Next Generation Structure of Management Information for the Internet

Jürgen Schönwälder and Frank Strauß

Technical University Braunschweig
Bültenweg 74/75, 38106 Braunschweig, Germany,
{schoenw|strauss}@ibr.cs.tu-bs.de

Abstract. Management Information Bases (MIBs) for use with the Simple Network Management Protocol (SNMP) are defined in a language called the Structure of Management Information (SMI). This language, based on an adapted subset of ASN.1 (1988), has been a constant source of confusion for Internet MIB designers, MIB implementors and users of network management products. This paper presents work towards a new SMI, tentatively called SMIng, which addresses some of the shortcomings of the current SMI. We describe the design of an embeddable C library to access SMI definitions as well as some tools using this library.

1 Introduction

The Internet Structure of Management Information (SMI) defines the language for describing Internet Management Information Bases (MIBs). The SMI is primarily used to define managed objects together with their data types and the notifications that can be emitted.

There are currently two versions of the SMI [1]. The first version of the SMI, called SMIv1, is defined in RFC 1155 [2], RFC 1212 [3] and RFC 1215 [4]. Work on SMIv1 started in 1988 and completed in 1991 with the publication of the SMIv1 specifications as Internet Standards.

The second version of the SMI, called SMIv2, is defined in RFC 2578 [5], RFC 2579 [6] and RFC 2580 [7]. SMIv2 was approved as a full Internet Standard in January 1999 and it will replace the SMIv1 standard. SMIv2 is an evolution of SMIv1 and offers some new features that were not part of SMIv1. Among the new features are macros to define conformance requirements for MIB modules and macros to describe implementation capabilities. The SMIv2 specifications also contain a set of general purpose data types (so called textual conventions).

The Internet Engineering Task Force (IETF) established a policy in 1994 which requires that all new MIB modules published in RFCs must use the SMIv2 format. This rule forced many vendors to add support for SMIv2. Furthermore, the possibility of automated conversion of SMIv2 MIB modules into SMIv1 MIB modules raised the acceptance of SMIv2 outside the IETF. Many vendors now use SMIv2 internally for maintaining their MIB modules, even if they distribute SMIv1 MIB modules to their customers.

Within the IETF, there are currently about 100 MIB modules on the standards track with definitions for several thousand managed objects and about hundred notifications. Additional MIB modules are still being developed within the IETF and other standardization bodies. Some vendor specific MIB modules reach the size of all IETF MIB modules taken together. These numbers clearly show that there is a non-trivial investment in MIB module definitions, and more important, MIB implementations. Future network management protocols are therefore forced to interface with Internet MIBs in order to protect the investment made during the past 10 years.

This paper is structured as follows. We first take a critical look at the current SMIv2 in Section 2. In Section 3, we present the core concepts of a proposed new SMI, tentatively called SMIng. Some examples of SMIng definitions are discussed in Section 4. A portable implementation of an SMI library which can be embedded into a broad range of management applications is described in Section 5. We relate our work with other approaches to describe management information in Section 6 before we conclude with some remarks about the current status of the SMIng effort and some thoughts about the future of SMIng.

2 Problems of SMIv2

The SMIv1 was defined at a time when the Abstract Syntax Notation One (ASN.1) [8] was the protocol specification language of choice. ASN.1 was selected as the base for the SMI, not realizing that ASN.1 with its syntactic rules is problematic as a data definition language. Furthermore, the decision to base the SMI on ASN.1 made the SMI standard depend on the ASN.1 specification, which is under control of the International Organization for Standardization (ISO). In fact, SMIv1 and SMIv2 are both based on the 1988 version of ASN.1, which is no longer available from the official ISO standards sources. This dependency makes it difficult for MIB developers or implementors to find answers to questions concerning the syntactical rules of the SMI language.

During the development of SMIv2, it was decided that pure ASN.1 is not sufficient to define SMIv2. New constructs were introduced as part of the definition of SMIv2 which are outside of ASN.1. Hence, the SMIv2 is now officially based on an "adapted subset of ASN.1 (1988)." This implies that it is not possible to use generic ASN.1 tools to process MIB definitions. In fact, the authors are not aware of a single MIB parser which is implemented using generic ASN.1 tools.

Existing MIB parsers are usually implemented from scratch. This requires to translate the SMI definition into a Backus-Naur Form (BNF) grammar and a set of lexical rules in order to use them with scanner and parser generators. Some programmers avoid this translation (since it is difficult to obtain and read the relevant ASN.1 specification) by implementing "fuzzy" MIB scanners that do not really check whether a given MIB module conforms to the SMI rules or not. As a consequence, we are faced with many erroneous MIB modules and it is relatively common that end users have to edit MIB definitions in order to feed them into management applications.

Another drawback of the current SMI is the insufficient ability to be parsed efficiently. This makes it unacceptable for most management applications, especially for those with short expected runtime or the need of a huge amount of management information, to parse MIB modules during startup. In consequence, many applications use proprietary intermediate file formats to store management information more efficiently and reduced to the main items, usually the numerical object identifier values together with associated descriptors, type information and probably display hints. Other information from the original MIB modules is usually not accessible from these files. Intermediate formats are generated as the output of MIB compilers like `mosy` or `SMICng`. By using intermediate files, detailed MIB parsing is no longer required on each application startup.

Another problem area is the lack of extensibility of the current SMI. Within the IETF, there is occasionally a need to define new management information that does not fit into macros provided by SMIv2. An example is the definition of protocol identifiers for RMON-2 probes [9]. There is also a desire to augment MIB definitions. One of the well known examples is the assignment of severity levels for notifications[1]. Some management systems actually use quite complex annotations to map SMI notifications into OSI alarm records. The lack of an annotation mechanism in SMIv1/SMIv2 forces people to store annotations either in SMI comments or in `DESCRIPTION` clauses. This leads to MIB maintenance problems in multi-vendor environments since end users are now forced to manually merge modifications made by different vendors.

Finally, there are several issues with programmatic access to SMI definitions. Advanced SNMP toolkits and management platforms require fast access to large amounts of SMI definitions, e.g. to implement basic type checking, to convert enumerations to labels, to convert object identifier values to descriptors, or to apply display hints. User interfaces use SMI information to format tables or to display values together with their units. Since an application needs to deal with thousands of SMI definitions, it is necessary to load MIB modules fast and to balance the memory usage for storing MIB definitions against the time needed to locate a particular piece of information. There are currently no generally accepted APIs that address this issue. Instead, users are faced with several intermediate formats used by the various management tools in use. This makes the task of maintaining MIB modules on large management systems complex and error prone and it makes the implementation of custom management tools more complex than necessary.

3 SMIng Design

The first goal of the SMIng design was to overcome the dependency on external standards and to make the SMIng definition self-consistent. This includes the requirement to define SMIng formally in a BNF grammar to simplify the imple-

[1] Although frequently used in practice, this example is questionable since the severity of a notification generally depends on the current state of a network and is not static.

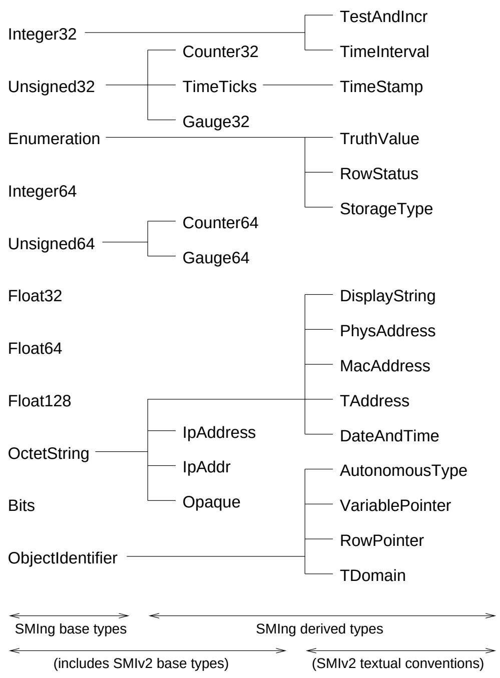

Fig. 1. SMIng base data types and fundamental derived data types

mentation of SMIng parsers. A decision was made to use the Augmented BNF (ABNF) [10] which is pretty powerful and under control of the IETF.

The second requirement for SMIng was to simplify the language where possible. An SMIng file simply contains a sequence of SMIng statements. Every SMIng statement starts with a leading lowercase keyword identifying the statement followed by a number of arguments. Statements are terminated by a semicolon. An argument may be quoted text, an identifier, a value of any base type, a list of identifiers enclosed in parenthesis or a statement block enclosed in curly braces. Since statement blocks are valid arguments, it is possible to nest statement sequences. These few rules build the core SMIng grammar rules and simplify the implementation of parsers. The strict statement separation by semicolon characters enables easy error recovery in parsers. SMIng uses case sensitive identifiers. Uppercase identifiers identify SMIng types and modules while lowercase identifiers identify statements, named numbers or nodes in the registration tree. Case sensitive identifiers simplify the conversion between SMIng and SMIv2 since ASN.1 identifiers are also case sensitive.

SMIng generally disallows forward references except in those cases where forward references are unavoidable (table indexing). Syntactic elements and the names of keywords were chosen to make MIB modules more understandable for the average network operator and programmer. Furthermore, SMIng definitions are compact since related definitions are not distributed over several locations within a MIB module. SMIng also avoids redundant definitions and it mandates a certain order of the statements in a MIB module.

The third goal was to overcome the current restrictions on SMI base data types. There is a long standing need for a more complete set of base data types (especially a complete set of 64 bit types) and SMIng provides them. The base types supported by SMIng as well as core derived data types are shown in Figure 1. Since the SMIng base types are not compatible with existing SNMP versions, a mapping needs to be defined from SMIng base types to the types available in a particular version of the SNMP protocol. The SMIv2 has been very close to the SNMP protocol. In fact, the protocol imports from the SMI definition. However, we believe that there are benefits in decoupling the SMI from SNMP since it is likely that MIB data will be carried by several different protocols in the future.

The fourth goal for SMIng was to support future SMI extensions without breaking deployed SMIng parsers. The solution adopted for SMIng achieves this goal using two simple mechanisms. First, the ABNF grammar contains productions that allow the parser to skip unknown statements. Second, the SMIng has an `extension` statement which can be used to define an SMIng extension. No attempts have been made to define the syntactic or even semantic aspects of an SMIng extension in order to keep SMIng simple by staying away from a meta level which adds considerable complexity and is likely to lead to implementation problems. However, having an identification of extensions is necessary so that parsers which "understand" a particular extension can decide whether a given statement is such an extension or not.

Finally, the fifth goal was to provide a mechanism to annotate SMIng definitions. MIB annotations have been used in several management systems to add information to a particular MIB definition. The SMIng provides an annotation construct which separates annotations from the annotated definition. This construct is actually defined through the SMIng extension mechanism and thus not part of the SMIng core.

4 SMIng Examples

This section presents some fragments of an SMIng MIB module. The examples are taken from the IF-MIB [11] converted into SMIng format.

Figure 2 shows the head of the IF-MIB module definition in SMIng. The first difference from SMIv2 is that module meta information (`organization`, `contact`, `description`, `revision`) is directly associated with the module. The `ifMIB` identifier in the `module` statement and the `oid` statement can be used to provide the information necessary to translate the SMIng module meta information into an invocation of the SMIv2 `MODULE-IDENTITY` macro. Another difference from SMIv2 is the absence of an explicit last update timestamp. SMIng uses the topmost `revision` statement to indicate the timestamp of the last modification. Figure 2 also shows the `import` statements for the IF-MIB module. It is generally not necessary to import SMIng constructs. However, derived SMIng types such as `Counter32` must be imported from the relevant SMIng module.

```
module IF-MIB ifMIB {

  import IRTF-NMRG-SMING-TYPES (Counter32, Gauge32, Counter64, TimeTicks
                               DisplayString, PhysAddress, TruthValue,
                               RowStatus, TimeStamp, AutonomousType,
                               TestAndIncr);
  import IRTF-NMRG-SMING (mib-2);
  import SNMPv2-MIB (snmpTraps);
  import IANAifType-MIB (IANAifType);

  oid           mib-2.31;

  organization  "IETF Interfaces MIB Working Group";
  contact       "   Keith McCloghrie
                    Cisco Systems, Inc.
                    170 West Tasman Drive
                    San Jose, CA  95134-1706
                    US

                    408-526-5260
                    kzm@cisco.com";
  description   "The MIB module to describe generic objects for
                 network interface sub-layers.  This MIB is an updated
                 version of MIB-II's ifTable, and incorporates the
                 extensions defined in RFC 1229.";

  revision {
    date        "1996-11-03 13:55";
    description "[Revision added by libsmi due to a LAST-UPDATED clause.]";
  };
  revision {
    date        "1996-02-28 21:55";
    description "Revisions made by the Interfaces MIB WG.";
  };
  revision {
    date        "1993-11-08 21:55";
    description "Initial revision, published as part of RFC 1573.";
  };

  //...
};
```

Fig. 2. Module definition in SMIng

Figure 3 shows a type definition. The statements in the `typedef` block define the base type including a range restriction, the display format and a description. Note, there is no status statement for this definition. The absence of a `status` statement associates the default status `current` with an SMIng definition.

SMIng allows to derive new types from any existing type, no matter whether the existing type is a base type or a derived type. Type definitions can use a `units` statement to define the units associated with that type. This is different from SMIv2 which only allows `UNITS` clauses in `OBJECT-TYPE` definitions.

SMIng makes a distinction between scalar objects and columnar objects. Figure 4 shows the definition of the scalar `ifNumber` and fragments of the definition of the table `ifTable`. Note that `column` statements are nested into a `row` statement which is itself nested into a `table` statement. This nesting naturally expresses the table registration structure and it avoids redundant definitions as can be found in SMIv2.

```
typedef InterfaceIndex {
  type        Integer32 (1..2147483647);
  format      "d";
  description "A unique value, greater than zero, for each interface
               or interface sub-layer in the managed system.  It is
               recommended that values are assigned contiguously
               starting from 1.  The value for each interface sub-
               layer must remain constant at least from one re-
               initialization of the entity's network management
               system to the next re-initialization.";
};
```

Fig. 3. Type definition in SMIng

```
scalar ifNumber {
  oid         interfaces.1;
  type        Integer32;
  access      readonly;
  description "The number of network interfaces (regardless of their
               current state) present on this system.";
};

table ifTable {
  oid         interfaces.2;
  description "A list of interface entries.  The number of entries
               is given by the value of ifNumber.";

  row ifEntry {
    oid         ifTable.1;
    index       (ifIndex);
    description "An entry containing management information applicable
                 to a particular interface.";

    column ifIndex {
      oid         ifEntry.1;
      type        InterfaceIndex;
      access      readonly;
      description "A unique value, greater than zero, for each
                   interface.  It is recommended that values are
                   assigned contiguously starting from 1.  The
                   value for each interface sub-layer must remain
                   constant at least from one re-initialization of
                   the entity's network management system to the
                   next re-initialization.";
    };
    // ...
  };
};
```

Fig. 4. Scalar and table definition in SMIng

The table definition in Figure 4 is incomplete, which is indicated by a comment. Comments in SMIng start with the character sequence // and end at the end of the line.

Notifications are defined using the SMIng `notification` statement, which is very similar to the SMIv2 `NOTIFICATION-TYPE` macro. Figure 5 shows the definition of the `linkDown` notification.

```
notification linkDown {
  oid         snmpTraps.3;
  objects     (ifIndex, ifAdminStatus, ifOperStatus);
  description "A linkDown trap signifies that the SNMPv2 entity,
               acting in an agent role, has detected that the
               ifOperStatus object for one of its communication links
               is about to enter the down state from some other state
               (but not from the notPresent state).  This other state
               is indicated by the included value of ifOperStatus.";
};
```

Fig. 5. Notification definition in SMIng

Object or notification definitions can be grouped using the `group` statement as shown in Figure 6. SMIng does not distinguish between notification groups and object groups. It is allowed to group notifications and objects together within a single `group` statement in SMIng. However, mixing notification and object definitions in an SMIng `group` makes it impossible to translate the SMIng module to SMIv2 automatically.

```
group ifCounterDiscontinuityGroup {
  oid         ifGroups.13;
  members     (ifCounterDiscontinuityTime);
  description "A collection of objects providing information
               specific to interface counter discontinuities.";
};
```

Fig. 6. Group definition in SMIng

Figure 7 shows parts of a compliance definition in SMIng. The `compliance` statement requires that all identifiers appearing in a compliance definition are locally defined or properly imported. This is different from SMIv2 which introduced special rules for resolving naming scopes in the `MODULE-COMPLIANCE` macro.

Finally, Figure 8 shows an IF-MIB annotation which makes the discontinuity indicator for counter objects machine readable. The `ifTableDiscontinuity` annotation uses an SMIng extension called `discontinuity` which is imported from the `TUBS-IBR-EXTENSIONS` MIB module. The `target` and `description` statements are part of the `annotation` extension and therefore not explicitly imported.

```
compliance ifCompliance {
  oid         ifCompliances.1;
  status      deprecated;
  description "The previous compliance statement for SNMPv2 entities
               which have network interfaces.";

  mandatory   (ifGeneralGroup, ifStackGroup);

  // ...

  optional ifPacketGroup {
    description "This group is mandatory for all network interfaces
                 which are packet-oriented.";
  };

  // ...

  refine ifAdminStatus {
    type        Enumeration ( up(1), down(2) );
    access      readonly;
    description "Write access is not required, nor is support for the
                 value testing(3).";
  };
};
```

Fig. 7. Compliance definition in SMIng

```
module TUBS-IBR-EXTENSIONS {

  //...

  extension discontinuity {
    description "Indicates the discontinuity indicating object for a
                given set of targets. The intended usage of this
                extension is within an annotation statement which
                identifies the targets. The ABNF definition for this
                extension is:

                  discontinuityStatement = discontinuityKeyword sep
                                           qlcIdentifier optsep ';'
                  discontinuityKeyword = 'discontinuity'
                ";
  };
};

module TUBS-IBR-IF-MIB-ANNOTATION {

  imports IRTF-NMRG-SMING-EXTENSIONS (annotation);
  imports TUBS-IBR-EXTENSIONS (discontinuity);

  //...

  annotation ifTableDiscontinuity {
    targets       (ifInOctets, ifInUcastPkts, ifInNUcastPkts,
                   ifInDiscards, ifInErrors, ifInUnknownProtos,
                   ifOutOctets, ifOutUcastPkts, ifOutNUcastPkts,
                   ifOutDiscards, ifOutErrors);
    discontinuity ifCounterDiscontinuityTime;
    description   "The discontinuity indicator for the ifTable.";
  };
};
```

Fig. 8. Annotation and extension in SMIng

5 SMI Library libsmi

There is no common way for management applications to access MIB definitions, neither in form of a standardized API to access SMIv1/SMIv2 files nor in form of a standardized intermediate file format. This section describes the design and implementation of a portable and reusable C library called `libsmi` providing access to all definitions contained in SMIv1, SMIv2 or SMIng MIB modules, except agent capability statements. In particular, the `libsmi` API provides access to:

- module definitions (`struct SmiModule`) including all revision information (`struct SmiRevision`),
- type definitions (`struct SmiType`) including named numbers in case of enumerated types like `Bits` and `Enumeration` (`struct SmiNamedNumber`) and size and range subtyping restrictions (`struct SmiRange`),
- all kinds of nodes in the registration tree defined by `module`, `node`, `scalar`, `table`, `row`, `column`, `notification`, `group`, `compliance` statements, or their SMIv1/v2 equivalents (`struct SmiNode`),
- all information associated with notifications, tables, groups, and compliance statements.

Figure 9 provides an overview over the data structures exported by the `libsmi` API.

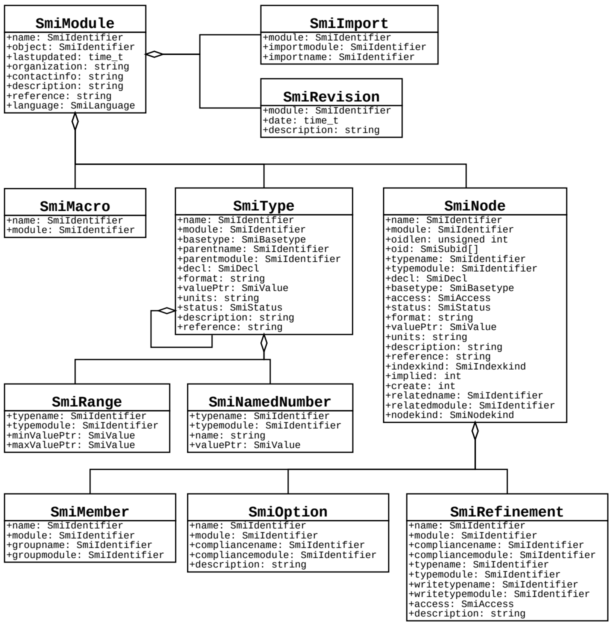

Fig. 9. Data structures exported by the SMI library

The access methods provided by the `libsmi` API include direct access functions by key values (e.g. names or object identifiers) as well as iterators like get-first/get-next functions to retrieve all available structures of a certain kind within a module. The library also provides functions to initialize and configure the behavior of `libsmi` and a function to add a module to the set of modules known by the application. Figure 10 shows a simple example program which prints an indented list of all nodes defined in a given module.

```
#include <stdio.h>
#include <smi.h>

int main(int argc, char *argv[])
{
    SmiNode *smiNode;

    if (argc != 2) {
        fprintf(stderr, "Usage: smitree module\n");
        exit(1);
    }

    smiInit();
    smiLoadModule(argv[1]);

    for(smiNode = smiGetFirstNode(argv[1], SMI_NODEKIND_ANY); smiNode;
        smiNode = smiGetNextNode(smiNode, SMI_NODEKIND_ANY)) {

        printf("%*s%s::%s\n", smiNode->oidlen * 2, " ", smiNode->module, smiNode->name);

    };

    return 0;
}
```

Fig. 10. Example usage of the SMI library

The internal library design consists of two layers as shown in Figure 11. The upper layer implements the API and the internal data structures that are used to store management information internally in memory. This layer retrieves the information through one or more backends residing on the lower layer of `libsmi`. The `libsmi` currently supports two parsers that are implemented using flex and bison, the improved GNU implementations of lex and yacc. One parser is derived from the SMIv1 and SMIv2 specifications. The other one strictly follows the SMIng ABNF grammar. Both parser backends resolve imports automatically by recursively parsing imported MIB modules.

The `libsmi` library includes two applications, a MIB checker and a MIB dump program. The MIB checker (`smilint`) uses the verbosity level of the underlying SMIv1/SMIv2 and SMIng parsers to generate errors and warnings about illegal or inadequate definitions in a MIB module. The MIB dump program (`smidump`) iterates through the definitions of a given module and writes the retrieved structures in one of several output formats. The supported output formats include SMIv1, SMIv2, SMIng, the `mosy` file format, a tree structure of recursively imported MIB modules, a tree structure of derived type definitions,

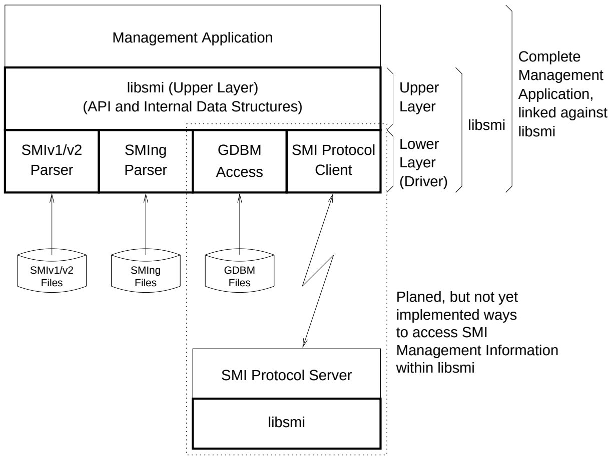

Fig. 11. Internal structure of the SMI library

a tree structure of node definitions, and CORBA IDL according to the JIDM specification translation rules [18].

The concept of backends and dump formats enables the conversion of MIB modules, e.g. dumping an SMIng module that is read by the SMIv2 parser backend and vice versa.

6 Related Work

Management Information Bases within the ISO management framework are defined using a set of templates, usually known as the Guidelines for the Definition of Managed Objects (GDMO) [12]. The GDMO language is very close to ASN.1 although it does not fully conform to the ASN.1 standard [13]. GDMO has similar problems like SMIv1/SMIv2 due to a close but not absolute relationship to ASN.1. The nine GDMO templates can be used to create very modular MIB definitions which enhances re-usability. However, the price for the modularity is that a reader of GDMO MIB definitions must follow several references in order to understand a managed object class definition. Instead, all existing SMI versions try to keep the relevant pieces of an object definition close together in order to enhance readability.

The Common Information Model (CIM) [14] is a relatively new management information model defined by the Desktop Management Task Force (DMTF). CIM schemas are used to define management information following an object-oriented approach. Schemas are stored in files using the Management Object Format (MOF). This format was inspired by the CORBA IDL language [15] but has substantial differences. The CIM model and the corresponding file format have an extension mechanism based on qualifiers. CIM distinguishes between meta, standard, optional and user-defined qualifiers. This allows to extend the

CIM and the MOF file format over time. However, user-defined qualifiers must be embedded in CIM schema files and are generally not recommended. There are currently no clear rules how to allocate qualifiers in an ordered way to avoid ambiguities. Recent work within the Web-Based Enterprise Management initiative aims to replace the MOF format with an XML [16] Document Type Definition (DTD) [17]. Using XML makes in our view a CIM schema definition less readable due to the space needed for the XML mark-up. However, the XML format allows developers to use off-the-shelf XML converters to produce hypertext versions for use with Web browsers. On the other hand, using XML means that the CIM file format depends on a technology not under control of the DMTF. This may result in derivations from the XML standard in the future, similar to the current situation with the derivation of the SMIv2 from ASN.1.

The CORBA standards of the Object Management Group (OMG) [15] have received quite some attention in the network management area. However, the Interface Definition Language (IDL) has so far not been used to standardize management information bases. This probably just shows the lack of an organization willing to standardize management information directly in IDL. It may also indicate that CORBA IDL is already too close to implementations and does not support the machine readable definition of other types of modeling information. The Joint Inter-Domain Management project (JIDM) has developed specifications how GDMO and SMI MIB modules can be translated into CORBA IDL definitions [18]. However, some machine readable information is usually lost during the translation since it shows up only in IDL comments and it is not possible to access this information at runtime easily.

7 Conclusions and Future Work

This paper proposes a new Structure of Management Information called SMIng for use with the Internet management framework. We presented the motivation and the requirements for SMIng. The main features of SMIng are the independence from ASN.1, a minimal and complete set of core data types, provisions for SMI extensions and annotations, compactness and simplicity. The specification of the SMIng has been submitted to the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF) for further development.

This paper also describes a reusable and portable C library which provides access to MIB definitions. It supports parser backends for SMIv1, SMIv2 and SMIng and hides the differences between the SMI versions as much as possible. Two applications using the `libsmi` library have been written to validate MIB definitions (`smilint`) and to convert MIB definitions between SMIv2, SMIng and other formats (`smidump`).

The implementation of the SMI library is freely available[2]. We plan to integrate the library into other open source management software packages (e.g. the UCD SNMP distribution) and to provide bindings to languages such as Tcl,

[2] `http://www.ibr.cs.tu-bs.de/projects/libsmi/`

Python or Java. Future work may also result in additional backends for the `libsmi`. Currently two ones are planned: A client `libsmi` might access management information through a protocol from a management information repository server that holds a large set of MIB modules, while clients do not have to load any modules, and consequently reach very short startup times. Another backend is planned to use a classical database file format like `gdbm` to store the management information. This will lead to short application startup times and low memory usage at the expense of slower `libsmi` function calls. Both plans require caching techniques in the library's upper layer and have to handle some problems with iterating access functions.

References

[1] D. Perkins. SNMP Versions. *Simple Times*, 5(1), December 1997.

[2] M. Rose and K. McCloghrie. Structure and Identification of Management Information for TCP/IP-based Internets. RFC 1155, May 1990.

[3] M. Rose and K. McCloghrie. Concise MIB Definitions. RFC 1212, March 1991.

[4] M. Rose. A Convention for Defining Traps for use with the SNMP. RFC 1215, March 1991.

[5] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Structure of Management Information Version 2 (SMIv2). RFC 2578, April 1999.

[6] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Textual Conventions for SMIv2. RFC 2579, April 1999.

[7] K. McCloghrie, D. Perkins, J. Schönwälder, J. Case, M. Rose, and S. Waldbusser. Conformance Statements for SMIv2. RFC 2580, April 1999.

[8] D. Steedman. *Abstract Syntax Notation One (ASN.1): The Tutorial and Reference*. Technology Appraisals, 1990.

[9] A. Bierman and R. Iddon. Remote Network Monitoring MIB Protocol Identifiers. RFC 2074, January 1997.

[10] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 2234, November 1997.

[11] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB using SMIv2. RFC 2233, November 1997.

[12] ISO. Information technology — Open Systems Interconnection — Structure of Management Information — Part 4: Guidelines for the Definition of Managed Objects. International Standard ISO 10165-4, ISO, 1992.

[13] M. Sloman. *Network and Distributed Systems Management*. Addison-Wesley, 1994.

[14] DMTF. Common Information Model (CIM) Specification Version 2.0. CIM Standards, Desktop Management Task Force, March 1998.

[15] J. Siegel. *CORBA Fundamentals and Programming*. Wiley & Sons, 1996.

[16] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, Textuality and Netscape, Microsoft, University of Illinois, February 1998.

[17] DMTF. Specification for the representation of CIM in XML Version 1.0.1. CIM Standards, Desktop Management Task Force, January 1999.

[18] The Open Group. Inter-domain Management: Specification Translation. Preliminary specification p509, The Open Group, March 1997.