P

A Web Services based Framework for Efficient Monitoring and Event Reporting

Aimilios P. Chourmouziadis

Submitted for the Degree of
Doctor of Philosophy
from the
University of Surrey

Centre for Communication Systems Research
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey GU2 7XH, UK

September 2008

Summary

Network and Service Management (NSM) is a research discipline with significant research contributions the last 25 years. Despite the numerous standardised solutions that have been proposed for NSM, the quest for an "all encompassing technology" [1] still continues.

A new technology introduced lately to address NSM problems is Web Services (WS). Despite the research effort put into WS and their potential for addressing NSM objectives, there are efficiency, interoperability, etc issues that need to be solved before using WS for NSM.

This thesis looks at two techniques to increase the efficiency of WS management applications so that the latter can be used for efficient monitoring and event reporting. The first is a query tool we built that can be used for efficient retrieval of management state data close to the devices where they are hosted. The second technique is policies used to delegate a number of tasks from a manager to an agent to make WS-based event reporting systems more efficient.

We tested the performance of these mechanisms by incorporating them in a custom monitoring and event reporting framework and supporting systems we have built, against other similar mechanisms (XPath) that have been proposed for the same tasks, as well as previous technologies such as SNMP. Through these tests we have shown that these mechanisms are capable of allowing us to use WS efficiently in various monitoring and event reporting scenarios.

Having shown the potential of our techniques we also present the design and implementation challenges for building a GUI tool to support and enhance the above systems with extra capabilities.

In summary, we expect that other problems WS face will be solved in the near future, making WS a capable platform for it to be used for NSM.

Key words: Network and Service Management, Web Services, Simple Network Management Protocol, eXtensible Markup language, XML Path language, Monitoring, Event Reporting, Policy, Efficient.

Acknowledgments

First of all, I would like to thank my supervisor, Professor George Pavlou, without whom I would not have been able to carry out this PhD research project, for his guidance, efforts and patience during his supervision since I joined the Centre for Communication Systems Research (CCSR) in University of Surrey.

I would also like to thank my colleagues in CCSR and in particular Stelios Georgoulas, Stelios Gouveris, Oscar Gonzalez Duque and Marinos Charalambides for their valuable help in my research.

I would also like to thank my family in Greece, my father Pelagios Chourmouziadis and my mother Anastasia Chourmouziadis for their support and guidance during this period of time.

Finally, I would like to thank all the anonymous reviewers of my submitted papers for their useful comments and suggestions.

Contents

List of Figures

Glossary of Terms

AM	Accounting Management
API	Application Programming Interface
ASN.1	Abstract Syntax Notation 1
BEEP	Blocks Extensible Exchange Protocol
BNF	Backus Naur Form
BPEL4WS	Business Process Execution Language For Web Services
CIM	Common Information Model
CLI	Command Line Interface
CMIP	Common Management Information Protocol
COPS-PR	Common Open Policy Service for Policy Provisioning
CORBA	Common Object Request Broker Architecture
CM	Configuration Management
CMOT	CMIS/P over TCP/IP
DCE	Distributed Computing Environment
DiffServ	Differentiated Services
DII	Dynamic Invocation Interface
DISMAN	Distributed Management
DME	Distributed Management Environment
DMTF	Distributed Management Task Force
DOM	Document Object Model
Dscp	Distributed code point
DTD	Document Type Definitions
EPR	EndPoint Reference
Expression MIB	Distributed Management Expression MIB
FD	Filtering Data
FEC	Forwarding Equivalence Class
FEC-to-NHLFE	Forwarding Equivalence Class to Next Hop Label Forwarding Entry
FM	Fault Management
FTP	File Transfer Protocol
GDMO	Guidelines for the Definition of Managed Objects
GIOP	General Inter-Operability Protocol
GUI	Graphical User Interface

HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IIOP	Internet Inter-Operability Protocol
IOR	Interoperable Object Reference
IP	Internet Protocol
ISO	International Organisation for Standardisation
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JIDM	Joint Inter Domain Management
LSP	Label Switched Path
LSR	Label Switching Router
MEP	Message Exchange Pattern
MIB	Management Information Base
MID	Multiple Instance Data
MIH	Message Information Header
MIM	Management Information Model
MO	Managed Object
MOWS	Management Of Web Services
MPLS	MultiProtocol Label Switching
MUWS	Management Using Web Services
NetConf	Network Configuration protocol
NSM	Network and Service Management
OASIS	Organisation for the Advancement of Structured Information Standards
ODP	Open Distributed Processing
OGSI	Open Grid Services Infrastructure
OID	Object Identifier
OMG	Object Management Group
ORB	Object Request Broker
OSF	Open Software Foundation
OSI-SM	Open Systems Interconnection System Management
PHB	Per Hop Behaviour
PIB	Policy Information Base
POA	Portable Object Adapter
PM	Performance Management
QoS	Quality of Service
QRBT	Query Relationships By Type
QRP	Query Resource Properties

RP	Resource Property/Properties
RFC	Request For Comments
RPC	Remote Procedure Call
SAX	Simple API for XML processing
Script MIB	Definitions of Managed Objects for the Delegation of Management Scripts
SID	Single Instance Data
SLA	Service Level Agreement
SLS	Service Level Specification
SM	Security Management
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SS	Service Selection
SSH	Secure Shell
TE	Traffic Engineering
TCP	Transmission Control Protocol
TMN	Telecommunications Management Network
UDDI	Universal Discovery Description and Integration
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WBEM	Web Based Enterprise Management
WS	Web Service(s)
WSA	Web Services Architecture
WS-BF	Web Service Base Faults
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WS-I	Web Services Interoperability
WSLA	WS-LevelAgreement
WS-Management	Web Services for Management
WS-MEX	Web Services Metadata Exchange
WS-Notification	Web Services Base Notification
WS-Reliability	Web Services Reliable Messaging
WSRF	Web Services Resource Framework
WS-RL	Web Services Resource Lifetime
WS-RP	Web Services Resource Properties

WS-RT	Web Services Resource Transfer
WS-SG	Web Services Service Group
XML	eXtensible Markup Language
XML Schema	eXtensible Markup Language Schema
XPath	eXtensible Markup Language Path Language

List of Publications

Journals

[1] A. Chourmouziadis, M. Charalambides, G. Pavlou, "On the Performance and Scalability of Web Services for Monitoring MPLS-based Networks", work accepted for publication in the Journal of Network and Systems Management (JNSM).

Conferences

[2] A. Chourmouziadis, G. Pavlou, "Efficient Information Retrieval in Network Management Using Web Services", DSOM 2006 Proceedings, October 2006, Volume 4269/2006 pp. 1-12.

[3] A. Chourmouziadis, G. Pavlou, O.Gonzalez-Duque, "Using MUWS for scalable distributed Network Monitoring of the state of Manageable Resources, work submitted to IM 2009.

[4] A.Chourmouziadis, G. Pavlou, "An Evaluation of WS Tools to Address Efficient Information Retrieval for Network Management", Proceedings of the 8th International Symposium on Computer Networks (ISCN'2008), Istanbul, Turkey, June 2008.

[5] A. Chourmouziadis, G. Pavlou, "Web Services Monitoring: An Initial Case Study on the Tools Perspective", Network Operations and Management Symposium (NOMS 2008), April 2008.

[6] A. Chourmouziadis, G. Pavlou, "Efficient Web Services Event Reporting and Notifications by Task Delegation", DSOM 2007, Vol. 4785/2007, September 2007, pp. 242-255.

[7] A. Chourmouziadis, G. Pavlou "A Web Services approach to address efficient information retrieval in network management", London Communications Symposium, University College London, September 2005.

Chapter 1

1 Introduction

1.1 Problem Definition and Motivation

Network, system and service management has grown into a significant research discipline during the last twenty years. Although research and standardisation took place starting from the mid 80's, the quest for a general enough technology to be used for network, system and service management still rages. Research work in the past and present has tried to cover aspects of network and service management such as the organisation of management applications, the modelling of the information reflecting the status of managed resources, and the logic and algorithms behind solving management problems. While though architectural and information modelling aspects of network and service management can be agreed upon leading to standardisation, the logic and algorithms behind management problem solutions is a subject that will still require research effort in the years to come. These are true since new networking environments and management needs always emerge requiring new problem solving techniques. Despite the extensive research that has taken place in the last twenty five years and the numerous standardised solutions that have been devised and agreed, the quest for an "all encompassing technology" [1] still continues. This reflects reality since many technologies were abandoned and others found only specific niche markets.

Throughout the last twenty years a lot of management technologies made their appearance. Historically the first network management technologies were procedure based (i.e. the Distributed Computing and Management environments (DCE & DME)) of the Open Software Foundation group (OSF)).These technologies were later abandoned in favour of object oriented protocol based approaches such as the Open Systems Interconnection System Management (OSI-SM) [2] protocol and the Simple Network Management Protocol (SNMP) [4]. Combining the characteristics of procedural based approaches and object-oriented protocol based approaches, distributed object technologies such as the Common Object Request Broker Architecture (CORBA) [6] were proposed for use in network and service management. Another distributed management approach, the management by delegation move in the early 1990's, followed a different path from other technologies. Instead of managing devices in a remote fashion, the main idea behind this approach is to send code to a managed device so that the latter can perform these tasks locally.

Quite recently, eXtensible Markup Language (XML) based approaches and protocols, the most notable of which is Web Services (WS) for building distributed applications, have been proposed as a management technology that could also be used for Network and Service Management (NSM). Considering the common characteristics that WS share with distributed object technologies, it is no surprise that WS were suggested for NSM. Nevertheless before this is possible, WS need to solve several problems before they even become capable of being used for NSM purposes such as (a) potential problems for dealing with the strict performance requirements of NSM (b) interoperability problems when building WS management applications (c) modelling and standardization problems when translating information models and operations from other technologies to WS etc. In the next sections we present an overview of previous technologies and WS as well as our motivation for providing mechanisms that can increase the efficiency of WS management applications, which is one of the necessary steps towards making WS a technology capable to be used for NSM.

1.2 An Overview of Procedure and Object Oriented based Approaches for Network and Service Management

The first network management technologies introduced to solve the NSM problem were procedure based. Procedure based approaches allowed manipulation of managed resources in a soft remote manner. These approaches achieved the latter by creating non object oriented based procedural software specific to a particular application. In essence these technologies allowed remote manipulation of managed resources by emulating protocol functionality that was specific to a particular application. The latter means that each protocol function was supported by specific procedures. Examples of procedural based protocols were the Distributed Computing and Management environments (DCE & DME)) of the Open Software Foundation group (OSF). At the time though that these approaches were introduced, their working groups decided not to support object oriented based procedural software. As such despite the great amount of research work invested in these technologies, they never caught up and were abandoned because they coincided with the appearance of object-oriented distributed management approaches as well as generic protocol based manager-agent approaches.

The first object oriented approach was the Open Systems Interconnection System Management (OSI-SM) [2] for managing OSI switches/routers and end systems. OSI-SM introduced the manager-agent model, in which managed device resources are represented by objects to which collective access is provided by an agent. OSI-SM boasted significant innovations for its time such as sophisticated selective information access capabilities through scoping/filtering, a powerful event model and a set of generic functions. On the other hand though, it was a fairly complicated technology, it was tied to the OSI protocol stack and it required a lot of know-how from a user's perspective to use it for NSM. Thus OSI-SM found its niche market only in telecommunication environments.

An approach to map OSI-SM onto TCP/IP protocols, in order to use it for the management of IP-based networks led to the introduction of CMIS/P over TCP/IP approach [3]. This approach never caught up because the Internet Management Community was already working on a simpler protocol, the Simple Network Management Protocol (SNMP) [4].

SNMP was envisioned as a management protocol solution with a simpler information model than OSI-SM which could be easily implemented and would incur smaller overhead on managed devices. SNMP uses a variable-based information model with a small set of generic operations to manage network resources. Although the simplicity of SNMP played a major role to its wide adoption and deployment, its simplicity introduced several problems. When SNMP was introduced, aspects such as efficient mechanisms to delete or create new data, bulk data retrieval, reliable event delivery and security were absent. These features were added in subsequent versions but the lack of other features especially in order to be able to perform changes on the network (i.e. transaction support for configuration management), led the management industry to use SNMP only as a monitoring tool [5]. Eventually the slow rate of development of SNMP contributed to IETF finally deciding not to evolve SNMP further in 2002 [5].

Trying to solve SNMP's problem for configuration management, IETF developed the Common Open Policy Service for Policy Provisioning protocol (COPS-PR). COPS-PR is a policy based protocol for configuration management over TCP and supports atomic transaction support and security. Its information model is similar to SNMP's Structure of Management Information (SMI) information model. Still COPS-PR is not backwards compatible with SNMP. The latter contributed a lot to the narrow adoption of COPS-PR. In addition IETF's fixation in continuing to promote a rudimentary information model eventually led COPS-PR "despite the original hype not to receive any significant uptake in the real world" [1], especially because it suffers from the same problems as SNMP [5].

1.3 An Overview of the Common Object Request Broker Architecture

Even though remote procedure call approaches for management eventually lost to the object oriented protocol based approaches, they had still set the foundations of managing network devices remotely through standardised dependent Application Programming Interfaces (APIs). The latter are important for the development of interoperable and portable applications. As such, remote procedure call approaches combined with principles from object-oriented approaches naturally evolved to distributed object based approaches. One such effort was led by the Object

Management Group (OMG) which produced the Common Object Request Broker Architecture (CORBA) [6]. In CORBA objects are accessed through interfaces which expose a number of methods to the clients that access them through stub objects. CORBA was initially conceived as an approach to support general purpose distributed applications. Its potential though made the management community to also consider it for network management by modelling managed objects as CORBA interfaces. Substantial research effort was invested into CORBA for remote management. This effort equipped CORBA with facilities for network monitoring and event reporting as well as a lightweight Portable Object Adapter (POA) to support large object populations representing different aspects of a managed device. Still CORBA was never used in large scale for network management because it did not support bulk data retrieval facilities for monitoring and because it had for that time a significant memory and latency footprint on managed devices. In the end, other technologies such as Java RMI and mainly Web Services supported at large by the industry, led CORBA into finding its niche market only in service management supported only by a number of telecommunication vendors.

1.4 An Overview of the Management by Delegation Move

A different approach from all the previous ones was the management by delegation move in the early 1990's. The main idea behind this approach is that instead of performing management tasks in a remote fashion, code is sent to a managed device so that it can perform these tasks locally. This idea found ground for network management when platform independent languages such as Tcl and Java appeared, allowing the agents/servers managing the underlying resources to host new code more easily. As soon as these languages appeared, two trends for applying management by delegation emerged. The first are manager-agent approaches such as the Script [7], and the Expression MIB [8]. In these MIBs the lifecycle of a script or program is controlled through specific management objects handled by an agent. The second trend involves mobile objects migration and execution close to the managed devices to which these objects need access to. Mobile agent platforms were considered for network management using either a constrained or full mobility approach. In constrained mobility approaches the agent is told before hand which locations to visit and in the full mobility case the mobile agent makes autonomous decisions as to where it should move depending on the information it receives from its surrounding environment. Despite the fact that a lot of research work has been invested in the management by delegation approaches, the fact that there were never any significant, real world implementations, the increased development costs (management of management problem) and the inherent insecure nature of mobile code or scripts in general, has resulted in these approaches receiving very little attention by the industry.

1.5 A New Player in Network and Service Management

On the contrary, XML based approaches for network and service management have been on the rise in the last few years. The ability of XML to define arbitrary tags to describe the context of information through Document Type Definitions (DTDs) [9] and XML Schemas [10] make it an excellent mechanism to define management protocols, interface specifications etc. Given that the industry supports XML in many of its applications, using XML or XML-based technologies for NSM becomes considerably attractive. The latter, as well as the promise of faster product development, interoperability and application integration, has led to the adoption and implementation of many XML based technologies and standards to address the problems of Web based management. One prominent technology in this area is Web Services. WS is a technology that allows creating web interfaces that can be accessed over the Internet. Given the similarity to distributed object technologies, there has been a lot of research targeting their use for network, system and service management.

Over the past few years, XML and WS have been used by various research groups to define many Web-based specifications for network management. The work of some groups was integrated with the work of other groups [38], [39], [143], [144]. As such currently there are two main groups working on WS management specifications. These are the Distributed Management Task Force's (DMTF) Web Based Enterprise Management (WBEM) collection of specifications [11], [12] and the OASIS (IBM, HP) Web Services Distributed Management (WSDM) group of specifications [13], [14]. The first group has designed specifications such as the Web Based Enterprise Management Framework encompassing a set of management technologies developed in order to unify the management of distributed computing environments and devices. A key aspect of WBEM is the Common Information Model (CIM) [15]. The latter provides a set of generic classes from which application-specific information models are derived in order to expose the state of managed devices. An extension to CIM is the WS-CIM mapping for WS-based management. In addition to WS-CIM, DMTF has also devised the Web Services for Management (WS-Management) specification [12]. This framework specifies how to identify a manageable resource (represented by WS-CIM objects) and how to initiate communication with it. On the other hand, the OASIS group has issued two specification documents (a) the Management Using Web Services (MUWS) specification [13], [14] (b) the Management Of Web Services [24] specification (MOWS). The former specification considers how to manage resources with the use of WS and the latter how to manage WS endpoints through WS protocols.

Another XML based approach for configuration management trying to address the relevant shortcomings of SNMP, i.e. transaction support and security, is the Network Configuration protocol (NetConf) [25]. NetConf uses a set of predefined operations (edit-config, copy-config etc) to change some of the configuration parameters of a managed device by uploading to the agent of the managed device a new configuration stored in an XML document. This document is parsed by the agent who then enforces if possible the new configuration values. To enable transaction support a configuration may be retrieved, deleted, copied, enabled, locked revoked etc. NetConf supports currently three transport mappings, NetConf over Secure Shell (SSH) [27], NetConf over the Blocks Extensible Exchange Protocol (BEEP) [28] and NetConf over SOAP [26]. All transport mappings support security features to ensure authentication, data integrity and confidentiality. Work on NetConf itself has been finalised although work on associated data models has just started.

From all the above it is evident that a lot of work and research effort has been invested into evolving new XML and WS based standards for network, service and system management. Despite the promising potential of using WS for NSM, WS need to solve several problems before they even become capable of being usable for NSM purposes such as (a) potential problems for dealing with the strict performance requirements of NSM (b) interoperability problems when building WS management applications (c) modelling and standardization problems when translating information models and operations from other technologies to WS etc. One of the main shortcomings of WS and XML is the relatively large overhead incurred by the XML tags used to describe the context of management information. This means that WS inherently represent a technology with large application footprint. Network management operations though require keeping resource usage, latency and traffic overhead low. This way a management technology can remain scalable and unobtrusive to a network's smooth operation. Creating unobtrusive management technologies is easier today that the technical characteristics of managed devices and networks have increased substantially during the last decade in terms of the speed, memory and bandwidth that the latter can handle. But still real-time IP traffic analysis on high speed links is challenging for traditional solutions and especially for WS, mainly due to the little time available to process a packet (in the order of nanoseconds for 10 Gbps links). As such if WS are going to be used for NSM, one of the many necessary steps towards that direction is to find mechanisms in order to alleviate the initial overhead imposed by XML tags which results in an increasing memory, latency and traffic footprint.

A promising characteristic of WS and WS standards for management in general though is that they are designed to be loosely coupled. This is an excellent characteristic because it provides a lot of ground for many optimisations. As such, it is possible to design and use mechanisms to solve several of the known problems WS face when they are used for NSM. It is the investigation of features and mechanisms that are capable of minimising the WS technology footprint (one of the problems of WS-based management) for management that has motivated the research work presented in this thesis. This way we are trying to provide some solutions to the efficiency problem of WS-based management so that it might become possible to use WS effectively and efficiently for addressing critical management tasks.

1.6 PhD Objectives

This thesis looks into mechanisms that minimise the footprint of WS-based management applications. This is one of the many problems that need to be solved before using WS for NSM. By providing solutions to this problem, WS could be potentially used in a scalable manner to handle management tasks such as monitoring and event reporting. The goal of this research is to invent new or use pre-existing Web Services mechanisms in order to improve the performance of WS for network management. This way we will also be able to assess and to test the applicability and scalability of these mechanisms in providing solutions that could minimise the large application footprint problem of WS management applications for monitoring and event reporting. The key objectives are:

- Perform an examination of what has been done so far in the field of XML and WS based management and in particular in the field of monitoring and event reporting.
- Investigate approaches and mechanisms in order to increase the performance of WS-based management applications for monitoring and event reporting. More specifically, examine mechanisms in order to perform load distribution and task delegation of the monitoring and event reporting operations in order to increase performance of WS management applications.
    - As part of this investigation, we examine mechanisms that can be used to perform selective or bulk information retrieval to facilitate low cost and efficient monitoring. We show that a WS based query tool used for information processing and filtering of management state data is such a potential mechanism. As such we investigate the characteristics of such a query tool so the latter could be used efficiently in addressing monitoring and event reporting requirements.
    - As part of this investigation we will also examine the use of policies for load and task distribution in order to make the event reporting process for WS more efficient.
- Build a scalable optimised and efficient monitoring system based on our query tool.
- Build a scalable, flexible, dynamic and efficient event reporting system based on policies.
- Build a distributed architecture for monitoring and event reporting using the above systems and mechanisms. This architecture should support
    - The operations and messages of a custom framework for potentially better performance within a network domain.
    - The operations, messages and concepts of a standard management framework for interoperability at the edges of a domain.
- Evaluate the performance and scalability of these systems and our framework.

- Complement our WS-framework by building a Graphical User Interface (GUI) tool that will use and enhance the above systems and mechanisms with extra capabilities. This system will be a high level manager and will be the heart of our WS-framework.

1.7 Achievements

In this thesis, the focus is on introducing mechanisms that will allow WS to be used efficiently for monitoring and event reporting of network devices. As part of this, we first introduce a custom query tool that can be used for monitoring and event reporting and has the following characteristics:

- Exploits the relationships between state data for effective monitoring.
- Minimises its footprint.
- Offers bulk and filtering retrieval capabilities.
- Supports task and load distribution as part of a distributed monitoring architecture.
- Supports its operation within custom and standardized frameworks.

Compared to other tools such as XPath, this tool will be shown to be more scalable for several monitoring and event reporting management tasks, as well as more flexible in exploiting any types of relationships between state data and not only tree relationships (i.e. XPath). Using this query tool as part of a custom framework and as part of a monitoring and event reporting system we will show that in various monitoring and event reporting scenarios, WS efficiency increases and becomes comparable and sometimes even better than older technologies such as SNMP.

As part of increasing the performance of WS for NSM, we also introduce and use policies as the means to manage the event reporting process and build WS based event reporting systems that

- Are more efficient in several scenarios than standard WS-based systems and SNMP traps.
- Provide useful notifications helping a manager in pinpointing network problems which was a common problem for other technologies such as SNMP.
- Have the potential of minimizing a manager's supervision.

In addition to the previous we also introduce a Graphical User Interface tool to support, enhance and complement the functionality of the monitoring and event reporting systems mentioned above.

1.8 Thesis Structure

The rest of this thesis is organised as follows. In chapter two we perform a literature review on previous key management technologies and discuss their advantages and disadvantages. In this chapter we also give an overview of WS and we discuss and analyse how they can be used for NSM. Based on this analysis, we introduce the research that has been performed so far in WS based management and especially in monitoring and event reporting. In this chapter we also discuss briefly what needs to be performed in order to extend or improve the current research work on monitoring and event reporting. In chapter three we introduce our work on a query tool and its design characteristics so that the latter can be used as an efficient mechanism for monitoring and event reporting. Also in this chapter we also introduce an architecture, a custom framework and a monitoring system that supports distributed monitoring using the query tool we have designed and implemented. In chapter four we introduce a set of scenarios based on which we will evaluate the capabilities of our query tool against XPath (XML Path language)-a general XML based query tool. Based on these scenarios and after showing that our query tool is more scalable and allows addressing several measurement scenarios, we also evaluate the performance of our custom monitoring framework that uses our query tool against a standard protocol such as SNMP. This way we show that our query tool, framework and architecture can support efficient and scalable distributed monitoring. In chapter five we introduce our work on managing the event reporting process with policies in order to increase the performance of WS-based event reporting applications. As part of this work we introduce our policy specific grammar for managing the event process and an event reporting system we have designed and implemented that uses policies and our query tool. As part of chapter five we demonstrate the benefits of the performance of our event reporting system in trying to minimise the footprint of WS based event reporting applications. In chapter six we introduce our work on the design and implementation decisions in building a graphical based management tool to support and enhance the functionality of our monitoring and event reporting systems. In chapter seven we draw conclusions about our achievements and propose future directions for our work.

Chapter 2

2 Literature Review and Related Work

2.1 Introduction to Network Management

Research work in the past and present has tried to cover aspects of network and service management such as the architecture of management applications, the modelling of the information reflecting the status of managed resources, and the logic and algorithms behind solving management problems. While though architectural and information modelling aspects of network and service management can be agreed upon leading to standardisation, the logic and algorithms behind management problem solutions is a subject that will still require research effort in the next years to come. This is true since new networking environments and management needs always emerge requiring new problem solving techniques. Despite the extensive research that has taken place in the last twenty five years and the numerous standardised solutions that have been devised and agreed, the quest for an "all encompassing technology" [1] still continues. This reflects reality since many technologies were abandoned and others found only specific niche markets.

Putting it plainly, Network and Service Management (NSM) is a scientific discipline that consists of many conceptual areas and aspects [29]. As such it is a discipline that involves a very complex subject with interchangeably related issues and facets and with requirements that are often conflicting. The main functional areas of NSM processes according to the International Organisation for Standardisation (ISO) based on [30] and [31] are five: (a) performance management (b) configuration management (c) accounting management (d) fault management and (e) security management. These areas involve:

- ***Performance Management (PM):*** PM involves taking measurements for various aspects of a network such as network throughput, user response times and line utilisation. These measurements help in maintaining internetworking performance at an acceptable or desirable level. To maintain performance at an acceptable level PM involves aspects such as data gathering, data analysis, defining threshold values that trigger events worthy of attention, determining performance metrics, running simulations for network planning and partitioning etc.

- ***Configuration Management (CM):*** CM usually involves the definition, collection, monitoring, and alteration of configuration data so that the effects on network operation and on various versions of hardware and software elements can be tracked and managed.
- ***Accounting Management (AM):*** AM is concerned with measuring network utilisation parameters so that individual network usage of the network can be regulated appropriately. To measure the utilisation of all important network resources a combination of usage patterns and usage quotas are used and maintained. These patterns yield billing information as well as information used to assess continual fair and optimal resource utilisation. This way it is possible to minimise network problems and maximise fairness of network access across all users.
- ***Fault Management (FM):*** FM entails detecting, logging and notifying users of potential issues that will cause network problems. FM also entails fixing automatically these network problems in order to keep a network running effectively. To do the latter FM tries to reduce network downtime or unacceptable network degradation by determining fault symptoms and isolating the problem at hand. Having isolated the problem, it is possible to fix it and evaluate the effectiveness of the solution given.
- ***Security Management (SM):*** SM necessitates controlling access to network resources according to local guidelines so that the network cannot be sabotaged (intentionally or unintentionally). In addition SM involves protecting sensitive information from unauthorised access. To do the previous SM entails identification of sensitive network resources, configuration of mappings between sensitive network resources and user sets, monitoring access points to sensitive network resources and logging inappropriate access to resources.

From all the above it becomes clear that NSM comprises many different conceptual areas and aspects. While it is relatively easy to identify these areas, it is not a simple task to provide simple solutions for each one. This occurs first because new requirements always arise so a solution given today to these problems may not be appropriate for tomorrow. At the same time all these NSM conceptual areas have strict objectives to fulfil. Thus solutions to these NSM objectives can have conflicting requirements. The next section presents some of the most important NSM objectives. In the next sections we will also analyze potential problems for each technology to meet the objectives of NSM for each functional area. Currently we just provide in Table 2-1 a brief overview of the functional areas that each technology has tackled and problems that arise.

	Technology				
Functional Area	OSI-SM	SNMP	CORBA	WS	Management by delegation
PM	Yes / telecoms specific	Yes but limited and efficiency problems	Yes / limited/ efficiency problems	Yes / efficiency problems	Yes but increased development cost
CM	Yes/ complex /	Yes/ difficult/ problematic	Yes	Yes / currently under standardization	Not known
AM	Yes / telecoms specific	Not known	Yes vendor specific	Not known	Not known
FM	Yes/ complex/ telecoms specific	Limited / sometimes unreliable	Yes	Yes/ convergence for interoperability pending	Yes / increased development cost
SM	Yes / telecoms specific	Limited / added SSL features / efficiency issues	Yes / added later	Yes / some issues arise	Inherent security problems

Table 2-1 Functional area aspects that each technology has tackled

2.1.1 Network Management Objectives

NSM comprises many facets and is applied through the use of many different technologies, approaches, algorithms, architectures, information models etc. Since many of these facets may have conflicting requirements, solutions are given on a trade off basis. The most important requirements and objectives of NSM, as well as potential problems that can arise in the enforcement of these objectives are given below.

- All technologies, software, information models, algorithms used for NSM should employ management solutions that are unobtrusive on the smooth operation of a network. The latter can be interpreted to employing management solutions for NSM that are scalable. This in turn is translated into having management solutions that incur a minimum overhead in terms of latency, traffic, memory and other network resources. The management problem though is sometimes very complex and may require complex solutions. Complex solutions to management problems though, can be more obtrusive to the network operation. On the other hand providing management solutions having in mind the simplification of the management problem at hand can lead to oversimplification. The latter may result in having difficulties when solving more complex problems such as configuration management (i.e. the SNMP simplicity makes it difficult to use it for configuration management). As such oversimplification can also increase the complexity of solving a management problem and thus it may also affect scalability.
- A management solution to management problems should be driven by the principles of low complexity, minimal development cost and software application reuse for the

reasons explained below. Following these principles can help maintain the management infrastructure simpler and promotes extensibility and adaptability. The latter is extremely important but also difficult to achieve as it is not an easy task to visualise all future management needs today. Nevertheless an adaptable and extensible management solution can cope with problems and new network management needs that were not foreseen at the time of its deployment.

- Interoperability is a necessity for any management solution. During the last 25 years a lot of management standards, architectures, protocols, algorithms, information models, software etc have been designed, used and applied to a variety of different networks. All these technologies should be able to co-exist. As such interoperability allows a management solution to be backwards compatible with other technologies and promotes technology reuse and cooperation.
- Security is a feature that should go hand in hand with all management solutions. As such security features should be developed as a core trait of any management framework or standard to prevent unauthorised access to management data. Such security features guarantee data integrity, confidentiality and the network's good operation. Applying these features in distributed environments is a difficult challenge both in terms of guarantying that these features can not be easily bypassed but also in terms of scalability (i.e. use of SSH in SNMP).
- A management framework should allow intermixing and atomicity of operations. Many tasks in NSM are quite complex and thus require a number of consecutive operations to be performed (i.e. configuration management). Thus a management framework should permit the intermixing of several operations together to perform a complex task. Sometimes though when intermixing operations the enforcement of one operation depends on the results and enforcement of previous operations. Therefore a framework needs to support atomic commit of operations. This way if an operation is not enforced, operations depending on its enforcement will not be enforced either. If the previous situation occurs, a management system should be able to return to its previous state. This though is not an easy task and it is not supported by some technologies as for example SNMP. In SNMP the simplicity of the information model hides the relationships between the data representing the state of a device. At the same time performing actions with SNMP is carried out by setting variable based data. Combining all this resulted in having cases where a single operation on data can turn into a sequence of SNMP interactions. This makes it difficult to maintain state until an operation is complete, or until failure has been

determined. Even if failure is determined, rolling the device back into a consistent state is difficult.

- A management solution should allow easy and effective access to management data for monitoring, event reporting and configuration (reading and altering management data). NSM to a great extent involves monitoring the network status in order to guarantee a network's good operation. The latter is required in order to find faults in the operation of the network to overcome these faults by altering its configuration. To do all these operations, tools are required to process, access and alter effectively the state of managed devices. The construction of these tools can be facilitated if a management framework is based on an information model to represent state data that promotes usability and expressiveness in reading or altering these data. A very simple information model though hides the relationships between management data that represent the state of a device (i.e. SNMP). As such, tools that uncover these relationships and offer facilities for processing and accessing data in a bulk or selective manner are often quite complex. Complexity increases the footprint of a management application and inhibits scalability. In addition, complexity also limits the use of management tools to specialists.

These, as well as some other features of secondary importance but none the less important, need to be part of any management framework. In the past a lot technologies and standards were introduced as part of an effort to offer an efficient solution for network and service management. Some of these solutions did not manage to adapt to the new management needs in the passing of time and were abandoned. Others that were more successful managed to find their own niche markets. This has happened because all frameworks and management technologies have their advantages and disadvantages towards addressing management problems and objectives. The next sections investigate the history of some of these technologies, according to their characteristics and looks into some of the key technologies in NSM and their problems.

2.2 History and Analysis of Key Network Management Technologies

A lot of research has been invested in the last twenty five years in Network, System and service Management. Research work in the past and present has tried to tackle aspects of NSM covering all its main functional areas as these were presented and analysed in the previous section. Despite the extensive research that has taken place, a series of problems with the use of management technologies still arise. In the next sections we will investigate the history of several NSM approaches. In the next sections we also examine some of the main technologies that were, are being used and will be used in the near future. As part of this investigation we also explore potential problems that arise with the use of each technology.

2.2.1 OSI-SM

The first significant object oriented approach for network management was the Open Systems Interconnection System Management (OSI-SM) [2] for managing OSI switches/routers and end systems. To solve management problems, OSI-SM introduced the manager-agent model (Figure 2-1). In this model, the resources of managed devices are represented by objects at different levels of abstraction [36] and are accessed by a manager. Collective access to these objects is provided to the manager by an agent. The agent offers access to collections of managed objects classes (these represent the underlying resource) called "clusters" across a management *interface*. Interfaces are defined in a formal way based on a standard specification and using the managed object types supported by this specification to represent the underlying resource.

The information model of OSI-SM mandates a specific way to represent the underlying resource. The information model of OSI-SM is written using the OSI Abstract Syntax Notation 1 (ASN.1) [37] language. The latter is a data structuring language that supports simple and constructed types for expressing the properties of an underlying resource. The Management Information Model of OSI-SM (MIM) is presented in [43], [44]. A fundamental ASN.1 type in this information model is the Object Identifier (OID) [32]. This represents a sequence of non-negative integers on a global registration tree to structure managed objects of devices hierarchically. OIDs are unique and are registered by standards bodies (i.e. ISO). The collective view of this tree of object classes through the management interface is called a Management Information Base (MIB). An OSI Management Information Base (MIB) defines a set of Managed Object Classes (MOCs) and a schema that defines the possible containment relationships between instances of these classes [1] (Figure 2-2). The OSI-SM information modelling principles for MOCs are labelled to as the Guidelines for the Definition of Managed Objects (GDMO) and are specified in [45].

To have access to managed objects in OSI-SM, *manager* applications are used to access the interfaces through which *agents* expose these objects. *Agents* are typically implemented in software that serves management requests, and dispatch events through the management protocol. Through the manager-agent model, OSI-SM standardised only how information is modelled, leaving aspects of the internal structure of managed systems undefined. As such highly optimised implementations could be devised since internal aspects and APIs need not be standardised [36]. As a result of all this, OSI-SM can be considered mainly as a communications framework and protocol through which access to managed objects is achieved [1].

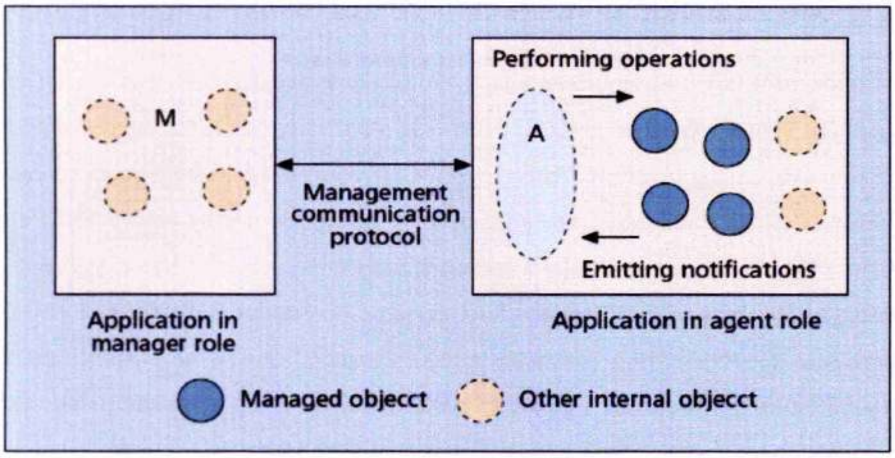

Performing operations
M
A
Management communication protocol
Emitting notifications
Application in manager role
Application in agent role
Managed objecct
Other internal objecct

Figure 2-1: The Manager-Agent Model [36]

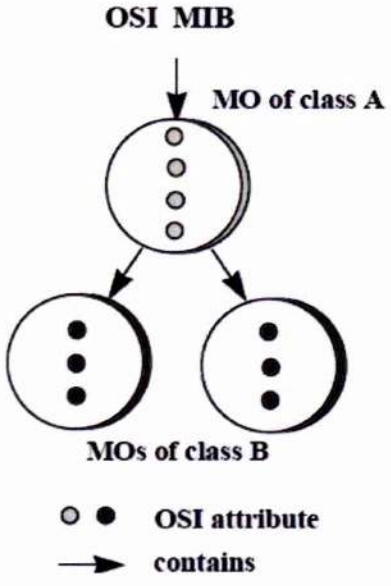


Figure 2-2: OSI Management Information MIB [36]

As mentioned previously OSI-SM left internal implementation aspects undefined. As such agent software could use implementation-specific mechanisms to allow access to managed objects. As a result many implementation specific mechanisms were defined. Part of these implementation-specific mechanisms, was the ability of the OSI-SM agent software to evaluate event notifications at the source of their production based on predefined criteria. As such, the agent was able to emit events only to entities that were interested in receiving these events. The event software was sophisticated for its time and allowed managers to distinguish between the events they wanted to receive, with filtering on the event type, time, object name of the object that emitted the event, and the actual notification information. Filtering was not only used for event reporting but was also used for monitoring so as to retrieve only the specific management information representing the state of a device that the manager was interested in receiving. OSI-SM also supported mechanisms for bulk retrieval of management state data through an operation called scoping. Based on all the previous it is evident that OSI-SM boasted significant innovations for its time. One of these innovations was also OSI-SM's support to allow intermixing and atomicity of operations. The latter permitted concerted configuration changes to take place through a series of *Set* operations allowing the agents of OSI-SM to keep track of state. As a result of all these sophisticated mechanisms, OSI-SM was adopted as the key technology for the Telecommunications Management Network (TMN) [35], and is accepted as being one the most sophisticated management technologies introduced so far, supporting features that should be present in any management technology.

On the other hand though, OSI-SM is a complicated technology with many implementation options and sophisticated features. As such it is expensive to deploy and implement, it is relatively difficult to use and requires a lot of know-how. All the previous limit its use only to specialists and experts. At the same time OSI-SM was tied to the OSI protocol stack, and thus found its niche market only in telecommunication environments. An approach to map OSI-SM onto TCP/IP protocols in order to use it for the management of IP-based devices led to the introduction of CMIP over TCP/IP approach [3] (CMOT). At the time though of the introduction of this protocol the Internet Management Community was working on SNMP for Network Management and as such CMOT never caught up. As a result despite the fact that a lot of research and standardisation has been invested into OSI-SM, resulting in characteristics that any management technology should possess, it never received large scale deployment. It should though be the base for influencing any future management technology.

2.2.2 SNMP

While OSI-SM found its niche market in the telecommunications area, the Internet Management Community was already working on a simpler protocol, the Simple Network Management Protocol (SNMP) [4]. SNMP is also based on the manager-agent model for accessing managed objects where a single interface to an agent is used to access a cluster of managed objects representing the underlying resources.

SNMP was envisioned as a simpler protocol that would have a simpler information model [32], it would be easily implemented, and at the same time would impose a smaller overhead to managed devices. As a result, the information model that SNMP uses is also based on ASN.1 but a sub-set of it. As such SNMP has limited support on the use of constructed types (only simple two dimensional tables are allowed). As a result of this, management data in SNMP are variable-based (object-based but no inheritance and classes since they are considered unnecessary complications). Again, as for OSI-SM, a fundamental ASN.1 [32] type in SNMP for representing variable-based data is the Object Identifier (OID). The latter represents a sequence of non-negative integers on a global registration tree used in order to structure managed device data hierarchically. The collective view of this tree of data through a management interface is also called a Management Information Base (MIB) (Figure 2-3). The SNMP information modelling principles for defining MIBs are labelled as the Structure of Management Information (SMI) and are specified in [RFC1155] for SNMPv1 [41] and in [RFC1902] for SNMPv2 [42].

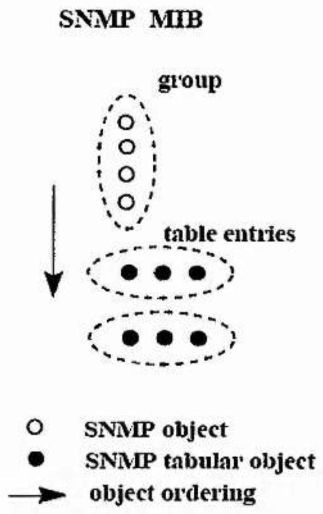


Figure 2-3 SNMP Management Information Base [36]

Based on the manager-agent model and simplicity, SNMP was designed to support a small set of operations so as it can be easily implemented. As a result of this as well as due to SNMP's operation over the Internet Protocol (IP), whose deployment was followed by its major adoption, contributed to SNMP's wide deployment.

The first version of SNMP (v.1) though, was missing a few very important aspects especially when compared to OSI-SM. Initially SNMP did not support (a) proper emulation of creation and deletion of data through the set operation (b) bulk data retrieval (c) reliable event delivery (d) security features. These were fixed in later versions (a) by adding a new operation (GetBulk) to support bulk data retrieval (b) by using TCP to support efficient and reliable management data delivery and (c) by introducing new security features and secure transport protocols such as SSH. Despite fixing these shortcomings in the sub-sequent versions, SNMP still faced many problems. According to [5] these are the main problems of SNMP:

- *Scaling Problems.* SNMP has very adequate performance when retrieving a small amount of data from many devices but is rather slow when retrieving large amounts of data from few devices. The latter is attributed to the lack of filtering and aggregation capabilities in order to reduce the management data that need to be collected. As a result the lack of these capabilities introduces scaling problems.

- *Limited transaction support (atomicity).* Because SNMP is variable-based and actions are performed by changing the state of a variable, a logical operation on a variable can turn into a sequence of SNMP interactions. The latter makes it difficult to maintain the state of an operation and roll back to a consistent state in case of failure.
- *No easy support for retrieval or playback of configurations*
- *Lack of high-level description of procedures.* It is often not easy from reading the MIB modules to ascertain how certain high-level tasks can be accomplished. In addition there is no description on how the various objects of the information model can be used to achieve certain management functions. As a result development cost increases.
- *Increased development cost.* MIB modules and their implementations are not available in a timely manner (sometimes MIB modules lag years). This happens because of the complex table indexing schemes and table interrelationships. The lack of structured types makes MIB modules much more complex to design and thus implement.
- *Few high level programming/scripting interfaces.* Operators view most of SNMP high level interfaces as too low-level and thus time consuming, inconvenient and impractical.
- *Poor performance on query operations that were not anticipated during the MIB design.* A typical example is when performing a more complex query. For example in order to determine which outgoing interface is being used for a specific destination address, the collection of more data than required has to be collected so that the manager can process them. This is the case because such queries were not anticipated when designing the already complex management MIBs.
- *The SMI language is quite complex and not very practical.*
- *SNMP traps do not usually contain much information to describe a problem.* As a result a SNMP trap usually is followed by information retrieval operations to figure the meaning of the trap or determine the cause of an event. The latter introduces more latency and traffic overhead and hampers scalability of SNMP operations.

All the previous suggest that the SNMP protocol was simplified in terms of the number of protocol operations and resource requirements on managed devices. It was not though simplified in terms of usability. This led to problems such as the lack of transaction support which is essential for configuration management. As a result, SNMP is being used mostly for monitoring and not for configuration management. This is true although SNMP supports configuration management of devices through the set operation and a set of guidelines determining best practices for configuration management have been specified in [138]. Even when some features were added in SNMP for configuration management, the fact that some of these features were added very late, contributed to IETF finally deciding not to evolve SNMP further in 2002 [5].

2.2.3 COPS-PR

Trying to solve SNMP's problem for configuration management, IETF suggested the Common Open Policy Service for Provisioning protocol (COPS-PR). COPS-PR was designed to support configuration management based on the manager-agent model with some very nice features such as:

- Support for high-level transactions on single devices such as deleting or replacing a configuration.
- Well defined atomicity of transactions. As a result if a failure occurs the manager is notified of it and the device is rolled back to the state of the last known "good" configuration.
- Guarantees that only a single manager can handle a specific configuration at a given point in time. This way the danger of corrupting a configuration from simultaneous access to it from many managers is minimised. To disallow corrupting a configuration, COPS-PR supports execution of configuration transactions in a specific order and permits only a single manager to have control, at any point in time, for a given subject category of a device.
- Synchronisation at all times between the manager and the device. This happens even if a communication failure occurs.
- COPS-PR is extensible with a use of features called capabilities. Manager applications are forced to adapt and use when communicating with a device only the capabilities that the given device supports.

The information model of COPS-PR is very similar to SNMP's (Policy Information Base (PIB)) Structure of Management Information (SMI) although is not backwards compatible with the latter. This hampered the wide adoption of COPS-PR. In addition IETF's fixation to continue promoting a rudimentary information model resulted in COPS-PR suffering from similar problems like SNMP. Some of these problems are the following:

- Very few standardised PIB modules due to increased development time. This happens because of the complex table indexing schemes and table interrelationships and also because of the lack of structured types which makes design and implementation more complex.

- No easy retrieval and playback of configurations for reasons similar to SNMP.
- The COPS-PR view of a managed device is data-centric. Thus mapping a task oriented high level view into a series of operations on management data and vice versa is very complex and difficult.

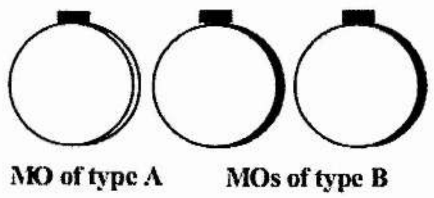
The above as well as other similar to SNMP related problems, combined with the fact that COPS-PR is not compatible with SNMP, contributed to the former not receiving any significant uptake in the real world" [1]. At the same time an XML-based approach is currently being standardised for configuration management, leading the internet community and the industry (NetConf) to abandon COPS-PR for configuration management of network devices.

2.2.4 CORBA

Remote procedure call approaches for management eventually lost to the object oriented protocol based approaches such as SNMP and OSI-SM. This did not happen though, before they had set the foundations of managing network devices remotely through a standardised Application Programming Interface (API). The latter allows programmers to develop distributed applications and promotes the construction of interoperable applications. As such remote procedure call approaches combined with principles from object oriented based approaches naturally evolved into distributed object based schemes.

One such effort was led by the Object Management Group (OMG) which produced the Common Object Request Broker Architecture (CORBA) [6]. In CORBA objects are accessed through interfaces which expose a number of methods to the clients that access them through stub objects. Unlike the manager-agent model, in CORBA every object has its own interface that is accessed separately by an application in client role. As such the fundamental building block of the CORBA information model is a programming language object-class (Figure 2-4). Defining the interfaces for these objects is performed using the Interface Definition Language (IDL).

CORBA was initially conceived as a method to access general purpose distributed applications. Its potential though made the management community to also consider it for network management by modelling managed object resource properties as CORBA interfaces. As such, substantial research effort extended CORBA with facilities for network monitoring and event reporting of managed devices. In addition, CORBA was equipped with a lightweight Portable Object Adapter (POA) to support large objects populations representing different aspects of a managed device. The latter is a mechanism that helps connect a request to access an object by linking the object reference used in the request with the proper code.



MO :Managed Object
▬ IDL interface
Note: attributes are not explicitly shown

Figure 2-4 CORBA Information Model building blocks

CORBA is based on the Open Distributed Processing (ODP) [46] framework for specifying and building distributed systems. ODP was conceived in order to solve the problem of inter-communication and inter-connection of heterogeneous systems. The goal of ODP was to promote software distribution, interoperability and portability. Using the ODP model of client-server interaction, where distributed application objects interact with other objects by accessing each other's interfaces, CORBA achieved access, location and replication transparency by hiding the complexities of the underlying platform (Object Request Broker (ORB)). The previous are achieved by providing:

- Well known server objects called name servers that provide interface references (Interoperable Object References (IORs)) to client objects. Name servers and IORs can be used by an object for the latter to acquire access to other objects.
- Hiding the underlying transport protocol for interoperability purposes inside the supporting software platform.

The ODP model is depicted in Figure 2-5. CORBA objects communicate with each other by the use of a Remote Procedure Call protocol (RPC) called the General Inter-Operability Protocol (GIOP). The most well known mapping of this protocol is over TCP/IP and is known as the Internet IOP (IIOP). This protocol provides reliable transport but at the same time hides the connection management processes from the ORB. Access to objects in Figure 2-5 can be static, through pre-compiled stubs, or dynamic, through the Dynamic Invocation Interface (DII). Events as shown in Figure 2-5 are disseminated through special servers called event notification servers or from event brokers [47], [48]. Event Brokers allow clients to specify the type of events they want to receive by filtering the event content.

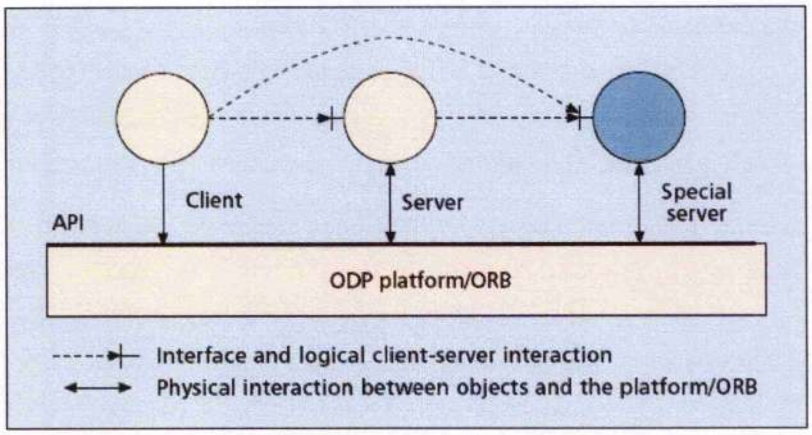

Client
Server
Special server
API
ODP platform/ORB
Interface and logical client-server interaction
Physical interaction between objects and the platform/ORB

Figure 2-5 The Distributed Object Model [1]

Though CORBA was initially seen as a unifying management technology, mainly because of its characteristics, it was never used in large scale for network management despite the fact it even supported atomicity of operations for configuration management. This is attributed to the following shortcomings.

- *Accessing management data values and attributes through a RPC can be expensive.* CORBA can become a very heavy technology even though it has a relatively lightweight POA to support large object populations. Translating for example the information models of SNMP and OSI-SM to CORBA would by default require exposing (a) one method per attribute to have access to the latter and (b) model dynamic entities such as TCP connections with separate objects. As a result a large number of objects each exposing a large number of methods (heavy-weight in terms of footprint objects) through their interfaces would have to be deployed. Thus a core network device such as a router may end up containing thousand of these objects which is not scalable. The solution commonly used according to the Joint Inter Domain Management taskforce (JIDM) [49] is to perform semantic and not syntactic translation of an information model. As a result commonly used attributes are accessed together through a single method. In addition dynamic entities are not modelled with a single object for each one but are returned through a single method as a "list of records" (this list is manipulated by the use of other methods). The previous suggestion can possibly reduce the resource overhead. The reduction size though depends on issues such as which attributes to group together for common access.
- *CORBA does not support selective retrieval facilities for monitoring.* There is no special server for supporting these operations. There are some proprietary solutions solving these problems but there are no standardisation efforts towards that direction.

Bulk retrieval can be supported by adopting the suggestion of the JIDM taskforce for semantic translation of a well known information model. As such collective access to a series of attributes is provided by a single method. The granularity of this scheme is though in question while selective retrieval using such a translation is not supported.

- *Despite support by some telecommunication equipment vendors, CORBA was never supported at large by the industry.* As a result, it was mainly used for service management and not for network management (i.e. Ericsson Radio Systems AB has used Orbix to develop its Cellular Management Operations System (CMOS) based on the Telecommunications Management Network (TMN) architecture).
- *Object references do not reveal any information about the object.* Object references in CORBA provide transparent access to objects. These references though are opaque types and have no internal structure. As such there are no means in CORBA to describe the services and the functionality of the interfaces that each object offers. Thus service discovery and service reuse in inhibited. This is not the case for newer technologies such as WS.
- *CORBA is not flexible enough for dynamic instantiation of new services.* CORBA does not provide a built-in facility for instantiating new interfaces to objects. As such, interface creation may only be supported by existing interfaces. This approach is not flexible as a factory interface is always necessary for every other interface that can be dynamically created. In newer technologies such as WS, specifications such as the WS-composition or the WS-ServiceGroup specifications allow composing new services from existing ones. In addition to this, web servers used in WS allow dynamic instantiation of new interfaces (one of the possible ways).

Based on the previous shortcomings and the fact that other technologies were introduced in the meantime for distributed management (i.e. WS) restricted CORBA's wide adoption for network and service management. CORBA found its niche market only in service management with support by a number of vendors in the telecommunications domain.

2.2.5 Management by Delegation

A different approach from all the previous ones was the management by delegation move in the early 1990's. The main idea behind this approach is that instead of performing management tasks in a remote fashion, code is sent to a managed device so that the latter can perform these tasks locally. This idea found ground for network management when platform independent languages such as Tcl and Java appeared, allowing the agents/servers managing the underlying resources to host new code more easily. As soon as these languages appeared two trends for applying management by delegation emerged.

The first trend is manager-agent approaches in which the lifecycle of a script or program is controlled through specific management objects handled by an agent. The introduction of the Script and the Expression MIB took place back in 1999 by the IETF Distributed Management Working Group (DISMAN) which was chartered to define a set of managed objects for specific distributed network management applications. The goal of this work was to employ the advantages of distributed management over the centralised concept to tackle the increasing demands of network management. Some of the main parts of this group's work was the Definitions of Managed Objects for the Delegation of Management Scripts (Script MIB RFC 3165) [7] and the Distributed Management Expression MIB (Expression MIB RFC 2982) [8].

The second trend involves mobile objects/agents migration and execution close to the managed devices to which these objects need access to. Mobile agent platforms were considered for network management using either a constrained or full mobility approach. In constrained mobility approaches the agent is told before hand which locations to visit and in the full mobility case the mobile agent makes autonomous decisions to where it will move depending on the information it receives from its close environment.

Management by delegation was initially seen as a more flexible approach through which task, CPU, and network load delegation (Script MIB, mobile code) could be performed. This allowed performing management of devices, close to where the actual operations on these devices should be performed. This in theory can reduce the footprint of management applications. Other benefits exist as well. For example some approaches such as the Expression MIB [8] were introduced as a way to create new, customised MIB objects for monitoring and event reporting. This is very useful as monitoring and event reporting of the state of a device is not limited to objects in predefined MIBs. Despite the obvious benefits of these approaches in terms of robustness, reliability and flexibility, inherent problems prevented these technologies from been deployed in large scale. This is attributed to the following shortcomings.

- *The "management of management" problem* [50]. Approaches such as the Script MIB and mobile code present managers with the need to distribute, run, update, control a script, and to gather and correlate the intermediate and final execution results. This increases maintenance costs.

- *Significant performance issues*. In [51], and [50] significant issues in terms of the performance of the Script MIB are identified (e.g. big memory overhead). In other cases such as that of the Expression MIB [8], the standard itself states that using the Expression MIB for monitoring or event reporting is often not a good trade-off for objects that are simply to be recorded or displayed. This is often the case when performing network management monitoring operations and in some cases for event reporting operations.

- *High development costs.* In specifications such as that of the Script MIB many aspects were left undefined such as a) how to get the Script MIB tables populated, b) how to get scripts re-fetched and restarted after updates, and, c) procedural details of event forwarding (where the call-back address for event forwarding is stored, how event correlation is performed etc). In other specifications such as that of the Expression MIB the DISMAN charter [54] states that "implementing the Expression MIB is non-trivial work and takes lots of months to complete". All the previous reveal that management by delegation approaches present increased development cost.
- *Severe security issues.* Running software or scripts is inherently very dangerous since it is difficult to check what the nature of a script or mobile code is. This makes it clear why the industry was not inclined to use and deploy at large scale approaches such as intelligent agents or the script MIB.
- *Integration to current devices is sometimes difficult.* Integrating for example the Expression MIB in existing agents that do not support it is not possible unless the agent uses an extensibility feature like AgentX [52], [53]. Even so, "sub-agent access from the master agent for MIB variables" as it is required to support the Expression MIB in existing agents [53], is a non goal for AgentX. As such, even with AgentX the Expression MIB can not be used for the purpose it was designed for.
- *Restricted access to resources.* The specification of the Expression MIB allows access only to local data since remote addresses are not supported. This eliminates the use of remote data in the evaluated expressions. Thus using the expression MIB for distributed monitoring is limited since access is provided only to local agent resources. The inherent distributed nature of other technologies introduced later such as WS, allows us to overcome such problems [55].
- *Support for bulk and selective retrieval is impractical.* Using the Expression MIB to support bulk and selective retrieval using a series of expressions even with wildcarding is not plausible. This happens because the objects that need to be accessed and evaluated by an expression have to be re-set each time for different monitoring tasks.

A lot of research work has been invested in the management by delegation approaches. Nevertheless there was never any significant deployment of the Script and the Expression MIB due to the increased development costs and due to the insecure nature of management scripts. This has led these technologies not to receive significant support by the industry.

2.3 WS for Network and Service Management

2.3.1 Introduction

Contrary to other technologies whose development during the last few years have diminished, XML-based approaches for network management have been on the rise. This is attributed to many reasons. The ability of XML to define arbitrary tags to describe the context of management information through Document Type Definitions (DTDs) [9] and XML Schemas [10], make it an exceptional mechanism to define management protocols, interface specifications etc. In addition the industry supports XML already in many applications. As such, using XML for network management is made considerably attractive.

The promise of faster product development, interoperability, application integration and industry acceptance has led to the adoption of XML to create technologies that can be used for Web based management. One such technology is Web Services. WS is a technology that allows creating web interfaces that can be accessed over the Internet. Given the similarities to distributed object technologies, there has been a lot of research targeting their use in network, system and service management.

Despite the promising potential of using WS for NSM and the common characteristics it shares with distributed object technologies, WS need to solve several problems before they even become capable of being used for NSM purposes such as (a) potential problems for dealing with the strict performance requirements of NSM (b) interoperability problems when building WS management applications (c) modelling and standardization problems when translating information models and operations from other technologies to WS etc.

In the next sub-sections we will provide an introduction to WS and also to the protocols and standards the WS industry has introduced for network and service management. This introduction is necessary to provide the background for understanding the design choices made as part of this thesis work in the next chapters. Also based on this introduction and standards, we will discuss how WS can be used for NSM. Following this discussion, we are going to introduce the state of the art approaches and research in this field. Based on this research we will identify advantages for using WS for NSM and elaborate on potential problems that need to be solved. One of these problems is performance and scalability of WS for NSM which also has been the motivation for the work in this thesis.

2.3.2 WS Background

Web Services are an XML document based technology designed to support interoperable interaction between computing system processes in a distributed manner. In essence "given that Web services are based on XML documents for exchanging information, it could be said that the technological underpinning of a WS is document-oriented computing" [56]. The core WS components for document oriented computing are the following:

(a) The service. This is the software that is used to process the document being exchanged through the use of a variety of application and transport protocols (i.e. programming language objects, stand alone system processes etc)

(b) The XML document. This contains all the application-specific information that a service consumer process sends to a producer process for processing. The format of this message is based on an XML schema [16], [17] which both the consumer and producer have access to. This way the consumer and producer of a service can validate and interpret the documents they exchange. The common means of doing the latter is through the Web Services Description Language (WSDL) [18], [57].

(c) The address. This is a combination of a protocol and a network address through which the consumer process is allowed access to the service (i.e. http protocol + network address)

(d) The envelope. This is a protocol that encapsulates the XML document and ensures that the latter will be separated from other data that the consumer and producer service might want or need to exchange.

2.3.2.1 Understanding SOAP

The protocol that is mostly used for the exchange of messages in Web services is the Simple Object Access Protocol (SOAP) [59], [60] (the envelope). SOAP "is a lightweight protocol intended for exchanging structured information in a decentralised, distributed environment" [60]. SOAP was designed to be 1) extensible, 2) usable over a variety of underlying networking protocols, and 3) independent of programming models. The first characteristic of SOAP is achieved by building the SOAP messaging framework on XML through the use of DTDs and XML Schemas. The latter are extensible and thus SOAP is extensible. Additionally for the second characteristic, SOAP allows the use of any transport protocol for the exchange of XML messages. Before this happens though, a binding with each transport protocol that SOAP uses has to be defined and standardised for interoperability purposes. To guarantee the third characteristic, SOAP is not bound to a specific programming model, such as the Remote Procedure Call (RPC) processing model for sending messages in a request-response style, although it supports it. This way SOAP is not bound to the pre-defined semantics that follow for example the RPC model. As such, SOAP is constructed so that it can be used with any message processing model (Message Exchange Pattern (MEP)). SOAP supports various MEPs including one to exchange one way messages as well as request response messages (the latter model is not necessarily RPC so as SOAP is not tied to the pre-defined semantics of the latter). In order to use these MEPs without SOAP attaching itself to their semantics, SOAP defines the general format of the actual messages exchanged between two message processing entities (a sender-receiver) and their intermediaries, but not how to treat certain aspects of the message [59](i.e. various element tags in the header). The semantics of how to treat specific aspects are defined in other specifications and have to be agreed between the various entities in the message communication path (sender-receiver-intermediaries).

In SOAP, XML messages are carried inside an XML structure called the *soap envelope*. The latter consists of two parts.

The first part of the *envelope* is the *header*. The *envelope* holds all the information necessary to process the SOAP message correctly. The *envelope* contains for example information important to create applications in which a message can be passed between multiple intermediaries before reaching its final destination [61]. To have SOAP support communication between two entities as well as intermediaries, certain information/rules in the header have to be processed by a SOAP processor as a message travels from its sender to its receiver. Example of specific information rules contained in the header are (a) the address of where the message is going (i.e. a WS-Addressing address [19]) (b) how the message should be treated by intermediaries (c) WS-Security signatures [23].

In the last example of header specific information, WS-Security signatures can be used to define the entities that can have access to the information contained in the XML message carried in the SOAP envelope.

A scenario for the second example of header specific information where a number of intermediaries exist between the sender and the receiver of a SOAP message is given Figure 2-6. In this figure a number of intermediaries exist between the sender and the receiver of a request. By looking into the header elements of the SOAP envelope the intermediaries can process and change the transport protocol binding used to route the request to its receiver.

For the first example of header specific information, a scheme of WS-Addressing addresses is given in Figure 2-7. In this example the header contains information about the address of the receiver and the sender of a message. WS-Addressing provides transport-neutral mechanisms to identify Web service endpoints and to secure end-to-end endpoint identification of WS messages that were not envisioned at the time of the definition of SOAP and WSDL. WS-Addressing enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner. To do all the above, WS-Addressing uses two mechanisms (a) EndPoint References (EPRs) and (b) Message Information Headers (MIHs).

EPRs support a set of dynamic usage patterns not covered by WSDL 1.1 so as to allow WS to be used in the following usage scenarios [98]:

- Dynamic generation and customisation of service endpoint descriptions.
- Identification and description of specific service instances that are created as the result of interactions between WS that involve manipulating state.
- Flexible and dynamic exchange of endpoint information in tightly coupled environments where communicating parties share a set of common assumptions about specific policies or protocols that are used during the interaction.

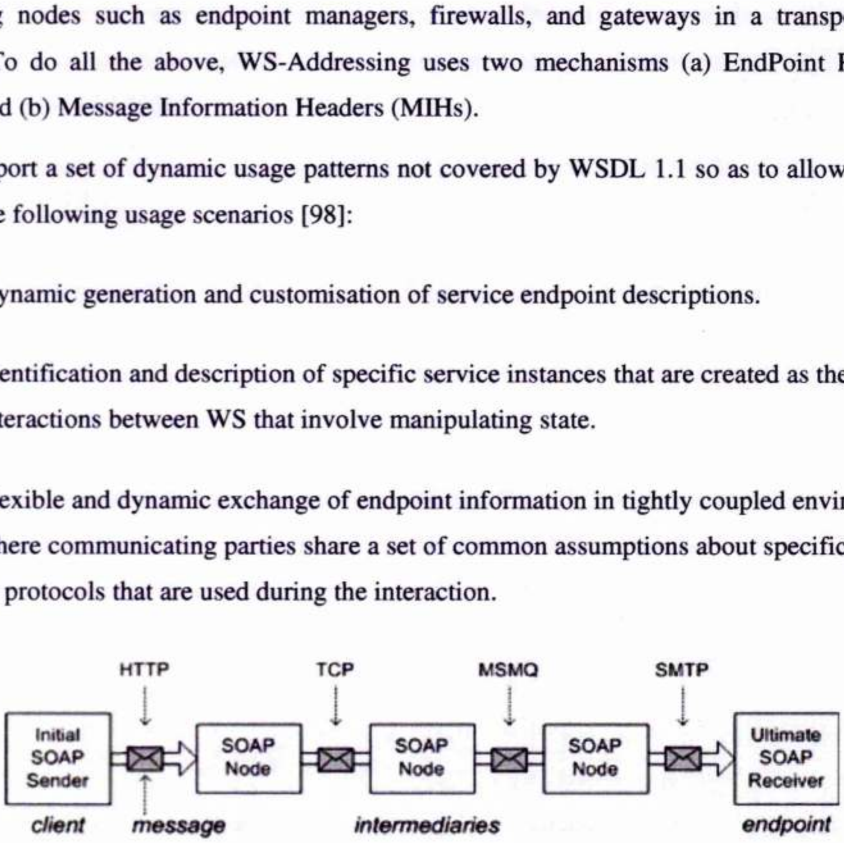
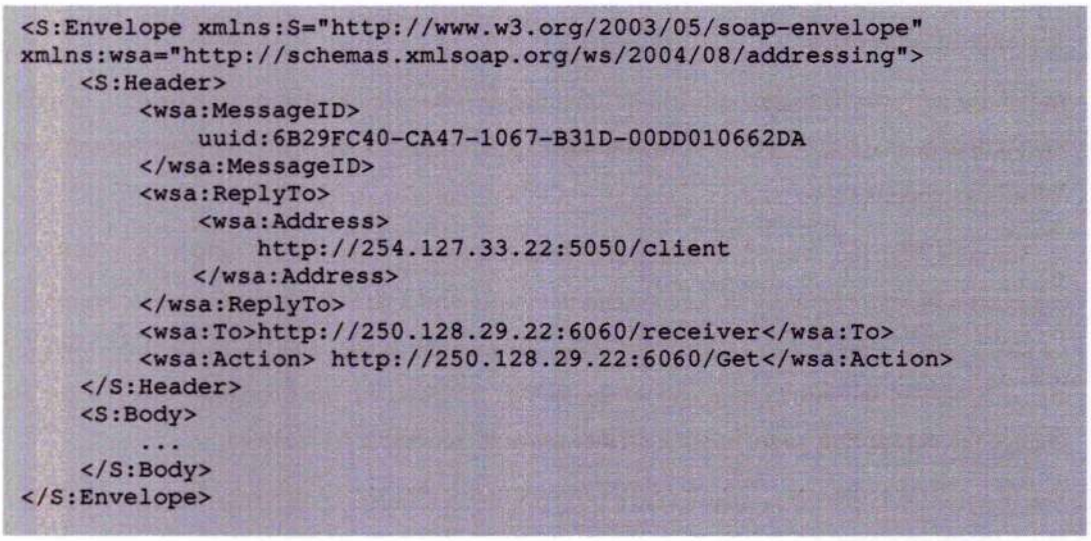


Figure 2-6 SOAP messaging with intermediaries [62]



```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
    <S:Header>
        <wsa:MessageID>
            uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
        </wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>
                http://254.127.33.22:5050/client
            </wsa:Address>
        </wsa:ReplyTo>
        <wsa:To>http://250.128.29.22:6060/receiver</wsa:To>
        <wsa:Action> http://250.128.29.22:6060/Get</wsa:Action>
    </S:Header>
    <S:Body>
        ...
    </S:Body>
</S:Envelope>
```

Figure 2-7 WS Addressing scheme: Message source and sink addresses

A detailed list of example of messaging scenarios addressed using WS-Addressing is given in the WS-ReliableMessaging specification. In essence though, EPRs extend or complement the WSDL description model (e.g. the portType, binding and service elements etc.) to support various messaging scenarios though they do not replace it.

MIHs augment a message exchanged between WS endpoints (in other words a sender and a receiver) with properties that enable the identification and location of the parties involved in an interaction. The latter is important in order to support various Message Exchange Patterns (MEPs). The properties with which MIHs extend an endpoint are [99]:

- **Destination properties:** Addresses of the intended receiver of a message.
- **Source endpoint properties**: References of the endpoints where messages originated from.
- **Reply endpoint properties**: Endpoint references that identify the entity to which replies to a message need to be sent.
- **Action properties:** Identifiers that uniquely identify the semantics implied by this message.
- **Message ids:** URIs that uniquely identify a message in time and space.
- **Relationship properties**: A pair of values that indicate how a message relates to other messages.

The second part of the *envelope* is the *body*. The latter contains the actual data that Web Service applications need to process. The formatting of the body can follow mainly two encoding styles (a) RPC and (b) Document. The first style is based on the concept of using SOAP messages to create Remote Procedure Calls. RPC calls result in issuing a command to the receiving entity of the message. When the style of a message is RPC, it is mandatory that the name of the method invoked when accessing a WS is the same as it appears in the body of the SOAP message. The Document style involves simply having data in XML format as the content of the SOAP body. The format of these data is agreed upon between the sender and the receiver [62] (i.e. through the WSDL definition of a service the sender and receiver produce stub and skeleton code which when used defines the appearance of the request and response of the SOAP body). How the receiving application will use or respond to the SOAP data in Document style is application specific.

SOAP also supports two techniques for deciding how to serialise the data contained into the body of a SOAP message over the wire (encoding). The first technique is using literal XML Schema definitions. The second technique involves using SOAP encoding rules. With the former approach an XML schema describing every piece of information in the body defines the XML format of the latter without ambiguity. Using SOAP encoding rules, the body is serialised always at runtime in a standard, specific manner. The latter technique though is more prone to errors, suffers from interoperability issues and leads to more verbose representations of SOAP data.

To understand how the style of the document and the encoding affect the body of the SOAP message, another important component of building WS has to be introduced. This component is the Web Services Description Language (WSDL).

2.3.2.2 Understanding WSDL

The Web Services Description Language was introduced to facilitate the definition of a service description. The latter indicates how potential consumers of services are intended to interact with the service. In essence WSDL is used to define in a standard, unambiguous, machine readable format [63], [64] (a) the information to send to a service in order to interact with it, (b) the information the service is going to send back, (c) the operations that a service exposes, (d) which parameters to pass to an operation, (e) via which protocols a service can be accessed, and (f) on which location the Web service resides so as to be able to access it. When defining all this information the services defined in WSDL have to be described in an interoperable and extensible manner. As such, when defining service interfaces with WSDL, the standard itself tries to facilitate interoperability and extensibility. Interoperability was actually one of the main reasons WSDL 2.0 was introduced since WSDL 1.1 was not very extensible. Also WSDL 2.0 was introduced so the latter could be compliant to the WS-Interoperability Basic Profile [68], [69] and also to support the SOAP 1.2 extensibility mechanisms. To facilitate extensibility, WSDL also differentiates between its abstract part and its concrete (implementation) part. This way the abstract part can be used as many times as required to define new services by defining only the concrete part of each new service [67].

As part of WSDL's abstract part the producer of a service has to first use an XML schema to describe the elements and the content of XML messages exchanged as part of the operations a service exposes. This is performed by using the basic elements and data-types from XML Schema or by defining new data-types of any complexity. In the WSDL language this is called defining the WSDL types element (*<wsdl:types>*). Using the elements and data-types inside the *<wsdl:types>* element, the producer of a service can define the operands that are going to be used as input and output of each operation that is exposed as part of a service's interface. In WSDL this means defining the *<wsdl:message>* elements. Following the definition of message elements, the producer needs to define the operations that are going to be exposed as part of a service's interface. Operations in WSDL are represented by *<wsdl:operation>* elements and the latter are part of the *<wsdl:portType>* elements. Each *<wsdl:operation>* element uses message elements to describe its input and output. Each *<wsdl:portType>* element contains only definitions of the operations an interface exposes, and not implementations. In this respect, *<wsdl:portType>* is much like an interface. This is why in WSDL 2.0 the name of a *<wsdl:portType>* element has been changed to *<wsdl:interface>*.

If *<wsdl:portType>* elements are like interfaces, the implementation of these interfaces are called bindings and are represented by *<wsdl:binding>* elements. In fact the *<wsdl:binding>* elements are the first concrete WSDL elements introduced so far. *<wsdl:binding>* elements are linked to the actual implementation of a service. Inside a *<wsdl:binding>* element, the producer of a service, references part or all of the *<wsdl:operation>* elements in the WSDL document. This way the producer of a service defines the operations that his service is going to implement. Having the binding elements defined, the next WSDL elements that need to be defined are the service elements (*<wsdl:service>)*. These elements represent the actual service that is offered. Each *<wsdl:service>* element is linked to a *<wsdl:binding>* by referring to it. This way the service specifies which operations it exposes. In addition, each service contains a number of addresses where it is deployed. A service may have more than one address through which it can be accessed. Each address is defined in its own *<wsdl:port>* element. Each *<wsdl:port>* element refers to a particular binding (interface implementation), and includes a URI in order to define how to access a service. In WSDL 2.0 port elements have been replaced with *<wsdl:Endpoint>* address elements since the latter naming reflects better that a WS is a communication point. The conceptual model of WSDL 1.1 and 2.0 is given in Figure 2-8.

From the above it is evident that WSDL and IDL are different in many respects. WSDL defines an abstract and concrete part for service reuse (an abstract part can be used by different concrete parts to define different services). IDL does not provide such features. WSDL and especially version 2.0 allows services to be more easily extended. To the best of our knowledge IDL does not provide extensibility features.

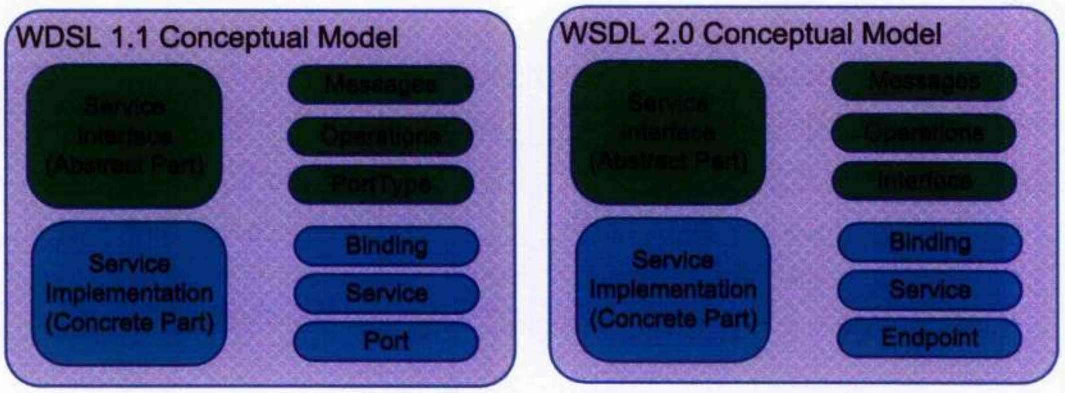

WDSL 1.1 Conceptual Model
Service Interface (Abstract Part)
Messages
Operations
PortType
Service Implementation (Concrete Part)
Binding
Service
Port
WSDL 2.0 Conceptual Model
Service Interface (Abstract Part)
Messages
Operations
Interface
Service Implementation (Concrete Part)
Binding
Service
Endpoint

Figure 2-8 WSDL 1.1 and 2.0 conceptual model

The conceptual models of WSDL 1.1 and 2.0 depict the changes in element tags that exist between the two versions. The changes between the two versions though are not just limited in the naming of these elements. One such change is that *<wsdl:message>* elements in WSDL 2.0 can no longer refer to more than one *<wsdl:part>* elements. In WSDL 1.1, *<wsdl:part>* elements allow creating *<wsdl:messages>* that can refer back to more than one data-types defined in the *<wsdl:types>* element. The WS-Interoperability [68] profile though does not allow a message element to have many children as the architects of this specification believe that the opposite would allow definition of the operations a service exposes in an unambiguous format. In addition WSDL 2.0 supports the use of other type systems apart from XML Schema. This allows for example constructs from semantic models and languages to be used to define the WS input and output data types of an operation. In essence this change supports the need of WSDL 2.0 to satisfy the WS-Interoperability [68], [69] basic profile and become more extensible. WSDL 2.0 supports two new extensibility mechanisms (a) an open content model (b) the concept of features and properties. The first allows XML elements and attributes from other (non-WSDL) XML namespaces to be accepted into a WSDL document. Even attributes from DTDs and Schemas from other service descriptions are accepted (there are specific rules on how this happens: we will see in chapter 3 how this feature and the WS-ServiceGroup specification can be used to provide collective access from one service to a series of others.). The second extensibility mechanism allows the definition of new attributes and elements inside a WSDL document. In essence both of the extensibility mechanisms are used so that WSDL 2.0 is better suited for integrating the demands of new specifications that have not been visualised when it was introduced. As part of these mechanisms WSDL 2.0 can now support more MEPs than WSDL 1.1 (8 compared to 4 [70]: to support the new MEPs WSDL 2.0 enables you to specifically state what message pattern you're using by referencing its definition over a web address) without the need of using WS-Addressing to cover some message exchange scenarios that the latter could not support. Another change in WSDL 2.0 is that the latter supports interface inheritance. The latter change increases the level of reusability when defining a service. Despite the changes performed to achieve the conformance of WSDL 2.0 to the WS-Interoperability basic profile though, some issues may arise when using different WSDL versions (i.e. v1.1). These problems have been noted [70], [71] and are the following:

- Commonly available WSDL 1.1-based proxy generators can't be used for generating the proxy stubs from WSDL 2.0 necessitating WSDL 2 proxy generators.
- Custom WSDL 2 proxy generator tools might not adhere to standards and can aggravate interoperability issues further.
- Tight coupling of the WSDL specification version on the client side for consuming the service will lead to interoperable issues as new versions of WSDL are released.

A number of solutions on these problems are provided in [70], a discussion on or evaluation of the applicability of these solutions to resolve the aforementioned problems has not been included in this thesis.

Now that both SOAP and WSDL have been introduced it is possible to explain how the style of the SOAP document and the encoding affects the body of a SOAP message. Imagine that the following Java method needs to be exposed as a service: *public void get (int x, float y).* A SOAP request message and the equivalent WSDL document using the RPC/encoded, RPC/literal, Document/literal styles are given in Figure 2-10, Figure 2-9, Figure 2-11 respectively (Document/encoded is hardly ever used).

From the Document/literal style it can be observed that the naming of the attributes of the Java method that will appear in the SOAP body, depends on the <wsdl:types> element naming. In addition, it can be observed that when using a Document/literal encoding style, the operation name in the SOAP message is lost. This can be rectified by using the Document/Literal Wrapped style of encoding (Figure 2-12). Using this encoding style the body of the SOAP message appears to be like it has an RPC/literal encoding (Figure 2-10). The wrapped encoding style is a sly way of putting the operation name back into the SOAP message. The latter style though is characterised by a more verbose WSDL document (Figure 2-12).

An important advantage of the document encoding styles is that they can be easily validated since they are based on an XML schema. Of the document styles though, only literal encodings are compliant to the WS-Interoperability (WS-I) basic profile (Document/literal, Document/literal Wrapped, RPC/literal) [69]. This is because the WS-I profile allows only one child per SOAP body element. Which document style though should it be used? Using the document wrapped style is not a good practice when a number of overloaded functions exist. The reason for this is that when using the wrapped style, the operation name and the element name of the attributes in the WSDL documents should be the same. Having multiple functions with the same name would require multiple wrapping elements for the attributes of the functions with the same name. This is not allowed in WSDL [66]. On the other hand with the Document/literal style the operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible. In addition the Document/literal style is WS-Interoperability profile compliant but with restrictions (allows for more than one root elements). As such both document styles are necessary.

In addition by observing Figure 2-10, Figure 2-9, Figure 2-11 and Figure 2-12, one can determine that the SOAP messages using literal styles are less verbose [66]. This occurs because encoding information is eliminated.

Based on the above, it seems that the Document styles of structuring the SOAP body are in most cases better than RPC ones. This is why the industry is turning gradually towards supporting only these styles in WSDL (in the WS-I profile). The only deficiency that can be attributed to these styles is that when type information is required (retrieved from the encoding info), the RPC/encoded style may be better suited. Still only the RPC/literal style is supported by the WS-I profile. Not many toolkits support this style though [148] leaving Document styles as the main option.

```
<soap:envelope>
  <soap:body>
    <Get>
      <x>5</x>
      <y>5.0</y>
    </Get>
  </soap:body>
</soap:envelope>
```

```
<message name="GetRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="Get">
    <input message="GetRequest"/>
    <output message="empty"/>
  </operation>
</portType>
```

Figure 2-10 SOAP and WSDL files for RPC/literal style

```
<soap:envelope>
  <soap:body>
    <Get>
      <x xsi:type="xsd:int">
        5
      </x>
      <y xsi:type="xsd:float">
        5.0
      </y>
    </Get>
  </soap:body>
</soap:envelope>
```

```
<message name="GetRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="Get">
    <input message="GetRequest"/>
    <output message="empty"/>
  </operation>
</portType>
```

Figure 2-9 SOAP and WSDL files for RPC/encoded style

```
<soap:envelope>
  <soap:body>
    <s>5</s>
    <t>5.0</t>
  </soap:body>
</soap:envelope>
```

```
<types>
  <schema>
    <element name="s" type="xsd:int"/>
    <element name="t" type="xsd:float"/>
  </schema>
</types>
<message name="AnyGetRequest">
  <part name="x" element="s"/>
  <part name="y" element="t"/>
</message>
<message name="empty"/>
<portType name="PT">
  <operation name="AnyGet">
    <input message="AnyGetRequest"/>
    <output message="empty"/>
  </operation>
</portType>
```

Figure 2-11 SOAP and WSDL files for Document/literal style

```
<soap:envelope>
  <soap:body>
    <Get>
      <x>5</x>
      <y>5.0</y>
    </Get>
  </soap:body>
</soap:envelope>
```

```
<types>
  <schema>
    <element name="Get">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
    <element name="GetResponse">
      <complexType/>
    </element>
  </schema>
</types>
<message name="GetRequest">
  <part name="parameters" element="Get"/>
</message>
<message name="empty">
<part name="parameters" element="GetResponse"/>
</message>
<portType name="PT">
  <operation name="Get">
    <input message="GetRequest"/>
    <output message="empty"/>
  </operation>
</portType>
```

Figure 2-12 SOAP and WSDL files for Document/literal Wrapped style

2.3.2.3 Understanding UDDI

Apart from the four core components of WS (SOAP, WSDL and a transport protocol binding), an optional but important component to build WS distributed management applications is service publication and discovery. Service publication and discovery requires mechanisms for efficient and simple discovery of the services that companies and business bodies have to offer by publishing them in a registry. In order to achieve this, a collection of interfaces defined in the Universal Discovery Description and Integration (UDDI) specification is required. UDDI builds on top of a network transport layer, the SOAP XML messaging layer, and the WS description layer. On the description layer, the Web Services Description Language (WSDL) provides a uniform XML vocabulary that describes Web Services and their interfaces. This vocabulary can be used by the UDDI [65].

The basis of UDDI is a registry. The registry holds (a) a number of programmatically accessible descriptions of businesses and the services they support (b) references to industry-specific specifications that a service might support (c) taxonomy definitions for efficient categorisation of businesses and services and (d) identification and discovery mechanisms for meaningful identification of businesses and services. As such the registry contains mainly three types of information [58]. First there is basic information about a business such as its name, the type of business it is involved with, contact information, a business identifier etc. Second there is information that categorises the business and service information and thus extends the ability of the user to find a business and a service. Lastly there is also information on how and where to programmatically invoke the services each business offers (pointers to discovery mechanisms, references to technical specifications). All the information described until now are contained in four primary data types within a UDDI registry. Figure 2-13 shows the naming of these types and the relationships they share. The *businessEntity* elements provide information about a business (service provider). The technical and business descriptions for a Web service are defined in *businessService* elements and its *bindingTemplate* elements. Each *bindingTemplate* element contains a reference to one or more *tModel* elements which are used to defined the technical specifications for a service.

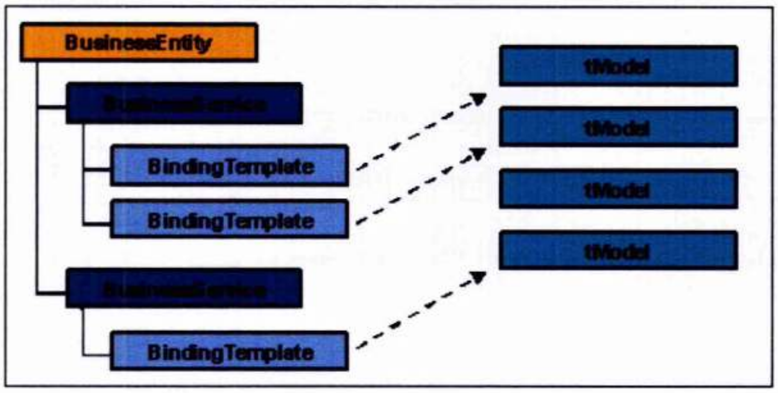


Figure 2-13 UDDI data types [73]

From what is described above, the service description information defined in a WSDL file has the potential of being complementary to the information found in a UDDI registry. Though UDDI provides support for many different types of service descriptions, there is no direct support for WSDL descriptions. Providing support for WSDL though would automatically allow users and clients of services (a) to find a business and the services they offer (b) to access information about the interfaces and the functionality a service exposes(c) to look up information on how to access a service that a business offers etc. As such the UDDI organisation has published a best practices document titled *Using WSDL in a UDDI Registry 1.05* [72]. This document describes how to publish WSDL service descriptions in a UDDI registry. In [72] WSDL files are divided in two parts (service interfaces and service implementations). Basically in this document a WSDL file is roughly divided in its abstract and concrete parts and each WSDL element is assigned to a UDDI type. Figure 2-14 shows how each WSDL element is assigned to a UDDI type. In [73], [74] examples are given on how to publish WSDL service descriptions and how a client can find these descriptions to use them afterwards in order to access a service.

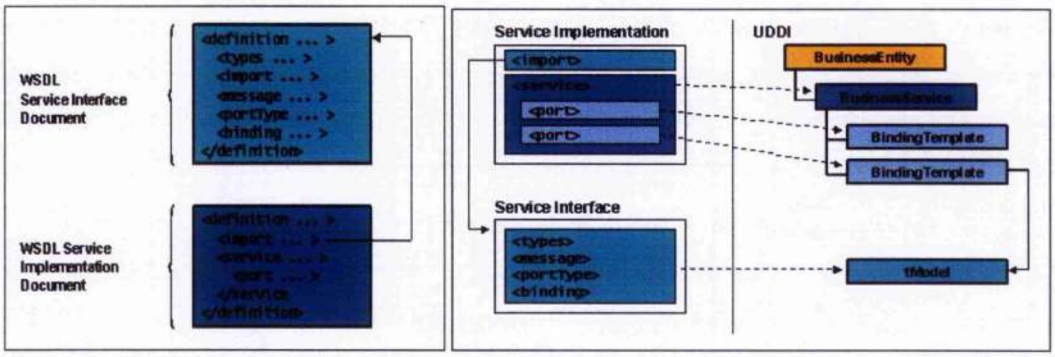


Figure 2-14 The implementation and service interface XML element tags of a WSDL file and how to assign them to UDDI types for service description and discovery [73].

2.3.3 XML and WS-based Standards and Frameworks

2.3.3.1 Service Oriented Architecture for QoS, Security and Management

Over the past few years, XML and WS have been used by various research groups to define many Web-based standards and specifications. The ultimate goal of this research was to achieve implementation of the Service Oriented Architecture (SOA) providing solutions for Network and Service Management, Quality of Service (QoS) and security in a distributed manner. The creation and support of standards is a critical component in achieving this goal [100]. The technologies that need to be standardised and implemented as part of this work involve three different sections of the conceptual WS stack in Figure 2-15.

The wire layer in Figure 2-15 involves standards and technologies required to transport messages between WS. In the wire layer, the transport sub-layer entails network connectivity using TCP/IP. The packaging sub-layer is concerned with the serialisation of the message payload. The extension sub-layer allows for extensible features to be added in the headers of a SOAP message. The common protocol used in the wire layer in order to provide the functionality of these three sub-layers is SOAP over HTTP supporting XML messaging between WS.

The description layer is based on XML Schema to describe a series of specifications in order to express all attribute types required to model a WS, the interactions of a WS with other services etc. The interface and implementation description sub-layers entail definition of the operations and messages supported by a WS, how to serialise messages over the wire, where to send messages etc. In essence these sub-layers describe the mechanics of a service. The common language used to describe the mechanics of a WS is WSDL. The policy sub-layer in the description layer consists of facts, assertions, and rules that apply to a particular Web service. Policies in essence are used to describe the requirements and capabilities of two WS endpoints when the latter want to initiate a conversation. A non normative way to describe the rules and capabilities that apply to a particular service is the WS-Policy specification [101]. In the description layer there is also a necessity to accompany a WS description document by another description document which defines how to display a service to a user and what the interactions between the service and the user should be. This is the role of the presentation sub-layer.

All the five sub-layers introduced so far in the description layer are enough to allow description of a WS but other sub-layers are also required to describe other features of WS applications. For example the composition and orchestration sub-layers describe the relationships and interactions between WS respectively. The composition sub-layer allows defining relationships such as containment, parent-child relationships and groupings that exist between properties of each WS. The orchestration sub-layer clarifies how to order the interactions between WS operations by defining choreographies, workflow charts etc. The Business Process Execution Language For WS (BPEL4WS) [102] and the WS-Transaction (consists of the WS-Coordination [103], WS-AtomicTransaction [104] and the WS-BusinessActivity [105] specifications) specifications are the most common specifications used for both orchestration and composition of WS. In the description layer the service level agreement sub-layer allows defining metrics for performance, usage and Quality of Service parameters that a service should conform to. The WS-LevelAgreement (WSLA) specification is a good example for defining the QoS parameters of a WS. The business sub-layer describes a contract between business partners having transactions using WS.

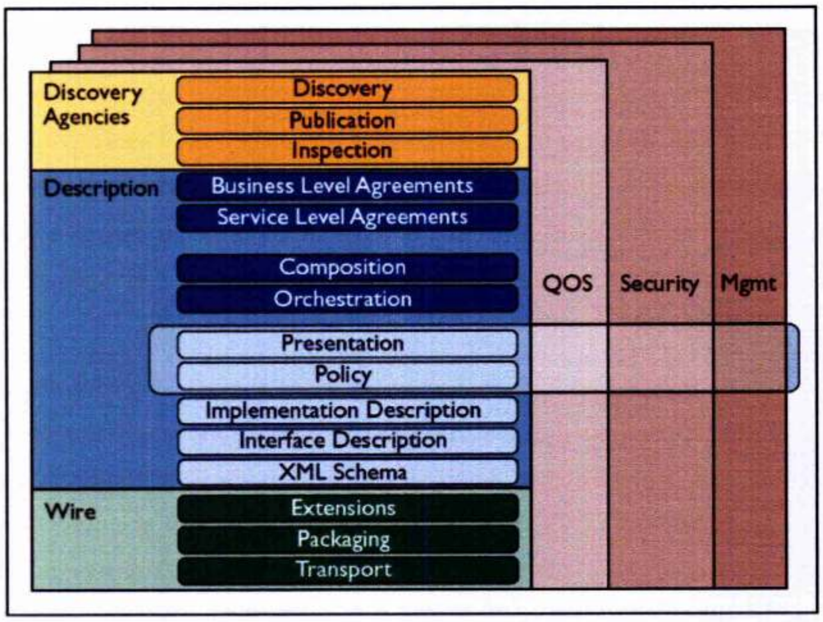


Figure 2-15 A more complete version of the WS stack [100]

Another important layer in the SOA architecture is the discovery layer. The Discovery layer encompasses standards and technologies to support publication, discovery and inspection of service descriptions. UDDI is currently seen by many as the means for discovery and publication of service interfaces. For inspection of WS the WS-Inspection specification seems to be promoted to handle this task. This specification aims at providing a standard way of indicating to web services consumers where to locate web services.

Behind all these specifications in each layer lies an overarching concern of the WS industry to supply reliable, secure and interoperable communication between distributed applications. By standardising these aspects, the WS industry hopes to provide applications that can be managed for the purposes of providing services to clients with specific Quality of Service guarantees (see Figure 2-15).

Ensuring the integrity, confidentiality and security of communication between Web Services requires the use of a comprehensive and complete security model. The latter necessitates use of a message security model (WS-Security [106]) along with a Web service endpoint policy (WS-Policy) model, a trust model (WS-Trust [107]), and a privacy model (WS-Privacy). All these specifications provide the foundation upon which it is possible to establish secure interoperable Web Services across several trusted domains. The WS-Security specification describes how to attach signature and encryption headers on SOAP messages to guarantee message integrity (XML signatures) and message confidentiality (XML encryption). The WS-Policy specification is used in the context of security in order to describe the capabilities and constraints the various WS based intermediaries in the communication path have (i.e. supported encryption algorithms). The WS-Privacy specification will describe a model on how Web Services and consumers of WS define their preferences for security. In practice WS-Privacy allows producers of services to state their security policies so as to require from incoming requests to adhere to these policies. The WS-Trust specification describes a framework that enables Web Services to interoperate in a secure manner. In essence WS-Trust describes how to establish both direct and brokered trust relationships between several parties.

Building on these four specifications, three new specifications can be defined, for secure conversations (WS-SecureConversation [108]), federated trust (WS-Federation [109]), and authorisation (WS-Authorisation). The WS-SecureConversation depicts how to manage and authenticate message exchanges between parties. This specification also defines how to establish session keys, derived keys, and per-message keys for authentication. The WS-Federation specification defines how to manage and broker trust relationships in a heterogeneous federated environment. In essence WS-Federation defines mechanisms to allow different security realms to federate, such that authorised access to resources managed in one realm can be provided to entities in other realms. The WS-authorisation illustrates how to manage authorisation data and authorisation policies to determine what an entity is allowed to do.

Combining all the above specifications leads to having a model for secure communications between WS. Nevertheless work in this domain is not final yet since some of these specifications have already become OASIS standards and some are still in the process of definition.

To guarantee reliable delivery of messages between WS, the industry came up with the WS-Reliability [110] protocol. WS-Reliability is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering. The concepts of the WS-Reliability are enforced as extensions to the SOAP header. The WS-Reliability specification is necessary since SOAP over HTTP over TCP/IP is not sufficient when an application-level messaging protocol must also guarantee some level of reliability and security. This happens because in a few scenarios, the TCP protocol cannot completely guarantee delivery of a message to a remote peer. This is true for example in the case of intermediaries where it is possible for a message to arrive successfully at the final station but it never gets processed by the appropriate SOAP processor i.e. a WS. In this case, the SOAP message is successfully delivered by the transport protocol but the message cannot be processed properly because of some errors [56]. As such there is a possibility that a message never gets processed by the final remote peer and if the sending entity closes down the connection, it will not receive an error notification. The WS-Reliability specification was introduced in order to minimise phenomena such as the latter from happening with the added cost of extra complexity. WS-Reliability was designed to be independent from the underlying transport protocol. Bindings with the underlying transport though have to be present for interoperability purposes. A common binding of WS-Reliability with a transport protocol is HTTP over TCP/IP.

All the above standards represent the Service Oriented Architecture (SOA) covering aspects such as security, reliable messaging, QoS etc. From all this work in standardisation of WS operations, it is evident that the WS industry tries to prepare WS in order for the latter to become a platform for application integration that could be used for Service Management. At the same time though, all these WS standard specifications could be considered for Network Management. In the next sections we will see how and also analyse the first approaches towards this direction.

2.3.3.2 Using WS for Network Management the CORBA way

Having introduced SOAP, UDDI and WSDL, a simplified version of the WS stack to support the description, the discovery, the registration and the messaging functionality of WS is given in Figure 2-16. This stack supports the basic functionality required for building and deploying WS but also provides the required functionality so as to use WS for network management.
One approach of using WS for Network is analogous to how distributed object technologies such as CORBA achieve the same thing. Although WS are a not a distributed object technology [56] the WS stack can support the functionality required for network management in a similar manner

to how this is performed in CORBA. To identify how CORBA and WS can fulfil the requirements of distributed management of network devices, these requirements have to be analysed. As such in order for a distributed technology to be used for network management, it has to fulfil amongst others the following important characteristics:

- Tools to describe the interfaces between communication entities.
- Hide the complexity of the underlying layers from the management application (protocol independence- applications operate in the same manner regardless of what protocol was used for communication).
- Mechanisms to access communication entities in a remote fashion.
- Communication protocols that support various MEPs to facilitate the communication between management entities (synchronous or asynchronous communication).
- Mechanisms to register the services offered by a management server so that clients can find them and consume them (access and location transparency).

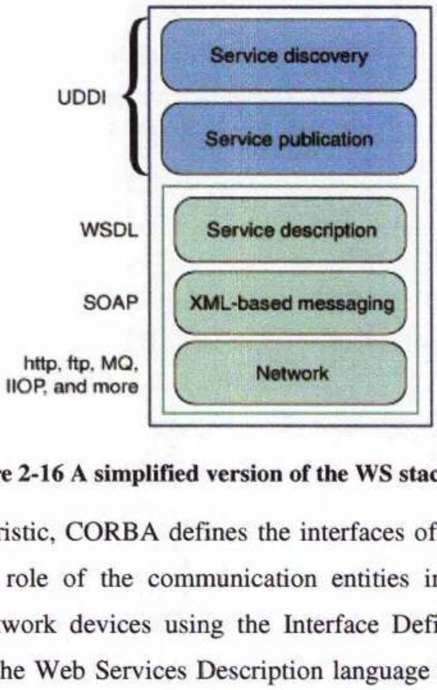


Figure 2-16 A simplified version of the WS stack [58]

Regarding the first characteristic, CORBA defines the interfaces of managed objects (client and server objects take up the role of the communication entities in CORBA) representing the underlying resources of network devices using the Interface Definition language (IDL). In a similar fashion to CORBA the Web Services Description language (WSDL) [57] can be used to describe the interfaces between two communication endpoints, a client and a server. In WS the client and server can potentially be implemented as objects of a programming language, they can be implemented as system processes or as any other software.

As far as the second characteristic, CORBA hides the complexity of the underlying transport protocol from the application by using the Internet Interoperability protocol (IIOP) that is independent of the underlying transport protocol. In WS, SOAP is also independent of the transport protocol, supporting different protocol bindings. WS though have to support processing models where the underlying protocol can change when passing through intermediaries. As such for interoperability purposes each of the bindings that SOAP supports has to be defined a priori before used. The common binding of SOAP is HTTP. CORBA's IIOP protocol (IIOP is over TCP/IP) is broadly equivalent to the default mapping of SOAP to HTTP thus TCP/IP.

For accessing management applications in a remote fashion CORBA uses Interoperable Object References (IORs) to describe the location at which an object can be found and the protocol that should be used to access it. Thus actually IORs represent references (pointers) through which access to managed objects is provided. Uniform Resource Identifiers (URIs) are the most common mechanism used in WS to have access to a service over the internet backbone from anywhere in the world. WS also offer other more elaborate mechanisms with extra functionality as the means of accessing a service (to support synchronous and asynchronous communication). One such example is the addresses and mechanisms of the WS-Addressing specification.

In order to support communication between objects, CORBA uses the General Inter-ORB Protocol (GIOP). This protocol uses a request response MEP similar to RPC. WS support various MEPs to support communication between services. One of those MEPs is the RPC messaging model offering the means to support a request-response MEP. RPC though is not the only way WS use in order to support a request-response MEP. Actually RPC is not preferred in the WS world because it attaches predefined semantics on how to use and structure the payload of a message.

Building distributed management applications requires client applications wishing to access a server application to be able to discover where to access the latter. In CORBA the name server object is a special object to do just that. In CORBA each server object registers an IOR with the name server so that objects in client mode can look them up. This way transparent access to server objects is achieved. The same can be achieved in WS using UDDI to discover the endpoint through which a service can be accessed (section 2.3.2.3). UDDI though can offers more functionality than the name server and at the same time offers the means to a consumer to retrieve more information about a service (i.e. the service owner, the type of a service, the functionality and the specifications a service depends on, etc). In general there are several other ways to emulate the functionality of CORBA's name server in WS. One such way is through a Web Server. A Web Server using a SOAP toolkit servlet can host WSDL descriptions of a service and thus can be a simple mechanism to emulate the functionality of the name server (i.e. the Apache Tomcat [75] Web Server and the Apache Axis servlet [77], [76] combined can provide the functionality of the name server).

Based on the above observations, it is made clear that the web services basic stack in Figure 2-16 can be used to support the CORBA object model in Figure 2-5 for building distributed management applications. Nevertheless CORBA and WS also have many differences. This is mainly attributed to the large number of technologies and standardisation efforts that have been invested in WS from the industry. As such WS have differences in comparison to CORBA regarding how to achieve NSM objectives. As such WS have been enriched with features that CORBA was never designed to address (i.e. for service discovery). In the next sections we will discuss how the WS industry envisions using WS for NSM.

2.3.3.3 WS-Based Management the WS way

Over the past few years, XML and WS have been used by various research groups to define many Web-based specifications for Network and Service Management. From the work introduced by these groups so far it seems that the WS industry tries to differentiate itself in certain aspects from how CORBA was used to address NSM objectives.

One of the main attempts that differentiate WS from CORBA is an effort to model the relationship between management data representing the state of network resources and WS. Behind this attempt a deep debate rages on how to guarantee interoperability in WS-based NSM. As part of this debate, WS purists [56] believe that WS have no notion of state and that interactions with WS are stateless. Thus purists believe that there is no need to model the relationship of network resources and WS in order to manage state. On the other hand others believe that interactions with WS are stateful and try to model such interactions [80] (Grid services when the OGSI was initially introduced). As such they try to assign predefined semantics on how to manage state. Others including [22] and [81] lie somewhere in the middle between these two views. In [22] and [81] authors acknowledge the critical role that state plays in distributed management and other disciplines. The latter group believes that managing state should be addressed within the Web Services architecture. The basis for this argument is that "a stateless WS implementation may act upon stateful resources since it may frequently interact with, and cause updates to, dynamic state that is maintained in other system components". In these cases, the identity of the state element(s) may be either passed in the request message or maintained as static data by the Web service. Thus the interface offered by such a Web service "is clearly stateful, in the sense that its behaviour is defined with respect to the underlying state" [81]. Lately the WS NSM industry has adopted the latter view for managing the state of WS representing the underlying resources. On the contrary CORBA and other distributed management approaches, used in the past GIOP and RPC to manage state. RPC messaging provides a network abstraction for executing procedure calls in a programming language in a remote fashion. To do this RPC offers mechanisms to identify a remote procedure, deciding which state must be provided to the procedure at invocation time, and what form to use to present the results to the invoker of the procedure at the time the call reaches its completion point. Thus RPC attaches predefined semantics in managing state (with RPC synchronous interactions across wide area networks is very difficult, large scale versioning is very difficult, interoperability issues over wide area networks arise). WS try to avoid that by decoupling the WS interface from the stateful resources representing the capabilities of the underlying resource. This way a WS remains stateless while it is possible to keep track of state in a separate entity (the resource) which is important for some applications. This promotes the building of loosely coupled WS applications, the latter making it easier to recover from partial faults when managing state, which in turn increases reliability [179]. Also scalability increases since WS do not have to keep state between requests and can free up resources or because WS do not have to manage state information across requests [179]. In theory statelessness can help solve also some interoperability issues [56].

A second characteristic that differentiates CORBA from WS is how the latter tries to model and group the resources properties and attributes representing the state of a device. For WS the Web Service Resource Framework is a specification that defines a standard approach on how to manage resource properties and attributes representing state, and to how to group them in order to provide collective access to state. JIDM was probably the equivalent move in CORBA. The approaches taken in these specifications are completely different. WS provide collective access through WS composition and grouping resources at the description layer (WSDL). CORBA groups state data of resources at the implementation layer.

A third characteristic that differentiates CORBA from WS is service discovery, description and inspection. Both WS and CORBA offer facilities for discovering services in order to access them in a transparent manner. Contrary to CORBA though, WS have built an entire framework around how to provide metadata about services describing the details of each service, the owner of a service, dependencies and interactions with other services etc. UDDI simply provides more functionality than the name server (i.e. for service discovery, for informing consumers where services are offered).

In addition to the previous and contrary to what CORBA has done in the past, the WS-Industry has issued two specifications to promote building interoperable applications. The WS Architecture (WSA) [92] and the WS-Interoperability (WS-I) profiles [68] are specifications that try to provide the means of interoperation between different WS based applications. WSA provides a common definition of a Web service, and defines its interactions and relationships to other components of the Web Services framework. As such, it tries to guide the community on how to build loosely coupled interoperable applications. To do this the WSA describes the minimal characteristics that are common and should be used by all Web services, and a number of characteristics that are needed by many, but not all, Web services. On the other hand the WS-

Interoperability profiles try to promote a set of conventions and guidelines and how Web services should be built and communicate with each other in order to achieve optimum interoperability.

These among other things differentiate CORBA from WS in using the latter for distributed management. The next sections describe how the WS industry envisions using WS for management.

2.3.3.3.1 Managing state using the WSRF framework

Based on the previous section, it is evident that the WS industry acknowledged the need of modelling stateful resources within a Web Services framework. As a result, mechanisms to enable the discovery, introspection and interaction with stateful resources in a standard manner were devised. These mechanisms of interacting with resources allow a WS to remain stateless while it is possible to keep track of state in a separate entity (the resource) which is important for some applications. At the same time only the interface of a WS becomes stateful and is standardized leaving other aspects of a service undefined. This promotes the robustness of WS applications since loosely coupled management applications can be built making it easier to recover from partial faults when managing state which increases reliability [179]. In theory statelessness can help solve also some interoperability issues [56] (like large scale versioning since the internal implementation aspects are left undefined) but the future will confirm this aspect. Two moves have made considerable impact on managing state in a WS environment; the Open Grid Services Infrastructure (OGSI) and the WS-Resource Framework (WSRF) [84]. Both frameworks define mechanisms to model several aspects of managing state. WSRF is more complete than OGSI, since it partitions the functionality required for managing state into several specifications, works with standard WSDL 1.1 etc [90].

The WSRF is a framework that defines mechanisms and constructs to enable WS to access state in a consistent and interoperable manner. It does this by proposing a set of mechanisms and conventions for managing state through stateful resources. To retain the WS characteristic of being stateless, WSRF proposes to distinguish WS from resources and model the relationship of the former with the latter. This is done by introducing the WS-Resource [85] construct. WS-Resources are constructs providing the means of declaring and implementing an association between a WS and a number of stateful resources. WSRF represents the relationship between a WS and stateful resources in terms of the *implied resource pattern*. The latter in simple terms is a WS-Addressing [19] endpoint with a number of metadata called End-Point References (EPRs) that contain identifiers to stateful resources.

WSRF partitions the functionality required for managing state into five interrelated specifications. Each specification tackles a different aspect of managing state. The WS-Resource standard [85] mentioned previously analyses how to associate a WS with a resource. The WSRF also proposes mechanisms to model the state of resources. This is performed with RP documents. The WS-ResourceProperties (WS-RP) standard [86] defines how RP documents expose a number of publicly available properties of a resource to users (consumers) that want to access, change or delete them. WS-RP also defines that the implementation of RP documents is resource specific. As such the implementation of a RP document may not be a document instance of a schema as commonly imagined. Other implementations may chose to dynamically construct the RP elements from data held in programming language objects. Any implementation of an RP is allowed. Linking a RP document with a WS interface is also necessary. According to WS-RP this is performed using an attribute declaration inside the WSDL portType definition of the WS interface. Having defined how to associate a RP document with a WS, WS-RP also standardises how to retrieve data from RP documents. The latter is performed by using a set of standard operations in order to access, and change the state of a resource in terms of its properties.

The WSRF also provides mechanisms to manage the lifecycle of a resource, to group resources, and a common way to report faults. The WS-Resource Lifetime (WS-RL) [87] specification standardises the message exchanges that need to take place in order to destroy a WS-Resource. In general WS-RL gives guidelines in order to manage the lifecycle of a resource. Apart from managing the lifecycle of resource an important aspect according to the WSRF in managing WS-Resources is how to group stateful resources in order to provide collective access to them. The WS-ServiceGroup (WS-SG) [88] specification provides the means to build WS offering collective access to properties of various WS-Resources. This is performed using membership constraints based on the properties that resources share. WSRF also recognises that when exchanging messages with WS-resources, faults can occur. The WS-BaseFaults (WS-BF) [89] standard describes a base fault type for describing errors produced when exchanging messages to access resources. Message exchanges for reporting and managing faults though are not defined. This is provided by other standards like WS-BaseNotification [91]. The WS-BaseNotification standard defines how events are filtered and delivered to their recipients, supports brokering relationships between consumers and producers of events when managing faults etc.

2.3.3.3.2 Web Services for Management specification

Based on the concepts of the WSRF, the Distributed Management Task Force (DMTF) has introduced a variety of WS-based management specifications [11], [15]. DMTF has invested considerable research effort in service management and has designed specifications such as the Web Based Enterprise Management framework (WBEM) [11]. The latter is a framework that encompasses a set of management technologies developed to unify the management of distributed computing environments and devices. A key aspect of the WBEM framework is the Common Information Model (CIM) [15]. The latter provides a set of generic classes from which application-specific information models are derived. DMTF has devised a mapping of CIM to a web services environment for the purposes of the WBEM framework. The WS-CIM specification [79] provides a simple way to map the vast management information available by DMTF to Web Services formats by enabling automatic translation of CIM management data. DMTF performs this by mapping the CIM (classes, properties and methods) to XML Schema and WSDL descriptions via an explicit algorithm programmed for automatic translation.

DMTF has also defined the Web Services for Management (WS-Management) specification [12]. The latter specifies how to identify a manageable resource and how to initiate communication with it. As part of this work DMTF has also devised a binding of CIM to WS-Management [78]. This specification describes how to communicate using the WS-Management framework operations with resources modelled with WS-CIM.

WS-Management recognises the need to manage state through stateful resources and defines its own conventions and operations based on the WS-Transfer specification [22]. In essence, WS-Management describes a general WS protocol based on SOAP that tries to address several aspects of the management process. As such WS-Management tries to tackle aspects such as (a) how to access, modify and delete single or multiple instance XML management data representations in an interoperable manner (b) security (c) event reporting and (d) fault management. To address these domains, WS-Management depends on many other specifications. As a result WS-Management uses specifications such as WS-Addressing [19], WS-Eventing [20], WS-Enumeration [21], WS-Transfer [22] and WS-Security [23] for several aspects of the management process.

The basis of WS-Management is the WS-Transfer specification [22]. WS-Transfer defines a set of generic operations for acquiring, deleting and changing XML-based representations of resource data offered by managed systems such as PCs, servers, routers etc, through WS interfaces. WS-Transfer defines two operations for sending and receiving XML representations of management data. Another two operations allow creating and deleting these representations. WS-Transfer defines that these operations are exposed as methods of two constructs that model the relationship between stateful management data and WS. Both of these constructs represent the need to manage state through stateful resources. WS-management uses these constructs to define its own conventions for managing resources in a similar fashion to the concepts of WSRF. The two constructs are (a) the *resource entity* and (b) the *resource factory entity*. The first construct represents WS offering access to XML representation of management data (resources) through EPRs of the WS-Addressing specification. The second construct represents WS that can be used as factories for creating new resources from XML representations of management data.

In addition to trying to model the relationship of WS with resources for interoperability, WS-Management tries to standardise a number of operations for managing the event reporting procedure. For this goal WS-Management uses the WS-Eventing specification [20]. The latter is a specification for subscribing and receiving events that carry information signifying that something worthy that needs to come to the attention of the receiving entity has happened (i.e. a fault when accessing or trying to modify the state of a resource). WS-Eventing supports a) discovering the event-types an event producer supports (b) discovering events a producer of events currently holds, (c) subscribing to an event, (d) event filtering to send events only to someone that wants to receive them and not to produce events no-one wants to receive, (e) defining an expiration date for receiving events, and, (f) asynchronous style messaging through a callback address for delivering events. To support the data filtering mechanisms required by WS-Eventing, the latter makes use of the XML Path language (XPath) [82], [83], which is a language used to select different portions of an XML document based on some constraints (merging and filtering constraints). To support an asynchronous style of delivering events, WS-Eventing uses the WS-Addressing specification. The latter is used as a callback mechanism by the entity (i.e. a WS) producing an event to call back the entity (i.e. a WS) that needs to receive the event. This is required since the time of the production of an event is not known and as such asynchronous communication is necessary. In addition to the previous, event subscription requires mechanisms to ensure message integrity, message confidentiality, and single message authentication. WS-Management performs this by using the concepts of the WS-Security specification.

Many of the message exchanges performed using the WS-Management specification can generate faults. Having a common way to describe faults in WS promotes sharing of a common understanding and view of faults between distributed applications. This is necessary because for example different programming languages like Java and C++ can map exceptions to different fault types and this can create confusion. WS-Management adopts SOAP 1.2 faults as the base fault type for describing errors produced when exchanging messages to access resources.

Finally WS-Management also acknowledges that there are going to be occasions where a resource might have multiple instances i.e. event log data etc. The specification of WS-Management does not try to tackle how to model resources with multiple instances (as the WS-RP specification does). It recognises though that if a resource has multiple instances and provides a mechanism for enumerating or querying the set of these instances, WS-Enumeration can be used to perform the iteration. Implementation specific details though are not given.

The operations that WS-Management adopts from WS-Transfer and WS-Eventing and their description are given in Table 2-2.

Get	Retrieves resource representations
Delete	Deletes resources
Create	Creates resources
Put	Updates a resource
Enumerate	Establishes an enumeration context for modelling resources with multiple instances
Pull	Iterates over a result set with an iterator
Release	Releases the enumerator and its associated resources
Subscribe	Subscribes to receive events
Unsubscribe	Cancels a subscription
Renew	Extend a subscription
SubscriptionEnd	Warns the event receiver that subscription is ending
Acknowledge of Delivery	Event receiver acknowledges receiving an event
Refusal of Delivery	Event receiver responds with a fault instead of acknowledging receiving an event

Table 2-2 WS-Management operations

2.3.3.3.3 The Web Services Distributed Management specifications

Another important move in the world of NSM is that made by the Web Services Distributed Management (WSDM) group. This group has issued two specification documents; (a) the Management Using Web Services (MUWS) specification [13], [14] (b) the Management of Web Services [24] specification (MOWS). The former details how to manage the resources of devices with the use of WS, and the latter how to manage the WS endpoints through which WS interfaces are accessed [93].

MUWS similar to WS-Management also recognises that the interface maintained by a WS is stateful, in the sense that its behaviour is defined with respect to the underlying state. MUWS adopts the concepts of WSRF for managing the state of resources. Based on WSRF's concepts, MUWS introduces the concept of the manageable resource which is a refinement of a WSRF resource. MUWS defines that a resource is manageable when it exposes a set of manageability capabilities. The latter is a set of resource properties, operations, events, metadata describing the specific behaviour of a resource and defining its ability to be managed.

According to MUWS, manageability capabilities are separated into (a) common manageability capabilities and (b) resource specific manageability capabilities. MUWS defines a set of standard model elements to describe common as well as resource specific manageability capabilities (properties of a resource). It also allows resource specific models such as the information models of CIM or SNMP to describe resource properties as a set of resource capabilities.

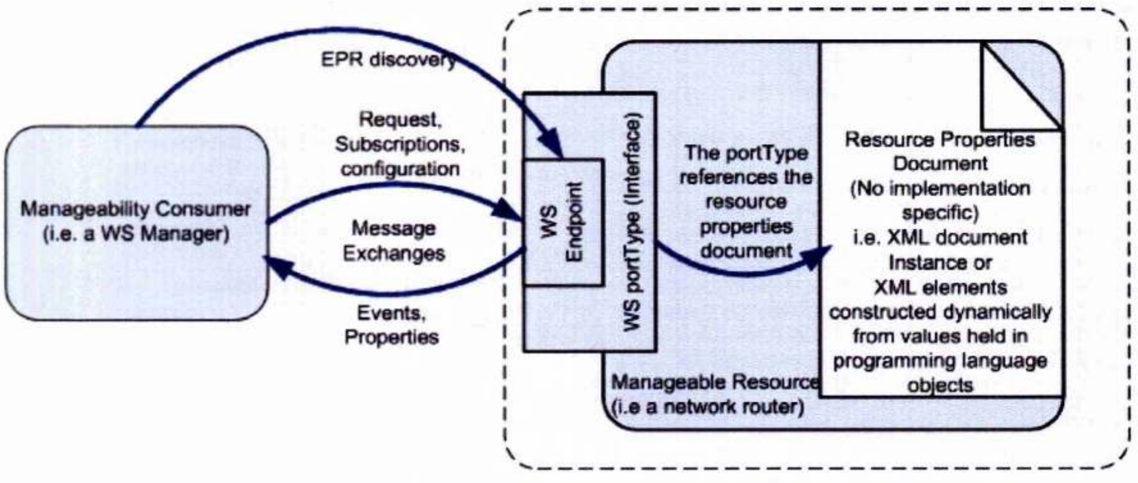

EPR discovery
Request, Subscriptions, configuration
Manageability Consumer (i.e. a WS Manager)
Message Exchanges
Events, Properties
WS Endpoint
WS portType (Interface)
The portType references the resource properties document
Resource Properties Document (No implementation specific) i.e. XML document Instance or XML elements constructed dynamically from values held in programming language objects
Manageable Resource (i.e a network router)

Figure 2-17 The concepts of the WSDM architecture

GetResourceProperty	Retrieves properties requested explicitly
GetMultipleResourceProperties	Retrieves multiple properties requested explicitly
QueryResourceProperties (QRP)	Retrieves selectively or in a bulk manner a number of resource properties using a query language (e.g. using XPath)
QueryRelationShipsByType (QRBT)	Retrieves information from a particular relationship type a resource shares with other resources
SetResourceProperties	Modifies (inserts, updates, and/or deletes) the specified properties of a resource
SubScribe	Requests that specific notifications are sent to an event consumer
GetCurrentMessage	Requests from the producer of notifications for a resource to send the last message on an event topic
Notify	Notifies a consumer about an event
PauseSubscription	Pauses a subscription to receive events
ResumeSubscription	Resumes a subscription to receive events that has been paused.
RegisterPublisher	Creates the registration of a resource as a notification publisher at an event broker
Destroy	Destroys the registration of a resource at an event broker

Table 2-3 MUWS operations that a WS interface must implement to manage resources

The basic concepts behind the MUWS architecture are shown in Figure 2-17. In this figure the WS endpoint is called a manageability endpoint and it provides access to a manageable resource through a WS interface. The WS interface is linked to the RP document's definition (its XML schema) by referring to it with a special attribute defined in the portType element of the WSDL document that describes the WS-interface. This way a WS-Resource is formed. From Figure 2-17 we can also observe that the RP document has no specific implementation. This permits the RP documents to have resource specific implementations. As a result the manageability consumer does not have to worry about the implementation of the WS-Resource. Therefore the only thing the consumer has to perform is to invoke a set of standard operations to access the properties of a WS-Resource in an interoperable manner. These operations allow the consumer to request information about the resource, subscribe to events, or, control the resource. Most of these operations are inherited by WSRF and the WS-BaseNotification [33] standard, the latter being used for manipulating event information within the MUWS framework. MUWS also defines a new operation in order to access the relationships that resources share. The operations that MUWS supports and their description are given in Table 2-3.

2.3.3.3.4 Converging the various Standards

HP, IBM, Intel and Microsoft are companies behind many of the management specifications introduced previously that define several aspects of WS-based applications such as notifications, management and resource handing. In March 2006 these companies decided to develop a common set of specifications for resource handling, eventing and management to promote interoperability. This will be performed by building on existing specifications and defining a set of enhancements that enable the convergence of their standards. The new specifications that will emerge from these enhancements will be designed to be extensible so as to cover for aspects that were not conceived at the time of their introduction. The areas where changes will occur and which all companies agreed to address include (a) resource management (b) events and notification management (c) and WS-based management [94].

In the area of resource management the companies agreed to extend and support WS-Transfer and WS-Enumeration with two specifications (WS-Transfer Addendum and the WS-ResourceTransfer) and a newer version of an existing specification (WS-MetadataExchange (WS-MEX) [96]). WS-Transfer Addendum extends WS-Transfer by revising the operations of the latter in order to allow a user to specify a subset of a resource to be retrieved, updated or changed as it may not be necessary in some cases to handle the entire state of a resource. All companies also agreed that the new WS-MEX specification will need to support exchange of metadata between WS applications through the *Get* operation of WS-Transfer. In the past WS-MEX defined its own operations for how metadata can be embedded in WS-Addressing [98] EPRs, and how metadata could be retrieved from a Web service endpoint. These operations though did not promote interoperability. As part of making the new WS-MEX specification more interoperable, the latter now defines also how metadata associated with a Web Service endpoint can be represented as WS-Transfer resources. To do this, the new WS-MEX specification introduces the mex:Metadata element. This is a new element contained inside WS-Addressing EPRs so as to provide an interoperable way to convey metadata as WS resources. As part of all the changes for resource management all companies agreed to jointly develop a new specification known as WS-ResourceTransfer (WS-RT) [95]. The latter borrows advanced concepts from the WSRF and extends operations Create', Get', and Put' of WS-Transfer to support creating, retrieving, and updating partial elements of a resource. This will result in having improved performance since it may not be necessary in some cases to retrieve the whole state of a resource as it may be very large to retrieve or update. WS-RT also borrows from WSRF the concept of managing the lifetime of a resource. Figure 2-18 shows the support of all companies to existing standards for resource management (shaded blocks represent jointly agreed upon specifications).

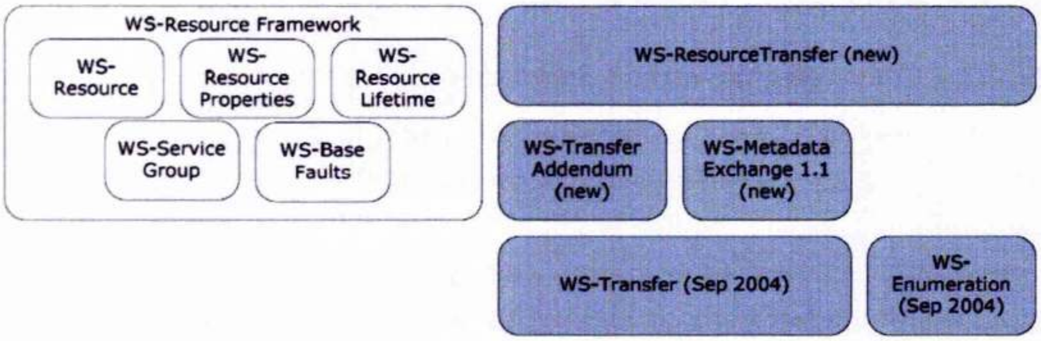


Figure 2-18 Jointly supported specifications in the area of resource management [94]

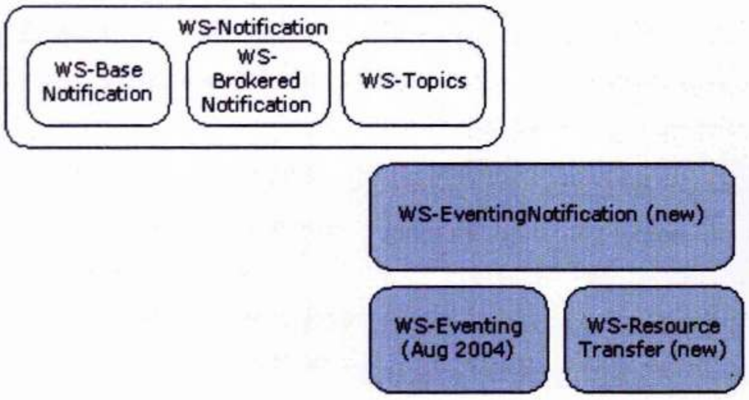


Figure 2-19 Jointly supported specifications in the area of event management [94]

All companies also agreed in defining a new specification for subscribing and receiving events based on concepts of the Web Services Base Notification (WS-Notification). The new specification will be called the WS-EventNotification and it is based on extended concepts of the

WS-Notification. WS-EventNotification is a superset of WS-Eventing, and uses WS-RT to support a WSRF resource model for managing event subscriptions. Figure 2-19 shows the support of all companies to existing standards for event management (shaded blocks represent jointly agreed upon specifications.)

Building on the joint work in the area of information distribution and event notification a new common WS management specification is being designed by all companies. This new specification is based on the WS-ResourceTransfer and WS-EventNotification specifications. Figure 2-20 provides an overview of the new specifications and their relationship to existing specifications for management. In essence Figure 2-20 shows that the reconciliation of the resource management and event/notification specifications of all these companies enables reconciliation of many of the functions of the management specifications. The latter promotes building interoperable management applications.

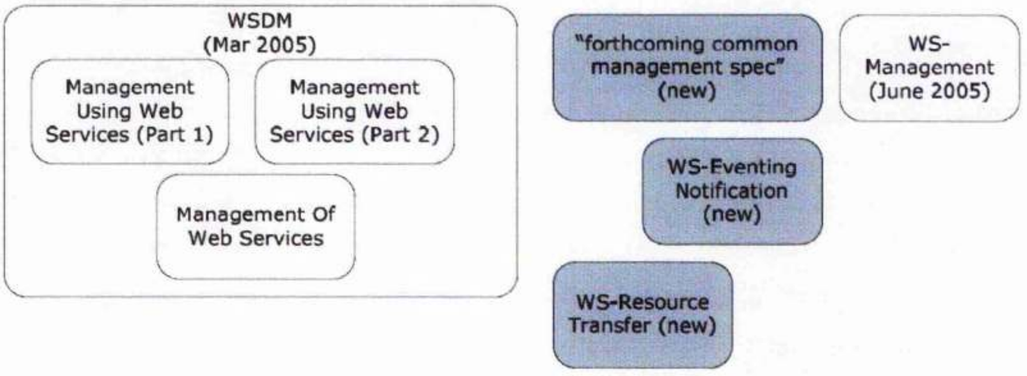

WSDM
(Mar 2005)
Management Using Web Services (Part 1)
Management Using Web Services (Part 2)
Management Of Web Services
"forthcoming common management spec" (new)
WS-Management (June 2005)
WS-Eventing Notification (new)
WS-Resource Transfer (new)

Figure 2-20 Jointly supported specifications in the area of WS-based management [94]

2.3.3.3.5 XML-based Configuration Management

Complementary to the work of all the previous industry companies for WS-based management, is another XML-based approach for configuration management. The Network Configuration protocol (NetConf) is the result of this work [25] trying to address the shortcomings of SNMP such as transaction support and security. NetConf uses a set of predefined operations (edit-config, copy-config etc) to change some of the configuration parameters of a managed device. This is achieved by uploading or configuring at the agent of a managed device a new configuration stored in an XML document. When this configuration document is completed, it is then parsed by an agent that enforces if possible the new configuration values. To enable transaction support for NetConf, its architects allow the document containing a device configuration to be retrieved, deleted, copied, enabled, locked, revoked etc. The architects of NetConf also introduced security features in NetConf through the transport mappings of the latter. NetConf supports currently three transport mappings, NetConf over SSH [27], NetConf over BEEP [28], NetConf over SOAP [26]. All transport mappings support security features to ensure authentication, data integrity and confidentiality. Work on NetConf itself has been finalized. The NetConf group though has also associated itself with work on associated data models to be used for configuration management (event models, information models). This work is still in the process of definition.

2.3.4 Research in WS-based Network and Service Management

From all the above standards and specifications that we have introduced in the previous sections it is evident that a lot of work and research effort has been invested into evolving XML and WS based standards for network, service and system management. WS and XML have great potential in being used for Web-based management but this potential raises also a few concerns. WS face (a) potential problems for dealing with the strict performance requirements of NSM (b) interoperability problems when building WS management applications (c) modelling and standardisation problems when translating information models and operations from other technologies to WS etc. An important concern about the use of WS for NSM is that WS may not be able to support efficient and optimised mechanisms for accessing, deleting or modifying management data representing the state of a device. A second concern is how to perform WS-based NSM, and at the same time be backwards compatible with previous management technologies such as SNMP. There are other problems with the use of WS for NSM but many research papers have focused in the past on evaluating the performance of WS-based NSM, some of which are also trying to provide solutions for backwards compatibility with previous management technologies. These papers can be broadly classified in five categories (a) performance of SOAP messaging and means to improve it (b) WS-based network management monitoring (c) WS-based event reporting (d) XML/WS based gateway schemes for backwards compatibility with SNMP for monitoring or event reporting (e) XML-based configuration management based on NetConf.

2.3.4.1 SOAP messaging performance

Many researchers have studied the performance of SOAP communication in the past. All of them reach to the conclusion that the performance of SOAP depends highly on the following 4 factors (a) software and toolkits used (b) parsing techniques (c) compression techniques (d) encoding and serialisation styles.

Regarding the first factor, the authors in [111] conclude that some web servers used for deploying toolkits for SOAP messaging or WS are better than others (i.e. Apache Tomcat and Sun Server are better than the IIS server of Microsoft). In addition to the previous, some toolkits for building and deploying SOAP messaging applications or WS perform better than others [112], [113], [114] (i.e. Apache Axis for Java [76], [77] is better than SOAP RMI [146] or SOAP Lite [147] or gSOAP [145] for C++ is better than the C# toolkit). This is attributed to the different parsing, binding, and serialization techniques etc that each toolkit uses. As such the choice of software and SOAP toolkit can make a dramatic difference in the performance of SOAP-based communication.

In relation to the previous, the techniques of parsing the XML documents carried within the body of a SOAP message also affect performance. Using general purpose XML parsers that have no knowledge of the structure of the data that needs to be processed and encoded, is a bad practice that deteriorates performance. Using *schema specific* parsers based on the data that needs to be encoded can seriously increase performance [115]. In addition, minimising the number of times a document has to be parsed and validated compared to its schema for the purposes of SOAP messaging also affects performance. In the past many parsing schemes were operating by parsing a document's XML schema for validation of every request to access XML data. It was only quite recently, for example, that suggestions in [116] to parse XML schemas only once have been adopted by the JAVA API for XML Processing [161].This move minimises encoding latency and increases performance substantially. In addition pull parsing techniques suggested in [114] in order to optimise WS performance in cases where the XML elements of a document need to be accessed in succession or when some element have been parsed before and do not need to be visited again, have been adopted by the Apache AXIS 2.0 and Codehaus XFire Toolkits. These SOAP toolkits use the Streaming API for XML processing (StAX) pull parser to efficiently split an XML stream into small sized chunks. As such they can build a partial XML infoset tree in memory in an incremental manner, allowing applications to start processing the XML content even before the entire document has been parsed.

Performance also depends on compression techniques. In the past general approaches to compression were used for compressing XML data with SOAP. These schemes were not very efficient. As such, this affected performance of SOAP communications greatly. The idea behind general approaches used in the past was that of coding the symbols that appear in an XML message according to the rate they appear within the message. A bigger number of bits are used for rare symbols and a smaller number of bits for frequent symbols (entropy coding). A new idea that increases the compression rate and minimises time overhead, is source encoding borrowed from the field of image, video and signal processing. One method of source encoding that can be applied to compression of XML data is differential encoding. Differential encoding schemes encode only the changes between a SOAP message sent at a moment in time $t+1$ and a previous message sent at time t. This reduces compression time [118] but also improves traffic overhead. Approaches using differential encoding are gaining ground and increase the performance of SOAP communications.

Another issue in SOAP performance is encoding and serialisation. As explained previously there are four styles of encoding RPC/encoded, RPC/literal, Document/literal, Document/encoded. Literal approaches produce less verbose documents and thus decrease latency and traffic overhead and as a result increase performance. Document styles can also be easily validated. Thus literal approaches should be the norm for building SOAP messaging applications [119], [113] especially since the WS-Interoperability profile mandates their use (Document literal, Document/literal Wrapped). Serialising data in XML, especially binary data is also an issue that affects performance. The use of MIME encodings to serialise binary data increases processing cost, code size and decreases performance. The use of DIME binary encodings as suggested in [116] can overcome these problems. DIME encodnigs are now supported in commercial and some open source toolkits (i.e. the Java API for XML Web Services (JAX-WS)). AXIS 2.0 and XFire toolkits also support the JAX-WS API. One of the most interesting features of Axis 2.0 and its AXIOM object model (AXIs Object Model), is its built-in support for the W3C XOP (XML-binary Optimized Packaging) and MTOM (Message Transmission Optimisation Mechanism) standards used in the latest version of SOAP attachments. These two standards work together providing a way for XML documents to logically include blobs of arbitrary binary data into SOAP messages. XOP and MTOM are crucial features of the new generation of Web services frameworks since they finally provide interoperable attachment support and end the current problems in this area [125].

By adopting all these optimisations the peformance of SOAP toolkits has increased. Quite recently in a comparison [97] between Apache AXIS 2.0 (1.4 version) and AXIS 1.4, the authors showed that using the AXIOM object model and StAX for pull parsing, and the ADB(Axis 2 Data Binding) or XML Beans binding framework for serialising data, has increased the latency performance of AXIS 2.0 by 3 to 5 times compared to AXIS 1.3 (this work extends the research work in [114]). In addition the same authors in [97] have shown that the ADB binding framework for serialising data compared to the Java API for XML Binding framework (JAXB) is better. This is the main reason why AXIS 2.0 is superior to XFire [40]. Currently these two frameworks are some of the faster WS toolkits available, and quite recently the XFire group has joined the Apache Software foundation. This is the main reason we use in the measurements in chapters 3, 4, 5 and in [180], AXIS version 1.x and 2.x since they are fast toolkits supporting the Java APIs we use in our work.

The performance of WS and SOAP though, is not just influenced by the parsing and serialisation techniques used in SOAP/WS toolkits to handle the XML infoset, but also in terms of the processing required to perform on this infoset. In our research in chapter 4 we will show that processing XML data, makes the performance of the XML Path language used to process and alter management data stored in XML to be worse compared to a custom query toolkit we have built that processes raw data instead. In chapter 4 we will show that the overhead introduced by processing management data is equally and in cases where the management data volume increases more important than the parsing and serialisation overhead (XPath implmentations introduce 10-

16 times more latency than the custom query tool when the volume of information to be processed is large). As such the performance of query tools to process management data affects the perfomance of management operations such as monitoring and event reporting. This is the main drive behind building our own query tool which we introduce in chapter three.

2.3.4.2 WS-based monitoring for Network Management

A lot of researchers have also investigated the potential of WS for network and service management. Researchers in [120], [121], [122] and [123] evaluated the performance of WS against traditional management technologies such as SNMP and CORBA. Their results suggest that WS are considerably worse than SNMP in terms of traffic overhead when the number of management objects retrieved is quite small. The situation becomes better when the number of objects that needs to be retrieved increases. In terms of latency, [122] suggests that although WS require bigger encoding time, the fact that SNMP does not support caching (resulting in a new search from the start of the MIB tree every time a new request arrives) results in WS having less latency overhead than SNMP. This happens both for when the volume of data retrieved is small or large. In terms of a comparison between CORBA and WS, the former according to [120] performs better in all cases in terms of latency and traffic overhead. All papers though agree that WS can be potentially used for NSM. An investigation in [123] of the performance of the two service management standards (MUWS, WS-Management) also verifies this. In [123] the authors have tailored these standards in order to use them for the needs of network management. As such they use these standards to simulate SNMP operations (get, getBulk) and evaluate their performance against the latter when performing polling based monitoring. These experiments have shown again that WS in terms of traffic overhead perform quite worse than SNMP. Regarding response time, WS-Management and MUWS perform a bit worse than SNMP but not to a point that would prevent their use for management. In terms of memory overhead the authors conclude that the two standards require quite a lot of memory allocated to the Java virtual machine, to the web server and the agent they developed. Still the authors conclude that the use of these standards for network management is not prohibitive.

Based on the above it would appear that WS could be used for network monitoring only in cases where a great amount of data needs to be retrieved [120], [121], [122], [123], as they constitute a relatively heavyweight technology in terms of memory, latency and traffic overhead. Two factors though influencing the performance of WS in all these research papers have not been investigated. The first factor is that all researchers assume that the complexity of management operations for monitoring is limited to retrieving management data sequentially or in a bulk way. This is not necessarily correct. Sometimes monitoring can involve scenarios that require more complex operations such as information processing, selective retrieval, or bulk retrieval from various areas

of the information tree etc. Acknowledging this need OSI-SM offered facilities for bulk and selective retrieval. At the same time SNMP architects recruited the DISMAN charter to provide mechanisms to cater for such scenarios showing how important it is for a technology to provide solutions for such situations. This is important because even with simple information models such as those of SNMP, retrieving data representing a part of the state of a device sequentially or in a bulk way from an information tree may not be enough. This occurs because usually management data hide a lot of relationships with other state data. Having a complete view of the state of a device with monitoring in such cases requires retrieving several values from different parts of the information tree. This requires information processing for bulk and selective retrieval of management information based on the relationships state data share. The second factor that has been overlooked, is that all the authors mentioned so far have not investigated the performance of WS and SNMP in an environment where distributed monitoring and task delegation is required to retrieve the state of device. To show the importance of load and task distribution, SNMP architects created several distributed management extensions to SNMP since several occasions arise when the manager has to process data from many agents which can be a daunting task. In cases like this, performance of WS compared to SNMP can be different.
Having not investigated scenarios where complex operations and task delegation for monitoring are required, one can not deduce any accurate and complete conclusions regarding the performance of WS for network management. One of our goals in this thesis is to use WS for network management under an enviroment where task delegation and load distribution is required, and design and build tools that will facilitate such operation. This way we can evaluate the performance of a WS-based network management framework and extract safer conclusions.

2.3.4.3 WS Network Management Gateway Schemes and Architectures

A lot of researchers have also tried to use WS for network management but at the same time remain backwards compatible with previous management protocols such as SNMP. This is very important since most devices that can be managed already support legacy protocols like SNMP.

In [126], [127], the authors suggest the use of an XML to SNMP gateway which operates between an XML-based manager and a SNMP agent. The gateway translates the operations of one system to the operations of the other and vice versa, for polling based monitoring. The authors in these papers propose a set of guidelines for translating the Structure of Management Information (SMI) SNMP MIBs to XML schemas using a version of the smidump tool [142]. The latter is a program used to dump the contents of a single MIB or PIB module or a collection of modules to the standard output channel in a selectable output format. This format may be a simple tree of nodes, but also a format fully compliant to SMIv1, SMIv2, or SMIng or CORBA IDL or an XML Schema etc. Smidump can thus be used to convert modules from SMI to XML Schemas.

Based on this translation scheme, the authors have also built a gateway implemented using Java servlet technology. Inside the gateway, using an XML parser (DOM) and an XML document syntax selection tool (XML Path Language), the authors translate XML/SOAP operations to a series of SNMP operations for monitoring and event reporting.

In addition to the previous work, the authors in [128] have also used smidump for translating the SNMP MIBs to XML. The authors have also introduced the concept of protocol level and object level translation gateways. These two types of gateways introduce the means to map management operations of established protocols such as SNMP, into operations supported by WS-based management systems. The protocol level gateway directly maps SNMP primitives to WS operations (e.g. Get, GetNext, and Set). An object-level gateway exposes a number of WS operations to the manager allowing the latter to access specific management information from a device. Retrieving for example a specific SNMP table can be offered in an object level gateway through a specific operation offered by the gateway (i.e. GetIfTable to get the interface table of the Network Management of TCP/IP-based internets MIB [129], [130]). When evaluating the performance of these gateway schemes the authors conclude that protocol level gateways are not a viable solution since the verbosity of XML tags incur a large overhead compared to SNMP. Object level gateways are viable for WS-based management of legacy devices only when the volume of management information that should be retrieved is quite large. Extending their work on gateways the authors in [133] compare the performance of their gateway schemes with SNMP in a management by delegation environment using the SNMP Script MIB. In this paper the authors introduce a new type of gateway called the service level gateway. Service-level gateways are built to expose through WS interfaces the set of services that a MIB provides (in the paper the authors implement the operations of the Script MIB). The MIB structure in service level gateways though is not strictly followed. In the investigation performed the service level gateway consumes less bandwidth than SNMP and the other two gateway types they have investigated in [128]. Service level gateways also exhibit a response time quite close to that of SNMP, making the former as the authors claim a good candidate to perform network monitoring operations.

In addition to the previous work, the authors of [131] introduce three methods for interactive translations of SNMP operations through a gateway scheme; (a) DOM-based translation (b) HTTP-based translation and (c) SOAP-based translation. In DOM-based translation an XML-based manager calls a DOM interface that is hosted in the gateway. After this call each XML request is translated to a series of SNMP operations between the gateway and the legacy managed device supporting SNMP operations. In HTTP-based translation the gateways analyse XPath and XQuery expressions sent to them by an XML-based manager which are then translated to SNMP requests. With this scheme bulk and selective retrieval can be supported reducing the management traffic between the XML-based manager and the gateway. This process though introduces processing and memory overhead. In SOAP-based translation the gateway exports a set of sophisticated resource specific operations to the XML-based manager. Using these operations the manager can also look up management information in a bulk or selective manner using XPath expressions.

Another gateway scheme has been introduced in [139] and [140]. The gateway system supports interactions of an XML-based manager with an SNMP based agent as well as interactions between pure XML-based managers and agents. The gateway supports monitoring, event reporting and configuration operations as well as bulk and selective retrieval of management information with XPath. The authors have configured their own XML schema with a number of statically predefined data types to map management information for monitoring, event reporting and configuration to XML.

Contrary to all the previous schemes, the authors in [135] have defined an architecture to support XML-based network management in a more flexible and dynamic way. Previous systems such as the one in [139] and [140] are static XML adapters to existing subsystem interfaces. As such all the previous architectures and schemes were not flexible and modular enough to support new subsystems added in a management system. A lot of performance gains are not exploited in this way. On the contrary, the new architecture in [135] proposes to process the XML transaction that will be mapped to a managed device operation in a dynamic manner. This means that when new subsystems are added in the management system, the software supporting pre-existing sub-systems does not have to be modified. To support this, the architecture in [135] allows providing new features just by modifying or adding extensions to the XML schema supporting the features of pre-existing sub-systems. The XML schemas contain all of the information necessary to validate and route management transactions to the new subsystems and to update the command line interfaces that can support the new features. The authors claim that they have no knowledge of another system that can be updated in this fashion without modification of the common management software. The authors also promote a model for representing management information within an XML schema. Using this model they try to increase latency and traffic overhead performance gains.

All the previous gateway schemes have not addressed though security aspects. In most management protocols and frameworks security is an important aspect. The work in [132] introduces a role based access paradigm to provide security extensions for XML to SNMP gateways in order to address authentication, confidentiality and authorisation issues. The authors show how to integrate the security extensions they propose inside pre-existing frameworks that support network management operations using gateways.

All the gateway schemes introduced so far recognise the importance of remaining compatible with previous management technologies. Many of these gateway schemes also recognise that it is equally important that the performance of network management operations is not compromised.

What also matters is how these two guidelines are achieved. In the gateway schemes presented, operations of legacy protocols such as SNMP are mapped to XML/WS/SOAP operations. This increases in some cases the performance of the XML-based solutions in terms of latency and traffic overhead. Such schemes though are not flexible since in order to deploy various levels of granularity in retrieving management data, a WS interface has to expose a very big number of functions (i.e. object level gateways). This increases memory overhead. In cases where a large population of WS is required in order to represent the underlying resources, approaches such as these with a big number of functions exposed by each WS interface are not scalable. In addition, some of the gateway schemes introduced provide support for tools such as XPath or XQuery to retrieve data in a bulk or selective way trying to minimise traffic and latency overhead in cases where the entire state of a device need not be retrieved. In essence schemes that use query tools to retrieve management data in a bulk or selectively manner may be more efficient. This can be true, because when using query tools to retrieve the exact state data from the underlying device, it is possible to use a single method per WS interface to achieve the same granularity as previous researchers did with many specialised methods. As such query tool schemes can be more lightweight. However there have been various concerns in the NetConf mailing list [25] that XPath or other query tools might be too heavyweight in terms of memory and latency overhead for handling management data for configuration management, monitoring and event reporting. In the next chapters we will show that XPath can be a heavyweight tool when performing certain management operations under certain conditions. Thus alternatives should be looked at, and possibly more lightweight tools for network management should be built. At the same time the WS industry, as we saw previously in the roadmap of the MUWS and WS-Management standards, tries to promote a common set of management operations to access management information for interoperability. Thus it would be more plausible when building gateways or any WS-based monitoring and event reporting sub-systems, to support the operations of these standards when mapping WS operations to legacy devices. Any gateway-based management solution should adhere to the concepts of these standards, otherwise interoperability will suffer (object level gateways, service level gateways etc introduced in this section do not support the operations of WS management standards).

2.3.4.4 WS-based Event Reporting for Network Management

Some researchers have also investigated the potential of using WS for event reporting. In [124] researchers have evaluated the performance of the WS-Notification standard and SNMP for event reporting. To do this the authors have used SNMP traps to evaluate SNMP's notification performance. At the same time, the authors used the WS-Base notification standard and three gateway schemes so as to map SNMP traps to the operations of the former, for WS based event reporting. The first gateway scheme maps the contents of the fields of an SNMP trap directly to

WS-Notification message fields. The second gateway pushes the content of an SNMP trap in binary format inside one of the fields of the WS-Notification standard. The authors describe this mapping strategy as a tunnelling approach. This is because the gateway between the Web Service manager and the SNMP agent uses the WS-Notification messages as a tunnelling mechanism between the notifying agent and the notified manager. The third gateway acknowledges the fact that in traditional management, SNMP traps usually trigger in the notified manager a sequence of SNMP requests back to the notifying agent. This means that in such cases the manager requests from the agent additional information related to the reported event. As such, the goal of the proposed gateway is to move the SNMP interactions to receive extra information about an event closer to the notifying device, and contact the distant manager only after having collected all the relevant information related to the event being reported. Testing the three schemes the authors conclude that the direct mapping gateway and the tunnelling gateway perform considerably worse than SNMP traps. The third gateway performs better than SNMP both in terms of latency and traffic overhead SNMP. The authors conclude that the third gateway scheme has great potential. Still, this scheme is not very flexible since the whole strategy of what the gateway should retrieve from the agent when a trap is transmitted, is hardwired. A more flexible scheme would be more promising.

At the same time authors in [34] have explored the performance of the WS-Notification standard messages for event reporting against simple text based approaches such as sending events in raw XML format or sending events data in raw binary format. From their examination of the three methods for reporting events the authors suggest that WS standards are best used at the edge of a domain and not as part of the core distribution because these standards are not scalable. On the contrary, the authors suggest using custom based solutions inside a domain that are proprietary or open source for better performance.

Combining the suggestions of [124] and [34], it is evident that a custom event reporting system based on a flexible (not hardwired) logic for configuring events and managing the event process can be a promising solution for event reporting. This system should be able to send WS-based standard messages for event reporting at the edges of a domain but also send event reports in an application specific manner within a domain. One of the goals of this project was to build such a system. In the next chapters we will investigate policies and the WS-Notification standard to facilitate the design and implementation of such a system. We will then evaluate its performance compared to other event reporting systems.

2.3.4.5 XML/WS Based Configuration Management

Although NetConf is under the scope of interest of most researchers for configuration management, most of its ideas come from a previous attempt to address configuration management problems developed by Juniper Networks. The JUNOScript API [141], [157] was designed to support configuration of the state of a device by alteration of the state data of the latter. JUNOScript is an XML-based configuration protocol supported by a lightweight remote procedure call oriented model of exchanging messages, sitting on top of a connection oriented transport protocol such as SSH or Telnet. The connection oriented transport protocols allow exchanging management configuration information in a secure reliable manner with no limitations in size. JUNOScript supports both an XML messaging system of operations for configuration of devices as well as a command line interface. The latter is supported by a rendering device translating between CLI commands and XML message exchanges (operations). To support the translation of these operations, the messages containing them are processed and translated by using a proprietary lightweight XML parser that uses a subset of XML features.

Although JUNOScript was a precursor of NetConf, most research in configuration management is focused on NetConf. Various implementations of the NetConf protocol have investigated the performance of NetConf for configuration management. In [134] the authors have proposed and implemented an architecture based on NetConf to support configuration management in order to address the problems of SNMP in this domain. In [136] the authors extend their work and through implementation of their architecture suggest ways to improve NetConf's performance. In this investigation the authors conclude that the transport protocol mapping does not affect performance of NetConf in terms of traffic overhead and latency. They do propose though that compression should be used, especially when the volume of configuration data transferred with NetConf is large. This is true since encoding large pieces of configuration data in XML format is not a scalable solution. The authors also propose usage of NetConf's pipelining mechanism as the means to reduce response time, although this mechanism does not also reduce traffic overhead. Pipelining is a mechanism that allows sending a series of request messages without waiting for the response to these requests to come back. This mechanism improves latency in terms of elapsed time when a series of operations need to be performed but does not improve traffic overhead. To improve the traffic overhead of NetConf operations, the authors propose the use of a multi command operation that uses the RPC layer used by NetConf to transfer a single request message containing several operations. This saves the traffic overhead incurred from the HTTP and SOAP header data that would be introduced if invoking a number of NetConf operations using multiple request messages. In addition to the previous, the authors propose the use of XPath compared to sub-tree filtering (alternative mechanism for filtering proposed by the NetConf Charter) as a more efficient solution to process configuration management data for filtering and merging operations.

In addition to the previous work, the authors in [137] have also introduced an open source implementation of NetConf with extended features. The architecture the authors propose in this

paper supports encryption, authentication and access control as part of a global security architecture and also compression to optimise bandwidth consumption. The authors evaluate also the performance of XPath versus sub-tree filtering and conclude that when used only for filtering, sub-tree filtering is slightly better. The opposite happens when combing merging different parts of an XML configuration document with filtering operations. In the latter case, XPath is better.
A lot of work has addressed increasing the performance of XML-based solutions for configuration management. Building lightweight mechanisms for communication and promoting best practices when performing configuration operations can dramatically increase the performance of the NetConf protocol. Despite these optimisations though, there is a lot of concern regarding the performance of XPath and sub-tree filtering for configuration management. Both solutions may incur large performance overheads under certain conditions and scenarios, and as such other solutions should also be looked at.

2.4 Summary

2.4.1 Overview of Research Work so far

A lot of research has been invested in many technologies and standards in order to address the objectives of network and service management. Despite the extensive research that has taken place in the last twenty five years and the numerous standardised solutions that have been devised and agreed, the quest for an "all encompassing technology" [1] still continues. A new player introduced lately that can be used for NSM is WS and XML. WS can be used to address NSM objectives for monitoring, event reporting, configuration management, transaction support etc. A variety of standards and specifications have been introduced to address all these aspects. Behind all these specifications the WS industry tries to guarantee reliable, secure and interoperable communication between distributed applications that can be managed for the purposes of providing services to clients with specific Quality of Service guarantees. Based on all these specifications, some of which are at the stage of design implementation and some of which are at the stage of already being a standard, it is evident that WS have great potential in becoming a promising and complete platform for application integration that could be used successfully for network and service management. In order to use WS for NSM, one possibility is to use the former in a similar manner to previous distributed management technologies such as CORBA. This is true due to the similarities that WS have with the latter technology. The WS industry though differentiates itself from how CORBA was used to address NSM objectives. WS differentiate themselves by standardising aspects such as (a) how to achieve interoperability in an end to end fashion, (b) how to model interactions between services and resources(c) how to manage, represent and access the state of devices etc. Eventually it is conceivable that WS, due to the large industry support, may have the potential to solve NSM problems that other technologies in the past have not solved.

Though WS and XML may have potential in being used for Web-based management, there are several problems that need to be solved before this is even possible. One of these problems is that WS may not be efficient in accessing, deleting or modifying management data representing the state of a device. A lot of research has been invested in trying to increase the performance of WS and XML (a) by improving the performance of SOAP messaging mechanisms (b) by improving the performance of WS for network management monitoring, event reporting and configuration and (c) by improving the performance of gateway schemes for backwards compatibility with previous management technologies such as SNMP for monitoring, event reporting and configuration.

From all the research performed so far it is evident that monitoring, event reporting and configuration management sometimes require more sophisticated operations on management data. This occurs because the data representing the state of a device often share relationships even with data of other devices. Thus retrieving or altering the complete state of a managed device requires having tools that support more sophisticated operations such as information processing, bulk and selective retrieval, task delegation for distributed management, navigation of the relationships between state data etc. Some tools have been suggested to address the goal of retrieving the state of a managed device effectively for network management operations. Various concerns that the efficiency of these tools is poor have been expressed. Thus alternatives should be looked at and possibly more lightweight tools for bulk and selective retrieval of management state data should be built.

At the same time it is observed that any management solution, for network management should conform to the concepts and the standards that will emerge from the convergence of the management standards described in [94]. The fact that these standards promote the building of loosely coupled management applications and they encourage the decoupling of management applications from the specifics of implementation, promotes the use of resource specific tools and applications to manage network resources. This opens the way to provide custom solutions for network and service management. This is very important now that the role of providing custom solutions within a network domain and interoperate with standards at the edges of the network domain is promoted as an efficient way to increase performance of WS based management [34].

2.4.2 Roadmap for the rest of this Thesis

Considering all the above, this thesis looks for mechanisms that present the potential of minimising the footprint of WS-based management applications so that the latter can be used more efficiently for monitoring and event reporting. As part of the thesis work we show the importance of mechanisms used for load distribution and task delegation in minimising the performance of WS-based monitoring and event reporting applications. As such, we present and analyse two such techniques. The first is a query tool we have designed and built that can be used for efficient retrieval and processing of management state data close to the devices where these data are hosted. The second technique is policies used in order to delegate a number of tasks from a manager to an agent to make WS-based event reporting systems more efficient. As part of the first technique we will show the characteristics that our query tool and every tool should possess for efficient WS based monitoring. As part of the second technique, we will introduce our policy grammar in order to make event reporting systems autonomous and capable of performing a variety of tasks efficiently, without having to hardwire the logic and the capabilities of these systems.

To evaluate the performance of our query tool we have build a monitoring system supporting bulk and selective retrieval of management state data. This system uses the operations of a lightweight custom monitoring framework we have defined for performance and can support the Management Using Web Services operations and concepts (MUWS) for interoperability. The entire monitoring system is part of an architecture that supports distributed polling based monitoring. We will evaluate the performance of our query tool against XPath suggested by several WS management standards for monitoring, a general use XML query tool, and we will show that it is more scalable under certain situations. Based on this,we will then test our lightweight framework using our query tool against SNMP and show that it can perform better in some cases or equally good in other cases to SNMP for monitoring (when bulk and selective retrieval and load distribution is required).

We will also use the WS-Notification framework to build a WS based event reporting system that supports policies and our query tool for efficient WS based event reporting. We will test the performance of this event reporting system against another WS based event reporting system and SNMP traps and we will show that it has the potential of performing better.

Having shown that the techniques we used for load and task distribution can be used effectively for building efficient WS based management applications we will present the design and implementation challenges for building a monitoring tool to support and enhance the above systems with extra capabilities. This system is a high level manager and represents the heart of our WS-framework.

Chapter 3

3 A Custom Query Tool for bulk and selective retrieval and distributed monitoring

3.1 Introduction

Since the introduction of the Simple Network Management Protocol (SNMP) in the early 1990's and the newer versions of it that followed, its use for sophisticated network management still raises a lot of concerns [5]. Back in the early 1990's, SNMP version one was initially equipped with (a) a relatively simple information model (b) facilities for retrieving and configuring the state of a device with a series of consecutive operations (c) traps for event reporting. Versions two and three of SNMP following the introduction of version one, included enchancements to support (a) proper emulation of creation and deletion of state data (b) bulk data retrieval (c) reliable event delivery (d) security features. Despite the enchancements though, SNMP may not be suitable for some monitoring and event reporting tasks.

Monitoring and event reporting tasks have certain requirements. Sometimes in order to have a complete view of the state of a device for monitoring, a manager has to retrieve several values from different parts of the information tree (i.e. SNMP MIB) based on the relationships the data in the tree share. Even more, sometimes data have to be processed close to the device from which they are retrieved, in order to return only the part of the state of the device that a manager is interested in receiving. In addition, information processing has to be performed as part of an architecture that supports distribution of the monitoring load to several entities. This is required, first because sometimes state data need to be retrieved from several devices, and second because if the monitoring processing load would be undertaken by a single entity such as a network manager, the latter could be overwhelmed by the task at hand.

SNMP hides relationships between state data in description clauses. Although these clauses can be accessed, it is very difficult in many cases just by reading the MIB modules, to understand which data values should be retrieved for some monitoring or event reporting tasks. This occurs, because it is sometimes difficult in a SNMP MIB to identify all of the relationships that state data share. Due to the above characteristics, SNMP operations do not help a manager in exploiting the conceptual relationships that state data share for monitoring or event reporting (it is not impossible to perform certain tasks exploiting the conceptual relationships of state data in SNMP, but certainly more difficult and sometimes impractical and not scalable).

In addition to the above, SNMP does not have filtering mechanisms for information processing and in some situations its bulk retrieval mechanisms are impractical to use. The lack of filtering mechanisms in SNMP does not facilitate load and task distribution, since without such mechanisms data can not be processed close to the devices where they are hosted. At the same time, SNMP's bulk retrieval operation is inadequate in some cases. This is especially true in cases where (a) multiple instance data must be retrieved (the volume of these data changes dynamically) or (b) when the data that need to be retrieved have to be extracted in a specific order. In the former case the GetBulk operation of SNMP does not permit a SNMP manager to dictate to a SNMP agent that the former needs to retrieve all the multiple instance data of a specific type. This is because for such operations the manager is forced to know in advance how many multiple instance data of a specific type exist. Sometimes the latter is not possible since for some multiple instance data their volume changes frequently (i.e. volume of TCP Connections on a device). GetBulk and in general all SNMP operations for monitoring are inefficient in terms of traffic overhead when data have to be retrieved in a specific order. This happens because in such cases, the manager has to specify all the OIDs of the data it wants to retrieve. Even if traffic overhead was not a problem the variable-based system of SNMP MIBs does not allow a manager to know the exact order of OIDs required in the GetBulk request. This is because the ordering of OIDs in the MIB tree is lexicographical and not based on the conceptual relationships these data share. For example the order of the Logical Switched Paths (LSPs) in the mplsXCTable of the LSR MIB [152] is not depicted according to which PHB each LSP belongs to, but it is lexicographical and based on computational requirements (for efficiency, stability etc). In order for the manager to find the order that specific data are organised in a MIB table (i.e. the LSPs a PHB consists of), the manager first needs to retrieve all these data and then to process them. If these data change frequently, this is time consuming, it increases traffic overhead, and forces a SNMP manager to process data from many devices very frequently which could be overwhelming in terms of the processing power required from a single entity. If SNMP had filtering facilities, a SNMP manager would not need to put all the OIDs in the GetBulk request message to retrieve data in a specific order, neither would it have to do the processing of a number of data from many devices. One way to support filtering mechanisms with SNMP is the Script and the Expression MIB. Still as explained in chapter two, these mechanisms are insecure, and sometimes they are inefficient, impractical and provide limited support for distributed monitoring and increase in some situations the monitoring footprint instead of decreasing it.

In addition, SNMP traps in some situations do not retrieve all the management information required to describe an event [5]. This forces in many cases the SNMP manager to ask for more detailed information from the SNMP agent after receiving an event. This increases latency, traffic and memory overhead.

Based on the above, it is evident that although SNMP provides opearations to retrieve data sequentially or in a bulk manner, these operations may not be adequate under some situations for either monitoring or event reporting.

Distributed object technologies such as the Common Object Request Broker Architecture (CORBA) were considered as unifying management technologies to solve many problems that other technologies such as SNMP did not solve. CORBA offered facilities for location and access transparency, transaction support for configuration management, efficient event reporting, reduced development & operational costs and supported security features. Although CORBA has come a long way to address the shortcomings of SNMP, it still has some inefficiencies. In CORBA federation is not supported, filtering mechanisms are basic and proprietary, scalability may be an issue in terms of the large agent footprint required for deploying large object populations, and there are no facilities to support the description and composition of the services that CORBA objects offer.

Web Services (WS) is an emerging XML technology whose promise of faster product development, interoperability, application integration and industry acceptance has led researchers to consider it for network management. As explained in the previous chapter though, WS is a technology that has to solve quite a few problems before it could be used for NSM. One of these problems is the substantial overhead WS introduce for performing management operations compared to other technologies such as CORBA and SNMP. This is attributed to the verbosity of XML tags describing the context of management data. In addition, as explained in chapter two, the tools and APIs used to built and deploy WS are still in the process of development and as a result performance is inhibited. While the issues of the software performance for building and deploying WS will probably be resolved, the verbosity of XML tags will always have a negative impact on WS performance (memory, latency, and traffic overhead). But mechanisms to overcome part of this overhead can be found (i.e. minimize the processing overhead by processing raw data but still use XML to form a response to a request).

As explained previously for SNMP though, in many monitoring and event reporting scenarios it may not be necessary to retrieve the whole state of a device as it may be very large to retrieve or update. Having efficient mechanisms/tools for bulk and selective retrieval can possibly improve the performance of WS in such situations. In the past, sub-tree filtering and XPath have been suggested as tools to support merging and filtering operations on XML data for configuration management, monitoring and event reporting. There are various concerns though, as explained in chapter two, that these tools may have a big footprint on management operations under certain situations. Thus alternatives should be looked at and possibly more lightweight tools for network management should be designed and built.

For WS-based management though, the aforementioned tools should also support a variety of other features apart from bulk and selective retrieval. They should for example permit communication using standardised operations from frameworks such as MUWS and WS-Management to promote interoperability. At the same time they should permit operation as part of a custom WS framework that may be performing better in terms of scalability. Furthermore they should exploit the relationships between state data in order to search and retrieve management data more efficiently. Finally they should be part of an architecture that supports distributed monitoring for task and load delegation in order to relieve a single entity such as a manager from this sometimes overwhelming burden.

Combining all the above, we have designed, built and deployed a custom query tool to retrieve management information representing the state of a device for polling based monitoring and event reporting. The tool allows navigating the relationships that exist between state data and supports bulk and selective retrieval mechanisms in order to retrieve the state of a device more efficiently. The tool is part of an architecture and framework that encompasses a distributed monitoring system supporting task delegation of the monitoring load from a manager to a series of agents. The framework used in this monitoring system supports a small number of functions so as to be lightweight for increased WS-based monitoring performance within a network domain. The monitoring system and architecture can also be converted in order to support the standard operations and functionality of the MUWS framework for interoperability purposes at the edges of a network domain. This way performance of WS-based operations can be optimised using a lightweight WS-based framework while at the same time interoperability does not suffer.

In the next sections we present the concepts and ideas upon which our custom query tool was developed. We also analyse how the tool operates as part of a distributed polling based monitoring architecture that supports load and task distribution of the monitoring load. Finally we present how to convert our distributed monitoring architecture in order to make it conformant with the concepts of the MUWS standard for supporting distributed monitoring and interoperability.

3.2 Concepts behind state data selection based on the relationships that state data share

Exploiting the relationships that exist between management state data in order to support efficient retrieval of management information is not a new idea. In the past OSI-SM introduced the manager-agent model as the means to provide collective access to clusters of programming language Managed Objects (MOs) organised in a management information tree according to containment relationships. These objects represented the underlying device resources and collective access to them was provided through an interface hosted at an agent overlooking the device resources. The protocol used to access this interface is the Common Management Information Protocol (CMIP) [149]. Using CMIP, management applications acting in *manager* roles in OSI-SM were able to access either a single object, by using its name, or multiple objects selected through *scoping* and *filtering* parameters. Scoping in CMIP selects objects for bulk retrieval based on containment relationships starting from a particular position in the information tree. Filtering further eliminates the selection of managed objects from scoping through Boolean expressions containing assertions on attribute values. The advantage of scoping and filtering is expressive power on selecting management data as well as minimisation of management traffic.

Still CMIP allowed only for containment relationships to be navigated in order to select management data representing the state of a device. As such the authors in [150] suggested extensions to CMIP to allow scoping be used to select management data based on other relationships the latter share. The idea is simple. Since the state data representing the underlying resources share a number of relationships, so do the objects encompassing them. Using these relationships the hierarchical tree of Managed Objects Classes (MOCs) in OSI-SM can be navigated. By adding mechanisms to CMIP to allow navigation of any relationship between MOCs, the resulting CMIP++ [150] allowed the manager of a management system to impose level restrictions in the scoping process. This way the manager can indicate the starting and final levels at which objects are extracted using a path expression. In addition CMIP++ also supported cascaded relationship restriction patterns. The latter is a series of relationships that should be followed going from one object to the other in order to reach the final object to be selected. CMIP++ also supported selecting managed objects where the relationship restriction pattern cannot be followed. The latter objects are called fringe objects. Examples of the above are given below.

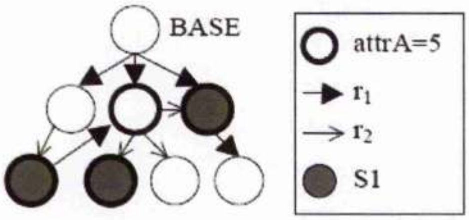


Figure 3-1 S1 = BASE.(rl.r2) where (attrA = 5) ([150]) : Searching for objects which can be reached following relationships r1 and r2 with attrA equal to 5

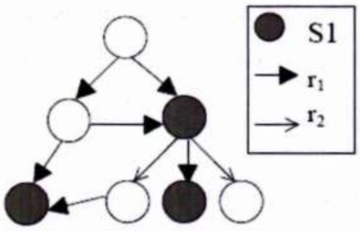


Figure 3-2 SI = BASE.(r1) [2] ([150]) : Searching for objects which can be reached following relationship r1 twice

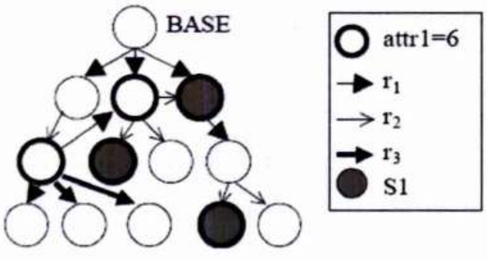


Figure 3-3 SI = BASE.(rl.r2) [2] where attr1=6 [150] : Searching for objects which can be reached following relationships r1 and r2 with attr1 equal to 6

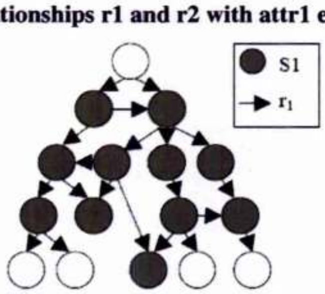


Figure 3-4 SI= BASE.(rl) [l…3] [150] : Searching for objects which can be reached following relationship r1 from 1 to 3 times

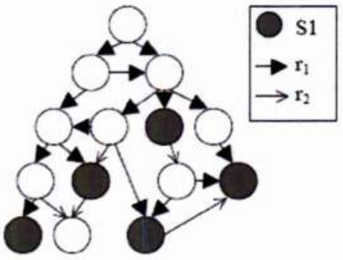


Figure 3-5 SI= BASE.(rl)! [1…n] [150] : Searching for objects where the relationship r1 can not be followed from 1 to n times

In Figure 3-1 the scoping path expression enables a manager to select only objects that can be reached following relationships first of type r_1 and then type r_2 (cascaded relationship restriction pattern). In the path expression of Figure 3-1 the search for managed objects starts from object base. Starting from base, the default level restriction is applied restricting selection of objects only to those which can be reached following the relationship pattern of the path expression just once. When object selection from scoping completes, the filtering expression at the end of the path expression in Figure 3-1 allows selecting only those objects that have attribute *1* equal to 5. In Figure 3-2 only objects that can be reached following relationships of type r_1 are selected. The search starts from object base and a level restriction is applied restricting the selection of objects, to objects that can be reached following relationship r_1 twice. In Figure 3-3 objects are selected following the sequence of relationships (r_1, r_2). The filtering path expression is applied at the end (where attr1=6). In Figure 3-4, objects are extracted between levels 1 and 3 that can be reached following relationships of type r_1. In Figure 3-5 objects are extracted between levels 1 and n where the relationship pattern r_1 cannot be followed (Fringe object selection). More examples are given in [150].

3.3 Moving from navigation of relationships between objects in CMIP++ to navigation of relationships between WS

As mentioned previously, relationships is something shared by the state data representing the underlying resources. As such objects encompassing these data also share these relationships. In the same way as for objects, WS can be used to encompass and expose the state data of the underlying resources of managed devices. Since the state data will keep on sharing relationships so will the WS encompassing them. As such the concept of scoping in CMIP++ where you select the objects from which to retrieve data from based on the relationships the latter share can also be used in WS-based management. This time though navigation of relationships will be performed to select the WS to retrieve state data from and not objects.

Having to search relationships between objects for data and having to search relationships between WS, presents a significant difference. In CMIP++ object oriented principles such as containment facilitated the structuring of state data in hierarchies with different levels of abstraction. This allowed searching for state data more effectively. WS offering access to management state data do not perform this by default. Nevertheless it is possible to structure WS in hierarchies. An example of how to organise WS encompassing management state data in hierarchies is given in Figure 3-6.

In Figure 3-6 examples of 5 types of relationships between management data of the Traffic Engineering MIBs (RFCs 3812 [151], 3813 [152], 3814 [153]) are given. One such relationship example is containment relationships. Containment relationships are the most common relationships between management state data and can be the basis for building hierarchies of objects or WS. In programming language terms, objects at higher levels of a hierarchy (i.e. Figure 3-7 object 1 at level 0) contain a portion of management state data from objects at lower levels of a hierarchy (i.e. Figure 3-7 object 2 at level 1) as well as their own data. The same can happen with WS. Using the concept of containment, a WS at level 0 (i.e. Figure 3-6 the QoS Resources

Web Service) can be created from WS at level 1 (i.e. Figure 3-6 the PHB WS at level 1). This way the higher layer WS can contain management state data from the lower layers as well as its own. While a higher layer WS contains data from lower layers it is necessary to provide access to WS state data at lower levels from WS at higher levels. This is common between programming language objects. Later we will show how this is performed with WS. A very good example of containment relationships is in SNMP Management Information Bases (MIBs). A SNMP table *contains* management state data populating its columns and rows. Another type of relationship common to SNMP MIBs which is also displayed in Figure 3-6 is *augmentation.* A table *augments* another table when both have common row identifiers. Another common relationship between SNMP MIBs is a *References* relationship. *References* relationships occur when for example an attribute from an SNMP MIB references another attribute. An example of such a relationship is the *mplsInSegmentTrafficParamPtr* attribute in the *mplsInSegmentTable* of the MultiProtocol Label Switching (MPLS) Router (LSR) [152] MIB. This parameter references a row of the *mplsTunnelResourceTable* in the MPLS Traffic Engineering MIB [151] containing the characteristics of a QoS Traffic Class (e.g. delay, jitter, loss). *AssociatesTo* and *AssignedTo* relationships are also very common between the traffic engineering MIBs. A Label Switched Path (LSP) is associated to a Per Hop Behaviour (PHB-traffic class). A Service Level Specification (SLS-traffic contract) is assigned to a PHB.

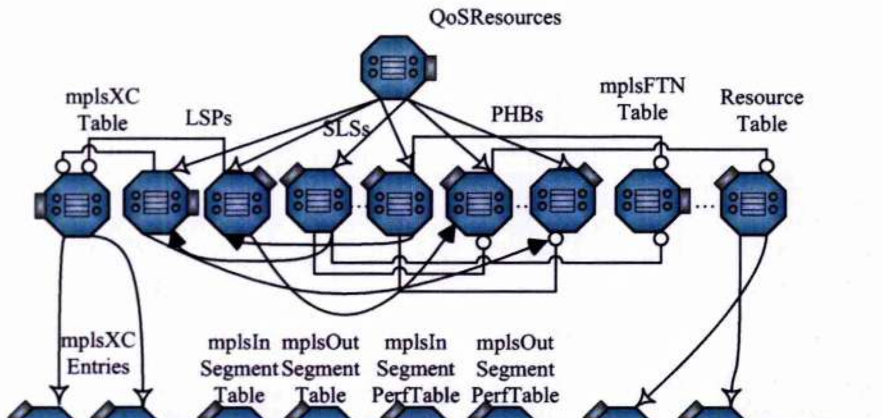


Figure 3-6 Organising WS encompassing state data in hierarchies

To structure WS as in Figure 3-6 three rules are required. Containment is the first rule. A WS containing data from other WS lies at a level higher from the latter in a hierarchy of WS. The second rule is that when *augments* relationships or any other type of relationship apart from containment exist between two WS, those WS lie at the same level of the hierarchy. The third rule is that if a WS shares both containment and other relationships with other WS, containment is a stronger relationship when classifying a WS in the hierarchy tree. Based on containment and the above rules, the hierarchy in Figure 3-6 is built between data of the traffic engineering MIBs (RFCs 3812 [151], 3813 [152], 3814 [153]). Figure 3-6 is not an exhaustive list of relationships between state data of these MIBs and should not be considered as normative but only as a possible way to structure WS. This is why as we will see later on, this is a conceptual view that an agent has on how the WS hierarchy is structured (only the agent has this view). This view may not be the way data are structured in SNMP MIBs or on the managed device. Using such a view though, we can structure a WS hierarchy that does not follow the lexicographical ordering of SNMP MIBs but an ordering based on the relationships state data share (conceptual ordering).

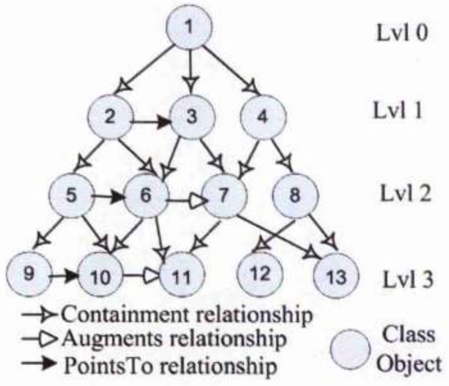


Figure 3-7 Example of relationships between programming language objects

As it will be seen in the next section, structuring WS as shown in Figure 3-6 can facilitate the process of retrieving WS state data in a bulk manner for monitoring or event reporting using relationships between state data. An example though would clarify why this is important. Consider a scenario where a manager needs to retrieve all the PHB related data from an agent which has the view of the conceptual tree in Figure 3-6. These data in SNMP lie in different tables and in different MIBs sharing a number of relationships. WS as in Figure 3-6 allow us to encompass the PHB related data using a different WS to encompass data for each PHB. This allows us to have a per single PHB view on the data (per PHB granularity). In some cases though, it is desirable to retrieve data from all PHBs or from several PHBs. Structuring data as in Figure 3-6 based on containment allows a manager to pick all PHBs at once by pointing one level higher from the WS-hierarchy where we want to retrieve data from (similar to scoping) and at the same time retrieve only the data we want from each PHB with filtering. Other reasons for the necessity of exploiting conceptual relationships are provided in [142].

To further clarify the potential of using conceptual relationships for monitoring, in the next section we analyse the functionality of the query tool we have built in order to support bulk and selective retrieval for polling based monitoring based on the relationships state data share. The tool is used as part of an architecture and a custom framework that supports delegating tasks from a manager to a series of agents in order to distribute the monitoring load.

3.4 Distributed Monitoring with a custom Query Tool

Being able to structure WS in a manner similar to that in Figure 3-6 is the first step for exploiting the relationships that exist between state data for distributed polling based monitoring. Four more steps need to be fulfilled in order to complete the process of supporting distributed polling based monitoring using WS. The first of the four steps is to find the means to expose these relationships as part of a WS. This way an entity such as an agent can exploit them for bulk retrieval in a similar way CMIP++ used the relationships between state data stored in programming language objects in scoping. The second of the four steps is to build a query tool that will support (a) bulk retrieval by exploiting the relationships between WS state data and (b) filtering for selective retrieval of WS state data. The third step is to integrate this tool as part of an architecture and a custom lightweight framework that supports distributed polling based monitoring. This way, task and load distribution from the manager to a series of agents can be performed and performance of WS-based management operations compared to other technologies can potentially increase. The final step is to make the necessary conversions to the distributed monitoring architecture which supports our query tool in order to make it compliant to the concepts and operations of MUWS or WS-Management for distributed monitoring. This is a very important step now that the role of providing custom solutions within a network domain for performance, and interoperation with standards at the edges of the network domain, is recognised as an efficient way to increase performance of WS-based management [34].

3.4.1 First step - Exposing the relationships between state data as part of a WS interface

In order to define relationships between WS as in Figure 3-6, a scheme to expose the relationships that exist between the state data that a WS encompasses is required. If containment was the only type of relationship between state data, this would be a simple thing to do. A simple scheme to define the relationships between WS state data would be to use the naming scheme of the Uniform Resource Identifiers (URIs) where services are deployed in our favour. When defining

URIs in a web browser, a slash "/" is always used to point to a web page contained in another web page. In a similar way we could use the URIs through which WS are accessed to denote the level in a hierarchy where a service is offered. In this scheme the location tag after a slash would denote the name of a WS that is contained in the WS whose location tag is before the slash (i.e. http://131.22.33.44/E1/E2, E2 is the child of E1). However, relationships between state data may not be only containment relationships. The definition of other relationships must also be possible so another scheme must be found.

One way to define relationships between services is to provide metadata about them. To provide such metadata we initially considered certain WS standards for the job. WS-Addressing [19] and WS-MetadataExchange (WS-MEX) [96] are such standards. WS-MEX specifies the messages that applications exchange in order to retrieve service metadata. WS-MEX is thus intended as a retrieval mechanism for only WSDL service description data. Thus WS-MEX cannot be used for the purpose we want to use it for. In addition to this, using WS-MEX to retrieve service relationship metadata would also require the introduction of metadata services from which these metadata should be retrieved. This will increase latency and memory requirements. Since we do not want to increase the latency and memory overhead of our query tool, WS-Addressing was considered as an alternative solution. As mentioned in chapter two, WS-Addressing was initially designed in order to support MEPs and communication scenarios that WSDL 1.1 did not support. Apart from supporting a series of MEPs though, WS-addressing could also be used in another way. The WS-Addressing specification supports the use of a metadata element inside the WSDL document of a WS so as to provide to the application that consumes the latter with service description information (WSDL information) and also other metadata about the service itself. Thus the metadata element could also be used to add information about service relationships.

Still at the time the work on our query tool was presented in the research community, the work on WS-Addressing was not finalised. In addition, while the standard and the metadata element were specified in the WS-Addressing specification, there were no open source toolkits that supported them. Thus, other means had to be found to support metadata information about service relationships until work in WS-addressing was finalised and open source toolkits supported it. Thus for the time being, we will present only how the problem of providing metadata for service relationships was solved at the time when we introduced our work on our query tool in the research community in [154]. In the section where we address the fourth step, in integrating our tool and architecture with the concepts of MUWS and WS-Management, we will show how the same problem can be solved using a standardised solution.

In order to provide metadata about service relationships, a simple and flexible scheme had to be devised. To understand the scheme we came up with though, a short summary has to be given on how WSDL is organised and how it allows deploying WS in three distinct ways. In WS, a WSDL document defines the interfaces that every WS exposes and the access points where these interfaces are accessed. In essence a WSDL document consists of an abstract part acting as an access stub and a concrete part affecting its behaviour (Figure 2-8). In the abstract part of a WSDL document the interface element describes the set of operations a service exposes. In the concrete part, the endpoint elements define the specific URI addresses where a service interface can be accessed. The binding element in the concrete part links the abstract part and the concrete part, allowing a user to define where the interface of a WS can be accessed.

Based on the above, the organisation of WSDL and the structure it enforces on its constituent parts allow a user to exploit three distinct ways to deploy a WS. The most common way of deployment is by allowing access to all the operations of a WS through a single interface. Service WS0 in Figure 3-8 shows this deployment scenario. The WSDL document for service WS0 contains one service element referring to one binding and one endpoint element. The second deployment scenario can be seen in Figure 3-8 for service WS1. There the access to service WS1 is provided through multiple access points. In this case, the WSDL document for WS1 contains one service element with multiple endpoint elements (two in this case) referring to the same binding element. The third deployment scenario can be observed in services WS2 and WS3. In this scenario two interfaces to the same programming language object are offered by defining different endpoint elements for different service elements. Each endpoint element refers to a different binding element.

In order to define metadata about the relationships between WS, we can make use of the second deployment scheme. Our scheme proposes to use several access points (Endpoints-URIs) so as to be able to define several relationships between WS. In our proposal, WS have a primary access point to provide access to them. For every relationship a service shares with another service, the latter will define a secondary URI. The secondary URI provides metadata about the relationship that the two services share with a syntax that complies with the rules of RFC 3986 [156] about constructing URIs. The syntax for the primary and the secondary URIs is given in (3.1) and (3.2). Parsing secondary URIs provides an entity like an agent with a conceptual view of a relationship tree such as that in Figure 3-6. In Figure 3-9 an example of primary and secondary URIs for WS sharing two types of relationships are provided. In this figure service SRV-E1 has only one URI to allow access to it. Services SRV-E2 and SRV-E3 have 3 URIs, the primary one and two secondary ones for the r1 and r2 types of relationships they share with other services. Services SRV-E4, SRV-E5 and SRV-E6 contain one primary and one secondary URI to show an association of type r1 with other services. Having a number of secondary URIs to denote the relationships between WS state data poses minimal overhead. This is the case because both the primary and secondary URIs point to the same object implementation and the same object instance. The minimal overhead that is introduced comes from registering each URI in the registry of the web server providing access to each WS.

Primary_URI =http://serverURL:serverPort/primaryServiceTag (3.1)

Secondary_URI=http://serverURL:serverPort/sendingServiceTag-serviceLevel_recipientServiceTag-serviceLevel.relationTag (3.2)

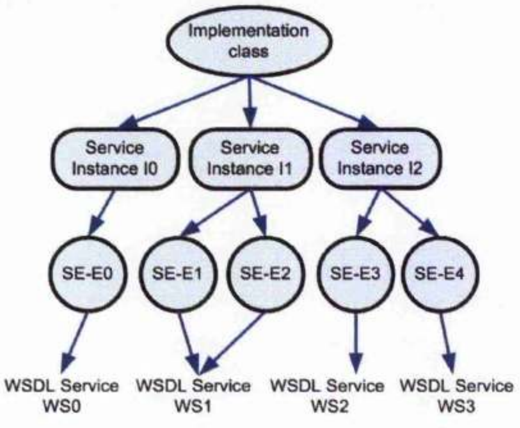


Figure 3-8 Service deployment scenarios

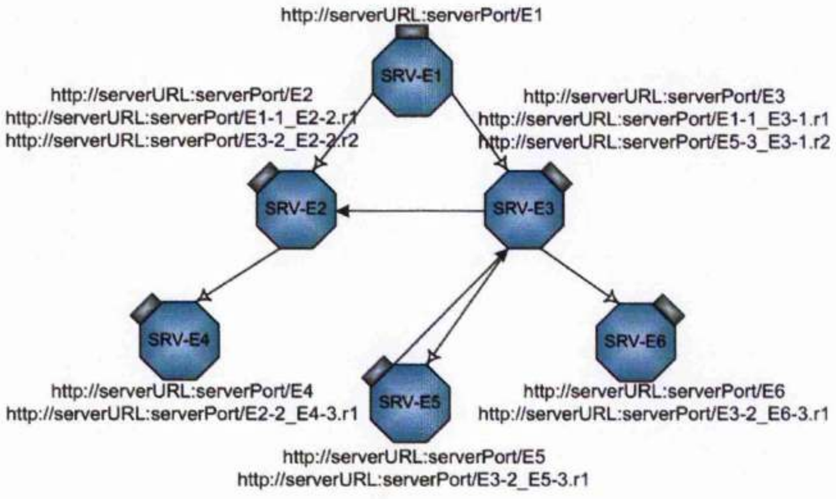

http://serverURL:serverPort/E1
SRV-E1
http://serverURL:serverPort/E2
http://serverURL:serverPort/E1-1_E2-2.r1
http://serverURL:serverPort/E3-2_E2-2.r2
http://serverURL:serverPort/E3
http://serverURL:serverPort/E1-1_E3-1.r1
http://serverURL:serverPort/E5-3_E3-1.r2
SRV-E2
SRV-E3
SRV-E4
SRV-E6
SRV-E5
http://serverURL:serverPort/E4
http://serverURL:serverPort/E2-2_E4-3.r1
http://serverURL:serverPort/E6
http://serverURL:serverPort/E3-2_E6-3.r1
http://serverURL:serverPort/E5
http://serverURL:serverPort/E3-2_E5-3.r1

Figure 3-9 Association scenario with endpoints

3.4.2 Second step – Building a query tool for bulk and selective retrieval from WS exposing management state data

So far we have explained how to exploit the relationships between state data for bulk retrieval of management data and how to expose these relationships as part of a service interface. Now we can show how to exploit these relationships. To support bulk and selective retrieval using the relationships between state data, a custom query tool was developed [155] based on Java's regular expression engine (regex). The tool supports four types of queries, each one with a special functionality. All types of queries are carried as parameters in operations supported by a custom WS monitoring framework we have developed. The operations of this monitoring framework are exposed by a series of WS interfaces. Each WS interface exposes management data from SNMP MIBs representing state data from a managed device. Using the endpoints of these WS interfaces, an agent can find out about the relationships between these interfaces and build a conceptual view of a WS tree such as that in Figure 3-6. By analysing the queries our query tool supports, the agent can select the WS and the state data to retrieve from the WS tree hierarchy. The types of queries our query tool supports are: (a) *Service Selection (SS)*, (b) *Single Instance Data (SID)*, (c) *Multiple Instance Data (MID)*, and, (d) *Filtering Data (FD)* queries. To validate the correct syntax of each query, our custom query tool uses a custom parser. In the next section we will show how this is achieved using an example.

SS queries are a combination of WS endpoint addresses, level and relationship restrictions in order to select local or remote WS hosting the state data of a managed device. Level and relationship restrictions can be used to retrieve state data from WS exposing them in a bulk manner similar to how CMIP++ used these relationships in scoping. Level restrictions are applied in order to specify from which levels of a hierarchy of WS state data can be retrieved. This is different to how CMIP++ applied level restrictions in the scoping expression. Relationship restrictions are applied to enforce selecting WS whose state data share specific relationships with the state data of other WS. The WS endpoint address inside a SS query is used in two ways. Primarily the WS endpoint address is used to point to the WS in a hierarchy of WS where the search for services will begin from. A secondary usage for the endpoint address is to show to the agent handling the relationship tree for bulk retrieval, whether local or remote WS need to be accessed for state data. As it will be shown in the next section, this is how distributed polling based monitoring can be supported. A simplified Backus Naur Form (BNF) [158] syntax for the SS query is the following:

<SS_query>::={< startpoint_tag> , <minlevel_tag> , <maxlevel_tag> , <pattern_tag>} . (3.3)

<pattern_tag>::=<identifier> | <pattern_tag> . <identifier> | (<pattern_tag>)! . (3.4)

<min_level_tag>::=<integer> . (1.5)

<max_level_tag>::=<integer> . (3.6)

<startpoint_tag>::=<URI_identifier> . (3.7)

To demonstrate how an agent can use the SS query for bulk retrieval of state data, some examples need to be given.

1) SS query with no restriction: A SS query such as that in equation 3.8, if an agent has a conceptual view of the tree of services shown in Figure 3-10, will cause a number of actions. Upon receiving the expression, the agent will use the parser of the custom query tool to evaluate its validity. If the expression is valid then the agent will extract the endpoint address from the SS_Query. By processing the endpoint address, the agent can start searching the tree of WS from the point that is defined by the endpoint address (Root in this case). The agent will search for services which can be reached by following relationships first of type r1 and then of type r2. The services selected are highlighted in Figure 3-11. If SID, MID, FD expressions are also dispatched, the agent will only return the values in each WS service selected that match the criteria posed by these expressions. How this is performed is presented in the next section.

$$SS\_query = \{http://192.168.30.4/Root,,,r1.r2\} \tag{3.8}$$

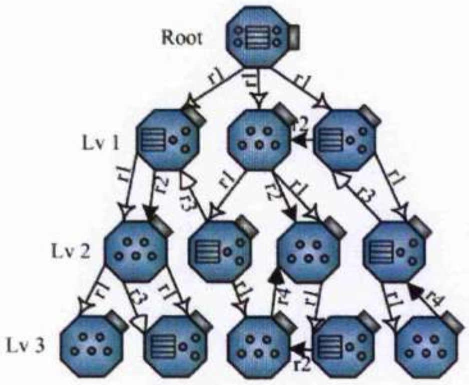


Figure 3-10 General relationship tree

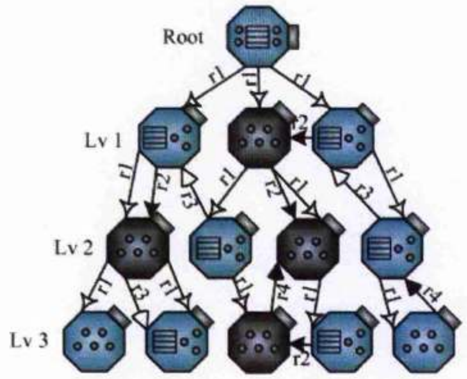


Figure 3-11 Service selection no restriction

2) SS query with single level restriction: For the path selection expression in equation 3.9, the agent will start searching the sub-tree shown in Figure 3-10 from the starting point indicated by the endpoint address (Root). It will search for services that reside only at level 2 to which you can reach following relationships first of type r1 and then r2. The selected services are highlighted in Figure 3-12.

$$SS\_query = \{http://192.168.30.4/Root,2,2,r1.r2\} \qquad (3.9)$$

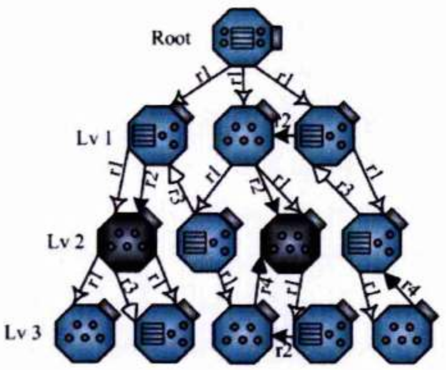


Figure 3-12 Service selection single restriction

3) SS query with multiple level restrictions: In the case where the SS query has a multi-level restriction, as in equation 3.10, the agent will search the WS sub-tree from the starting point (Root) for services that reside in level 2 and 3. Only services which can be reached by first following relationships of type r1 and then r2 will be selected. The selected services are highlighted in Figure 3-13.

$$SS\_query = \{http://192.168.30.4/Root,1,3,r1.r2\} \qquad (3.10)$$

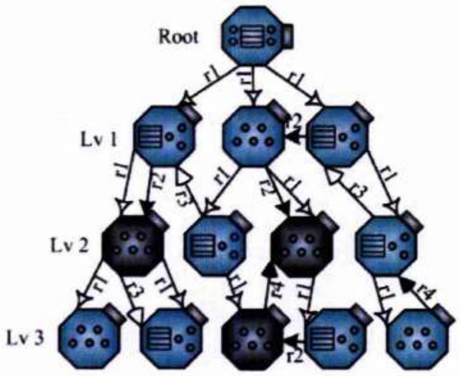


Figure 3-13 Service selection multiple restriction

4) Fringe Services: In all the above service selection examples the agent visits one after the other all the services included in the sub-tree starting from the WS-node to which the endpoint address is pointing to. For every selection the agent makes, it evaluates for every service node whether each relationship tag in the relationship restriction pattern can be followed or not. Thus for every relationship tag there is a recursive evaluation of the binary state of the relationship that a WS shares with another WS. The recursive evaluation of each relationship in the sequence of relationships that the agent follows, can also allow detection of services where the relation pattern cannot be followed (fringe services). An example of a SS query that captures services where relationships of type r1 cannot be followed is given in 3.11. The services that are selected are highlighted in Figure 3-14.

$$SS\_query = \{http://192.168.30.4/Root, 1, 3, (r1)!\} \qquad (3.11)$$

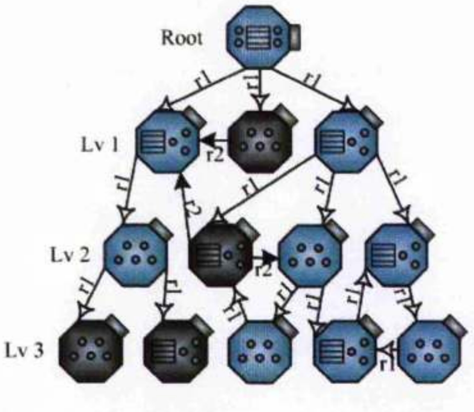


Figure 3-14 Service selection for fringe Services

SID, MID and FD queries are called data queries because they are used to retrieve the state data of a managed device hosted in a WS. *SID and MID* queries allow the retrieval of single and multiple instance data respectively from a WS hosting device state data (i.e. a table's rows are represented by multiple instance objects and MID queries can used to extract them). *FD* queries can be applied to *MID* queries to filter the collected data. The BNF syntax for the MID, SID and FD queries is the following:

<SID_query>::={<mult_inst_tag> | <mult_slct_exp>, <mult_inst_tag >} . (3.12)

<MID_query>::=<identifier>([] | [<integer>-<integer>] | [< integer>] | [< integer>(< | >)><letter>(> |<)< integer>]) . (3.13)

<FD_query>::={<mult_inst_tag ><relational operator> <value>| <flt_exp > <space><logical_operator><space> <flt_exp>} (3.14)

<value>::=<integer>|<string> (3.15)

An example of queries for retrieving all TCP connections from the TCP table in the RFC 1213 MIB whose type is FTP (File Transfer Protocol) or HTTP is given in equations 3.16 and 3.17.

MID_query={tcpConnEntry[ ]} (3.16)

FD_query= {tcpConnLocalPort = 22 OR tcpConnLocalPort =80} (3.17)

3.4.3 Third step – Using the query tool as part of an architecture for distributed polling based monitoring

This far we have introduced how the query tool queries can be used (a) to select the services from which state data will be retrieved based on the relationships these services share (b) how bulk and selective retrieval is performed using data queries. It is possible now to show how to use the query tool as part of a custom framework over an architecture that supports distributed polling based monitoring. In Figure 3-15 we depict the steps of a process where distributed polling based monitoring is performed using the query tool. In this figure a manager sends a request to an agent to retrieve data from a managed device (i.e. a router in this case). The manager performs this by accessing one of the three operations that our custom framework supports. These operations are offered by every WS interface that the agent exposes, and allow three different views to the state data of managed devices (single instance view, multiple instance view, all data view). As part of each operation's operands the manager dispatches SS, SID, MID, FD queries and a callback address (Figure 3-15 step 1). Each SS query is associated with a set of data queries and all together are used to retrieve management state data. The manager can send many combinations of SS and data queries to the agent, each of which can be used to retrieve different portions of state data. On receiving a number of queries, the agent extracts each SS query. Using an instance of the query tool and its parser, the agent validates each SS query. After validation the agent extracts the constituent parts of each SS query. By processing the endpoint address of each SS query, the agent can determine whether local or remote data need to be accessed from a WS that exposes data from a SNMP MIB (Figure 3-15 step 2 and 3). As shown in Figure 3-15, the agent has view on data that looks like a tree of WS. The agent acquires this view by processing the relationships WS share from the secondary endpoints each service exposes. This is a conceptual view which the agent builds when it processes all the secondary URIs each WS interface exposes. This view may not be the actual way state is represented in managed devices (i.e. in an SNMP MIB).

In the case that the endpoint address of a SS query points to a WS that is hosted in a remote router, the current agent tries to route the SS query and its associated data selection queries to the remote agent of these devices (Figure 3-15 step 4 alternative). This process continues from agent to agent until the remote agent is reached. Each request made to a next hop agent contains the SS query and their associated data queries. The request also contains the callback address of the manager that sent these queries in the first place. At the remote agent, the manager's callback address is used to send back the required data to the latter using a process similar to the one for retrieving data from a local router explained below (Figure 3-15 step 4). This process distributes the monitoring load to several agents.

If the agent determines from the Endpoint address of a SS query that a local router needs to be accessed for data (Figure 3-15 step 4), the agent extracts the relationship and level restrictions in the SS query. Using these restrictions it searches in his conceptual WS tree to find the WS which meet these restrictions. This process is called service selection. During this process the agent picks the services to retrieve data from. After service selection, the agent dispatches the data queries associated with each SS query to each WS that was selected. This is performed using an operation of the custom framework we have designed and which each service interface exposes. Now that each WS has a number of data queries dispatched to it, each one uses its query tool instance and parser to validate and analyse the queries it received. After analysing each query each WS can determine which data to retrieve. Having determined the data to retrieve each WS sends a request for the data to the associated managed device (Figure 3-15 step 5 and 6). After retrieving the data from the managed device (i.e. router), each WS responds to the agent with the required data in XML format. For performance reasons as we will see later on, the management data in each WS are held in programming language objects and not in an XML document instance (Figure 3-15 step 8). The agent concatenates the data it received from the various WS selected during service selection and sends back the response to the manager (Figure 3-15 step 9). An implementation of all this was presented in [159] and [164] using Linux PCs supporting state data from MPLS MIBs demonstrating that our custom query tool is scalable compared to XPath in addressing a number of scenarios. We will present this study in the next chapter.

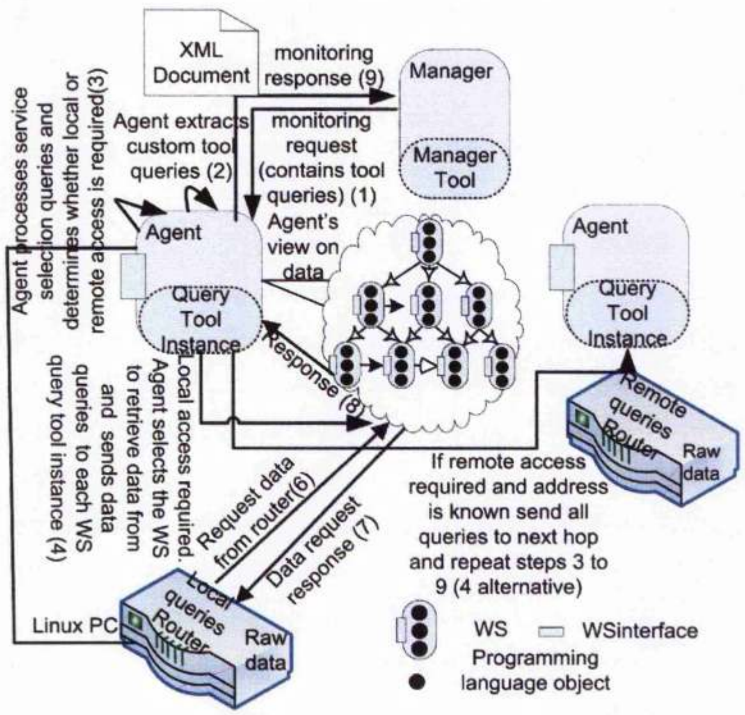

XML Document
monitoring response (9)
Manager
Agent extracts custom tool queries (2)
monitoring request (contains tool queries) (1)
Manager Tool
Agent processes service selection queries and determines whether local or remote access is required(3)
Agent
Agent's view on data
Query Tool Instance
Agent
Query Tool Instance
Response (8)
Local access required. Agent selects the WS to retrieve data from and sends data queries to each WS query tool instance (4)
Remote queries Router
Raw data
If remote access required and address is known send all queries to next hop and repeat steps 3 to 9 (4 alternative)
Request data from router(6)
Data request response (7)
Linux PC
Local queries Router
Raw data
WS
WSinterface
Programming language object

Figure 3-15 Distributed monitoring using the custom query tool to support bulk and selective retrieval monitoring

Up until now a high level view of the distributed polling based monitoring architecture has been presented. It is now possible to have a look at a few important internal implementation details. These details are given in Figure 3-16 where we portray (a) how the agent acquires a conceptual view of the tree of WS and (b) why each WS has its own query tool and associated parser instance.

In Figure 3-16 we can observe how the agent acquires the conceptual view of a WS tree. Initially the agent deploys all the Web Services that are required in order to expose the state data from a series of SNMP MIBs (Figure 3-16 step 1 – In the measurement scenarios we will present in the next chapter, the traffic engineering MIBs (RFC 3813, 3814) and the Interfaces group of state data from the RFC 1213 MIB have been implemented). Before each WS is deployed, the latter creates a query tool instance in the sense that each query tool has its own personalised parser instance (Figure 3-16 step 2, 3, 4). Why each WS tool has a personalised parser instance is explained later. During the process of deploying each WS, the agent registers to the web server that will host each WS both the primary and secondary access points of the latter. Once all services are deployed, the agent also deploys itself as a WS (Figure 3-16 step 5). The agent also has a personalised query tool and parser instance. Before deploying itself the agent analyses the secondary URIs from the registry of the web server and builds its conceptual view of the WS tree hosting management state data.

As mentioned previously, the query tool of each WS is personalised in the sense that each query tool has its own parser instance. The reasons for having a personalised parser are two. The first reason has to do with minimising the processing overhead for data queries that request for inexistent data. As such, before the constructor of the query tool class is invoked to create an instance of the query tool for each WS interface, the constructor of the query tool provides to the constructor of the parser class a set of object tags. These tags represent the name tags of every type of object that is contained in every MIB exposed as a WS interface. As such if a data query comes for data that do not exist, the query tool through its parser can determine the latter without searching the data structure hierarchy (data-gnostic query tool). The second reason behind having a query tool with a personalised parser for each WS interface is to improve the footprint of the query tool by acknowledging what sort of data are contained in each WS (i.e. single instance data, single dimension multiple instance data, multidimensional multiple instance data etc). In essence the parser of each WS interface is aware of the types of data structures contained in the latter in order to minimise the latency and memory footprint. This is similar to why schema specific parsers are aware of the XML structures contained in an XML document in order to minimise the footprint of WS applications. Based on the above it is evident that having a data structure aware and a data-gnostic personalised parser for each query tool instance can save memory as well as latency overhead.

Memory and latency overhead is also saved by performing other optimisations on the query tool. One of these optimisations is that the data contained within each WS interface are represented through programming language objects (raw data not XML). All these objects are kept in a linked list and the navigation of the latter is being performed by special pointers in each object. As such the data hierarchy of each MIB can be preserved and navigation of the data hierarchy is possible. It would be possible to hold all these data in XML document instances. The inherent hierarchical nature of an XML document would make it easier to capture the data hierarchy of a MIB. As it will be shown in the next chapter though, this would introduce a significant memory and latency overhead since the performance of general XML parsers (i.e. DOM, SAX) is not good. Performing data processing operations on raw data is much more efficient. After performing these operations, the result can be structured in an XML document using a general XML parser. The latter process is more efficient. Another optimisation that has been performed for the query tool is how the custom parser represents multiple instance data inside the linked list of raw data. Figure 3-17 displays how an SNMP table is represented through the DOM XML parser and our custom parser. In this figure it is evident that DOM (the XML parser used in Figure 3-17) creates a detailed hierarchy of element, value or attributes nodes by processing the equivalent of a table in an XML document. This occurs because in XML documents the context of each piece of information is described by explicit data nodes etc. For the query tool, a different approach was followed. The query tool treats multiple instance data of the same type (like a column from a table) as a single entity with a single tag describing their context. The benefits of this are twofold. First, the volume of data required to be searched decreases since fewer tags to describe data context are required. Second, the hierarchy of single or multiple instance objects connected with pointers in every linked list is simpler. Thus data of specific type can be found easier and faster. The later as we will see in the next chapter has a big impact on latency as well as memory overhead.

Having analysed the optimisations that have been performed on our custom query tool, we can now link the operations in Figure 3-15 to the ones in Figure 3-16 and depict a few implementation details not displayed in the former figure. Steps 6 to 14 in Figure 3-16 are the equivalent ones to steps 4 to 9 in Figure 3-15 for distributed monitoring. An aspect that can be observed in Figure 3-16 but not in Figure 3-15 though, is that each WS allows three views on its data through three different functions (a) a single instance view (getSObj) (b) a multiple instance view (getMultiObj) (c) an all data view (getObj). These are the functions that our custom framework supports and through which distributed monitoring is possible. Each WS in our distributed monitoring architecture must expose these operations. In the next section we will show that in order to support the MUWS framework operations for interoperability purposes, the WS at the edge devices of a network domain must also support these operations.

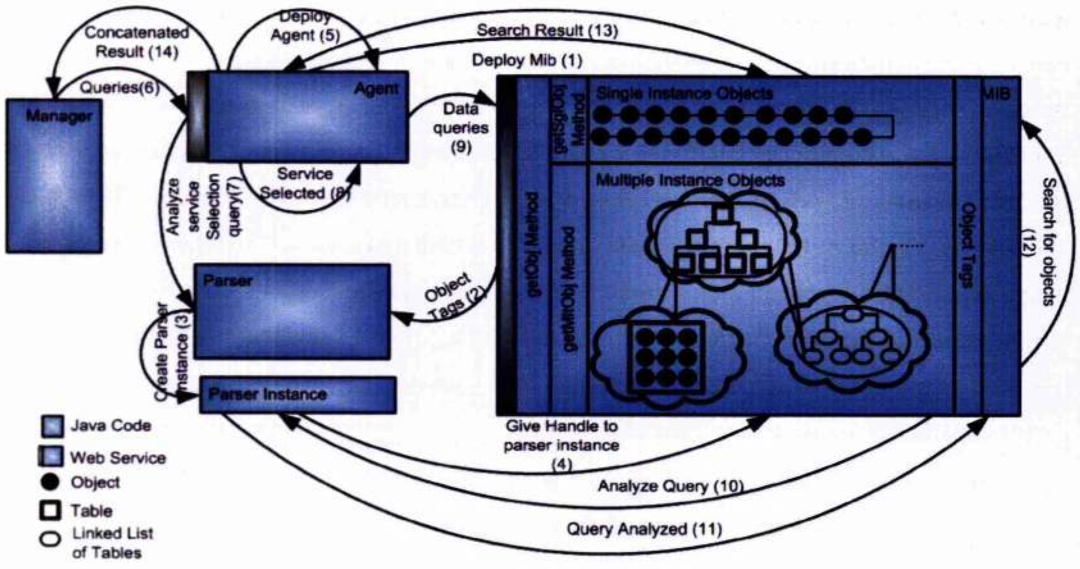


Figure 3-16 Interactions between various components of the WS polling based monitoring scheme to retrieve management information.

(This is for a scenario where the agent retrieves data from local WS. Otherwise other agents and interactions with these agents would need to be depicted.)

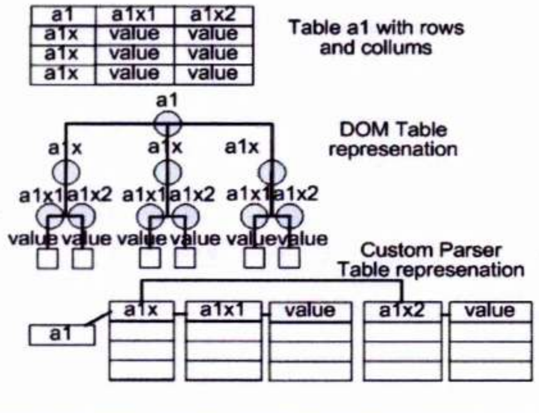

a1
a1x1
a1x2
a1x
value
Table a1 with rows and collums
DOM Table represenation
Custom Parser Table represenation

Figure 3-17 Custom parser and DOM representation of a table

3.4.4 Fourth step – Adjusting the query tool and the architecture to the concepts of the MUWS Standard

As mentioned in chapter two, during the last few years various research groups defined many WS-based specifications for Network and Service Management (NSM). Some of the most prominent work in this field has been carried out by two groups. The WS for Management and the Management Using Web Service (MUWS) specifications are the result of the standardisation efforts of these two groups. Both groups recognise in these specifications that when managing network devices the need to model, access and manage state is a key issue. While though both groups recognise the need to manage state, WS Management defines its own conventions and operations (based on the WS-Transfer specification), while MUWS adopts concepts from WSRF. Based on WSRF's concepts, MUWS introduces the concept of the manageable resource which is a refinement of a WSRF resource. A manageable resource is a resource that exposes a group of state-data/properties, operations, and metadata representing its ability to be managed. In MUWS all these aspects are called capabilities and as such MUWS defines an XML Schema and WSDL description documents to describe them.

The groups behind MUWS and WS-Management have agreed on a roadmap to create new standards for resource, event and service management. This will eventually promote building of interoperable management applications for managing the state of resources. Despite this standardisation effort though, and until this effort is complete, three features that the MUWS Framework possesses make it very promising for integrating it with our distributed monitoring architecture and our query tool.

The first feature is a *dialect* attribute in the *QueryResourceProperties* (QRP) operation (Table 2-3) which MUWS adopted from WSRF. This attribute permits the usage of a query language for retrieving a number of resource properties. Thus the QRP operation presents great potential in using it to retrieve Resource Properties (RP) of underlying resources and devices in a selective or bulk manner. As such the QRP operation can be eligible for using it with our query tool to retrieve management state-data/properties in this manner.

Another feature of MUWS is an XML element that allows defining the relationships that resources share as RP. MUWS mandates that relationships between state data can be represented as common resource properties. MUWS defines the QRBT operation (Table 2-3) in order to retrieve relationships between state data as resource properties. In this way MUWS not only standardises where and how to store the relationships between state data but also standardises how to retrieve them.

The third feature of MUWS that seems promising is the fact that the latter adopts the WSRF's WS-ServiceGroup (WS-SG) specification [88] for providing collective access to resources. The concepts in WS-SG can be used for composition of resources based on RP the latter share. This way collective access to RP can be achieved. Even more, resources can be grouped forming hierarchies of resources enabling better access to RP for monitoring.

Therefore the QRP and QRBT operations, the support of MUWS to describe and retrieve the relationships between stateful resources, and the adherence of MUWS to WS-SG, present great application potential to support our distributed monitoring architecture and gain the interoperability of the MUWS framework in return. This is very important especially now that the integration of custom lightweight solutions with standards is recognised as the means to increase scalability [34] of WS-based management applications and at the same time preserve interoperability. This is even more important today since the use of MUWS for network management in an investigation about its performance in [123] presents potential performance problems as MUWS has quite a heavy footprint for monitoring. Using MUWS only at the edges of a network domain and not through the entire network would allow a management application to preserve interoperability and at the same time use a potentially more lightweight solution in the core of the network.

In the next sections we will explain how to use the features of MUWS to support our distributed monitoring architecture and our query tool. We will explain the changes that have to be performed in our architecture and we will give an example of using our query tool for distributed monitoring of manageable resources (WS-Resources) with MUWS.

3.4.4.1 The MUWS potential to support our distributed monitoring scheme

The QRP operation in Table 2-3 enables retrieval of resource properties in a selective or bulk manner. This is achieved using a query language. MUWS supports two well known query languages; XPath v1.0 [82] and v2.0 [83]. The use of one or the other is supported by a dialect attribute pointing to the specification of their syntax. In theory any query language can be used although currently MUWS does not support other languages. In the roadmap of convergence of MUWS and WS-Management [94] for interoperability purposes though, their working groups agreed on using an extended version of the WS-Transfer specification (WS-ResoureTransfer) to support RP selection using a query language. In this roadmap it is defined that query languages are resource specific. This clearly paves the way for using our own query tool with MUWS for data retrieval, especially since in the next chapter it will be shown to be more scalable under certain situations compared to XPath in retrieving management state data when a varied number of merging and filtering operations are performed over various volumes of data.

The QRBT in Table 2-3 also presents great application potential for use with our monitoring scheme. The QRBT operation is an operation that can be used for retrieving information about relationships that exist between resource properties/state-data. MUWS standardises that relationships between state data are stored as common resource properties and defines that the QRBT can be used to retrieve these properties. As such, the QRBT operation can be potentially be used by the agents in our distributed monitoring scheme to retrieve relationship information between state data in order to build a conceptual tree like the one in Figure 3-6, this time not of WS but of WS-Resources. The hierarchy of Figure 3-18 shows the equivalent of the hierarchy in Figure 3-6 with WS-Resources.

To build the conceptual tree of WS-Resources in Figure 3-18, the use of the WS-SG specification is required. MUWS adopts the concepts of the WS-SG specification so as to support collective access to WS-Resources. Collective access to WS-Resources though, is something also required to support the containment relationships in the WS-Resource tree in Figure 3-18. This way it is possible for a WS-Resource at i.e. level 0 to contain a portion of management state data from WS-Resources at lower levels of the tree (i.e. level 1) and provide access to these data. Based on the above, it is evident that the WS-SG can be the basis for building the conceptual tree of WS-Resources that the agents in our distributed monitoring architecture need to have a view upon.

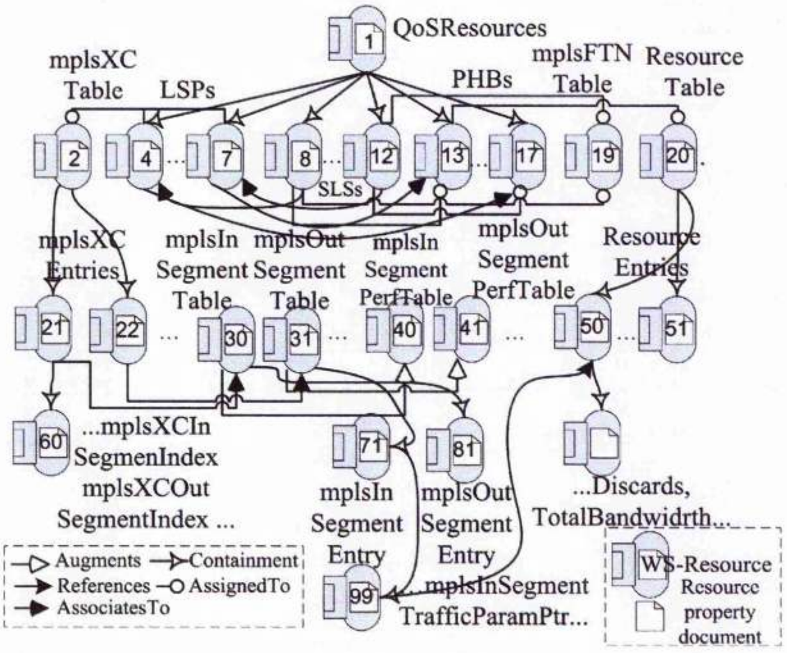


Figure 3-18 The equivalent WS-Resource hierarchy of figure 3-6

(This figure is not an exhaustive list of relationships between traffic engineering resources and should not be considered as normative but only as a possible way to structure resources)

Based on the above, it is evident that MUWS has the potential of supporting the functionality of our distributed polling based monitoring architecture of Figure 3-15. Before this is possible though, a number of other requirements have to be met. These are the following:

- In the architecture of Figure 3-15 a number of agents enable the distribution of the monitoring load in order to retrieve state data from a series of managed devices. Thus access to resource properties/state-data either when using our custom framework or the MUWS framework should be provided by a series of agents.
- Managed devices at the edges of a network domain should expose state data as WS-Resources. Within a network domain state data can be exposed through simple WS interfaces.
- WS interfaces should support the standard operations of MUWS in managed devices at the edges of the network domain and our custom framework operations across the entire domain.

This way the operations of MUWS can be used to preserve interoperability when monitoring operations need to be performed outside the domain, while our custom framework operations can be used within a network domain.

- It should be possible to use custom tool queries with the QRP operations of MUWS.
- The WS-SG specification should be used to build a hierarchy of WS-Resources such as that shown in Figure 3-18 in managed devices at the edges of a network domain.

All the previous characteristics can be supported with adjustments to the architecture described in Figure 3-15. The details of this are presented in the next section.

3.4.4.1.1 Fulfilling the MUWS requirements for distributed monitoring

In this section we will explain how the requirements introduced previously can be met so our distributed monitoring architecture and our query tool can be supported by MUWS. The next four sections address the requirements set in the five bullet points in the previous section. Each section addresses a requirement in each bullet point with the exception that the second section addresses bullet points two and three together.

3.4.4.1.1.1 First Requirement

In terms of the requirement to have a series of agents manage the monitoring process, the MUWS framework specification defines that the consumer (i.e. manager) [13], [14] of a WS-Resource is isolated from the specifics of the implementation of the WS endpoint and the manageable resource. Thus MUWS supports both agentless or with an agent implementations when managing WS-Resources. Thus supporting the agents of the architecture in Figure 3-15 with MUWS does not present any compatibility problems with the concepts and guidelines of the latter.

3.4.4.1.1.2 Second & Third Requirements

In terms of exposing management data using the WS-Resource concept for the architecture in Figure 3-15 for managed devices at the edge of a network domain, there are four conditions that need to be met. First each WS exposing a manageable resource should expose the MUWS operations through a WS addressing endpoint. Second resources should be exposed in terms of RP. Third RP documents should be linked with the WS interfaces through their WSDL portType elements. Fourth the agent itself should be a WS-Resource having access to all the other WS-Resources.

The first condition can be satisfied as long as the WS in managed devices at the edges of a network domain in Figure 3-15 are built in order expose the MUWS Framework operations. This also means though that we have to make alternating use of the MUWS operations for distributed monitoring (at the edges of a domain for interoperability) and the operations of our custom framework (within a network domain for increased performance). This should not pose a problem as devices at the network edge can use MUWS operations to communicate with other edge devices and the custom framework operations to communicate with network core devices.

The second condition can also be satisfied since the WS-RP specification and thus MUWS also support implementations of the RP document that are not instances of an XML Schema. This is necessary since the architecture in Figure 3-15 dynamically constructs the RP document and its information from data held in programming language objects, and then binds these elements to an XML document instance. Since MUWS allows resource specific implementations of the RP document, the second condition can be met without risking being non-conformant to WSRF's and MUWS' concepts for exposing state as resource properties.

For the third condition each WS exposing management data should be linked with a RP document by referring to it in its WSDL portType. This can be achieved with any WSDL implementation.

For the fourth condition to be met, the agent should use the concept of the WS-SG specification to group WS-Resources so that collective access to resources is provided. This can be achieved the way it is explained in the fifth requirement section below, but also requires the agents of the devices at the network edges to also be represented by a WS-Resource. The latter can be supported by the architecture of Figure 3-15 by making the appropriate changes so that the WS representing the agents at the edges of a network to be turned into WS-Resources.

Fulfilling the second requirement necessitates that both the agent and the WS exposing management data at the edges of a network to be converted to WS-Resources exposing the standard operations of MUWS. Thus fulfilling the second requirement (conditions 2 and 4) automatically means the third requirement is also satisfied.

3.4.4.1.1.3 Fourth Requirement

The fourth requirement can be achieved using the *QRP* and *QRBT* operations of MUWS.

The *QRBT* operation can be used to retrieve the relationships that WS-Resources share in order for an agent at the edge of a network domain to build its conceptual view of WS-Resources. This way when a service selection query is dispatched from a manager to an agent, the latter will be able to select which WS-Resources to retrieve data from.

The *QRP* operation can be used to perform bulk and selective retrieval of resource properties from WS-Resources using our query tool. To do this, first the QRP operation should support our custom tool queries. Using the dialect attribute of the *QRP* operation to point to a specification of our query tool and language would make this possible. Since it is in the future goals of MUWS and WS-Management to support resource specific query languages, this should not be a problem. Figure 3-19 shows an example of using the QRP operation to support our custom tool queries.

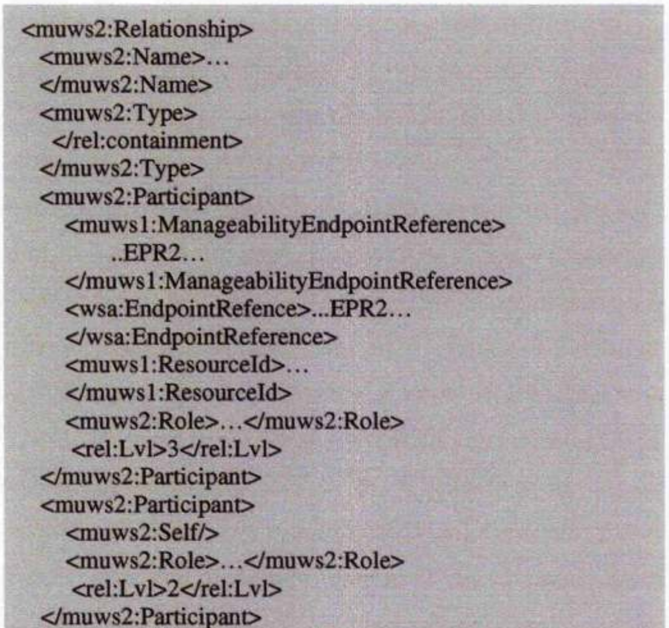
```
<wsrp:QueryResourceProperties>
  <wsrp:QueryExpression
    <qrt:SS_Query>{http://192.168.50.4:8080/WS-
        Resource/1/,2,3,AssociatesTo*Augments}
    </qrt::SS_Query>
    <qrt:MID_Query>
        {mplsInSegmentPerfEntry[ ]}
    </qrt:MID_Query>
    <qrt:FD_Query>
        {mplsinSegmentPerfDiscards<=500}
    </qrt:FD_Query>
   <qrt:SS_Query>{http://192.168.60.3:8080/WS-
        Resource/2/,2,3, AssociatesTo*Augments}
    </qrt:SS_Query>
    <qrt:MID_Query>
        {mplsInSegmentPerfEntry[ ]}
    </qrt:MID_Query>
    <qrt:FD_Query>
        {mplsinSegmentPerfDiscards>=1000}
    </qrt:FD_Query>
    <qrt:CBackAddress>...</qrt:CBackAddress>
  </wsrp:QueryExpression>
</wsrp:QueryResourceProperties>
```

Figure 3-19 Custom tool queries with MUWS's QueryResourceProperties operation

3.4.4.1.1.4 Fifth Requirement

Fulfilling the fifth requirement requires using the WS-SG specification to build a hierarchy of WS-Resources. Using this hierarchy, data from WS-Resources can be retrieved more efficiently. In order to build such a hierarchy, containment can be the relationship between WS-Resources that can serve as the basis of a member constraint to build the levels of the hierarchy. Figure 3-20 gives an example of a containment relationship between two WS-Resources one of which resides at level 2 of the hierarchy and one at level 3. In this figure, the relationship type and level association elements can be used by the agents of our architecture to build a conceptual hierarchy of WS-Resources and are defined in a separate XML schema (referred by the *rel* namespace in Figure 3-20). Inserting schema specific information such as those in the rel schema inside the *type* and *participant* elements of the MUWS schema is allowed by the latter in order to describe any schema specific information about WS relationships (see Figure 3-20). WS-Resources can share other types of relationships apart from containment relationship such as the one given in Figure 3-20. As long as relationships are defined in MUWS relationship elements and stored as resource properties of a WS-Resource, our agents can look them in order to build the hierarchy of Figure 3-18.

In order to actually support the tree of Figure 3-18, collective access from a WS-Resource at a higher level to WS-Resources of lower levels is necessary. As such it is necessary for the higher level resource to be able to access the WSDL operations and RP documents of WS-Resources at lower levels. For WSDL 2.0 this is easy due to its extensible nature. Accessing operations from the WSDL document of another WS-Resource necessitates that the latter references these operations in its portType. WSDL 2.0 allows this through an extension attribute in the portType definition. In order for a WS-Resource to be able to have access to other RP of other WS-Resources, the WSDL RP document schema of the former has to reference the WSDL RP schemas of the latter resources. This in WSDL 2.0 can be achieved by first defining each RP document of a WS-Resource in a separate XML schema. Then using the import attribute of an XML schema inside the WSDL document of a WS-Resource it is possible to refer to elements of other schemas of other resources. WSDL 1.1 though is not as extensible as WSDL 2.0. As such, in order to perform the above with WSDL 1.1, the schemas and operations of lower level resources have to be manually imported in the WSDL document of the higher level WS-Resource. It is obviously more flexible to build hierarchies of WS-Resources and provide collective access to RP with WSDL 2.0.



```
<muws2:Relationship>
  <muws2:Name>...
  </muws2:Name>
  <muws2:Type>
   </rel:containment>
  </muws2:Type>
  <muws2:Participant>
    <muws1:ManageabilityEndpointReference>
        ..EPR2...
    </muws1:ManageabilityEndpointReference>
    <wsa:EndpointRefence>...EPR2...
    </wsa:EndpointReference>
    <muws1:ResourceId>...
    </muws1:ResourceId>
    <muws2:Role>...</muws2:Role>
     <rel:Lvl>3</rel:Lvl>
  </muws2:Participant>
  <muws2:Participant>
    <muws2:Self/>
    <muws2:Role>...</muws2:Role>
     <rel:Lvl>2</rel:Lvl>
  </muws2:Participant>
</muws2:Relationship>
```

Figure 3-20 Defining relationships of WS-Resources as resource properties

3.4.4.2 Monitoring Example

Having explained how to use MUWS to achieve the requirements of the distributed monitoring architecture in Figure 3-15, it is now possible to provide an example. This example will show how to use MUWS operations for distributed monitoring across an entire network domain. Normally within a network domain only the operations of our custom lightweight framework would be required. The MUWS operations would be used for communication between domains for interoperability. Other issues need to be solved though before using MUWS for distributed monitoring between network domains. As such in this example we will only show how to use the operations of MUWS for distributed monitoring within a single domain.

In the distributed monitoring example using MUWS operations we have to imagine that the architecture in Figure 3-15 has been transformed to support the WS-Resource concept and the operations of MUWS. In addition, we need to assume that each agent has a view on a hierarchy of WS-Resources that looks like the one between WS-Resources in Figure 3-18. The example begins by having the manager of Figure 3-15 query the WS-Resource interface of the agent associated with the local queries router by invoking its *QRP* operation. This operation carries the data shown in Figure 3-19. The agent extracts the <qrt:SS_Query> elements from the QRP operation and checks the addresses they contain. The agent thus realises that the first query is for the local router and the second for a remote router. As such, it dispatches the remote SS query and its associated MID and FD queries to the next hop remote agent by invoking the latter's *QRP* operation. In this operation the agent also inserts the callback address of the manager. Back to the local queries agent, monitoring resumes by having the latter process the local SS query. The agent then determines that the manager wants to retrieve properties from the WS-Resources in level 2 and 3 that can be reached by first following relationships of type *AssociatesTo* and then type *Augments* starting the search from WS-Resource 1. The agent then searches the conceptual tree by invoking the *QRBT* operation of each WS-Resource's searching for relationships of type *AssociatesTo* with other WS-Resources. This eliminates all WS-Resources apart from 30,31,4 to 7, 13 to 17 because only the latter resources can be reached by an *AssociatesTo* relationship. The agent then queries again the remaining WS-Resources for relationships of type *Augments*. The latter results in having the agent select WS-Resources 40 and 41 in order to retrieve state data. Then the agent applies the level restrictions which mandate selecting only WS-Resources between and including levels 2 and 3. Since WS-Resources 40 and 41 belong to level 2, they remain selected. The agent then dispatches the MID and FD queries to WS-Resources 40 and 41 using their QRP operations. In each WS-Resource the *mplsInSegmentPerfEntry* instances are selected which have *mplsInSegmentPerfDiscards* values less than 500 (*mplsInSegmentPerfEntries* and *mplsInSegmentPerfDiscards* do not appear as WS-Resources in Figure 3-18 because the figure would look crowed). WS-Resource 41 does not contain any information as the ones requested by the agent and thus sends an empty response. WS-Resource 40 responds to the agent with the result contained in an XML document. The agent would normally concatenate results, but in this case it does not happen because from the WS-Resources selected, only WS-Resource 40 provides a number of state data. The agent then sends back to the manager the result in XML format. A similar process as for the local agent also takes place in the remote agent when it receives the remote SS, MID and FD queries.

3.5 Conclusions & Summary

In this chapter we have shown that state data representing the underlying resources share a number of relationships. As a result, the objects encompassing these data also share these relationships enabling efficient information retrieval as in CMIP++. Since not only objects but also WS can be used to encompass the state data of underlying resources, WS can also share a number of relationships. By having WS share relationships with other WS we can use these relationships to facilitate information retrieval for WS-based monitoring. In contrast to programming language object-oriented technologies that use relationships to structure objects into hierarchies for easier searching of the latter, WS do not support this by default. As such in this chapter we have introduced three rules in order to build a hierarchy of WS. Using these rules we have shown how to build a conceptual hierarchy of WS encompassing management state data and how to exploit these relationships for data selection. The latter enables us to search state data for monitoring, not only based on the lexicographical ordering of the state data of managed devices but also on the internal some times hidden relationships that these data share.

As a result, we have designed and built a custom query tool and parser to facilitate bulk and selective retrieval from WS hierarchies sharing a series of relationships. The query tool supports bulk retrieval by having an agent process a number of special queries, called SS_Queries, to navigate a conceptual WS hierarchy of state data which the agent builds automatically by exploiting the relationships between state data. The query tool also supports selective retrieval through information processing using special queries called data queries. The functionality of our query tool is not limited though just to bulk and selective retrieval. We integrated this query tool as part of an architecture that combines the use of a series of agents that process a number of SS_queries to support distributed polling based monitoring. As such, we have achieved the delegation of a series of monitoring tasks from a manager to a number of agents using the operations of a custom WS framework.

Based on the above, the gains of using our query tool for monitoring are threefold. Bulk and selective retrieval exploiting the relationships between state data and distribution of the monitoring load represent two of the benefits.

The third benefit is a potential benefit but with several restrictions. By introducing extra functionality to a series of management agents processing our SS_Queries using our query tool, our architecture gives a complete view of management services through ONE agent by supporting federation of management requests. As such, a manager does not need to contact many agents for different parts of the global MIB but it can contact one, assuming of course that services supported by agents are hierarchically linked. Introducing extra functionality to a series of WS-based management agents that will need to already be there for another purpose in the first place, to support distributed monitoring, is a viable solution but it raises a few concerns. Currently there are no WS agents supported in any network. If WS catch up in the future as a potential management technology, introducing such functionality to agents is possible, but nobody guarantees that there are going to be WS agents in the future. Even if this scheme is going to be used with SNMP agents, there are still problems with how pre-existing agents will support such a feature. It is possible to use extensibility features like AgentX to support such a feature but even then companies have to agree to support such functionality. Despite the several limitations, our approach can support distributed monitoring, and if adopted by future WS-based agents it can be considered as an alternative solution to distributed monitoring.

We must though be able to use our query tool not only as part of our custom monitoring framework, but also as part of a standard framework for monitoring. As such we have shown how to transform our monitoring architecture based on the concepts of the MUWS standard for distributed management of WS-Resources. MUWS as well as WS-Management are two frameworks in the process of standardising the WS operations on manageable resources for the purpose of increasing the interoperability of WS management applications. As a result, finding ways to integrate our work on distributed monitoring with the work performed in MUWS for standardising the operations performed on manageable resources is of great benefit. Using two of MUWS operations, the ability to define relationships as WS resource properties and the WS-SG specification in order to build organised hierarchies of WS-Resources, we highlight the changes that are required in our distributed monitoring architecture to support the concepts of MUWS and WSRF. This is extremely important since it allows us to use the MUWS standard at the edges of a management domain to preserve the interoperability of our distributed monitoring architecture. At the same time we can use our custom framework or any lightweight framework within a network domain so as to increase the performance of WS monitoring operations since MUWS may have quite a big overhead for monitoring as shown in [123]. Such approaches are gaining ground as for example in [34] where a similar scheme was suggested for event reporting.

In the next chapter we will perform a comparison of our query tool with XPath. This way we will show that in scenarios where a varied number of merging and filtering operations are required over various volumes of data, our custom-based tool can be more scalable from general purpose tools when used for network management monitoring operations. Driven by the scalability of our query tool we will use it in a case study for polling based monitoring over MPLS enabled networks. We will investigate three scenarios over these networks where bulk and selective retrieval is required. Based on these scenarios, we will compare the performance of our custom monitoring framework with a standard protocol such as SNMP. This way we will be able to demonstrate that our custom monitoring framework is lightweight enough to be used for WS-based monitoring.

Chapter 4

4 Testing the Efficiency of our Query Tool and our Monitoring Framework

4.1 Introduction

In chapter two we have analysed the work that has been carried out by other researchers in evaluating the performance of WS for polling based monitoring of the state of network devices. Based on that work we mentioned that it would appear that WS could be used for network monitoring only in cases where a great amount of data needs to be retrieved. This happens because it seems that WS by nature constitute a relatively heavyweight technology in terms of memory, latency and traffic overhead. This is attributed to the verbosity of XML tags describing the context of management data. In addition the tools and APIs used to built and deploy WS are still in the process of development and, as a result, performance is inhibited. While the issues of the tools and APIs performance for building and deploying WS are being resolved as shown by the increasing performance of WS toolkits in [40], [97] , the verbosity of XML tags will always have a negative impact on WS performance. Still there are ways to minimise this impact. Minimising the processing overhead of data stored in XML may be more important in some cases than solving the encoding, serialisation and parsing problems that affect the performance of SOAP toolkits and thus the performance of WS management operations. For example as shown in an investigation of our own in [180] the latency performance of Axis 2.1.4 versus its predecessor (Axis 1.1.4) has increased by 3 times for discovering the PHB IDs that need to be retrieved in scenario 2 that we introduce in section 4.2.2 (1000ms difference when 980 objects are retrieved). As will be shown later using the same network setup, the latency performance of our custom query tool against XPath version 1.0 or 2.0 for processing the same amount of objects is better by 9 or 17 times respectively (2600ms or 6000ms difference).

Irrespective of the factors that affect the performance of WS-based management operations studied by other researchers, a series of factors influencing the use of WS for network monitoring have not been investigated yet. Previous research has not examined scenarios where bulk and selective retrieval and information processing on management data is required. On the contrary, researchers have only investigated scenarios where management data are retrieved in a sequential (by defininiing each piece of information to be retrieved explicitly) or bulk manner. Even so in these scenarios, researchers tested the performance of retrieving management state data in a bulk manner by using schemes that deployed special functions to offer bulk access to management data. These schemes are not flexible since in order to have an adjustable level of granularity for monitoring operations, the WS interface offering access to management data has to expose a very big number of functions. This increases memory overhead. In addition using these schemes for bulk retrieval of management data requires to visualise before hand which management data should be grouped together so that collective access to these data is provided through special functions. This is also not flexible. Furthermore, in cases where a large population of WS is required in order to represent the underlying resources, these approaches are not scalable because the big number of functions required to access state data increase the footprint of each WS. In addition to the above, some schemes that have been researched for monitoring mandate the retrieval of each piece of information explicitly (using a series of identifiers). The latter is not optimal since this way traffic overhead increases uncecessarily.

On the contrary, schemes that use query tools to retrieve management data in bulk or selectively, have the potential of being more efficient. This is because when using query tools to retrieve state data representing the underlying resources, it is possible to use a single method per WS interface to achieve the same granularity as previous researchers did with many specialised methods. As such, query tool schemes theoritically have the potential of being more lightweight. In addition, query tools can be a great addition to several monitoring scenarios. Examples of such scenarios can be found in cases (a) where the entire data of a device needs not to be modified or retrieved (b) when state data have to be retrieved based on the relationships they share (c) when it is necessary to retrieve management information without specifying explicitly each piece of information that needs to be retrieved. For the first case it is plausible to process data close to the devices from where they are retrieved, rather than retrieving the entire state of a device and process it at the manager. Performing the latter will increase traffic overhead and at the same time could have an ovewhelming effect on the manager. For the second case especially when using SNMP MIBs, data usually hide a lot of relationships and processing is required to exploit these relationships. The latter is necessary in many cases where the need to achieve a number of high level monitoring tasks requires from a management entity to process a number of relationships between state data. A query tool can be used to uncover these relationships on the fly whereas a scheme with a series of specialised functions has to visualise these relationships beforehand and provide special functions to achieve every high level task that is necessary for monitoring. For the third case it is evident that defining a simple Boolean expression containing assertions on attribute values to pinpoint the data that need to be retrieved for monitoring is better than defining each piece of information to be retrieved explicitly. The latter is not very efficient, especially in terms of traffic overhead. Query tools used for WS-based monitoring can be used effectively to address all three scenarios described previously. In the past though researchers have not invested a lot of effort in checking the performance of query tools for WS-based management. As such, the scalability of these query tools in processing management data and how the latter affects the performance of WS-based management operations need to be investigated.

Based on the above it is evident that it is important first to find and then analyse scenarios where a number of high level tasks need to be achieved using bulk and selective retrieval facilities and information processing. Based on these scenarios it is important to study the performance of various query tools, including our own tool introduced in chapter three, to evaluate how lightweight each tools is. This way we can also identify potential problems with each tool and possibly suggest solutions to these problems. After having investigated the performance of the various query tools, it will then be possible to select the one that is more scalable and evaluate the performance of WS-based polling based monitoring against other technologies.

In the next sections we will introduce a number of scenarios based on the monitoring system of a network that provides Quality of Service (QoS) guarantees to clients over Multi-Protocol Label Switching (MPLS) routers. These scenarios necessitate but do not mandate the use of tools for retrieving state data in a bulk or selective retrieval manner. Based on these scenarios we will compare the performance of our query tool with XPath v1.0 and v2.0. The latter are suggested as potential candidates for handling the XML configuration payload of the NetConf protocol using filtering (selective) and merging (bulk) operations. XPath implementations have also been suggested as potential candidates for extracting the properties of manageable resources for monitoring and event reporting (MUWS, WS-Management). We will show that XPath may have potential scalability problems handling time critical scenarios and that our custom query tool can perform better. Based on these results, we decided to use our query tool in order to evaluate the performance of WS polling based monitoring against SNMP. This way we will be able to extract conclusions on the performance of our custom WS framework and whether the latter can be used for efficient polling based monitoring.

4.2 QoS Monitoring Requirements and Scenarios

4.2.1 Introduction

As mentioned in the previous section, a number of scenarios will be introduced in this section for the purpose of comparing our custom query tool with XPath and our custom WS-based framework with SNMP. Before introducing these scenarios though, it is necessary to provide a background on how to provide Quality of Service (QoS) guarantees. This is necessary since the scenarios introduced in the next section involve monitoring operations on QoS-enabled networks

Providing QoS in a network domain or across different domains has been the subject of extensive research over the past years. Currently QoS is provided on the basis of Service Level Agreements (SLAs). These represent a set of terms that clients and providers of services have to abide by, when they are accessing and providing a service respectively. The technical part of an SLA is called a Service Level Specification (SLS) and it represents the means for defining QoS-based IP connectivity services [166]. IP Differentiated Services [167] (DiffServ) is currently seen as the framework for providing QoS-based services. In this framework, routers aggregate traffic that belongs to several service classes according to predefined QoS policies. These policies are quantified through performance parameters such as throughput, delay, loss and delay variation. A common approach to support the DiffServ architecture is over Multi-Protocol Label Switching (MPLS) traffic-engineered networks. Maximising network resource utilisation and at the same time meeting the QoS demands of services contracted to customers in these networks is a key target for operators offering QoS-based services.

Based on the above it becomes evident that monitoring of the network status and its resources is an essential process in order to ensure a network's smooth and reliable operation. When providing QoS-based value-added services, this becomes even more important. Providing value added services is a challenging task and requires the deployment of resource management techniques, such as the use of Traffic Engineering (TE). The latter requires the collection of monitoring data enabling both off-line/proactive and dynamic/reactive operations to be performed. These operations can ensure the smooth operation of the network. As such, the role of a scalable monitoring system in terms of network size, speed, and number of contracted services to customers, is of paramount importance as the monitoring system in QoS networks triggers the operations that can ensure the smooth operation of the latter. Given the multitude of services with various performance requirements and the need to have measurement data with the finest granularity possible, the design and implementation of scalable monitoring systems constitutes a significant challenge. This is especially true as such systems must be capable of providing measurements for network provisioning, dynamic resource allocation, route management, and in-service verification of value-added services. Therefore, Quality of Service monitoring is a very important aspect in the process of providing quantified QoS-based services.

Providing QoS over MPLS enabled networks has been the focus of various research projects. Examples of such work are the frameworks proposed by the TEQUILA [162], [163], the CADENUS, and the ENTHRONE [165] projects. The architectures developed by these projects rely on their monitoring systems to provide them with up to date information about the state of the network. To address the needs of Traffic Engineering and QoS provisioning in general, a monitoring system must be able to acquire long and short term data. All the monitoring systems of the aforementioned frameworks collect these data using the Manager-Agent paradigm. The fact though that QoS enabled networks may have a large number of nodes and a large number of routes with different QoS guarantees may result in an exponential increase of monitoring requirements. As such in QoS networks where MPLS is used, their monitoring systems for scalability reasons perform measurements only at the MPLS Label Switched Path (LSP) level, the QoS traffic class level (Per Hop Behaviour – PHB) and the traffic contract level (Service Level Specification-SLS). The types of measurements performed at these levels are two: active and passive ones. Active measurements are performed by injecting synthetic traffic to the network in order to monitor delay, jitter and packet loss. Passive measurements can be conducted using Management Information Bases from SNMP and involve measuring throughput, load and packet discards at the PHB, SLS and LSP levels [162], [163], [164]. The type and the specific points where passive measurements need to be performed in QoS networks are the following:

- LSP load at the ingress router (LSP L-I)
- LSP throughput at the egress router (LSP T-E)
- PHB throughput at every router (PHB T)
- PHB packet discards at every router (PHB D)
- Offered load at ingress per SLS or flow (SLS L)
- Offered throughput at egress per SLS or flow (SLS T)

As our investigation focuses on evaluating the performance of our custom WS framework against SNMP and our query tool against XPath, active monitoring will not be considered. Active measurements are not considered because they can be performed in the same manner for both SNMP and WS. As such active measurements are of no benefit in the context of evaluating the performance of our query tool to XPath or testing the performance of WS against SNMP for monitoring.

4.2.2 QoS monitoring scenarios

In this section we introduce three scenarios based on which the performance and scalability of WS-based management will be examined.

The first scenario will be used to compare the performance of WS and SNMP for retrieving the state data required in order to perform the passive measurements of a QoS network. In this scenario SNMP and WS could be used in the same way. This means that it is not necessary to use our custom query tool or XPath to perform any of the measurements involved in this scenario.

Normally though the process of performing the passive measurements for QoS can be facilitated by having bulk or selective retrieval capabilities. We will explore this possibility in the second scenario. For the time being we want to use the first scenario in order to establish a base for the performance of WS and SNMP. As such in the first scenario only bulk retrieval capabilities of WS and SNMP will be exploited. This applies both for the operations that WS expose based on our custom framework or the SNMP's GetBulk operation. For some of the passive measurements though, information would have to be retrieved from different groups of data and from various MIBs. This for WS would normally require a number of different functions and potentially will affect latency overhead. To keep the comparison between WS and SNMP on equal terms, we decided to make a change. We decided to use just one of the functions of our custom framework for the measurements (the one providing access to all the data of a MIB exposed through a WS interface). This method will use our query tool or XPath (depending which is more scalable) to retrieve data in a bulk way from the various groups of data and the various MIBs required. Using our query tool or XPath for bulk retrieval should not affect performance because of the nature of the measurements required for this scenario. This is true because for the measurements of this scenario no information processing is required. In addition, this scenario will also evaluate the performance of consecutive SNMP GetNext operations to acquire the same data as with GetBulk or with WS.

The second scenario involves performing the same passive measurements as the first scenario. This time though, our custom query tool or XPath will be used with our custom WS framework for data processing at the agent side. This way the passive measurement data to be retrieved will be sent back to the manager in an ordered manner. By an ordered manner we mean on per PHB, SLS, or LSP basis. As such the manager can be relieved from the task of processing data from a big number of agents which is a load that could be overwhelming for a single entity. Consequently, a WS agent will be used to offer access to passive measurement data in a bulk manner but with capabilities of information processing. This is possible by using the selective retrieval facilities of XPath or our query tool before delivering management data to the manager. For SNMP we do not use facilities for information processing. Although it would be possible to use the Script or the Expression MIB to handle this task, as explained in chapter two, the mechanisms offered by these MIBs create a lot of problems. They are not secure, they are sometimes inefficient, impractical, they have limited support and as explained in many cases such as for monitoring they introduce more overhead to SNMP operations than they save. As such, SNMP will be used with the same capabilities as in the first scenario. In general with this scenario we will try to show that query tools for WS can be used to delegate and distribute the data processing load for monitoring to a number of agents. As a result, we can examine if such an approach could be a more viable option for monitoring using WS.

The third scenario assumes that the MPLS interface of the ingress router of a QoS network fails. On receiving an event about the failure of this interface, the manager needs to determine the affected LSPs and service contracts (SLSs). This is a selective retrieval scenario where only the relevant information from a MIB hosted at an agent will be retrieved. This scenario is going to be used both for testing our custom query tool with XPath v1.0 and v2.0 as well as our WS-based custom framework and SNMP. Through this scenario we will determine whether XPath 1.0 or 2.0 is more scalable than our custom query tool. Based on this scenario we can decide whether to use XPath or our custom query tool in order to evaluate the performance of SNMP against our custom WS-based monitoring framework (for all three scenarios).

4.2.3 Management Information used for the scenarios

To perform the measurements required for the three QoS scenarios, a number of commercial SNMP Management Information Bases (MIBs) must be used. The MPLS Label Switching Router (LSR) MIB (RFC 3813) [152] and the MPLS Forwarding Equivalence Class to Next Hop Label Forwarding Entry (FEC) MIB (RFC 3814) [153] are necessary to perform the measurements required for any of the three scenarios. The LSR MIB can be used to perform PHB and LSP measurements whereas the FEC MIB can be used to perform SLS measurements. The Interfaces group from the RFC 1213 MIB must also be implemented. This is required since the MPLS MIBs are associated with many of the Interfaces group data.

4.3 Comparing XPath with our custom query tool

Polling-based monitoring is a key domain in using WS for Network Management. Monitoring in many scenarios involves retrieving state data in a bulk or selective manner from a managed device in a synchronous manner. Thus in many monitoring scenarios, mechanisms to retrieve data in a bulk or selective way are important. When using such mechanisms though, their scalability is a very crucial parameter. This is because in cases such as that of a network providing QoS guarantees, collection of management data is critical. In WS-Based management this is a great challenge since the verbosity of XML tags increase the coding and processing latency as well as the traffic and memory overhead requirements. In addition the performance of general-purpose XML parsers may be poor for network monitoring operations. Therefore the tools that will be used to manipulate XML data for bulk and selective retrieval should be efficient enough, if not to reduce the extra overhead imposed by XML tags and parsers, at least to increase it in a scalable manner.

Two of the industry's options for processing and querying management data in XML documents are the XML Path (XPath) Language v. 1.0 & 2.0 [82], [83]. XPath 1.0 and 2.0 are W3C candidate recommendations in 1999 and 2005 respectively. They took their name from the path notation syntax with slashes "/" they use to control the selection of specific portions of an XML document. In addition to this functionality, XPath can also be used to perform operations on data after selecting the latter. In order to be able to select or modify XML data, XPath is using the Document Object Model (DOM) parser for analysing and keeping the structure of an XML document in memory. Using DOM to analyse an XML document, XPath can then process special queries sent to it in order to select or modify the data inside an XML document. These queries contain merging and filtering operations of arbitrary complexity, making XPath a very expressive tool when used for handling XML content.

In the past, XPath has been proposed for altering or selecting configuration data in NetConf. NetConf uses XPath in the special operations it defines to manage the configuration data of a managed device stored in XML. NetConf's performance though, in retrieving or altering configuration data in terms of memory, latency and traffic overhead relies on the XML parsing and handling techniques used by the supporting technologies it utilises. In essence NetConf's performance depends on technologies such as XPath and DOM. In the past, various concerns have been expressed in the NetConf mailing list that XPath might be heavy in terms of memory and latency overhead for handling configuration data. As such they have suggested sub-tree filtering as an alternative. Comparing XPath 1.0 and sub-tree filtering, in performing the filtering and merging operations required for handling configuration data, the authors in [134] and [137] conclude that when used only for filtering, sub-tree filtering is slightly better. The opposite happens when combining merging and filtering operations. This work though in essence implies that both XPath and sub-tree filtering may have problems in accessing or altering configuration data.

In the same way as for handling configuration data in NetConf, XPath has been suggested by some management technologies (MUWS, WS-Management) as a candidate technology for monitoring and event reporting. In MUWS and WS-Management, XPath is used to retrieve state in a bulk (XPath merging operations) or selective manner (XPath filtering operations). Still XPath may have limitations when used for this purpose. A limitation XPath 1.0 has, in comparison to its successor is the limited expressiveness of queries it supports. This can sometimes inhibit the performance of XPath 1.0 since it may increase the number of times a document needs to be searched for retrieving or altering XML data. In the next section we will elaborate further on this issue. Another important shortcoming that all XPath implementations share is their dependency on DOM. DOM allows dynamically accessing and updating the content and structure of XML documents by loading all its data into memory. Handling an XML document the way DOM does can unnecessarily in some cases increase the memory and latency requirements for retrieving or modifying state. As such, it is evident that the concerns that XPath may be heavy for configuration management, may also be applicable for monitoring and event reporting.

As a result of the above conclusions it is necessary to compare our custom query tool and XPath in terms of their performance in executing monitoring operations. When comparing the performance of our custom query tool with XPath in terms of their scalability we need to examine (a) potential bottlenecks in the performance of these tools (b) how these tools perform when a small or a large volume of data needs to be accessed or processed (c) how these tools perform when executing monitoring operations that require bulk and selective retrieval and (d) how different XPath implementations perform compared to one another.

As such in the next sections we will explain how we need to set up our third QoS scenario in order to achieve the above goals. Having done that, we will then elaborate on the management model as part of which we will compare the performance of XPath and our custom query tool. Then we will give examples and analyse the queries that can be formed with XPath or our custom query tool. This way it will be possible to explain the actual queries that will need to be formed as part of the third QoS scenario for testing the performance of each query tool. Finally we will present the software and hardware setup we used for the measurements of the third QoS scenario and then analyse the measurements themselves.

4.3.1 The Set Up of the QoS Scenario

To evaluate the performance of our query tool and XPath we need to make the volume of information that needs to be processed volatile. This way we can identify how each tool performs when processing a big or a small volume of data. To perform the above, the routers in the third QoS scenario will be configured to have on one occasion a small number of LSPs and on another occasion a big number of LSPs. This way we can simulate a large and a small network and thus change the volume of information that has to be processed. For the needs of the measurements we will configure the ingress router of the third QoS scenario to have 900 or 30 LSPs. When configuring the network as described above though, another requirement is to keep the measurements either when the network is small or big on the same terms. This way we can extract safe conclusions as to how the volume of information to be processed affects the performance of each query tool. As such we assume that the number of LSPs and SLSs affected by the failing interface in the third scenario is six. This may be a plausible number of LSPs assigned to a single interface for a small network, but it may not be plausible for a large one. The aim though when making this selection is to keep the number of affected SLSs and LSPs for small and large networks the same so as to be able to extract safe conclusions about any potential bottlenecks in each tools' performance.

To check how different XPath implementations perform compared to one another and also how each tool performs when performing bulk and selective retrieval actions, the QoS scenario contains a series of queries with a varied number of merging (bulk retrieval) and filtering operations (selective retrieval) for each query. By altering the number of these operations from one query to another, we can check how this affects performance both of the custom query tool but also of XPath. Later it will be shown that some XPath implementations perform better than others. By varying the number of merging and filtering operations we will be able to show that the increased expressiveness of XPath 2.0 can be beneficial, decreasing memory and latency overhead.

Checking for potential bottlenecks in each tool's performance using the third QoS scenario is provided by default from the moment we chose to compare our query tool with XPath. The parser of our custom query tool operates on raw data. DOM, which is the parser that XPath uses, operates on XML. By making this experiment we can check the potential of DOM in being a bottleneck for XPath since the parser of our query tool operates in a different way (Figure 3-17 and section 3.4.3). By altering the volume of data for the scenario measurements, we also gain another advantage when checking for potential bottlenecks in each tool's performance. We can identify potential problems with each tool and suggest ways to increase their performance.

Our scenario though has also two limitations. The first is that although the scenario we have chosen allows for checking the performance of different XPath v1.0 & 2.0 implementations, this is not currently possible. This is because there are not many Java implementations of XPath or even in other languages that are fully conformant to standards, are mature and come from reliable bodies. The implementations we have chosen though, meet all these requirements. SAXON 8.9 by Michael Kay which is one of the XPath 2.0 standard authors [160] is a good implementation of XPath 2.0, it is conformant to the standard specification, and is mature. The Java API for XML Processing (JAXP) implementing XPath 1.0 from SUN [161] comes from a very reliable body that needs no introduction.

4.3.2 The management model for the measurements

In the third QoS scenario based on which we will evaluate each tool's performance, a Multi-Protocol Label Switching (MPLS) network will be used as the means to provide QoS guarantees to clients. The scenario requires performing passive measurements at the Label Switched Path and the Service Level Specification (SLS-traffic contract) level. As such for this reason we have selected two of SNMP's MPLS MIBs to represent management data. These are the MPLS Label Switching Router (LSR) MIB [152] and the MPLS Forwarding Equivalence Class to Next Hop

Label Forwarding Entry (FEC-To-NHLFE) MIB [153]. These MIBs are used to perform Per Hop Behaviour (PHB-traffic trunk) LSP and SLS measurements.

For the scenario measurements, the MPLS MIBs are used to expose management data through WS interfaces. The model used for monitoring these data through their interfaces is the manager-agent model. Though our tool can also support distributed monitoring, this is not possible for XPath. As such we do not use SS queries in this scenario and we do not structure WS in hierarchies since we need to test all tools on the same terms. For each MPLS MIB, a WS interface is created offering access to different portions of the latter's data using three methods. For XPath and the custom tool these methods can be seen in Figure 4-1. The *sglDataGet and simpleDataGet* methods allow access to single instance MIB data. The *multiDataGet and complexDataGet* allows access to multiple instance data. The *sglMultiDataGet* and *simpleComplexDataGet* allow access to all the MIBs' data.

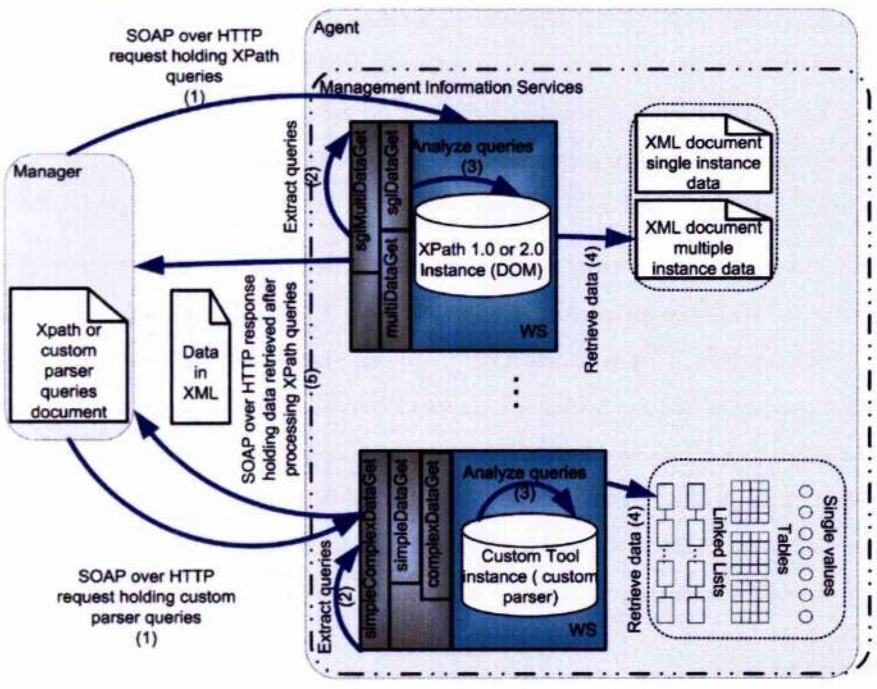

SOAP over HTTP request holding XPath queries (1)
Agent
Management Information Services
Manager
Extract queries (2)
sglMultiDataGet
sglDataGet
multiDataGet
Analyze queries (3)
XPath 1.0 or 2.0 Instance (DOM)
WS
Retrieve data (4)
XML document single instance data
XML document multiple instance data
Xpath or custom parser queries document
Data in XML
SOAP over HTTP response holding data retrieved after processing XPath queries (5)
simpleComplexDataGet
simpleDataGet
complexDataGet
Analyze queries (3)
Custom Tool instance (custom parser)
WS
Retrieve data (4)
Linked Lists
Tables
Single values
SOAP over HTTP request holding custom parser queries (1)
Extract queries (2)

Figure 4-1 Data Retrieval using XPath 1.0 or 2.0 or the custom tool

In order to retrieve data from a WS exposing an MPLS MIB, each WS incorporates an instance either of our custom query tool or XPath 1.0 or 2.0 (Figure 4-1). Retrieving state data from a MIB requires a manager to send either XPath or custom tool queries to an agent. These queries are sent as arguments of a Remote Procedure Call (RPC). The queries are then analysed by the custom query tool or XPath instance of each service. When the required data are selected by each tool, an XML parser is used to form the response in XML format to send back to the manager. The above process is the same for the WS that use XPath implementations as well as those that use our custom query tool. The only difference is that the parser of our custom tool has view upon raw data whereas XPath (DOM) has view on XML. All tools support bulk and selective retrieval through merging and filtering operations, though with different syntax.

4.3.3 An analysis of XPath queries and their custom query tool equivalents

4.3.3.1 XPath 1.0 and 2.0 queries

The most common perception that people have about XPath's view of an XML document is that of a tree of nodes. In XPath, the most common kind of nodes are (a) element (b) attribute (c) text, (d) namespace, (e) processing-instruction (f) comment and (g) document (root). The root node of an XML document tree is called the document node. Element nodes describe the context of data. Attribute nodes, are nodes that describe attributes of an element node. Text nodes are nodes that contain the actual data of an element or attribute node. More details about nodes and XPath are given in [168]. Nodes in an XML document also share relationships. In XPath there are five types of node relationships: (a) Parents (b) Children (c) Siblings (d) Ancestors (e) Descendants. Selecting nodes in XPath is performed with appropriate path expressions to the node that needs to be extracted, depending on the type of the node to be extracted and the relationship it shares with other nodes. Most common symbols to depict the type of node and its relationship with other nodes are given in Table 4-1.

Nodename	Selects all nodes after this node
/	Selects nodes having as start point root node
//	Selects nodes from the current node that match the selection no matter where they are
.	Selects the current node
..	Select the parent of the current node
@	Selects attributes from a node
*	Selects any element node
[ ]	Find a node that contains a specific value.

Table 4-1 Symbols for representing nodes in XPath 1.0 and 2.0 [168]

Useful examples of XPath expressions similar to the queries that will be used in the measurements of the third QoS scenario are given in Table 4-2. The first query in this table selects all the element nodes of type b that are descendents of element nodes of type a. In the second expression all the element nodes of type *a* are selected if they have as descendents element nodes of type b whose value is equal to 1. This is an example of a filtering operation. In the third expression all the element nodes of type c are selected starting the search from element nodes of type *a* that have as descendants, nodes of type b with a text node value equal to 1. An interesting feature in expressions four and five is the use of operators. For example the "|" operator computes multiple node-sets and the equals "=" operator tests for equality between two values. From these expressions it is also possible to observe a difference in the operator expressiveness of XPath 1.0

& 2.0. What is depicted here is that while the use of parentheses in XPath 1.0 at the first step of a path expression is allowed i.e. (a|b|c)/d/e, expressions such as /a/b/(c |d | e) are not permitted. XPath 2.0 allows use of the latter expression along with the use of more operators and functions. This allows a user of XPath 2.0 to form less verbose queries that require fewer merging operations in some cases than XPath 1.0. This enables XPath 2.0 as we will see later to impose a smaller latency or memory overhead than XPath 1.0, since it requires searching an XML document fewer times.

a//b	Selects all b element nodes that are descendants of a element nodes, belonging under the latter
//a[b='1']	Selects all a elements with b elements equal to 1
//a[b='1']/c	Selects all c elements starting the search from a tags with b=1
//a[b='1']/(c|d)	**XPath2 only** Selects all c and d elements starting the search from a elements with b=1
//a[b='1']/c	//a[b='1']/d	**XPath2 & 1** Same result as the previous for XPath 1& 2

Table 4-2 Sample expressions for XPath 1.0 and 2.0

4.3.3.2 Comparison of XPath with our custom query tool

Our custom query tool and XPath 1.0 & 2.0 were designed with a different perspective. The custom tool uses special queries to select the type of programming language structures from which to retrieve data from. XPath uses the keys in Table 4-1 to select XML data nodes based on the containment relationships nodes share with other nodes. This way XPath can find the path to the XML data that needs to be retrieved. The custom parser operates on raw data and does not need to define a path to the data that need to be retrieved. This happens because it exploits pointers and relationships between raw data objects or WS respectively to guide itself across the data hierarchy (raw data objects are stacked in a linked list). XPath uses absolute or relative path expressions and symbols to find its way to the data that need to be retrieved. The custom tool uses a minimal set of operators to perform filtering (=,!=,>=,<=,<,>, AND,OR) or bulk retrieval (merging ",") operations. XPath implementations and especially version 2.0 have loads of operators and functions to perform filtering and merging. Based on the above, it is easy to deduce that the query tool does not have as much functionality as XPath 1.0 and 2.0 and it cannot be as expressive and flexible as the latter. Still the tool can be used to address the purpose of retrieving management state data for monitoring and event reporting as used for example in [169]. In this work the custom query tool was used to provide bulk and selective retrieval capabilities for event reporting. Limiting the tool's functionality like this was done to keep things simple, keep the grammar of the tool simple and easy to learn, and keep the memory and latency overhead low.

Examples of the custom tool queries and their XPath counterparts in Table 4-2 are given in Table 4-3. Similar queries to the ones in Table 4-3 are used in the evaluation section for the QoS scenario between XPath and the custom tool. These queries contain a varied number of filtering and merging operations to evaluate each tool's performance.

MD:{a[ ]}	a//b
MD:{a[ ]} FD:{b=1}	//a[b='1']
MD: {c[ ]} FD:{b=1}	//a[b='1']/c
MD: {c[ ],d[ ]} FD:{b=1}	//a[b='1']/(c|d)
MD: {c[ ],d[ ]} FD:{b=1, b=1}	//a[b='1']/c | //a[b='1']/d

Table 4-3 Custom tool queries and XPath equivalents

4.3.4 Scenario Measurement Set up and Analysis

4.3.4.1 Network Setup for Measurements

For the evaluation aspects of our QoS scenario the MPLS LSR and FEC-to-NHLFE MIBs are exposed as WS interfaces. For XPath 1&2 and the custom query tool management data are exposed as shown in Figure 4-1. The Apache Axis 1.4 SOAP toolkit is used to deploy WS interfaces using a Document/literal encoding style. For XPath 1.0 functionality, we use the Java API for XML Processing v1.3. XPath 2.0 is supported by SAXON 8.9. Java's 1.5.6 regex engine is used for the query tool. To measure traffic overhead for the third QoS scenario we use Linux's *tcpdump* utility. To perform latency measurements, the *currentTimeMillis( )* method of java is used. Each latency sample is calculated based on averaging the results of 10 measurements. Memory overhead was measured by monitoring the maximum consumption of memory (swap/RAM) of Linux PCs where the manager and the agent in Figure 4-1 are deployed. The manager and agent of Figure 4-1 were deployed on a 1000MHz/256MB RAM and 466MHz/192MB RAM machine respectively, thus simulating a lower end system for the agent. Both machines ran Red-hat Linux 7.3.

4.3.4.2 Measurements Analysis

The measurements we will analyse in this section involve traffic, latency and memory overhead for the third QoS scenario analysed in section 4.2.2. In this scenario we need to determine using any of the three tools and the MPLS traffic Engineering MIBs, the affected LSPs and SLSs after the interface of an ingress router of an MPLS network fails. To do this for the management model in Figure 4-1, three queries must be sent from the manager to the agent. The first query determines the interface indices of the LSPs associated with the failing interface. The second query uses the interface indices returned by the agent from the first query to determine the affected LSP IDs. In the third query the LSP IDs from the previous step are used to determine the affected SLSs IDs. Part of the queries, since they are quite large and cannot be listed as a whole, for XPath 1.0, 2.0 and the custom query tool are given in Figure 4-2, Figure 4-3 and Figure 4-4.

From the XPath queries in Figure 4-2 and Figure 4-3, it is possible to observe that apart from the required data, each query also retrieves the row identifiers of each row of the SNMP table from which data are extracted. For the WS using the custom query tool in Figure 4-1 , this is performed by default and as a result it increases latency overhead. Retrieving the row identifiers for multiple instance data in SNMP is mandatory (i.e. a Table). This way an SNMP manager and agent is able to distinguish table rows. As such for WS that use XPath v1.0 or 2.0 implementations, we retrieve the row identifiers explicitly as it happens for WS that use the custom query tool implicitly. This way we can keep the measurements for all tools relatively on the same terms.

XPath 1.0 Query 1	//mplsInSegmentEntry[mplsInSegmentInterface=ifIndex$_1$]/mplsInSegmentInterface |
	//mplsInSegmentEntry[mplsInSegmentInterface=ifIndex$_1$]/mplsInSegmentIndex |
	//mplsOutSegmentEntry [mplsOutSegmentInterface=ifIndex$_1$]/mplsOutSegmentInterface |
	//mplsOutSegmentEntry[mplsOutSegmentInterface= ifIndex$_1$]/mplsOutSegmentIndex

XPath 1.0 Query 2	//mplsXCEntry[mplsXCInSegmentIndex= mplsInSegmentIndex$_1$ or
	mplsXCOutSegmentIndex= mplsOutSegmentIndex$_1$ or ...]/mplsXCLspId |
	//mplsXCEntry[mplsXCInSegmentIndex= mplsInSegmentIndex$_1$ or
	mplsXCOutSegmentIndex= mplsOutSegmentIndex$_1$ or ...]/mplsXCOutSegmentIndex|...

XPath 1.0 Query 3	//mplsFTNEntry[mplsFTNActionPointer=mplsXCLspId$_1$.mplsXCIndex$_1$. mplsXCInSegmentIndex$_1$.mplsXCOutSegmentIndex$_1$ or ...]/mplsFTNDscp | //mplsFTNEntry[mplsFTNActionPointer=mplsXCLspId$_1$.mplsXCIndex$_1$. mplsXCInSegmentIndex$_1$.mplsXCOutSegmentIndex$_1$ or ...]/mplsFTNIndex

Figure 4-2 XPath 1.0 queries

XPath2 Query 1	//mplsInSegmentEntry[mplsInSegmentInterface=ifIndex$_1$]/ (mplsInSegmentInterface | mplsInSegmentIndex) | // mplsOutSegmentEntry[mplsOutSegmentInterface=ifIndex$_1$]/ (mplsOutSegmentInterface | mplsOutSegmentIndex)

XPath2. Query 2	//mplsXCEntry[mplsXCInSegmentIndex=mplsInSegmentIndex$_1$ or
	mplsXCOutSegmentIndex=mplsOutSegmentIndex$_1$ or ...]/
	(mplsXCLspId | mplsXCIndex | mplsXCInSegmentIndex |
	mplsXCOutSegmentIndex)

XPath 2.0 Query 3	//mplsFTNEntry[mplsFTNActionPointer=mplsXCLspId$_1$ .mplsXCIndex$_1$.mplsXCInSegmentIndex$_1$. mplsXCOutSegmentIndex$_1$ or ...]/(mplsFTNDscp | mplsFTNIndex)

Figure 4-3 XPath 2.0 queries

Custom parser Query1	{mplsInSegmentInterface[ ], mplsOutSegmentInterface[ ]}
	{value=ifIndex$_1$,value=ifIndex$_1$}

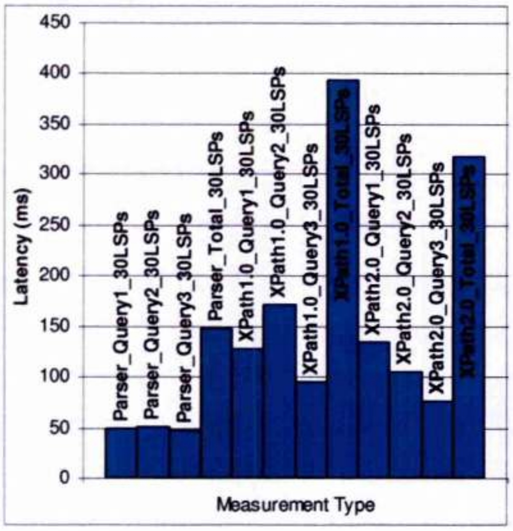
Custom parser Query 2	{mplsXCLspId[ ]}
	{mplsXCInSegmentIndex=mplsInSegmentIndex$_1$ OR
	mplsXCOutSegmentXCIndex=mplsOutSegmentIndex$_1$ OR ...}

Custom parser Query 3	{mplsFTNDscp[ ]}
	{mplsFTNActionPointer=mplsXCLspId$_1$.mplsXCIndex$_1$.
	mplsXCInSegmentIndex$_1$.mplsXCOutSegmentIndex$_1$ OR ...}

Figure 4-4 Custom based tool queries

In Figure 4-5 and Figure 4-6 we provide latency overhead measurements for a small and a big QoS network. Before analysing the results, we need to make some clarifications. The first clarification is that for both the small and the big network we retrieve the same data. This happens either for XPath or the custom query tool. The volume of data though, that needs to be searched when the size of the network increases changes. The exact figure of this change is thirty times. In addition to the above clarification it has to be noted that XPath implementations need to search a bigger volume of data than the custom query tool. This occurs for a simple reason. In XML, the context of each piece of information requires usually two element node tags. This means that for every single instance or for every multiple instance entry of data, two separate element tags are required. A different approach was followed for the custom query tool for multiple instance data. For the custom tool all the management data of each WS are stored in a linked list. In every linked list multiple instance data of the same type (like a column from a table) are described so as to appear as several entities with a single tag describing their context. The benefits of this are twofold. First, the volume of data required to be searched decreases since fewer tags to describe the data context are required. Second, the hierarchy of single or multiple instance objects connected with pointers in every linked list is simpler. Thus data of specific type can be found easier and faster (Figure 3-17). This has a big impact on latency.

From analysing the latency measurements in Figure 4-5 and Figure 4-6, it can be observed that when comparing tools together the latency of XPath 1.0 is bigger than that of the custom query tool with the difference ranging from 166% (small networks) to 1589% (big networks). For XPath 2.0 when comparing tools together this percentage varies from 116% to 968%. Comparing how the latency of each tool increases with respect to the data volume, the custom query tool latency increases only by 2.5 times when the network size and the volume of data increase 30 times. For XPath 1.0, latency increases between 13 to 19 times depending on the query used, and for XPath 2.0 between 9 to 18.5 times. Since the main difference between the query tool and XPath, is the parsers they use to handle data (raw data and XML respectively), we can realise that DOM is not scalable when handling an increasing volume of XML data. On top of DOM's inefficiency, as mentioned in the previous chapter the custom query tool and parser have two extra optimisations to minimise latency even further. The first optimisation is that the parser of the custom query tool instance inside each WS becomes aware of the type of objects the WS contains when the latter is deployed. This allows the custom query tool to refrain from searching for data that do not exist. The second optimisation is also present in XPath 2.0. The custom query tool supports merging of data located anywhere inside a WS MIB using the symbol ",". XPath 2.0 allows merging of data using the symbol "|" and as mentioned before, the latter allows path expressions such as (a|b|c)/d/e or /a/b/(c |d | e). XPath 1.0 does not allow the latter type of expressions. The fact that XPath 2.0 and the custom query tool allow merging of any data located anywhere in the MIB tree, allow them to perform sufficiently less filtering operations and search the document more effectively for merging operations than XPath 1.0 (XPath1: query1 (4xF-3xM), query2(24xF-3xM), query3 (12xF-1xM) XPath2-Custom tool :query1 (2xF-3xM), query2(6xF-4xM), query3(6xF-1xM) M=merging F=filtering). For this reason XPath 2.0 and the custom tool reduce latency more than XPath 1.0. Nevertheless XPath 2.0 is burdened from DOM's inefficiency to handle an increased volume of XML data. In addition XPath 2.0 is burdened from the need of DOM to hold a detailed structure of an XML document in memory (Figure 3-17). As such, as we can observe from Figure 4-5 and Figure 4-6, the latency of XPath 2.0 varies quite a bit when the number of filtering and merging operations changes. On the contrary, the latency performance of the custom tool for merging and filtering operations of varied complexity is not affected much. Each query for the custom query tool introduces similar latency overhead.



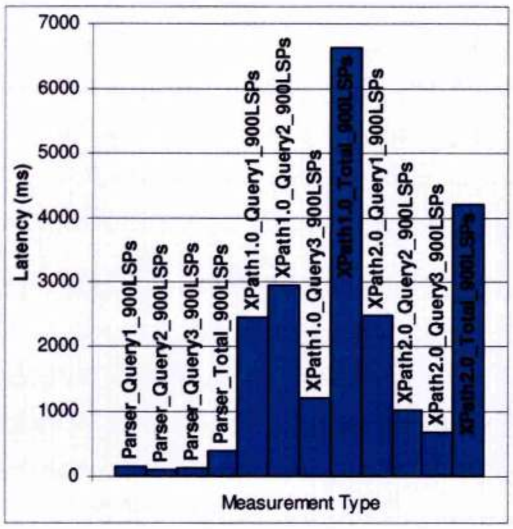
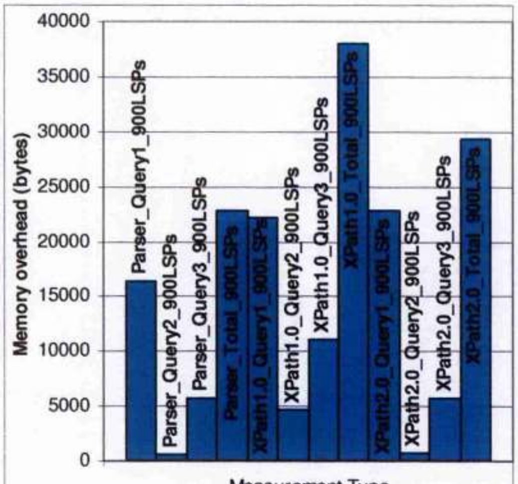


Figure 4-5 Latency measurements for small networks

Figure 4-6 Latency measurements for large networks

In terms of memory overhead the results are presented in Figure 4-7 and Figure 4-8. The memory consumption in these figures involves maximum run-time swap/RAM memory consumed for operations as a whole. The memory consumption does not involve only memory consumed by XPath 1.0 & 2.0 or the custom tool but also memory consumed from all processes related to the monitoring process i.e. the web server and its libraries, axis libraries to deploy and access WS, etc. The maximum system memory consumption though, can give us an indication of the memory footprint of each tool. For small networks the memory footprint of XPath 2.0 is the smallest of the three tools. XPath 1.0 follows and the custom query tool lies in the last position. The custom query tool consumes 28% (1.5MB) and 24% (1.4MB) more memory than XPath 1.0 & 2.0 respectively for small networks. This situation though changes for big networks. In big networks since the data volume increases, so is the memory requirement for processing or loading XML data in memory using DOM. As such XPath 2.0 & 1.0 consume 28% (6.4MB) and 66% (15.3MB) more memory respectively compared to the custom query tool. From the above it is evident that the verboseness of XML tags to describe each piece of data, and the cost of using DOM to search a more elaborate hierarchy of objects as that of an XML document, has its toll on XPath when the network size increases. Larger memory consumption though, when the network size increases, may not be a scalable option.

Comparing the two XPath implementations with one another in terms of memory overhead also reveals something not expected. XPath 2.0 consumes less memory compared to XPath 1.0 irrespective of the volume of data when the number of merging and filtering operations performed by the former is smaller (query2 and query3). On the contrary, when the number of merging and filtering operations is about the same or larger (query1) for XPath 2.0, the latter consumes more memory. This happens because inherently XPath 2.0 is a much heavier application than XPath 1.0. This is due to the increased functionality it has compared to its predecessor. Still the increased expressiveness that is inherent to XPath 2.0, allows it to perform sufficiently less filtering operations for some queries and search the document more effectively for merging operations. As such in cases where increased expressiveness is required, XPath 2.0 will consume less memory. If though queries have to be formed that cannot take advantage of its increased expressiveness, memory overhead for XPath 2.0 will be equal to XPath 1.0 or greater.



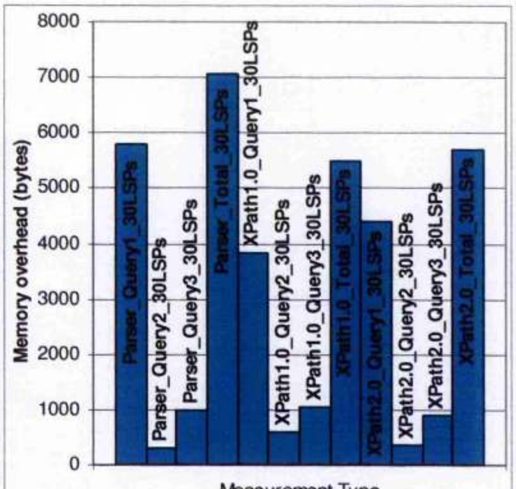


Figure 4-7 Memory overhead for large networks

Figure 4-8 Memory overhead for small networks

XPath implementations impose more traffic overhead than the custom query tool (Figure 4-9: query1- C:2327 X1:2777 X2:2670, query2- C:2346 X1:3807 X2:3203, query3- C:2442 X1:2941 X2:2620, C=custom tool X1=XPath1 X2=XPath2). This happens because data representations as well as queries are different for XPath implementations. For the custom query tool, data are retrieved for example in the format of <xxxxDataTag> xxxxx.rowid$_1$ </xxxxDataTag>. We adopted this format because in various SNMP tools, the row identifier and the actual value of a piece of management information are printed together. In XPath 2.0 data are represented as <xxxxDataTag> xxxxx </xxxxDataTag>, <rowid$_1$DataTag> rowid$_1$ <rowid$_1$Data Tag> since this is the manner data are organised and extracted from an XML document. It is feasible to have the same representation of state data as for the custom query tool for all the XPath implementations. This though would require extra processing overhead for XPath implementations. This would happen because when the data and the row identifier element nodes are returned after an XPath query, these data would have to be processed even further to structure them as in the custom query tool. As such latency overhead will increase. Also from Figure 4-9 we can observe that the queries of the custom tool are more compact even than the queries of XPath 2.0. This happens because the custom query tool retrieves row identifiers of multiple instance data by default, without the need to request these explicitly. XPath 1.0 queries are even less compact that the XPath 2.0 ones because path expressions similar to /a/b/(c |d | e) cannot be used. Based on the above it is evident that XPath implementations incur a larger traffic overhead in certain scenarios. Representing data more compactly and having less verbose queries though, should be an option for any query tool. This is especially required when polling based operations need to be performed frequently with a finer granularity, for example for some time critical monitoring tasks in QoS networks. In XPath implementations, this may not be possible for tasks that require measurements with finer granularity since the latency overhead for XPath implementations in certain scenarios should be reduced before this is feasible.

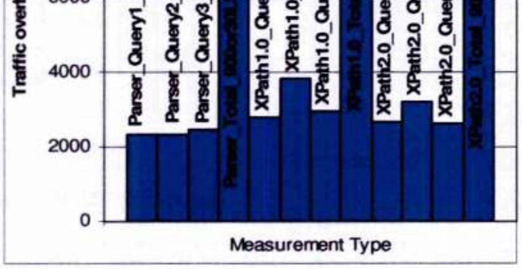


Figure 4-9 Traffic overhead for small and large networks

4.3.4.3 Discussion of the results

Comparing the performance of three WS-based tools for bulk and selective retrieval, we have shown as expected that our custom query tool is more scalable when handling certain scenarios. This was expected because several optimizations were performed on our custom query tool to scale its performance for monitoring and event reporting. On the contrary general tools such as XPath can be used for information retrieval as suggested by various standards (MUWS, WS-Management) but they may suffer from various problems. This investigation has shown that the major bottleneck of XPath implementations is using DOM to handle increasing volumes of management data stored in XML. DOM and XPath operate in order to find specific portions of XML data by keeping an elaborate hierarchy of XML nodes (element, text, attribute nodes etc). In many cases, such as for multiple instance data of the same type, this hierarchy could be simplified. This way the volume of data that needs to be searched but also the number of search operations required would be reduced. Our custom query tool uses this approach and as a result the latency overhead of monitoring operations is minimised. In addition to keeping an elaborate hierarchy of XML nodes, DOM's memoryless state adds to the problem of increasing latency overhead even further. DOM is a general parser that is not aware beforehand of the types of data types that exist inside an XML document. As such, in cases where XPath has to process queries for inexistent data, DOM can not help XPath to reject these queries instantaneously. This increases latency overhead. On the contrary, our custom query tool keeps state of the type of data that exist inside a WS and thus rejects queries for inexistent data immediately. This way latency overhead is reduced. Moreover due to their dependence on DOM, the performance of XPath implementations varies based on the number of merging (bulk) and filtering operations that need to be performed. The situation for XPath 1.0 might be worse, since due to its lack of expressiveness, it usually may require performing more of these operations.

For the above reasons and though XPath implementations were suggested by various management standards for monitoring and event reporting, it seems that a lot of work has to be performed in improving the performace of these tools if they are going to be used for bulk and selective information retrieval in situations where time critical tasks need to be performed. Nevertheless by comparing XPath 1.0 & 2.0 we have seen that increasing the expressiveness of XPath 2.0 is a step in the right direction in order to decrease latency and memory consumption. A lot more improvements are required. A different memory management scheme may be a good option instead of the one DOM is using since in many monitoring scenarios it is not required to keep all the data of an XML document in memory. Different ways to represent the structure and the hierarchy of an XML document for multiple instance data in DOM may also prove beneficial. Different strategies to process XML data and better memory management is a must in order to reduce the overhead that XML tags put on any query tool that handles XML payload. Maybe the option of saving management data in smaller size documents, since the performance of XPath is better when handling small volumes of data, is a step in the right direction. Then employing an intelligent scheme to search the appropriate documents each time can be beneficial in reducing the latency and memory overhead. In addition, employing a schema specific parser other than DOM might be plausible and required. Schema specific parsers can employ a better memory management scheme since they are aware of the type of data they need to process. Such a parser should load in memory only the data required in order to process an XPath query to completion and not the entire document containing the management data. As such memory consumption can be reduced.

Developing custom tools to meet management objectives for information retrieval in monitoring and event reporting might be a solution but it still is a solution that can have several limitations. Processing raw data to keep latency and memory overhead low does not allow us to deploy a common way to access data as through an XML document. This increases development time and makes the process of building interfaces to expose management information a more time consuming process. In addition custom implementations such as our query tool may have limited functionality and expressiveness than industry based tools. Moreover XPath is a well known tool that many are familiar with. Learning a new tool even with a simpler syntax might not be desirable. As such improving on the performance of standard tools such as XPath is imperative. This is especially true for interoperability purposes since XPath is a standard whereas custom tools are not.

Nevertheless our custom tool and parser implementation and scenario have shown that XPath implementations have to be improved if they are going to be used for time critical monitoring and event reporting tasks. Using XPath for monitoring operations, in systems such as those of a network offering QoS guarantees, should not be considered at this time if changes are not performed to increase DOM's performance.

4.4 Comparing our custom WS-based Framework with SNMP

In this section we will investigate the performance of WS-based monitoring against SNMP. To do this, performance will be compared upon the three QoS scenarios introduced in section 4.2.2. As explained in section 4.2.2 all of these scenarios can benefit from the use of a query tool. Using a query tool for monitoring can be invaluable in these scenarios. This happens because these scenarios reflect monitoring cases where it is not necessary to retrieve the entire state of a device or where it is necessary to retrieve the state of a device in a specific manner. As such bulk and selective retrieval mechanisms are necessary for these scenarios. Previous researchers have not investigated, as explained in chapter two, scenarios where such mechanisms are required in order to delegate a series of data processing tasks from a manager to an agent. This is because they have examined only scenarios where data are retrieved sequentially or in a bulk manner with no information processing. As such any conclusions they made about the performance of WS for management and especially for monitoring may not reveal the entire picture . Using a query tool to delegate the task of processing management data from a manager to the agent will allow us to investigate how WS can perform compared to SNMP in such occasions. This way we will also be able to explore if the initial overhead that the verbosity of XML tags incur on WS performance can be overcome.

In the previous section we have shown that our custom query tool is a more scalable option than XPath for a number of monitoring scenarios where bulk and selective retrieval mechanisms are required. As such for the performance measurements of all three scenarios we are going to use only our custom query tool as part of our custom framework operations.

For SNMP measurements it would be possible to use the Script or the Expression MIB in order to provide the bulk and selective retrieval capabilities that our query tool offers. As explained in chapter two though, the use of these MIBs for monitoring or event reporting raises a lot of issues. The use of the Expression or the Script MIB often introduces performance issues since in many cases even the specifications of these MIBs state that the latter MIBs do not represent a good trade off especially for monitoring. The Script MIB presents increased development costs due to the management of management problem. The capability of the Expression MIB to provide access to remote resources is limited. In addition the Expression and the Script MIB present integration problems with current devices. Moreover implementation and open source software that supports these MIBs is either limited or not existent. Finally, using the Expression MIB to deploy bulk and selective retrieval mechanisms is impractical and increases the footprint of monitoring operations. Based on the above we decided that for SNMP measurements we are going to use only the standard mechanisms of SNMP that provide capabilities for sequential or bulk retrieval of state data. The reason for this as we will show later in our analysis, is that if distributed management extensions of SNMP like the Expression MIB will be used, overhead is going to increase rather than decrease. This is attributed to some of the characteristics of these MIBs. It is possible to alleviate some of the problems that emanate from these characteristics but this would require revisions to these MIBs and work in the DISMAN charter responsible for the definitions of these MIBs has finished.

4.4.1 Monitoring with SNMP

Before any measurements performed for the three scenarios are presented and analysed, a few aspects on how SNMP measurements need to be carried out will need to be examined. It is important to (a) explain what software and which MIBs are going to be used for the measurements (b) analyse the steps required for making the measurements required for the scenarios (c) examine the granularity required for each one of the measurements. Analysing all these aspects is important as it will allow us to have a greater understanding of the measurement requirements. This way it is possible to understand whether SNMP or our WS-based custom framework can satisfy these requirements.

In terms of the MIBs required for the measurements of the three QoS scenarios we will use the MIBs listed in section 4.2.3. These MIBs can be used to perform five of the six passive measurements required for the first and second scenario of section 4.2.2. The LSR MIB can be used to perform the four PHB and LSP related measurements. The FEC MIB can be used to perform the SLS load measurement at the ingress router. For the sixth measurement SLS data have to be retrieved from the egress router. Thus the sixth measurement cannot be performed with the FEC MIB as this is only deployed at an ingress router. As such, this measurement is not considered in the scenarios that will be investigated later. Regarding the measurements of the third scenario, the LSR MIB will be used to find the LSP IDs that are affected by the failing interface of the ingress router. Then the FEC MIB will be used to find the affected contracts (SLS IDs).

In terms of the software used for SNMP measurements, we use a Net-SNMP agent and the AdventNet SNMP toolkit. For all scenarios, the software allows for bulk retrieval operations to be performed but selective retrieval mechanisms are not possible as plain SNMP does not support them. Distributed management extensions are also not applicable since as we will show they introduce more overhead than they save but also because the Net-SNMP agent does not support these MIBs.

The steps for retrieving SLS, PHB and SLS related data for the first and second scenario is explained in the next section. The process of retrieving data for the third scenario was explained in section 4.3.4.2 when analysing the steps required in order to carry out the performance measurements between XPath and our custom query tool.

4.4.1.1 Retrieving passive measurement data with SNMP for the first and second scenario

For the first five passive measurements using SNMP, a number of MIBs, tables and entries must be accessed. For LSP measurements, the manager must query the agent for the following data:

- LSP-IDs in order to determine how LSPs are organised in a specific table. The row identifiers (row IDs) of the relevant table are also returned so as to perform the next step.

- LSP load at the ingress (LSP L-I) or throughput at the egress (LSP T-E) from tables holding statistical information using the row-IDs returned in the previous step.

For PHB measurements the process is the following:

- Retrieve the PHB IDs in order to determine the PHB each LSP belongs to.
- Query for the throughput (PHB T) or packet discards (PHB D) for each PHB, based on the row IDs retrieved from the previous step.

To determine SLS load, the manager must do the following:

- Query first the relevant agent in order to find out which LSPs each SLS is associated with (SLS IDs, Differentiated Service Code Point field -DSCP- and LSP IDs).
- Based on the row IDs obtained, the manager can access the statistics table for each LSP in order to compute the load for each traffic contract (SLS).

Determining the row identifiers of LSPs, SLSs and PHBs as a first step is an essential and required process for these measurements. For all measurements it is assumed that the manager is aware of the network topology through a topology repository. This implies that the manager, apart from topology information also has knowledge of the ID of every contract, the ID of each LSP at the ingress and egress routers, as well as the various traffic classes supported. Data queries in SNMP though are based on the exact location of tabular information. As such, the manager needs to initially determine the order in which LSP, PHB, and SLS data are organised in the SNMP tables and then retrieve the information relevant to the passive measurement scenarios.

4.4.1.2 Granularity of SNMP operations on passive measurement data

Another aspect that requires special consideration is the sampling granularity for each type of data. Since each class of service in a QoS network has different requirements, it would be expected that different sampling frequencies should be applied to the traffic of each class. A premium class of service, for example, requires more frequent measurements than a best-effort service. Using different sampling frequencies for different SLSs though would make the monitoring architecture more complex. For this reason, we use a selection of three different sampling frequencies based only on the type of data that needs to be retrieved. Determining, for example, the IDs for each LSP requires a relatively long sampling period (long granularity, in the order of days or weeks) since the same LSP configuration is typically retained over a relatively long period of time (a provisioning period). The value of tabular objects referring to PHB and SLS specific data, on the other hand, may change more frequently. This occurs due to failures on MPLS-capable interfaces, load balancing to meet the requirements of each traffic contract, etc. During the load balancing process the traffic on heavily utilised LSPs is assigned to less utilised ones. Solving such problems might be possible without reconfiguring the entire network, by routing the traffic mapped on a failing interface or on a heavily utilised link to another. As such, PHB and SLS related information change more frequently and the polling period for these data is of medium granularity, in the order of 5-10 minutes. Lastly, sampling periods for more dynamic data such as load and throughput are in the order of 5-20 seconds (short granularity).

4.4.2 Monitoring with WS

For WS the LSR and FEC MIBs have been re-implemented from scratch as native Web Services, following the tree structure described in their RFCs. SNMP agents are not re-used. All data are stored in single values, tables and linked lists (raw format). To represent the MIB hierarchy, each programming language object representing a MIB's data is connected to another with special pointers. All these data are stored in a linked list allowing the navigation of the data hierarchy. Each MIB is deployed as a WS. Every WS exposes through its interface the three functions of our custom framework. As a result a manager can have access to different portions of the MIB's data (single instance, multiple instance, and all data view, Figure 3-16).

The steps and the granularity that is required to retrieve passive measurement data for the three scenarios using WS are the same as the ones in sections 4.1.1.1 and 4.1.1.2 for SNMP.

In terms of software, our custom query tool is used to perform the filtering and merging operations for bulk and selective retrieval of state data. Apache AXIS version 1.4 is used to build and deploy all WS that the architecture we analysed in chapter three requires.

4.4.3 Measurements

4.4.3.1 Evaluation Setup

For the evaluation aspects of our scenario, a big number of LSPs needs to be setup for some of the measurements. As this is difficult to be achieved in a small test-bed, we resorted to other means for evaluating the performance overhead for SNMP. Thus in order to calculate the traffic overhead for SNMP, the average size of each message is calculated by looking into the message and analysing the size of its subparts. This analysis is presented in the next section. In the next section we will also explain that every traffic overhead measurement of SNMP has a maximum and a minimum value. As such the traffic overhead for SNMP was calculated using the average for each type of measurement between these two values. To compute latency for SNMP a similar number, size and type of objects as that in the MPLS MIBs, need to be extracted for each type of measurement. As such we use the Advent-Net SNMP v3.3 toolkit to access a Net-SNMP v. 5.0.2 agent for data of the same size, number and type as that required for the measurements of the scenarios. Latency measurements for SNMP were performed using Java's currentTimeMillis() function by averaging 20 measurements for each sampled result.

For WS the Apache Axis 1.4 SOAP toolkit was used to deploy the LSR, the FEC and the Interfaces Group of the RFC 1213 MIB as WS with the same information as in SNMP. The information of each MIB is replicated exactly as it would be in a router. All MIBs were deployed using an RPC/literal encoding style so that the verboseness of XML tags is reduced and traffic overhead as well as coding latency is minimised. For the parser of the custom query tool that all WS use to support selective or bulk information retrieval, we used Java 1.4.2.10 and its regular expression matching engine. Traffic and latency overhead and the machine setup for WS measurements are the same as in section 4.4.2.1. Latency measurements for WS were performed by averaging 20 measurements for each sampled result.

4.4.3.2 SNMP traffic consumption calculation

Since a big number of LSPs need to be setup for some of the measurements we needed to resort to other means for evaluating the traffic overhead for SNMP. Thus in order to calculate the traffic overhead for SNMP we need to look into each SNMP message and analyse the size of its subparts. SNMP messages use ASN.1 syntax [37]. When retrieving data, a SNMP manager can use 3 types of messages or Protocol Data Units (PDUs): *Get*, *GetNext* and *GetBulk*. All three types of PDUs consist of several fields including the version, the community, the PDU type, the request ID field, and the variable-bindings field. Get or GetNext also include an error status field and an error-index field, whereas GetBulk includes a Non-repeaters and a Max-repetitions field. In the experiments performed:

- The version field is v1/v2.
- The community field is "public".
- The PDU type is GetNext or GetBulk.
- Object requests will not exceed 2000 and so the request-ID field will not exceed this value.
- The error-status field can take only 5 values.
- Since the error-index shows the variable in the variable binding list that caused an error and the number of variables we retrieve in a packet will not exceed 127, this value cannot exceed this number
- For the experiments the Non-repeaters=0 and the Max-repetitions field will not exceed 2000.

By calculating the size of each field using the Basic Encoding Rules in [170] and taking into account how the measurement requirements affect the size of each field we can calculate the size of each SNMP message. Based on a previous study on the traffic of SNMP [122], the size of an SNMP request or response message can be computed as follows:

$$L_{get,\ getNext,\ getBulk} \approx 27 + n * (6 + L_1 + L_2) \qquad (4.1)$$

In equation 4.1, L_1 is the size of the Object Identifier (OID) of a variable, L_2 is the size of the variable value itself, n is the number of OIDs to retrieve, and the numbers 27 and 6 relate to the encoded size of the message subparts. The request PDU in all three scenarios contains one or two OIDs of different types of SNMP data. In the case of GetBulk the number of OIDs is different than the number of objects returned. Therefore, the size of SNMP operations, if n_1 objects are retrieved and a single OID exists in the request packet, is given in equations 2 and 3.

$$L_{get,\ getNext} \approx n_1 * (54 + 12 + 2L_1 + L_2) \qquad (4.2)$$

$$L_{getBulk} \approx 54 + 1 * (6 + L_1) + n_1(6 + L_1 + L_2) \qquad (4.3)$$

Table 4-4 presents the size of L_1 and L_2 which has been calculated for each measurement type. These values can be substituted in equations 2 and 3 to determine the maximum, minimum and average traffic of SNMP operations for all scenario measurements. In the traffic overhead results for SNMP in the next section, we list average traffic between a maximum and a minimum value for each type of measurement.

Measurement Type	L1	L2
LSP IDs	16-19 (Max 16000 LSPs)	6 (CR-LDP)
LSP Load Ingress (L-I) Th/put (T-E) Egress	14-15 (Max 16000 LSPs)	1-4 or 1-8
PHB Th/put (T) and packet Discards (D) and SLS Load (L)	14-15 for each (Max 16000 LSPs)	1-4 or 1-8 (L) and 1-4 (D)
PHB IDs	14-15 (Max 16000 LSPs)	14
SLS IDs	14-16 (Max 16000 LSPs)	1-3 (Max 16000 LSPs)
SLS LSP IDs	14-16 (Max 16000 LSPs)	16-20 (Max 16000 LSPs)
Interface IDs	14-16 (Max 16000 Ifs)	1-4

Table 4-4 L_1 and L_2 length analysis for all scenarios

4.4.3.3 Scenario Measurements

Before analysing the measurements for scenario one and two it is imperative to highlight a few aspects. For measurements of scenario one but also for scenario two the following apply:

- For both WS and SNMP, PHB packet discards and throughput (T+D) are retrieved by asking for both variables in the request packet (2 OIDs for different types of data in the request packet).

- For SNMP, SLS IDs and SLS LSP IDs (SLS+SLS LSP IDs) are retrieved by asking for both variables in the request packet. For WS only SLS LSP IDs are retrieved since processing occurs at the agent side and the SLS IDs need not to be retrieved for processing them at the manager.
- In the measurement figures we differentiate between the WS PHB and SLS measurements between scenario 1 and 2 by incorporating a filtering flag (denoted *nof* / *f* respectively).
- LSP measurements are performed in the same way for both scenario one and two as no processing is required due to the way data are organised in tables.
- For SNMP, latency is about the same for all types of objects retrieved for scenario one and two and depends roughly on the number of objects retrieved and their location in the MIB tree. As such in order to make the latency figures for scenario one and two (Figure 4-10, Figure 4-17) more readable, we present only average latency plots for all measurement types in Table 4-4. The latency values in these plots depend on the type of request packet (GetNext/GetBulk), the number of OIDs that refer to different types of data in the request packet (1 or 2 OIDs for the experiments) and the ordering of the OIDs in the request packet (random, sequential).

4.4.3.3.1 Scenario One: Bulk Retrieval

In this scenario data are retrieved with SNMP either with consecutive GetNext operations or with a single GetBulk operation. For WS, data are retrieved with a call to one of the operations of our custom framework analysed in chapter three. The operands of each call (parser queries) are appropriately interpreted by the parser of our custom query tool to retrieve data in a bulk manner similar to how GetBulk performs the same thing. For this scenario traffic overhead for LSP, PHB and SLS measurements are presented in Figure 4-11 to Figure 4-16, whereas latency results are provided in Figure 4-10.

From the graphical representations (Figure 4-11-LSP-IDs Figure 4-12-LSP load or throughput) it is evident that WS start producing less traffic than SNMP's consecutive GetNext operations if more than 30 objects are retrieved. This occurs because with the RPC/literal encoding, XML tags are less verbose and thus the initial overhead of HTTP/SOAP can be overcome. Still each XML node inside the body of the SOAP message for WS consists of two element nodes and a text node representing the value of the element node. In GetBulk the return message contains for every piece of information an OID (the equivalent of a single XML element tag) and a value (the equivalent of the text node). Thus it is not possible to overcome with WS the traffic overhead of a GetBulk operation. It is possible to reduce the traffic overhead of WS even further in order to match the performance of the GetBulk operations if we use empty XML element nodes. In such a scheme the context information of state data would be represented with one element node whereas the values of management information would be given with an attribute node within the empty element node. Empty element nodes can be used though only to represent leaf node data of a management information tree. In the information model of SNMP this is possible and plausible because only leaf nodes contain management information and only leaf nodes contain accessible management data. In other information models this may not be true though. As such in our WS custom framework we do not use the empty node element scheme even though we use SNMP MIBs.

Considering latency measurements that involve retrieval of a single type of data (GET 1 OID nof), the performance of WS is better than that of consecutive GetNext operations with SNMP (Figure 4-10). Although SNMP encoding is faster, most SNMP agents do not support caching [122]. The latter inhibits SNMP performance. When two types of data are requested (GET 2 OIDs nof), the SNMP GetNext operation produces even more latency as more data must be retrieved from different locations in the MIB tree. Latency in this case increases more abruptly than in the case where a single type of data was retrieved (Figure 4-10 SNMP GET 1 OID nof, SNMP GET 2 OID nof). This emphasises how the lack of caching capabilities affects performance. Regarding GetBulk, the latency is better than that of WS. In the case of requesting two variables though, the gradient of the GetBulk curve (Figure 4-10 SNMP GETB 2OIDs nof) is higher in some cases than that of WS (Figure 4-10 WS T+D nof). Thus, when the number of objects retrieved exceeds 1000, the absence of caching makes even GetBulk's latency worse than that of WS in some cases (Figure 4-10 WS T+D nof, SNMP GETB-2OIDs nof).

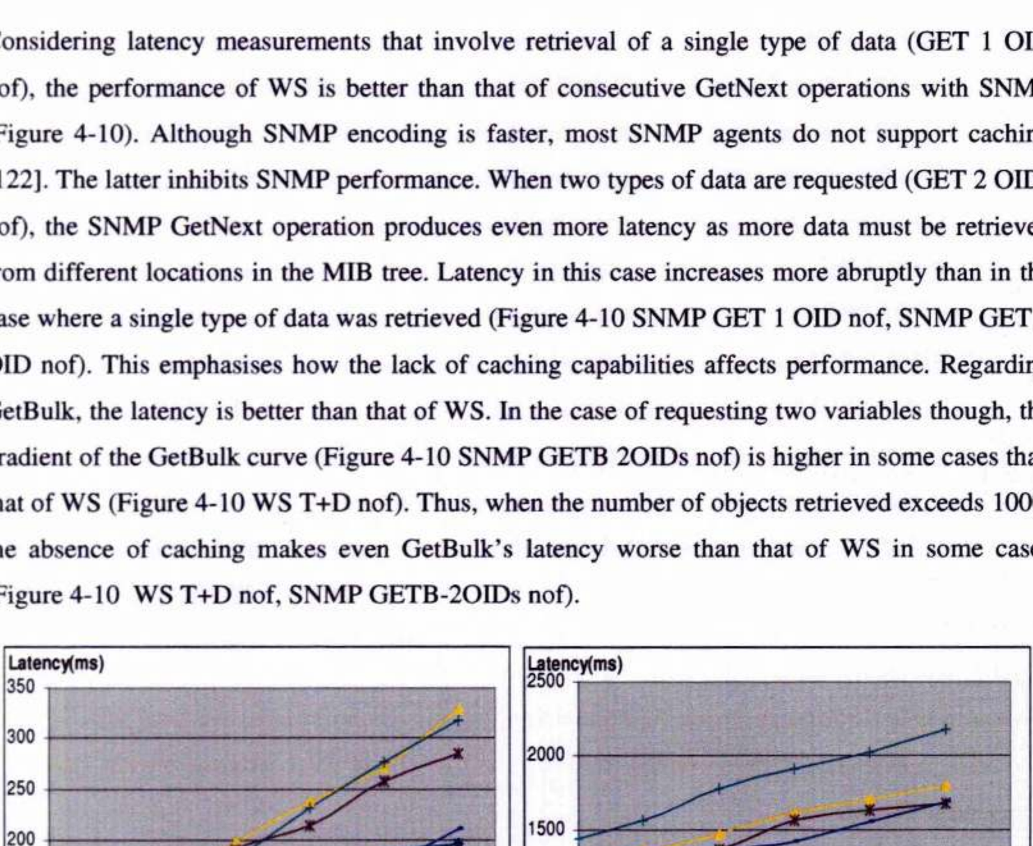


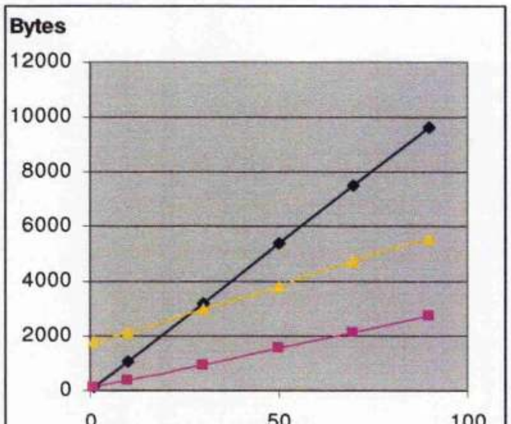
Figure 4-10 Latency for all measurements of scenario 1 for small and large data networks

4.4.3.3.2 Scenario Two: Data Processing

In this scenario the same information for the passive measurements of a QoS network as in the previous scenario is retrieved. This time though for WS, PHB and SLS related data are processed at the agent using the parser of the custom query tool. This way, the manager receives state data on a per PHB or SLS basis. As such, load, throughput or packet discards can be computed by the manager without any processing at the manager side. This approach is better because although the manager is usually hosted in a high-end system, it will have to access many agents to retrieve PHB or SLS related data. Thus the manager runs the risk of being easily overwhelmed by the amount of processing required. Our custom query tool allows us to distribute the load to several agents. This is feasible today especially since the myth of the dumb agent is no longer valid [55]. Using WS with tools for distributing the monitoring load may improve performance of WS monitoring operations in some cases as we will see below.

Before analysing the measurements of this scenario some aspects need to be highlighted. As such it must be noted that for PHB measurements the LSPs in each router are assigned to 6 different traffic classes. Also for SLS measurements we decided to investigate two cases. The first case concerns individual SLSs, each bound to a different LSP (1LSP/C, C=contract), for which we investigated traffic and latency for 1 to 300 Contracts (C). In the second case we investigated a Virtual Private Network (VPN) scenario for the same number of contracts where each SLS in the ingress router is bound to 3 different LSPs (3LSPs/C).

In all the measurements of these scenario WS perform better than successive GetNext operations in terms of traffic overhead, but worse than GetBulk. It is worth analysing though certain aspects, in order to show that if SNMP was used to retrieve data on a per PHB or SLS basis the performance of the latter would be worse.



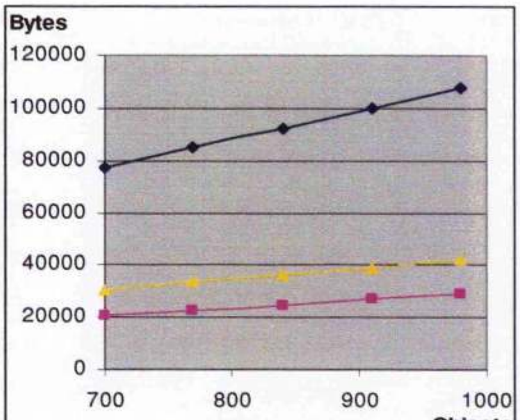


Figure 4-11Traffic overhead for retrieving LSP IDs, scenario 1&2, small and large networks

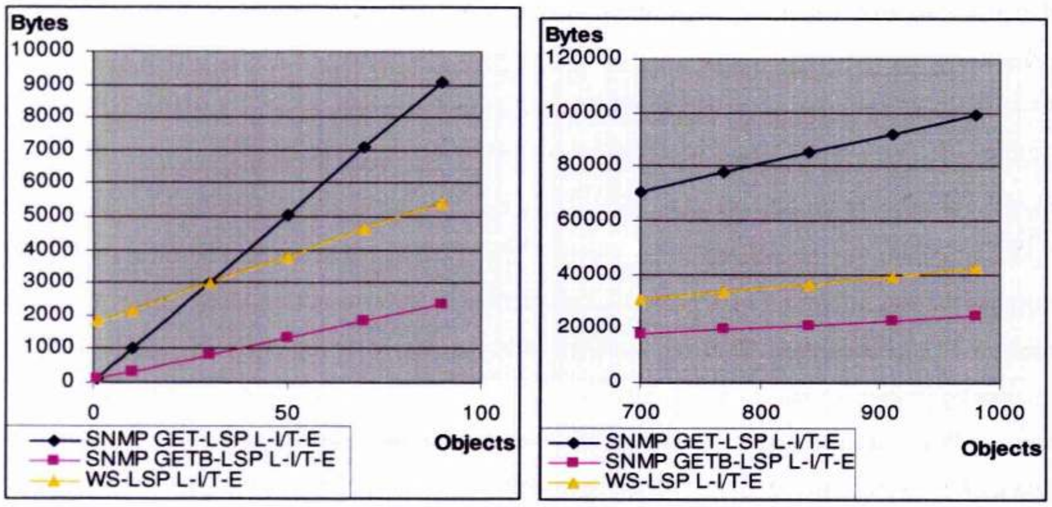


Figure 4-12 Traffic overhead for retrieving LSP load/throughput, scenario 1&2, small and large networks

To acquire data on a per PHB basis using WS, the following data queries must be sent to the agent:

$$\{\text{mplsInSegmentTrafficParamPtr}[\,], \text{mplsInSegmentTrafficParamPtr}[\,], \ldots\} \quad \text{(MID query)} \qquad (4.4)$$

$$\{\text{value} = \text{OID}_1, \text{value} = \text{OID}_2, \ldots\} \quad \text{(FD query)} \qquad (4.5)$$

$$\{\text{mplsOutSegmentPerfHcOctets}[id_1, id_2, \ldots], \text{mplsOutSegmentPerfDiscards}[id_1, id_2, \ldots]\} \quad \text{(MID query)} \qquad (4.6)$$

Analysing the traffic overhead in Figure 4-13 (PHB IDs f) it is evident that the traffic incurred for discovering PHB IDs is roughly the same as in the previous scenario. In terms of latency we observe that an increase of a maximum of 300 ms occurs when 980 LSP IDs must be retrieved (Figure 4-10-PHB IDs nof & Figure 4-17-PHB IDs f). This happens as the relevant tables need to be searched 6 times one for each traffic class. This increase in latency using WS could be avoided if the relevant tables of the MPLS MIBs could be searched only once. Our custom query tool though does not allow us to use more flexible and expressive FD queries than the ones used in expression 4.5. Thus our query tool does not allow us to limit the number of searches and still be able to receive data on a per PHB basis. Consequently, latency can improve if our query tool is optimised to support more flexible queries that allow us to be more expressive. For throughput and packet discards, the traffic overhead also increases by a maximum of 8000 octets for 980 LSPs (Figure 4-14 PHB D+L f & nof). The latter increase in traffic overhead occurs because the manager needs to access throughput on a per PHB basis. The latter implies that the WS manager has to include in the request packet all the row identifiers for each LSP segment assigned to each PHB. Despite the extra overhead, throughput (PHB T) and packet discards (PHB D) for each PHB are fairly easy to compute by the manager when data are retrieved this way. Accessing throughput (PHB T) and discards (PHB D) on a per PHB basis also increases latency by a maximum of 200ms when the number of LSPs inside an MPLS router is 980 (Figure 4-10-PHB T+D nof & Figure 4-17 -PHB T+D f). Nevertheless, for all PHB measurements in this scenario, the granularity requirements for polling data from the WS agent are satisfied (5-10 minutes for PHB IDs and 5-20 seconds for T+D).

If SNMP was used without distributed management extensions (no facilities for processing data at the agent side) to compute load and packet discards on a per PHB basis it is possible that the granularity requirements for PHB measurements could not be met. This happens because in order for a SNMP manager to compute load and packet discards on a per PHB basis, it must follow a two step procedure. In the first step of this procedure the SNMP manager must process data from a number of agents in order to discover to which PHB the row entries in the statistics table of each LSP segment belong to. Even for a high end system like the manager, processing data for the first step from a series of routers that have quite a few LSPs can be prohibitive (i.e. for a series of 500 routers/devices which have 200 configured LSPs assigned to 6 traffic classes (PHBs) processing of the 200 LSPs takes 72 ms to complete on a 1000 MHz/256MB PC for all 6 traffic classes, thus it would take the manager 72x500=36000msecs=36secs to complete the above step). In the second step after processing the data from the previous step, the SNMP manager must include in the request packet it sends to query for throughput or discards on a per PHB basis, the row identifiers of each LSP segment in order. The latter though even if the GetBulk operation of SNMP was used for the request message of the second step, would increase the traffic and latency overhead of such an operation by a big amount compared to using GetBulk to retrieve the same amount of objects using the Max-Repetitions field sequentially. In Figure 4-17 we can observe that if we retrieve data in a random way from an SNMP MIB table as it would be required to retrieve throughput or discards on a per PHB basis, the latency of the GetBulk operation increases (Figure 4-17 GETB 1or 2 OIDs random). More specifically, the traffic overhead of GetBulk will become bigger than that of the operations of our WS custom framework for the same type of PHB measurements when more than 420 objects will be requested (Figure 4-13 GETB-PHB IDs random, Figure 4-14-GETB-PHB D+L for load and discards).

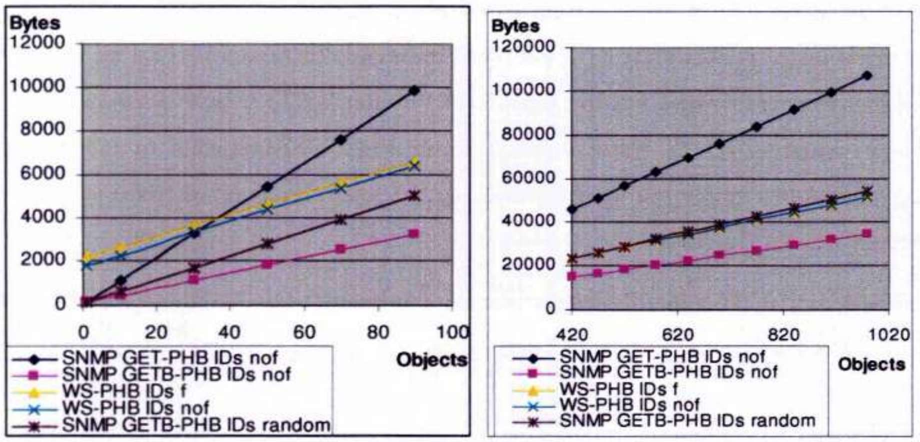


Figure 4-13 Traffic overhead for retrieving PHB-IDs, scenario 1&2, small and large networks

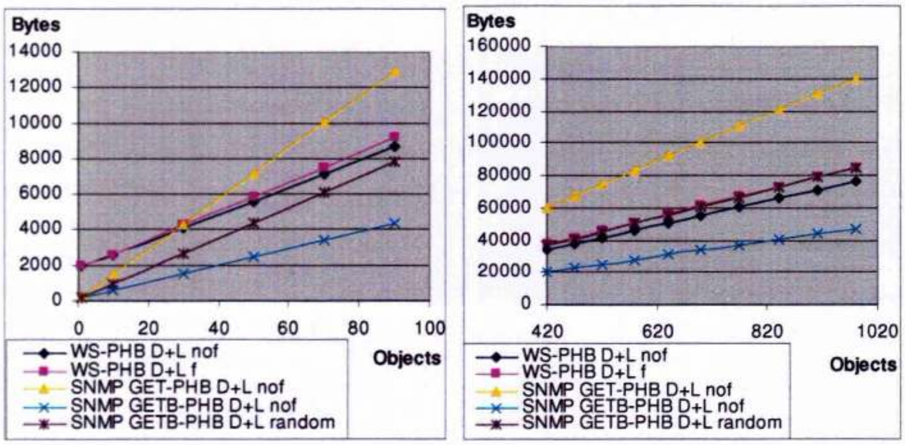


Figure 4-14 Traffic overhead for retrieving PHB discards and load, scenario 1&2, small and large networks

Even with distributed management extensions such as the Expression MIB that could provide SNMP with bulk and selective retrieval mechanisms, use of the latter would not be a good trade-off. This is because usage of the Expression MIB to retrieve load or packet discards on a per PHB basis would incur considerably more overhead than if SNMP was used without distributed management extensions. This would happen because in order to retrieve load or packet discards on a per PHB basis using the Expression MIB, the latter would also require from the SNMP manager to follow the first step of the procedure described previously in order to determine to which PHB the row entries in the statistics table of each LSP segment belong to. This as shown previously could be prohibitive when the manager has to process data from a series of agents. Even after determining to which PHB each LSP belongs, in order for the manager to calculate the throughput and packet discards for each PHB, the latter has to set a number of parameters in the tables of the Expression MIB by following a two step procedure. In the first step of this procedure the manager has to set in the expObjectTable of the expression MIB a series of four parameters (4

OIDs expObjectID, expObjectWildcard, expObjectSampleType, expObjectRowStatus) for each LSP throughput or packet discards parameter that will be used in order to calculate PHB throughput or packet discards. Even though the expression MIB permits wildcarding to set all the LSP throughput or packet discards parameters in the expObjectTable that will be used in order to calculate PHB throughput or packet discards in one go, this is not possible in this case because the order according to which each LSP is assigned to a PHB in the mplsXCTable of the LSR MIB is random. As such the LSP throughput or packet discards parameters that will be used in order to calculate PHB throughput or packet discards have to be set in the expObjectTable explicitly. In the second step the, manager will have to set in the expValueTable of the expression MIB a separate expression to calculate the PHB throughput or packet discards for each traffic class by referring to the expObjectID parameters of the expObjectTable. After completing the two steps described, the manager can retrieve using a getNext operation all the values in the expValueTable to acquire PHB throughput or packet discards. Just for the first step, usage of the expression MIB would roughly quadruple the monitoring overhead (4 OIDs and 4 values for each LSP throughput or packet discard parameter have to be set) in order to calculate PHB throughput or discards compared to using SNMP without distributed management extensions (1 OID and 1 value for each LSP throughput or packet discard parameter), not even calculating the latency overhead incurred in this step for setting such a big number of objects in the expression MIB. In addition for each object in the expObjectTable, the expression MIB would have to query the agent hosting the data referenced in the expObjectTable with a series of consecutive getNext operations (1 OID and 1 value for each LSP throughput or packet discard parameter). This increases latency and traffic overhead even further. This means that in order to calculate throughput and discards on a per PHB basis using the expression MIB, it might be necessary every few minutes to change the order of the expObjectID parameters used in the expressions of the expValueTable. It is evident from the above that using the Expression MIB for calculating PHB throughput or discards may not be plausible or practical. As such it would be nice if MIBs such as the expression MIB would be updated to support more features to handle scenarios such as the above gracefully. Unfortunately the DISMAN charter who could do these updates has finished its work.

Using the Script MIB to calculate throughput or packet discards on a per PHB basis apart from the security issues may also not be a very lightweight option. In devices running at high speeds, a simple script would need 15 MBs of memory just to run distribute, update and control it [50], [51], ignoring the data that would have to be processed by it. Our custom query tool requires for the Java libraries of Apache AXIS, the Java libraries of tomcat and the query tool itself not more than 6.5 MBs for queries when the volume of data to be processed is small and 16 MBs for queries when the volume of data is extremely large (Figure 4-7, Figure 4-8).

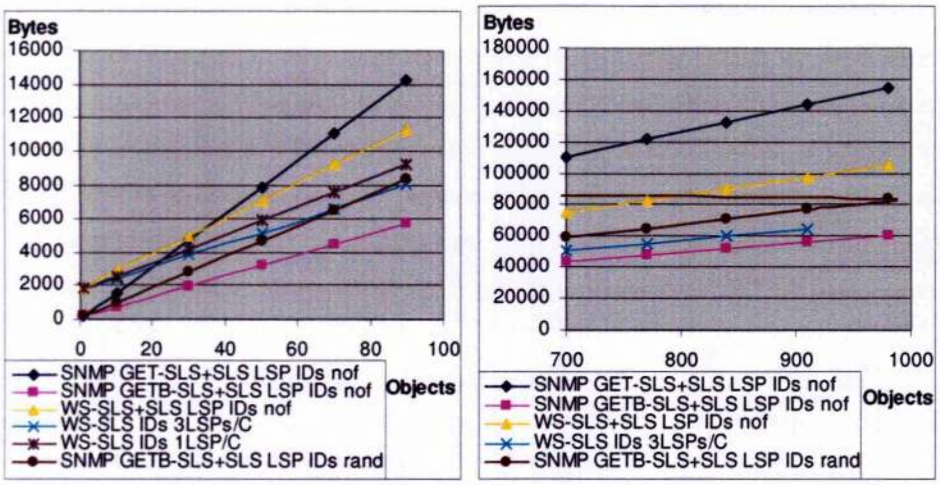


Figure 4-15 Traffic overhead for retrieving (SLS)/(SLS LSP)-IDs, scenario 1&2, small and large nets

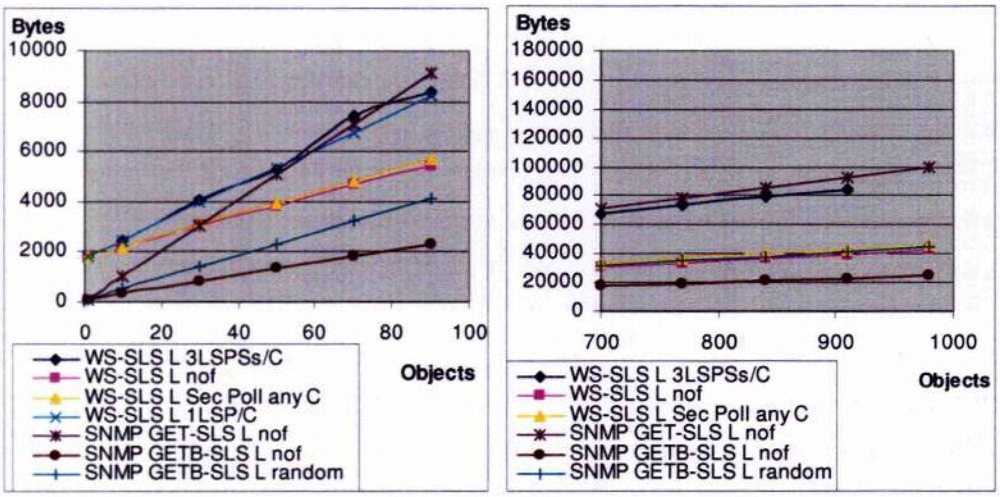


Figure 4-16 Traffic overhead for retrieving SLS load, scenario 1&2, small and large networks

Analysing the SLS measurements with WS for the second scenario is a bit of a more complicated issue as it was for PHBs. For this aspect of the second QoS scenario we need to explain first that the WS manager needs to first determine to which LSPs (mplsFTNActionPointer=SLS LSP IDs) each contract belongs to (DSCP-SLS IDs). In the case of WS though, the agent returns the SLS LSP IDs of each SLS but does not have to return the SLS IDs. This is because the latter are not required for the next query and because the WS manager does not need to perform any processing on these data. In the case of SNMP though the agent has to send both the SLS and SLS LSP IDs because the manager needs to process these in order to determine to which LSPs an SLS belongs to. The custom tool data queries for the SLS measurements of this scenario are:

$\{\text{mplsFTNActionPointer}[\ ]\}$ (MID query)

$\{\text{mplsFTNDscp} = \text{value}_1, \text{mplsFTNDscp} = \text{value}_2, \ldots\}$ (FD query)

{mplsInSeg mentPerfHc Octets[ ], (MID query)
mplsInSegm entPerfHcO ctets[ ],... }

{mplsInSegmentIndex = value$_1$ OR mplsInSegmentIndex = value$_2$..., (FD query
mplsInSegmentIndex = value$_3$ OR mplsInSegmentIndex = value$_4$...,... }

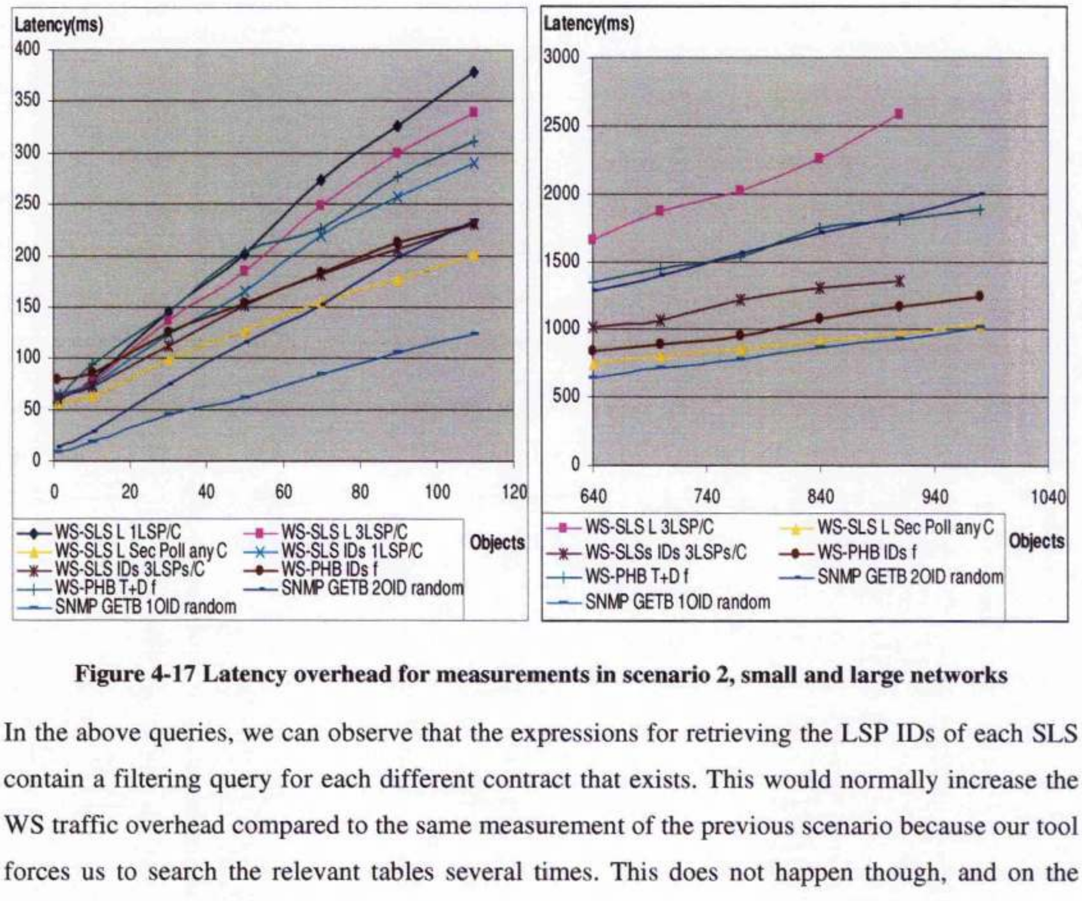


Figure 4-17 Latency overhead for measurements in scenario 2, small and large networks

In the above queries, we can observe that the expressions for retrieving the LSP IDs of each SLS contain a filtering query for each different contract that exists. This would normally increase the WS traffic overhead compared to the same measurement of the previous scenario because our tool forces us to search the relevant tables several times. This does not happen though, and on the contrary, the overall WS traffic is much less in comparison to the same measurement of the previous scenario. This occurs because the SLS IDs are not included in the response packet and thus the encoding latency of the SOAP message is reduced (very close to GetBulk traffic with no filtering involved - see SLS IDs 3LSPs/C or 1LSP/C on Figure 4-15). Latency is also less because less data need to be encoded in the SOAP body compared to the same measurement in the previous scenario (maximum 200 ms less than GetBulk – see Figure 4-10 & Figure 4-17 SLS IDs 3LSPs/C or 1LSP/C). Even when the amount of processing required is great as it happens for the VPN scenario (3LSPs/C) where a 900 entry traffic table has to be searched 300 times, WS performance is substantially better as less data need to be encoded in the SOAP body.

In terms of SLS load measurements using our custom framework on the other hand, traffic and latency (Figure 4-16 SLS-L 1LSP/C or 3LSP/C & Figure 4-10 SLS-L nof) have increased compared to the previous scenario (maximum 3 times more latency and maximum 20K more traffic). This occurs only once per granularity period (in minutes) when load (L) is polled for the first time. Polling for load for the first time requires the manager to determine which entries in the statistics table of the incoming segment refer to each LSP. After that, the SLS load can be retrieved by using the row identifiers obtained in the first measurement. After the first polling period the traffic overhead and latency for determining SLS load, is similar to the WS measurement for SLS load in the previous scenario (Figure 4-16 SLS L Sec Poll any C, Figure 4-12 SLS L nof, and Figure 4-17 SLS L nof).

Using our query tool as described above in order to perform the two steps required for SLS load and packet discards measurements still allows us to meet the granularity requirements for these measurements. Nevertheless, performance of WS SLS measurements could be improved if the query tool was enhanced to support more flexible and more expressive queries so as to search relevant tables fewer times (300 searches for SLS load in the first granularity period).

For the same reasons as for PHBs measurements, satisfying the granularity requirements for short-term SLS measurements with SNMP might not be possible especially when processing SLS load state data from many ingress routers and especially if the distributed management extensions of SNMP were used. For a series of 50 edge routers/devices which have 300 contracts each one assigned to 3 LSPs (VPN scenario) a 900 entry mplsFTNTable will have to be processed 300 times one for each contract (1369 ms to complete on a 1000 MHz/256MB PC). Thus it would take a SNMP manager 50x1369=68450msecs=68,450secs to find to which LSPs each contract is assigned to for every router, in order that afterwards the manager can find the load for each contract. Alternatively for the load of each traffic contract to be calculated at the agent on a per SLS basis, the SNMP manager would have to retrieve data in a random way from an LSP statistics table for each LSP an SLS is assigned to. This as for PHB load and discards increases the latency and traffic overhead of a GetBulk operation (see Figure 4-16-GETB-SLS L random, Figure 4-17- GETB-1OID random, GETB-2OIDs random and Figure 4-15 GETB-SLS+SLS LSP IDs rand). Alternatively using the distributed management extensions to compute the load of each traffic contract will not save but on the contrary will increase the traffic and latency overhead, for the same reasons as explained previously for PHB discards and throughput.

4.4.3.3.3 Scenario Three: Data Filtering

In the third QoS scenario we can demonstrate how a WS-based monitoring system can benefit from data filtering. For the measurements of this scenario, the ingress router is configured to have 910 and 50 LSPs to simulate a large and a small network respectively. Each LSP is assigned to a different customer (SLS). The reason behind assigning a different customer to each LSP is to keep things simple with respect to checking the validity of the data retrieved with each query performed with the custom query tool. A further assumption in this experiment is the number of

LSPs and SLSs affected by the failing interface, which we assume to be six. Although it is not easy to determine a plausible number of LSPs assigned to a single interface, six would be a reasonable number for small networks. It may not be realistic though to use the same number of affected LSPs in the case of large networks. The aim though is to keep the volume of data to be retrieved relatively low. This way we can show that WS can benefit from sophisticated retrieval mechanisms and exhibit superior performance against SNMP even though a small volume of data needs to be retrieved (not shown in [120] and [122] or any other WS performance related work). In addition, by keeping the same number of affected SLSs and LSPs for both small and large networks we can maintain the traffic overhead and latency comparison between both types of networks on the same terms. The queries required for this scenario were analysed in section 4.3.4.2 and were given in Figure 4-4.

The measurements for this scenario are presented in Figure 4-18 to Figure 4-21. The total amount of traffic produced as a result of the three queries for a big or a small network using WS is 6653 bytes. This happens because as already explained for this scenario the amount of information that is retrieved for either type of network is the same although the volume of information that needs to be searched changes. The total latency for WS in the small and the big network is 134 and 425 ms respectively.

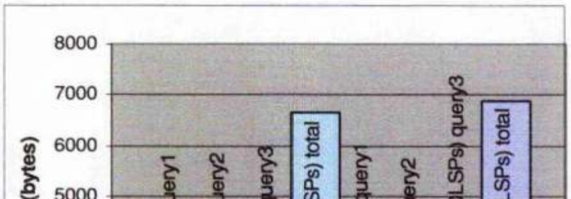


Figure 4-18 Traffic overhead measurements for WS & SNMP for a small network

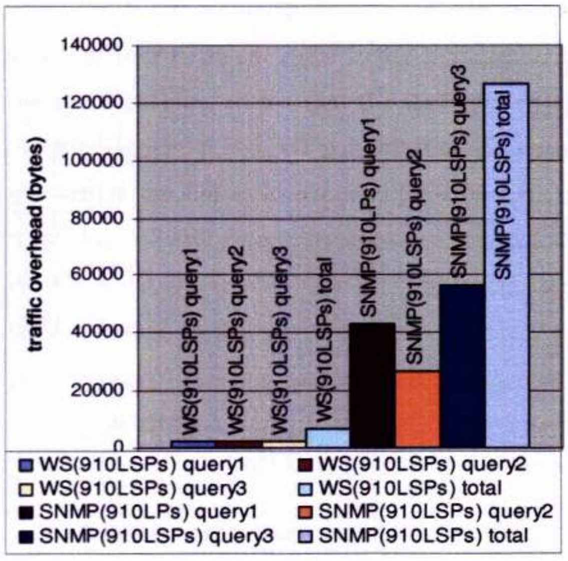


Figure 4-19 Traffic overhead measurements for WS & SNMP for a large network

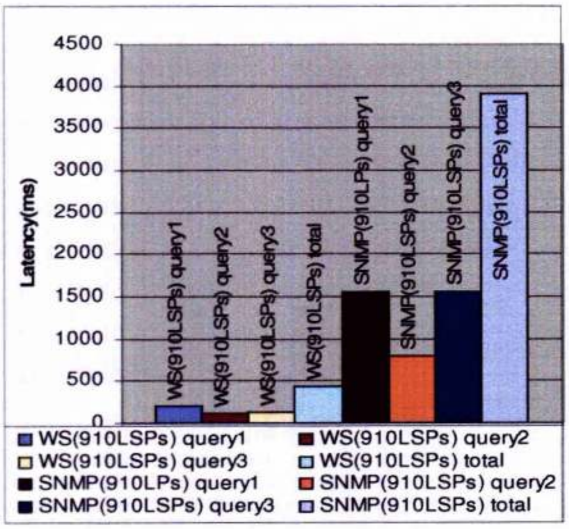


Figure 4-20 Latency measurements for WS & SNMP for a large network

For SNMP measurements, filtering is not supported and processing has to be performed at the manager. As such for this scenario all the interface indices along with the LSP IDs and the SLS IDs need to be retrieved in order for the SNMP manager to be able to process them. Furthermore as we can see from the SNMP measurements there are differences between the traffic and latency overhead between queries 1, 2 and 3. These differences are attributed to the amount of data retrieved for each query which affects both traffic and latency overhead. The total traffic for SNMP, using GetBulk, is 6890 bytes for the small network and 126360 bytes for the large network. The total latency for SNMP is 247ms for the small network and 3902 ms for the large network. If more LSPs and SLSs were affected by the failing interface it would mean that the volume of data that would be retrieved by WS would be bigger. It is not realistic though to have a big number of LSPs inside a router fail at the same time. The only case where SNMP might have shown better performance than WS for this scenario would be if very small networks were investigated so that the amount of information to be retrieved or processed would be very small. Such QoS-enabled networks though do not exist in practice. As such for this scenario WS exhibit better performance than SNMP.

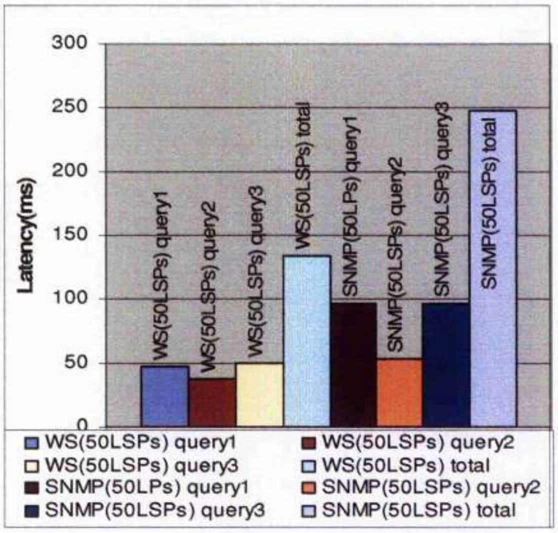


Figure 4-21 Latency measurements for WS & SNMP for a small network

4.5 Summary and Conclusions

In this chapter we have compared the performance of three WS-based tools for bulk and selective retrieval over a QoS scenario for polling based monitoring. Through this comparison we have shown as expected that our custom query tool is more scalable due to the several optimisations that have been performed on it (raw data are processed instead of XML, the hierarchy of data for multiple instance data is simplified, queries for inexistent data are rejected immediately etc). Still by comparing XPath 1.0 & 2.0 we have seen that increasing the expressiveness of XPath 2.0 implementations is a step in the right direction to decrease its latency and memory consumption. A lot more improvements though are necessary. As such we have suggested ways that could possibly increase the performance of XPath implementations. This is necessary because developing custom tools to meet management objectives for monitoring also has limitations (increased development time, limited functionality and expressiveness, etc). As such, a wider range of solutions for bulk and selective retrieval is always desirable. Nevertheless our custom query tool and parser implementation have shown that general tools such as XPath have to be improved before using them for time critical management tasks.

Having investigated the performance of our tool we decided to test the effective use of WS and our custom framework for performing network monitoring tasks compared to SNMP. In contrast to what other researchers have done in this field, we chose to investigate polling based monitoring scenarios where task and load distribution are required to perform a series of high level tasks requiring filtering and data processing. Before using our query tool to evaluate the performance of our WS monitoring framework and SNMP though, we tried to investigate the best possible ways to offer facilities for bulk and selective retrieval for distributing monitoring load in the context of SNMP. As such we came upon the distributed management extensions of SNMP. Our study has shown though that problems such as security, restricted access to resources, luck of industry support, increased development time and performance costs and limited open software support has led the management community to practically not use the distributed management extensions of SNMP.

Based on the above we have decided to use our query tool for WS based monitoring and SNMP without distributed management extensions to test their performance. In contrast to previous research though, we tried to investigate and compare the performance of our custom WS framework against that of SNMP using a realistic case study. The latter is that of a monitoring system of a network that provides QoS guarantees using MPLS. As part of this investigation we have (a) analysed the measurements required for QoS, (b) shown how to perform these measurements with the MPLS MIBs both for SNMP and WS, and, (c) chosen three scenarios to demonstrate how the monitoring process can benefit from bulk and selective information retrieval facilities.

In the first QoS scenario, we have investigated the performance of our WS monitoring framework and SNMP when retrieving data in a bulk manner. This is similar to what previous researchers have done, with the difference that we used literal encoding for the data in order to minimise the verboseness of XML tags. This way we can minimise the data encoding latency and also the traffic overhead. We demonstrated that WS can perform better in terms of traffic overhead and latency, in comparison to retrieving data with consecutive SNMP getNext operations. We also showed that when the required management information increases, the latency of WS can sometimes become even less compared to SNMP's GetBulk operation, mainly because SNMP agents do not support caching. In addition if, as we showed in chapter two, a newer SOAP toolkit was used, such as Axis 2.0 (version 1.4) compared to Axis v1.4 that we used for our study, the performance of our WS monitoring framework could be improved even further (see [180]). Therefore, as long as care is taken to meet management objectives and improved versions of XML tools are used to deploy a WS framework, WS can become better compared to the past in addressing the monitoring requirements of network management.

In the second QoS scenario we have used our custom query tool to distribute the monitoring load for processing management data to the agents. This way the manager can receive the required information it needs in a specific manner (e.g. per PHB, SLS).). This scenario is also a way to show how important it is to minimise the processing overhead of WS operations for management in addition to minimising the serialisation and parsing overhead of WS toolkits. Furthermore, with this scenario we can show that by distributing this load to several agents the manager is relieved from this task. Though SNMP could do the same with its distributed management extensions, we have explained that the use of the expression MIB for example would potentially increase the latency and traffic overhead of SNMP operations instead of reducing it. Even if the expression MIB is not used and we select the manager to do all the processing, more latency would be introduced since several agents must be monitored, for example in the order of hundreds for a large ISP network. We have shown that the latter may not be scalable. Even when retrieving data on a per PHB or SLS basis with SNMP by assuming that the manager knows the exact location of tabular information inside the various tables of the LSR and the FEC MIB, the footprint of SNMP operations would increase. This is because in these cases all the OIDs of the required objects that would have to be retrieved must be defined in an appropriate order in the request packet. Doing this makes even the latency and the traffic overhead of the GetBulk operation of SNMP worse than that of our WS framework in certain cases when the number of objects retrieved exceeds a certain number. Thus by having an optimized parser and framework and by allowing processing of management data at the agent we managed to meet any of the granularity requirements for monitoring required for this scenario. We have shown that the same may not be true in some situations for SNMP's short term measurements showing how important it is to minimise the processing overhead of management operations.

In the third scenario, the better performance of our custom query tool and framework in using it for monitoring operations that require filtering is demonstrated. The traffic overhead and latency in this scenario is in some cases 25 or 15 times less than the equivalent measurements with SNMP. This would not be the case if XPath was used (XPath implementations as shown in the comparison of our query tool with XPath 1.0 and 2.0 incur a minimum of ten times more latency for monitoring operations over a large network for the third QoS scenario).

Still, the measurements in the second scenario suggest that our parser has room for improvement. Our query tool should support more flexible queries so as to minimise the number of times it is required to search for example a series of tables for state data. The latter would allow us to improve the latency and traffic overhead that our query tool incurs. Additionally from comparing our query tool with XPath, we have also realised that our query tool must be converted in order to allow it to be more extensible (i.e. using a Java compiler such as JavaCC and not Java's regular expression matching engine). This way it would be possible to allow the addition of many functions and operators that could increase the expressiveness of the queries that are formed with it. As a result, it would be possible to perform a series of other statistical operations on data compared to what our tool currently allows; this could be useful.

The approach and measurements presented in this chapter suggest that our custom WS framework can be used for distributed management meeting the monitoring requirements of a complex environment. Distributing load to the agents is extremely important for a WS framework, resulting in a more distributed scalable system that can support sophisticated management operations. Of course, our agent, query tool and any WS framework that supports it would need to be installed in routers, interfacing with the services providing elemental management information. For this reason, the query tool was designed so that it can be used as an add-on service functioning over existing services and to be easily integrated with other standardised WS frameworks (i.e. MUWS). Security features need also to be incorporated in our custom framework. The latter can be achieved by integrating our query tool and architecture with the MUWS framework (see [180] for the work on integrating MUWS and our query tool and architecture). This is possible since the MUWS framework allows the use of resource specific query tools, and also since MUWS enables us to define and exploit the relationships between state data which our query tool utilises for more effective monitoring (bulk retrieval from several WS).

In summary, we anticipate that sophisticated WS management interfaces and lightweight tools will be supported in future managed devices, avoiding mistakes made with SNMP whose simplicity eventually became a boomerang, restricting solutions for sophisticated distributed management. Realistic scenarios in this chapter demonstrate that WS-based approaches performing load and task distribution can exhibit good performance, in addition to expressiveness in addressing management monitoring requirements.

CHAPTER 5

5 Efficient WS based Event Reporting

5.1 Introduction

One significant aspect of using WS for network management is event reporting. In order to use WS for event reporting two problems have to be addressed (a) asynchronous communication and (b) efficiency. The former is required since the time of the occurrence of an event is not predictable and thus synchronous communication is not possible. Efficiency is also an important aspect of event reporting. For example, it does not make sense to produce events that nobody wants to receive as this will incur unnecessary use of resources. To clarify this, an example is in order. In an event reporting system that supports brokering relationships it does not make sense for an event source to produce events that nobody has subscribed in the broker to receive them.

In order to provide asynchronous communication between WS, a callback mechanism is required. A Uniform Resource Locator (URL) is such a mechanism but is inadequate since (a) it only allows a single protocol to be defined to reach a service (b) it does not necessarily convey interface information [172] etc. The proprietary WS-Addressing [19] specification tried to solve the callback problem by defining two XML structures that can be used as an efficient callback mechanism: (a) endpoint references, (b) message-information headers. These mechanisms can be used to support various message exchange patterns that WSDL 1.1 did not support. Despite the drawbacks analysed in [172] and [173], this specification has opened the way for three event reporting specifications to be defined: WS-Events [174], WS-Eventing [20], and WS-Notification [91].

In the WS-Events specification by HP [174], the consumer of an event can primarily (a) discover the types of events an event producer supports, (b) discover the events a producer currently holds, (c) subscribe in order to receive events, (d) perform filtering on the type of events that a consumer wants to receive, (e) define an expiration date for receiving events, and, (f) provide a callback URL for delivering an event. Filtering mechanisms are not specified in [172] but the means for unwanted events not to be produced or consumed are provided. In essence the specification does not clarify what tools are going to be used for event filtering [172]. Still the authors of the specification have created the framework based on the operations of which it is possible to perform event filtering.

In the WS-Eventing [20] specification things become clearer; this specification supports the XML Path (XPath) language for event filtering and WS-Addressing to provide a better callback mechanism than the simple URI scheme used in WS-Events.

When the WS-Notification [91] standard was introduced, more features for event reporting were added compared to the previous standards. In addition to what other standards support, the WS-Notification standard also:

- Allows consumers to receive content in an application-specific manner (raw format) or in a standard manner for increased interoperability.
- Enables a consumer to define several types of expressions for event data filtering (XPath, database queries etc).
- Permits a consumer to define the types of events it needs to receive with expressions called topics.
- Provides support for notification brokering.

All the above standards are on the right track for providing efficient and reliable WS-based event reporting communication. Nevertheless, WS-Notifications could be used more efficiently for network management. Consider the third QoS scenario in chapter 4 where a manager has to be notified when an interface of a Quality of Service (QoS) enabled router fails. Upon receiving this event, the manager needs to determine the traffic contracts and LSPs affected by this interface and requests for more data. In cases such as these, event reporting triggers a set of actions at the event receiver. These actions may be requests to retrieve more system data (i.e. because the event itself was not descriptive enough to describe the root of a problem) or perform changes on the network (i.e. configuration of the network to solve a problem). Finding standardised ways to perform a set of actions and tasks, normally performed in network management by the entity receiving an event, in order for these actions to be performed by the entity producing them (i.e. a management agent hosting an event service), has the potential of making the notification process more efficient and more reliable. As explained in chapter 4, the process where an entity is given the task to perform a set of actions for another entity is called task delegation. Task delegation can be used for WS-event reporting as long as (a) the entity with the responsibility to perform a set of tasks does not live on a very resource-constrained system and (b) this is performed in a standardised manner. For the latter, the use of a WS standard to perform task delegation would be more than enough. For the former, this is more of a reality today [55] since the myth of the dumb agent does not apply, given that the capabilities of devices have been significantly enhanced.

Using task delegation in WS-based event reporting can be important for two reasons. The first one applies to data retrieval. In many event-reporting scenarios event data represent a small amount of the data carried over the network in comparison to the HTTP and the Simple Object Access Protocol (SOAP) header data. The use of WS-Notifications is not justified in these cases since WS can perform badly when retrieving relatively small amounts of data [120], [122]. As such, adding additional data normally retrieved after the receipt of an event in the initial event report in order to reduce latency and traffic overhead should be beneficial. Second, by task delegation a higher degree of autonomy and reliability can be achieved as a manager's supervision is limited and faults can be rectified as and where they occur.

There is a disadvantage though when performing task delegation. The disadvantage is that application complexity increases. As we are going to see later on, task delegation requires that a certain number of tasks are performed dynamically at run time. This can be difficult. For example performing processing on data returned after a task is sometimes quite difficult due to the variety of data that can be returned. Furthermore task delegation requires managing of tasks. This can be quite difficult especially for tasks that are dependent on one another.

In chapter two we suggested that when performing task delegation from one entity to another it is necessary not to hardwire the logic and the tasks that can be performed as happened in [124]. In addition, we also mentioned that a WS-based event reporting system should use its own conventions and mechanisms for event reporting but also should be able to interpolate with standards at the edges of a domain as suggested in [34]. It is thus evident that a WS-based event reporting system has to be characterised by the following traits (a) it must have a flexible, dynamic non hardwired logic for configuring the tasks that need to be delegated to it (i.e. an event service at a management agent) by a manager (consumer of events) (b) it should be able to cooperate or use WS standards messages for any part of the communication that it is required to have with a consumer of events for interoperability purposes, as well use its own application specific messages for increased performance.

A promising way to configure the event process for WS-based event reporting dynamically without hardwiring the logic of the event system is through policies. The WS-Notification standard supports the use of policies. The WS-Notification specification also supports the use of standard messages to report events as well as application specific messages. As such the WS-Notification standard becomes an excellent candidate for event reporting using task delegation.

Delegating tasks through policies to improve the communication between two entities is not a new idea. Applying it to WS-based event reporting to check if it is feasible and if potential benefits can be gained, is something that needs to be explored. As such, we have designed and built a WS-based event service supporting task delegation with the use of WS-Notification messages and policies. To prove the viability and the gains of this system, an event reporting scenario is analysed. This scenario is the third QoS scenario presented in chapter 4. Based on this scenario we analyse the performance of event reporting for three systems:

- A WS-based notification system where only event data are reported in the initial report. Then a set of actions triggered by the event are performed by the manager to collect more data.
- A policy WS-based event system where event data and data collected from subsequent tasks are gathered and sent by the entity that produces events in the initial event report.
- An SNMP trap system.

The remainder of this chapter is structured as follows. Section 5.2 analyses the WS-Notification standard messages that are going to be used to configure the event reporting reporting process with policies. Based on section 5.2 we will show in section 5.3 that the WS-Notification standard can be used for configuring the event reporting process with network management policies. Using network management policies we will show that it is possible to support the delegation of a series of tasks of varied complexity from an event consumer (manager) to an event producer (event service at an agent) using a policy-like language. As such, in section 5.4 we will present the policy-like language we have devised to configure an event service we have built for task delegation. In this section, we will also analyse the WS-Notification compliant messages that need to be exchanged for configuring our event service for handling a set of varied complexity tasks assigned to it by task delegation. As part of the task delegation process we will explore the interactions between our event service and an event consumer (a manager) using as an example the interactions required to handle the requirements of the third QoS scenario analysed in chapter 4. In section 5.5 we will present a perfomance evalution between the two WS-based systems and the system based on SNMP traps using this scenario. In section 5.6 we finally present our conclusions.

5.2 The WS-Notification standard messages for event reporting

The WS-Notification family of specifications defines a complete system architecture to support event reporting based on WS. In this architecture a *publisher/producer* is an entity that sends notifications about a range of events called *topics* to other entities called *consumers*. *Brokers* are also defined as intermediate entities between producers and consumers that control the flow of events based on filtering. For a consumer to receive events, the latter must register with the broker or the event producer by selecting the appropriate topics. Publishers of events must state to the broker which topics they support, or advertise on their own the topics they maintain.

The WS-Notification standard defines a variety of features and messages that are exchanged between the various entities participating in the event reporting procedure. In our investigation we are only interested in (a) the request message a consumer sends to a producer to register for an event topic, (b) the response to this request message and (c) the messages the producer sends to the consumer in order to report an event. We do not tackle other messages involving notification brokering, event topic filtering at the broker, etc. This is because all these are out of the scope and requirements of the third QoS scenario that will be examined to evaluate the performance of various event-reporting systems. As such, there is no need to analyse any of the other messages the WS-Notification standard supports.

Based on the above and for the needs of the third QoS scenario, we will first analyse the structure of the basic WS-Notification messages used for subscribing and receiving events. Based on this analysis we will show that the WS-Notification standard supports the use of policies. We will also show how the WS-Notification standard supports the use of standard and application specific messages to report events. Both of these features that the WS-Notification standard supports, form the two essential elements required to build an event reporting system supporting execution of a number of varying complexity tasks in a dynamical manner.

In the next two sections we analyse the basic messages of the WS-Notification standard (request response subscription messages and notification messages). This is the next incremental step in order to explain, in subsequent sections, how to increase the performance of WS-based event reporting systems with policies.

5.2.1 The WS-Notification Subscription Message

The WS-Notification specification specifies that in order for an event consumer to receive a notification from a producer, the former has to send a subscription message in order to subscribe for a series of events. The format of such message is given in Figure 5-1.

- The *consumer reference* element tag is a URL providing a callback mechanism for event delivery.
- The *topic expression* element tag defines the event topics a consumer can register to receive. A topic can have sub-topics and topic filtering can be used by an event consumer to define specific topics of interest. Our event service implementation which we will analyse in section 5.4 supports four general topics; (a) a threshold is exceeded going upwards (notify-high), (b) a threshold is exceeded going downwards (notify-low), (c) the state of a state attribute has changed to active (notify-up) and (d) the state of a state attribute has changed to non-active (notify-down). Thus topic filtering is not required.
- The *UseNotify* element tag is used by a consumer to select whether events will be formatted in an application specific manner or using a WS-Notification standard *Notify* message. In

addition to topic filtering, the selector and precondition expressions are used for data filtering. XPath expressions, database queries or any application specific tools can be used for this.

- The *InitialTerminationTime* tag is used define the period for which an event consumer registers for events.

```
<wsnt:Subscribe>
 <wsnt:ConsumerReference>
   http://131.227.88.70:8080/
   notifications /notifications_
   Consumer
 </wsnt: ConsumerReference>
 <wsnt:TopicExpression dialect=
  "http://131.227.88.70/eventTopics">
    tns:notify-down
 </wsnt:TopicExpression>
 <wsnt:UseNotify> True/False</wsnt:UseNotify>?
 <wsnt:Precondition>
  wsrp:QueryExpression
 </wsnt:Precondition>?
 <wsnt:Selector>
  wsrp:QueryExpression
 </wsnt:Selector>?
 <wsnt:SubscriptionPolicy>
  Event-Condition-Action
  Policy-like XML document
 </wsnt:SubscriptionPolicy>?
 <wsnt:InitialTerminationTime>
  2007-03-11T13:00:00
 </wsnt:InitialTerminationTime>?
</wsnt: Subscribe>
```

Figure 5-1 WS-Notification Subscription message [91]

In the subscription message, the *subscription policy* element is an open component that can be used to specify application-specific policy requirements/assertions. The semantics of how an event producer will react to these assertions depends on the application-specific grammar used. A non-normative way to define policies within the subscription policy element is with the WS-Policy standard [101]. The WS-Policy standard defines a base set of constructs that can be used and be extended by other WS standards to describe a broad range of service requirements and capabilities. Applied to WS, a policy defined using the WS-Policy standard can be used to convey conditions that need to be met when an interaction between two WS endpoints occurs. In reality though, the subscription policy element and the WS-Policy standard in the WS-Notification standard subscription message was envisioned by IBM so as to be used by subscribers for setting their requirements or specifying their directives to the services managing the underlying resources (i.e. for managing their subscription maintained by an event source/service). This is necessary because each WS may have different approaches for implementing subscriptions and notifications. The greater vision of IBM is to use the subscription policy element so as to be able to define concrete policies that allow a service (i.e. a WS-based event service) managing the underlying network resources, to describe its approach for subscriptions and subscription management, and also to give the opportunity to the subscriber to specify directives that it must follow [94].

5.2.2 Other WS-Notification messages

The response to a subscription message may contain a lot of information. Primarily it contains (a) the address of a WS that defines the messages that can be exchanged to manipulate subscription resources, (b) a resource id for the subscription and (c) fault information in case of subscription failure (Figure 5-2 left).

The WS-Notification standard *Notify* message (Figure 5-2 right) contains the following: (a) a *topic* header that describes the event topic an event consumer subscribed initially to receive (b) a *producer reference* element that describes the endpoint of the service that produced the event, and (c) *message* elements where the actual payload of a notification is inserted.

The event service that we will analyse in section 5.4 will support both the WS-Notification *Notify* messages but also the application specific messages for reporting events. As a result, our event service can be used at the edges of a network domain, to report events to entities belonging in other domains using the WS-Notification standard message format for interoperability purposes. At the same time, our event service can use application specific messages (i.e. less verbose messages) to report events to several entities within a domain that subscribed to these events. This is one of the requirements suggested as a way to increase the performance of WS-based event reporting in [34].

```
...
<wsnt:SubscribeResponse>
  <wsnt:SubscriptionReference>
     <wsa:Address>
        Address of Subscription Manager
     </wsa:Address>
     <wsa:ReferenceProperties>
        Subscription Identifier
     </wsa:ReferenceProperties>
     ...
  </wsnt:SubscriptionReference>
  ...
</wsnt:SubscribeResponse>
...
```

```
...
<wsnt:Notify>
  <wsnt:NotificationMessage>
     <wsnt:Topic dialect="xsd:anyURI">
       {any}
     </wsnt:Topic>
     <wsnt:ProducerReference>?
       wsa:EndpointReference
     </wsnt:ProducerReference>
     <wsnt:Message>xsd:any
     </wsnt:Message>
  <wsnt:NotificationMessage>+
</wsnt:Notify>
...
```

Figure 5-2 WS-Notification Subscription message response and Notification message [91]

5.3 Policy-like Configuration of Events for Network Management

So far we have analysed the structure of the standard messages of the WS-Notification standard. We have also seen the vision of IBM for using the policy element. Now we can explore the potential of the policy element in making the communication of a WS event reporting system more efficient and reliable.

Apart from the IBM specifics on policies, the vision of policies for network and service management is described in [175] and [176]. According to [175], policies are an aspect of information influencing the behaviour of certain components within a system. In [175] all policies can be expressed as if they belong to a hierarchy where a high level policy goal can be refined into multiple levels of lower level policies and eventually into a set of policy rules. Effectively, as defined in [175], policies are rules that can be used as the means to successfully achieve a goal. Based on what goals policies are trying to achieve they can be broadly classified into (a) *authorisation* policies that define what is permitted, or not, to be performed in a system, and (b) *obligation* policies that define what must be performed, or not, in order to guide the decision making process of a system. Both types of policies can be defined using an event-condition-action model of definition. Based on this model for defining policies, it is evident that policies can be reduced to a set of rules, actions, utility functions that can be used to (a) ensure compliance, (b) define behaviour, and (c) achieve adaptability of a system.

Based on the above, it is evident that the view of WS-Policy on policies and that of the NSM management community have quite a few similarities. The WS-Policy standard defines that any part of a policy can be considered as a domain specific assertion of policy information. This may be an assertion for an action, an assertion about the state of a system or an assertion for a goal to be achieved. As such, the event condition action model parts of a policy definition in the network management world can be expressed with the use of WS-Policy assertions. As a result of the latter, it is evident that network management policies can be defined using the WS-Policy standard. It is even possible to express network management policies, using any domain-specific grammar that manages to express the event, condition and action parts of a policy.

By having a view of what events represent in the network management world, we can observe that events have a lot of the characteristics that policies also embody. In the network management world, events are viewed as a method for notifying an entity (i.e. a manager) about the state of a managed device or underlying resource that usually demands an action to be taken. Thus, events can contain information about (a) the event itself, (b) the condition that produces the event, and (c) the type of actions to be performed after an event is generated. All this information is consistent with the network management perspective of defining policies (event-condition-action). Hence, it is clear that event information can be defined as policies. In general, it is even possible to describe the information of an event and the processes required in order to generate an event, or the actions performed when an event is generated, as part of a grammar used to describe the event condition action subparts of a policy.

To do what was described above with WS, we can either use WS-Policy or any domain specific grammar to configure events and the event process with policies. In this respect, the *subscription policy* element of the WS-Notification standard can be used as a wrapper to contain event information as well information in order to know when an event has to be generated and what to do when an event is generated.

Based on all the above, we have decided to design our own domain specific grammar to manage the event process and event information with policies. We have also built an event service that uses the basic WS-Notification standard messages to manage the event process with policies. In our event service implementation we use the *subscription policy* element as part of a subscription message to send to our event service (event producer) an XML document that consists of three sections: (a) general event data and how to collect them, (b) the conditions that trigger event-production, and (c) subsequent actions. As such, this XML document is used to configure the event process with policies. The grammar used to validate the XML document (against a schema) represents our domain specific grammar to manage the process of event production with policies. This grammar constitutes by no means a formal policy language. This is because our grammar based on its current condition needs to be enhanced in order to support concepts like policy refinement. Enhancement is necessary since our grammar currently supports actions as calls to a number of operations of a WS interface. These operations may be comprised by a number of actions which are not defined using assertions or alternatives clauses as in WS-Policy. For policy refinement this is necessary. Still the grammar supports an event condition action of defining policies. It is thus a policy-like language and not a policy language in the strict sense. Still we can use it within the *subscription policy element* so as to be able to configure our event service in order for the latter to become capable of performing a set of varying complexity actions or tasks. This allows us to delegate a set of tasks that the manager would otherwise perform to other entities (i.e. our event service hosted in an agent) so that the WS event reporting process is made more efficient.

In the next section we analyse our domain specific grammar. Completing this analysis we can then give an example on how to use the event service we have built to support the event reporting requirements of the third QoS scenario introduced in chapter 4. As part of this example we will explain the interactions between our event service (event producer) and an event consumer (manager) to configure the former to perform a set of tasks of varying complexity as part of the event process. We believe that this can make WS-based event reporting more efficient.

5.4 Managing our event service with our policy-like grammar

5.4.1 Policy-like event configuration document and grammar

The policy-like document consists of an event, a condition and an action part.

The event part (Figure 5-3) consists of sections which define (a) which parameter(s) need(s) to be monitored (*OIDstoMonitor*), (b) how to retrieve the state of the data to be monitored for event reporting (*EvenTask and its sub-elements*), and (c) how to handle and process the retrieved data for event reporting (*Result and its sub-elements*).

The *OIDtoMonitor* element is a comma separated list of the data that need to be monitored for event reporting.

The *EventTask* element is an element that describes the various information required in order to perform a WS call so as to be able to retrieve event data. These data can be processed in order to recognise if an event has to be produced or not. The WS call can be a call to a WS exposing management data from SNMP MIBs. It can also be a call to a system process offered through a WS interface (i.e. Command Line Interface (CLI) calls). Our view is that any action/task our event service is allowed to perform, can be provided through a WS interface. In this way, we can deploy and expose in a standardised fashion a set of capabilities that our event service can handle (these capabilities are provided by other WS). Based on the above, the *ServiceEndpoint* element points to the URI where a WS interface can be accessed. The *Method* element is used in order to describe the method of the WS interface that will be invoked i.e. to retrieve event data or to make any other call. The *Use* element describes the format of the SOAP body that will be used when communicating with a WS (i.e. Document or RPC). The *Style* element describes the encoding style of the messages dispatched in a WS call (i.e. Literal, Encoded, and Wrapped). The *MethodParam* and *Param* elements are used to describe the operands of the method of the WS interface that is invoked. Each *Param* element contains mainly three attributes, a parameter id *(pmid)*, a namespace attribute and a type attribute. The *pmid* attribute uniquely identifies a parameter in order to be able to reference it within the policy-like document. The *type* attribute describes the type of each operand used in a WS method call (i.e. string, integer etc). The *namespace* attribute is used to describe the schema where this parameter is defined. In general, the parameters in the *EventTask* elements allow us to make dynamic calls to WS offering a variety of capabilities. Dynamic WS calls are the only means through which we can configure our event reporting system without hardwiring the logic and the tasks it can perform.

The *Result* element is an element used to provide directives to our event reporting system on how to process the results returned after a dynamic WS call, i.e. to retrieve event data. In this way, the data from these calls can be processed and stored in memory for later use, i.e. to identify whether an event has been produced or not. The *ResultParam* element is used to describe data (single or multiple instance data) of different types that are returned as part of a response to a WS call. The *pmid* attribute in the *ResultParam* element is used to uniquely identify a collection of data of the same type. Using this attribute we can uniquely reference each type of information returned after a WS call so at to use it for example as a method operand for another WS call. The *type* attribute describes the data type of a collection of state information (i.e. string, integer etc). The *ResultFormat*, the *FormatValue* and the *FormatPattern* elements are used in order to extract with regular expression matching techniques, each type of information included in the response of a WS call. This is required because the data in the response of a WS call can be intermixed somehow (the result can contain multiple instance data of various types (i.e. LSP interface data and their row identifiers of the table where these data lie).

```
<ns:EventSpec name="" jobid="" date="" time="">
 <ns:OIDsToMonitor>...</ns:OIDsToMonitor> {1}
 <ns:EventTask actionid="">
   <ns:ServiceEndpoint>...
   </ns:ServiceEndpoint> {1}
   <ns:Method namespace="">...</ns:Method> {1}
   <ns:Use>...</ns:Use> {1)
   <ns:Style>...</ns:Style>{1}
   <ns:MethodParams>
     <ns:Param name="" pmid="" namespace="" type="">
   <ns:Param> +
   </ns:MethodParams> ?
      <ns:Result resid="" type="" namespace="" qname="" name="">
          <ns:ResultParam pmid="" type="">...
          </ns:ResultParam>*
          <ns:ResultFormat forid="" dependsON="">
            <ns:FormatValue>...</ns:FormatValue>?
            <ns:FormatPattern>...</ns:FormatPattern> ?
         </ns:ResultFormat> ?
     </ns:Result> *
 </ns:EventTask> {1}
</ns:EventSpec> +
```

Figure 5-3 Event part of the policy-like document

The condition part of the document (Figure 5-4) contains information to determine whether an event has been produced or not. Information that need to be defined in the condition part refer to (a) the type of monitor used (*MonitoringObjectType* element and *monid* attribute – i.e. mean monitor to compute the average between two counter values, variance monitor to compute the variance of a number of counter values, etc.), (b) the measurement granularity (*granularity* element), (c) the smoothing window size (*window* element), (d) the clearing value that re-enables event reporting if it has been disabled (*clearvalue* element) and (e) the value that determines if a threshold has been exceeded or not. The latter signifies if we need to report an event (*value* element).

```
<ns:EventCondition jobrefid="">
 <ns:MonitoringObjectType    monid="">
  <ns:granularity>...</ns:granularity> {1}
  <ns:window>...</ns:window>{1}
 </MonitoringObjectType> {1}
 <ns:Threshold>
  <ns:tType>...</ns:tType> {1}
  <ns:value>...</ns:value> {1}
  <ns:clearvalue> </ns:clearvalue> ?
 </ns:Threshold> {1}
</ns:EventCondition> +
```

Figure 5-4 Condition part of the policy-like document

```
<ns:ActionOnEvent jobrefid=""actionid="">
  <ns:ServiceEndpoint>...
  </ns:ServiceEndpoint> {1}
  <ns:Method  namespace="">...
  </ns:Method> {1}
  <ns:Use>...</ns:Use> {1)
  <ns:Style>...</ns:Style>{1}
  <ns:MethodParams>
    <ns:Param name="" pmid=""namespace="" type="">
    <ns:Param> +
  </ns:MethodParams> ?
  <ns:Result resid="" type="" namespace="" qname=""  name="">
       <ns:ResultParam pmid=""type="">...
       </ns:ResultParam>*
       <ns:ResultFormat forid="" dependsON="">
           <ns:FormatValue>...</ns:FormatValue>?
           <ns:FormatPattern>...</ns:FormatPattern> ?
       </ns:ResultFormat> ?
  </ns:Result> *
</ns:ActionOnEvent>
```

Figure 5-5 Action part of the policy-like document

The action part(s) of the policy-like document contains data similar to the *EventTask* and *Result* elements in the event part of the policy-like document. These data, as in the *EventTask* element, allow us to call the appropriate WS to perform a series of tasks of various complexities in the case that an event has been produced. The *Result* element in the action part of the policy-like document allows us to process the results of a response to a WS call (Figure 5-5). In this way we can use these results as operands of another task performed as part of another WS call. There are attributes inside the *Param* and *ResultParam* elements (not shown in Figure 5-5) that allow us to form the operands of a WS method call dynamically. This is extremely useful when using our parser queries or any custom tool queries. The former are used as operands of our scheme to retrieve management data in a bulk or selective retrieval manner. Sometimes these queries need to be formed on the fly, since they may also contain data that are not known in advance and are collected as part of executing a set of tasks sequentially. The attributes inside the *Param* and

ResultParam elements allow us to form these queries or any other operands on the fly (recursively).

5.4.2 Event reporting process description

Having explained how each piece of information in the policy-like document can be used, we can now analyse the interactions that take place between our event service (event producer) and an event consumer (manager) for event subscription. As part of the subscription process, we will show how to configure a set of tasks of varying complexity through our policy-like grammar so as to make our WS-based event reporting system more efficient. We will analyse the subscription process based on the steps required to handle the third QoS scenario in chapter 4.

To configure the event service we have developed for the QoS event reporting scenario presented in chapter 4, the event consumer has to send a subscription message to the event producer. In reference to Figure 5-6, the consumer is a network manager and the producer (event service) lies in a network agent.

In Figure 5-6 we give an overview of the operations that need to be performed as part of the subscription process for a receiver of events to actually start receiving notifications. Here, the subscription process starts by validating the policy-like document to avoid subscription request failure (Figure 5-6 step 0). Then, the request is compressed (Figure 5-6 step 0) and is sent to the agent (Figure 5-6 step 1). At the agent the subscription request is decompressed (Figure 5-6 step 2), the policy-like document is extracted and split into its event-condition-action sub-parts (Figure 5-6 step 3). After a DOM parser validates each message part (Figure 5-6 step 4), the XML policy-like document is also searched for any discrepancies not captured by XML validation (Figure 5-6 step 5). This is necessary because the policy-like document contains inter-dependencies between some of its attributes and elements. These interdependencies cannot be expressed in an XML schema. Therefore, any problems relating to these inter-dependencies have to be found by checking for them explicitly. If any errors are found, the manager's SOAP messaging service is notified (Figure 5-6 step 8). In the opposite case, the agent's messaging service tries to add an event job to the event service (Figure 5-6 step 6). An event job can still be rejected for various reasons (i.e. a job already exists, etc.) (Figure 5-6 step 7). A successful or unsuccessful addition of a job is reported to the manager (Figure 5-6 step 8). Apart from adding an event subscription job, the event service supports features for job subscription such as (a) resume, (b) suspend, (c) remove, and (d) update.

Upon successful addition of a job, the event sub-part is processed and event data are collected using the Java reflection API to dynamically invoke the appropriate WS exposing management data (Figure 5-6 step 9). Selective data retrieval is performed using the custom query tool presented in chapter 3. Because we define these queries as part of the operands of each WS call within the *Param* elements of the policy-like document, the *selector* and *precondition* expressions offered by the WS-Notification standard subscription message for filtering are not used. Following the phase of collecting data (Figure 5-6 step 10), the condition part of the policy-like document is processed in order to determine whether an event has been produced (Figure 5-6 step 11). If no event is produced, the process is repeated according to the granularity of operations. If an event is produced, the action sub-parts of the policy-like document are executed (Figure 5-6 step 12). The actions in our event reporting scenario involve tasks to gather extra data to determine the LSPs and SLSs affected by a failing interface. Calling the appropriate WS to gather these data is performed dynamically and any queries to retrieve management data are formed on the fly using recursive methods. This happens because these queries contain data not known in advance and are collected during the execution of each task. When the event data and data from the configured tasks are collected (Figure 5-6 step 13), an event report is sent to the manager (Figure 5-6 step 14), which confirms its receipt (Figure 5-6 step 15). The event report data at the manager are finally stored in HTML format (Figure 5-6 step 16).

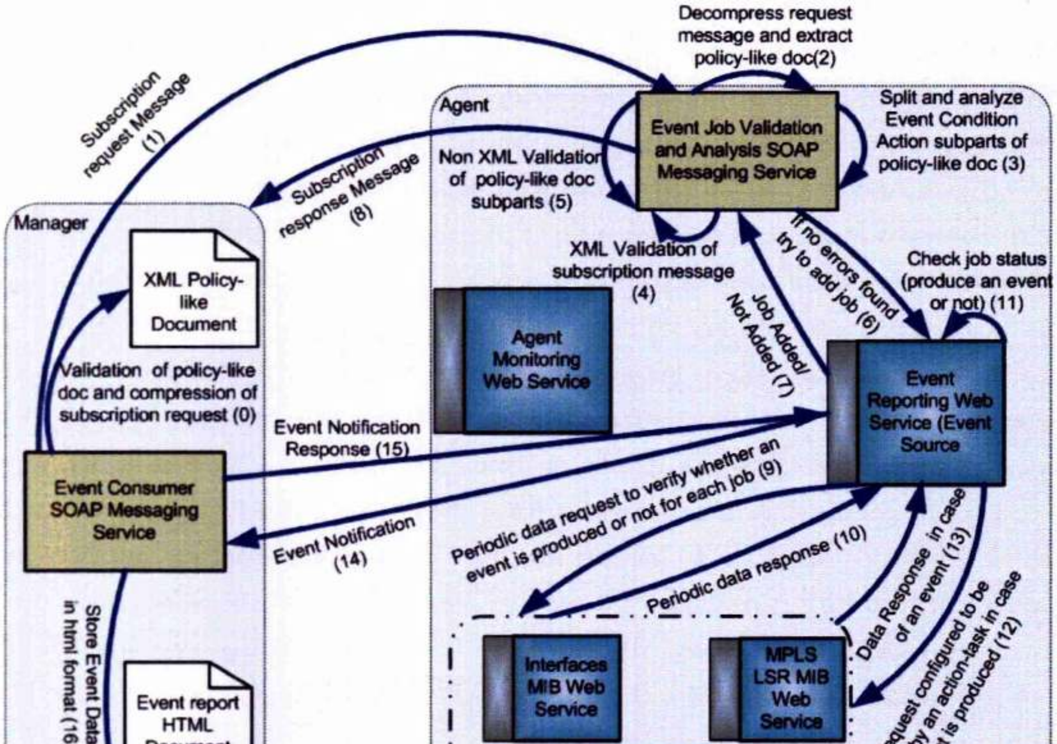

Decompress request message and extract policy-like doc(2)
Subscription request Message (1)
Agent
Event Job Validation and Analysis SOAP Messaging Service
Split and analyze Event Condition Action subparts of policy-like doc (3)
Non XML Validation of policy-like doc subparts (5)
Subscription response Message (8)
Manager
XML Policy-like Document
XML Validation of subscription message (4)
If no errors found try to add job (6)
Job Added/ Not Added (7)
Check job status (produce an event or not) (11)
Agent Monitoring Web Service
Validation of policy-like doc and compression of subscription request (0)
Event Reporting Web Service (Event Source)
Event Notification Response (15)
Event Consumer SOAP Messaging Service
Periodic data request to verify whether an event is produced or not for each job (9)
Event Notification (14)
Periodic data response (10)
Data Response in case of an event (13)
Data request configured to be performed by an action-task in case an event is produced (12)
Store Event Data in html format (16)
Interfaces MIB Web Service
MPLS LSR MIB Web Service
Event report HTML Document
MPLS FEC to NHLFE Web Service
Monitoring and Event Reporting Rep
Management Information Retrieval Services

Figure 5-6 Components interaction for event reporting

5.5 Event Reporting Scenario Measurements

So far we have analysed how to use our event reporting system to configure through policies a set of varying complexity tasks so that they can be performed by our event service (hosted an agent). Since the tasks delegated to our event service are performed dynamically this allows us to configure our event service to perform a variety of tasks in a flexible manner. Nevertheless, our event service is capable of performing a variety of different tasks as long as we expose a set of different capabilities through WS interfaces. As such, we have given an example of the tasks that our event service can perform. This example revolves around the requirements of the third QoS scenario presented in chapter 4. As part of the requirements of this scenario we explained how our event service can be used to handle a set of tasks the manager would normally perform when the latter is notified that the interface of the ingress router failed. In this way, our event service reports both the event data that it normally needs to report and also the data normally collected from the manager.

In this section we will try to explore if task delegation can make WS-based event reporting more efficient. As part of the task delegation process required for the third QoS scenario we analyse in the next sections (a) the setup for the measurements of the third QoS scenario (b) we elaborate on the event reporting systems we will be comparing and (c) we examine the measurements themselves in order to extract any conclusions.

5.5.1 Event reporting systems tested

The event reporting systems we will be evaluating the performance of WS-based management are three. The first of the three systems is our WS-based event reporting system supporting task delegation through policies. The second system is also WS-based and it is also using the WS-Notification standard messages. It does not support though task delegation through policies. The third system is based on SNMP traps.

For the requirements of the the third QoS scenario, the second WS-based event reporting system notifies a manager that the interface of the ingress router failed. The event service of this system is hosted at the agent side. When the manager is notified about this event, it uses our query tool or XPath 1.0 & 2.0 to enquire the agent in order to determine the affected LSPs and traffic contracts (SLSs). This system is implemented again using the WS-Notification standard this time though policies are not used to manage the event reporting process. As such the manager initially subscribes for an event type (this time for an event of type *"the state of a state attribute has changed to non active"*). When such an event takes place the event system notifies the manager. The manager uses the information in the event to understand the nature of the event and then uses a query tool to inquire the agent in order to find out which LSPs or SLSs are affected.

In the first WS-based system the event service is configured by a manager to perform dynamically the set of tasks the latter would otherwise perform after delivery of an event. When these tasks are performed by the event service, the latter sends back all the collected data to the manager (event data + data collected from the tasks). The approach followed by our policy based event reporting service is more complex. It requires from the event service to call the appropriate WS to determine the affected LSPs and SLSs at run time (WS dynamic call using Java's reflection API). Also for this approach, the queries performed with the query tool to determine the affected SLSs and LSPs need to be formed on the fly. This is necessary since some of the queries contain data that are not known in advance. These data are collected and processed by the event service while it performs each task successively. Based on policy-like configuration and data filtering and processing using our custom query tool, we show that this approach is plausible and results in traffic and latency benefits compared to the second WS-based event service.

The SNMP trap event reporting system handles the QoS scenario as the second WS event service. It reports that the interface of the ingress router failed and then an SNMP manager polls for data to determine the LSPs and SLSs affected. The manager of the SNMP trap system does not use any filtering mechanisms to process data, as for example the distributed management extensions of SNMP, for the reasons explained in section 4.4.3.3.2.

5.5.2 Evaluation Setup

For the evaluation aspects of our scenario we need to set up a QoS network with 30 LSPs and a network with 900 LSPs. The former network simulates a small QoS network and the latter a large one. For the reasons explained in section 4.5.3.1 we had to resort to other means to calculate traffic overhead and latency for SNMP the same way as explained in section 4.5.3.1. The new traffic analysis is presented in the next section. For traffic overhead measurements of SNMP average traffic was calculated between a maximum and a minimum value. The software used for SNMP, the MIBs used and the utilities for measurement are the same as in section 4.4.2.1. For each SNMP latency measurement we used Java's currentTimeMillis() function to average 10 measurements for each sampled result.

For WS the Apache Axis 1.4 SOAP toolkit was used to deploy the LSR, the FEC and the Interfaces Group of the RFC 1213 MIB as WS, with the same information as in SNMP. The information of each MIB is replicated exactly as it would be in a router. All MIBs were deployed using a Document/literal encoding style so that the verboseness of XML tags is reduced and traffic overhead as well as coding latency is minimised. The Document style is also recommended for use with our event reporting system since it makes the process of handling the data returned after a WS call easier (i.e. XML tags in this style provide context information and it is fairly easy with this style to handle a volume of information not known in advance instead of having raw data). The SOAP messaging services used in the two WS-based event reporting systems use JAXP 1.3 to parse XML documents, SAAJ 1.3 to exchange SOAP messages [117] and JAXM 1.1 for XML messaging. Java's zip facilities are used to compress/decompress the WS-Notification standard messages. Java's reflection API was used to make dynamic calls on WS. For bulk and selective retrieval in the WS-based event reporting systems, we used the query tool presented in chapter 3, based on Java's 1.5.6 regex engine. For the WS-based event reporting system without policies we used SAXON 8.9 for XPath 2.0 functionality and JAXP 1.3 for XPath 1.0 functionality. For WS-based systems the Linux *tcpdump* utility was used to measure the traffic overhead. Latency measurements for WS were performed using Java's currentTimeMillis() function by averaging 10 measurements for each sampled result.

The manager and agent used in the event reporting system of Figure 5-6 and in general for all the WS-based systems are deployed on a 1000MHz/256MB RAM and 466MHz/192MB RAM machine respectively. Both PCs run Red-Hat Linux 7.3, thus simulating a lower end system for the agent.

5.5.3 Measurements

The measurements presented in this section demonstrate the potential benefits of data filtering, processing and task delegation for WS-based event reporting. A comparison with SNMP traps is also performed.

For SNMP traffic overhead measurements we rely on previous research performed in [122] and [124] about polling based monitoring and event reporting. In these papers the traffic overhead for SNMP operations is given by:

$$L_{get,\,getNext} \approx n1 * (54 + 12 + 2L_1 + L_2) \tag{5.1}$$

$$L_{getBulk} \approx 54 + 1 * (6 + L_1) + n_1(6 + L_1 + L_2) \tag{5.2}$$

$$L_{trapSNMPv1} = 49 + n_1 * (3 + L_1 + L_2) \tag{5.3}$$

$$L_{trapSNMPv2} = 75 + L_3 + n_1 * (3 + L_1 + L_2) \tag{5.4}$$

In these equations L_1 is the size of the Object Identifier (OID) of a variable, L_2 is the variable value size, n_1 is the number of OIDs to retrieve and L_3 is the trap OID. Taking into account the size (in Table 5-1) of the data that needs to be collected for polling based operations and SNMP traps, the traffic overhead for SNMP can be computed.

Monitoring

Measurement Type	mplsXCLspId. mplsXCIndex. mplsXCInSegmentIndex. mplsXCOutSegmentIndex	mplsInSegmentInterface. mplsInSegmentIndex / mplsOutSegmentInterface. mplsOutSegmentIndex	mplsFTNDscp. mplsFTNIndex	mplsFTNActionPointer. mplsFTNIndex
L1/L2	16-19 (Max 16000 LSPs) / 6 (CR-LDP)	14-16 (Max 16000 Ifs) / 1-4	14-16/ 1-3 (Max 16000 LSPs	14-16/ (16-20) (Max 16000 LSPs

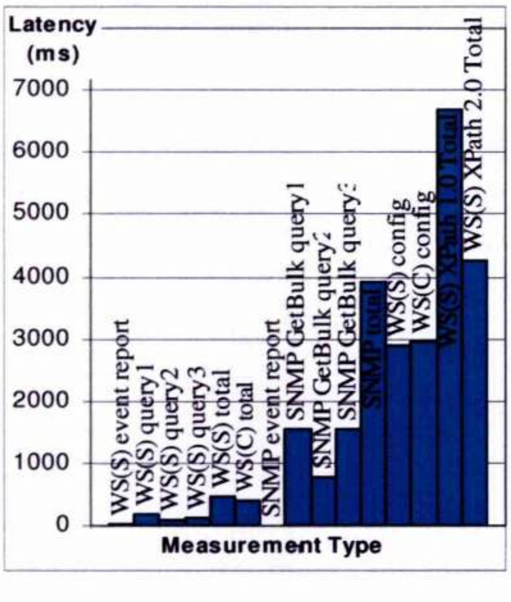
Event Reporting

Measurement Type	ifOperStatus. ifIndex	ifAdminStatus. ifIndex	Trap OID L3=10
L1/L2	10+(1-3)/1	10+(1-3)/1	

Table 5-1 Information size in ASN.1 format inside an SNMP message

The network measurement setup for these measurements is the same as in 4.3.1 and 4.5.3.3.3 for the reasons explained in these sections. The queries required for this scenario to determine the affected LSPs and SLS were analysed in section 4.3.4.2 and were given in Figure 4-4.

The measurements for the three event reporting systems are presented in Figure 5-7 to Figure 5-10. In these figures the measurements for the policy WS-based system are depicted with (C) and for the other WS-based system with (S).



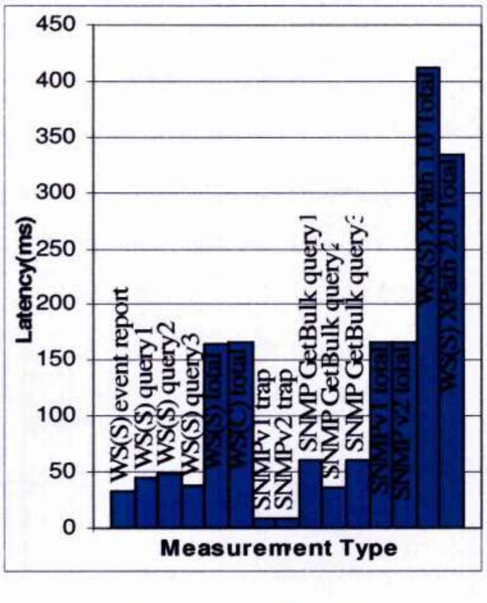


Figure 5-7 Latency measurements for SNMP and for the two WS-based approaches (900 LSPs)

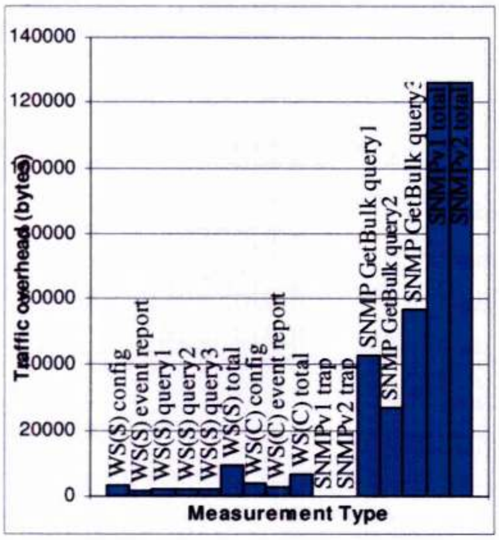
Figure 5-8 Latency measurements for SNMP and for the two WS-based approaches (30 LSPs)

In Figure 5-7 we can observe that the latency for configuring the WS-based event reporting services is quite significant both for the system without policies but also for the one with policies. This happens because both systems require (de)compression of the subscription request. Also XML validation of the subscription message is performed for both of the WS-based systems (Figure 5-7 WS(C)/ WS(S) config). The latter also increases latency. Configuring the event service though is not a time critical task and it happens once for a specific event-subscription-job. Therefore, we do not consider subscription latency in the event reporting overall latency of the two WS-based systems, since it is not a time critical task and since subscription operations can be considered as offline operations.

Comparing the two WS-based approaches in terms of latency for small networks, it can be seen that the latency difference between the two systems is very small (Figure 5-8). This at first seems strange. Normally we would expect that the event reporting system with task delegation would incur less latency. We expect that because the policies-WS-based system is performing local-inside-the-agent WS calls to determine the affected SLSs and LSPs. Hence, for these operations the network latency overhead does not contribute to the overall latency. Nevertheless, the policy event reporting system suffers from latency incurred from performing dynamic WS calls and building data queries on the fly. As such, any gains from performing local WS calls are counter-balanced from the fact that these calls have to be made dynamically. For big networks though, latency is less by around 75 ms for the policy-like based event reporting system (Figure 5-7).

Comparing the two WS-based event reporting systems with SNMP traps, latency is about the same in the case of small networks (Figure 5-8). This is quite a good result for WS-based systems, considering that the amount of information retrieved is small (WS-based systems tend to perform worse when compared to SNMP for retrieving small amount of information due to the processing overhead of the HTTP and SOAP header data). For big networks though, SNMP incurs more latency (Figure 5-7). This occurs for two reasons. The first reason is that SNMP does not offer facilities for task delegation through policies so that the data retrieval operations for determining the affected LSPs and SLSs could be performed locally. The second reason is that SNMP does not offer filtering capabilities. Therefore, determining the LSPs and SLSs affected from the failing interface, requires retrieving more data than required from the relevant tables in the MPLS MIBs. This is required so that all these data can be processed by the manager.

The latency performance of the two WS-based systems would not be better than SNMP if XPath 1.0 or 2.0 were used. As we saw in the previous chapter, XPath incurs at least 10 times more latency overhead compared to our custom query tool to offer bulk and selective retrieval capabilities to the agent. As such, if XPath was used, the latency overhead for these operations and in general for the WS-based event reporting system without policies would be considerably more than that of the SNMP trap system (Figure 5-7, Figure 5-8 WS(S) XPath 1.0 Total, WS(S) XPath 2.0 Total).



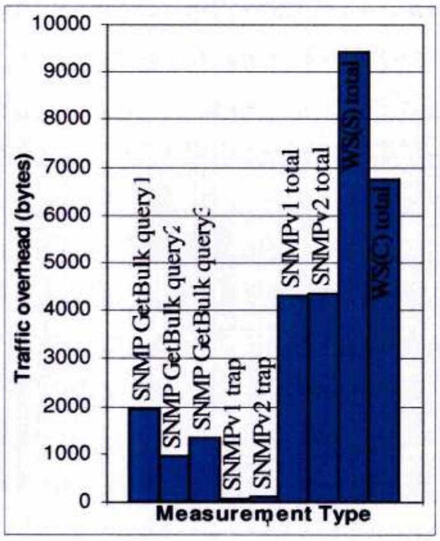


Figure 5-9 Traffic measurements for SNMP and the two WS-based approaches (900 LSPs)

Figure 5-10 Traffic measurements for SNMP (30 LSPs) and total traffic for the two WS schemes

As far as traffic overhead is concerned, 2700 bytes are saved by task delegation for both small and large networks when comparing the two WS-based event reporting systems (Figure 5-9). This reduction occurs because in the policy-like based event reporting system, the SOAP and HTTP header data contribute to the total overhead only once. The latter happens when sending the notification data along with all the data collected from each task (event data + task data). This may be a significant difference, because for every time an event will need to be reported, the policy based event reporting system will save more traffic and latency.

Therefore, it is desirable when configuring our policy based event reporting system to be able to monitor for example all the interfaces of the ingress router or any other managed device. This makes it more probable for an event to be produced and as such save more traffic. As such, we structured our policy-like document grammar in order to support processing data for event reporting from many underlying resources (i.e. interfaces, LSPs, PHBs). We also make use of our query tool to retrieve any data from any device. These queries are compact and thus, the overhead incurred for retrieving information from many underlying resources is small. To do the same for an SNMP trap system, we would need to define all the OIDs of the state data that represent the underlying resources for event reporting. This will increase the traffic overhead of SNMP operations.

Comparing SNMP's traffic overhead with our WS-based event reporting systems, we can observe that for big networks SNMP incurs a bigger overhead by 120 kilobytes. This is primarily attributed to the lack of filtering capabilities but also due to lack of task delegation facilities (Figure 5-9). For smaller networks though, SNMP's traffic overhead is less by 2300 bytes when compared to the WS approach based on task delegation and policies (Figure 5-10). If more events are produced though, configuration traffic overhead included in the total traffic overhead of any WS-based approach (3767 bytes for configuration overhead of the policy based system) will not be contributing again since this happens only once for each event job. As such, SNMP's traffic overhead increases and as a result our WS event reporting system with task delegation consumes 1467 bytes less traffic than SNMP (3767-2300) for each new event produced (Figure 5-10). This would not be the case for the WS-based event reporting system without task delegation (3100 for configuration overhead - 2700 more overhead than the policy WS-based system-2300 bytes less overhead from SNMP = -1900 bytes, thus 1900 bytes more traffic overhead than SNMP for each new event produced).

5.6 Conclusions

In this chapter we have investigated how to make WS-based event reporting more efficient. As part of this effort we have observed that in many monitoring scenarios, event reporting usually triggers a manager to perform a set of actions. These actions may be requests to retrieve more system data. This happens because in many cases such as in SNMP, events do not contain enough information for the manager to understand the true nature and cause of a fault or a problem. Also, in many cases event reporting triggers a manager to perform changes on the network in order to fix faults (i.e. configuration of the network). Finding ways to perform a set of actions and tasks, normally performed by the network entity receiving an event, in order for these actions to be performed by the network entity producing the event in the first place, can make the notification process more efficient. The latter process is called task delegation.

In chapter 2 we suggested that when performing task delegation of a number of actions from one entity to another, a necessary requirement is not to hardwire the logic and the tasks that can be performed as happened in [124]. Another requirement mentioned in chapter 2, is that a WS-based event reporting system should be able to use its own conventions and mechanisms for event reporting but also should be able to cooperate with standards at the edges of a domain, as suggested in [34].

In our search to achieve the above we came across the notion of policies. Policies effectively are sets of rules aiming to define under what guidelines a management system will operate. Policies essentially involve techniques to make management systems more autonomous or to guide the decision making process of a management system. As such, policies can also be used for task delegation where a decision is taken to delegate a number of tasks from one entity to another. As a result, policies would also make an excellent selection for making WS-based event reporting systems more efficient and more autonomous.

Based on the fact that policies can be used for task delegation, it so happens that the WS-Notification standard supports the use of policies. Therefore, the WS-Notification standard becomes an excellent candidate for supporting the first requirement a WS-based event reporting system must satisfy. In addition, the WS-Notification standard supports both standard messages and also application specific messages to report events. As such, the WS-Notification standard becomes an excellent candidate for supporting the second requirement that WS event reporting systems must satisfy.

To prove the viability and also to explore if anything can be gained from performing task delegation through policies we decided to explore one of the various event reporting scenarios that would benefit from it. This scenario is the third QoS scenario analysed in chapter 4. For the needs of this scenario we have designed our own application specific grammar to manage the event process with policies. This grammar also reflects the network management view for defining policies (event-condition-action). We have also built an event service that uses the basic WS-Notification standard messages to manage the event process with policies. In our event service implementation, we use the *subscription policy* element of this standard to send to our event service (event producer) a policy-like XML document that consists of three sections containing information on: (a) how to collect and process general event data, (b) the conditions that trigger event-production and (c) subsequent actions after event production. Using this policy document we can delegate a set of tasks normally performed by the manager of a system to the agent of a system hosting our event service.

To assess the performance of our event reporting system we analysed its performance with two other event reporting systems: (a) A WS-based notification system where only event data are reported and then a set of actions triggered by the event are performed by the manager to collect more data (b) An SNMP trap system. The agents of the WS-based event reporting systems support our query tool. The WS-based event reporting system without policies also supports XPath 1.0 and 2.0. As such the WS-based systems are equipped with facilities for bulk and selective retrieval. The SNMP trap system is not.

From measurements performed with these systems we have shown that facilities for task delegation and scalable mechanisms for bulk and selective retrieval can lead to gains for WS-based event reporting systems. We have also shown that such facilities result in performance gains for WS in certain scenarios against SNMP in terms of (a) latency overhead, (b) traffic overhead and (c) the variety of tasks an event service and as a result an agent is capable of performing. Offering such facilities is more than plausible today since the technical capabilities of devices used for management are not as limited as in the past. Still the complexity of building such systems and providing such facilities increases.

Our work on event reporting needs also to be improved. We have to refine our policy-like grammar to meet closely the requirements of policy management. Currently, our event reporting system is manually configured through policies to perform a set of tasks dynamically at run-time. The essence of policy based management for event reporting though would be to design an event reporting system that will autonomously deduce the actions to perform. This is in our future plans and it can happen through a process called policy refinement. Furthermore, it is in our goals to apply our event reporting system to other fields that need and would benefit from a more autonomous event service but at the same time would have more constrained resource requirements (i.e. policy based event reporting in ad-hoc networks, detecting node misbehaviour in ad-hoc networks with policies, configuration of a managed device in case of network failure where a network manager can not communicate with the device to overcome this failure).

Nevertheless, our event reporting system even as is, has nice application potential. Through a realistic scenario we have demonstrated that it can be a scalable and viable option in certain situations and can have the potential of equipping event reporting systems with the ability to perform a variety of management tasks (configuration, data collection for monitoring), without impairing much the operation of such systems. Distributing task load through policies can become an important aspect towards the direction of achieving more distributed, scalable and self adaptive event reporting systems.

Chapter 6

6 A Graphical User Interface for efficient Monitoring and Event Reporting

6.1 Introduction and Motivation

The main concern of our work so far was to minimise the footprint of WS-based management applications in order to handle management tasks such as monitoring and event reporting in a scalable manner.

As such we have presented our work on polling based monitoring. As part of this work we have built a WS-based monitoring system that relies on a custom framework and a data query tool, in order to support scalable monitoring operations. This system is part of a WS architecture we have introduced to support distributed polling based monitoring. We have shown that the architecture and the monitoring system we have introduced are scalable. This is attributed to a custom query tool we have built for efficient and more scalable monitoring compared to other query technologies such as XPath. Evaluating the performance of our WS-based monitoring framework (with our query tool) compared to SNMP, we have shown that the former can be used in a scalable manner compared to other technologies.

In the previous chapter we have also introduced our work on a scalable WS-based event reporting system. The latter supports mechanisms such as task delegation through policies and bulk and selective retrieval through our query tool to minimise the footprint of WS event reporting applications. Our event reporting system also supports the WS-Notification framework messages and operations for reporting events as well as application specific messages. We have shown that our event reporting system is scalable and efficient compared to another WS-based system that does not support policies and task delegation and uses XPath instead of our custom query tool, and also against a system that uses SNMP traps.

Though it is true that the ultimate goal of this project was to develop mechanisms in order to build an efficient WS-based framework for monitoring and event reporting our work is not complete. Mechanisms are required in order to provide a manager with a high level view of the functionality of the above systems. This will allow a manager (a) to have a view of the conceptual relationships between state data that are shared between WS hosted at an agent (b) to have a view of the state data and services an agents supports. As such we decided to build Graphical User Interface (GUI) offering high level functionality to a manager. This GUI represents a management tool on top of the polling based monitoring and event reporting systems we have introduced so far. The management tool is effectively a higher level manager enhancing the functionality of our monitoring and event reporting systems. The development of this tool consisted of two phases. The first was a design phase where we defined with a high level perspective the requirements of the GUI taking into account how the underlying systems operate. The first phase was followed by an implementation oriented design phase of the requirements set in the first phase. The implementation phase involved aspects such as coding, unit testing, integration with existing systems and efficiency of the proposed solutions. The end result is a working prototype that follows the manager-agent paradigm for distributed WS-based monitoring and event reporting.

The management tool is a standalone application. It has four main modules a *data and service module*, a *monitoring module*, a *notification module* and a *connection management module*. The *data and service module* is responsible for displaying to the end user what sort of services and data the agent supports. The *monitoring module* is responsible for creating monitoring jobs in order to retrieve management state data from a series of agents. The *connection module* is used to manage the parameters of the connection between a manager and an agent in order to be able to access the monitoring and event reporting functionality provided by the latter.

In the next section we are going to present only the design decisions for the monitoring tool. Other aspects such as the design as well as the implementation phase details for each module are presented in the appendix. It is suggested for the user to read first the details in the appendix and then the rest of this chapter. At the end of this chapter we present our conclusions.

6.2 Design Decisions

This section is dedicated on the design decisions taken during the development of the management tool. Details such as why certain software tools and techniques were selected during the development of the management tool will be provided in this section on a per module basis. All the main design decisions concern the monitoring module.

The monitoring module is responsible for providing the user with facilities for creating, managing and serialising monitoring jobs. This panel is given in Figure 8-3. This panel allows a user to run simultaneous monitoring jobs with different granularities and parameters.

To be able to manage a collection of monitoring jobs efficiently and program different granularities for each job there are two options. The first option is using Java threads and the second option is the Java Timer Class. The latter class allows us to create a Timer object. This object uses a single thread in order to plan when each job (represented by a Java TimerTask class object) is going to be executed. The alternative from using the Timer class would be to use Java threads for each job task. Using Java threads though could potentially cause regular lockdowns of the functionality of the monitoring tool if the threads are not managed appropriately. This is less probable with a Timer object because Java manages and schedules when a TimerTask (a job) inside it will run. In addition, threads are not efficient in managing the processor time slots. When a Thread is in sleep mode (inactive) the time slot the processor assigns to it is wasted unnecessarily. As such using many threads can lead to creating quite heavy applications especially when their volume increases. On the contrary the Timer object is more lightweight because it is a single thread. If sufficient jobs (TimerTask objects) are added in a timer object they can be run in parallel minimising the time the processor would spend in idle mode. Based on the above, we decided that the Java Timer class is the most appropriate class for creating and running monitoring jobs in parallel.

A second decision we had to take when creating monitoring jobs is how large would be the history of results that each job can keep in memory. If we allow this to be infinite, the manager could run out of memory and the management tool would become unstable. So we limited this number to 30 results per job. Thus the user can enter values only between 1 and 30 when selecting the history size in the create job panel (Figure 8-3).

A third concern for creating jobs with the monitoring module is what tools to use for XML validation. When a user sends a request like the one in Figure 8-1, it has to validate it against a schema so that the request will not fail when it reaches the agent. Depending on the method the user selects from the service tree to retrieve management data for each job, the create job panel highlights the type of queries that are allowed and dims the queries you are not allowed to use. Depending on the queries allowed, the monitoring tool selects the appropriate schema to check its validity. There are three schemas depending on whether the user wants to retrieve multiple instance data, single instance data or whether the user wants to be able to access all the data in a WS MIB. Selecting one of the methods supported by our custom framework defines also which schema to use for validation.

For validating an XML document against a schema we had three options (a) SAX [171] (b) DOM [178] (c) JAXB [177] (Java Architecture for XML Binding). SAX cannot be used by default for validation or for manipulation of XML data but only for displaying XML data. As such SAX is immediately rejected. To select between JAXB and DOM we have to look at their features. We have to keep in mind that our goal in selecting either of these tools is memory efficiency and also efficiency in validating a request message. The latter minimises latency overhead and the former memory overhead.

JAXB is an API that contains a set of classes and a binding compiler allowing us to represent an XML schema through a set of interfaces and Java classes. When a schema is represented by a set of classes the latter can be used in order to un-marshal or marshal an XML document. During the un-marshalling process an XML document is broken down into a set of Java objects. During the marshalling process an XML document can be created from a number of Java objects. During the marshalling or un-marshalling process an XML document can be validated for its conformance against its schema represented as previously said by a number of Java interfaces and classes. Since JAXB breaks down a schema or many schemas into a set of interfaces and classes only once, it then compiles them so that it can be used by any application for XML validation. Using DOM to validate an XML application against its schema would require parsing of an XML schema once for each application, and once when a job is resumed after being suspended. In the monitoring tool each job can be considered as a separate application that has to parse an XML schema and then compare the request document against this schema. Using DOM to do this makes unnecessary use of resources. On the contrary, using JAXB would require each job to use the same precompiled set of classes to validate an XML document. This saves time (latency overhead). As such JAXB is better than using DOM for validating monitoring requests.

JAXB as previously mentioned also allows an XML document to be manipulated through a set of Java objects. Each object can be processed or manipulated instantly simply by accessing that object. On the contrary using DOM to manipulate a document, would require us to go from node to node until finding the node that needs to be manipulated. This wastes a lot of memory. As such JAXB is also better than DOM when manipulating an XML document in terms of the memory it uses.

The only limitation of JAXB is that it works better only when the XML schemas it uses are static. In our case this is exactly what is required. Based on the above we decided to use JAXB for our monitoring module since is saves memory as well as latency overhead.

Having solved all the problems above using the best options available from the software we were aware of, a final problem had to be solved. When a user sets all the parameters for a monitoring job in the create job panel, pressing the "OK" button in this panel would create a monitoring job request which is then validated (Figure 8-3). The custom tool queries in this request need also to be validated for their correctness. To do this the only option available was to use the parser of our custom query tool. This means that validation of the queries is performed both at the manager and the agent. This increases latency a bit for each request but without it a monitoring request could fail at the agent wasting network resources. Not performing validation at the agent is also not possible. This is true because if a manager is not using the monitoring tool for validation of the queries, validation would not be performed neither at the agent nor the manager. Since our query tool is scalable in terms of latency and especially when compared to XPath we decided to perform validation at the monitoring tool and also at the agent. We believe that this is better than spending network resources (traffic) for queries that can not be validated and that will fail when they reach the agent.

When displaying monitoring jobs it is possible to view the data result history of a monitoring job in the form of a real time table and a real time chart. The Java API we selected for displaying results in the form of a chart is the JFreeChart time series API. We used this API because if fulfils almost all of our design requirements. JFreeChart has the following characteristics.

- It is relatively lightweight.
- It is open source.
- It is well documented as it is one of the most matured projects for charting and graphing.
- It supports most types of charts like bar charts, line charts, pie charts, time series charts etc.
- It can be embedded in any Java application.
- It supports dynamic repainting of the chart in the case where for example the entire data pool changes. The latter is required when the user deselects a job and selects another.

The only problem that might exist with this API is that its drawing performance may not be very good if the frequency for displaying result samples is too high (i.e. frequency of change is in the order of milliseconds). As such we restricted the granularity of measurements in the create job panel to that of seconds (1 second is the minimum value that can be selected). We believe this is more than enough. For example in the QoS passive network measurements in chapter four, the granularity requirements were in the order of seconds, minutes or hours. For these measurements JFreeChart would satisfy any of the required granularity requirements.

A user should also be able when he/she quits the management tool to be able to view the jobs he/she added in a previous session. This task can be facilitated using a process called serialization. With serialisation before the end user shuts down the management tool, the parameters of each monitoring job are saved in persistent storage. Each job and the TimerTask object associated with it are serialised in persistent storage by implementing the java.io.Serializable interface. Object serialisation through this API allows us to take an object's state and convert it into a stream of data for storage. With object serialisation, making any object persistent becomes easy as you do not have to write custom code to save object member variables in a file. Each object can be restored at a later time, even from another location. In fact, it is possible to even move an object from one computer to another and have the latter maintain its state. The persistent storage of the Serializable API is a file on the hard disk. When the management tool is started, the serialised jobs are extracted from this file and are restored in memory. Then the monitoring process for each job resumes, as the TimerTask object for each job is reinserted back in the Timer object.

Serialisation is a better and more flexible option than storing data for each job in text files and retrieving them back again when they are required. This is why we opted for it.

6.3 Summary and Conclusions

In this chapter we have presented a Graphical User Interface (GUI) offering high level functionality to a manager controlling the monitoring and event reporting systems we have introduced in the previous chapters. The management tool is effectively a higher level manager for accessing and enhancing the functionality of our monitoring and event reporting systems. In this chapter we also have presented the design decisions behind this monitoring tool in order to support the functionality of the aforementioned systems. These decisions are critical in order for the proposed solutions to be appropriate enough so that the efficiency of our monitoring and event reporting systems is not harmed. The development of this tool consisted of two phases. The first phase was a design phase where we defined with a high level perspective the requirements of the GUI taking into account how the underlying systems operate. The second phase was the implementation phase where we analysed the implementation aspects of the tool. Both of these phases are described in the appendix.

Through our analysis in the appendix and in this chapter we can observe that the monitoring tool has the following characteristics:

- It supports a variety of monitoring jobs with different granularity requirements and parameters. These jobs can be run simultaneously allowing the end user to perform lots of simultaneous monitoring tasks.
- It supports efficient management of monitoring jobs by providing facilities for creating, suspending, resuming, deleting and editing a monitoring job.
- It supports displaying monitoring results in a user friendly manner through a real time table or through a real time chart. This way the user can study the behaviour of certain management counters.
- It supports serialisation of monitoring jobs so that the user does not have to create the same monitoring jobs every time he/she turns on the management tool after turning it off.
- It uses a timer object to manipulate a series of monitoring jobs so as to prevent application lock downs and also to make efficient use of the processor.
- It supports user friendly viewing of the services and the data the agent supports. As such the user can easily select the service to use to retrieve data and also which data to retrieve for monitoring.
- It uses JAXB for efficient validation of a monitoring request or an event reporting

subscription request against its schema. This minimises latency and memory overhead requirements.

- Through the notification module the monitoring tool supports the use of the WS notification standard messages. It also supports receiving and displaying application specific event reporting messages.
- Through the notification module the monitoring tool supports displaying events in a user friendly manner through a web browser.

The management tool though also has some limitations:

- Currently the management tool does not support the standard messages of MUWS but only those of our custom framework. It is in our future objectives to support the messages supported by the MUWS standard. We also are in favour of supporting the message operations of the common management standard that will be developed for interoperability purposes by IBM and Microsoft. This can happen as soon as we will make the changes proposed in chapter three and when the schemas and the WSDL descriptions for these standards appear.
- The notification module needs to become more user-friendly. Currently the notification module supports event subscription based on an XML document template that provides information on how to complete it. In the future we will provide a more interactive manner for creating a subscription request than using a template. This will allow the end user to make no mistakes when he/she completes a subscription request.
- Currently the monitoring tool supports only a limited number of functions for displaying results. The WMA function is one of them. In the future we will include functions for (a) finding the minimum and the maximum of the data in the result history (b) compute the variance and the standard deviation of the data in the result history (c) summation of a series of results to compute for example PHB throughput for the second QoS scenario etc. These functions will be provided as an option in the create job panel.

Still these limitations do no affect the functionality of our tool. It is in our future goals to amend these limitations. In essence though, this is a necessary tool supporting and enhancing the functionality of the monitoring and event reporting systems we have introduced in previous chapters.

Chapter 7

7 Conclusions and Future Work

In this final chapter we bring together the work presented in chapters 3-6 of this thesis. We remind to the reader that the main objectives of this thesis were:

- To perform an investigation of mechanisms in order to improve the scalability and performance of WS for WS-based monitoring and event reporting.
- To build a custom framework and associated systems as part of an architecture that supports distributed and scalable monitoring and event reporting based on these mechanisms.
- To evaluate the scalability of the proposed mechanisms and solutions.
- To design and implement a Graphical User Interface to enhance the capabilities of the above systems.

Based on these objectives, in Section 7.1 we highlight the research contributions and discuss the importance of the main achievements with respect to these objectives. In Section 7.2 we identify directions and areas for potential future research in this area.

7.1 Conclusions

The detailed research contributions of this thesis were given during the analysis and conclusions of chapters 3-6. In this section we re-iterate through our achievements.

With respect to the investigation of mechanisms in order to improve the scalability and performance of WS-based monitoring and event reporting we have identified and proposed a number of things.

WS are a technology with quite a substantial overhead compared to other technologies such as CORBA and SNMP, due to the verbosity of the XML tags describing the context of each piece of information. This increases the application footprint of WS applications when using them for management purposes substantially. Despite the big application footprint though, when using WS and also other technologies such as SNMP and CORBA for monitoring and event reporting it may not be necessary to retrieve the whole state of a device as the latter may be very large. In such cases, mechanisms that can support information processing for bulk and selective retrieval can be extremely beneficial in minimising the management application footprint. Some of the previous technologies in the past did not provide efficient mechanisms for bulk or selective retrieval.

SNMP for example does not support any selective retrieval mechanisms whereas its bulk retrieval mechanisms in some situations may prove inefficient. CORBA on the other hand can support bulk retrieval mechanisms (JIDM) but its filtering mechanisms may be limited and proprietary. In WS sub-tree filtering and XPath have been suggested as mechanisms/tools to support merging (bulk) and filtering (selective) operations on XML data for configuration management, whereas the latter has also been suggested for monitoring and event reporting of the state of network devices. There are various concerns though as explained in chapter two, that these mechanisms/query-tools may be too heavyweight under certain situations. It is thus evident from all the above that offering mechanisms/query-tools for bulk and selective retrieval is not a trivial task and requires careful consideration of the characteristics that these tools should have.

In CMIP and CMIP++ the engineering architects of these solutions acknowledge the fact that state data representing the underlying resources can share a number of relationships such as containment etc. These relationships in CMIP/CMIP++ were used in order to retrieve the state of a device more efficiently. In CMIP++ for example, the relationships that state data and the objects encompassing these data share, are used to support bulk retrieval of management information though scoping operations.

Programming language objects though are not the only programming construct that can be used to encompass management state data. WS can also encompass state data and as a result WS can also share relationships due to the state data they encompass. Exploiting these relationships between WS would allow a WS management application to perform bulk information retrieval for monitoring or event reporting. But there are two problems moving from relationships in CMIP++ to relationships in WS: Firstly having to search relationships between objects for data and having to search relationships between WS to do the same, presents a significant difference. In CMIP and CMIP++ object oriented principles such as containment, facilitated the structuring of state data in hierarchies with different levels of abstraction. This allowed searching for state data more effectively. WS offering access to management state data do not perform this by default. Secondly even if there was a way to build hierarchies of WS, supporting the containment relationships in order to build these hierarchies, requires providing collective access from the state data of one WS to the state data of other WS.

To solve the first problem in order to be able exploit the relationships between WS for bulk retrieval using a query tool, we introduced a number of rules in order to be able to structure WS in hierarchies (trees). As such we had to introduce three rules. The first rule is that if a WS shares a containment relationship with other WS (contains data from other WS as well as its own) it should lie a level higher in the WS hierarchy tree. The second rule is that if a WS shares any other type relationship apart from containment with another WS, both WS should lie on the same level of the hierarchy tree. The third rule is that if a WS shares both containment and other relationships with other WS, containment is a stronger relationship when classifying a WS in the hierarchy tree. Based on all these rules, it is possible to form a hierarchical tree of WS to facilitate bulk information retrieval for monitoring. Then a query tool can be used to navigate these relationships to retrieve management state data from the WS hierarchy tree.

To solve the second problem in order to support the containment relationships required to build WS-hierarchies, we suggested the usage of a currently available specification to support collective access from the state data of a WS-Resource to the state data of other WS-Resources. The WS-SG specification allows us to perform such a feat combined with certain features of WSDL 2.0. We elaborated in chapter 3 how this is possible.

In addition to the previous, in many cases a management station requires retrieving management state data from many managed devices. It may not be scalable in these cases for a single manager to handle the monitoring and processing load from many managed devices because this task could be overwhelming for one entity. As such a WS-based query tool used in order to retrieve management state data should enable the manager to distribute the monitoring load to a series of agents where the processing of data can take place, before returning the latter to the manager. In essence a query tool should enable a manager to delegate the monitoring processing load to a series of agents.

Furthermore as it was shown in previous research, the cost of processing and encoding a great volume of state data stored in XML format can be large due to the verbosity of XML tags. The cost of encoding XML information can not be alleviated. The cost of processing XML data though can. As such, instead of having a query tool to process XML data, query tools can be used instead to process raw data. After performing processing operations on raw data, it is possible for a query tool to use XML tools to structure an XML response. The latter may be a better tactic in minimising the processing cost of WS-based monitoring and event reporting applications.

Moreover as shown by other research in [34] for event reporting applications, custom WS-based solutions for polling based monitoring may also be more efficient than general tools. As such it may be more favourable to build solutions that operate in an application specific manner for monitoring within a network domain and in a standard manner at the edges of a domain for increased interoperability. As such a query tool should be able to work not only as part of a custom framework within a network domain but also as part of a standard framework at the edges of that domain.

Combining all the previous, we proposed that a custom query tool should encompass all the above requirements for polling based monitoring and also for event reporting. Based on this we proposed that a query tool for retrieving management information in a bulk or selective retrieval manner from WS should (a) exploit the relationships state data share in order to be able retrieve the state of a device in a bulk manner (b) perform processing operations on raw data and not on XML (c) help distribute the monitoring processing load to a series of entities instead of restricting the latter to a single entity (d) work with lightweight frameworks that use application specific messages for monitoring or event reporting for performance when used within a network domain and with a standard framework at the edges of a domain for interoperability (e) allow selective retrieval of state data through information processing and filtering.

In addition, we have also observed that in many monitoring scenarios, event reporting usually triggers a manager to perform a set of actions. These actions may be requests to retrieve more system data. This occurs because in several cases as for example in SNMP, events may not contain enough information so that the manager understands the true nature and cause of a fault or a problem. Also in many event reporting scenarios, events trigger a manager to perform changes on the network in order to fix faults (i.e. configuration of the network). Finding ways to perform a set of actions and tasks, normally performed by the network entity receiving an event (manager), in order for these actions to be performed by the network entity producing the event (agent) in the first place has the potential of making the notification process more efficient. The latter process is called task delegation. When performing task delegation though, we highlighted that it is necessary not to hardwire the logic and the tasks that can be performed by the entity handling the delegated tasks as in [124].

As a way to perform task delegation as part of the event reporting process we proposed the use of policies. Network management policies represent information influencing the behaviour of a management system. Effectively policies are rules that can be used as the means to successfully achieve a goal. Policies use an event-condition-action model of definition in order to (a) ensure compliance, (b) define behaviour, and, (c) achieve adaptability of a system. We can observe that events have a lot of the characteristics that policies also embody. In the network management world events are viewed as a method for notifying an entity (i.e. a manager) about the state of a managed device or underlying resource that usually demands an action to be taken. Thus events comprise information about (a) the event itself, (b) the condition that produces the event, and, (c) the type of actions to be performed after an event is generated. All this information is consistent with the network management perspective of defining policies (event-condition-action). As such it is possible to describe the information of an event, and the processes required to generate an event or the actions performed when an event is generated, as part of a grammar used to describe the event condition action subparts of a policy.

Based on all the above we have proposed that a WS event reporting system should support task delegation and configuration of the event process through policies. In addition, as suggested in [34] a WS-based event reporting system should also be able to use its own conventions and messages for event reporting within a network domain but also should be able to use standard messages at the edges of that domain.

With respect to the above suggestions and the need to build a custom framework and associated systems as part of an architecture that supports distributed and efficient monitoring and event reporting, we have achieved the following:

- We have built and deployed a custom query tool to retrieve management information representing the state of a device for polling based monitoring and event reporting. The tool exploits the relationships that exist between state data hosted in WS to facilitate bulk retrieval from the latter through a series of special queries called Service Selection Queries (SS_Queries). The tool also supports selective retrieval capabilities in order to retrieve the state of a device more efficiently through a series of queries called data queries. This tool was designed to operate on raw data in order to minimise the processing overhead cost, and it has several other optimisations to minimise its footprint.
- We have designed and built a custom framework supporting distributed monitoring and task delegation as part of a distributed monitoring architecture. This architecture uses the SS_Queries of our query tool and the concept of a callback address mechanism borrowed from event reporting to support distributed polling based monitoring. Through the use of SS_Queries, our architecture gives a complete view of management services through ONE agent by supporting federation of management requests. As we discussed in the end of chapter 3 this can become an alternative solution for distributed monitoring under certain conditions. Still as discussed in chapter 3 this solution has some limitations.
- We have shown how to convert our query tool and our architecture to support distributed monitoring using messages and concepts from standardised solutions managing state (i.e. the MUWS standard). We have integrated MUWS and our custom query tool in [180]. In [180] we have shown using the third QoS scenario that MUWS and our custom framework using our custom query tool can be potentially used within a network domain for increased performance. At the same time in [180] we also suggest the usage of XPath and MUWS at the edges of a network for increased interoperability recognising that XPath is a standard and thus more appropriate for this task (although this solution is shown not to be as scalable as using MUWS with our custom query tool).
- We have built a policy grammar in order to perform task delegation through policies for event reporting.

- We used this grammar as part of the WS-Notification standard subscription messages and operations so as to be able to support standardised as well application specific messages for event reporting.
- We have built an event service supporting task delegation through policies, and our policy grammar and bulk and selective retrieval through our query tool. Our query tool enables us to collect the appropriate information for event reporting. Our policy grammar allows us to delegate a number of varied complexity tasks from a manager to an agent. This way our event service
    - Can be more flexible in the range of tasks it can perform.
    - Becomes capable of collecting more information in order to help the manager in pinpointing the root of a problem reported inside an event.
    - Can minimise under certain situations the footprint of event reporting operations.
    - Has the potential of promoting the use of more autonomous solutions in managing the event reporting process limiting the supervision of a manager.
    - Adds as shown extra complexity in WS-based event reporting systems.

With respect to evaluating the scalability and efficiency of the proposed mechanisms, solutions and frameworks, we conducted a series of tests based on measurement scenarios we come across in QoS networks over MPLS enabled devices.

Initially we evaluated the performance of our query tool with XPath implementations of version 1.0 and 2.0. When testing the efficiency of our query tool we have shown as expected due to its optimisations that it is more scalable compared to XPath in terms of latency and memory overhead especially when the volume of management information increases. In addition our query tool is also more efficient compared to XPath in terms of traffic overhead.

Based on the fact that our tool is more scalable than XPath implementations we decided to use it in evaluating the performance of our WS based custom framework for polling based monitoring against another management protocol, SNMP. We tested the performance of these technologies for scenarios where processing, filtering and bulk retrieval of state data is required. Through our tests we have shown that our custom framework can be more scalable than SNMP in some situations and generally exhibits good performance compared to SNMP in other situtations. This might not be the case if XPath was used instead of our query tool. Our tests have also shown, that if SNMP was used to retrieve management information in a specific order (i.e. on a per PHB basis) or in scenarios where filtering is required, which is the case for several monitoring scenarios, the performance of the latter might not be better as expected, even if the distributed management extensions of SNMP would be used. Based on the above we believe that our custom framework using our query tool can efficiently be used for WS polling based monitoring especially if the newest advancements in parsing/serialization/binarization of XML are used (as shown in [180]).

In order to show that a WS-based event reporting using policies and our query tool for bulk and selective can be more efficient, we compared its performance against two other systems. One of the two systems was also a WS-based event reporting system supporting the use of the WS-Notification standard messages. This system though did not support policies, and supports bulk and selective retrieval either through our query tool or XPath version 1.0 and 2.0. The other system we investigated was a system based on SNMP traps that does not support either policies or filtering mechanisms. Testing the performance of our event reporting system we have shown it can perform equally good to SNMP traps when the volume of management data is small. For a small volume of data, our event reporting system performs equally well to the other WS based system in terms of latency and better in terms of traffic overhead. For a large volume of information, our event reporting system saves latency and traffic overhead compared to the WS based system without policies. When the WS based system without policies uses XPath, its latency is worse in comparison to either our event reporting system or SNMP traps. When testing our event reporting system with SNMP when a large volume of data needs to be processed, the performance of our system is better than that of SNMP, both in terms of latency and also traffic overhead. This is not the case for the second WS-based event reporting system without policies when the latter uses XPath. With all the above we have shown that our event reporting system supporting the use of policies for task delegation and our query tool for bulk and selective retrieval can be used efficiently for WS-based event reporting. At the same time, our event reporting system uses the WS notification standard to be able to send application specific messages within a network domain and WS Notification standard messages at the edges of a domain. This as suggested in [34] is an efficient way to increase performance of WS-based event reporting even more and still preserve the interoperability of WS based event reporting solutions.

With respect to designing and implementing a Graphical User Interface to enhance the capabilities of our monitoring and event reporting systems we have designed and implemented a high level manager tool supporting:

- Simultaneous running of a variety of monitoring jobs with different granularity requirements and parameters.
- Efficient management of monitoring jobs by providing facilities for creating, suspending, resuming, deleting and editing a monitoring job.
- Displaying monitoring results in a user friendly manner through a real time table or through a real time chart. This way our graphical management tool enables the end user to study the

behaviour of certain management counters more effectively.

- Serialisation of monitoring jobs so that the user does not have to create the same monitoring jobs every time he turns on the management tool after turning it off.
- User friendly viewing of the services, the conceptual relationships between them and the data a management agent supports. As such the user can easily select the service to use to retrieve data and also which data to retrieve for monitoring.
- Efficient validation of a monitoring request or an event reporting subscription request against its schema through JAXB. The latter minimises latency and memory overhead.
- Creation of subscription requests though policies for task delegation.
- Receiving and displaying the history of standard and application specific event reporting messages in a user friendly manner through a web browser.

7.2 Future Work

There exist certain directions and areas towards which the work presented in this thesis can be extended.

- Tackle the modelling aspects of relationships between state data and as a result of WS and WS-Resources encompassing these data. This is also an important step in using these relationships for effective monitoring and event reporting apart from exploiting these relationships.
- Expand the capabilities of our query tool so that it can be used not only for selecting data for monitoring and even reporting but also for altering data. This is an essential step in tackling in the future also aspects of configuration management
- We have to refine and extend our policy-like grammar to meet closely the requirements of policy management. Currently our event reporting system is manually configured through policies to perform a set of tasks dynamically at run-time. The essence of policy-based management for event reporting though would be to design an event reporting system that will autonomously deduce the actions to perform within a solution space. The latter requires the refinement of our policy like grammar.

Bibliography

[1] G. Pavlou, "On the Evolution of Management Approaches, Frameworks and Protocols: A Historical Perspective Source", Journal of Network and Systems Management (JNSM) Vol 15, Issue 4, December 2007, pp. 425 – 445.

[2] "Information Technology-Open Systems Interconnections--Systems Management Overview", ITU-T Recommendation X.701 / ISO/IEC 10040, 1992.

[3] U. Warrier, L. Besaw, L. LaBarre, B. Handspicker, "Common Management information Services and Protocols for the Internet (CMOT & CMIP)", IETF RFC 1189, 1990.

[4] J. Case, M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol (SNMP)", IETF, RFC 1157, 1990.

[5] J. Schoenwaelder, "Overview of the 2002 IAB Network Management Workshop", IETF Informational RFC 3535, 2003.

[6] Object Management Group, "The Common Object Request Broker: Architecture and Specification (CORBA) version 2.6", Object Management Group, December 2001.

[7] D. Levi, J. Schoenwaelder, "Definitions of Managed Objects for the Delegation of Management Scripts", IETF RFC 3165, August 2001.

[8] R. Kavasseri, B. Stewart, "Distributed Management Expression MIB", IETF RFC 2982, October 2000.

[9] WikiPedia resources, "Document Type Definitions", June 2008, http://en.wikipedia.org/wiki/Document_Type_Definition.

[10] D.C. Fallside, P. Walmsley, "XML Schema Part 0: Primer Second Edition", W3C recommendation, October 2004, http://www.w3.org/TR/xmlschema-0/.

[11] Distributed Management Task Force, "Web Based Enterprise Management (WBEM)", DMTF Standards and specifications, http://www.dmtf.org/standards/wbem/.

[12] A. Arora et al. Web Services for Management (WS-Management). Desktop Management Task Force (DMTF), DSP0226, December 2008, http://www.dmtf.org/standards/wsman.

[13] W. Vambenepe, "Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 1", Organization for the Advancement of Structured Information Standards (OASIS), August 2006, http://www.oasis-open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf.

[14] W. Vambenepe. Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 2. Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/committees/download.php/20575/wsdm-muws2-1.1-spec-os-01.pdf.

[15] Distributed Management Task Force, "Common Information Model (CIM) Standards", DMTF standards, http://www.dmtf.org/standards/cim/

[16] H. S. Thompson, D. Beech, M. Maloney, N. Mendelsohn, "XML Schema Part 1: Structures version 2.0", W3C Recommendation, October 2004.

[17] P. V. Biron, A. Malhotra, "XML Schema Part 2: Datatypes version 2.0", W3C Recommendation, October 2004.

[18] R. Chinnici, J.J. Moreau, A. Ryman, S. Weerawarana, "Web Services Description Language version 2.0 Part1: Core Language", W3C recommendation, June 2007, http://www.w3.org/TR/wsdl20/.

[19] D. Box, E.Christensen, F. Curbera, et al, "Web Services Addressing", W3C member Submission, August 2004, http://www.w3.org/Submission/ ws-addressing/.

[20] D. Box, L. Cabrera, C. Critchley, et al, "Web Services Eventing (WS-Eventing)", W3C Member Submission, March 2006, http://www.w3.org/Submission/WS-Eventing/.

[21] J. Alexander, D. Box, L. Cabrera, et al, "Web Services Enumeration (WS-Enumeration)", W3C Member Submission, March 2006, http://www.w3.org/Submission/WS-Enumeration/.

[22] J. Alexander, D. Box, L. Cabrera, et al, "Web Services Transfer (WS-Transfer)", W3C Member Submission, September 2006, http://www.w3.org/Submission/WS-Transfer/.

[23] A. Nadalin, C. Kaler, P.H. Baker, R. Monzillo, "Web Services Security: SOAP Message Security 1.0 (WS-Security)", OASIS Standard 200401, March 2004.

[24] H. Kreger, K. Wilson, I. Sedukhin, "Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1", OASIS Standard, August 2006, http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.pdf.

[25] R. Enns, "Network Configuration Protocol", IETF RFC 4741, December 2006, http://www.ietf.org/rfc/rfc4741.txt.

[26] T. Goddard, "Using the Network Configuration Protocol over SOAP, IETF RFC 4743, December 2006, http://www.ietf.org/rfc/rfc4743.txt.

[27] M. Wasserman, T. Goddard, "Using the Network Configuration Protocol over SSH", IETF RFC 4742, December 2006, http://www.ietf.org/rfc/rfc4742.txt.

[28] E. Lear, K. Crozier, "Using the Network Configuration Protocol over BEEP", IETF RFC 4744, December 2006, http://www.ietf.org/rfc/rfc4744.txt.

[29] Alliance for Telecommunications Industry Solutions (ATIS), "ATIS Telecom Glossary", 2007, http://www.atis.org/glossary/.

[30] M. Sloman, "Network and Distributed Systems Management", Copyright Addison-Wesley, 1994.

[31] Internetworking Technologies Handbook, "Chapter 6 Network Management Basics", Cisco, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/nmbasics.pdf.

[32] W. Stallings, "SNMP, SNMPv2 and CMIP: The practical guide to Network Management Standards", Copyright Addison Wesley, 1993.

[33] S. Graham, B. Murray, "Web Services Base Notification 1.2 (WSBaseNotification)", OASIS Working Draft 03, June 2004, http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf.

[34] H. Saiedian, S. Mulkey, "Performance Evaluation of Eventing Web Services in Real-Time Applications", IEEE Communications magazine, Vol 46, No. 3, March 2008, pp. 106-111.

[35] ITU-T Recommendations. M.3010, "Principles for a Telecommunications Management Framework (TMN)", Study Group IV, 1996.

[36] G. Pavlou, "OSI Systems Management, Internet SNMP and ODP/OMG CORBA as Technologies for Telecommunications Network Management", Book Chapter on Telecommunications Network Management: Technologies and Implementations, Wiley-IEEE Press Series on Network Management, December 1997, pp. 63-109.

[37] International Telecommunication Union (ITU-T) X.680, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", July 2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf

[38] N. Catania , P. Kumar, B. Murray, H. Pourhedari, W. Vambenepe, K.Wurster, "HP Web Services Management Framework – Web Services Management V2.0",HP specification, July 2003,

[39] N. Catania , P. Kumar, B. Murray , H. Pourhedari , W. Vambenepe , K. Wurster, "HP Web Services Management Framework – Overview V2.0", HP specification, July 2003.

[40] D. Srinivas, P. Fremantle, A. Suriarachchi, et al, "Web Services are Not Slow – Axis2/Java Performance Testing Round #2 - Apache Axis2 vs. Codehaus XFire in a Contract First Scenario", WS02 OxygenTank The Developer Portal for SOA, January 2007

[41] M. Rose, K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based Internets", IETF RFC1155, May1990, http://www.faqs.org/rfcs/rfc1155.html.

[42] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, "Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)", IETF RFC 1902, January 1996, http://www.faqs.org/rfcs/rfc1902.html

[43] ITU-T Recommendation X.720, "Open Systems Interconnection - Structure of Management Information -Management Information Model (MIM)", 1991.

[44] ITU-T Recommendation X.721, "Open Systems Interconnection - Structure of Management Information: Definition of Management Information (DMI)", 1992.

[45] ITU-T Recommendation X.722, "Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects (GDMO)", 1992.

[46] ITU-T Recommendation X.900, "Open Distributed Processing, "Basic Reference Model of Open Distributed Processing (ODP)", 1995.

[47] R.E. Gruber, B. Krishnamurthy, E. Panagos, "CORBA Notification Service: design challenges and scalable solutions", 17th International Conference on Data Engineering, 2001, pp. 13-20.

[48] M. Tomono, "An event notification framework based on Java and CORBA", 6th IFIP/IEEE International Symposium on Integrated Network Management on Distributed Management for the Networked Millennium (IM 1999), May 1999, pp. 563-576.

[49] T. Rutt, ed., "Comparison of the OSI Systems Management, OMG and Internet Management Object Models", A Report of the Joint XOpen/NM Forum Inter-Domain Management Task force, March 1994.

[50] F. Straub, "Advantages and Disadvantages of the Script MIB infrastructure", A Jasmin Project report, October 2000, http://www.ibr.cs.tu-bs.de/projects/jasmin.

[51] J. Schoenwaelder, "Traditional Approaches to distributed management", NMRG meeting, 2006,http://www.ibr.cs.tu-bs.de/projects/nmrg/meetings/2006/stockholm/nmrg-19-schoenw.pdf.

[52] R. Lopes, J. Oliveira, "Delegation of Expressions for Distributed SNMP Information processing", 8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003), March 2003 pp. 395-408.

[53] M. Daniele, B. Winjen, D. Fransisco, "Agent Extensibility Protocol Version 1.0", IETF RFC 2257, January 1998, http://www.ietf.org/rfc/rfc2257.txt.

[54] IETF Distributed Management Proceedings, December 1997 http://www3.ietf.org/proceedings/97dec/ 97dec-final-67.h

[55] J.P Martin-Flattin, "Web-Based Management of IP networks and Systems" copyright Wiley series, 2003.

[56] W. Vogels, "Web Services are not Distributed Objects: Common Misconceptions about the Fundamentals of Web Service Technology", IEEE Internet Computing, November-December 2003.

[57] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "The Web Services Description Language (WSDL) 1.1" , W3C Note, March 2001, http://www.w3.org/ TR/wsdl.

[58] T. Bellwood, "Understanding UDDI: Tracking the evolving specification", IBM article, July 2002, http://www.ibm.com/developerworks/library/ws-featuddi/.

[59] D. Box , D. Ehnebuske, G. Kakivaya, A. Layman , N. Mendelssohn , H. F. Nielsen ,S. Thatte , D. Winer, "The Simple Object Access Protocol (SOAP) 1.1" ,W3C Note, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[60] M. Gudgin, M. Hadley, N. Mendelsohn et al, "SOAP Version 1.2 Part 1 & 2: Messaging Framework and Adjuncts", W3C Recommendation, April 2007 http://www.w3.org/TR/soap12-part1/, http://www.w3.org/TR/soap12-part2/.

[61] N. Chase, "Understanding Web Services specifications, Part 1: SOAP", IBM article, May 2006, http://www.ibm.com/developerworks/edu/ws-dw-ws-understand-web-services1.html.

[62] A. Skonnard, "Understanding SOAP", Microsoft article, October 2003, http://msdn2.microsoft.com/en-us/library/ms995800.aspx.

[63] N. Chase, "Understanding Web Services specifications, Part 2: WSDL", IBM article, July 2006, http://www.ibm.com/developerworks/edu/ws-dw-ws-understand-web-services2.html.

[64] A. Skonnard, "Understanding WSDL", Microsoft article, October 2003, http://msdn2.microsoft.com/en-us/library/ms996486.aspx.

[65] N. Chase, "Understanding Web Services specifications, Part 3: UDDI", IBM article, July 2006, http://www.ibm.com/developerworks/edu/ws-dw-ws-understand-web-services2.html.

[66] R. Butekk, "Which Style of wsdl should I use", IBM article, May 2005 http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/.

[67] J. van Sloten, A. Pras, M. van Sinderen, "On the standardisation of Web service management operations", Proceedings of the 10th IFIP Open European Summer School and Workshop, June 2004, pp. 143-150.

[68] C. Ferris, A. Karmarkar, P. Yendluri, et al, "WS-Interoperability basic profile version 1.2", WS Interoperability Organisation Approval Draft, October 2007, http://www.ws-i.org/Profiles/BasicProfile-1_2(WGAD).html

[69] M.Mandal, H. Silberman, "Understanding Web Services specifications, Part 2: WS-Interoperability", IBM article, April 2007, http://www.ibm.com/developerworks/edu/ws-dw-ws-understand-web-services6.html.

[70] S. Padmanabhuni, A. Chaudhari, S Bharti, S. Kumar, "WSDL 2.0: A Pragmatic Analysis and an Interoperation Framework Minimizing interoperation issues with a WSDL version management framework", SYS-CON media magazine article, June 2007, http://wldj.sys-con.com/read/219029.html

[71] R. Nagappan, "Adopting to WSDL 2.0: Rationale and Known Issues" OASIS committees article, June 2007, http://www.oasis-open.org/committees/download.php/24991/Adopting %20WSDL%202.pdf.

[72] F. Curbera, D. Ehnebuske, D. Rogers, "Using WSDL in a UDDI Registry 1.05", UDDI organisation Working Draft Document, http://www.uddi.org/pubs/wsdlbestpractices-V1.05-Open-20010625.pdf, June 2001.

[73] P. Brittenham, F. Cubera, D. Ehnebuske, S. Graham, "Understanding WSDL in a UDDI registry, Part 1: How to publish and find WSDL service descriptions", IBM article, Sept 2001 http://www.ibm.com/developerworks/webservices/library/ws-wsdl/.

[74] P. Brittenham, "Understanding WSDL in a UDDI registry, Part 2: How to publish and find WSDL service descriptions", IBM article, Sept 2001 http://www.ibm.com/developerworks/ webservices/library/ws-wsdl2/.

[75] The Apache Software Foundation, "The Apache Tomcat Web Server Documentation version 6.0", http://tomcat.apache.org/tomcat-6.0-doc/index.html.

[76] The Apache Software Foundation, "The Apache AXIS Servlet Documentation version 1.4", http://ws.apache.org/axis/java/index.html.

[77] The Apache Software Foundation, "The Apache AXIS Servlet Documentation version 2.0", http://ws.apache.org/axis2/1_4/userguide.html.

[78] Distributed Management Task Force, "WS-Management CIM Binding Specification (CIM-Binding)", DMTF Specification DSP0227, June 2006, http://www.dmtf.org/ standards/wsman.

[79] Distributed Management Task Force, "WS-CIM Mapping Specification (CIM-Mapping)", DMTF Specification DSP0230, http://www.dmtf.org/standards/ published_documents/DSP0230.pdf

[80] S. Parastatidis, J. Webber, P. Watson, T. Rischbeck, "A Grid Application Framework based on Web Services Specifications and Practices", A Web Services Grid Application Framework Project Technical Report, 2003.

[81] I. Foster, J. Frey, S. Graham, et al, "Modelling stateful Resources with Web Services", HP, IBM, Fujitsu White Paper, May 2004, www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf.

[82] J. Clark, S. DeRose, "XML Path Language v 1.0", W3C recommendation, November 1999, http://www.w3.org./TR/xpath.

[83] A. Berglund, S. Boag, D. Chamberlin, et al, "XML Path Language Version 2.0 (XPath)", W3C Recommendation, January 2007, http://www.w3.org/TR/xpath20.

[84] K. Czajkowski, D. Ferguson, I. Foster, et al "The Web Services Resource Framework Version 1.0 (WS-RF)" OASIS Standard, May 2004, http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[85] S. Graham, J. Treadwell, "The Web Services Resource Version 1.2 (WS-Resouce)" OASIS standard, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.

[86] S. Graham, J. Treadwell, "The Web Services Resource Properties Version 1.2 (WS-ResourceProperties)" OASIS Standard, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf.

[87] L. Srinivasan, T. Banks, "The Web Services Resource LifeTime Version 1.2 (WS-ResourceLifeTime)" OASIS Standard, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf.

[88] T. Maguire, T. Banks, D. Snelling, "The Web Services Service Group Version 1.2 (WS-ServiceGroup)", OASIS standard, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf.

[89] L. Liu, S. Meder, "The Web Services Base Fault Version 1.2" OASIS standard, April 2006, http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf.

[90] I. Foster, K. Czajkowski et al, "Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF", IEEE Proceedings of the IEEE, Vol. 93, no. 3, MARCH 2005.

[91] S. Graham, B. Murray, "Web Services Base Notification Version 1.2", OASIS Working Draft, June 2004, http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-03.pdf.

[92] D. Booth, H. Haas, F. McCabe, et al, "The Web Services Architecture (WS-ARCH)", W3C Working Group Note, February 2004, http://www.w3.org/TR/ws-arch/.

[93] H. Kreger, "A Little Wisdom about WSDM", IBM article, March 2005, http://www-128.ibm.com/developerworks/webservices/library/ws-wisdom/.

[94] K. Cline, J. Kohen, D. Davis, et al, "Towards converging WS standards for resources events and management", March 2006, White Paper by IBM, HP, Intel and Microsoft, http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/Harmonization_Road map.pdf.

[95] B. Reistad, B. Murray, D. Davis, et al, "Web Services Resource Transfer Version 1.0", August 2006, http://schemas.xmlsoap.org/ws/2006/08/resourceTransfer/WS-ResourceTransfer.pdf

[96] K. Ballinger, B. Bissett, "Web Services Metadata Exchange Version 1.1 (WS-MEX)" August 2006, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-mex/metadataexchange.pdf.

[97] D. Srinivas, P. Fremantle, A. Suriarachchi, et al, "Web Services are Not Slow- Axis2/Java – Performance Testing Round #1", WS02 OxygenTank The Developer Portal for SOA, May 2006

[98] M. Gudgin, M. Hadley, T. Rogers, "Web Services Addressing 1.0-Core", W3C Proposed Recommendation, March 2006, http://www.w3.org/TR/2006/PR-ws-addr-core-20060321/.

[99] B. Linker, "Introduction to WS-Addressing", By Developers for Developers Article, January 2005, http://dev2dev.bea.com/pub/a/2005/01/ws_addressing_intro.html.

[100] H. Kreger, "Fullfilling the Web Services Promise" Communications of the ACM, Vol. 46, No. 6, June 2003.

[101] S. Bajaj, D. Box, D. Chappell, et al,"Web Services Policy 1.2 - Framework (WS-Policy)", W3C Member Submission, April 2006, http://www.w3.org/Submission/WS-Policy/.

[102] T. Andrews, F. Curbera, H. Dholakia, et al, "Business Process Execution Language for Web Services Version 1.1", May 2003, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf.

[103] M. Feingold, R. Jeyaraman, "Web Services Coordination (WS-Coordination) Version 1.1", OASIS Standard Incorporating Approved Errata, July 2007, http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os.pdf.

[104] M. Little, Andrew Wilkinson, "Web Services Atomic Transaction 1.1 (WS-AtomicTransaction)", OASIS Standard Incorporating Approved Errata, July 2007, http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-errata-os.pdf.

[105] T. Freund, M. Little, "Web Services Business Activity (WS-BusinessActivity) Version 1.1", OASIS Standard Incorporating Approved Errata, July 2007, http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os.pdf.

[106] A. Nadalin, C. Kaler, R. Monzillo, P. Baker, "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", OASIS standard, February 2006, http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf.

[107] A. Nadalin, M. Goodner, M. Gudjin, A. Barbir, H. Granqvist, "WS-Trust Version 1.3", OASIS standard, March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf.

[108] A. Nadalin, M. Goodner, M. Gudjin, A. Barbir, H. Granqvist, "WS-SecureConversation version 1.3" OASIS standard, March 2007, http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ ws-secureconversation-1.3-os.pdf.

[109] H. Lockhart, S Andersen, J. Bohren et al, "Web Services Federation Language Version 1.1 (WS-Federation)", December 2006, http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf.

[110] K. Iwasa, J. Durand, T. Rutt, et al, "Web Services Reliable Messaging TC (WS-Reliability 1.1)", OASIS standard, November 2004, http://docs.oasis-open.org/wsrm/ws-reliability/v1.1.

[111] S. J. Zilora, S. S. Ketha, "Think Inside the Box! Optimizing Web Services Performance Today", IEEE Communications magazine, Vol. 46, No. 3, pp. 112-117.

[112] D. Davis, M. Parashar, "Latency Performance of SOAP Implementations", Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2002, http://www.caip.rutgers.edu/TASSL/Papers/p2p-p2pws02-soap.pdf

[113] A.Ng, S. Chen, P. Greenfield, "An Evaluation of Contemporary Commercial SOAP Implementations", 5th Australasian Workshop on Software and System Architectures (AWSA2004), pp. 64-71, April 2004.

[114] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. van Engelen, M. J. Lewis, "Toward Characterizing the Performance of SOAP Toolkits", Proceedings of 5th IEEE/ACM International Workshop on Grid Computing, November 2004, http://www.extreme.indiana.edu/xgws/papers/soap_perf_char_grid2004.pdf

[115] K. Chiu, M. Govindaraju, R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, pp. 246-254, July 2002, http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/hpdc/2002/1686/00/1686toc.xml&DOI=10.1109/HPDC.2002.1029924

[116] R. Engelen, "Code Generation Techniques for Developing Light-Weight XML web services for Embedded devices", Proceedings of the 2004 ACM symposium on Applied computing, pp. 854-861, March 2004.

[117] SUN, "SOAP with Attachments API Version 1.3 (SAAJ) Documentation", https://saaj.dev.java.net/source/browse/*checkout*/saaj/saaj-ri/docs/index.html

[118] C. Werner C. Buschmann. S. Fischer, "Compressing SOAP Messages by using Differential Encoding" IEEE International Conference on Web Services, pp. 540-547, July 2004.

[119] A. Ng, P. Greenfield, S. Chen, "A Study of the Impact of Compression and Binary Encoding on SOAP performance", 6th Australasian Workshop On Software and System Architectures (AWSA 2005), p.p. 46-55, March 2005, http://mercury.it.swin.edu.au/ctg/AWSA05/ Papers/ng.pdf.

[120] G. Pavlou, P. Flegkas, S. Gouveris, A. Liotta, "On Management Technologies and the Potential of Web Services", IEEE Communications Magazine, Vol. 42, Issue 7, pp. 58-66, July 2004.

[121] T. Drevers, "Performance of Web Services based Network Monitoring", Thesis for a Master of Science degree in Telematics from the University of Twente, Enschede, the Netherlands.

[122] A. Pras, T. Drevers, R. Meent, D. Quartel, "Comparing the performance of SNMP and Web Services Based Management", Electronic Transactions on Network and Service Management (eTNSM), Vol. 1, Issue 2, Fall 2004.

[123] G. Moura, G. Sanchez, R. Gaspary, L. Granville, L. Zambenedetti, "On the Performance of WS Management Standards-An Evaluation of MUWS and WS-Management for Network Management", May 2007, pp. 459-468, IM 2007.

[124] W. Lima, R. S. Alves, R. L. Vianna, et al "Evaluating the performance of SNMP and Web Services Notifications", 10th IFIP/IEEE Network Operations and Management Symposium (NOMS 2006), pp. 546-556, April 2006.

[125] D. Sosnoski, "Java Web services, Part 2: Digging into Axis2: AXIOM -The AXIOM document object model is the foundation for Apache Axis2 Web service", IBM article, Nov 2006

[126] T. Klie, F. Strauβ, "Integrating SNMP agents with XML-Based Management Systems", IEEE communications Magazine, Vol. 42, Issue 7, pp. 76 – 83, July 2004.

[127] T. Klie, F. Strauβ "Towards XML oriented Internet Management", 8th International IFIP/IEEE Symposium on Integrated Network Management (IM 2003), pp. 505-518, March 2003.

[128] R. Neisse, L. Vianna, L. Granville, M. Almeida, L. Margarida, R. Tarouco, "Implementation and Bandwidth Consumption Evaluation of SNMP to Web Services Gateways", IFIP/IEEE Network Operations and Management Symposium, 2004 (NOMS 2004), Vol. 1, pp. 715 – 728, April 2004.

[129] K. McCloghrie, M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", IETF RFC 1213, March 1991, http://www.faqs.org/rfcs/rfc1213.html

[130] M. Rose, "Management Information Base for Network Management of TCP/IP-based internets: MIB-II", IETF RFC 1158, March 1991, http://www.faqs.org/rfcs/rfc1158.html.

[131] J. H. Yoon, H. T. Ju, J. W. Hong "Development of SNMP-XML translator and gateway for XML-based Integrated Network Management", International Journal of Network Management, Vol. 13, Issue 4, July 2003.

[132] V. Cridlig, O. Festor, R. State, "Role-Based Access Control for XML Enabled Management Gateways", IFIP/IEEE Distributed Systems and Operations Management Workshop (DSOM 2004), Vol. 3278/2004, pp. 183-195, October 2004.

[133] T. Fioreze, L. Z. Granville, M. J. Almeida, L. R. Tarouco, "Comparing Web Services with SNMP in a Management by Delegation Environment". IFIP/IEEE Integrated Management conference (IM 2005), pp. 601-614, May2005.

[134] M. J. Choi, H. M. Choi, J. W. Hong, H. T. Ju, "XML Based Configuration Management for IP network devices", IEEE communications Magazine, Vol. 42, Issue 7, pp. 84-91, July 2004.

[135] L. E. Menten, "Experiences in the Application of XML for Device Management", IEEE communications Magazine, Vol. 42, Issue 7, pp. 92-100, July 2004.

[136] S. M. Yoo, H. T. Ju, J. W. Hong, "Performance Improvement Methods for NETCONF-Based Configuration Management", APNOMS 2006, Vol. 4238/2006, pp. 242-252.

[137] V.Cridlig, H. Abdelnur, J. Bourdellon, R. State, "A NetConf Network Management Suite: ENSUITE", 5th IEEE International Workshop on IP Operations and Management (IPOM 2005), Vol. 3751/2005, pp. 152-161.

[138] M. MacFaden, D. Partain, J. Saperia, W. Tackabury, "Configuring Networks and Devices with the Simple Management Network Protocol", IETF RFC 3512, April 2003.

[139] Mi-Jung Choi, J. W. Hong, H.T Ju, "XML-Based Network Management of IP Networks", ETRI journal, Vol. 25, Number 6, December 2003.

[140] J.H.Yoon, H.T. Ju and J. W. Hong "Development of SNMP-XML Translator and Gateway for XML-based integrated network management", International Journal of Network Management, Vol. 13, Issue 4, July 2003.

[141] P. Shafer "XML-based network management", white paper from Juniper Networks on JUNOScript, http://www.juniper.net/solutions/literature/white_papers/200017.pdf

[142] J. Schonwalder, A. Muller, "Reverse Engineering Internet MIBs", International Symposium of Integrated Network Management (IM 2001), May 2001.

[143] N. Catania HP, P. Kumar, B. Murray, H. Pourhedari HP, W. Vambenepe, K. Wurster, "HP Web Services Management Framework – Overview V2.0", July 2003.

[144] N. Catania, P. Kumar, B. Murray, H. Pourhedari, W. Vambenepe, K. Wurster,"HP Web Services Management Framework – Foundation V2.0", July 2003.

[145] R. van Engelen," gSOAP 2.7.10 User Guide", February 2008, http://www.cs.fsu.edu/~engelen/soapdoc2.pdf.

[146] A. Slominski, "XSOAP a.k.a SOAP RMI Documentation", http://www.extreme.indiana.edu/viewcvs/~checkout~/xsoap-java/README.html.

[147] M. Kutter, P. Kulchenko, B. Reese, "SOAP::Lite Documentation", http://search.cpan.org/~mkutter/SOAP-Lite-0.710.07/lib/OldDocs/SOAP/Lite.pm.

[148] W. Ye, "Web services programming tips and tricks: Improve interoperability between J2EE technology and .NET, Part 1-WSDL, RPC/encoded style, and WS-I conformance", IBM article, December 2004, http://www.ibm.com/developerworks/webservices/library/ws-tip-j2eenet1

[149] ITU-T Recommendation X.711, "Open Systems Interconnection - Common Management Information Protocol Specification version 2.0'', 1991.

[150] G. Pavlou, A. Liotta, P.Abbi, "CMIS/P++: extensions to CMIS/P for increased expressiveness and efficiency in the manipulation of management information", 7th Annual Joint conference of the IEEE Computer and Communications Societies, INFOCOM 98, Vol. 2, April 1998, pp.430 – 438.

[151] C. Srinivasan, A. Viswanathan, T. Nadeau, "MPLS Traffic Engineering MIB (TE-STD MIB)", IETF RFC 3812, June 2004, http://www.rfc-archive.org/getrfc.php?rfc=3812.

[152] C. Srinivasan, A. Viswanathan, T. Nadeau, "MPLS Label Switching Router MIB (LSR MIB)", IETF RFC 3813, June 2004, http://www.ietf.org/rfc/rfc3813.txt.

[153] T. Nadeau, C. Srinivasan, A. Viswanathan, "MPLS Forwarding Equivalence Class To Next Hop Label Forwarding Entry MIB (FEC-To-NHLFE)", IETF RFC 3814, June 2004, http://www.ietf.org/rfc/rfc3814.txt.

[154] A. Chourmouziadis, G. Pavlou, "Efficient Information Retrieval in Network Management Using Web Services", DSOM 2006 Proceedings, October 2006, Volume 4269/2006 pp. 1-12.

[155] A. Chourmouziadis, G. Pavlou, O.Gonzalez-Duque, "Using MUWS for scalable distributed Network Monitoring of the state of manageable Resources, work submitted in IM 2009.

[156] T. Berners-Lee, R. Fielding, L. Masinter, "The Uniform Resource Identifier (URI): Generic Syntax", IETF RFC 3986, January 2005.

[157] J. Schonwalder, A. Pras, and J. P. Martin-Flatin, "On the Future of Internet Management Technologies", IEEE Communications Magazine, Oct. 2003, Vol. 41, Issue 10, pp. 90- 97.

[158] P. Naur, "Revised Report on the Algorithmic Language of ALGOL 60," Communications of the ACM, May 1960, pp. 299-314.

[159] A.Chourmouziadis, G. Pavlou, "An Evaluation of WS Tools to Address Efficient Information Retrieval for Network Management", to appear in the Proceedings of the 8th International Symposium on Computer Networks (ISCN'2008), Istanbul, Turkey, June 2008.

[160] M. H. Kay, "SAXON 9.0 The XSLT and XQuery Processor", January 2007, http://www.saxonica.com/download/saxon-resources8-9.zip

[161] SUN, "The Java API for XML Processing (JAXP) version 1.4", May 2007, http://java.sun.com/webservices/jaxp/

[162] A. Asgari, R. Egan, P. Trimintzios, G. Pavlou, "Scalable Monitoring Support for Resource Management and Service Assurance", IEEE Network, Vol. 18, Issue 6, Dec. 2004, pp. 6-18.

[163] A. Asgari, P. Trimintzios, M. Irons, et al "Building Quality-of-Service Monitoring Systems for Traffic Engineering and Service Management", JNSM, Vol. 11, No. 4, December 2003, pp. 399-426.

[164] A. Chourmouziadis, G. Pavlou, "Web Services Monitoring: An Initial Case Study on the Tools Perspective", Network Operations and Management Symposium (NOMS 2008), April 2008.

[165] O. Negru, "End-to-End QoS through Integrated Management of Content, Networks and Terminals: The ENTHRONE Project", 1st ACS/IEEE International Workshop on Wireless Internet Services (WISe'08), April 2008, http://uuu.enseirb.fr/~negru/ENTHRONE_paper_WISe.pdf

[166] D. Goderis, M. Buchli, Y. T'Joens, et al., "Service Level Specification Semantics and Parameters", Internet Draft-Draft-tequila-sls-02.txt, Expired Aug. 2002.

[167] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss., "An Architecture for Differentiated Services," IETF RFC 2475, Dec. 1998, http://www.ietf.org/rfc/rfc2475.txt.

[168] W3 Schools, "XPath tutorial", 1999, http://www.w3schools.com/xpath/default.asp.

[169] A. Chourmouziadis, G. Pavlou, "Efficient Web Services Event Reporting and Notifications by Task Delegation" DSOM 2007, Vol. 4785/2007, September 2007, pp. 242-255.

[170] ITU-T Recommendations X.690, "ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", July 2002, http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf.

[171] D. Megginson, "The Simple API for XML processing (SAX)", David Megginson Ltd, http://www.saxproject.org/copying.html

[172] S. Vinoski, "Web Services Notifications", IEEE Internet Computing, Vol. 8, Issue 2, April 2004, pp. 86 - 90.

[173] S. Vinoski, "More Web Services Notifications", IEEE Internet Computing, Vol. 8, Issue 3, Jun 2004, pp. 90-93.

[174] N. Catania, P. Kumar, B.Murray, H. Pourhedari, W.Vambenepe, K. Wurster, "WS-EVENTS 2.0 specification", HP technical report, July 2003, http://xml.coverpages.org/WS-Events20030721.pdf.

[175] M. Sloman, "Policy Driven management for Distributed Systems," Journal of Network and Systems Management, Vol. 2, No 4, 1994.

[176] A. Westerinen, "Terminology for Policy Based Management" IETF RFC 3198, Nov 2001, http://www.faqs.org/rfcs/rfc3198.html.

[177] E. Ort, B. Mehta, "Java API for XML Binding Architecture (JAXB)", SUN technical article, March 2003, http://java.sun.com/developer/technicalArticles/WebServices/jaxb/.

[178] A. L. Hors, P. L. Hégaret, L. Wood, "Document Object Model Level 3 Core Specification (DOM)", W3C recommendation, April 2004, http://www.w3.org/TR/2004/ REC-DOM-Level-3-Core-20040407/.

[179] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", Phd Dissertation, 2000, http://www.ics.uci.edu/~fielding/pubs/dissertation /top.htm

[180] A. Chourmouziadis, G. Pavlou, O.G. Duque, "Experiences in Using MUWS for scalable Distributed Monitoring", work under review in IM 2009.

8 Appendix

8.1 Design Phase Details of the Monitoring Tool

The management tool should enhance the work performed on our WS-based monitoring and event reporting systems. As such, the tool should act as a manager offering a high level view of the monitoring and event reporting functionality supported by the agent. As a result of this, the tool should provide a simple and user friendly interface to the end user in order for the latter to be able (a) to send monitoring requests and receive responses on these requests or in other words create and manage a number of monitoring jobs (b) to manage our event service through policies for task delegation. In general the requirements that need to be addressed by this tool are the following:

- The end user should able to define monitoring jobs to access particular data from the agent.
- The end user should be able to register for an event and also be able to receive events.
- Any messages exchanged between the monitoring tool and the agent systems (event reporting and monitoring systems) should be exchanged using SOAP. The syntax used in these messages should conform to the syntax used for the operations of our custom framework for monitoring, and to the WS notification standard for event reporting. The monitoring tool should also accept and be able to process event messages with application specific syntax. It is in our future objectives to support the messages used for the operations supported by the MUWS standard. This can happen as soon as we will make the changes proposed in chapter three to our monitoring system. These steps are important for interoperability and require a number of software changes.
- Before sending any request for event subscription or monitoring, the SOAP message should be validated by the management tool. This is necessary so that requests do not fail due to bad syntax when they reach the agent.
- The monitoring tool should depict the information and the services supported at the agent in the form of a tree. This way the manager can know what type of data are supported and what type of services he/she can use to retrieve these data.
- The end user should be able to create monitoring and event reporting jobs. The end user should be able to see the results from these jobs whether these are values on state data from monitoring or events generated from event reporting. Monitoring and event reporting results from relevant jobs should be displayed in a user friendly manner.

The management tool is a Java application that should equip an end user with a simple and user friendly interface to create monitoring jobs and to subscribe and receive notifications. Its main components are a monitoring module, a notification module, a connection module and a data and services view module. We elaborate on the design phase requirements of these modules in the next sections.

8.1.1 Data and Service View Module

This module will display details about the information that can be accessed from the WS-MIBs (data type, status i.e. accessible or not accessible data) hosted at the agent as WS interfaces. Examples of such MIBs are the traffic engineering MIBs used for all the scenarios we have examined in the previous chapters (LSR MIB, the FEC-to-NHLFE MIB and the interface group from the RFC 1213 MIB). This module will also need to display the operations offered by each WS interface. Using this module, the end user should be able to view

- ♦ What types of operations are supported by the WS hosted at the agent.
- ♦ What types of data are supported for retrieval at the agent.

All this information should be displayed in a user friendly manner. As such

- ♦ The data information and service information should be displayed in the form of a tree for easy viewing by the end user.
- ♦ A separate tree should be used to display the services supported and a separate tree should be used for the data types each WS interface supports.

All this information can help the end user in selecting the type of state data he wants to retrieve for monitoring and event subscription purposes and also the services available to use for these purposes. Eventually the data and service view module should display the relationships between WS and state data for effective retrieval of management data. A preview of the data and service view module and how it looks like can be seen in Figure 8-2. Implementation details are provided in the implementation phase section.

8.1.2 Monitoring Module

The Monitoring module should enable the end user (manager) to create a monitoring request using our custom tool queries. The monitoring module should be able to handle the responses to these requests and also display the results returned as part of a response to a request. These three requirements represent the essence of a monitoring job. For each monitoring job the following are required:

- ♦ Each job should use the operations supported by our custom framework.
- ♦ Each job should use SOAP messaging for requests and responses.

- Before sending a request to an agent, the monitoring module is responsible for validating the correct syntax of the request contained in a SOAP message (through an XML Schema).
- The monitoring tool should also check the correct syntax of the custom tool queries included in the request.
- The monitoring module should consist of two panels facilitating this process. The first panel is a create job panel. A preview of this panel is given in Figure 8-3. This panel should allow the end user to create monitoring requests as part of a series of monitoring jobs. The second panel is a display result panel where the results of each monitoring job should be displayed. A preview of this panel can be seen in Figure 8-4.
- Depending on the service, the operation, the type of information the user selects from the data and the service view module, and the queries he/she will form using the create job panel, a request should be formed such as that in Figure 8-1 containing custom tool queries.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<soapenv:Envelope
 xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xmlns:qrt=http://127.33.44.2/QueryToolSchema3">
  <soapenv:Body>
    <in0 xmlns="urn:AgentService"
    xmlns:ns1="http://xml.apache.org/xml-soap"
    xsi:type="ns1:Document">
      <request xmlns="">
        <qrt:SS_Query>{http://MplsLsrMibDoc,NULL,NULL,NULL}
        </qrt:SS_Query>
        <qrt:MID_Query>
         {mplsInSegmentInterface[],mplsOutSegmentInterface[]}
        </qrt:MID_Query>
        <qrt:FD_Query>{value==32,value==31}</qrt:FD_Query>
      </request>
    </in0>
  </soapenv:Body>
</soapenv:Envelope>
```

Figure 8-1 Monitoring request to the agent WS

This request message should be created using the functionality of the monitoring module and of the monitoring tool that should support the following features in managing monitoring jobs:

- ***Creation of Monitoring Jobs***: The main function of this feature would be to assist the end user in creating monitor jobs. Each monitoring job should be responsible for dispatching monitoring requests to the agent at regular intervals. Each monitoring job should have (a) a name (b) a method that will be invoked with it (c) a granularity period (d) a smoothing window size (for averaging) and (e) a number of custom tool queries that will be dispatched with it to retrieve state data. All this information should be provided in the create job panel.

- ***Viewing of Monitoring Jobs***: The monitoring jobs and their results should be displayed in the result panel in the form of a tree. When a user selects a job in the result panel, its result data should be displayed in the form of (a) real time table and/or (b) a real time chart. The size of the results history displayed in these two forms should be provided by the end user for each job. The user should provide the result history size as part of a parameter in the create job panel.

- ***Managing of Monitoring Jobs:*** Each job in the result panel should provide the means to manage it by the use of a pop up menu (Figure 8-5). The pop up menu should allow the following options to manage a monitoring job:

  - ***Suspend Job:*** This option should suspend the selected job. Once a monitor job is suspended, the management tool should not dispatch further monitoring requests to the agent for this job.

  - ***Resume Job:*** This option should resume only suspended monitoring jobs or otherwise do nothing. When a suspended job is resumed monitoring requests are sent every granularity period to the agent.

  - ***Delete Job:*** This option should remove a job from memory and from being executed again. This operation should also delete the parameters of a job kept in volatile storage by the monitoring tool.

  - ***Edit Job:*** This option should help the user in editing the parameters of an existing job. When a user selects this option, the create job panel should appear to amend any of the parameters included in a job. When the create job panel appears the job should be temporarily suspended. Once the edited job is saved, the job should resume with the new parameters.

- ***Serialising of Monitoring Jobs:*** When a user exits the monitoring tool, all the monitoring jobs and their parameters that are active or suspended should be stored in permanent storage. When the application comes up again, the monitoring tool should load these parameters and start dispatching monitoring requests for each job that was not in suspend mode based on its parameters. This functionality saves the end user from creating the same monitoring jobs every time the management tool is turned off and then on again.

The implementation details of the monitoring module are given in the implementation phase section. A preview of the monitoring module is given in Figure 8-6.

8.1.2 Notification Module

The Notification module should be responsible for event registration through policies using SOAP messaging. The general requirements of the notification module are:

- Provide support to the WS notification standard messages.
- Support validation of a subscription request message through an XML schema before sending it to the event service at the agent.
- Support receiving event reports, and display them in the form of a tree in an events received panel.
- Support the end user in structuring and sending subscription requests using our policy like grammar.

The notification module should support the following functionality:

- ***Event Subscription***: The main function of this feature is to assist the end user in subscribing for events with the event service deployed at the agent. For event subscription a template should be provided with our policy grammar. Before sending any subscription request, the end user's request should be validated with an XML schema. The end user should be able to (a) load the subscription template (b) save a subscription request for later use (c) load a subscription request that has been saved (d) examine the XML schema of the subscription request to understand what is the appropriate syntax for a request. All this functionality should be supported by a panel. A preview of the *notification subscription messaging panel* is given in Figure 8-7. It is in our future intensions to make the notification module for subscription more user friendly by supporting a more interactive and dynamic manner of creating a subscription request than using a template. This will allow the end user to make no mistakes when he/she is creating subscription requests.
- ***Viewing of Events Received***: The user should be able to view all the events received. A separate panel should assist in this respect. This panel will display events in the form of a tree. When the user clicks on a particular event the latter should be displayed as an HTML document in a web browser. Figure 8-8 is a preview of the *events received panel.*

Figure 8-9 is a preview of an integrated view of the notification module. It consists of the *data and service module*, the *notification subscription messaging panel* and *the events received panel.*

8.1.3 Connection Management Module

Every time the end user needs to create a subscription in order to receive notifications or send a request for data as part of a monitoring job, a connection should be created. The user should be able to specify the connection details of the agent on which he/she wants to connect. Details such as (a) the IP address of the connection (b) the port of the connection and (c) the folder path at which to connect on the web server hosting the agent should be provided / typed by the user. These details should be given using a dialog box provided by the file menu of the monitoring module. If these details are not provided, the monitoring tool should assume default values. A preview of the dialog box is given in Figure 8-10.

8.2 Implementation Phase Details of the Monitoring Tool

This section is dedicated on the implementation details and aspects of the management tool. Details will be provided in this section on a per module basis.

8.2.1 Data and Service View module

The data and service view module as mentioned previously describes the services and the methods that an agent exposes by using a tree representation. In order to acquire the service information the agent supports, the monitoring tool queries the registry of the web server hosting the agent (the Tomcat server is used). When this happens it retrieves the WSDL files of each service. From these files the monitoring tool extracts the IP at which each service is deployed, the name of the service and the methods it supports. After all this information is extracted, a tree is formed. The root node of this tree is called "*Services*". The children of this node are the names of the services supported. The leaf nodes of this tree are the methods each service supports (Figure 8-2 left).

The data and services view module also contains a tree for displaying the management information that the agent offers. The monitoring tool acquires this knowledge by inquiring a getTree method exposed by each WS interface the agent hosts. In order to populate the data tree, the end user has to click on one of the WS in the services tree (Figure 8-2 right). This automatically invokes the getTree method of that service. The "getTree" method offered by each WS is not displayed in the service tree because the user should not be able to use it. Only the internal processes of the monitoring module are allowed to use this method. If the getTree method of the agent is invoked (the agent is also a WS) a view of the relationships between WS and state data can be acquired. This can facilitate as shown in chapter three, the effective retrieval of management data (in a bulk manner). For other WS except the agent each node in the data tree, represents information the agent can retrieve from managed devices and also specifies whether it is accessible data or not. This is necessary for SNMP information models because in an SNMP information tree only leaf nodes provide accessible nodes. Leaf nodes of this tree represent single instance objects. Other nodes represent multiple instance objects (Figure 8-2 right).

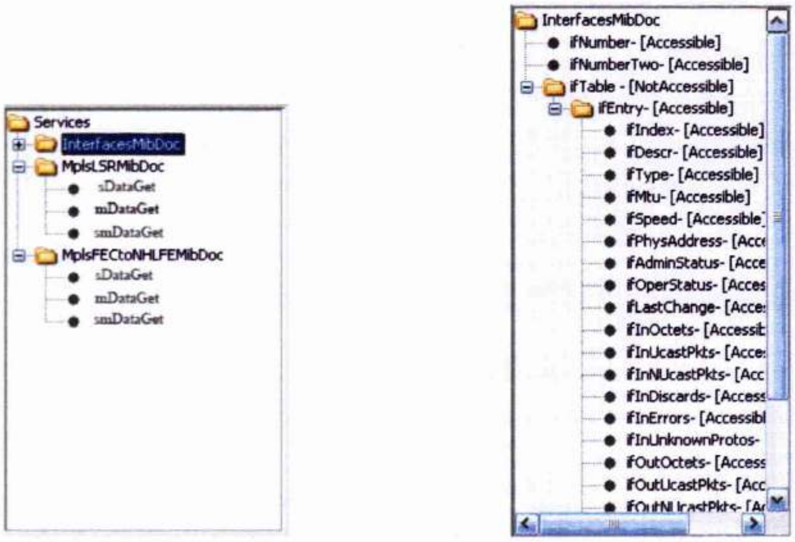


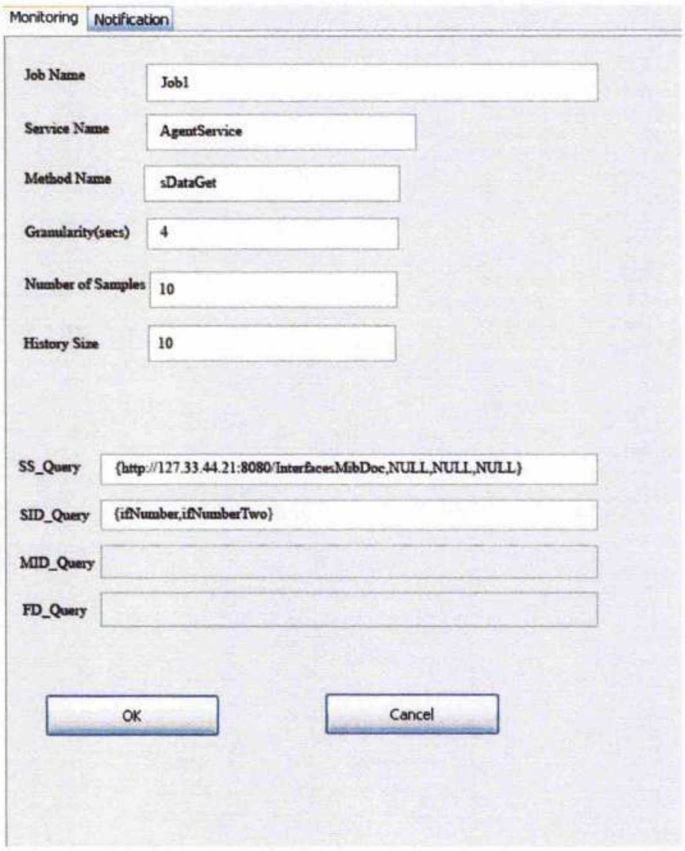
Figure 8-2 Data and Service View module

8.2.2 Monitoring Module

This module is responsible for providing the user with facilities for creating, managing and serialising monitoring jobs.

Creating Monitor Jobs: This facility is provided by the create job panel. This panel is displayed in Figure 8-3. In this panel the user is allowed to

- Enter a name for each job.
- Type or select the service to invoke.
- Type or select the operation name to be invoked.
- Type the queries which he/she is interested in enquiring.
- Select the granularity period for each job.
- Select the smoothing window for applying the Weighted Moving Average algorithm for averaging the displayed results (Figure 8-3). It is in our goals in the future to provide a series of other functions apart from the Weighted Moving Average (WMA) function. The user would be able to select among these functions to enforce statistical operations on the data results collected as part of a monitoring job. Currently the user can use implicitly the WMA function when he/she selects a window size other than 1 or 0. If the user selects the window size to be 1 the result of the subtraction between two consecutive counter values will be displayed. If the user selects the window size to be 0, the actual counter value collected is displayed.



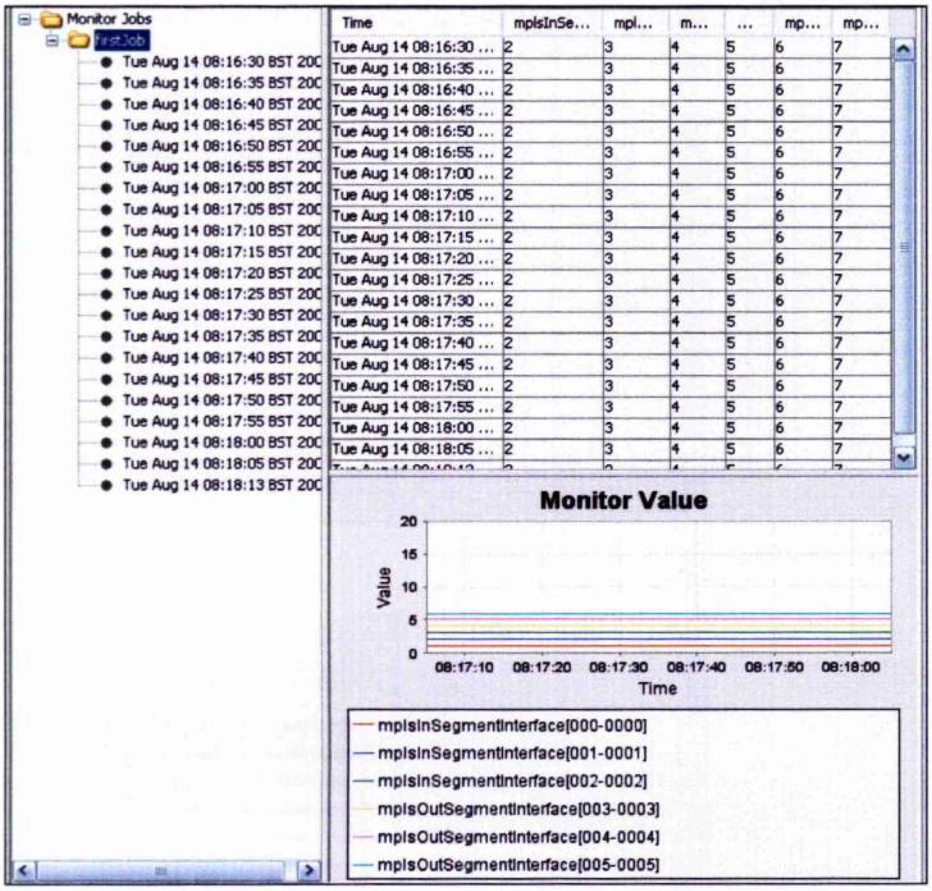
Job Name	Job1
Service Name	AgentService
Method Name	sDataGet
Granularity(secs)	4
Number of Samples	10
History Size	10
SS_Query	{http://127.33.44.21:8080/InterfacesMibDoc,NULL,NULL,NULL}
SID_Query	{ifNumber,ifNumberTwo}
MID_Query	
FD_Query	

Figure 8-3 : The Create Job Panel of the monitoring module

Viewing Monitors Jobs: Monitoring jobs are displayed in the form of a tree in the result panel (Figure 8-4). The root node of the tree is a node titled "*Monitor jobs*". Monitoring jobs are displayed as children of this node. The results of each monitoring job are added as children of a monitoring job node (results history). The user can view the result history of a monitoring job in the following forms.

- **Real time table:** This table displays the time that a result was collected and a value for the result. The table gets updated as and when new monitoring job results arrive depending on the granularity period. The maximum number of rows in this table is equal to the volume of the result history.

- **Real time Chart:** The user can select to view the result history in a time series chart. This chart is displayed under the real time table. This chart can help the end user in studying the behaviour of certain network counters.


- Monitor Jobs
  - FirstJob
    - Tue Aug 14 08:16:30 BST 200
    - Tue Aug 14 08:16:35 BST 200
    - Tue Aug 14 08:16:40 BST 200
    - Tue Aug 14 08:16:45 BST 200
    - Tue Aug 14 08:16:50 BST 200
    - Tue Aug 14 08:16:55 BST 200
    - Tue Aug 14 08:17:00 BST 200
    - Tue Aug 14 08:17:05 BST 200
    - Tue Aug 14 08:17:10 BST 200
    - Tue Aug 14 08:17:15 BST 200
    - Tue Aug 14 08:17:20 BST 200
    - Tue Aug 14 08:17:25 BST 200
    - Tue Aug 14 08:17:30 BST 200
    - Tue Aug 14 08:17:35 BST 200
    - Tue Aug 14 08:17:40 BST 200
    - Tue Aug 14 08:17:45 BST 200
    - Tue Aug 14 08:17:50 BST 200
    - Tue Aug 14 08:17:55 BST 200
    - Tue Aug 14 08:18:00 BST 200
    - Tue Aug 14 08:18:05 BST 200
    - Tue Aug 14 08:18:13 BST 200

Time	mplsInSe...	mpl...	m...	...	mp...	mp...
Tue Aug 14 08:16:30 ...	2	3	4	5	6	7
Tue Aug 14 08:16:35 ...	2	3	4	5	6	7
Tue Aug 14 08:16:40 ...	2	3	4	5	6	7
Tue Aug 14 08:16:45 ...	2	3	4	5	6	7
Tue Aug 14 08:16:50 ...	2	3	4	5	6	7
Tue Aug 14 08:16:55 ...	2	3	4	5	6	7
Tue Aug 14 08:17:00 ...	2	3	4	5	6	7
Tue Aug 14 08:17:05 ...	2	3	4	5	6	7
Tue Aug 14 08:17:10 ...	2	3	4	5	6	7
Tue Aug 14 08:17:15 ...	2	3	4	5	6	7
Tue Aug 14 08:17:20 ...	2	3	4	5	6	7
Tue Aug 14 08:17:25 ...	2	3	4	5	6	7
Tue Aug 14 08:17:30 ...	2	3	4	5	6	7
Tue Aug 14 08:17:35 ...	2	3	4	5	6	7
Tue Aug 14 08:17:40 ...	2	3	4	5	6	7
Tue Aug 14 08:17:45 ...	2	3	4	5	6	7
Tue Aug 14 08:17:50 ...	2	3	4	5	6	7
Tue Aug 14 08:17:55 ...	2	3	4	5	6	7
Tue Aug 14 08:18:00 ...	2	3	4	5	6	7
Tue Aug 14 08:18:05 ...	2	3	4	5	6	7

Monitor Value
20
15
10
5
0
Value
08:17:10
08:17:20
08:17:30
08:17:40
08:17:50
08:18:00
Time
mplsInSegmentInterface[000-0000]
mplsInSegmentInterface[001-0001]
mplsInSegmentInterface[002-0002]
mplsOutSegmentInterface[003-0003]
mplsOutSegmentInterface[004-0004]
mplsOutSegmentInterface[005-0005]

Figure 8-4 : The Result Panel of the monitoring module

Manage Monitoring Jobs: In order for the end user to manage monitoring jobs he/she should right click on a monitoring job in the result panel for a pop menu to be displayed. The pop up menu provides four options which are the following:

- *Suspend Job:* When a user selects the *suspend* option from the popup menu, the Timer Task of the selected job is cancelled by calling the cancel method of a TimeTask object (job) explicitly from the main class of the monitoring tool. This kills the job immediately terminating even a request from this job that is in progress. The job parameters are still kept in memory (inside a job vector maintained by the monitoring tool) in case the suspended job is resumed later but requests are not sent to the agent.

- *Resume Job:* If and only if a job is previously suspended by the user, selecting this option will cause the monitoring tool to restart this job and start sending monitoring request messages to the agent. In order for this to happen the monitoring tool maintains a vector containing all the parameters associated with each job. When the end user selects a job to be resumed a new TimerTask object is created for this job and is reinserted back in the timer object.

- *Delete Job:* This option deletes a job from memory. This necessitates that the TimerTask object parameters associated with this job should be extracted from the job vector (volatile storage of the parameters of each monitoring job). Also the TimerTask associated with this job is cancelled. Then the purge method for this task is called for the timer object by the main class of the monitoring tool so that the garbage collector reclaims its memory.
- *Edit Job:* This option loads the parameters of each job on the create job monitoring panel by accessing these parameters from the job vector. The user is enabled to change any parameter of a job. While this process occurs, the TimerTask object associated with this job is retracted from the timer object. When the user presses the OK button in the create job panel a new TimerTask is created for this job and is inserted back in the Timer object.

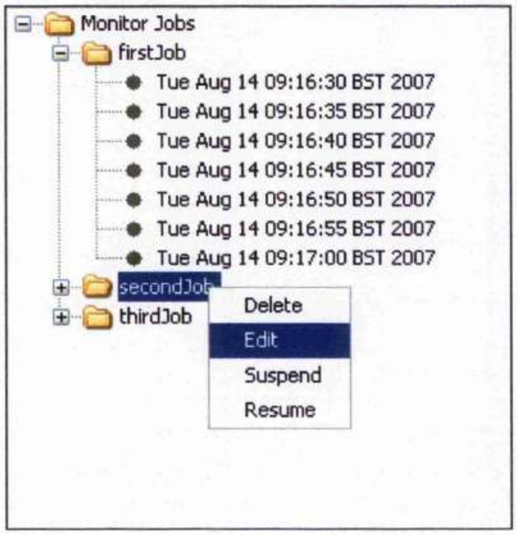
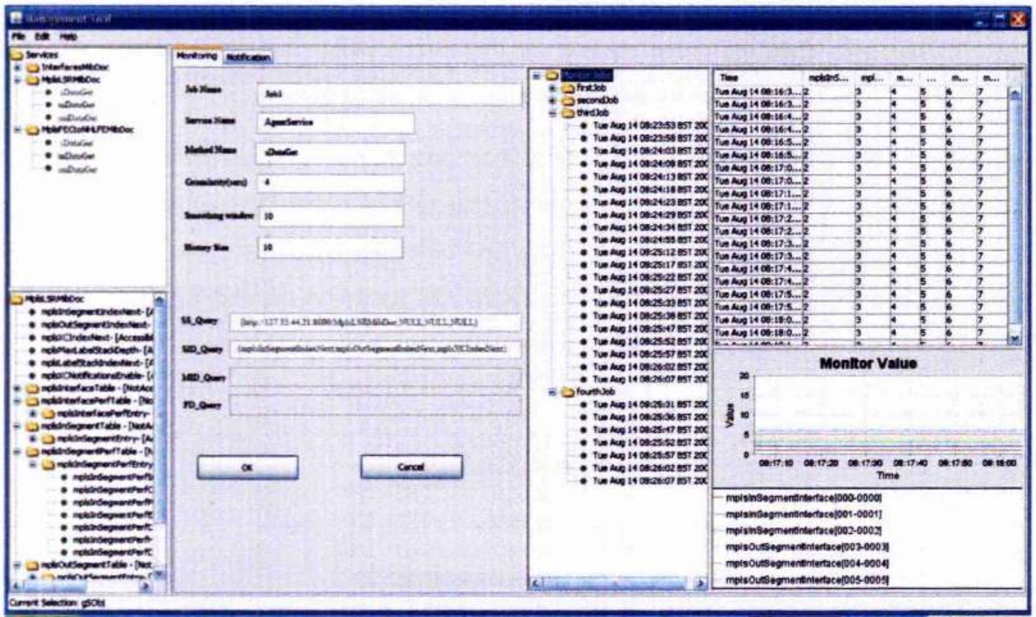

Monitor Jobs
firstJob
Tue Aug 14 09:16:30 BST 2007
Tue Aug 14 09:16:35 BST 2007
Tue Aug 14 09:16:40 BST 2007
Tue Aug 14 09:16:45 BST 2007
Tue Aug 14 09:16:50 BST 2007
Tue Aug 14 09:16:55 BST 2007
Tue Aug 14 09:17:00 BST 2007
secondJob
thirdJob
Delete
Edit
Suspend
Resume

Figure 8-5 : Pop Up Menu to manage monitoring jobs

Serialising Monitor Jobs: Before the end user shuts down the management tool, the parameters of each monitoring job are saved in persistent storage. Each job and the TimerTask object associated with it are serialised in persistent storage by implementing the java.io.Serializable interface. Object serialisation through this API allows us to take an object's state and convert it into a stream of data for persistent storage. The persistent storage of the Serializable API is a file on the hard disk. When the management tool is started, the serialised jobs are extracted from this file and are restored in memory. Then the monitoring process for each job resumes, as the TimerTask object for each job is reinserted back in the Timer object.

Figure 8-6 shows the integrated view of the monitoring module. As seen from the figure, there are four panels in the monitoring module. The leftmost panel is the data and services module model view panel. The second panel is the create job panel also enabling us to edit the parameters of a job. The third panel is the result panel. Inside the result panel there is another one allowing the user to display results in the form of a chart or table.



Time	mplsInS...	mpl...	m...	...	m...	m...
Tue Aug 14 08:16:3...	2	3	4	5	6	7
Tue Aug 14 08:16:3...	2	3	4	5	6	7
Tue Aug 14 08:16:4...	2	3	4	5	6	7
Tue Aug 14 08:16:4...	2	3	4	5	6	7
Tue Aug 14 08:16:5...	2	3	4	5	6	7
Tue Aug 14 08:16:5...	2	3	4	5	6	7
Tue Aug 14 08:17:0...	2	3	4	5	6	7
Tue Aug 14 08:17:0...	2	3	4	5	6	7
Tue Aug 14 08:17:1...	2	3	4	5	6	7
Tue Aug 14 08:17:1...	2	3	4	5	6	7
Tue Aug 14 08:17:2...	2	3	4	5	6	7
Tue Aug 14 08:17:2...	2	3	4	5	6	7
Tue Aug 14 08:17:3...	2	3	4	5	6	7
Tue Aug 14 08:17:3...	2	3	4	5	6	7
Tue Aug 14 08:17:4...	2	3	4	5	6	7
Tue Aug 14 08:17:4...	2	3	4	5	6	7
Tue Aug 14 08:17:5...	2	3	4	5	6	7
Tue Aug 14 08:17:5...	2	3	4	5	6	7
Tue Aug 14 08:18:0...	2	3	4	5	6	7
Tue Aug 14 08:18:0...	2	3	4	5	6	7

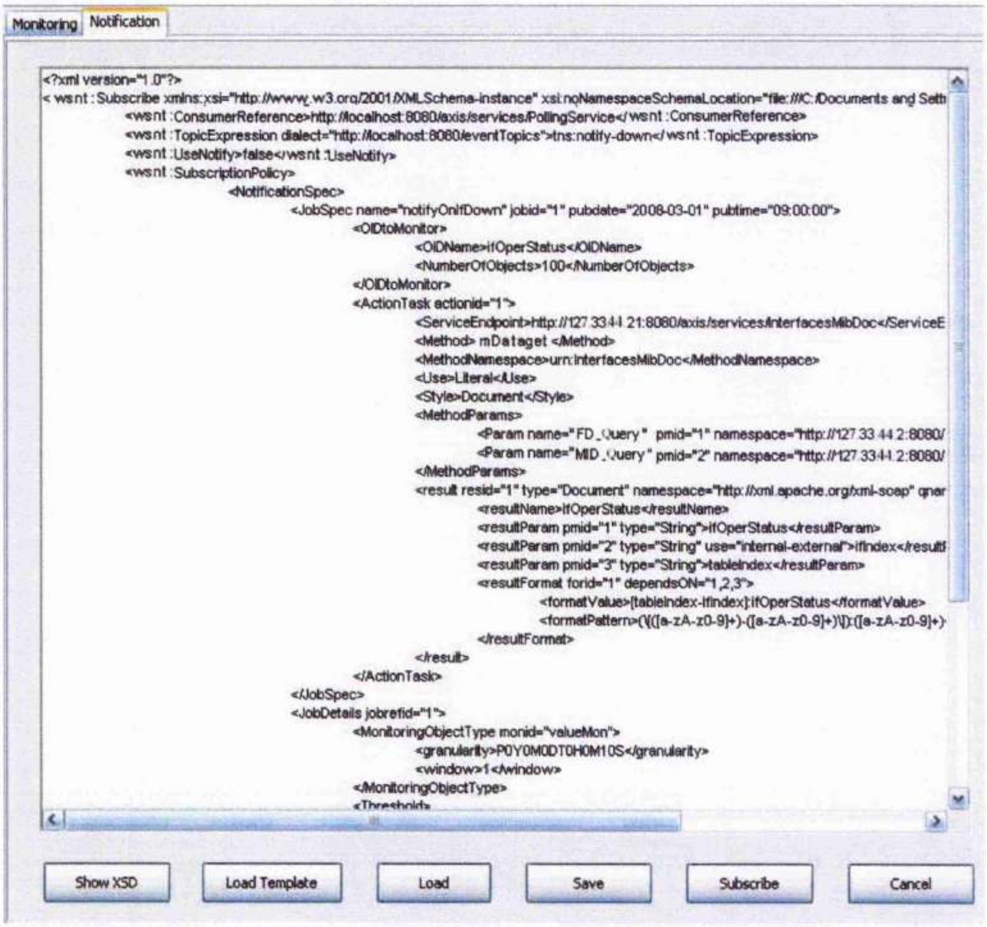
mplsInSegmentInterface[000-0000]
mplsInSegmentInterface[001-0001]
mplsInSegmentInterface[002-0002]
mplsOutSegmentInterface[003-0003]
mplsOutSegmentInterface[004-0004]
mplsOutSegmentInterface[005-0005]

Figure 8-6 : Integrated view of the monitoring module of the management tool

8.2.3 Notification Module

The notification module enables an end user to subscribe for events and receive events from our event service using WS Notification standard messages. This module supports the following functionality:

Event Subscription: The user is provided with a template for creating a new WS subscription message in order to start receiving events (Figure 8-7 Load Template button). This template provides information to the user on how to use each XML element in the subscription message even for the elements of the policy like document. The user is given the option to write a new subscription message using the template or is given the option of an empty message. The user can also save a subscription request message in memory (Figure 8-7 Save button) and recall this message at a latter time (Figure 8-7 Load Button). The user can also see the XML schema of the subscription message to understand what types of options are allowed in every element of the document template (Figure 8-7 "*Open XSD*" button). Initially this module was kept simple. In the future, more user friendly features will be added so that a more interactive and dynamic creation of a subscription request than using a template will be supported.



```
<?xml version="1.0"?>
< wsnt : Subscribe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="file:///C:/Documents and Setti
        <wsnt :ConsumerReference>http://localhost:8080/axis/services/PollingService</wsnt :ConsumerReference>
        <wsnt :TopicExpression dialect="http://localhost:8080/eventTopics">tns:notify-down</wsnt :TopicExpression>
        <wsnt :UseNotify>false</wsnt :UseNotify>
        <wsnt :SubscriptionPolicy>
                <NotificationSpec>
                        <JobSpec name="notifyOnIfDown" jobid="1" pubdate="2008-03-01" pubtime="09:00:00">
                                <OIDtoMonitor>
                                        <OIDName>ifOperStatus</OIDName>
                                        <NumberOfObjects>100</NumberOfObjects>
                                </OIDtoMonitor>
                                <ActionTask actionid="1">
                                        <ServiceEndpoint>http://127.33.44.21:8080/axis/services/InterfacesMibDoc</ServiceE
                                        <Method> mDataget </Method>
                                        <MethodNamespace>urn:InterfacesMibDoc</MethodNamespace>
                                        <Use>Literal</Use>
                                        <Style>Document</Style>
                                        <MethodParams>
                                                <Param name="FD_Query"  pmid="1" namespace="http://127.33.44.2:8080/
                                                <Param name="MID_Query" pmid="2" namespace="http://127.33.44.2:8080/
                                        </MethodParams>
                                        <result resid="1" type="Document" namespace="http://xml.apache.org/xml-soap" qnar
                                                <resultName>ifOperStatus</resultName>
                                                <resultParam pmid="1" type="String">ifOperStatus</resultParam>
                                                <resultParam pmid="2" type="String" use="internal-external">ifIndex</resultF
                                                <resultParam pmid="3" type="String">tableIndex</resultParam>
                                                <resultFormat forid="1" dependsON="1,2,3">
                                                        <formatValue>[tableIndex-ifIndex]:ifOperStatus</formatValue>
                                                        <formatPattern>(\[([a-zA-z0-9]+)-([a-zA-z0-9]+)\]):([a-zA-z0-9]+)
                                                </resultFormat>
                                        </result>
                                </ActionTask>
                        </JobSpec>
                        <JobDetails jobrefid="1">
                                <MonitoringObjectType monid="valueMon">
                                        <granularity>P0Y0M0DT0H0M10S</granularity>
                                        <window>1</window>
                                </MonitoringObjectType>
                                <Threshold>
```

Figure 8-7 : Notification subscription messaging panel

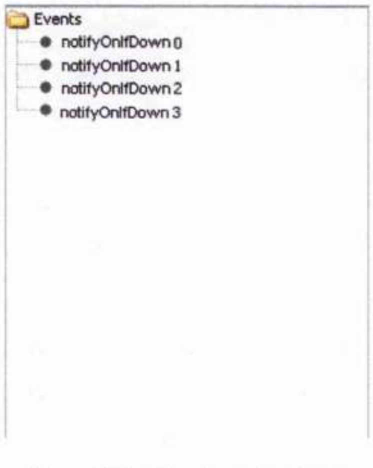
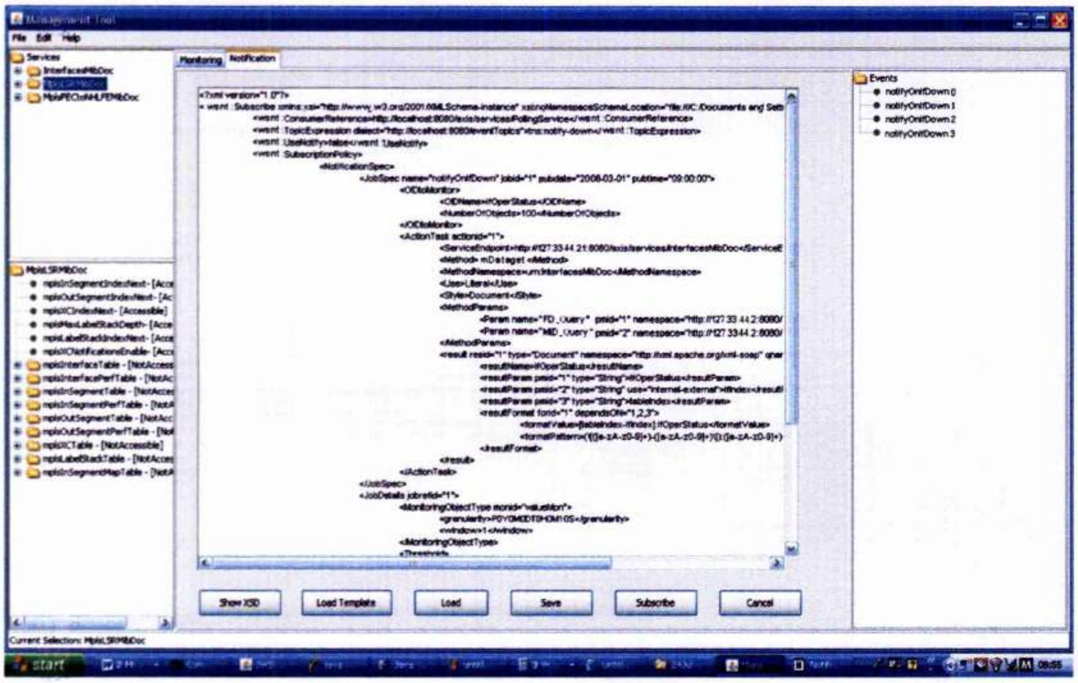


Figure 8-8 : Event received tree

View Events Received: The events received by the event service at the agent are displayed in the form of a tree as that in Figure 8-8. The root node of this tree is called "*Events*". When the monitoring tool receives an event it looks into it in order to find the event name/topic. In Figure 8-8 we can see four events produced for the third QoS scenario in chapter four. The user can view the particular details of an event by clicking on its corresponding node in the event tree. This will trigger the default web browser to open and the details of the event to be displayed as an HTML document.

Figure 8-9 shows the integrated view of the notification module. The notification nodule consists of three panels. The leftmost panel is the data and service module. It displays information about the services and the state data exposed at the agent. The middle panel is the notification subscription messaging panel. The rightmost panel displays the events received tree.


Management Tool
File Edit Help
Services
InterfacesMibDoc
MplsLSRMibDoc
MplsFECtoNHLFEMibDoc
MplsLSRMibDoc
mplsInSegmentIndexNext- [Acc
mplsOutSegmentIndexNext- [Ac
mplsXCIndexNext- [Accessible]
mplsMaxLabelStackDepth- [Acce
mplsLabelStackIndexNext- [Acce
mplsXCNotificationsEnable- [Acc
mplsInterfaceTable - [NotAccess
mplsInterfacePerfTable - [NotAc
mplsInSegmentTable - [NotAcce
mplsInSegmentPerfTable - [NotA
mplsOutSegmentTable - [NotAcc
mplsOutSegmentPerfTable - [No
mplsXCTable - [NotAccessible]
mplsLabelStackTable - [NotAcce
mplsInSegmentMapTable - [NotA
Current Selection: MplsLSRMibDoc
Monitoring
Notification
Events
notifyOnIfDown 0
notifyOnIfDown 1
notifyOnIfDown 2
notifyOnIfDown 3
Show XSD
Load Template
Load
Save
Subscribe
Cancel
start

```
<?xml version="1.0"?>
- wsnt :Subscribe xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="file:///C:/Documents and Setti
        <wsnt :ConsumerReference>http://localhost:8080/axis/services/PollingService</wsnt :ConsumerReference>
        <wsnt :TopicExpression dialect="http://localhost:8080/eventTopics">tns:notify-down</wsnt :TopicExpression>
        <wsnt :UseNotify>false</wsnt :UseNotify>
        <wsnt :SubscriptionPolicy>
                <NotificationSpec>
                        <JobSpec name="notifyOnIfDown" jobid="1" subdate="2008-03-01" subtime="09:00:00">
                                <OIDtoMonitor>
                                        <OIDName>ifOperStatus</OIDName>
                                        <NumberOfObjects>100</NumberOfObjects>
                                </OIDtoMonitor>
                                <ActionTask actionid="1">
                                        <ServiceEndpoint>http://127.33.44.21:8080/axis/services/InterfacesMibDoc</ServiceE
                                        <Method> mDataget </Method>
                                        <MethodNamespace>urn:InterfacesMibDoc</MethodNamespace>
                                        <Use>Literal</Use>
                                        <Style>Document</Style>
                                        <MethodParams>
                                                <Param name="FD_Query" pmid="1" namespace="http://127.33.44.2:8080/
                                                <Param name="MD_Query" pmid="2" namespace="http://127.33.44.2:8080/
                                        </MethodParams>
                                        <result resid="1" type="Document" namespace="http://xml.apache.org/xml-soap" qnar
                                                <resultName>ifOperStatus</resultName>
                                                <resultParam pmid="1" type="String">ifOperStatus</resultParam>
                                                <resultParam pmid="2" type="String" use="internal-external">ifIndex</resultP
                                                <resultParam pmid="3" type="String">tableIndex</resultParam>
                                                <resultFormat forid="1" dependsOn="1,2,3">
                                                        <formatValue>[tableIndex-ifIndex]:ifOperStatus</formatValue>
                                                        <formatPattern>(\[([a-zA-z0-9]+)-([a-zA-z0-9]+)\]):([a-zA-z0-9]+)
                                                </resultFormat>
                                        </result>
                                </ActionTask>
                        </JobSpec>
                        <JobDetails jobrefid="1">
                                <MonitoringObjectType monid="valueMon">
                                        <granularity>P0Y0M0DT0H0M10S</granularity>
                                        <window>1</window>
                                </MonitoringObjectType>
                                <Thresholds>
```

Figure 8-9 : Integrated view of the management notification module

8.2.4 Connection Management module

Every time the end user needs to create a subscription to receive notifications or to add a monitoring job, a connection needs to be created. The user can provide details about the connection using the connection management module. The user can access this module by clicking the "*File->New*" Menu option. By selecting this option a dialog box appears enabling the user to specify details about the connection. The user can enter (a) the name of the connection (b) the IP address of the connection (c) the port number of the connection and (d) the path to the web server where the agent and the services such as the event service are hosted. If the user does not specify connection details the default options are used. The default address, port and folder path of the agent, are localhost, 8080 and "/axis/services". The connection management module dialog box is given in Figure 8-10.

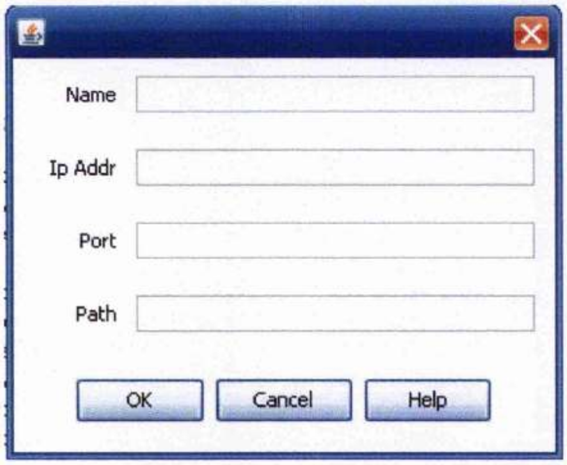


Figure 8-10 : Connection Dialog Box