Submission for the Feature Topic on
”XML-based Management of Networks and Services”
in the
IEEE Communications Magazine

(Publication: July 2004)
(Paper No. 352)

Integrating SNMP Agents with XML-based Management Systems

Torsten Klie and Frank Strauß,
Technical University of Braunschweig

2004-04-15

Abstract

XML-based network management systems become more and more popular these days. However, it has to be taken into account that there is a broad existing management infrastructure which is based on non-XML protocols. Today, the most dominant technology in Internet management is based on the SNMP framework.

This article presents an approach for a seamless integration of management information supplied by SNMP agents into XML-based systems. This is done by an automated transformation of SMI MIB definitions into XML Schema definitions, which implies a corresponding transformation of SNMP management information into XML documents. Furthermore, we present an SNMP-to-XML gateway that allows to retrieve such XML management information at runtime.

Introduction

Internet management as it is done today is heavily based on the Simple Network Management Protocol (SNMP) and Management Information Bases (MIBs) conforming to the Structure of Management Information (SMI). The reason for the success of the SNMP Framework is at the same time its primary disadvantage: SNMP is simplistic with respect to its data structures and protocol operations. This allowed early implementations even on devices with limited resources already in the late 1980s. Now, 14 years later, we are faced with the third generation of the protocol specification (SNMPv3), the second version of the MIB language (SMIv2), and no less than 185 IETF Standard MIB modules defined by various working groups, let alone hundreds of vendor specific MIBs[1].

Over time, a number of drawbacks turned up and have been solved just to some extent: Performance issues with the retrieval of large chunks of data have partly been addressed by the GetBulk protocol operation which was introduced with the now historic SNMPv2 and is present in the Standard SNMPv3. Major security issues have been discussed intensively and finally been solved with the User-based Security Model (USM) and the View-based Access Control Model (VACM) which are also part of the SNMPv3 framework. The result of this solid standardization work is a fundamental technology upon which many hardware vendors, software manufacturers, network operators, and administrators have built management systems to manage a wide range of computer networks. Obviously, these implementations also represent huge investments.

However, some issues are still open: There are no standard formats for the storage and processing of management data. Transferring large amounts of data is still relatively inefficient compared to stream based bulk transfers (e.g. TCP) [2]. Configuration management is yet an open issue and currently heavily discussed in the IETF NETCONF Working Group [3]. Efficient software development cycles are hard to achieve based on a low-level technology such as SNMP.

To some extent, XML and its companion technologies offer opportunities to solve these problems [4]. XML supports by its nature a well-defined encoding and syntax. Although XML is still a newcomer compared to SNMP, there is a huge community of XML professionals and also a large number of software tools (parsers, editors, validators, database systems, transformation tools, etc.) that can make the development of management applications more efficient.

In the long run, a fully XML-based architecture that affects all entities in a managed network promises the best benefits. In the short term, the most pressing needs arise on the managers' side where operators could take advantages of an XML-based processing of their management information, e.g., event correlation and rendering of aggregated information based on the family of Extensible Stylesheet Language specifications (XSL), management of device configurations based on XML and XSL Transformations (XSLT), ensuring configuration integrity based on a schema language like XML Schema, etc.

This situation increases the demand for an inte-

[1] The freely available online publication of the *Simple Times* [1] regularly summarizes all SNMP related IETF Standards.

gration of managed nodes based on SNMP technology on one side and management systems based on XML technology on the other side. The remainder of this article is targeted to an approach that bridges this gap: First, we explain how some of the core XML technologies are related and how they could be applied to typical network management tasks. Then, we present and discuss, how two core XML technologies, namely general XML documents and XML Schema definitions, can be mapped from SNMP management data and their SMIv2 MIB definitions, respectively. We will spot on the goals that were driven by the XML perspective rather than the SNMP point of view. The major design decisions for this mapping will be presented. This is followed by the presentation of an SNMP-to-XML gateway architecture and the implementations we have made. The gateway architecture allows to access management information through HTTP requests for XML documents that comply to the presented mapping. Finally, we conclude with some experience statements and a rough outlook on future directions in SNMP/XML integration.

XML Technologies

Complete and precise specifications of protocols are a key prerequisite to any deterministic and robust distributed system. This includes standard data structures, encoding rules, and protocol operations. Such standards exist for many different areas: Databases are queried through the query language SQL and clients use APIs compliant to ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity), remote procedure calls and remote object access can be realized through various RPC standards or CORBA, etc.

Whereas such technologies evolved over time and work quite well today, they have been developed almost independently from each other. They do not build upon each other and they hardly use common basic concepts, although many of them have to address similar questions like byte ordering, data framing and data encryption.

In contrast, XML is a core building block upon which other related technologies are being developed. This 'toolkit concept' allows for more efficient development processes of applications. For example, existing XML parsers can be used to develop XSLT processors and XML Schema based validators, since XSL and XML Schema themselves are XML compliant. XML Schemas can be defined and used by validating XML parsers to ensure XML data integrity. Specific XML compilers can be built as XSL stylesheets using any existing XSLT processor. Figure 1 illustrates some of the rather prominent XML technologies and according tools as well as their relationships. Although this Figure is not complete, it presents the technologies that are currently most intensively discussed to be applied for Internet management purposes. Specifications and tutorials for all of these technologies as well as pointers to detailed literature and implementations are available from the W3C web site [5]. The rest of this Section describes how these technologies can be applied to Internet management:

- **Use XML to represent management data.** SNMP specifies the encoding of management data in the protocol, but it does not define how data should be represented when it has to be stored for further processing. XML can easily be applied for both. Furthermore, in contrast to SNMP, even large amounts of data can be represented and transmitted in a single XML-based message. This helps to realize atomic protocol transactions, since the conversation to one agent no longer has to span multiple PDUs. Last but not least, XML-based representations are widely accepted as a good compromise between human readability (to ease debugging and to allow operators to take corrective actions) and computer readability (to ensure parsability and integrity).
- **Use HTTP to ship management data.** When SNMP was designed, CPU, memory and bandwidth consumption were the most influencing factors. Today, reliability, throughput, ease of use and overall development costs are considered much more important. Even low-cost printers and WLAN access points are equipped

with HTTP agents to support easy manual configuration. This protocol infrastructure can easily be used to allow a more robust transfer of XML documents. URLs represent a common notation to address requested data.

- **Use DOM and SAX APIs to access management data from applications.** Once management data is available in XML format, it can easily be processed by any management applications. Common APIs, such as the Document Object Model (DOM) and the Simple API for XML (SAX) make the development of such applications much more efficient.
- **Use XSL to process management data.** Extensible Stylesheet Language Transformations (XSLT) allow to define rules for transforming input XML documents into other XML documents, HTML web pages, plain text files, or even PDF documents. The accompanying XPath language is used to specify filtering expressions within such transformations. XSLT and XPath provide a very powerful tool to process XML-based management information, e.g., to generate network maps, statistical overviews, filtered alarm logs based on correlated events, etc.
- **Use XML Schema to define the structure of management data.** Most computer languages and data models are specified through formal grammars. While SNMP uses SMI, which is a domain specific definition language (an 'adapted subset' of ASN.1) to define management data models, XML Schema is a pure XML standard. It provides an elegant means for defining the structure, content and semantics of XML documents in specifications which themselves are XML documents. Using XML Schema to define XML management data models helps to ensure the integrity of management data like, e.g., configuration documents.

Converting SMI MIBs to XML Schema Definitions

The structure of MIB modules is formally described with the SMIv2 language. These MIB modules define the structure and semantics of the instance data which is supplied by SNMP agents. In other words: SMIv2 is the modeling language that is used to define data models (MIBs) which in turn define the structure of management data. In the XML world, the XML Schema language is the formal modeling language. Schema definitions are documents written in that language and can define the structure of XML documents which may represent management data (see Figure 2).

When designing the mapping, we had two aims in mind. First of all we wanted the XML documents to be as convenient for reading and processing as possible. We tried to look at management data from an XML point of view. Achieving a clean XML structure makes the conversion algorithm more complicated compared to most other approaches that are more destined to a 1:1 SMI mapping and look more like an 'XML-ified SMI'. The most notable of these other approaches known to the authors are those from Mazumdar [6], from Hong et al [7, 8], and from Martin-Flatin [4]. Some major aspects of these approaches are compared in Table 1. The second aim was to reduce the loss of information during the conversion process. The generated XML Schema definitions should contain almost all information of the underlying SMI MIB module. In the following, the most important characteristics of our mapping will be described and compared to other translation approaches.

XML Structure of Management Data

The first step that has to be taken when designing the structure of an XML document is to define its scope. Several scope restrictions are possible. For example, each XML document could represent instance data of one specific MIB module at a determined agent at a given point in time. The XML documents of the approach from Hong et al are designed that way. Another possibility is to restrict the range to special MIB objects such as tables and scalar groups. That increases readability because only a fragment of the MIB will be presented such that the structure can be kept very simple. The XML documents of Mazumdar's ap-

proach are designed that way. We propose a root element that is independent from specific MIBs, agents or point in time in order to ensure a more flexible data range of the documents. The agent, the SNMPv1 community, and the point in time are encapsulated in `<context>` elements at the second level. This way, it is possible to store data from multiple agents or snapshots made at different points in time within a single document, because the root element can store an arbitrary number of these second level elements. However, the XML document structure becomes slightly more complicated than in the other two approaches, because an additional element level is used.

There are two possibilities of how to deal with the OID tree of a MIB. The first one is a straightforward mapping of all nodes in the MIB to nested elements in the XML instance document. SNMP experts should be quite familiar with such a mapping because the XML document tree represents exactly the OID tree they know. The three related approaches use this kind of direct mapping. In our opinion, the straightforward mapping leads to unnecessarily complicated XML instance documents. Therefore, we designed a 'flattened' element hierarchy and tried to avoid deep element nesting. Groups of scalar objects and table rows are mapped to elements on the third level (the level below `<context>`) regardless of their position in the OID tree. This flattened hierarchy is much more readable than a deep nested element structure. The OID information of a node is put into an `<xsd:appinfo>` clause in order retain it for applications that rely on it.

Unique naming in SMI is based on modulename-descriptor pairs. In XML, unique naming can be achieved using XML namespaces. Therefore, SMI modules should be mapped to XML namespaces. Mazumdar's approach uses multiple XML Schema definition files which define a common namespace, one schema file for every data container. Hong's approach does not use namespaces, but every MIB module is mapped to a single XML Schema definition file. In our approach, we map each MIB module to a single XML Schema definition file which defines its own namespace.

`INDEX` clauses in a MIB module are used to specify the objects that uniquely identify table rows. Columnar objects that are used in an `INDEX` clause are not mapped to sub-elements, but to attributes of the table row element. An additional `<xsd:key>` construct is created to denote that this combination of columnar objects serves as a unique key that identifies an XML table row. In Mazumdar's approach, there is just one index attribute which contains a concatenated string of the index object values. This could cause ambiguity problems. Hong's and our approach put the value of the index variables into attributes. To eliminate redundancy, index objects are only listed as attributes and not as normal elements in our approach.

In SMI, there are confusing constructs to represent nested data structures such as augmentation tables and tables that share a common prefix list of index objects with another table. This is a downside of SNMP's simplicity. In XML, there is a more intuitive and elegant way of expressing similar relationships: element containment. Therefore, columnar objects of augmentation table rows are in our approach added as child elements of the common parent table row. However, this is only possible if the augmentation table is part of the same MIB module, because MIB modules cannot contain references to (potentially future) augmenting MIB modules. The same applies to sub-tables, which are mapped to child elements of their parent table element.

Mapping Data Types

Type and `TEXTUAL-CONVENTION` definitions in MIB modules are mapped to XML Schema types which are derived from base types with appropriate `<xsd:restriction>` clauses. This mapping is automated except for the base types, which are defined in a single hand-crafted XML Schema definition. The differences between the mentioned approaches are only marginal with respect to the base type mapping. For details on our base type mapping see Table 2.

The value space can be restricted with `<xsd:minInclusive>` and

<xsd:maxInclusive> for numbers. Similarly, with <xsd:minLength> and <xsd:maxLength> it is possible to restrict the length of strings. This kind of mapping is supported by all mentioned approaches. However, some limitations such as multiple ranges (e.g. -1 | 60..600, as in DISMAN-EXPRESSION-MIB::expResourceDeltaMinimum) cannot be expressed in XML Schema with simple facets. A union type with a subtype for each subrange can solve this problem. However, union types are 'a dead end' with respect to further inheritance.

With DISPLAY HINT clauses, MIB authors specify, how they think the data should be displayed. Therefore, we interpret display hints also for rendering instance data in XML documents. Thus, if the type defined in the MIB has a display hint, that hint is used to further restrict the value space of the mapped XML Schema type. Integer types, for example, may get an additional <xsd:fractionDigits> facet, which specifies the number of decimal places. For string types, even complex display hints are translated automatically to <xsd:pattern> constructs in the XML Schema definition with regular expressions that formally restrict the value space. In this case, the <xsd:minLength> and <xsd:maxLength> facets are not used, because the generated regular expression usually implies length restrictions. Note that these length restrictions may be different from the one defined in the MIB because the MIB's restriction corresponds to the length of the unformatted octet string whereas the restriction implied in the <xsd:pattern> facet corresponds to the formatted string.

As mentioned above, one of our aims was to represent management data in a human readable fashion. Therefore, the text of leaf elements and index attributes should be represented in an intuitive way. Enumeration types and bit sets are written as the according names. Strings are written in ASCII form, if the underlying MIB type has a display hint that 'suggests' an according representation. If the display hint recommends another representation, the string is rendered accordingly. Otherwise, if there is no display hint, the type <xsd:hexBinary> is used. This type represents binary data in hexadecimal form. Thus, in this case the string will be written in hexadecimal digits.

To reduce the loss of information, SMI MIB module information that is not necessarily required in the XML Schema definition is put into <xsd:appinfo> clauses so that it remains available for other applications. This is useful especially for XSLT-based MIB compilers or MIB browsers.

Figure 3 shows an example of an XML instance document conforming to our proposal. The shown data is defined in the IF-MIB module [9] and has been retrieved from the agent on a router named ciscobs.rz.tu-bs.de.

An SNMP-to-XML Gateway

In this Section, we will present the architecture of a gateway that makes the presented mapping viable in a seamless way.

The manager interface of the gateway is based on HTTP, where request parameters are contained in the HTTP query string. Only the data of SET requests is transfered in the body of HTTP POST requests. SOAP would be an alternative, but for the gateway it would add further protocol overhead on top of (usually) HTTP without additional functionality. Furthermore, in the presented approach, documents retrieved from the gateway can directly be incorporated in XSL transformations without a SOAP client engine.

Figure 4 shows the architecture of the gateway and its three major operations:

(a) An XML-based manager that attempts to read management data, sends an HTTP GET request to the gateway. The URL comprises the protocol (http or https), the gateway host and the path to the gateway service on the host, and finally an HTTP query string that specifies the operation (get) and an expression to specify the requested management data. Note that this expression is an XPath compliant node set expression: it

starts with the common root node and a context sub-node with a set of context attributes to specify agents, access security credentials, and caching properties. This is typically followed by a node name and according attributes to specify a filter for the requested portion of data supplied by the agent. To illustrate this, imagine the following request to retrieve a list of descriptions of those interfaces on the device `talisker.ibr.cs.tu-bs.de` that are currently not in the operational state to which they have been administratively configured:

```
lynx -dump 'http://www.ibr.cs.tu-bs.de\
 /snmp-xml-gw?get=/snmp-data/context\
 [@hostname="talisker.ibr.cs.tu-bs.de"]\
 /ifEntry[ifOperStatus!=ifAdminStatus]\
 /ifDescr' > interfaces.xml
```

When this request is received by the gateway's HTTP engine, the location path of the XPath expression is interpreted and the translator's core unit constructs a sequence of SNMP GET operations to fulfill the request. To achieve this, the gateway refers to its repository of supported XML Schema definitions which supply the required information, e.g., the mapping of node names to OIDs, element attributes to table indexing information, labels of enumeration values to numeric values, etc.

The SNMP responses received from the agent are converted to XML element information items, again with the help of the XML Schema repository. These fragments of XML information are added to the translator's DOM tree. Once the retrieved data is complete (or the retrieval timed out), the DOM tree is filtered based on the predicate part of the request's XPath expression. The requested portion is passed to the HTTP engine to make up an XML document which is then passed back as the response to the initial HTTP GET request.

In some situations, XML-based management applications access managed objects or groups of managed objects multiple times to achieve a single higher-level operation. In these cases it is usually not required that the gateway retrieves the same SNMP data multiple times from the agent. To take advantage of this circumstance and to improve performance, the gateway could support the caching of information retrieved from the agent and an HTTP client could specify the maximum tolerated age of requested data, e.g., through an XPath expression on the `@time` attribute of the `<context>` element.

(b) Similar to GET requests, the XML-based manager can issue HTTP POST requests in order to control or configure agents through XML documents. In this case, the URL's query string contains just the mode of operation (`set`) in the HTTP query string. The passed XML document is parsed into an internal DOM representation. Then the translator inspects the tree to identify those nodes for which the underlying MIB object definitions allow SNMP write access. Again, the XML Schema repository supplies all the information to do this. Based on these nodes, the translator constructs SNMP SET requests and sends them to the agent. SNMP responses are used to approve the modifications in the DOM tree or to add error information to the DOM tree, if the SNMP SET operations failed. Finally, the resulting DOM tree is used to return an XML document as the response to the initial HTTP POST request.

Note that the XML document submitted through the HTTP POST request can be based on a document previously retrieved through an according HTTP GET request. This allows to easily realize management applications that support simple configuration modifications in pull-edit-push cycles.

Let's assume that in the above GET example, the operator (or an automated management algorithm) decided to administratively

take down some of the interfaces. This can be done by adding the following element to those interfaces in the retrieved XML document:

```
<IF-MIB:ifAdminStatus>down</IF-MIB:ifAdminStatus>
```

Then the document gets resubmitted to the gateway (this Java program just cares about the correct HTTP content encoding):

```
java GatewayClient http://www.ibr.cs.tu-\
 bs.de/snmp-xml-gw set interfaces.xml
```

(c) Finally, the gateway can also work as an SNMP trap sink. When the translator receives a notification it looks up the concerned OIDs and values from the XML Schema repository and constructs a small element information item per notification. These information items are processed in two ways. They can be sent as short XML documents in HTTP POST requests to XML-based notification receivers. This would require the gateway to act as an HTTP client. Furthermore, all notification information items are added to a local XML document that represents a general notification log. This allows XML-based managers to elegantly access the notification log at any subsequent point in time in the same way as regular management information is accessed through HTTP GET requests, i.e., XPath based filtering of notifications is supported and the gateway is not required to act as an HTTP client in this case.

Implementations

The approaches presented in this article have been implemented as follows:

The mapping of SMI MIB modules to XML Schema definitions has been realized as a new output driver added to the `smidump` MIB compiler toolkit. It is available with the regular open source distribution of the `libsmi` [10] that runs on almost any conventional operating system.

The presented gateway architecture has been implemented — except for the support of notifications — as a Java Servlet. It has been developed and used successfully on the Linux/Apache/Tomcat platform. The code is also available through the `libsmi` web site.

Limitations and Future Work

Future work that has to be done towards a fully functional XML-based Internet management can roughly be divided into short-term transitional work and long-term work towards purer XML solutions. The long-term solution for overall XML-based management systems comprehends functionality that cannot be realized efficiently if parts of the manager-agent communication remain based on today's SNMP, which is technology that is known for some limitations, especially with respect to performance. The field of configuration management is currently the best example that makes this clear.

With this in mind, the previous Sections of this article present an approach and a prototype implementation for a transitional phase, of which some aspects can still be improved in the near future and others cannot be solved:

- The processing of notifications has to be implemented.
- Current write operations support only the modification of existing objects, but not the creation and deletion of instances.
- Some MIB objects and types have very special functional semantics, for which a translation to data-oriented XML information sets is not practicable. Examples are objects of types `RowStatus` or `TestAndIncr` or objects that change their values upon read access, like `DISMAN-SCRIPT-MIB::smLaunchRunIndexNext`. This is especially problematic in combination with the gateway's caching support.
- Future revisions of the gateway have to support SNMPv3 security.
- Scalability issues have to be addressed. This affects parallel and conflicting HTTP requests,

huge numbers of managed nodes and objects, and caching strategies.

Summary

This article gave an overview of core XML technologies and how they fit in Internet management systems. It has been explained that the widely deployed infrastructure of SNMP-based managed nodes requires an integration effort to allow a seamless transition. We have presented an automated mapping of SMI MIB data models to XML Schema definitions that implies the mapping of SNMP management data to XML documents. An essential difference from other mapping approaches is that the presented mapping is driven by the goal to tap the full potential of XML and XML Schema so that the translation output fits well in future pure XML-based systems without any inherited burden from SNMP, i.e., the attempt is driven by the target, not the source.

To prove the mapping concept, we have presented an architecture and a prototype implementation for an SNMP-to-XML gateway that transparently translates HTTP requests for XML documents to SNMP operations.

Some items for future work have been discussed, but it has to be acknowledged that this effort is just a transitional step towards pure XML-based Internet management systems. However, it allows the integration of legacy devices that will never support XML based management, and devices with very limited resources into such systems.

Acknowledgements

An early presentation of this work with little information on the gateway was published in [11]. The presented gateway implementation has been done by Jens Müller at the Technical University of Braunschweig. The authors would like to thank the attendees of the 12th meeting of the Network Management Research Group (NMRG) in Colorado Springs, March 2003, as well as Subrata Mazumdar, Frank Fock and Jean-Philippe Martin-Flatin for their helpful comments.

References

[1] J. Schönwälder and A. Pras. The Simple Times. An openly-available online publication on SNMP, http://www.simple-times.org/.

[2] R. Sprenkels and J. P. Martin-Flatin. Bulk Transfer of MIB Data. *Simple Times*, 7(1), March 1999.

[3] S. Leinen and A. Bierman. The IETF Network Configuration Working Group (NETCONF). http://www.ietf.org/html.charters/netconf-charter.html.

[4] J.-P. Martin-Flatin. *Web-Based Management of IP Networks and Systems*. Wiley, 2002.

[5] W3C. W3C – The World Wide Web Consortium. http://www.w3c.org/.

[6] S. Mazumdar. XML-Based Management Interface for SNMP Enabled Devices. WWW Page, 2001. http://www.research.avayalabs.com/user/mazum/Projects/XML/.

[7] Y. Oh, H. Ju, M. Choi, and J. W. Hong. Interaction Translation Methods for XML/SNMP Gateway. In M. Feridun, P. Kropf, and G. Babin, editors, *Proc. 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, pages 54–65. Springer, October 2002.

[8] J.-H. Yoon, H.-T. Ju, and J.W. Hong. Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management. *International Journal of Network Management*, 13:259–276, 2003.

[9] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB. RFC 2863, Cisco Systems, Argon Networks, June 2000.

[10] F. Strauß. Libsmi - A Library to Access SMI MIB Information. http://www.ibr.cs.tu-bs.de/projects/libsmi/.

[11] F. Strauß and T. Klie. Towards XML oriented Internet Management. In G. Goldszmidt and J. Schönwälder, editors, *Proc. 8th IFIP/IEEE International Symposium on Integrated Network Management*, pages 505–518, Colorado Springs, March 2003. Kluwer.

Biographies

TORSTEN KLIE (tklie@ibr.cs.tu-bs.de) is a research staff member at Learning Lab Lower Saxony, Hannover, Germany and a Ph.D. student at the Institute of Operating Systems and Computer Networks at the Technical University of Braunschweig, Germany. He received his diploma (M.Sc.) in business computer science from the Technical University of Braunschweig in 2003. His research interests include Internet management, distributed systems, XML data modeling, and E-Learning.

FRANK STRAUSS (strauss@ibr.cs.tu-bs.de) is a research staff member and Ph.D. student at the Institute of Operating Systems and Computer Networks at the Technical University of Braunschweig, Germany. He received his diploma (M.Sc.) in computer science from the Technical University of Braunschweig in 1999. His research interests include Internet management, distributed systems, network security, and operating systems.

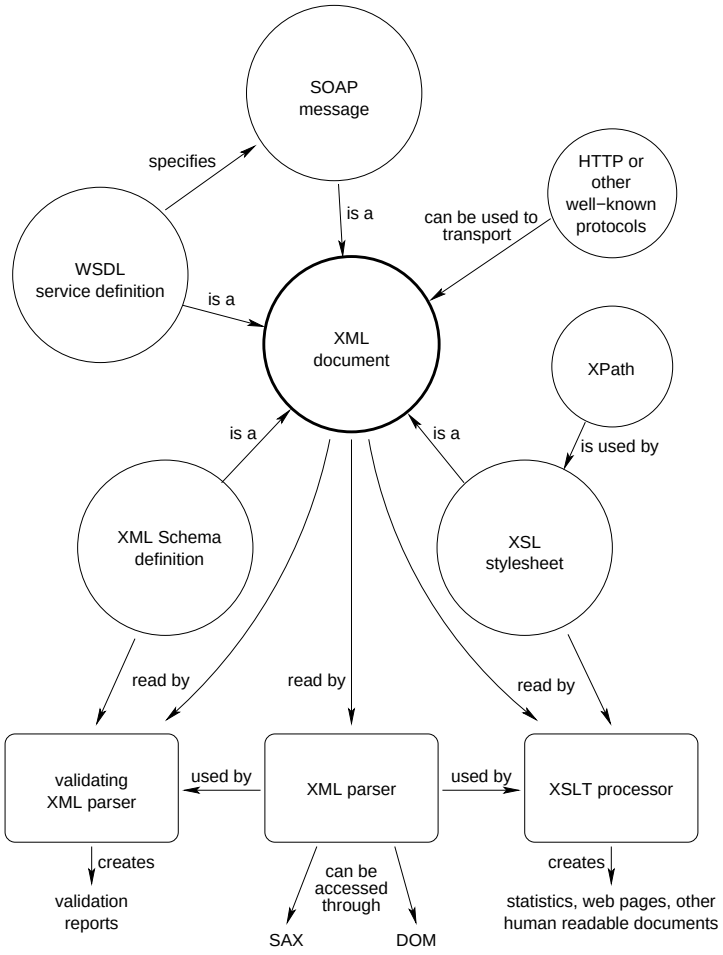

Figure 1: XML technologies and tools [11].

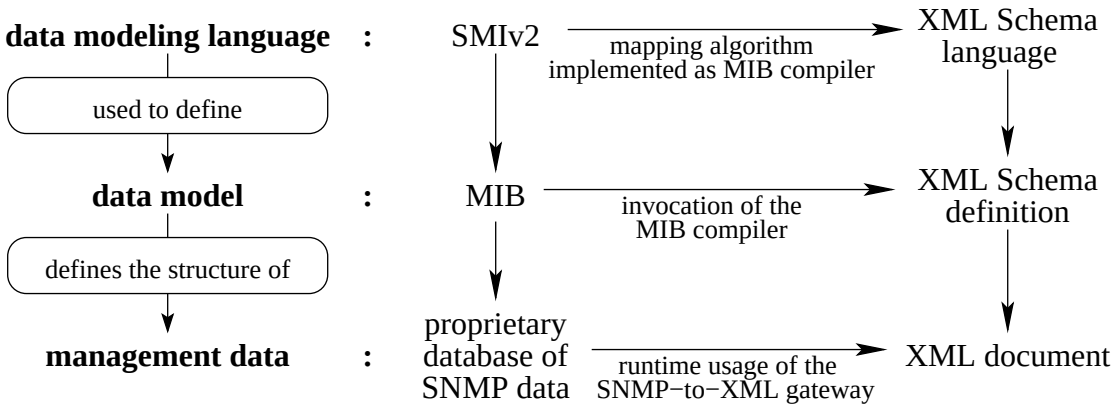

Figure 2: The mappings of data modeling languages, data models and data.

	Mazumdar [6]	Hong et al [7, 8]	Martin-Flatin [4]	Klie/Strauß
XML data model design	driven by usability in XML applications	driven by SMI MIB design	driven by SMI MIB design	driven by usability in XML applications
separation of data model and data	strict (data types, OIDs and access modes only optionally in the XML documents)	not strict (parts of OIDs and full MIB tree hierarchy in the XML documents)	not strict (parts of data type and OID information in the XML documents)	strict (no data types, OIDs and access modes in the XML documents)
SMI MIBs are mapped to...	modular XML Schema definitions (one per table and per scalar group, one for type definitions, and one per MIB which includes the others)	XML Schema definitions	DTDs or XML Schema definitions (usually, one per MIB)	XML Schema definitions (one per MIB)
implementation status	MIB compiler implemented, gateway implementation under way (information from the project web site, last updated in 2001!)	MIB compiler and several gateway approaches implemented	research prototype implementation ('JAMAP'), MIB compilation not yet implemented	MIB compiler and gateway implemented, available as open source

Table 1: Comparison of SNMP-to-XML mapping approaches.

SMI type	XML Schema builtin type
`INTEGER`, `Integer32`	`xsd:int`
`Unsigned32`, `Counter32`	`xsd:unsignedInt`
`Counter64`	`xsd:unsignedLong`
`INTEGER` enumeration	`xsd:NMTOKEN`
`OCTET STRING`	`xsd:hexBinary` or `xsd:string`
`BITS`	`xsd:list` of `xsd:NMTOKEN`
`IpAddress`	`xsd:string` with regexp restriction
`OBJECT IDENTIFIER`	`xsd:string` with regexp restriction

Table 2: The mapping of SMI base types to XML Schema types.

```
<?xml version="1.0" encoding="UTF-8"?>
<snmp-data xmlns="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/snmp-data"
           xmlns:IF-MIB="http://www.ibr.cs.tu-bs.de/projects/libsmi/xsd/IF-MIB"
           [...] >
  <context ipaddr="134.169.246.1" hostname="ciscobs.rz.tu-bs.de" port="161"
           community="public" time="2003-11-10T16:57:31Z">
    <IF-MIB:interfaces>
      <IF-MIB:ifNumber>10</IF-MIB:ifNumber>
    </IF-MIB:interfaces>
[...]
    <IF-MIB:ifEntry ifIndex="2">
      <IF-MIB:ifDescr>FastEthernet0/0</IF-MIB:ifDescr>
      <IF-MIB:ifType>ethernetCsmacd</IF-MIB:ifType>
      <IF-MIB:ifMtu>1500</IF-MIB:ifMtu>
      <IF-MIB:ifSpeed>100000000</IF-MIB:ifSpeed>
      <IF-MIB:ifPhysAddress>00:03:fd:32:e4:00</IF-MIB:ifPhysAddress>
      <IF-MIB:ifAdminStatus>down</IF-MIB:ifAdminStatus>
[...]
      <IF-MIB:ifName>Fa0/0</IF-MIB:ifName>
      <IF-MIB:ifLinkUpDownTrapEnable>enabled</IF-MIB:ifLinkUpDownTrapEnable>
[...]
      <IF-MIB:ifRcvAddressEntry ifRcvAddressAddress="00:00:00:00:03:00">
        <IF-MIB:ifRcvAddressStatus>active</IF-MIB:ifRcvAddressStatus>
        <IF-MIB:ifRcvAddressType>other</IF-MIB:ifRcvAddressType>
      </IF-MIB:ifRcvAddressEntry>
[...]
    </IF-MIB:ifEntry>
[...]
    <IF-MIB:ifStackEntry ifStackHigherLayer="2" ifStackLowerLayer="0">
      <IF-MIB:ifStackStatus>active</IF-MIB:ifStackStatus>
    </IF-MIB:ifStackEntry>
  </context>
</snmp-data>
```

Figure 3: An XML instance document conforming to the IF-MIB XML Schema.

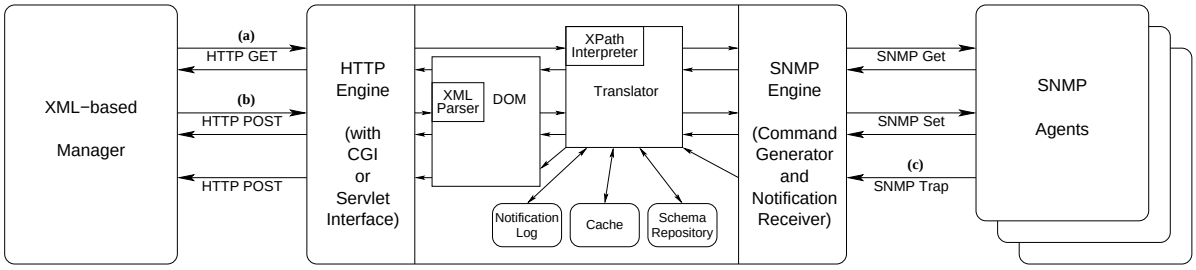

Figure 4: Architecture of the SNMP-to-XML gateway and its three operations: (a) retrieval of management data, (b) control and configuration, (c) propagation of notifications.