Development and Evaluation of Web Services based Self-Organizing Management Functionality in Transport Networks

PREMANAND THANGAMANI

Master of Science Thesis

Supervisors:
Catalin Meirosu, Ericsson Research, Stockholm, Sweden.
Andreas Johnsson, Ericsson Research, Stockholm, Sweden.

Examiner:
Prof. Mihhail Matskin, The Royal Institute of Technology (KTH), Stockholm, Sweden.
Abstract

The management of Ethernet connectivity services is a labor-intensive task for network operators. The manual approach to managing such services allows for little flexibility and adaptability to the network traffic conditions. This thesis attempts to automate this manual process by bringing in a middle-ware i.e. a management module, which will reduce the burden on the human operator thereby providing an efficient way of managing the Ethernet service. This thesis also explores the possibility of using web services to implement this management module.

Concepts of IBM's Autonomic Network Management System, MPLS protocol, GMPLS architecture, SNMP protocol were studied. Web service standards like OASIS' MUWS and DMTF's WS-Management were studied. The proposed system contains a management module that interacts with the client and the underlying In-network functionality. WS-Management standard was chosen to provide the web service interface for the management module. Clients interact with the management module through this web service interface. Wiseman framework which provides an implementation of the WS-Management standard was used for the implementation.

The prototype developed successfully, met all the mandatory requirements for the system. The usage of Wiseman framework also provides the option to define service specific custom operations. JAXB and XML provided an easy object oriented manipulation of the Ethernet service that was represented as XML schema.
Acknowledgements

I hereby thank everyone who helped me to carry out this Master Thesis successfully.

First of all, I would like to thank Mr. Catalin Meirosu, Ph.D and Mr. Andreas Johnsson, Ph.D for providing this great opportunity to do my Master Thesis at Ericsson Research and guiding me through out. I am thankful to both Catalin and Andreas for patiently clarifying my technical doubts, providing valuable inputs whenever necessary and sharing with me their valuable knowledge about Ethernet services, GMPLS, Network Management Systems and several other telecom technologies.

I also thank Mr. Sonny Thorelli, former Manager for Packet Technologies Research department, for his excellent support and motivation right from the beginning.

I express my gratitude to my KTH supervisor Prof. Mihhail Matskin, Ph.D for providing valuable feedbacks to improvise my thesis.

I hereby express my warmest gratitude to Mr. Denis Rachal, one of the developers of the Wiseman Framework. When the light seemed to be fading away, it was Denis‘ excellent and timely support that showed me the way out of the maze and helped me to progress with the prototype development. I feel very much indebted to Denis without whom this thesis would have taken much more time to complete.

Furthermore, I also would like to thank Mrs. Alisa Devlic for her great support and motivation throughout the course of the thesis.

I would also like to thank Shankar Gopinath, Shadid Rashid Chowdhury, Victor Duran, Afshin Amighi and several other friends for guiding me throughout my studies at KTH.

I also thank Mr. Bob Melander for providing me a valuable opportunity to design an interface between our MeSON system and their WIND system.

Finally, I would like to thank Che Zhong, Rajveer Singh, Wolfgang John, Conny Larsson, Eric Svensson, Mustahsan Nawaz, Umair Aslam, Eduardo Medeiros and Ourania Lympouridou for making my stay at Ericsson Research a very memorable one.
Table of Contents

Abstract ........................................................................................................ 2
Acknowledgements ........................................................................................................ 3
Table of Contents ................................................................................................. 4
List of Figures ....................................................................................................... 6
1. Introduction ....................................................................................................... 8
   1.1 Research Questions .................................................................................. 9
2. State of the Art Survey .................................................................................... 10
   2.1 Introduction .......................................................................................... 10
   2.2 Background ........................................................................................ 10
      2.2.1 Web Services ................................................................................ 10
      2.2.2 Comparison of SNMP and Web Services in network management .... 11
      2.2.3 Organization for the Advancement of Structured Information Standards (OASIS) ................................................................................................ 11
      2.2.4 Distributed Management Task Force (DMTF) .................................. 16
      2.2.5 Performance comparison of MUWS and WS-Management ............. 18
   2.3 Related Work ...................................................................................... 19
      2.3.1 Autonomic Network Management System ......................................... 19
      2.3.2 Multi-Protocol Label Switching ......................................................... 21
      2.3.3 Simple Network Management Protocol ............................................. 21
      2.3.4 Ethernet services ........................................................................... 22
      2.3.5 RSVP-TE ....................................................................................... 22
   2.4 Previous Master Thesis Work ................................................................. 23
   2.5 Summary ............................................................................................ 23
3. System Requirements ....................................................................................... 24
   3.1 System Architecture ............................................................................... 24
   3.2 Support for Create Read Update and Delete (CRUD) operations ............... 25
   3.3 Handling one request at a time ................................................................ 25
   3.4 Asynchronous communication .................................................................. 25
   3.5 Prioritization of Requests ....................................................................... 25
   3.6 Support for subscription and notification .................................................. 26
4. Proposed System Architecture ......................................................................... 27
   4.1 Internal Architecture of SON .................................................................. 27
   4.2 Internal Architecture of SON Engine ....................................................... 29
   4.3 Architecture to handle one request at a time .............................................. 29
5. Interface Definition .......................................................................................... 32
   5.1 Producer Interface ............................................................................... 33
      5.1.1 The sendRequest () method ............................................................ 33
      5.1.2 The putRequest () method .............................................................. 34
   5.2 Client Interface ...................................................................................... 34
6. System Implementation ...................................................................................... 36
   6.1 Why WS-Management over MUWS? ....................................................... 36
   6.2 The Wiseman Framework ........................................................................ 37
      6.2.1 Wiseman Conceptual Architecture .................................................... 37
   6.3 Implementation Architecture .................................................................. 39
   6.4 The Prototype ...................................................................................... 40
1. Introduction

Ethernet services are provided by several vendors over various transport technologies and protocols like MPLS, SONET, DWDM etc. These Ethernet services are defined by Metro Ethernet Forum (MEF). The Metro Ethernet Forum (MEF) [21], founded in 2001, is a nonprofit international industry consortium, dedicated to worldwide adoption of Carrier Ethernet networks and services. Currently MEF has defined 2 service types. They are Ethernet Line (E-Line) service and Ethernet LAN (E-LAN) service type. E-Line service type is concerned with point-to-point service and E-LAN is concerned with multi point-to-multi point service. In this thesis we are concerned about the E-Line services provided over the MPLS protocol.

Traditionally these Ethernet services were managed manually. The Ethernet Service Providers (SP) and the Service Consumers (SC) agree upon a set of specifications called Service Level Agreements (SLA)s in order to provide a service of good quality. The SP needs to ensure that these SLAs are delivered according to specifications at all times. Given the dynamics of the network traffic and connections, it is difficult for the human operators to manage these networks in order to maintain the SLA.

Functionality that automates network management is needed to target the increasing complexity of provisioning and commissioning connectivity services over transport networks (e.g. Ethernet services over MPLS). The aim is to simplify and automate the fulfillment and assurance of Ethernet connectivity services through a middle-ware. By interacting with a high-level interface and an intelligent management module, the operator conducts these tasks without requiring detailed knowledge of the connectivity service itself or the underlying transport network or Operations and Management (OaM) functionality.

Also large networks are composed of nodes from different vendors. This raises the compatibility issue not only between the nodes but also between the nodes and their network management application. Traditionally, it was IETF’s Simple Network Management Protocol (SNMP) that is most widely used for network management. Opportunities to use web services for network management were explored by Moura et al [15], Anedda and Atzori [7], Pras and Martin-Flatin [9] and Verdi et al [8].

The task for this thesis is to investigate the use of web services for automating the network management functionality in transport networks.
1.1 Research Questions

The following list provides the research questions put forth before the start of the thesis.

1. What kind of web services (SOAP or RESTful) is preferred to be used for managing Ethernet services?

2. How could we use standards like OASIS’ MUWS and DMTF’s WS-Management in managing Ethernet services?

3. How could we make our solution to be used by other systems in a platform independent way?

4. How could we automate most of the manual tasks in managing the Ethernet services?

The four research questions are addressed as part of this thesis.
2. State of the Art Survey

2.1 Introduction

This section reviews literature from academia, standard bodies and industry that are relevant and related to the research questions addressed in this thesis. Under the Background section, information about web services standards like MUWS and WS-Management and organizations like DMTF and OASIS are discussed. Under the Related Work section, Autonomic Network Management System, Multi-Protocol Label Switching (MPLS), Ethernet Services, Resource Reservation Protocol for Traffic Engineering (RSVP-TE), Simple Network Management Protocol (SNMP) and other related topics are discussed.

2.2 Background

2.2.1 Web Services

Making an attempt to answer the first and third research question, we have made a study on web services in general, SOAP based and RESTfull web services.

The W3C Working Group has defined Web Services [22] as follows

"A Web service is a software system designed to support inter operable machine to machine interaction over a network. It has an interface described in a machine process able format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."

Web Services through the usage of XML based standards, allows loose coupling of distributed components. This ensures the easy integration of applications built in different environments. There are two types of web services. SOAP based and RESTfull based web services.

**SOAP-based web services:** These web services uses an XML based protocol called Simple Object Access Protocol (SOAP). The communication messages in the form of XML are sent over the HTTP. There exists a wide range of extensions to SOAP like WS-Security, WS-Transaction etc.
RESTful web services: Representational State Transfer (REST) web services use principally HTTP for communication. It could also use any other application layer protocol for transfer of the state of the resource being managed. It is lightweight compared to SOAP.

With this web service standard being accepted world-wide, web service promises to provide compatibility for the interactions between Network Management System (NMS) and the network elements that are built on different platforms. Thus web services can provide us a solution that can be used by other systems in a platform independent way thereby answering our third research question. Standardization organizations like OASIS and DMTF have explored opportunities to use web services for IT resources management. They are discussed in the forthcoming sections.

2.2.2 Comparison of SNMP and Web Services in network management

In [9], a performance comparison done in 2004 at University of Twente, between SNMP and Web Services is discussed. Performance parameters used were bandwidth usage, CPU time, memory usage and round-trip time.

Regardless of the number of objects being retrieved, SNMP outperforms Web Services. But, in case of compressed messages, when the number of objects retrieved is more than 70, Web services perform better than SNMP.

2.2.3 Organization for the Advancement of Structured Information Standards (OASIS)

Organization for the Advancement of Structured Information Standards (OASIS) not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society [18]. It was founded in 1993 as SGMLOpen. SGML stands for Standard Generalized Markup Language. Initially it was concentrating in developing guidelines for interoperability amongst SGML based products. Due to the expansion in its scope to support more technical work, it was later renamed to OASIS. OASIS has several technical committees like OASIS Provisioning Services TC, OASIS Solution Deployment Descriptor (SDD) TC, OASIS Web Services Transaction (WS-TX) TC, OASIS Web Services Distributed Management (WSDM) TC, OASIS Web Services Business Process Execution Language (WS-BPEL) TC etc. These technical committees, through industry wide consensus have established several technical standards in
areas like Cloud Computing, Service Oriented Architecture, Web Services, Security, Emergency Management etc.

Some of the standards established by OASIS are [18]
1. Service Provisioning Markup Language (SPML)
2. Security Assertion Markup Language (SAML)
3. Universal Description Discovery and Integration (UDDI)
4. Web Services Distributed Management (WSDM) - Management Using Web Services (MUWS)
5. Web Services Distributed Management (WSDM) - Management Of Web Services (MOWS) and several other standards.

The standard that we will be exploring more for this thesis will be WSDM-MUWS.

2.2.3.1 WSDM-Management Using Web Services (MUWS)

MUWS is a specification that defines the management of IT resources using web services technologies. Some of the manageability functions that are provided by MUWS are Monitoring the Quality of Service (QoS), Enforcing a Service Level Agreement (SLA), Managing a resource life cycle etc. MUWS uses a set of manageability capabilities to express the attributes, operations and events associated with any given resource. Some of the capabilities are Identity, Manageability characteristics, Correlatable properties, State, Operational Status, Metrics etc.

2.2.3.1.1 Definitions for some MUWS terminologies

Before going into the details of the MUWS architecture, it is important to provide the definitions of some of the terminologies in the MUWS standard.

Manageable Resource: As said in [12] a resource capable of supporting one or more standard manageability capabilities.

Web Service Endpoint: As said in [12] an entity providing a destination for Web service messages. A Web service endpoint has an address (URL) and is described by the content of a WSDL 1.1 port element. This definition is consistent with the definition provided in the WS-Addressing specification [WS-Addressing].
**Capability:** As said in [12] a group of properties, operations, events and meta-data associated with identifiable semantics and information and exhibiting specific behaviors.

**Manageability:** As said in [12] the ability to manage a resource, or the ability of a resource to be managed.

**Manageability Capability:** As said in [12] a capability associated with one or more management domains. This capability is considered to be a resource property.

**Manageability Consumer:** As said in [12] a user of manageability capabilities associated with one or more manageable resources.

### 2.2.3.1.2 MUWS Architecture

To have a better understanding about MUWS, it is important to understand its basic architecture [12] as shown below

![MUWS Basic Architecture](image)

The IT resource to be managed is defined with its attributes and operations as a web service. An End-Point Reference (EPR) to access this web service is defined as the Web service end point. The EPR confirms to the WS-Addressing specification. The consumer/client that wants to manage this IT
resource discovers the web service endpoint in one of the following two ways.

a) Discovery using Relationships
b) Discovery using Registries

From the discovered web service endpoint, the client could take the EPR that has the URI to access the web service. Using this EPR, the client can now send management requests, subscriptions etc and receive the corresponding responses back. MUWS mandates that both the Manageability Consumer and Manageability Resource should understand the messages communicated between them.

2.2.3.1.3 Resource Representation

Resources are represented as models. The reference material [13] defines some standard model elements like version, caption and human readable description. Apart from these standard elements, resource specific elements can also be defined. An XML document that represents the resource can be generated from the resource model. WS-ServiceGroup specification provides the possibility to generate XML for a system of resources.

2.2.3.1.4 Resource Composability

Resource Composability refers to the ability to compose a manageable resource with some additional aspects and capabilities. This is depicted as follows [12]

![Figure 2.1 MUWS Resource Composability](image-url)
Some of the aspects are messaging, addressing, security, notification etc are provided by Web Services specifications like WS-Messaging, WS-Addressing, WS-Security, WS-Notifications etc respectively. The manageability capabilities can be 'generic' like identifying a manageable resource or 'resource-specific' like capabilities for printer management. Composability allows to enrich a manageable resource with additional features like security, notification etc.

2.2.3.1.5 Manageability Capabilities

OASIS has a defined set of manageability capabilities to manage IT resources. These capabilities are identified uniquely in time and environment. They have defined semantics and are associated with a set of properties, operations and events. Apart from the defined ones, resource specific manageability capabilities can also be defined. We shall take a look at some of the existing capabilities as defined by OASIS.

Identity:
This is used to uniquely identify a resource. It is a mandatory capability for any resource managed using MUWS. The property in Identity manageability capability is ResourceID. It is read-only capability and there should be only one instance of it for any given resource. Identity should be unique with respect to time i.e. regardless of whether the current resource is active or not, its ResourceID should never be used for any other resource in the future. The ResourceID should persist during the entire life time of a resource. A resource that provides multiple end points, should provide the same ResourceID across all of its end points. Characteristics of a managed resource can never be inferred from the ResourceID.

Manageability Characteristics:
Manageability Characteristics [12] defines those properties that provide necessary information regarding the end-point implementation rather than the resource itself. The property in Manageability Characteristics capability is ManageabilityCapability. It is non-mutable and non-modifiable. It can have zero to infinite occurrences.

Correlatable Properties:
Correlatable Properties helps to establish the oneness of resources. It exposes only those properties that help to establish the oneness of resources. There could be resources that expose more than one manageability endpoints each with different ResourceIDs. Say for example, a printing machine that also does scanning operations. There could be 2 manageability endpoints defined for the machine. One for the printer and the
other for the scanner. But the entire machine will have the same Host
number, IP number etc. This host number and IP number can be defined as
the correlatable property for the machine i.e. resource, as they help the
manageability consumer to understand the oneness of the machine, even
though it exposes two different manageability endpoints i.e. different
ResourceIDs.

2.2.4 Distributed Management Task Force (DMTF)

DMTF is an organization to develop, maintain and promote
standards for IT resources management. DMTF has developed some
standards for building components for systems management in a platform-
independent and technology-neutral way. Some of the standards established
by DMTF are [19]

1. Common Information Model (CIM)
2. Web Services Management (WS-Management)
3. Web Based Enterprise Management (WBEM)
4. Common Diagnostic Model (CDM)
5. System Management BIOS (SMBIOS)
6. Directory Enabled Network (DEN)

WS-Management was the standard of our concern.

2.2.4.1 WS-Management

It specifies a SOAP based web services protocol to manage IT
resources. It provides a common set of operations to enable management
systems to manage resources across different platforms. WS-Management
provides the Create, Delete, Put and Get operations to manipulate resources.
Apart from these standard operations it also allows the usage of custom
operations. It allows the enumeration of collections, subscription and
notification operations, discovery of WS-Management services, security, fault
handling etc.

2.2.4.1.1 Resource Addressing

WS-Management uses SOAP based addressing mechanism to define
references to other web service endpoints [14]. Basically WS-Management
uses 3 addressing models. They are basic addressing, default addressing
model and other addressing models that were not specified in WS-
Management. Every WS-Management compatible web service has a web service endpoint to access the managed resource. Thus WS-Management defines a web service endpoint [14] as an entity, processor or resource that can be referenced and targeted for web service messages. A web service endpoint contains an End Point Reference (EPR) that contains the necessary information to identify and reference a web service.

### 2.2.4.1.2 Default Addressing Model

In the default addressing model, an EPR is represented as a tuple of wsa:Address, wsman:ResourceURI and wsman:SelectorSet. wsa:Address denotes the network address of the managed resource. wsman:ResourceURI denotes the class or instance of the managed resource. wsman:SelectorSet denotes a particular instance of the resource, if the wsman:ResourceURI denotes multiple instances of the same type of resource. While the first 2 elements are mandatory, wsman:SelectorSet is optional. This tuple gets translated to SOAP message headers. So, the corresponding headers for wsa:Address, wsman:ResourceURI and wsman:SelectorSet are wsa:To, wsman:ResourceURI and wsman:SelectorSet.

### 2.2.4.1.3 Resource Access

WS-Management managed resources can be manipulated using any of the following requests.

**CREATE Request:** Used to create an instance of the managed resource. This request is sent to a resource factory. The body of the Create request contains resource specific representation of the constructor for the resource. Successful execution of the command returns the EPR of the resource. If the initial representation of the resource is incorrect, then the service returns wsmt:InvalidRepresentation fault data in the response. Fragment-level create request can be used to create an attribute of the managed resource.

**GET Request:** Used to retrieve the one-time snapshot of the resource. A snapshot [14] is a complete XML representation of a resource at the time the service processes the request. As the GET request is targeted to a specific managed resource, the SOAP body content of GET request is empty. If a resource representation could not be retrieved due to concurrency issues then wsman:Concurrency fault should be returned. Fragment-level GET operation can be used to retrieve a particular attribute or set of attributes of the managed resource. The request contains the XPath expression that will be executed against the complete resource representation by the service.
**DELETE Request:** Used to delete a resource in its entirety. A successful delete (empty response) invalidates the current representation of the targeted resource. Deletion can be either literal (Eg: a row from a database table) or logical (Eg: a printer or scanner). An extension specification written as optional SOAP header values provides the necessary pre-conditions to delete a resource. Fragment-level DELETE can be used to delete a particular attribute of the managed resource. The SOAP body element for normal DELETE and fragment-level DELETE is empty.

**Fragment-level access:** This denotes the facility to manipulate only parts of a resource rather than the entire resource itself. `wsman:FragmentTransfer` element is used to denote the attributes to be manipulated. It is used within the SOAP header part. As discussed earlier, we have fragment-level CREATE, DELETE, GET and PUT operations.

**Custom Actions:** Apart from the above defined 4 standard operations, WS Man also provisions the definition and usage of custom actions. Each custom action should have a unique `wsa:Action` defined.

### 2.2.5 Performance comparison of MUWS and WS-Management

In [15], the authors have compared the performance of OASIS’s Management Using web Services (MUWS) and DMTF’s WS-Management specifications in network management alongside with SNMP. The performance metrics used were network usage, response time, CPU usage and memory consumption. The evaluation scenarios included the usage of plain messages, compressed messages, secured messages and compressed + secured messages.

The evaluation architecture consists of 3 pairs (SNMP, MUWS and WS-Management) of Manager-Agent modules. While the Manager issues the necessary commands, the Agent carries out the task and reports back to the Manager. In this case, the routing table inside a router is taken as the network entity to be managed. The platforms on which the Manager-Agent pairs are running were the same. Apache Muse and Wiseman frameworks were used to implement Manager-Agent pairs for MUWS and WS-Management specifications respectively.

In terms of network usage, in [15], it has been observed that WS-Management performed better than MUWS in both plain messages and secured messages scenario. In case of WS-Man fragment-level access, network usage becomes more or less similar, but still WS-Man performs
better than MUWS. In compressed messages scenario, the performance between the two were almost similar. In compressed + secured messages scenario, there was only a slight difference between the two.

Considering the response time, in [15], it has been observed that WS-Management performed better than MUWS in both plain messages and compressed messages scenario. In secured messages scenario, the performance between the two were almost similar. In compressed + secured messages scenario, WS-Man has outperformed MUWS.

When it comes to CPU usage, in [15], WS-Management has outperformed MUWS in all the scenarios.

With respect to memory consumption, in [15], it has been found that WS-Management has consumed 35Mb and MUWS 52 Mb, which includes the memory for JVM and Tomcat.

### 2.3 Related Work

#### 2.3.1 Autonomic Network Management System

There exists several traditional and automated network management approaches like SNMP, data models, CORBA, software agents, active networks and policy agents. But the above approaches had their own drawbacks. For instance, software agents and active networks involve executing the management code on the managed devices. This poses a serious security threat, if the management functionality contains any malicious scripts. Active networks also bring in processing overhead and delays. Other burdens using these approaches are increased human involvement and their reactive nature rather than being proactive. Due to these drawbacks, autonomic network management becomes the need.

A Network Management System (NMS) is said to be autonomic if it satisfies the self-CHOP properties. The self-CHOP properties refer to self-Configuration, self-Healing, self-Optimizing and self-Protection properties. A NMS is said to be self-Configurable if it could detect the violation of Service Level Agreement (SLA) and configure the network appropriately to meet the SLA. A NMS is said to be self-Healing if it could restore its operations in the event of any failures. A NMS is said to be self-Optimizing if it could perform its management tasks in the most efficient way by making the best use of the available resources. A NMS is said to possess the ability of self-Protection if it could protect its management capabilities from any external threats.
As defined by IBM’s generic framework, an Autonomic Network Management System (ANMS) has an autonomic manager that should be monitoring the managed entities or managed element, analyzing their performance and planning and executing a set of actions [1]. These actions are referred to as the MAPE (Monitor, Analyze, Plan and Execute) loop [1] that revolves around a knowledge-base. Managed element has a couple of interfaces, sensors and effectors. Sensor is the one that sends the measurement information to the autonomic manager. Effector is the one that sends the corrective commands to the managed element.

The Network Knowledge Base (NKB) defines a model of the managed system which in our case is the Ethernet service. Two different forms of knowledge [1] used are domain knowledge and control knowledge. Domain knowledge denotes the view or conceptualization of the managed domain. Control knowledge denotes the ways to manage the system. Domain knowledge [1] is further divided into structural knowledge and behavioral knowledge. An ANMS achieves self-awareness through this NKB.

Autonomic network management architectures are classified based on Degree of activity, degree of adaptability, degree of intelligence, degree of awareness, memory strength and degree of autonomy and autonomicity. Some of the existing approaches to ANMS architectures are Unity project, DASADA, Autonomia and The ACCORD project [1]. A challenge in building NKBs is the development of an expressive network model that can be adapted by different network hardware and software vendors.

The authors of [2] discuss about a generic framework called Autonomic Service Architecture (ASA) for the automated management of both Internet services and the network resources they use. It also ensures the Service Level Agreement (SLA) between the Service Provider (SP) and the Customer. In ASA, everything is viewed as a service. It ensures the self-CHOP properties of any autonomic management system. ASA makes use of MPLS label stacking technique and path-oriented bandwidth management to support VN-based network provisioning.

ASA defines service, as the engagement of a resource for a certain period of time, based on the SLA. ASA contains two views. The operation view and the management view. Operation view is concerned with the control plane and data plane, presented from traditional service views of different service layers. Management view is concerned with the management of the services provided. Management functions include resource management, policy and SLA management, accounting and billing management. The possibility of using Web Services to provide a common interface to interact with different physical resources is discussed in [2]. But, owing to the verbose nature of
the XML and its processing overhead, the authors have agreed on a Common Resource Format (CRF).

2.3.2 Multi-Protocol Label Switching

MPLS [17] is a high speed data carrying mechanism in data and telecommunications network. Each and every data packet is encapsulated with a MPLS label containing the routing information, beforehand. These packets are forwarded completely based on this routing information thus providing interoperability between different types of traffic like ATM, Ethernet etc. and scalability. The intermediary routers need to inspect only the MPLS label rather than the actual packet. This reduces the packet processing time at each router and therefore increases the communication speed which becomes useful in multimedia data communication services like video/audio conferencing etc. Before the actual packet transmission starts, paths between different destination end-points are setup. These paths are called Label Switch Path (LSP). A node/end-point at the start of a LSP is called ingress router whereas the one at the end of a LSP is called egress router. All the other nodes in the intermediate path are called as the Label Switch Routers (LSRs). The authors of [6] discusses on how to achieve Quality of Service (QoS) using MPLS.

2.3.3 Simple Network Management Protocol

SNMP is a protocol proposed by IETF to manage networks. It is accompanied by Structure of Management Information (SMI) and Management Information Base (MIB-II). SMI is a standard that describes the specification of the managed objects [9]. MIB is a document that describes the hierarchy amongst the managed objects [9]. A network managed via SNMP has the following 3 components [16]

1. A managed device.
2. Agent, a software unit that runs on the managed device.
3. Network Management System (NMS), software that runs on the manager.

A managed device is any network node that implements the SNMP interface. This interface allows the managed device for the exchange of node specific information to the NMS’ via its agent. Task of the agent is to convert this information from node specific format to that of SNMP and vice versa. The NMS monitors and administers the network nodes to achieve the specified management goals.
Attributes describing a managed system are defined and arranged in a hierarchical format in MIB. NMS uses SNMP to operate upon these MIB variables to manage the network device. SNMP provides Protocol Data Units (PDU) to manage the network nodes. A few of the PDUs are GetRequest, SetRequest, GetNextRequest, GetBulkRequest, Response and Trap.

2.3.4 Ethernet services

Ethernet services refer to a variety of services offered by Service Providers (SP) to their customers over an Ethernet network. In this thesis we are concerned about the services offered over MPLS based Ethernet network. Ethernet services were standardized by an industry consortium named Metro Ethernet Forum (MEF). The Ethernet services defined by MEF are E-line, E-LAN and E-tree. E-Line service type is concerned with point-to-point service and E-LAN is concerned with multi point-to-multi point service. E-tree is about point-to-multipoint service.

2.3.5 RSVP-TE

RSVP-TE stands for Resource Reservation Protocol (RSVP) for Traffic Engineering (TE) [26]. It is used to reserve the resources along a LSP over which Ethernet services can be provisioned. The resources are reserved taking the SLA into consideration. The resource reservation style to be used in a RSVP session along a LSP is decided by the egress node rather than the ingress node.

The protocol uses three types of resource reservation styles. They are Fixed Filter (FF), Wildcard Filter (WF) and Shared Explicit (SE) styles [26]. In FF style [26], resource reservation is unique for each ingress node. This style is used when there are multiple LSPs running concurrently and independent of each other. FF style is used in point-to-point communication scenario. In SE style [26], a common set of resources are shared by a group of ingress nodes. The egress node decides the list of ingress nodes to be included in the style. This style is used in a multipoint-to-point communication scenario. In WF style [26], a common set of resources are shared by all the ingress nodes independent of the egress nodes’ choices.

RSVP-TE uses two types of messages. The path message and the resv message.
2.4 Previous Master Thesis Work

The authors of [25] discuss about a novel approach for automatic provisioning and validation of Ethernet services over MPLS based transport networks. Some parts of the proposed system have been automated. It includes translation of Ethernet specific parameters to MPLS specific attributes, GMPLS signaling to provision a LSP and provisioning of measurement points through a point-and click graphical user interface. The prototype developed has a RESTfull web service interface implemented for communication between the GUI and Management Module.

2.5 Summary

From this related study, it can be seen that extensive research has been made in the area of self-organizing networks. IBM’s proposal of ANMS architecture looks promising. Also, though SNMP is simple and easy to use, it has its own drawbacks. Web services, with its platform independence feature seem to provide promising support for achieving scalable self-organizing management functionality on transport networks. Standards like OASIS’ MUWS and DMTF’s WS-Management will be providing the necessary platform to develop our web services based solution.
3. System Requirements

3.1 System Architecture

The Metro Self-Organizing Network (MeSON) system which is discussed in [25] is concerned with the provisioning and management of Ethernet services on GMPLS networks. This MeSON system will be denoted as SON system hereafter. The architecture is given below.

The human operators (Network Operations Center) represent the client-side of the MeSON system. The Self-Organizing Network (SON) module in the middle will simulate most of the manual tasks. The In-Network Functionality (INF) represents the nodes over which the Ethernet services are provided. The thesis is concerned with creating a web service based interface between the client and SON module. The requirements discussed below are the ones that this web service based interface needs to meet. It includes some mandatory and optional requirements.
3.2 Support for Create Read Update and Delete (CRUD) operations

The system needs to support creating, reading, updating and deleting operations with respect to managing an Ethernet service. That is, it needs to provide a Create operation to provision a service, Read operation to get the information about a service, Update operation to update a service and Delete operation to tear-down a service. This is a mandatory requirement.

3.3 Handling one request at a time

In the SON system, before an Ethernet service is provisioned, the RSVP-TE protocol needs to reserve the necessary resources to setup the GMPLS tunnel. The Ethernet service is then provisioned over this tunnel. So, when multiple clients send in the requests to provision different Ethernet services over the same group of nodes, the possibility of reserving an already reserved resource exists. So, in order to maintain simplicity, requests from the clients needs to be handled one at a time. This is a mandatory requirement.

3.4 Asynchronous communication

As said earlier, the client that communicates with the SON module could either be a human operator or any other independent system. Normally, the client after sending a request might need to wait to get a response back. But optionally, the response can be sent back to the client asynchronously. This will allow the client to carry out other tasks while its request is being processed by SON. This is an optional requirement.

3.5 Prioritization of Requests

SON’s clients are categorized as gold, silver and bronze types. Gold type clients have the highest priority and for their services more resources will be allocated. The bronze type customers have the lowest priority and for their services limited amount of resources will be allocated. So, accordingly, their requests could also be prioritized. A problem in implementing this requirement would be the starvation of silver and bronze requests in case of large number of gold requests. This is an optional requirement. A possible solution to implement this prioritization of requests is discussed under the system architecture section. A possible solution for the starvation problem is discussed in the future aspects section.
3.6 Support for subscription and notification

Self-Organizing Network (SON) systems, as the name suggests, should be able to continuously monitor the SLA and be able to take corrective actions to ensure that the SLA is not violated. Optionally, SON could send some notifications to the client regarding SLA violation and the corrective steps taken. There arises the requirement for subscription and notification. Clients could subscribe to SON for several events and can get notified whenever needed. This is an optional requirement.
4. Proposed System Architecture

In order to reduce the manual involvement in managing Ethernet services, a new Self-Organizing Network (SON) module or Management Module (MM) has been proposed. This SON module will act as the middleware between the manual network operators and In-Network Functionality (INF). INF here refers to all the nodes belonging to the Generalized-MPLS (GMPLS) network over which the Ethernet service is provisioned. SON will automate many of the manual operations in order to meet the Service Level Agreement (SLA), thus reducing the burden on the human network operator. This proposed SON module will be an attempt to solve the research question 4.

4.1 Internal Architecture of SON

The internal architecture of the SON module is depicted and discussed below. Though the entire internal architecture is depicted below, the thesis is concerned on SON-GUI and SON Engine modules.

![Figure 4.1 Internal Architecture of SON module](image-url)
**SON Engine:** It acts as the brain of the SON module. It communicates and coordinates with the other internal modules to establish the management functionality.

**SON-GUI:** It acts as the interface through which the clients will interact with the SON module. Clients could either be human network operators or any other independent software.

**SON-INF:** It acts as the interface between the SON module and the INF.

**Common Information Model (CIM):** This refers to the Common Information Model (CIM) schema that contains the mapping between MEF and GMPLS parameters. This mapping information will be used by the SON Engine to translate the MEF parameters sent by the client to the GMPLS parameters in order to be understood by the GMPLS control plane.

**Ontology:** This module contains the mapping between the performance parameters and the protocols and tools that will be used to measure them.

**Aggregation and Filtering:** This module aggregates the measurement data sent by the INF and filters out the unnecessary data. This filtered data will be sent to the Data database for storage. This module is outside the scope of the thesis.

**Root Cause Analysis (RCA):** Whenever there is a SLA violation, as identified from the measured data, the RCA module will try to figure out the reason behind the SLA violation. This module is outside the scope of the thesis.

**Service DB:** It refers to the Service database. It stores the information about the provisioned Ethernet services. Information could be service name, service Id etc.

**Data DB:** It refers to the Data database. It stores the measurement data obtained from the INF. This data will be used to study and analyze the behavior of the Ethernet network.

For this thesis, both the databases were implemented and used to store the Ethernet service information and the measurement information.

As the task in this thesis is to investigate and implement a web service based self-organizing network functionality, we will be concerned with the SON-GUI and SON Engine modules only.
4.2 Internal Architecture of SON Engine

The internal architecture of the SON Engine module is discussed as follows.

![Diagram of SON Engine Architecture]

Controller communicates and coordinates with the other modules inside the SON Engine. It also handles the requests coming in through the SON-GUI interface. MEF-GMPLS translator does the translation of MEF parameters to GMPLS parameters by using the CIM mapping information. Ontology translator identifies the measurement tools and protocols needed for the GMPLS control plane to perform the measurement task. DB Communicator module provides the necessary database operations on the Service and Data database.

4.3 Architecture to handle one request at a time

The Producer-Consumer model has been adopted to meet the requirement of handling one request at a time. The following architecture has been proposed to meet this requirement.
APP and GUI represent the clients that will be interacting with the SON module. P represents the Producer process, C represents the Consumer (Controller) process, R represents the clients’ Requests and Q represents the requests queue. Producer, Consumer and request queue will be a part of SON Engine. The architecture will implement the following algorithm to handle one request at a time.

1. The clients send in their requests to the SON module.

2. The incoming requests are received by the Producer process and added into the request queue.
3. The Consumer process gets the request from the top of the queue and processes it.

4. Consumer waits until it gets a response from the In-network node.

5. When the Consumer gets the response, it sends it to the appropriate client who made the request.

6. Consumer repeats steps 3 to 5 until the request queue is not empty.

7. If the request queue is empty then Consumer stays idle.

As the request queue is a shared data structure, mutual exclusion needs to be provided amongst the processes accessing it i.e given at any time $t$, either the Producer or the Consumer should be allowed to access the request queue. This mutual exclusion can be achieved either by using a lock or a semaphore. So with respect to the internal architecture of the SON module, the Producer process can be considered to be a part of the SON-GUI interface.

The Consumer (Controller) process and the request queue $Q$ will be a part of the SON Engine itself. This architecture will also meet the optional requirement of asynchronous communication between the client and the SON module. By default, this request queue $Q$ works on First Come First Serve basis. By changing it into a Priority queue, the architecture also meets the optional requirement of Prioritization of Requests.
5. Interface Definition

One of our research questions is on how to achieve platform independence between the other systems i.e. client and our network management functionality. In this section we have defined a web service based interface to answer the question. The SON module will be implemented as an independent web service. Also, the client application will be wrapped around by another web service application. This will make sure that, regardless of the programming language or platform the client might be implemented in, the SON module will be able to communicate with it in order to send back the response.

The signature of some of the web service methods have been defined and depicted in the following diagram.

![Diagram of SON Interface Definition](image)

Figure 5.1 SON Interface Definition
5.1 Producer Interface

In the producer interface, we have defined two methods namely sendRequest() and putRequest().

5.1.1 The sendRequest() method

This public method takes in a string parameter named reqType and a string array reqParams. This generic method will be called by the client application to send any type of requests like service provisioning, service tear-down etc. The string parameter reqType denotes the type of request being sent by the client. The string array parameter will contain all the necessary information to execute the corresponding request type. A value in the reqParams array will be of the following format

parameter_name=value

where

parameter_name denotes the service attributes like service name and other MEF defined Ethernet service parameters.

value denotes the corresponding value that is constrained by MEF’s definition.

For any given sendRequest() call, the reqType parameter could take one of the following values.

CREATE: Denotes that the request is to provision a new Ethernet service. The reqParams array will contain all values like service name and other MEF based parameters and values that are required to provision a new service.

DELETE: Denotes that the request is to tear-down an existing Ethernet service. The reqParams will contain the service name or service ID of the service to be torn down.

UPDATE: Denotes that the request is to update the attributes of an existing Ethernet service. The reqParams will contain the service name or service ID and other MEF parameters and values that are to be updated for a provisioned service.
**READ:** Denotes that the request is to get the information about an existing Ethernet service. The `reqParams` will contain the name or ID of the service whose attributes are to be read.

**MEASURE:** Denotes that the request is to get the measurement data for an Ethernet service that is already provisioned.

The `sendRequest()` returns a String value immediately to the client. This value can be an OK or an ACK.

This simple definition of the web service method of `sendRequest()` will ensure the scalability of the SON module to handle other types of requests as well in the future with very less code modification. The usage of the Factory Design Pattern and Singleton design pattern has been explored to achieve scalability with the least code modification and efficient memory usage.

**5.1.2 The putRequest() method**

This method is private to the Producer process. The Producer process calls this method to add the request object in to the request queue. The mutex semaphore has to be obtained in order to add the request object in to the queue. When the request object is created and added into the queue, the Producer process releases the mutex semaphore.

**5.2 Client Interface**

The following 3 methods in the wrapper web service on the client side have been proposed.

```plaintext
sendResponse (provResponse: ProvisioningResponse)
sendResponse (resultResponse: ResultResponse)
sendResponse (tearDownResponse: TearDownResponse)
```

These methods are called by the Controller process in the SON module in order to send back the response.

The `sendResponse(provResponse: ProvisioningResponse)` method returns the result for the CREATE request through the `ProvisioningResponse` object.
The `sendResponse(resultResponse:ResultResponse)` method returns the result for the GET, UPDATE and MEASURE requests through the `ResultResponse` object.

The `sendResponse(tearDownResponse:TearDownResponse)` method returns the result for the DELETE request through the `TearDownResponse` object.
6. System Implementation

An essential part of this thesis was to develop a proof-of-concept for the solution proposed here within. Here we discuss about the prototype that was developed and the research question 2 is also answered.

6.1 Why WS-Management over MUWS?

As discussed already, two web services based resource management standards have been studied. Though both the standards provided enough features for managing IT resources, WS-Management was chosen over MUWS for the following reasons.

6.1.1 Relevance for Ethernet Service Management

The CRUD operations of WS-Management were more relevant for the typical management of an Ethernet service. For example, the CREATE operation corresponds to provisioning an Ethernet service, DELETE operation corresponds to tearing down an Ethernet service. The same for GET and PUT operations for retrieving and re-provisioning an Ethernet service respectively. On the other hand, in case of MUWS, the developer has to define resource specific manageability capabilities. This prolongs the development time. So, in case of WS-Management it is more straightforward whereas in case of MUWS it is more un-defined by the standard itself.

6.1.2 Usage of CIM schema

As seen already in the architecture section, CIM schema is being used to map the MEF parameters to that of GMPLS. DMTF provides specifications for WS-Management CIM binding and WS-CIM mapping. So, developing a prototype using DMTF based standard like WS-Management will provide flexibility in development in the future.

6.1.3 Open source implementation

Two frameworks namely Wiseman and Apache Muse were identified to be implementing DMTF’s WS-Management and OASIS’ MUWS respectively. Both Muse and Wiseman were available as open source implementations.

6.1.4 Resource Addressing

WS-Management provides basically 3 mechanisms for addressing a managed resource. They are the basic addressing mechanism, default addressing mechanism and resource-specific addressing mechanism. Combination of the above mechanisms can also be used. On the other hand, we do not have such options with MUWS.
6.1.5 Performance Issues

As already seen under the state of the art section, WS-Management has given a better performance than MUWS.

6.2 The Wiseman Framework

Wiseman is an open-source, java based implementation of DMTF’s WS-Management standard. It was developed by Hewlett-Packard. It provisions the development of WS-Management compliant SOAP based web services for managing resources. It uses XML schema to represent resources. It uses JAXB and DOM XML parsing techniques to manipulate the resources. So, a good conceptual and practical knowledge in Java, XML, XML schema, JAXB, DOM and XPath queries were necessary to work on this framework.

Wiseman also provides a web service runtime environment for hosting Wiseman based web services like our prototype. Additionally, Wiseman also provides some of the following tools [23].

**Xsd2WsdI tool:** It generates the WSDL file from the XML schema representing the resource.

**Wsdl2Wsman tool:** It generates the java source files from the WSDL file.

**Ant scripts:** To clean and build the web service so that it could be deployed under a J2EE web server as a web archive (WAR).

**Project Template:** Provides all the necessary library files, Ant script files and the other configuration files.

**Command line utility:** It helps to communicate and thereby testing a Wiseman based web service.

**Sample applications:** There were a couple of sample applications to help the developer.

6.2.1 Wiseman Conceptual Architecture

The conceptual architecture of Wiseman, taken from [23] is shown below.
As one can see from the above diagram, Wiseman API is built over Java API for XML based Web Service (JAXWS). As the web service is Wiseman compliant, the client also has to be Wiseman compliant in order to understand the WS-Management specific messages communicated between them.
6.3 Implementation Architecture

The implementation architecture is depicted in the following diagram.

![Implementation Architecture Diagram]

Figure 6.2 Implementation Architecture

The highlighted parts as seen in the above diagram, the Client and Controller parts are implemented using Wiseman framework. Client part represents any other system that interacts with the SON module. As already discussed in the previous section, the client was developed to be Wiseman specific in order to understand the WS-Management specific messages. Other parts DB Communicator, MM-INF, Measurement Task, INF, Service DB and Data DB were built independent of the Wiseman framework.
6.4 The Prototype

Following were the tools/environment used for development and deployment of the prototype.

6.4.1 Development Environment

The system was developed in Windows Vista operating system. Eclipse4Ericsson was used as the IDE. Wiseman 1.0 framework was used.

6.4.2 Deployment Environment

The bundled WAR file of our prototype was deployed in Tomcat web server under Linux Fedora 11 environment.

6.4.3 Semaphores

We have a requirement to handle only one request at any given instance of time. It has been identified that Wiseman framework does not allow asynchronous mode of communication. So, it was realized that the Producer-Consumer model of asynchronous communication cannot be implemented. Therefore the requirement to handle one request at a time could not be implemented using the Producer-Consumer model. An alternative way to implement it would be to provide mutual exclusion in accessing the create(), put() and delete() operations i.e. allowing only one request to access any one of the three methods at any instance of time. This mutual exclusion has been achieved using a semaphore.

6.4.4 Databases

A couple of databases have been designed to store the Ethernet services' information and measurement data information. The Service DB database stores the information about all the Ethernet services provisioned. The Data DB database stores all the measurement data collected from the In-Network Functionality nodes. They both have been implemented in MySQL.

6.4.5 Service Management Process

Client requests to manage an Ethernet service will be handled in the following way.

1. The incoming request is received by the sonserviceresource_Handler class. Through its handler functions it delegates the request to the SonserviceresourceHandler class.
2. The SonserviceresourceHandler class delegates the incoming request to the SonserviceresourceHandlerImpl class through its request handler functions.
**Provisioning an Ethernet Service:**
a. **SonserviceresourceHandlerImpl** calls the `create()` method of `MMINFInterface` class to provision the service.

b. Make an entry for the new service into the service database.

c. Trigger the automatic measurement process.

**Tearing down an Ethernet Service:**
a. **SonserviceresourceHandlerImpl** class calls the `delete()` method of `MMINFInterface` class to tear-down a service.

b. The service is marked as deleted in the service database but not deleted.

**Retrieving information about an Ethernet service/Measurement Data:**
a. The `get()` method of **SonserviceresourceHandlerImpl** class checks whether the XPath request “`measurementdata`” is contained in the incoming GET request.

b. If “`measurementdata`” is there in the request then, the measurement data for the concerned service is retrieved from the data database and sent back to the client.

c. If “`measurementdata`” is not there, then only the information about the service is retrieved and sent back to the client.

**Updating an Ethernet service:**
a. **SonserviceresourceHandlerImpl** class calls the `modifyService()` method of `MMINFInterface` class to update a provisioned Ethernet service.

b. The corresponding entry for the service in the service database is updated.

**6.4.6 Measured Parameters**

Following are the parameters that were measured. These parameters are MEF defined.

- measuredbandwidth
- numberofpacketslostup
- numberofpacketslostdown
- delayvariationup
- delayvariationdown
- numberofoctetsup
- numberofoctetsdown
- numberofpacketsup
- numberofpacketsdown
7. Analysis

Our Wiseman based prototype is being analyzed against a typical SNMP based or RESTfull web services based prototypes.

7.1 WS-Management versus SNMP

SNMP has always been the traditional way of managing networks. So, here we make a qualitative analysis between the WS-Management standard and SNMP.

7.1.1 Node-centric and Resource-centric

On using a SNMP based Network Management System, the managed system is treated as a node rather than a resource. Eg: router, host etc. Each and every individual physical entity will be treated as independent nodes. In our case, the managed entity is the Ethernet service that spans multiple physical entities. Using SNMP will require the interaction with all the physical nodes involved which will complicate managing this Ethernet service. In WS-Management, the entity that is being managed is treated as a single resource in itself. So, with respect to WS-Management, the Ethernet service is treated as a single entity to be managed even though it spans multiple physical nodes. This makes it easier to manage the Ethernet service.

7.1.2 Support for custom actions

In SNMP, operations are executed on the variables residing in the Management Information Base (MIB). Operations are pre-defined as Protocol Data Units (PDU) s. The manager has to only use these pre-defined PDUs to manage the concerned node. SNMP does not allow the possibility to define custom actions. In case of WS-Management, the developer can define custom actions specific for the managed resource. Though our current prototype does not have any custom action defined, Wiseman framework provides the room to define some. For example, custom actions like ProvisionService and TearDownService can be defined and used instead of Create and Delete actions.

In SNMP, the manager can only read and update the values of node attributes as defined in the MIB. Creation of new values and deletion of existing values for the node attributes is not possible. Whereas using WS-Management, the manager can create, update, get and delete values for any attribute of the resource whenever required. For example, if a new attribute named “numberoflsrs” is to be added, then this new attribute needs to be defined in the xml schema representing the Ethernet service resource and
the JAXB command for unmarshalling the xml schema needs to be executed to generate the corresponding JAXB objects.

7.1.3 Readability of variables

A SNMP based manager addresses a MIB variable through object identifiers. These object identifiers (OIDs) denote the hierarchical structure in the MIB database to identify a managed object (an attribute of the node). These OIDs have poor readability compared to the readability of the WS-Management managed resources’ attributes.

In WS-Management, the variables describing a resource are well defined in the XML schema. These variables’ name and data type clearly communicate their purpose of usage. WS-Management based SOAP messages can manipulate a resource using these variables. This makes the development of WS-Management based Management Systems easier.

7.1.4 Unique identifier creation

WS-Management allows for the dynamic creation of service with a unique End-Point Reference (EPR) on the network level. In case of SNMP, MIBs are created statically. Even if the MIBs are created dynamically there is no guarantee that its variables will be unique on the network level.

7.1.5 Language translation

Response messages sent back contain human readable descriptive information. Apart from English, these messages can also be translated to other languages like French, Spanish etc. using WS-Management. This feature is not available in SNMP.

7.1.6 Message size

WS-Management allows the application to set the size for each and every response message by using the wsman:MaxEnvelopeSize header value. The maximum size of the SNMP messages is specific for the SNMP implementation used. As SNMP runs over UDP, this maximum size of a SNMP message is limited by the maximum size of the UDP datagram packet used. Thus, in SNMP, the application does not have any control over the message size. Whereas WS-Management allows the application to specify the message size. Though the application is not going to send large messages of measurement data, we have denoted this comparison just to highlight the advantage that WS-Management offers over SNMP.
7.2 WS-Management versus RESTfull web services

Previously, before this master thesis, a RESTfull web service interface for resource management was developed [25]. As both WS-Management (SOAP based) and RESTfull web services are based on HTTP, we made a comparison between both the prototypes.

7.2.1 Support for Custom Actions

The REST based interface used only the basic HTTP GET, PUT, POST and DELETE methods. The GET request can be used to retrieve the information about the resource managed. The PUT request can be used to update the resource’s information. The POST method can be used alternatively to GET. The DELETE method can be used to remove a resource. Also, in general, any given RESTfull web service could use only the above said HTTP methods but not any resource specific custom methods. On the other hand, WS-Management standard, apart from providing the basic four actions for CRUD operations, it also provides the feature to define resource specific custom actions.

7.2.2 Nested Resource Access

Resources can be nested within each other. Accessing and manipulating such nested resources should be simple and efficient. In the REST based prototype, the Uniform Resource Identifier (URI) of nested resources is represented as a simple string. Thus it requires complex string parsing to access such resources. The URI represents the address and name of the resource in order to access it and manipulate it. In case of our WS-Management based prototype, the JAXB parsing technique is used to access resources. JAXB is an object-oriented XML parsing technique that creates Plain Old Java Objects (POJOs) from the parsed XML string. So, whenever the user wants to add more resources or add more attributes to any existing resources, only the xml schema data representing the resources needs to be updated and the JAXB command for unmarshalling the xml schema needs to be executed to generate the corresponding JAXB objects. While the user can provide the necessary resource specific data to the developer, the developer has to manipulate the xml schema file and run the JAXB command. Thus it is easier to work with nested resources in our WS-Management based prototype compared to the REST based prototype.
7.3 Advantages and Disadvantages of using Wiseman

The usage of any framework has its own pros and cons. In this section we shall discuss about what we gained and lost by using the Wiseman framework. Following are the features we gained by using the Wiseman framework.

7.3.1 Support for CRUD operations

The create(), get(), put() and delete() methods defined in the Wiseman framework allows us to easily implement the Create, Read, Update and Delete operations respectively.

7.3.2 Handling one request at a time

A semaphore named 'mutex' has been defined. Any request needs to access this semaphore in order to execute a Create or Put or Delete operation. As the semaphore is global for all the 3 above mentioned operations, this ensures only one request will be executed at a time for manipulating resources. Note that as the read operation does not do any resource manipulation, there is no need to provide mutual exclusion for this operation.

7.3.3 Custom actions

Though we did not define any Ethernet service management specific custom action, Wiseman provided the facility to define custom actions. A possible custom action that could be implemented is “GetMeasurement” to retrieve the measurement data from the In-Network Functionality nodes.

7.3.4 Object-oriented interaction

Wiseman provides the ease to interact with the resource in an object-oriented fashion. Resources in Wiseman are represented as XML schema. Wiseman uses Java API for XML Binding (JAXB) that provides an object-oriented way of interaction, to manipulate the XML schema file.

Following are the features we lost by using the Wiseman framework.

7.3.5 Asynchronous communication

Wiseman framework supports only synchronous communication between the Wiseman client and service. So the requirement, to establish asynchronous communication between the SON service and client could not be established.
7.3.6 Prioritization of requests

Given the fact that the Wiseman framework does not support asynchronous communication, all the requests could be handled only in the First Come First Serve (FCFS) order. So, the requirement to handle requests based on its priority could not be met.

7.3.7 Wiseman based client

The prototype developed in this thesis, requires that the client has to be Wiseman compliant in order to understand the WS-Management based SOAP messages. Wiseman currently supports other toolkits like JAX-WS and Axis to consume WS-Management specific WSDLs and to define Wiseman specific clients.
8. Answers to the Research Questions

In this section we shall discuss on whether the research questions are answered.

8.1 What kind of web services (SOAP or RESTful) should be used for managing Ethernet services?

As there was already a prototype developed using RESTful web services, we decided to explore the possibilities of using SOAP based web services standards for resource management like OASIS’ MUWS and DMTF’s WS-Management. Also the decision to use SOAP based web services helped us to realize a couple of advantages that SOAP provides over RESTful web services. They are, SOAP based web services support custom actions and nested resource access.

8.2 How could we use standards like OASIS’ MUWS and DMTF’s WS-Management in managing Ethernet services?

Sections 2.2.3 and 2.2.4 discusses about the standards. 2.2.5 compares the performance of WS-Management and MUWS. It was observed that WS-Management outperformed MUWS in terms of network usage, response time, CPU usage and memory consumption. Finally, based on the details provided in the section 6.1, it was decided to use DMTF’s WS-Management standard.

8.3 How could we make our solution to be used by other systems in a platform independent way?

As discussed in section 2.2.1, Web Services provided the possibility for other systems to communicate with the SON module in a platform-independent way, though the prototype implemented required that the clients would need to link against the Wiseman framework which is open source.

8.4 How could we automate most of the manual tasks in managing the Ethernet services?

As discussed in the system architecture section, an explicit module called Self-Organizing Network (SON) module has been proposed and implemented only on the interface level. The complete implementation of SON module in the future will aim at implementing the Self-CHOP properties thereby automating most of the manual tasks and reducing the burden on the human operators.
9. Conclusion

The aim of this thesis is to investigate and find a possible way to manage Ethernet services on self-organizing networks through a web service interface. Web service standards like OASIS' MUWS and DMTF's WS-Management were explored. Based on our requirements and the comparative study made on both the standards, it was identified that DMTF's WS-Management standard provided a suitable platform to meet the aim.

Every framework has its own pros and cons. The Wiseman framework is no exception. As already seen in the analysis section, we were not able to meet some of the requirements like asynchronous communication between the client and server, prioritization of requests etc. The mandatory requirements of supporting CRUD operations and handling one operation at a time have been met. The Wiseman framework had some bugs in itself. Workarounds were done to overcome these bugs and finally a working prototype has been implemented. Also, the documentation provided for the framework were not enough. So, communication with its developer community was established. One of the developers named Denis Rachal provided valuable support at crucial moments.

From an academic perspective, this thesis has proved that it is possible to use SOAP based web service standards like DMTF’s WS-Management for network management.
10. Future Aspects

In this thesis, a web services based interface to manage Ethernet services is implemented. Apart from the existing features, some new features can also be added. They are discussed as follows.

10.1 Support for Subscription and Notification

The customers and service providers agree on a set of Service level Agreements (SLA) that should not be violated. So, in order to maintain this SLA, the SON engine takes corrective actions whenever it is violated. Such corrective actions could be allocation or de-allocation of resources. Clients can register/subscribe their interest for such events and can get notified. WS-Management standard provides enough specifications to implement this subscription-notification feature.

10.2 Support for Publishing and Discovery of service

The very essence of web service technology is the dynamic discovery and establishment of communication between the client and web service. Currently in our prototype, the WS-Management client identifies the web services encapsulating the Ethernet service, in a static way. That is, the Resource URI of the managed resource is hard coded in the implementation. A possible enhancement will be to publish this Resource URI in some directory service like UDDI or DNS and allow the client to contact our web service dynamically.

10.3 Solution for starvation of Silver and Bronze requests

As a first step to solve the starvation problem, we allocate a separate queue for each type of requests. Let $Q_g$, $Q_s$ and $Q_b$ represent the three different queues for Gold, Silver and Bronze requests respectively. Let $g$, $s$, $b$ represent the number of requests handled by the consumer process from $Q_g$, $Q_s$ and $Q_b$ queues respectively at any given point of time. The solution is discussed in the following scenarios.

**Case 1:** Number of requests in $Q_g>0$, $Q_s>0$ and $Q_b>0$.
For every $g$ number of requests handled, there should be at least $s$ and $b$ number of requests handled, such that the condition $g>s>b$ is always true.

**Case 2:** Number of requests in $Q_g=0$ and number of requests in $Q_s>0$ and $Q_b>0$. 


For every s number of silver requests handled, there should be at least b number of bronze requests handled such that the condition s>b always holds true.

The conditions g>s>b and s>b ensures that the silver and bronze requests will be eventually handled thus resolving the starvation problem.

Also, there should be only one mutex lock for all the three queues.
11. Bibliography


Annex 1  Directories

Following are the various packages for the SON web service.

xsd\schemas.wiseman.dev.java.net\schema\ - contains the XML schema file. This XML schema file represents the SON Ethernet service that is to be managed.

wsdls - contains the WSDL file that describes the SON web service deployed.

com.sun.ws.management.server.handler.net.java.dev.wiseman.schemas .schema.sonservice_xsd.sonserviceservice - contains the resource handler class <resource>_Handler that will delegate the incoming requests to the appropriate WS-Management operations defined in the <resource>Handler class.

net.java.dev.wiseman.schemas.schema.sonservice_xsd.sonserviceserv ice.sonserviceresource - contains the <resource>Handler class that will handle all the requests delegated to it by the <resource>_Handler class.

net.java.dev.wiseman.son.service.impl - contains all the classes that implement the WS-Management operations and custom operations if any.

dist - contains the WAR file to be deployed under the Tomcat server.
Annex 2 Classes

Following were some of the classes in the SON web service.

`sonserviceresource_Handler.java` - denotes the class that will be receiving all the incoming requests from the client. It then delegates these requests to the `<resource>`Handler class. This class represents the SON-GUI block in the SON internal architecture.

`SonserviceresourceHandler.java` - denotes the class that will be handling the requests delegated to it by the `<resource>`_Handler class.

`SonserviceresourceHandlerImpl.java` - denotes the class that will contain the actual implementation of the WS-Management operation handlers. This is where the application logic is implemented.

The above 2 classes represents the Controller block in SON Engine – internal architecture.

`MMINFInterface.java` - It is a class that will act as the interface between the SON engine and the In-Network Functionality (INF) nodes. It is defined only on the skeletal level. Except the function `startMeasurements(Logger log, String service)`, all the other functions are not implemented. This function will be triggered by the CREATE request handler as soon as a new service has been provisioned. This function will trigger automatic measurements in the INF layer via `MeasurementTask.java` for every 10 seconds for the given service. This class represents the SON-INF block in SON internal architecture.

`MeasurementTask.java` - This denotes the TimerTask class that will be triggered by the `MMINFInterface.java` class. This class will be one that communicates with the INF nodes and gets the measurement data. Currently it will generate dummy measurement data and add it into the Data database. This class will be a part of the Controller block in the SON Engine – Internal Architecture.

`DBCommunicator.java` - This class provides the common interface to communicate with the two databases. It provides functions to create, update, retrieve and delete Ethernet services information into the database Service DB. It is also called by the `MeasurementTask` class to add the measurement data into the database Data DB. This class represents the DB Communicator block in the SON Engine – Internal Architecture.
SONClientApp.java - This class denotes the client application that will interact with the SON web service through the standard WS-Management CRUD operations. It provides a GUI through which a human network operator can interact.

Apart from the above stated classes, Wiseman also use JAXB classes for marshalling and un-marshalling the objects.
Annex 3   Other Files

**resource-handler-config.xml:** It contains the XML tags `<resource-pattern>` and `<resourcehandler-class>`. The `<resource-pattern>` contains the value of "resource.uri" as found in the project.properties file. The `<resource-handler class>` tag should contain the full path of `<Resource>_Handler class`. In our case it is the `<sonserviceresource_Handler>` class. Without proper values for the above said XML tags, the client will not be able to communicate with the server. This is very important.

**project.properties:** Contains the SON web service's configuration information.

**DOS BAT files:** Eclipse IDE was used for development. Also, the Wiseman framework has its own set of working directories which were different from those of Eclipse. So, in order to keep them synchronized with each other, several files needs to be copied from and to the directories in both the environment. Also, the deployment files needs to be copied from the local development machine to the Linux-based deployment machine. To accomplish all the above said tasks, some DOS batch files were written.

**data.txt:** Contains the SQL script to create the tables in data database.

**service.txt:** Contains the SQL script to create the tables in service database.

**MeSON_UML.mdl:** It is the Rational Rose model file that contains the relationship between various classes used.
Annex 4  System Execution

The system can be executed as follows.

1. Start the Tomcat server.

2. Run the 'ant' command to generate the WAR file.

3. Run the CPY2VM command to copy the WAR file and all the other necessary files to the deployment machine.

4. Run the client application.

5. In the client application window, set the 'HostName' field value to the URL where the SON web service can be accessed. Eg: 192.36.157.153:8080.

6. Fill in the necessary fields and click the 'Create' button to create the service. Once when the service is created, it is listed in the service list on the left side of the window.

7. Choose the service from the services list, to see information about it.

8. If any of the service parameter is to be updated then, choose the respective service from the service list, modify the necessary attributes and click on the Put button.

9. If any of the service is to be deleted then, choose the respective service from the service list and click on the Delete button.

10. If the measurement data for any service is to be retrieved then, choose the respective service from the service list and click on the 'Get Measurements' button to get the measurement data for the chosen service.