Journal of Network and Systems Management, Vol. 13, No. 4, December 2005 (© 2005)
DOI: 10.1007/s10922-005-9007-4

XMLNET: An Architecture for Cost Effective Network Management Based on XML Technologies

Dimitris Alexopoulos [1,3] and John Soldatos[2]

Published online: 16 November 2005

XML technologies have been recently introduced in network management towards alleviating limitations of SNMP. The XML W3C standard, along with XML technologies has the potential to boost open, interoperable, cost-effective and standards-based management solutions. This paper highlights recent efforts towards XML network management and introduces an architecture supporting XML-based network management applications. This architecture specifies a runtime environment that parses XML documents containing composite operations for individual devices, as well as for heterogeneous mutli-vendor networks. Management operations in the scope of XML documents are defined in a programmable fashion based on an XML-based composition language supporting aggregations of elementary operations, looping commands, conditional statements, as well as simple rules signifying the occurrence of specific events. The introduced environment allows network managers to define management operations featuring high-level semantics and accordingly produce sophisticated applications through XML authoring. Following the illustration of the architecture, its composition language and issues relating to security and error handling, the paper ends up presenting a prototype implementation, along with associated performance evaluation results.

KEY WORDS: XML Management; SNMP; EML; NML; MIB; XML schema; DTD; Programmability.

1. INTRODUCTION

During the last decade, IP-based networks and services have evolved, matured, and proliferated. Nevertheless, few advances have taken place in the area of network management of IP networks. The simple network management protocol (SNMP)

[1]National Technical University of Athens, School of Electrical & Computer Engineering, Heroon Polytechneiou Street, Zografou, Greece.
[2]Athens Information Technology, Markopoulo Avenue, Peania, Greece.
[3] To whom correspondence should be addressed at National Technical University of Athens, School of Electrical & Computer Engineering, 9 Heroon Polytechneiou Street, GR-15773, Zografou, Greece; Email: dalexo@telecom.ntua.gr.

1064-7570/05/1200-0451/0 © 2005 Springer Science+Business Media, Inc.

is still the dominant network management protocol for enterprise IP networks, as well as for the Internet. SNMP features high penetration and a large base of applications. However, network managers identify several limitations of the SNMP management framework, mainly relating to configuration management, application development complexity and scalability [1]. Configuration management inefficiencies stem from SNMP's ineffectiveness in bulk information transfers. SNMP application development has to deal with low-level programming, due to the poor semantics of SNMP commands and management information bases (MIB). Also, the use of proprietary MIBs complicates the management of heterogeneous, multi-vendor, multi-device networks. Moreover, reusability of management applications can hardly be achieved.

In view of these limitations, network engineers and researchers have been investigating solutions boosting openness, interoperability and standardized management interfaces, while also facilitating application development. The extensible markup language (XML), standardized as a meta-markup language by W3C, can provide such solutions [2], since:

- XML is easy to generate, parse and process, which provides flexibility in handing XML representations of management information.
- XML supports sophisticated data structuring, and can therefore handle complex organizations of management information. XML DTD (document type definitions) and XML schemas specify and validate the structure of XML documents, thus alleviating developers from tedious tasks.
- XML comes with numerous W3C technologies (http://www.w3c.org) supporting rapid development of XML based network management applications. Characteristic examples are the extensible stylesheet transformations (XSLT), which transform XML documents to other formats (e.g., hyper text mark-up language—HTML) and XPath/XQuery discovering XML elements subject to criteria.
- XML operations can be transformed to simple object access protocol (SOAP) operations allowing management functions to be exported as Web Services. This allows for loose integration of heterogeneous distributed management systems based on the Web Services paradigm.
- XML has high-level semantics and is therefore appropriate for performing bulk configuration operations.

These benefits of XML for network management have given rise to research, standards and industrial initiatives. Early research work focused on designing DTDs corresponding to SNMP MIBs and vice versa, as well as on algorithms for translating MIBs to XML. The most prominent of these efforts is libsmi [3], which allows translation of SNMP SMI to other languages, including XML. Libsmi has been extended with the mibdump utility, which translates MIBs directly to XML. Recent work includes the implementation of XML/SNMP gateways [3, 4], which execute

XML based operations on SNMP devices. While gateways concentrate on groups of individual management operations, other XML-based architectures move towards more sophisticated operations (e.g., the Avaya work included in [2, 4]). Closely related to the XML-based research developments are other research prototypes implementing management solutions based on Web services (e.g., [5, 6]).

Standardization activities have been launched in the Internet and network management communities. The distributed management task force (DMTF) has worked towards representing its common information model (CIM) as an XML DTD. OASIS (management protocol technical committee) is working towards open industry standard management protocols supporting XML mechanisms to manage elements in distributed environments. The internet engineering task force (IETF) has recently established the network configuration working group (netconf) [7], which adopts XML for data encoding and data transfer in configuration management. Furthermore, the network management research group (NMRG) of the internet research task force (IRTF), has allocated significant effort in identifying benefits of XML network management, as well as in developing related solutions [8].

There also industrial efforts on XML network management, which have produced proprietary products, the most prominent examples being Juniper's JUNOScript (an XML application programming interface—API to JUNOS [9]), 2Wire (http://www.2wire.com/) using a modified version of the XML remote procedure call (XML-RPC) protocol to manage their DSL system over a Web-enabled infrastructure, Cisco (http://www.cisco.com/) with the CNS Netflow Engine being manageable through XML messages, and NextHope (http://www.nexthop.com/) offering an XML management interface to their GateD vendor independent router software .

Most of the previous efforts have focused on managing individual network elements or devices [10]. Few efforts have addressed architectures dealing with both network element management operations (element management layer—EML), and network wide management operations (network management layer—NML). XML technologies can be applied in both EML and NML towards supporting operations featuring high-level semantics. The latter can be implemented based on standard XML documents, which provide an alternative to the proprietary languages used by state of the art Network Management Systems (NMS). This is also in line with recent efforts towards exploiting XML technologies in automating management tasks, such as for example in the case of Microsoft's web services for management extension (WMX) [11].

This paper presents an XML based network management architecture addressing both EML and NML. The architecture provides the means for structuring complex management operations (at the EML and NML levels) as XML documents. EML composite operations consist of several atomic management operations each one affecting or querying a single MIB object. NML composite operations are structured as a set of composite EML operations. Authoring specifications (XML schemas) guide the development of XML documents, according to a composition language. This composition language reflects the core programmability of the system, since it makes provisions for aggregating primitive SNMP (`get`/`set` operations) into higher-level operations. Moreover, the composition language supports additional features such as allowing for repeatedly executing operations (i.e., looping), processing information elements and enforcing actions when particular conditions are met. Based on the XML schemas specifying this composition language, network managers define composite management operations as XML-based APIs at both the EML and the NML levels. XML management applications are authored through assembling API operations and defining parameter values. XML authoring of APIs and applications can be facilitated through an editing environment (e.g., graphical user interface—GUI editor). Editing environments can boost the extensibility and reusability of XML documents, and overall increase network managers' productivity.

The architecture specifies also a runtime environment that parses and executes XML-based applications. This environment renders application development a matter of XML authoring. This approach results in cost effective application development, while increasing authoring flexibility and boosting the programmability of management operations [12]. Also, a potential standardization of APIs can allow third parties (e.g., vendors, NMS providers) to produce network management applications. This idea is pertinent to network programmability defined in the scope of the IEEE P1520 initiative [13]. Network programmability can boost a host of network and traffic management functionalities including quality of service (QoS) management [13–15], routing [16] and SLA management and configuration in DiffServ environments [17, 18]. Note also that through adding XML functionality on NML operations, we can extend the introduced architecture to service management [19].

Based on this architecture we have implemented a network management system enabling authoring and execution of XML management documents comprising EML and NML management applications. This system, namely XMLNET (http://www.telem.ntua.gr/xmlnet), makes use of a rich set of XML technologies and XML-based programming techniques. Furthermore, we have conducted a set of performance experiments relating to the execution time of composite management operations. These experiments demonstrate that the programmability of the XMLNET system comes at a very low performance overhead.

The paper is structured as follows: Section 2 following this introduction presents the overall architecture and highlights the main building blocks, namely the EML and NML XML engines. Section 3 presents the XML EML Engine, and Section 4 emphasizes the composition constructs available to network managers for defining XML based APIs. Following the EML implementation details, Section 5 describes the NML engine. Section 6 illustrates the techniques and technologies that support a prototype implementation of the architecture. Based on this prototype, Section 7 presents measurements addressing performance implications of the implemented solutions. Finally Section 8 concludes the paper.

2. XML NETWORK MANAGEMENT ARCHITECTURE

Figure 1 depicts the overall architecture for programmable XML-based network management. We introduce the following terms:

- *Atomic management operation:* An element-level operation querying or altering the value of a particular MIB object or object instance. Practically, atomic operations are simple `get` or `set` operations referring to an object identifier (OID).
- *Composite EML operations:* Higher-level EML operations combining one or more atomic operations. Composite EML operations may access several OIDs.
- *Composite NML operations:* Higher-level operations combining one or more composite EML operations. Composite NML operations may target several OIDs residing on multiple devices.
- *XML EML API:* XML document containing a set of composite EML operations, their breakdown into constituent atomic operations, as well as 'default' values for all `set()` operations.
- *XML NML API:* XML document containing a set of composite NML operations, their resolution into constituent composite EML operations, as well as 'default' values for EML operations involving `set()` operations.
- *XML EML application:* XML document combining EML API invocations with an appropriate set of parameters overriding default API values.
- *XML NML application:* XML document combining NML API invocations with an appropriate set of parameters overriding default API values.

The architecture relies on defining and structuring XML EML APIs for each network element and XML NML APIs for the whole network. Network managers define these APIs based on the management requirements of their network. Each API's structure conforms to an XML schema. XML schemas specify how atomic operations and composite EML operations are respectively combined to composite EML and NML operations. Similarly, XML applications are also authored based on XML schemas. Using XML schemas for specifying the structure of both API and applications facilitates authoring and validation of the respective XML documents. In the tables, we include XML schemas specifying the structure of:

– XML EML APIs and Applications (Tables I and III)
– XML NML APIs and Applications (Tables II and IV).

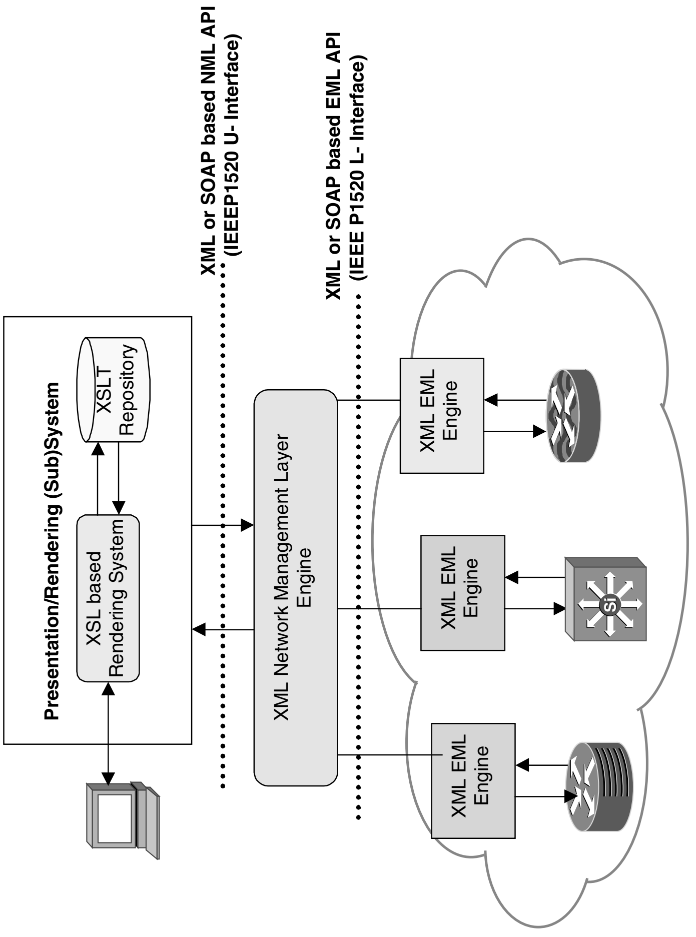
Presentation/Rendering (Sub)System
XSL based Rendering System
XSLT Repository
XML or SOAP based NML API (IEEEP1520 U- Interface)
XML Network Management Layer Engine
XML or SOAP based EML API (IEEE P1520 L- Interface)
XML EML Engine
XML EML Engine
XML EML Engine
SI

Fig. 1. Programmable XML network management architecture.

These schemas correspond to those used in the scope of the XMLNET system implementation, which is described in a later paragraph.

The architecture depicted in Fig. 1, specifies a run-time XML parsing environment allowing execution of XML application documents. Thus, the architecture specifies a system that accepts XML application (EML or NML) documents,

Table I. EML API Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
       <xs:element name="author" type="xs:string"/>
       <xs:element name="description" type="xs:string"/>
       <xs:element name="dns-name" type="xs:string"/>
       <xs:element name="element-management-scheme">
              <xs:complexType>
                     <xs:sequence>
                            <xs:element ref="name"/>
                            <xs:element ref="dns-name"/>
                            <xs:element ref="ip-address"/>
                            <xs:element ref="description"/>
                            <xs:element ref="author"/>
                            <xs:element ref="version"/>
                            <xs:element name="operations" type="operationsType"/>
                     </xs:sequence>
              </xs:complexType>
       </xs:element>
       <xs:complexType name="execType">
              <xs:attribute name="command" type="xs:string" use="required"/>
       </xs:complexType>
       <xs:complexType name="getType">
              <xs:sequence>
                     <xs:element name="if" type="ifType" minOccurs="0"/>
              </xs:sequence>
              <xs:attribute name="parameter" type="xs:string" use="required"/>
              <xs:attribute name="oid" type="xs:string" use="required"/>
              <xs:attribute name="type" use="required">
                     <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                   <xs:enumeration value="C"/>
                                   <xs:enumeration value="S"/>
                                   <xs:enumeration value="T"/>
                                       <xs:enumeration value="I"/>
                            </xs:restriction>
                     </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="seq" type="xs:string" use="required"/>
              <xs:attribute name="deriv">
                     <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                   <xs:enumeration value="false"/>
                                   <xs:enumeration value="true"/>
                            </xs:restriction>
                     </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="dt" type="xs:string"/>
       </xs:complexType>
       <xs:complexType name="ifType">
              <xs:sequence>
                     <xs:element name="exec" type="execType" minOccurs="0"/>
                     <xs:element name="set" type="setType" minOccurs="0"/>
                     <xs:element name="get" type="getType" minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="comparator" use="required">
                     <xs:simpleType>
                            <xs:restriction base="xs:NMTOKEN">
                                   <xs:enumeration value="greater"/>
                                   <xs:enumeration value="equals"/>
                                   <xs:enumeration value="less"/>
                            </xs:restriction>
                     </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="value" type="xs:string" use="required"/>
       </xs:complexType>
       <xs:element name="ip-address" type="xs:string"/>
       <xs:element name="name" type="xs:string"/>
       <xs:complexType name="operationType">
              <xs:sequence>
                     <xs:element ref="description"/>
                     <xs:element name="process" type="processType"/>
```

Table I. Continued.

```
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="snmp-interface" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="java"/>
                                        <xs:enumeration value="xml"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="loop" use="optional">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="true"/>
                                        <xs:enumeration value="false"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="loop-period" type="xs:string" use="optional"/>
                <xs:attribute name="priority" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:complexType name="operationsType">
                <xs:sequence>
                        <xs:element name="operation" type="operationType" maxOccurs="unbounded"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="processType">
                <xs:choice maxOccurs="unbounded">
                        <xs:element name="get" type="getType"/>
                        <xs:element name="calc" type="calcType"/>
                        <xs:element name="set" type="setType"/>
                </xs:choice>
                <!--<xs:sequence>
                        <xs:element name="get" type="getType" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="set" type="setType" minOccurs="0" maxOccurs="unbounded"/>
                        <xs:element name="calc" type="calcType" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>-->
        </xs:complexType>
        <xs:complexType name="setType">
                <xs:attribute name="parameter" type="xs:string" use="required"/>
                <xs:attribute name="type" type="xs:string"/>
                <xs:attribute name="value" type="xs:string" use="required"/>
                <xs:attribute name="seq" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:complexType name="calcType">
                <xs:sequence>
                        <xs:element name="if" type="ifType" minOccurs="0"/>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="type" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="add"/>
                                        <xs:enumeration value="substract"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="parameter1seq" type="xs:string" use="required"/>
                <xs:attribute name="parameter2seq" type="xs:string" use="required"/>
                <xs:attribute name="seq" type="xs:string" use="required"/>
                <xs:attribute name="deriv">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="false"/>
                                        <xs:enumeration value="true"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="dt" type="xs:string"/>
        </xs:complexType>
        <xs:element name="version" type="xs:decimal"/>
</xs:schema>
```

executes them on the network elements, and returns the results. This system receives XML (EML or NML) application documents from remote (e.g., based on XML-RPC). XML processing is performed by two parsing systems:

1. The EML engine that processes EML XML applications based on the EML XML API. According to the P1520 [13], the XML EML API constitutes an L-interface.

Table II. NML API Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="nml-api">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root node of all NML API documents.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="name"/>
				<xs:element ref="description"/>
				<xs:element ref="nm-functions"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="name" type="xs:string"/>
	<xs:element name="description" type="xs:string"/>
	<xs:element name="nm-functions">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root of the declaration of a NML function consisting of
several EML operations.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="func"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="func">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root of the declaration of a NML function consisting of
several EML operations.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="description"/>
				<xs:element ref="process"/>
			</xs:sequence>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="priority" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="process">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="operation" maxOccurs="unbounded"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="operation">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root of declaration of the EML operations which comprise
the NML function.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="param" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="eml-operation" type="xs:string" use="required"/>
			<xs:attribute name="node" type="xs:string" use="required"/>
			<xs:attribute name="snmp-interface" use="required">
				<xs:simpleType>
					<xs:restriction base="xs:NMTOKEN">
						<xs:enumeration value="java"/>
						<xs:enumeration value="xml"/>
					</xs:restriction>
```

Table II. Continued.

```
                                </xs:simpleType>
                        </xs:attribute>
                </xs:complexType>
        </xs:element>
        <xs:element name="param">
                <xs:complexType>
                        <xs:annotation>
                                <xs:documentation>
                                        The declaration of the parametr of the several EML operations
which comprise the NML function.
                                </xs:documentation>
                        </xs:annotation>
                        <xs:simpleContent>
                                <xs:extension base="xs:string">
                                        <xs:attribute name="name" use="required"/>
                                        <xs:attribute name="type" use="required"/>
                                </xs:extension>
                        </xs:simpleContent>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Table III. EML Application Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
        <xs:element name="caller" type="xs:string"/>
        <xs:element name="dns-name" type="xs:string"/>
        <xs:element name="ip-address" type="xs:string"/>
        <xs:element name="name" type="xs:string"/>
        <xs:complexType name="operationType">
                <xs:sequence>
                        <xs:element name="parameter" type="parameterType" maxOccurs="unbounded"/>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="snmp-interface" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="java"/>
                                        <xs:enumeration value="xml"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="priority" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:complexType name="operations-requestType">
                <xs:sequence>
                        <xs:element name="operation" type="operationType"/>
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="parameterType">
                <xs:attribute name="name" type="xs:string" use="required"/>
                <xs:attribute name="oid" type="xs:string" use="required"/>
                <xs:attribute name="seq" type="xs:string" use="required"/>
                <xs:attribute name="type" use="required">
                        <xs:simpleType>
                                <xs:restriction base="xs:NMTOKEN">
                                        <xs:enumeration value="S"/>
                                        <xs:enumeration value="T"/>
                                        <xs:enumeration value="C"/>
                                </xs:restriction>
                        </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="value" type="xs:string" use="required"/>
        </xs:complexType>
        <xs:element name="xmlnet-request">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="name"/>
                                <xs:element ref="dns-name"/>
                                <xs:element ref="ip-address"/>
                                <xs:element ref="caller"/>
                                <xs:element name="operations-request" type="operations-requestType"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Table IV. NML Application Schema Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:element name="NML-Application">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root node of all NML application documents.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="func"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="func">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The root node of the NML function to be invoked.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="operation" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="priority" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="operation">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The declaration of the various EML
					applications that comprise the NML function.
				</xs:documentation>
			</xs:annotation>
			<xs:sequence>
				<xs:element ref="param" maxOccurs="unbounded"/>
			</xs:sequence>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="eml-operation" type="xs:string" use="required"/>
			<xs:attribute name="node" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
	<xs:element name="param">
		<xs:complexType>
			<xs:annotation>
				<xs:documentation>
					The declaration of the parameters passed to the
					EML application that comprise the NML function.
				</xs:documentation>
			</xs:annotation>
			<xs:attribute name="name" type="xs:string" use="required"/>
			<xs:attribute name="value" type="xs:string" use="required"/>
		</xs:complexType>
	</xs:element>
</xs:schema>
```

2. The NML engine that processes NML XML applications based on the NML XML API. In P1520 terms [13], the NML API constitutes a U-interface.

The EML engine processes XML EML application documents, identifies composite EML operations, resolves them to atomic operations and executes them on the network element. At a higher level, the NML engine processes NML application documents, identifies composite NML operations, resolves them to EML operations, constructs appropriate EML documents and resorts to EML engines to execute them. Both the EML and NML engines produce XML documents containing the results of the composite management operations. These XML documents feature a structure identical to the XML applications, with values representing the actual status of the device after performing the operations.

A key element of this architecture is a *composition language* specifying how atomic operations are combined into composite ones, as well as how composite ELM operations are combined to composite NML operations. The composition language realizes the programmability of the architecture, since it transforms XML documents to simple network management programs that can be parsed and executed by the run-time environment. The XML schemas defining the structure of application and API documents reflect the constructs and capabilities of the composition language.

As far as the presentation and visualization of the network management applications is concerned, a rendering subsystem presents XML results to a console. This subsystem fulfills visualization requirements relating to the application and/or user preferences. XML technologies (e.g., extensible stylesheet—XSL) can be exploited towards developing presentation mechanisms. XSL can be used to filter XML documents returned by XML applications. Thus, XSL presentation templates are required, along with (EML or NML) XML application documents. XSL templates can be stored in a repository and retrieved based on application requirements.

Overall, the major benefit of this architecture is that application development becomes a matter of XML authoring, which is more cost effective than conventional SNMP programming or scripts (e.g., in Perl, Tcl) authoring. Moreover, the XML APIs constitute XML protocols allowing execution of NML and EML operations. These protocols expose an interface to potential management applications. A standardization of this interface can make network management operations open and programmable. Openness hinges on that third party vendors and/or network managers can use the APIs to develop applications. Programmability allows different network management applications to be authored through writing and assembling XML documents.

XML interfaces (i.e., APIs) are accessible in a distributed fashion through conventional distributed programming mechanisms (e.g., XML-RPC, RMI—remote method invocation, SOAP). Following paragraphs provide an anatomy of the major XML building blocks of the architecture.

3. ELEMENT MANAGEMENT LAYER ARCHITECTURE

The EML system implements the conventional manager-agent model. Different combinations of XML and SNMP on the manager and agent sides are possible [3]. Our design features an XML-based manager communicating with the SNMP agent of the device through an XML/SNMP gateway. This paradigm exploits most of the benefits of XML management, while also incorporating the installed base of

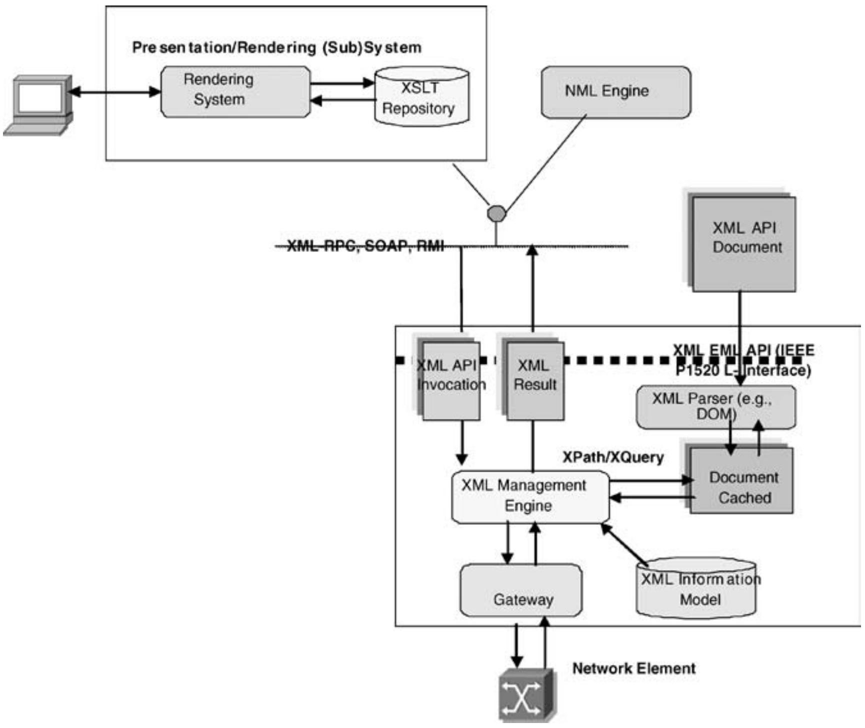

Fig. 2. Element management layer engine.

SNMP devices. Thus, the architecture is applicable to the vast majority of legacy IP-based networks.

The EML engine parses XML EML documents. Composite management operations are accordingly resolved into atomic/elementary operations according to the composition language specified in a subsequent section.

3.1. EML Engine Components

The XML EML engine (Fig. 2) consists of the following components:

- *XML DOM parser:* Upon the element's initialization the XML parser receives the XML EML API of the device. The XML parser processes the XML API document based on DOM (Document Object Model) processing and accordingly instantiates a cached, in-memory representation of the supported composite management operations. The cached representation accelerates the processing of the EML application document.

- *DOM Cache:* The DOM cache module maintains the cached representation of composite EML operations, which is constructed during the EML system's initialization. Having such a representation at hand, the XML Management Engine can easily match operations contained within the XML application document to cached operations. Thus, the DOM cache boosts the performance and scalability of the EML XML manager.
- *XML Management Engine:* The XML management engine receives XML EML application documents through a distributed invocation mechanism (e.g., XML-RPC, Web Services). Accordingly, it parses them, resolves the composite EML operations from the cache and executes the necessary atomic operations. The process of looking up operations in the cache can be greatly facilitated by XPath/XQuery technologies.[4] Atomic operations are executed through the XML/SNMP gateway. The result of these operations follows the reverse direction: results are conveyed from the gateway to the XML engine, which assembles and delivers them to the application.
- *XML Information Model:* The XML information model constitutes an XML representation (e.g., XML schema) of the MIBs supported by the network elements. Such a representation is produced based on utilities converting SMI information to XML schemas (e.g., `mibdump`).
- *XML/SNMP gateway:* The gateway accesses the low-level management capabilities (i.e., the SNMP agent) of the device. The XML/SNMP gateway translates between XML and SNMP representations of the managed objects. Moreover, it executes SNMP (`get`()/`set`()/`GetNext`()) operations, collects the results and delivers them to the XML Management Engine.
- *Rendering System:* A rendering system making use of XSL technology interfaces with the EML engine and presents EML information.

The EML engine parsing services can be invoked by either the rendering subsystem, or a higher-level NML engine. In the former case XML application documents are concerned with element manager applications, while in the later they constitute a part of a composite application on a multi-device network.

Note that trap operations are not straightforward in this design and should be handled through separate XML-based flows. Separate XML information flows are required, since some gateway implementations (e.g., the one used in our prototype) do not support SNMP trap handling. To overcome this limitation an autonomous process listens for the node's trap-sending requests. This process retrieves each trap message, forms an XML message and dispatches it to the rendering engine.

The EML engine is a runtime environment that can be embedded in the device or hosted in an attached management workstation. The instantiation of the

[4]Such a mechanism can be implemented using Xpath/XQuery based search engines (e.g., the Java based XQEngine).

EML engine presupposes that an EML API has been defined. The definition of an EML API demands that the network manager has access to the XML Information model. Whenever a new device is added, the network manager produces an XML representation of its MIBs, defines the XML EML API and instantiates the EML engine.

The EML engine can be instantiated with different parameters towards controlling multiple identical devices supporting the same EML API and XML information model. In heterogeneous multi-vendor environments different EML APIs will have to be produced based on different XML information models. To alleviate this complexity, the EML engine can make use of an abstract information model (AIM), corresponding to a generalized XML-based vendor independent description of MIB objects engaged in the atomic operations. In this case atomic operations will be mapped to vendor independent objects of the AIM. Defining and realizing an AIM, as well as performing this mapping are demanding tasks. A possible solution is the DMTF's CIM (Common Information Model) and its related DTD. Alternatively, an AIM can be constructed through abstracting MIB objects entailed in the operations of interest and mapping them to vendor dependent objects [20].

3.2. Error Handling in EML Composite Operations

A composite EML operation is likely to correspond to a unit of work requiring atomicity from a network manager's perspective. Specifically, network managers will bundle together atomic operations towards automating a series of operations that must be executed in their entirety. As a result, atomic operations comprising composite EML operations should be executed based on an 'all-or-nothing' guarantee: Either all operations should succeed, or all together fail. Thus, the successful completion of composite EML operations hinges on the graceful execution of all comprised atomic operations. This is particularly important for series of set operations that alter the state of the network element.

The all-or-nothing guarantee raises issues relating to error handling for composite operations. The manager application invoking the EML engine service should be notified of the overall success of a composite operation. In the case of a failing configuration management operation, provisions should be made to keep the network element in a consistent state. A strategy for achieving consistency is to restore the original values for all objects and instances altered in the scope of a non-successful composite operation. For every set operation, the *old* value of altered object should be initially retrieved and stored, to allow for a potential restoration. To this end, a transaction manager intercepts all composite operations stemming from the EML system. The transaction manager retrieves and caches object value pairs for each elementary operation contained in the composite operation. Accordingly, it audits the success of all the atomic SNMP operations

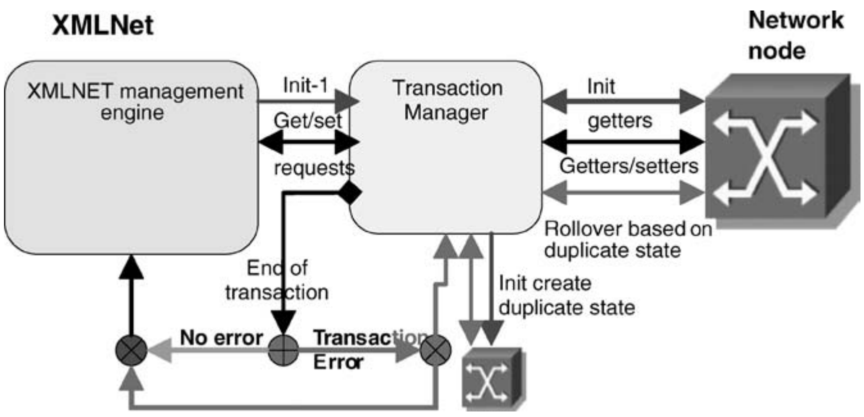

Fig. 3. Error handling.

comprising the composite operation. In the case where the whole set of operations are successful, the transaction manager gracefully competes the transaction and clears the cached values. On the contrary, if at least one operation fails, it undertakes to restore the element to its previous status, as illustrated in Fig. 3. Even though restoring an element's state from cached values seems straightforward, it entails several practical problems. A major one is that several SNMP agents, '*lock*' MIB instances and set them in a pending state, with a view to implementing concurrent access control (i.e., as other processes may attempt to set the target instance). '*Locking*' an item prevents restoration to its old value until a time out '*unlocks*' the object. The transaction manager may therefore need to retry the restoration process until this succeeds for all instances.

3.3. Security

Security issues should be handled towards robust and secure management systems. Authentication and encryption of XML messages is a subject of intensive work, both within the W3C (e.g., in the XML signature working group), and within OASIS (e.g., the security services committee). At the first place WS-security describes how to attach signature and encryption headers to Simple Object Access Protocol messages. Moreover, it specifies how to attach security tokens, such as X.509 certificates and Kerberos tickets, to messages. At a lower layer SNMPv3 [21] includes authentication and encryption mechanisms (reflected in IETF RFCs 2271-2275). RFC 2274 describes the User-based Security Model (USM) for SNMPv3. In particular RFC 2274 defines the process for providing SNMP message-level security. Moreover, the USM emphasizes on protecting users against four threats, namely modification of information, masquerade, message stream modification and disclosure. As a result, enriching the proposed architecture with state of the art security features demands a framework for mapping XML security mechanisms to SNMPv3, which is however out of the scope of this paper.

4. EML APPLICATIONS COMPOSITION LANGUAGE

The EML composition language specifies possible ways for combining atomic operations, while also determining the programmability constructs of the architecture. Based on these constructs the architecture allows network managers to automate a wide range of routine management tasks through XML authoring. In subsequent sections we describe the main composition constructs, which are also reflected in the XML schema definition outlined in Table I.

4.1. Serial Combination of Atomic Operations

Atomic operations can be combined in a serial fashion. This is accomplished through defining composite operations that contain multiple elementary actions (i.e., conforming to the operationType in the schema). Elementary actions may correspond to individual set (), get () SNMP operations, or operations performing simple calculations (i.e., addition, subtraction) between previously derived parameters.

Combining operations in a serial manner is particularly useful in the scope of configuration management operations comprising several atomic operations and impacting a host of different MIB objects. As an example, consider a composite EML operation creating a Virtual Path Connection (VPC) on an asynchronous transfer mode (ATM) switching node, e.g., Create_VPC. This operation sets different OIDs, corresponding to the circuit name (channelname), the usage parameter control (UPC) profile of the connection (channelupc), the end points (channelVPIIn, channelVPIOut) and the lifecycle status of the channel (channelstatus).

4.2. Processing of Management Information (Derivatives)

In several cases network managers are concerned with processing a quantity (e.g., it is usually handy to derive the rate of change of some quantity). A classical example involves the requirement for computing network bandwidth from a cell/packet count on a network interface. Therefore, the XML schema (Table I) guiding the composition of EML management operations, allows one to specify retrieval of the derivative a quantity (through setting the flag deriv). Such a specification implies that the necessary get operations are executed and accordingly the derivative is computed subject to a specified time interval (i.e., specified by the dt attribute).

4.3. Conditions/Thresholds

Conditional functionality is also supported based on an if XML element, as illustrated in Table I. Allowable conditions concern comparisons (i.e. $<$, $>$, $=$ operators) on a management quantity, which is retrieved through a get() function. This composition construct provides flexibility in supporting polling functions, which are commonly associated with particular actions taking place upon the occurrence of specific events. Actions are supported by allowing invocation of external programs (i.e., through the exec element) or services whenever a user-defined threshold is exceeded.

4.4. Looping

Looping allows the same operation to be executed multiple times. Repeating an operation is handy towards auto-polling element level quantities (e.g., for monitoring purposes). The XML schema specified in Table I allows the XML author to specify the number of times an operation will be executed (i.e., see the loop attribute). Looping can be combined with the other constructs described in this section.

4.5. EML API Example

Network managers can combine the constructs towards implementing EML and NML APIs. API documents comprise sets of operations that are combined in a programmable fashion based on the composition language. These composite operations will typically correspond to the routine management tasks. Table V depicts an example of an XML EML API conforming to the schema definition described in Table I. This API consists of the following (composite) functions:

- A batch retrieval of parameters (i.e., serialParameterRetrieval), demonstrating the combination of elementary management commands in a serial fashion
- A function (i.e., bandwidthCheck) performing a bandwidth calculation based on cell counting on an interface and accordingly displaying a console message if a threshold is exceeded. This function demonstrates the threshold functionality, as well as the support for extracting derivatives of management quantities.
- A function (i.e., Create_ATM_PVC) conditionally creating a permanent virtual circuit (PVC) on an ATM interface. This PVC is created only if the number of ATM PVCs established on an interface is less than a given value. This function combines the conditional logic construct (i.e., if statement), with the serial combination of functions.

The API functions outlined above are specified for a particular network node (i.e., an ATM switch). The IP address and DNS name of the IP interface of the node that will ultimately receive the SNMP commands are also specified within the EML XML documents. Using a selected set of these API functions, a management application can be specified in XML format based on the EML schema definition contained in Table III. As an example, Table VI depicts a sample EML application invoking the API functions outlined above.

Table V. Sample EML API

```
<?xml version="1.0" encoding="UTF-8"?>
<element-management-scheme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=".\elm_api_v.1.1.xsd">
       <name>NMScheme</name>
       <dns-name> atm1.atm.ntua.gr</dns-name>
       <!-- required -->
       <ip-address>147.102.7.20</ip-address>
       <!-- required -->
       <description>Sample EML API Document</description>
       <author>Dimitris Alexopoulos</author>
       <version>0.8</version>
       <operations>
              <operation name="serialParameterRetrieval" snmp-interface="java" priority="1">
                     <description>Serial operation comprised of simple GETTERS
                     </description>
                     <process>
                            <get parameter="system/sysName" oid=".1.3.6.1.2.1.1.5" type="S" seq="1"/>
                            <get parameter="system/sysDescr" oid=".1.3.6.1.2.1.1.1" type="S" seq="2"/>
                            <get parameter="system/sysUpTime" oid=".1.3.6.1.2.1.1.3" type="T" seq="3"/>
                            <get parameter="system/sysObjectID" oid=".1.3.6.1.2.1.1.2" type="S" seq="4"/>
                            <get parameter="system/sysContact" oid=".1.3.6.1.2.1.1.4" type="S" seq="5"/>
                            <get parameter="snmp/snmpInBadVersions" oid=".1.3.6.1.2.1.11.3" type="C"
seq="6"/>
                            <get parameter="snmp/snmpInNoSuchNames" oid=".1.3.6.1.2.1.11.9" type="C"
seq="7"/>
                            <get parameter="snmp/snmpInPkts" oid=".1.3.6.1.2.1.11.1" type="C" seq="8"/>
                            <get parameter="snmp/snmpOutPkts" oid=".1.3.6.1.2.1.11.2" type="C" seq="9"/>
                            <get parameter="snmp/snmpInTotalReqVars" oid=".1.3.6.1.2.1.11.13" type="C"
seq="10"/>
                     </process>
              </operation>
              <operation name="bandwidthCheck" snmp-interface="java" priority="1" loop="true" loop-
period="1000">
                     <description>
                     composite operation demonstrating the functionality of a bandwidth check on a specific
interface
                     </description>
                     <process>
                            <get parameter="portUsedBWIn" oid=".1.3.6.1.4.1.326.2.2.2.1.2.2.1.11" type="C"
seq="1"/>
                            <get parameter="portUsedBWOut" oid=".1.3.6.1.4.1.326.2.2.2.1.2.2.1.17" type="C"
seq="2"/>
                            <calc name="bandwidth" type="add" parameter1seq="1" parameter2seq="2" seq="3"
deriv="true" dt="1000">
                                   <if comparator="greater" value="100">
                                          <exec command="echo 'Throughput greater than 100 packets/s' "/>
                                   </if>
                            </calc>
                     </process>
              </operation>
              <operation name="Create_ATM_VPC" snmp-interface="java" priority="1">
                     <description>
                            This operation creates conditionally an ATM VPC (if number of VPs on Part is
less than 10
                     </description>
                     <process>
                            <get parameter="PortNumPathsIn" oid=".1.3.6.1.4.1.326.2.2.2.1.2.2.1.8" type="I"
seq="9" deriv="false" dt="100">
                                   <if comparator="less" value="10">
                                          <get parameter="PortNumPathsOut"
oid=".1.3.6.1.4.1.326.2.2.2.1.2.2.1.14" type="I" seq="9" deriv="false" dt="100">
                                                 <if comparator="less" value="10">
                                                        <set parameter="pathupc"
oid=".1.3.6.1.4.1.326.2.2.2.1.3.2.1.13" type="I" value="1" seq="5"/>
                                                       <set parameter="InVpcPathr"
oid=".1.3.6.1.4.1.326.2.2.2.1.3.2.1.2" type="I" value="1" seq="6"/>
                                                       <set parameter="OutVpcPathr"
oid=".1.3.6.1.4.1.326.2.2.2.1.3.2.1.4" type="I" value="2" seq="7"/>
                                                       <set parameter="origPathStatus"
oid=".1.3.6.1.4.1.326.2.2.2.1.3.3.1.3" type="I" value="0" seq="8"/>
                                                 </if>
                                          </get>
```

Table V. Continued

```
                                        </if>
                              </get>
                        </process>
                  </operation>
            </operations>
</element-management-scheme>
```

5. NETWORK MANAGEMENT LAYER ENGINE

NML management provides a higher-level view of the network and can therefore significantly boost the scalability [22] of our framework. In particular, the NML engine and the respective XML NML API can be used to implement XML management applications targeting mutli-element networks. The network management engine makes use of the EML XML engine to interface to individual network elements. Based on several EML systems, the NML engine can carry out operations involving multiple devices and combining multiple EML operations.

The NML engine reveals many similarities to the EML engine. Figure 4 depicts the main building blocks of the NML system. The NML Engine exposes an XML API that is structured according to an XML schema. Tables II includes the XML schema definition used in our prototype implementation. The NML API contains composite NML operations consisting of composite EML operations, which are invoked through a distributed mechanism. Composite EML operations are executed from the EML XML engines residing on each network device. The NML engine operates as follows:

- Parses the XML NML API document and resolves the composite EML operations. This includes identifying the node targeted in each one of the EML operations and extracting parameter values. An XPath/XQuery module performs these tasks through locating the EML operations within the EML APIs. Alternatively, composite EML operations can be cached and lookup up in a repository.

Table VI. Sample EML Application

```
<?xml version="1.0" encoding="UTF-8"?>
<xmlnet-request xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation=".\elmApp_v1.0.xsd">
       <name>NMScheme</name>
       <!-- required -->
       <dns-name>atm1.atm.ntua.gr</dns-name>
       <!-- required -->
       <ip-address>147.102.7.20</ip-address>
       <caller>Soldatos</caller>
       <operations-request>
              <operation name="bandwidthCheck" snmp-interface="java" priority="1">
              </operation>
              <operation name="Create_ATM_VPC" snmp-interface="java" priority="1">
                     <parameter name="channelupc" seq="5" type="I" value="2"/>
                     <parameter name="channelValidateStatus" seq="8" type="I" value="1"/>
              </operation>
       </operations-request>
</xmlnet-request>
```

- Delegates composite EML operations to EML XML systems. This entails the task of constructing the appropriate EML XML application documents.

An XML management engine resolves composite NML operations to composite EML operations, constructs the respective EML applications, and delivers them to the EML engines. At the same time it collects the XML documents resulting from the EML operations and assembles the XML result. The NML API and application XML schema definitions are respectively illustrated in Tables II and IV.

The NML engine architecture addresses error handling and security issues. Error handling and consistency issues arise because NML operations consist of several EML operations. The NML engine keeps track of the success of the underlying operations. Accordingly it notifies network managers of the results. Element consistency hinges on the underlying capabilities of the EML engine. As the EML engines can restore state, the NML engine can delegate NML restoration to them through analyzing NML state restoration to EML state restoration processes. As far as security is concerned, the NML engine should make provisions for encrypting NML messages, as well as generated EML messages. EML engines handle lower-level security issues.

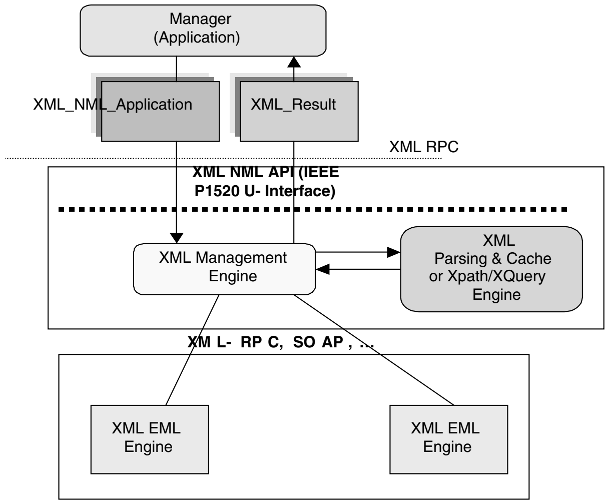

Fig. 4. Network management layer engine

6. PROTOTYPE IMPLEMENTATION

A prototype of the EML engine has been implemented, along with a proof-of-concept implementation of the NML subsystem. The resulting system is called XMLNET and is publicly available at http://www.telecom.ntua.gr/xmlnet. We specified two XML schemas (i.e., depicted in Tables I and II) defining structuring of composite EML and NML operations. The implementation supports the composition language described in Section 4. While this is not a fully fledged programming language, it clearly demonstrates the benefits of composite operations.

The EML engine implements the architecture in Fig. 2. The SNMP/XML gateway (SXG) from Strauss and Klie [2, 3] is used to execute atomic operations based on HTTP calls from upper layers. Alternatively, a Java-SNMP API can be used to execute atomic SNMP operations on the device. Executing atomic SNMP operations based on a Java API (i.e., non XML management) system constitutes a slight deviation from the pure XML-based nature of the system. Note however that providing an alternative way of executing SNMP operations was expedient since the version of the SXG gateway module that we make use of, does not support instance creation/deletion. This resulted in lack of functionality for configuration operations, which was alleviated based on the Java-API implementation, as shown in Fig. 5. Therefore, XMLNET relies on Java-SNMP API for creating and destroying instances in the scope of SNMP *set* operations, while *get* operations can be executed based on either the Java API or the SXG. We expect future versions of the gateway to support instance creation and deletion, thus allowing for a pure XML implementation.

The XML Information model used in the EML engine is produced through running mibdump and libsmi for the target devices. A script constructing the XML Information model for a given SNMP device has been implemented. Running this script for each new device, allows the device to be managed through the XMLNET system. In the case of instance creation/deletion operations, the XML information model can be bypassed, so that atomic operations are executed through passing OIDs to the Java API. These exceptional cases making use of the Java API are supported in the XML schema defining the EML API.

The resolution of composite EML operations to atomic operations is based on the DOM cache paradigm. The cache is initialized following a DOM-based parsing of the EML API document. A DOM parser is favored over SAX (Simple API for XML), since the performance penalty of the DOM parsing is imposed only during initialization of the EML engine. Furthermore, global EML information (e.g., IP address, SNMP community words) is supplied during initialization.

The XML management engine exports an XML-RPC interface and is capable of executing XML EML applications based on Java SAX processing. XML operations are resolved in the DOM cached representation and accordingly mapped to atomic operations. Atomic operations are in the sequel mapped to the appropriate

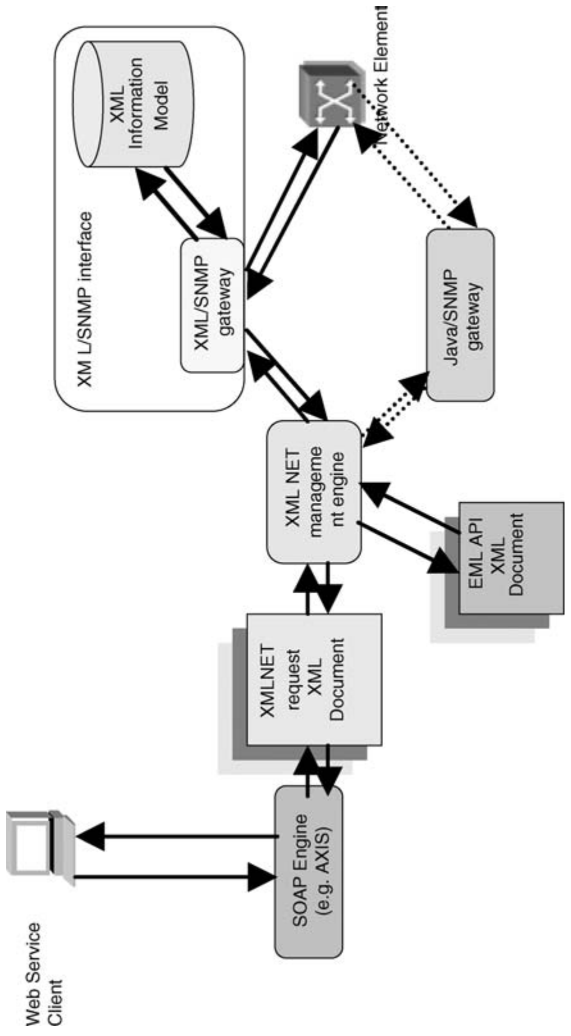

Fig. 5. XMLNET SNMP interfacing (Java and SXG).

vendor-specific MIB object. Using the EML engine network managers can specify and implement EML management operations through authoring XML files. The engine has been tested with various devices (e.g., workstations, ATM switches, routers).

As far as network-level management is concerned a simple implementation has been carried out. A rendering system has also been implemented based on Cocoon (http://cocoon.apache.org/), which provides inherent support for component-based development, flexibility in selecting and visualizing particular pieces of management information and XSL mechanisms for transforming documents.

7. PERFORMANCE EVALUATION

Based on the prototype implementation of our XML system, we conducted a set of performance measurements relating to the response times of composite operations contained in XML documents and executed through the XMLNET system. Our main objective was to investigate the additional performance penalty imposed by the parsing of composite XML applications, compared to conventional SNMP managers executing atomic operations. To this end, we compared the response times of composite EML management operations executed through the XMLNET system, with the respective response times of the atomic operations contained within the composite EML operations. Response times presented in the sequel constitute average figures over 10 measurements to absorb statistical fluctuations in the load of the hosting machines, as well as in network conditions.

Figure 6 depicts the response times of composite EML operations executed through the XMLNET system using the Java API as an underlying execution mechanism of SNMP operations. Response times are shown for composite operations comprising 3, 5, 8, 10, and 15 atomic SNMP operations corresponding to MIB II objects. We also illustrate the times required to execute the same number of atomic operations, as a set of distinct functions through the Java-SNMP library. The main result stemming from Fig. 6 is that the XMLNET system incurs a very slight performance overhead, being approximately in the range of very few milliseconds (e.g., 3–4 ms). This is mainly due to the additional processing for assembling the resulting XML document. Note that this overhead is quite high for composite operations comprising a few constituent atomic operations (e.g., around 50% for three operations), but becomes gradually insignificant as the number of atomic operations increases.

Similarly Fig. 7 illustrates the additional delay contributed by the XMLNET system, when the SXG module is used for executing atomic SNMP operations on the target device. In absolute numbers the additional delay is comparable to that observed in the case of the Java-SNMP library (e.g., 3–4 ms). However, this delay represents a very small percentage of the total delay, given the performance of the version of the SXG module, which we use in the scope of the XMLNET. Note that

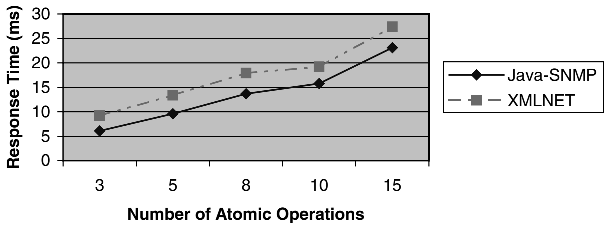

Fig. 6. XMLNET vs. Java-SNMP response times for different numbers of atomic operations.

the two curves in Fig. 7, one corresponding to SXG and the other to the XMLNET over SXG almost coincide. The SXG performs much worse that the Java-SNMP library given that it is performing time consuming tasks (e.g., expensive DOM parsing) and invoked as http servlet.

The extra delay introduced by the XMLNET system does not constitute a performance bottleneck for enterprise network management. We therefore argue that the flexibility of XMLNET for sophisticated application development, along with the resulting cost effectiveness comes at a very low performance cost. Absolute delay figures are decent though not appropriate for enterprise management. This is due to the numerous overhead factors (e.g., Java EML parsing, SXG, HTTP SXG invocation, Java Virtual Machine) entailed in the prototype implementation. The overall performance can be significantly improved in the scope of an embedded commercial implementation.

8. CONCLUSIONS

This paper presented recent advances in using XML technologies for network management. Moreover, it introduced a framework for structuring and executing XML management applications. This framework exploits XML technologies in order to specify XML APIs for network devices, but also composite networks. Based on this framework network managers can produce applications through minimal effort (i.e., authoring XML documents). High-level network manage-

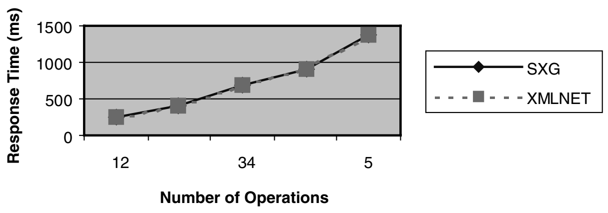

Fig. 7. XMLNET vs. SXG response times for different numbers of atomic operations.

ment interfaces are defined based on a format that can be standardized. This is an advantage of the architecture over commercial NMS systems, which provide proprietary high-level interfaces. High-level interfaces are important to automating a host of network and traffic management operations [13–18, 23].

Key to this framework is the ability to define composite management operations by synthesizing elementary SNMP operations. Composition is based on a set of programming constructs including conditional statements, processing of management quantities, looping as well as serial combinations of programming operations. Based on these constructs a large number of routine network management tasks relating to configuration and performance management can be supported through XML authoring. The programmability of the system can be augmented, by enhancing the XML schema driving operations composition.

The proposed architecture opens new paths to cost effective network management, while at the same time increasing flexibility in producing network management applications. The architecture supports also Web Services, allowing for standards-based integration with other systems. The benefits of XML technologies for network management are evident in the prototype implementation.

ACKNOWLEDGEMENTS

The authors gracefully acknowledge F. Strauß and T. Klie for providing the source code of the SXG (Version 1.0) and mibdump.

REFERENCES

1. J. Schönwälder, A. Pras, and J. P. Martin-Flatin, On the Future of Internet Management Technologies, *IEEE Communications Magazine*, Vol. 41, No. 10, Oct 2003.
2. F. Strauß and T. Klie: Towards XML oriented Internet Management, in *Proceedings of 8th IFIP/IEEE International Symposium on Integrated Network Management*, Colorado Springs, pp. 505–518, 2003.
3. T. Klie and F. Strauss, Integrating SNMP Agents with XML-Based Management Systems, *IEEE Communications Magazine*, Vol. 42, No. 7, July 2004, pp. 76–83.
4. Jeong-Hyuk Yoon, Hong-Taek Ju, and James W. Hong, Development of SNMP-XML Translator and Gateway for XML-based Integrated Network Management, *International Journal of Network Management (IJNM)*, Vol. 13, No. 4, July–August 2003, pp. 259–276.
5. R. Boutaba, W. Golab, Y. Iraqi, and B. St. Arnaud, Lightpaths on Demand: A Web-Services-Based Management System, *IEEE Communications Magazine*, Vol. 42, No. 7, July 2004, pp 101–107.
6. G. Pavlou, P. Flegkas, S. Gouveris, and A. Liotta, On Management Technologies and the Potential of Web Services, *IEEE Communications Magazine*, Vol. 42, No. 7, July 2004, pp. 58–66.
7. IETF Network Configuration Working group, http://www.ietf.org/.
8. Network Management Research Group, http://www.ibr.cs.tu-bs.de/projects/nmrg/.
9. ‘XML based Network Management’, Juniper Networks White Paper, (electronically available at: http://www.juniper.net/solutions/literature/white_papers/).
10. L. E. Menten, Experiences in the Application of XML for Device Management, *IEEE Communications Magazine*, Vol. 42, No. 7, July 2004, pp. 92–100.

11. ‘WMX for Hardware Management Overview’, WinHEC 2004 Version — June 11, 2004, http://www.microsoft.com/whdc/system/platform/server/default.mspx.
12. J. Soldatos and D. Alexopoulos, Cost effective IP networks Management based on XML Authoring, in the *Proc. of the Terena Network Conference, TNC 2004*, Rhodes, Greece, 7–10 June 2004.
13. J. Biswas, A. Lazar, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein, The IEEE P1520 Standards Initiative for Programmable Network Interfaces, *IEEE Communications Magazine*, October 1998, pp. 64–71.
14. E. Al-Shaer and Y. Tang, QoS Path Monitoring for Multicast Networks, *Journal of Network and Systems Management*, Vol. 10, No. 3, September 2002, Special Issue on Internet Traffic Engineering and Management.
15. J. Jun and S. Papavassiliou, Providing End-to-End Quality-of-Service with Optimal Least Weight Routing in Next-Generation Multi-Service High-Speed Networks, *Journal of Network and Systems Management*, Vol. 10, No. 3, September 2002, Special Issue on Internet Traffic Engineering and Management.
16. R. Boutaba, W. Szeto, and Y. Iraqi, DORA: Efficient Routing for MPLS Traffic Engineering, *Journal of Network and Systems Management*, Vol. 10, No. 3, September 2002, Special Issue on Internet Traffic Engineering and Management.
17. F. Boavida et al. PROQOS Dynamic SLA Management in DiffServ Space Links, *Journal of Network and Systems Management*, Vol. 14, No. 4, December 2004.
18. Michael A. Bauer and Hadee A. Akhand, Managing Quality-of-Service in Internet Applications Using Differentiated Services, *Journal of Network and Systems Management*, Vol. 10, No. 1, March 2002 (Special Issue on Management of Converged Networks).
19. D. Alexopoulos and J. Soldatos, Programmable Network Management based on the Web Services paradigm, *WSEAS Transactions on Computers*, Vol. 3, No. 1, January 2004, pp. 250–255.
20. K. Kontovasilis, G. Kormentzas, N. Mitrou, J. Soldatos, and E. Vayias, A Framework for Designing ATM Network Management Systems by Way of Abstract Information Models and Distributed Object Architectures,’ *Computer Communications, Elsevier* Vol. 24 (2001), pp. 641–653.
21. W. Stallings, SNMPv3: A Security Enhancement for SNMP, *IEEE Communication Surveys and Tutorials*, Vol. 1, No. 1, Fourth Quarter 1998.
22. R. Boutaba and A. Polyrakis, Extending COPS-PR with Meta-Policies for Scalable Management of IP Networks, *Journal of Network and Systems Management*, Vol. 10, No. 1, March 2002 (Special Issue on Management of Converged Networks).
23. H. C. Ozmutlu, N. Gautam, and R. Barton, Managing End-to-End Network Performance via Optimized Monitoring Strategies, *Journal of Network and Systems Management*, Vol. 10, No. 1, March 2002 (Special Issue on Management of Converged Networks).

Dimitris Alexopoulos was born in Athens, Greece, in 1979. In 2002 he received the diploma of electrical engineering and computer science from the National Technical University of Athens (NTUA). He has worked as software engineer for INTRACOM S.A and INTRASOFT INTERNATIONAL S.A. Since 2003 he is a PhD candidate and a research associate at the NTUA (Telecommunication Laboratory). His research interests are distributed and programmable network management, service oriented programming, grid computing and web services.

John K. Soldatos was born in Athens, Greece in 1973. He obtained his Dipl-Eng. degree in 1996 and his PhD in 2000, both from the National Technical University of Athens. He has had an active role in numerous EC funded research projects (ESPRIT, ACTS, IST, FP6). He has also consulted in many ICT projects for leading greek enterprises. He has co-authored more than 50 papers published in international journals and conference proceedings. Since March 2003 he is with AIT, where he is currently an Assistant Professor. His current research interests are in Network Control and Management, Grid, Ubiquitous and Autonomic Computing.