A Multi-protocol architecture for SNMP entities

Rui Pedro Lopes
Instituto Politécnico de Bragança, ESTiG, 5300 Bragança, Portugal
Tel.: +351 273 303109
rlopes@ipb.pt

José Luis Oliveira
Universidade de Aveiro, DET, 3810 Aveiro, Portugal
Tel.: +351 234 370523
jlo@det.ua.pt

Abstract – **Current network management requirements impose the omnipresence of technicians and tools. With the advent of UMTS, wireless networks are moving fast to the IP world, opening an unpredictable mix of users and services.**

This paper presents an architecture that can help to spread the typical centralized network management console along a multitude of terminals, namely handheld devices, capable of monitoring and controlling SNMP agents.

I. INTRODUCTION

The dependency of enterprises and business activities upon Intranets and the Internet imposes that the communication services are available and well performing all the time. To avoid and to repair unexpected failures, special procedures must be performed in the more expedited way and with the minimal impact on users' services and on business agreements. On the other hand, automation and reactivity are not, unfortunately, common characteristics in current network management systems. This kind of actions is typically restricted to minor issues such as notification delivering, triggered by some predefined conditions. As a result, the management staff must be permanently available to consult and to configure network infrastructure operational parameters whenever necessary.

While this circumstance do not affect too much most of the nowadays small and mid-size LANs due to typically over-dimensioning strategies, network management will become increasingly complex as massive quantities of very diverse elements, ranging from resource-limited devices (palmtops, mobile phones, etc.) to large-scale distributed applications, become widespread across different domains in the Internet. This can be quite far from basic Internet management, until recently guided by the simplicity and minimalism that originated the very successful SNMP framework. Despite its on-going evolutions (the most recent being the SNMPv3), it still lacks scalability due to the inherent broad range of management information and to polling based operations.

An obstacle to ubiquitous management is the traditional dependency on the central workstation. In fact, it is not easy to have an updated view of the network state if the user is somehow apart from the management station. Ideally, it should be possible to consult network parameters with a common Web browser or with a cellular phone. At the limit, we can even base the management interface on any telecommunication terminal using DTMF for signalling or, using a newer and upcoming paradigm, a VXML-based voice synthesis/recognition to interact with the management system.

Considering management distribution (or delegation, using a well-know terminology in the area [1]), the IETF chartered the Distributed Management (DISMAN) working group [2] to define a set of standards that integrates distributing management functionality into the Internet management framework. Its basic concepts define a standalone manager that, by the way of pre-defined policy definitions, can record management information and respond to state changes even in the absence of user interaction. By definition, these elements do not need user interface. The user interface, as well as higher-level managers, communicates with these elements through SNMP whenever necessary. A pleasant side effect of this separation is that it allows sharing management functionality among different user interfaces thus providing better adjustment to user requisites and particular situations.

This paper describes the development, implementation and practical evaluation of a multi-protocol agent that can have both manager and agent role according following the main concepts of the DISMAN model. This work has supported and have enrich the development of the *Schedule*, the *Event*, and the *Expression MIBs* [3-5] but can be applied in a simple SNMP agent where we want to put extra features besides its SNMP conformance.

II. AN ARCHITECTURE FOR DISTRIBUTED MANAGEMENT

Management distribution allows reducing the processing load on traditional centralized management station (NMS) by delegating tasks upon several Distributed Managers (DM) or upon more powerful agents. A DM is an SNMP entity that receives requests from another manager and executes those requests by performing management operations on agents or other managers.

Since the management entities are split over the network, a hierarchy of several interacting "islands" is created, increasing the robustness and fault tolerance of the overall management system. Each DM may handle locally critical situations when the access to the central manager is not possible.

The DISMAN model implies that a minimum but essential set of management functions is incorporated in each DM [2][6]. Basically, the distributed manager must autonomously react to specific conditions that are obtained through the polling of other SNMP agents. Modifying network parameters or generating special notifications are two kinds of expected reactions. Each manager is build upon a selection from a set of services, which have been defined through specific MIB interfaces.

Our main goal for this work is the achievement of better mechanisms for handling the drawbacks of a centralized SNMP-based management [7][8]. This approach must solve, on one side, the restricted access to network management console by diversifying the access along a multitude of terminals, namely handheld devices, and, on the other, the dependence of management operations (SNMP requests) on a central management system – through the adoption of distributed management mechanisms.

The result was a multi-tiered model that includes: the user access interface, the middleware and the agent role (Figure 1). A broader set of protocols can be used both on the manager and on the agent side. For instance, if web-oriented interfaces are a must, the system can benefit from using the

HTTP protocol. To maintain compatibility with traditional network management stations any SNMP version can be used.

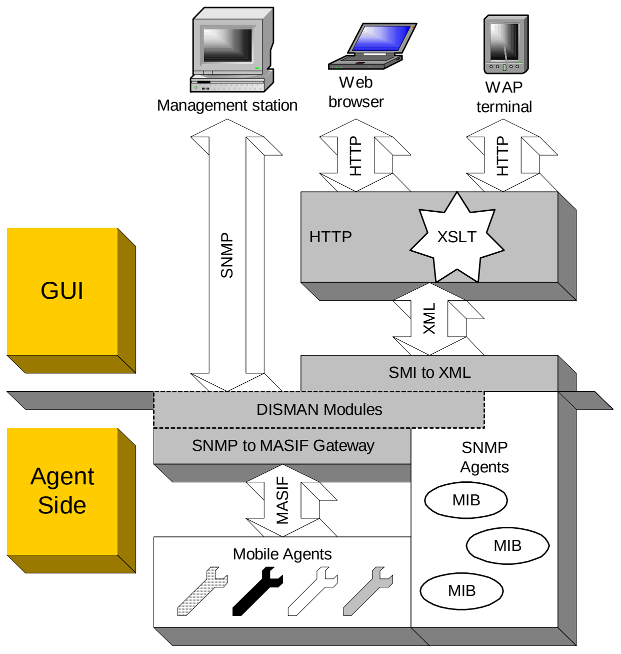


Figure 1 – Architecture for distributed management.

Essential management operations are moved to the agent side through DISMAN modules. This allows sensing network working parameters and taking associated actions, as well as the possibility to initiate operations on a schedule basis.

The user can gain access to management infrastructure from a diverse set of terminal. For instance, using a common WAP phone it is possible to monitor systems and network states or activate any management procedure, remotely. In this case it is necessary to follow through HTTP, XSL transformation, XML, SNMP.

The shaded blocks constitute the parts of the architecture that were developed. The others were used from network equipment or from open source products.

A. Agent API

The implementation and maintenance of MIB modules, namely the *Schedule*, *Event* and *Expression MIBs*, can put several problems such as how to deal with persistency, with management tasks definition and with the SNMP configuration. In order to separate instrumentation procedures (MIB side) from agent services facilities (protocol side) we have build a package which we call *AgentAPI* and that permits a simple implementation of common SNMP agents[1]. The agent is built through the specialization of the base classes.

The *AgentAPI* uses a binary tree structure to store references to SNMP managed objects. This structure has higher efficiency for “walk” operations than arrays or hash tables, due to the OID based ordering scheme. The SNMP objects may be specializations of three kinds of classes: *simple objects*, *SNMP tables* or *SNMP conceptual tables*. Each object inherits the same set of operations from the SNMP related classes and introduces a new set, therefore providing the specialization required by the MIB (Figure 2).

III. USING XML TO DEFINE COMPLEX SNMP OPERATIONS

SNMP MIBs presents, normally, a useful set of functionalities to the user but using strictly SNMP to perform remote configuration can be an unpleasant task. Also, DISMAN implies building agents that can cope with rather complex information structures. In the case of information loss, for example, due to an agent reset, it is not practicable to force the management station to define and configure the information once again. Clearly, any solution produced at the interface level to solve these problems can also benefit the operation and management of the whole Internet management framework.

Inside the IETF there is yet some expectation around XML, namely if it will be adequate to replace SMI definitions [9]. There are also several proposals to use XML in management frameworks [10,11].

In this section we present an XML-based data model that simplifies the definition and enforcement of complex transactions, and provides a structure for distributed managers persistency.

SNMP defines several kinds of commands and associated PDUs with different number of parameters (*variable bindings*). Moreover, a management operation is typically built upon several SNMP commands. However, the normalized SMI guidelines merely describe structures and schemes for the definition of management information. The specification of concrete SNMP messages has been left out of SNMP recommendations.

The XML allows describing structured information by defining specific tags. This tag arrangement builds a document that can be used to exchange information independently on the platform, programming languages and applications. These characteristics make it ideal to define and to exchange SMI information, and, as well, to describe SNMP commands, tasks, or management operations.

SNMP commands are described as XML tags, as well as the associated parameters (variable bindings, destination, port, …). The tag <task> joins logical sequences of commands, while other tags provide facilities to reuse code (<property>), load MIB modules (<mib>) and execute tasks defined on different files. Figure 3 shows a simple example of a task definition.

There are several scenarios where this approach can be used. Management applications may use it to automate complex series of operations by grouping them together within a single task. Instead of manually performing sequences of time-consuming and repetitive actions, it is

[1] Source code and samples are freely available in http://nms.estig.ipb.pt/

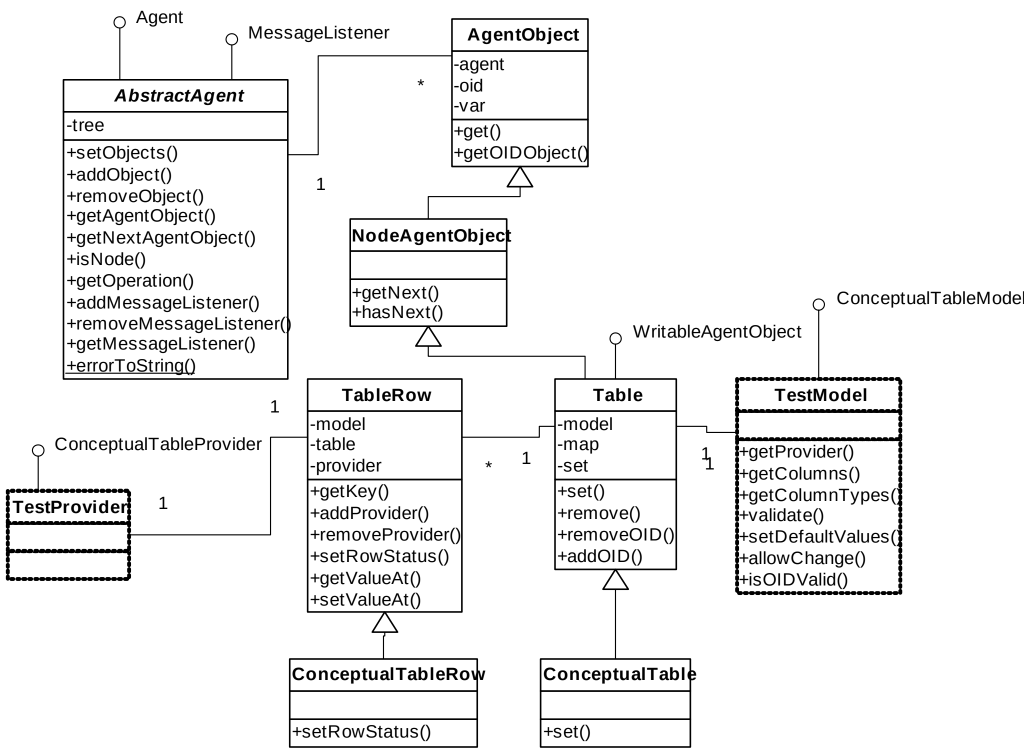


Figure 2 – Agent API class diagram.

possible to create and run a single macro (a custom command) that accomplishes the task.

```
<snmp version="3" user="rp" authProtocol="MD5"
 authPassword="adm" privPassword="senior">
 <mib name="RFC1213-MIB" location="file:/usr/local/mibs"/>

 <property name="trapObject" value=".1.3.6.1.2.4.5.6"/>
 <property name="otherObject" value=".1.3.6.1.2.4.5.7"/>
 <property name="destination" value="192.168.168.168:162"/>

 <task name="myTrap">
   <trap version="2" destination="$destination">
     <!—implicit VarBinds
     <varBind name="sysUpTime" oid=".1.3.6.1.2.1.1.3.0"/>
     <varBind name="snmpTrapOID" oid=".1.3.6.1.6.3.1.1.4.1"/>
     -->
     <varBind name="someOID" oid="$trapObject"/>
   </trap>
   <trap version="2" destination="$destination"
         name="someOID" oid="$otherObject"/>
   <runTask name="someTask"
         document="file:/usr/local/operation/someOp.xml"/>
 </task>
</snmp>
```

Figure 3 – SNMP macro example.

Also a powerful application is in assurance of data persistence. SNMP agents may have a huge number of stored values and some of them may be consequence of set-request operations issued by the manager. The *Event MIB*, for example, has several tables that are configured by the manager to define trigger conditions, events, notifications and actions. If the agent has a failure, for some reason, these values are lost. In this case, if the agent does not have some persistency mechanism the manager must issue them once again.

MIBs usually do not have explicit control of storage. This capability can be provided in an ad-hoc way and it may be very different according to the agent manufacturer and implementation details (such as the programming language employed). XML can be used to explicitly control how the objects must be created after some agent breakdown. Moreover, it allows agents to replicate data to other agents, which is an important characteristic for management delegation and for management fault tolerance. Although, if this is not desirable on the agent side, the XML definition can be used in the manager side to store configurations in a simple and human readable format. In [12] we have propose the use of a similar schema to describe management scripts for mobile agents.

SNMP macros definition can be simplified by using a GUI tool, so that the user will not be required to know all the tags and parameters. We developed an application based on Java Beans where any task can be modelled as a Bean and used as easily as we use a geometric component in an normal graphical editor (Figure 5).

IV. XML EXCHANGE OF SMI DEFINITIONS WITH XSL TRANSFORMATION

This section presents the design and implementation of a HTTP server that uses XML to describe management data in a SMI-like style and that makes use of XSL transformation to adapt the output interface to different kind of terminals. The system can be employed in two distinct situations: 1) as an embedded communication module to enhance an SNMP agent, namely a complex one such as a DM, or 2) as an HTTP-to-SNMP gateway, acting as a proxy for handling several SNMP agents.

A. The multi-protocol agent

The main idea was to include new interfaces in SNMP agents in order to allow a wider range of points of management. Naturally this strategy implies larger agents. However, this price is acceptable when dealing with complex agents where the overcharge is negligible, and when it is important to have ubiquitous access to agents. The proposed model separates the SNMP agents in two layers (*Message Adapter* and *Agent Information*) in order to allow different protocols and information structures (Figure 4). These two blocks represent the core of the *Agent API* briefly presented in II.

The *Agent Information* module contains the instrumentation part of the agent, which follows the SMI description of required MIBs. This information store defines how protocol commands are mapped to platform operations, by modifying or retrieving MIB parameters.

The *Message Adapter* consists of an adaptation layer that allows using different protocol stacks to access the same instrumentation information. To achieve this, the *Message Adapter* registers itself as a “protocol listener”, in each communication module (HTTP engine or SNMP stack, for example).

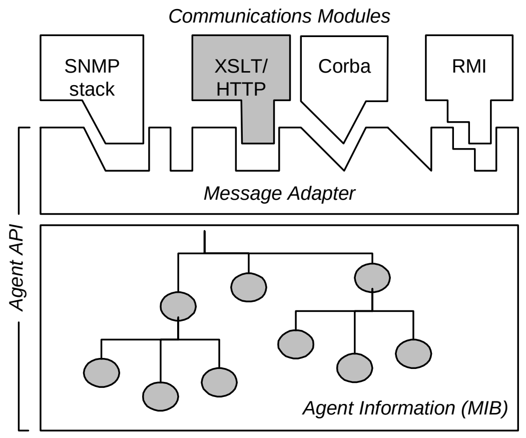
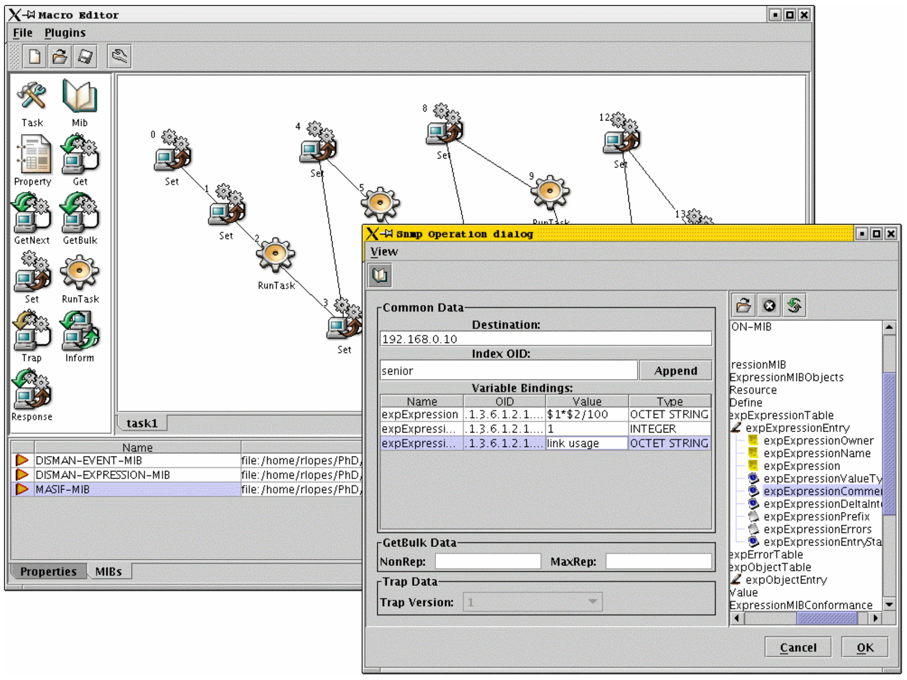


Figure 4 – A multi-protocol management agent.

Finally, the *Communication Modules* are created according to user requirements. For example, an agent using exclusively SNMP will only need the SNMP stack (this is the normal SNMP agent). Other agents may require other protocols, such as HTTP, SSL, RMI, CORBA and so on. In order to check the performance of different protocols for handling management information we have built RMI, CORBA and HTTP engines. However this study is out of the scope of this article so we just concentrate on the HTTP engine and the usage of XSL for granting a generalized and simplified access.

B. Proxy agent

By replacing the *Agent Information* module with an SNMP stack we have an SNMP proxy agent or, according to the



	Name	
▶	DISMAN-EVENT-MIB	file:/home/rlopes/PhD/
▶	DISMAN-EXPRESSION-MIB	file:/home/rlopes/PhD/
▶	MASIF-MIB	file:/home/rlopes/PhD/

Common Data

Destination: 192.168.0.10

Index OID: senior Append

Variable Bindings:

Name	OID	Value	Type
expExpression	.1.3.6.1.2.1....	$1*$2/100	OCTET STRING
expExpressi...	.1.3.6.1.2.1....	1	INTEGER
expExpressi...	.1.3.6.1.2.1....	link usage	OCTET STRING

GetBulk Data

NonRep: MaxRep:

Trap Data

Trap Version: 1

```
ON-MIB

ressionMIB
ExpressionMIBObjects
Resource
Define
expExpressionTable
  expExpressionEntry
    expExpressionOwner
    expExpressionName
    expExpression
    expExpressionValueTy
    expExpressionCommen
    expExpressionDeltaInt
    expExpressionPrefix
    expExpressionErrors
    expExpressionEntrySta
expErrorTable
expObjectTable
  expObjectEntry
Value
ExpressionMIBConformance
```

Figure 5 – SNMP Macro Editor.

supported communication adapters, a *proxy-to-anywhere*. For instance when using an HTTP engine with the XSLT processor we provide an HTTP to SNMP gateway.

The gateway provides a Web view to several SNMP agents. For the user, the main difference of this approach is on the access control interface. In the embedded model, the user authentication is granted through the simple login/password method. In this gateway model, several agents can be accessed through a single HTTP engine. Thus it is necessary to choose the target agent address and after that we can use the same XML code.

C. *The XML Adapter*

We have build an XML *Communication Module* where the XML definition is complemented with a XSLT post-processor that dynamically generates management views in a format that is the best suited for the client's interface.

The *Communication Module* feeds on two sources: SMI files for the agent MIB structure and the agent information for the values. This joined information is converted to XML by the *XML Generator* and is then forwarded to the *XSLT Processor*. The XSL sheets provide the guidelines for transforming the common XML input to different output documents, such as HTML, WML or VXML. The result is then sent to the embedded HTTP server.

The reason for having this module feeding on both SMI and agent values is that SMI, although defining managed objects, does not represent agent values, corresponding to those objects. For the current architecture we have included a special tag, <value>, so that the user can have access not only to the information structure but also to the attribute value. The XML generator builds a document based on SMI definition and augments the structure with the data from the local SNMP agent.

When the user accesses the HTTP engine, a login page shows up[2]. This provides a minimum level of security through user authentication. After being successfully authenticated, the user can monitor and control (through HTML or WML pages) the state of the agent. We choose to use a pair <host/user> as the login name. This approach maintains unchanged the XML code as well as the XSL documents and allows telling the gateway which is the agent under query.

With this approach, through the aid of a current HTTP browser or WAP device, the user can "navigate" over the agent's SMI, minimizing delays. The output representation will be different according to the type of object:

- a node (XML <node> tag) is presented as a link to allow the user to select the next page.
- a scalar (XML <scalar> tag) is represented by its current value.
- a table (<table> tag) is represented in a tabular form.

Moreover, depending on the access policy for SMI objects (<access> tag), the XSLT should only output a value for read-only objects and create forms for read-create and read-write objects. The modification of selected parameters is performed by HTTP *post* operations and SNMP *set* commands.

Another aspect associated with the XSLT transformation output is that for a small screen terminal, such as cellular phones, the page should show only essential information, such as node label, OID and value, to save screen space. For conventional Web browsers, the screen is larger and permits more information to be displayed, such as SMI object descriptions and access privileges. The differentiation is achieved through the selection of the adequate XSL file.

The presented architecture was implemented using the Java language with several public domain tools and utilities. For the HTTP engine we used the embeddable web server Jetty [13], which includes also an HTTPS engine for higher security. The Apache Foundation contributes with two modules: the XML parser Xerces [14] and the XSLT processor Xalan [15]. The SNMP stack (SNMPv3) is from AdventNet [16].

V. CONCLUSIONS

Management paradigms are largely based on centralized solutions. This problem occurs, for instance, within the user interface that is typically too much dependent on a single technology. Other examples can be provided by SNMP polling strategy and by the absence of distributed management in SNMP administrative domains.

At the lower level of the network management models, the distribution of management operations can be simplified by the implementation and wide use of new solutions such as the one provided by *disman* working group inside the IETF. Looking at the other edge of the management framework, at the user interface level, distribution can also be performed by providing access from anywhere, from anyplace, from any terminal.

We have presented several components for distributed and ubiquitous management. The first one was an Agent API that simplify the SNMP agent development burden and can easily accommodate other access mechanisms such as RMI, CORBA or HTTP. The second consists of an XML-based formalism that can replace SMI in the definition of management information and simultaneously retains data values within each MIB structure. This formalism and a component-based GUI were also proposed for the definition of complex management configuration tasks..

VI. REFERENCES

[1] Y. Yemini, G. Goldszmidt, S. Yemini, "Network Management by Delegation", in *Proc. IFIP/IEEE Int. Symp. on Integrated Network Managemen*t, Apr. 1991.

[2] Distributed Management (DISMAN) Charter, (http://www.ietf.org/html.charters/disman-charter.html).

[3] D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for Scheduling Management Operations", RFC2591, May 1999.

[4] R. Kavasseri, B. Stewart, "Event MIB", RFC2981, Oct. 2000.

[5] R. Kavasseri, B. Stewart, "Distributed Management Expression MIB", RFC2982, Oct. 2000.

[2] from any *wapalizer*: http://nms.estig.ipb.pt/see/disman?wml=true

[6] J. Schönwälder, “Network Management by Delegation – From research prototypes towards standards”, in *Computer Network and ISDN Systems*, Vol. 29, Nov. 1997, pp. 1843-1852.

[7] G. Goldszmidt, Y. Yemini, “Delegated Agents for Network Management”, *IEEE Communications Magazine*, Vol. 36 No. 3 (1998) pp. 66-71.

[8] R. Sprenkels, J-P Martin-Flatin, “Bulk Transfer of MIB Data”, *The Simple Times*, Vol. 7, No. 1, March 1999.

[9] Next Generation Structure of Management Information (sming) (http://www.ietf.org/html.charters/sming-charter.html).

[10] C. Ensel, A. Keller, “Managing Application Service Dependencies with XML and the Resource Description Framework”, in *Proc. IFIP/IEEE Int. Symp. on Integrated Management 2001 – IM2001*, May 2001.

[11] D. Lewis, J. Mouritzsen, “The Role o XML in TMN Evolution” in *Proc. IFIP/IEEE Int. Symp. on Integrated Management 2001 – IM2001*, May 2001.

[12] R. Lopes, J. Oliveira, “SNMP Management of MASIF Platforms”, in *Proc. IFIP/IEEE Int. Symp. on Integrated Management 2001 – IM2001*, May 2001.

[13] Jetty Java HTTP Servlet Server (http://jetty.mortbay.com/).

[14] The XML parser Xerces (http://xml.apache.org/xerces-j/index.html).

[15] The XSLT processor Xalan (http://xml.apache.org/xalan-j/index.html).

[16] AdventNet SNMP stack (http://www.adventnet.com/).