNMP framework (COPS-PR does not support notifications). Figure 7 illustrates the use of events whithin SMIng objects. `System` and `If` are object classes that represent the system and the interface group of MIB-II. Events are defined in classes (lines 3 and 8), after attributes. The `snmp` statement (lines 10-21) contains the implementation of these events. Standard notifications (lines 11, 12 and 15, 16) are related to object events by a `signals` statement (lines 13 and 17) with a dotted notation (*ClassName* . *eventName*). If the trap carries SNMP object values that are also SMIng attribute values, a mapping can be specified (line 18). In this case, a `linkdown` notification received will update the `adminState` SMIng object attribute (line 6). Figure 8 shows how SNMP traps are catched by a dedicated ob-

```
(1)  class System {                          (12)     oid snmpTraps.2;
(2)    ...                                   (13)     signals System.warmStart{};
(3)    event warmStart;                      (14)   };
(4)  };                                      (15)   notification linkDown {
(5)  class If {                              (16)     oid snmpTraps.3;
(6)     attribute AdminState adminState;     (17)     signals If.linkDown {
(7)     ...                                  (18)       object If.adminState;
(8)     event linkDown;                      (19)     };
(9)  }                                       (20)   };
(10) snmp{                                   (21) };
(11)   notification warmStart {
```

Figure 7. SMIng events definition and implementation

ject playing the role of an SNMP trap catcher. Its role is also to map incoming SNMP notifications to java object that are forwarded to registered SMIng object instances. In the context of JMX, the trap catcher and SMIng objects (that are all JMX MBean objects) must implements specific interfaces. The `NotificationListener` interface allows SMIng object to receive notification signal by registering itself with the standard notification `oid` to a `NodificationBroadcaster` interface implemented by the `TrapCatcher`.

Generated SMIng events follow a similar operation, i.e., they are distributed using the JMX notification model.

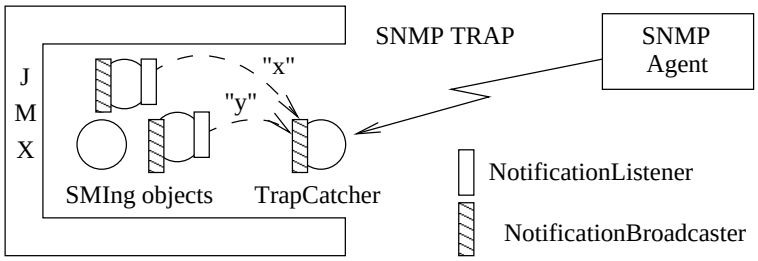


Figure 8. SMIng events support

Some standard notifications are related with SNMP table entries because they contain an index value that specifies which line of the table is the root of the generated SNMP trap.

When an SMIng class containing event definition is implemented by an SNMP table and when there is a standard notification that carries an index value then each SMIng object (one object per line) should be able to capture the corresponding event. This index value is used by the `TrapCatcher` to identify the SMIng object.

In addition, some notifications carry values that should be updated in the object instance, or be interpreted as the need to refresh some object attribute. We exploit this in conjunction whith a regular polling service in charge of maintaining the consistency between SNMP MIBs and the proxy.

5.4 Code generation tools

For generating the Java classes, we use an SMIng extension of the MODERES framework. MODERES is basically a set of Open Source Java tools maintained by our research group dedicated to the parsing and processing of multi-approaches management information models (GDMO/ASN.1 SMIv1, SMIv2, CIM-MOF, . . .). The toolkit is available on the group's web page[2].

Figure 9 shows the use of these compiler tools for the SMIng proxy generation. First a syntax and semantics check is performed on incoming SMIng specifications (both core definition and protocol mappings) by the parser. The specifications are then stored in a repository in the form of a decorated syntax tree. This repository is the source for the SMIng agent toolkit (`SMIngAtk`) that generates Java classes and interface mappings to SMIng classes. Note that SMIng `typedef` as well as events, are also mapped to Java classes and interfaces (dashed arrows). Additionally some Java classes are generated in order to realize the necessary operation to get/set values in a specific protocol (SNMP, COPS-PR).

[2]http://www.madynes.org

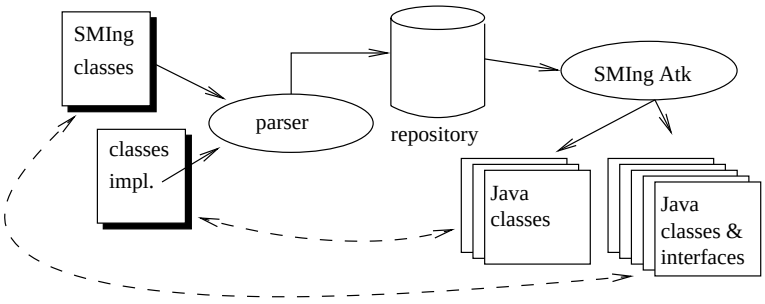


Figure 9. SMIng tools

6. APPLICATION TO FLAME

Having all components defined as SMIng objects was of great use especially for those applications which access objects mapped to the two worlds. For example, one policy application automatically updates the policy repository of all nodes, if it finds out that an application uses too many resources on a node (e.g., number of crashes). The updated policy forbids instantiation of the application with the parameters that cause the trouble in one node. Building such an application with our framework is straight-forward since one only needs to know the SMIng class that corresponds to this policy family and set up a monitoring service for the attribute that represents the number of crashes of an active application. To push a configuration policy that forbids the execution and further instantiation of an application, only the policy object `AAInstance` defined in Figure 2 lines 34-39 need to be instantiated. Once instantiated, the decision is mapped onto a COPS-PR service invocation and pushed towards the concerned PEP. A java.rmi adaptor to the SMIng proxy agent was designed to allow processing in our network management java application.

7. RELATED WORK

The object oriented network modelling is well described in [1]. The concept of Meta Managed Objects [14] contains the same base elements as those proposed by SMIng with a separate definition of data and their different representations. Other mappings exists from objects to TMN or SNMP management information [16, 4] or the opposite way e.g., from WBEM to OSI based management [5].

The *libsmi* project of the Technical University of Braunschweig [3] provides a library to access SMI information. A component of this library is an SMIng parser that allows a syntax and semantical analysis of modules. An HTML version can be tested on line at the Simple Web site[4]. An API provides access to MIB and PIB modules information to ease the development of management applications. Our project has a structure similar to the *libsmi* parser. The main differences are the programming language envi-

[3] http://www.ibr.cs.tu-bs.de/projects/libsmi
[4] http://www.simpleweb.org

ronment (C for *libsmi* and Java in our case) and the application domains (information model core tools for *libsmi* and agent toolkit in our case).

As mentioned in different parts of the paper, the FLAME environment configuration is done through policy-based management. The use of policy-based approaches for the management of active networks is investigated in several other places and is not explained in this paper. The reader will find studies on this topic in [15] and [8] for policies dedicated to resource management in active networks.

8. CONCLUSION AND FUTURE WORK

In this paper, we described a software architecture based on the SMIng approach and its application to the management of an active network infrastructure called FLAME (with is itself an environment dedicated to the management of IP networks).

Several lessons can be learned from this experience. First, the situation where both policy-based approaches and the standard SNMP framework need to be combined exists and the number of occurrences will probably grow in the next few years (e.g., the COPS model is proposed for provisioning and outsourcing in several 3G architectures). The second lesson is that SMIng appears to be a reasonable evolution in the standard framework, in the sense that it provides enough support for our needs namely object-orientation and automated integration of monitoring and provisioning. As it was demonstrated in this paper, the approach can be implemented to build an operational management framework. The third lesson, which is obvious, is that the design and development of applications that combine policy manipulation and MIB object access is much more convenient within a common framework. This has been demonstrated while we developed the applications for the active environment.

The first evolution of the presented work is to let the designed management framework follow the evolution of the outcome from future evolutions of IETF groups working on this topic. Work is still in progress on this subject and extended proposals will emerge. In parallel to this work, we are looking at how the mapping principles defined in SMIng can be reused in other approaches like WBEM since the same need for dynamic mapping will appear for these approaches as well. Finally, we will continue the refinement of policy definitions for managing the FLAME environment. The final goal is to end-up with a complete automated monitoring environment driven by pushing configuration policies into the active nodes.

References

[1] S. Bapat. *Object-Oriented Networks : Models for architecture, operations and management*. Prentice Hall, 1994.

[2] B. Braden, A. Cerpa, T. Faber, B. Lindell, G. Phillips, and J. Kann. ASP EE: An Active Execution Environment for Network Control Protocols. Technical report, USC/ISI, December 1999.

[3] K. Chan, J. Seligon, D. Durham, S. Gai, K. McCloghrie, S. Herzog, F. Reichmeyer, R. Yavatkar, and A. Smith. COPS Usage for Policy Provisionning (COPS-PR), RFC3084, March 2001.

[4] T.R. Chatt. TMN/C++: An object-oriented API for GDMO, CMIS, and ASN.1. In *[7]*, pages 177–191, 1997.

[5] O. Festor, P. Festor, L. Andrey, and N. Ben Youssef. Integration of WBEM-based Management Agents in the OSI Framework. 1999. in Integrated Network Management, VI, Sloman, M. and Mazumdar, S. and Lupu, E. editors,IEEE Press, Proceedings of the IFIP/IEEE 6th International Symposium on Integrated Management, Boston, MA, 24-29 Mai, 1999.

[6] F. Strauss J. Schoenwaelder. Next generation structure of management information for the internet. In R. Stadler & B. Stiller, editor, *Active Technologies for Network and Service Management, DSOM'99*

Zurich, Switzerland, pages 93 – 106. Lecture Note in Computer Science, IFIP/IEEE, Springer, October 1999.

[7] A. Lazar, R. Saracco, and R. Stadler, editors. *Integrated Management V*. IFIP, Chapman & Hall, May 1997.

[8] I. Liabotis, O. Prnjat, and L. Sacks. Policy-Based Resource Management for Application Level Active NEtworks. August 2001. Proc. Second IEEE Latin America Network Operations and Management Symposium, LANOMS'2001, Belo Horizonte, Brazil.

[9] K. McCloghrie, M. Fine, J. Seligson, K. Chan, S. Hahn, R. Sahita, A. Smith, and F. Reichmeyer. Structure of Policy Provisionning Information (SPPI), RFC3159, August 2001.

[10] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Conformance Statements for SMIv2., April 1999. IETF, STD58, RFC 2580.

[11] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Structure of Management Information Version 2 (SMIv2), April 1999. IETF, STD58, RFC 2578.

[12] K. McCloghrie, D. Perkins, and J. Schoenwaelder. Textual Conventions for SMIv2., April 1999. IETF, STD58, RFC 2579.

[13] H. Sallay, O. Festor, and R. State. A distributed Management Platform for Integrated Multicast Monitoring. pages 483–496, 2002. Proc. IEEE/IFIP Network Operations and Management Symposium NOMS'2002, R. Stadler and M. Ulema, editors, IEEE ISBN 0-7803-7382-0, Florence, Italy, April 2002.

[14] J. Seitz. Meta managed objects. In *[7]*, pages 650 – 660.

[15] M. Sloman and E. Lupu. Policy Specification for Programmable Networks. In S. Covaci, editor, *Active Networks: Proc. First International Working Conference, IWAN'99*, pages 73–84, Berlin, Germany, June 1999. Springer Verlag, LNCS 1653.

[16] N. Soukouti and U. Hollberg. Joint Inter Domain Management: CORBA, CMIP and SNMP. In *[7]*, pages 153–164, 1997.