Open Programmable XML based Network Management: Architecture and Prototype Instantiation

Dimitris Alexopoulos (*) and John Soldatos (**)

(* )National Technical University of Athens
Department of Electrical and Computer Engineering
Zografou, Attiki, GR-15773, Greece
Tel: +302107722549, Fax: +302107722534
E-Mail: dalexo@telecom.ntua.gr

(**)Athens Information Technology
Peania, Attiki, GR-19002, Greece
Tel: +302106682700, Fax: +302106682703
E-Mail: jsol@ait.gr

Abstract
XML technologies have been recently proposed and gradually adopted to alleviate limitations of conventional SNMP based management. The XML W3C standard, along with a wide range of XML technologies have the potential to boost open, flexible, interoperable, standards based management solutions that are easy to develop and cost effective to operate. This paper presents recent efforts towards XML based network management and introduces an architecture for programmable XML based network management applications. This architecture exploits XML technologies, as well as advances in XML based management in order to specify XML APIs for managing individual devices, but also composite heterogeneous multi-vendor networks. Based on this architecture network managers can produce sophisticated applications simply by authoring XML documents. Following the detailed presentation of the architecture, we elaborate on the technical details of a prototype implementation.

1. Introduction
During the last decade, IP based networks and services have evolved, matured and proliferated. Nevertheless, very few advances have taken place in the area of network management of IP networks. The Simple Network Management Protocol (SNMP) devised in the late 1980’s is still the dominant network management protocol for both enterprise IP networks, as well as for the Internet. SNMP presents many advantages mainly due to its simplicity, and therefore has high penetration and a large base of applications. At the same time network managers have identified several limitations of the SNMP management framework, mainly relating to configuration management, application development complexity and scaling for large networks [1]. Configuration Management inefficiencies stem from the fact that SNMP is not effective in bulk information transfers. SNMP application development has still to deal with low level programming tasks, given that the semantics of SNMP commands and Management Information Bases are quite poor. In addition, several management tasks and applications rely on proprietary MIBs, which complicates the task of managing the current heterogeneous, multi-vendor and multi-device networking infrastructures. Moreover, interoperability, reusability of management applications can hardly be achieved.

In view of these limitations, network managers, engineers and researchers have been looking into solutions that could boost openness, interoperability, standardized management interfaces, while at the same time making application development easier, more flexible and cost effective. The Extensible Markup Language (XML), standardized as a meta-markup language by W3C, as well as a wide range of technologies built around XML, have the potential to boost network management efficiency in these directions [2]. The major drivers behind employing XML and XML technologies for advancing state of the art network management systems are the following:

- XML being an ACSII format is easy to generate, parse and process. XML representations of management information can be flexibly handled based on a rich set of existing libraries and tools.
- XML allows for sophisticated data structuring, thus being able to handle complex organizations of management information. Moreover, XML DTD (Document Type Definitions) and XML
Schemas can impose and validate the structure of XML documents, thus alleviating application developers from tedious validation tasks.

- XML is surrounded by a wide range of W3C technologies (accessible through http://www.w3c.org). As characteristic examples, XSLT can be used to transform an XML document to another, XPath/XQuery to support discovery of XML elements and data conforming to certain criteria, XML databases to allow for storing and retrieving XML documents. These technologies are evolving in a rapid pace, and come with a rich set of tools, libraries, patterns and components. The latter provide a basis for rapidly developing XML network management applications.

- XML documents can be enhanced with new tags to incorporate new formats and accommodate emerging standards. DTDs and XML Schemas can be employed to define and enforce new standards, while XSL facilitates migration from legacy formats.

- XML management operations can be transformed to SOAP operations allowing management functions to be exported as web services. This can allow for loose integration of heterogeneous distributed management systems based on the Web Services paradigm. Similarly, integration can be supported with other types of systems (e.g., Operations Support Systems (OSS)).

- XML can support the development of open, standardized and programmable interfaces. This is because XML is a structured (meta)language that could be interpreted in a standardized manner. XML messages can form whole network management applications, consisting of numerous individual operations.

- XML is ideal for overcoming SNMP limitations in configuration management applications. The rich set of XML based configuration options provides support for provisioning applications, centralized network databases, configuration patch files and archives. Note that XML has higher level semantics comparing to SNMP, and can therefore be more efficient in implementing configuration management applications. This results in a better utilization of network resources as well.

These acknowledged benefits of XML have given rise to several research, standards and industrial initiatives, which have already produced XML management components and software libraries, as well as network management products. Many efforts have concentrated on applying XML management to the large number of legacy SNMP based devices. In the research community, early SNMP/XML integration work focused on mapping DTDs to SNMP MIBs and vice versa, as well as on algorithms for translating MIBs to XML. The most prominent of these efforts is probably 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. More recent work includes the implementation of XML/SNMP gateways (e.g., [4]), which move beyond simple MIB to XML mapping, to supporting management XML based operations on SNMP devices. While, XML/SNMP gateways concentrate on groups of individual management operations, other XML based architectures move one step further towards supporting more sophisticated network element management operations (e.g., [5]).

The potential impact of XML in network management has instigated standardization activities, both in the World Wide Web community, as well as 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, for use in the scope of Web Based Enterprise Management Applications. OASIS (Management Protocol Technical Committee) is working towards open industry standard management protocols supporting XML web-based mechanisms to manage elements (including networking devices) in a distributed environment. The interest in the use of XML-based technologies to manage IP-based networks has also incited the Internet Engineering Task Force (IETF) to organize a BOF (Birds of Feather) initiative, which discussed high-level requirements for network configuration, and explored the potential application of XML in network management. Recently, the network configuration working group was established within IETF (IETF netconf)[6]. This group has adopted XML for data encoding and data transfer processes in configuration management. Furthermore, the Network Management Research Group (NMRG) of the Internet Research Task Force (IRTF), has allocated much effort in identifying the benefits of XML and Web Services based network management, as well as in boosting the development of XML solutions [7].
The picture of XML based network management is completed by industrial efforts that have produced proprietary products, the most prominent examples being Juniper’s JUNOScript (an XML API to JUNOS) [8], 2Wire (http://www.2wire.com/) using a modified version of the XML-RPC protocol to manage their DSL system over a Web-enabled infrastructure, Cisco with the CNS Netflow Engine being manageable through XML messages [9], and NextHope offering an XML management interface to their GateD vendor independent router software[10].

Note that most efforts have focused on managing individual network elements. Few efforts have addressed network management architectures that can deal with both management of individual network elements (i.e. the Element Management Layer (EML)), as well as with more complex operations targeting the whole network (i.e. the Network Management Layer (NML)). It is envisaged however that the benefits of XML technologies can be leveraged in both EML and NML operations.

Aligned to this direction the present paper presents an XML based network management architecture addressing both EML and NML. Based on the architecture XML management applications can be produced in the form of XML documents. Our aim is to expose open XML management interfaces allowing individual EML operations, as well as NML operations expressed as an assembly of EML operations. This results to two distinct APIs, one for the individual element and another for a wider network (e.g., a campus (sub)network). The idea is conceptually pertinent to network programmability at different layers as defined in the scope of the IEEE P1520 initiative [11]: the NML functions make use of EML APIs. Thus, we make use of the P1520 terminology and reference model in conveying our ideas. We argue that our architecture could render network management operations programmable. Based on this architecture we can provide a programmable infrastructure allowing development of a rich set of network management applications as a set of XML documents. We also believe that through adding XML functionality to more layers of the P1520 model, we could extend this architecture to account for Service Management [12].

The paper is structured as follows: Section 2 following this introduction presents the overall management architecture and highlights its main XML based building blocks, namely an EML and an NML XML engine. Section 3 presents the architecture of the XML Element Management Layer Engine. Having presented an EML XML engine, Section 4 elaborates on the NML architecture, assuming that an EML engine is available for every managed device of the network. Section 5, provides details on the specific implementation techniques, tools and technologies that support a practical prototype instantiation of the architecture. Finally, section 6, concludes the paper and provides directions for relevant future work.

2. Programmable XML based Network Management Architecture

Figure I, depicts the overall architecture for programmable XML based network management. In the scope of this architecture we structure network management applications and operations as XML documents. These documents specify and comprise sets of individual management operations concerning particular devices. Therefore, an XML document can be decomposed at element level operations featuring a finer granularity. At the heart of this architecture lies an NML system than receives XML documents from remote (e.g., based on XML-RPC), processes them and interprets the contained sophisticated multi-device operations. This system delegates element management operations to particular instances of XML EML engines that focus on particular devices. EML Engines receive XML documents (e.g., through XML-RPC) containing operations for individual elements, execute the operations on the devices and return the results to the NML engine. The latter assembles these results so as to produce the expected XML output.

In this way an XML protocol is defined allowing execution of NML operations. This protocol exports an interface (API) to potential management applications, thus making network management operations open and programmable. Openness hinges on the fact that third party vendors and/or network managers could use this API to develop management applications. Programmability lies in the fact that different network management applications can be authored merely through writing and assembling XML documents. This facilitates the task of developing and deploying network management applications: as soon as the XML engines receive valid XML documents, they are able to execute the application.

As already outlined an EML system and an associated XML EML API are key prerequisites towards delivering higher level functions and APIs. Thus, our architecture consists of two main building blocks:
The Element Management Layer (EML) Engine responsible for processing XML documents targeting operations on individual devices. According to the P1520 terminology, these XML documents constitute an L-interface.

The Network Management Layer (NML) engine exporting an XML based API for managing the whole network. In P1520 terms, this constitutes a U-interface.

The APIs in both engines are XML based, since the invocation and its parameters (XML_API_Call) as well as the results (XML_API_Result) to be returned in the scope of the management application are XML documents (Figure II). The structure of these documents is essentially the programmable API exported by the nodes and the network. This structure could be defined in terms of either a DTD or an XML Schema.

The APIs are accessible as XML interfaces in a distributed fashion through conventional distributed programming mechanisms (e.g., XML RPC (Remote Procedure Calls), RMI (Remote Method Invocation)). Alternatively, they could be exported as Web Services. This requires that special interfaces to the corresponding XML engines (Web Services Gateway) are hosted and deployed in SOAP containers so that they process SOAP XML documents. As a result, the architecture is perfectly in line with the W3C Web Services paradigm. Nevertheless, this implies a change in the invocation mechanism: while the XML API is accessed through XML-RPC like calls, Web Services API will be accessed using SOAP over HTTP. The management application may also access a WSDL (Web Services Description Language) document.
describing the programmable network management interfaces. Selecting between an XML or a Web Services API, hinges on application specific requirements. Web Services should be the primary choice when standards based integration with other management systems, within an intranet, extranet or even the Internet is required.

No matter whether standard Web Services or other distributed XML based mechanisms are used for invoking the management functions, the management application needs to present the results to a management console. To this end, a rendering subsystem based on XML technology is proposed. The purpose of this system is to adapt the management information to the particular console, as well as other visualization requirements stemming from the nature of the management application or user preferences. We propose that the rich set of XML technologies is exploited towards developing the presentation mechanisms. In our implementation (described in a later paragraph), we leverage the Cocoon application framework [13], due to its inherent support of component-based development and flexibility in selecting and visualizing particular pieces of management information for XML document databases. Cocoon includes XSLT mechanisms for transforming XML documents to other XML documents (mark ups). Based on XSL the network management application developer can control the presentation of XML API results focusing on particular pieces of management information. To this end, the developer has to provide an appropriate (XML based) XSLT template for the particular management application. As a result, authoring an application based on the architecture demands that:

- an XML document comprising a selected set of Network Management Layer API operations is produced, along with
- an XSL document specifying the presentation aspects of the management application.

A technical anatomy of the major building blocks of the system follows.

### 3. Element Management Layer Engine

In designing and implementing the XML EML engine for our system, we leverage as much as possible existing XML developments relating to managing individual network devices. The EML system is based on the conventional manager-agent model. In the scope of this model, four different XML based element management paradigms are possible depending on whether the manager and agent are XML or SNMP based. In our architecture we adopt the paradigm featuring 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 based management, while being able to incorporate the large installed base of legacy SNMP enabled devices. Indeed the XML manager is the module where most of the beneficial XML network management concepts can be applied. The adoption of the XML manager – SNMP agent paradigm renders our architecture applicable to the vast majority of IP based networks and devices.

In terms of functionality, the EML engine parses XML documents and executes management operations on target devices. XML documents contain composite management operations that allow sophisticated management functionality to be implemented on the device. Composite management operations are accordingly broken into atomic management operations, which correspond to certain OIDs in the target SNMP MIBs. As a example, consider a composite element level operation creating a Virtual Path Connection on an Asynchronous Transfer Mode (ATM) switch, e.g., `Create_VC`. This operation asks for several operations setting different Object Identifiers (OIDs), such as setting the circuit name (`channelname`), the Usage Parameter Contol (UPC) profile of the connection (`channelupc`), the end points of the connection (`channelVPIIn`, `channelVPIOut`) and the lifecycle status of the channel (`channelstatus`). The latter constitute atomic operations. Figure III depicts the architecture of the XML EML engine, while Listing I provides a simple example of an EML XML Management operation (and the atomic operation it contains) The corresponding XML based reply would had the same format with the values set to the actual status of the device after performing the operation.
Core components of the EML engine are the:

- **XML parser**: The XML parser acts upon the element’s initialization. At that time it receives XML documents containing the full set of composite management operations that can be performed on the device. These operations constitute the API of the element. The parser processes the XML document and instantiates a cached representation of the whole set of allowable composite operations on the network device. This cached representation is maintained in a local repository or accessed from remote (e.g., using the RMI protocol). Towards a well-performing implementation, we favor SAX (Simple API for XML) based parsing over DOM (Document Object Model), given the potential size of the XML documents for state of the art network devices.

- **Cache**: The cached representation of the composite operations aims at boosting the performance of the EML XML manager, so that is scales to the numerous management requests. The cache is constructed during the EML system’s initialization and maintained as long as the element is active.

- **XML Management Engine**: The XML management engine is the heart of the system. It receives management commands from applications (e.g., a manager) in the form of XML documents, through a distributed mechanism like XML-RPC. Upon receiving such a command, it parses it, resolves the composite operations from the cache, identifies and performs the necessary atomic operations on the device. Thus, the XML management engine undertakes to analyze a composite operation into atomic ones, and determines (XML-MIB) objects engaged in the atomic operations. Following this
determination, it invokes the appropriate SNMP Get/Set operations on the devices using the XML/SNMP gateway. The result of the operations follows the reverse direction: the atomic operations results are conveyed from the XML/SNMP gateway to the XML engine, which assembles and delivers them to the management application in the form of an XML document. The structure of the XML document to be returned is specified as part of the element’s management API.

- **Abstract Information Model:** In order to support vendor independent management an abstract XML based information model should be implemented. In this case the XML Management Engine maps the atomic operation to a particular vendor independent MIB object (i.e. an object from the abstract information model) and conveys it to the particular vendor’s devices through a vendor adapter module. The Abstract Information Model could well be based in DMTF’s CIM (Common Information Model) and the related DTD. Several mechanisms for developing vendor independent abstract information models, as well as related implementation techniques exist (see for example [14]).

- **Vendor Adapter:** The role of this module is to map a particular object of the Abstract Information Model (i.e. the vendor independent MIB) to a vendor dependent MIB object. Accordingly, the vendor adapter communicates with the XML/SNMP gateway for actually performing the required Get/Set operation on the device.

- **Rendering System:** A cocoon based rendering system making use of XSL technology can interface with the EML Engine to allow for information presentation and visualization at the device level.

- **SOAP Engine / Web Service Gateway:** These modules allow the XML based interface of the node/device to be exported as a set of web services. In particular, all composite operations offered by the node are exported as Web Services. In this case, clients of the EML management interface (i.e. managers and higher level management systems) communicate directly with the XML management engine, through invoking the appropriate composite operations that are available through SOAP/HTTP. This obviates the need for developing the XML parser and caching modules, at a cost of the additional web services infrastructure.

- **XML/SNMP gateway:** This gateway provides the means to access the low-level management capabilities (i.e. the SNMP agent of the device) upon receiving (in XML format) the set of atomic operations than need to be executed on the device. The XML/SNMP gateway is responsible for initiating the necessary SNMP Get/Set/GetNext operations on the device, and accordingly to collect and deliver the results as XML documents. These XML documents are structured based on an (SMI based) DTD, or XML Schema, depending on the gateway’s structure and functionality.

An alternative design (depicted in Figure IV) could rely on Xpath/Xquery mechanism towards resolving the composite operations contained in the XML_API_Call document sent by the manager, instead of the Cache mechanism outlined above. Specifically, in Figure III the XML engine receives the document sent through the distributed mechanism, identifies the composite operations, and passes a request to the Xpath/Xquery module. The latter resolves the composite request into atomic management operations by accessing the EML XML document and returns those atomic operations to the XML engine in order to be passed to the XML/SNMP gateway. As a result, the Cache database is no longer required. The selection between XML/Xquery and caching should be based on performance evaluation and implementation complexity criteria pertaining to the target management system.
In both of the above designs there is a possibility that the vendor adapter and abstract information model is not present. This can be the case when vendor independent management is not targeted. From a practical viewpoint, the XML EML engine is a piece of network management software that can be hosted in a management workstation attached to the device. It can be instantiated several times with different configuration information towards controlling multiple devices. Note that managing a newly added device, requires the instantiation and initialization of an EML engine for the target device.

The above functionality emphasizes on replacing SNMP Get/Set operations with XML based operations. Trap operations are not straightforward in this design and should be handled through separate information flows delivering XML documents to the rendering subsystem. A modification is however required to the EML engine design given that most of the existing XML/SNMP gateway implementations do not support SNMP trap handling. To overcome this limitation an autonomous process could be configured to listen for the node’s trap-sending requests. This process would retrieve the context of each trap message, form an XML message of a standard type, and finally send it to the rendering engine. Thus, traps could be handled following the same paradigm as other API requests.

The EML engine implementation effort can be kept to a minimum when existing components (e.g., the XML/SNMP gateway) are reused. Other implementations could make use of other mechanisms (e.g., Java Dynamic Management Kit (JDMK)) to access the low level capabilities of the target device. In terms of implementation complexity the XML management engine is certainly the EML module requiring the most significant effort.

4. Network Management Layer Engine
Moving at a higher programmability layer the NML engine and respective API can be used to implement management applications targeting the state of the art multi-vendor and heterogeneous networking environments. The network management engine makes use of the EML XML engine and related APIs to interface to individual network elements. Based on several EML systems, the NML engine can carry out operations involving multiple devices and demanding combination of multiple element layer operations. From an architectural perspective the NML engine reveals many similarities to the EML engine described
in the previous paragraph. Figure V depicts the main building blocks of the NML system, while Listing II provides a simplistic invocation of an NML operation, utilizing the EML operation (request) included in Listing I.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<network-management-scheme>
    <name>ExampleNMScheme</name>
    <description>Example NM Scheme</description>
    <nm-functions>
        <func name="Create_End2End_VPC_Connection" priority="1">
            <description>.........</description>
            <process>
                <operation name="Create_PVC_1" eml-operation="Create_VPC" node="atm1">
                    <param name="channelValidateStatus" type="int">NetworkDefaultValue</param>
                    <param name="channelVPIOut" type="int">NetworkDefaultValue</param>
                    <param name="channelVPIIn" type="int">NetworkDefaultValue</param>
                    <param name="channelName" type="String">NetworkDefaultValue</param>
                    <param name="channelUpc" type="int">NetworkDefaultValue</param>
                </operation>
                <!-- More EML Operations... -->
                <operation name="Create_PVC_2" eml-operation="Create_VPC" node="atm2">
                    <param name="channelValidateStatus" type="String">NetworkDefaultValue</param>
                    <param name="channelVPIOut" type="String">NetworkDefaultValue</param>
                    <param name="channelVPIIn" type="String">NetworkDefaultValue</param>
                    <param name="channelName" type="String">NetworkDefaultValue</param>
                    <param name="channelUpc" type="String">NetworkDefaultValue</param>
                </operation>
                <!-- More NML functions -->
            </process>
        </func>
    </nm-functions>
</network-management-scheme>
```

Listing II: XML based NML Operation

This architecture reveals several similarities to the EML architecture. The NML Engine exposes an XML API that can be invoked based on a distributed mechanism (i.e., XML – RPC, Web Services). This API contains NML operations consisting of several EML operations. Each one of the element level operations
can be executed as illustrated in the scope of the EML XML engine on the various devices of the network. Therefore, the role of the NML engine can be summarized in the following tasks:

- Parsing the XML document corresponding to the API call and resolving it to element level operations. This includes the task of identifying the particular device targeted in each one of the EML operations, as well as extracting the parameters of the operation. A module based on Xpath/Xquery can perform these tasks though locating the appropriate element level functions based on the DTD or schema defining the API. Alternatively, element level operations could be cached in a repository to achieve improved performance during XML_API_Call documents processing. The two options are similar to the respective ones presented in the scope of the EML description.

- Delegating the element level operations to the appropriate EML XML systems. This entails the task of constructing the XML documents that conform to the API specifications of the EML systems. This is a task performed by the XML Management Engine module. Note also that the NML acts as a manager for the EML XML engine. The NML consumes the XML results of the EML API. As far as the remote interface is concerned this is based on the type of interface exported by the EML system. In addition the XML management engine undertakes to collect the XML documents resulting from the element level operations and accordingly to assemble the XML result, as the latter is specified in the NML API. The same mechanism could be also used to deal with combinations of SNMP trap documents on the individual devices.

The above operations presuppose that the XML Management engine and the Xpath/Xquery module are fully aware of the DTDs or Schemas defining the NML & EML API. Based on these definitions they can perform all the necessary processing operations. Note also that the DTDs or Schemas defining the structure of the API documents could contain a reasonable amount of metadata to allow the XML parsing and processing modules to discover management operations and how they translate to MIB or other objects understandable by the XML/SNMP gateway. Defining the format and structure of these schemas to allow programmability of network management based on XML APIs, could well become a subject of standardization activity for the network management research community, as well as a subject of programmable networks research.

The XML schemas or DTDs defining the XML_API_Calls at both the NML and EML levels allow network managers to produce sophisticated XML based management applications. These specifications could be available within a proper network management repository or directory (e.g., LDAP or COPS based).

5. Prototype Implementation

A prototype implementation of various subsystems of this architecture has been carried out to validate the core concepts of the framework. This prototype instantiation includes a complete implementation of the EML engine and a partial implementation of the NML subsystem. The EML system implementation follows the paradigm with the cache database, mainly for performance reasons. The implementation is almost entirely Java based, the only exception being some modules of the XML/SNMP gateway. Upon initialization of a target device a Java SAX parser module reads an XML document signifying the Composite Operations allowed on the device, as well as how these map to atomic operations. Moreover, global EML information (e.g., the IP address required for managing the node, vendor, model, SNMP community words) is also supplied in this XML document. Composite operations are registered within a cache repository, which can reside either in the same or even in a different machine from that hosting the EML engine software. In the latter case the cache repository is accessed through RMI calls.

The XML management engine exports a (Java based) XML-RPC interface and is capable of processing XML_API_Call based on Java SAX processing mechanisms. The XML operations are resolved in the RMI cache and accordingly mapped to atomic operations. Atomic operations are in the sequel mapped to the appropriate vendor specific MIB object, corresponding to the XML vendor independent representation of this particular network device. Atomic operations are executed based on HTTP calls on the XML/SNMP gateway from F.Strauss and Torsten Klie [2], which relies on mibdump. Using the XML/SNMP gateway minimized significantly our implementation effort given that this module comprises the usually tedious (see for example [15]), low-level part of the interfacing with network devices.
The EML engine is complete and can operate on top of every node. Network managers can specify and implement EML management operations merely by authoring XML files conformant to our proprietary XML Schema defining the structure of the EML API. The particular EML operations have been tested through XML APIs targeting operations on terminal nodes (i.e., workstations) and ATM switching nodes.

As far as network level management is concerned a proof of concept implementation has been carried out. This includes a simple XML schema for the NML APIs, allowing EML operations to be combined. Based on this schema, XML files have been authored and used for more sophisticated management operations.

A simple rendering system has been implemented as well. This utilizes Cocoon to display configuration management information in a web-based manner. Simple XSL files are used to transform the XML files resulting from the specified configuration management operations.

6. Conclusions

This paper presents recent advances regarding XML based technologies for network management and illustrated their momentum. It also introduces an architecture for programmable XML based network management applications. This architecture exploits XML technologies, as well as recent developments in XML based management in order to specify XML APIs for managing individual devices but also composite networks. Based on this architecture network managers can produce rather sophisticated applications through a minimal effort (i.e. authoring XML & XSLT documents).

This architecture constitutes a first class manifestation of the potential benefits of XML technologies for network management, particularly for configuration management. This is evident in the prototype implementation. Among our imminent priorities is to extend this implementation with a view to capturing all conceptual aspects analyzed in the paper. Also, particular emphasis will be put on supporting web services, which will allow for standards based distributed integration with other management system. Moreover, we intend to investigate the scalability of our programmability solution through enhancing and testing it in more complex environments. By and large, we strongly believe that the proposed framework can lead to effective solutions for small to medium scale campus networks, including Web Based Enterprise Management solutions. It is also important that this framework could serve as placeholder hosting a variety of information models, management protocols and distributed web based management paradigms.

Acknowledgements

The authors gracefully acknowledge Frank Strauß and Torsten Klie for providing the source code of the XML/SNMP gateway and the mibdump module, to allow experimentation in our labs and development of a working prototype of our XML management system.

References