Master Thesis

REST-based network management solution for accessing SNMP infrastructures

by
Duc-Lam Vu

Supervisors: Prof. Dr. rer. nat. habil. Dr.h.c Alexander Schill
Dipl.-Inf Marius Feldmann

1 March 2008 - 20 August 2008
Acknowledgment

This report is a result of the Master thesis provided by the Chair of Computer Network at TU Dresden. My warm gratefulness and gratitude go to Prof. Alexander Schill and Dipl.Inf Marius Feldmann at the Chair of Computer Network, TU Dresden for being my supervisors.

I would like to take the opportunity to present my sincere thanks to Dipl.Inf Marius Feldmann for giving me the unique opportunity and the continual 6 months guidance and encouragement.

This work is dedicated to my wife Ha Nguyen who has given me strong support during the period of writing the Master thesis.
Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

Dresden, 20 August 2008

Duc-Lam Vu.
# Table of Contents

Chapter 1 Introduction ................................................................. 5  
1.1 Motivation ........................................................................... 5  
1.2 Thesis Goals ........................................................................ 6  
1.3 Structure of Thesis ............................................................... 7  

Chapter 2 Technological Overview and State-of-the-Art ................. 9  
2.1 REST .................................................................................... 9  
  2.1.1 REST Background .......................................................... 9  
  2.1.2 Stateless Client-Server ................................................... 10  
  2.1.3 Resource and Resource Identifier .................................. 10  
  2.1.4 Resource Representations ............................................. 10  
  2.1.5 Uniform Interface .......................................................... 11  
  2.1.6 Advantages of REST ...................................................... 11  
2.2 SNMP Technology ............................................................... 12  
  2.2.1 SNMP Background ....................................................... 12  
  2.2.2 Internet Management Model ........................................ 13  
  2.2.3 Layered Communication Model ..................................... 14  
  2.2.4 MIBs and Object Identifiers ......................................... 16  
  2.2.5 SMI Definitions ............................................................ 18  
  2.2.6 SNMP Operations ......................................................... 19  
  2.2.7 SNMP Message Format ............................................... 20  
2.3 Ajax ..................................................................................... 21  
  2.3.1 Mechanisms ................................................................. 22  
  2.3.2 Ajax Frameworks .......................................................... 24  
  2.3.3 Advantages of Ajax ....................................................... 27  
  2.3.4 Disadvantages of Ajax .................................................. 27  
2.4 Comet .................................................................................. 28  
  2.4.1 Comet Background ...................................................... 28  
  2.4.2 Comet Mechanism ....................................................... 28  
  2.4.3 Long Polling Mechanisms ............................................ 28  
2.5 Related Works .................................................................... 29  

Chapter 3 Architecture Overview ............................................... 32  
3.1 Contribution and Requirements Analysis ................................ 32  
3.2 General Communication Model .......................................... 34  
3.3 General Approach ................................................................ 35  
  3.3.1 Protocol ....................................................................... 35  
  3.3.2 Resource Design .......................................................... 36  
  3.3.3 URI Design ................................................................. 36  
  3.3.4 Resource Representations ........................................... 37  
  3.3.5 REST-based General NWM Architecture ...................... 39  
3.4 Specification of REST Uniform Interface ............................... 40  
  3.4.1 SNMP Read Request .................................................... 40  
  3.4.2 SNMP Modify Request ................................................ 44  
  3.4.3 Trap Asynchronous Operation ...................................... 45  
  3.4.4 Get List of MIB Nodes ................................................ 46  

Chapter 4 Design and Implementation ......................................... 48  
4.1 Used Technologies and Libraries ......................................... 48
Chapter 1 Introduction

1.1 Motivation

The World Wide Web (commonly shorted to the Web) has gone beyond as a system of interlinked hypertext documents accessed via the Internet. Web integration allows any Web brows-able content to integrate into many legacy applications which are not the originally intended areas of Web technologies such as content management systems (CMS), databases (DB), B2B (Business-to-Business) services and especially network managements (NWM). Moreover, Web integration can prevent these legacy applications from working only in isolation. Because of Web integration's simplicity, it leverages the enormous success of the Web browser to access services and information provide legacy applications in the Web.

In respect of a network management, it refers to a set of activities, methods, procedures and tools that relates to the operation, administration, maintenance and provisioning of computer networks. Therefore, it plays a very important role in dealing with keeping computer networks running smoothly. Because of its simple design, SNMP (Simple Network Management Protocol) is widely used for the network management on the Internet around the world today to enable a management station to configure, monitor and receive Trap messages from SNMP-enabled network devices. For example, hubs, bridges, routers, printers are manufactured by most major vendors of networking hardware. Due to this importance role of network management in general and the SNMP protocol in particular, XML-based network management approaches based on different communication technologies such as XML-RPC, SOAP or HTTP-based using URI extension with XPath and XQuery have been proposed as the alternatives to traditional SNMP-based network management tools because XML-based network management tools can confront better with

![Figure 1.1: An importance of integrating legacy applications into the Web](image-url)
scalability and integrability issues in expand heterogeneous network. Nevertheless, the main limitation of the SOAP-based approach is the overhead of packaging SOAP messages and it is too powerful in some case. The issue of the two other approaches is that they must confront with the complexity of the implementation of both components at the server side and the client side. Moreover, these above-mentioned network management approaches lack of abilities to simply integrate with modern Web applications which offer a variety of benefits.

Nowadays, REST (Representation State Transfer) has become a very popular term in the Web. In terms of building a Web service, a REST-based approach might provide equivalent functionalities in comparison with alternative approaches such as SOAP-based or XML-RPC-based regarding of communication power. In other respects, REST offers a lot of advantages over these two approaches such as lightweight, simplicity, scalability and evolvability without breaking backwards or forwards compatibility and better network performance. Moreover, it is really easy to integrate a REST-based Web service into modern Web clients like Ajax-based Web clients because of the following arguments. Firstly, Ajax-based Web clients are embedded inside Web browsers. Secondly, the Web browser environment strengthens the arguments for REST because REST paradigm is perfectly suitable for facilitating communication between Web browsers and Web servers. Therefore, a REST-based approach is the ideal alternative to the above-mentioned network management approaches in providing a perfect solution for the SNMP infrastructure integrating into modern Web applications.

1.2 Thesis Goals

The primary goal of this Master thesis is to show possibilities of integrating any legacy application like the SNMP infrastructure into REST. It leads to develop a Web-based network management solution to access SNMP-enabled devices. The Web application can be implemented practically on a basis of Hypertext Transfer Protocol (HTTP) and other Internet technologies such as Hypertext Markup Language (HTML), Universal Resource Identifier (URI) and so forth. In addition, there are essential differences between the HTTP and the SNMP protocols. Hence, the gateway which bridges these two different networks protocols is required to translate HTTP requests to SNMP operations. In order to improve scalability, simplicity and efficiency of the proposed solution, the gateway that offers a generic interface will be defined and implemented based on constraints of Representation State Transfer (REST). It covers all available operations from simple ones such as Get and Set operations to complex ones like Get-Bulk. SNMP Trap is an important part of the SNMP protocol because it enables an agent to notify a management station of significant events. Instead of that, the management station has to poll or request information from every object on every device. Especially, it is impractical if the management station is responsible for a large number of devices while each device has a large number of objects. Therefore, the implementation of the REST-based gateway must contain a module that
provides a possibility of receiving Trap messages from agents and converting these original SNMP Trap messages to a format that Web clients can understand and eventually forward them to Web clients. For the exchange of information between Web clients and the gateway, a XML-based protocol must be introduced. To access the gateway, a Web 2.0 client based on asynchronous HTTP communication has to be implemented. It offers basic functionalities realizing configuration and monitoring tasks of the underlying SNMP-managed infrastructure. Finally, the central goal regarding all parts of the implementation is efficiency. The thesis goals can be concretely summarized as follows:

- Showing a possibility of integrating many legacy applications into REST.
- Specification and implementation of REST-based SNMP gateway.
- Implementation of the uniform interface based on the constraints of REST.
- Available operations of SNMPv2c such as Get, Get-Next, Get-Bulk, Set, Trap provided by the gateway.
- Implementation of a Web 2.0 network management application for accessing SNMP information.
- Scalability, interoperability, simplicity and efficiency of the proposed solution.

1.3 Structure of Thesis

The structure of this Master thesis include six chapters with the first one as an introduction. The chapter two gives readers a closer look at fundamental technologies such as REST, SNMP, Ajax and Comet. Furthermore, arguments about advantages and disadvantages of each technology will be rendered as a part of this chapter in order to give readers a better understanding why these technologies are superior to other competitive technologies in the current market. This chapter also cites literatures from reputable and appropriate sources in order to introduce the background and rationale for the present thesis, to state how relevant previous research is related to this study and to find out which methods have been tried to solve it. Especially, the chapter is concluded with a summary of the previous research results as a basis for the thesis to develop further and differently. The third chapter presents a general approach of the mentioned problem stating how to integrate the Web-based network management application to the SNMP infrastructure via the REST-based gateway. After that, the overall architecture of the whole system which is tightly bound to constraints of REST will be proposed. Note that the proposed solution in the third chapter should be abstract and completely independent from any specific technologies, programming languages and frameworks, so that different detailed implementations can be carried out based on the generic and abstract approach. Afterwards, chapter four discloses arguments why some specific programming languages, technologies and frameworks, such as Java, Javascript, HTML, Comet and Ajax, can be selected to implement the REST-based SNMP gateway and the Web-based network management application. Afterwards, an insight look at a detailed implementation of the gateway and the Web client will be elaborated in the last section of this chapter. The main
purpose of chapter five is to judge functional accomplishments and achievements of the proposed solution whether or not they fulfill the thesis's goals and the system's requirements that have been stated in the previous chapters. The last chapter of the thesis concludes what have been accomplished in the thesis and exemplifies some proposed possibilities of the future works.
Chapter 2 Technological Overview and State-of-the-Art

As it is mentioned, the thesis aims to implement the generic interface of the gateway based on the constraints of REST. Thus, section 2.1 of this chapter is started with an overview of the REST architecture approach, stating what REST actually is, how it relates to the Web, as well as how it relates to different standards, such as SOAP, XML-RPC, that govern the Internet today. The next section 2.2 is devoted to present essential concepts of SNMP (Simple Network Management Protocol) because SNMP is a typical example of legacy applications which can be integrated to Web technologies via the REST-based SNMP gateway. In section 2.3 and 2.4, mechanisms together with the pros and cons of Ajax and Comet will be elaborated. Ajax and Comet which are arguably the most popular technologies on the Web will be used to implement Web-based network management application and a module of the gateway receiving and sending Trap messages to Web clients respectively. Finally, some related works will be presented in section 2.5.

2.1 REST

The section is dedicated to bring out all basic aspects of the REST architecture style.

2.1.1 REST Background

Representational State Transfer (REST) is a style of a software architecture for distributed hypermedia systems like the World Wide Web. The terms “Representational State Transfer” and “REST” were coined in 2000 in the doctoral dissertation of Roy Fielding [45], one of the principal authors of the Hypertext Transfer Protocol (HTTP) specification, to describe the existing architecture of the Web. REST is in deed an architectural style, not a standard or an implementation specification. The largest REST application is the Web itself, characterized by the use of HTTP for transport and URI as addressing mechanism. REST can support any type of media while XML is the most popular method used to transport and represent structured information. REST is used with HTML, XHTML, RSS and proprietary XML vocabularies.

There are several basic elements involved in REST. Firstly, data elements are resources (e.g. data objects), resource identifiers (e.g. network addresses, URI), and representations of resources (e.g. HTML documents, JPEG images). Secondly, components can be origin servers, gateways, proxies and user agents. These components communicate by transferring representations of resources through the interface, not by operating directly on the resources themselves. Thirdly, connectors which are clients, servers and caches, together with tunnels such as sockets and SSL connections, present an abstract interface for communication and hiding the implementation details of communication mechanisms.

Additionally, REST contains some essential constraints or principles, namely client/server constraint, stateless constraint, caching constraint and especially uniform
interface constraint. Stateless constraint means that every request of connectors must contain all the necessary information to understand it without depending on any previous request. Therefore, this fact contrasts with the way many Web sites using cookies to maintain data between sessions. With REST, all messages must include all information necessary for understanding the context of an interaction. Uniform interface constraint means that data elements of REST are accessed through a standardized interface which is done by using well-defined HTTP operations such as Get, Post, Put and Delete. Systems that follow these Fielding’s REST principles can be called REST-full.

2.1.2 Stateless Client-Server

Since the stateless client-server style derives from client-server with the additional constraint, no session state is allowed on the server component. In a stateless application, the server considers each client request in isolation and in terms of the current resource state. Interaction is stateless when a single request is processed without knowing which requests have been made previously. Therefore, if a client wants any application state to be taken into consideration, the client must submit it as part of the request. For example, authentication credentials are submitted with every request.

This stateless constraint has many advantages and disadvantages. On one side, it induces the properties of scalability. Scalability is improved because it does not have to store state between requests allowing the server component to quickly free resources, and further simplifies implementation because the server does not have to manage resource usage across requests. On the opposite side, the stateless constraint's design trade-off is that it might decrease network performance by increasing the repetitive data (per-interaction overhead) sent in a series of requests. Hence, data cannot be left on the server in a shared context.

2.1.3 Resource and Resource Identifier

The key abstraction of information in REST is a resource. A resource is anything that is important enough to be referenced as a thing in itself. A resource is something that can be stored on a computer and represented as a stream of bits: a document, a row in a database, or the result of running an algorithm. A resource may be a physical object like an apple, or an abstract concept like courage.

Resources are named with URIs (Universal Resource Identifier). It has to have at least one URI. If a resource has multiple URIs, Web clients have different ways to refer to the resource. In other words, a URI is the name and the address of a resource. According to principles of REST, if a piece of information does not have a URI, it will not be a resource and not on the Web, except for a bit of data describing some other resource.

2.1.4 Resource Representations

A resource is a source of representations. A current state of a resource can be
captured by its representation. In other words, a representation is just some data about the current state of the requested resource, the desired state for the requested resource, or the value of some other resources, such as a representation of the input data within a client's query form, or a representation of some error condition for a response. A representation comprises data, meta data describing the data, and itself on occasion. Meta data is in the form of name-value pairs, where the name corresponds to a standard that defines the value's structure and semantics.

Representations should be human-readable, but computer-oriented. The job of the human Web is to present information for a direct human consumption. Representations should be useful: that is, they should expose interesting data instead of irrelevant data that no one will use. A single representation should contain all relevant information necessary to fulfill a need. A client should not have to get several representations of the same resource to perform a single operation.

2.1.5 Uniform Interface

The central feature that distinguishes the REST architectural style from other network-based styles is its emphasis on a uniform interface between components. All interaction between clients and resources is mediated through a few basic generic methods. In terms of Web technologies, HTTP will be employed as a transport protocol to deliver data between a REST-based server and its Web clients.

An HTTP Get request is used to retrieve a representation of a resource. The information is delivered as a set of headers and a representation. A Web client never sends a representation along with a Get request.

To create or modify a resource, a Web client sends an HTTP Put request that usually includes an entity-body. The entity-body contains the client’s proposed new representation of the resource.

A Post request is an attempt to create a new resource from an existing one. The existing resource may be the parent of the new one. In a data structure sense, the root of a tree is the parent of all its leaf nodes. Or the existing resource may be a special “factory” resource whose only purpose is to generate other resources. The representation sent along with a Post request describes the initial state of the new resource.

A Delete request is an assertion that a resource should no longer exist. The Web client never sends a representation along with a Delete request.

2.1.6 Advantages of REST

A Web service using REST have many advantages over the well-known SOAP (Simple Object Access Protocol) approach, which requires writing or using a provided server program to serve data and a client program to request data. First of all, it is more lightweight because it does not require a lot of extra XML markups. Secondly, REST does not require a separate resource discovery mechanism, due to the use of hyper-links in
content. Finally, it is easier to build REST-based Web service because there is no need for a complex development environment.

The REST representation caching constraint provides improved response times and server loading characteristics. Its stateless constraint improves server scalability by reducing the need to maintain communication state. This means that different servers can be used to handle initial and subsequent requests. This constraint also provides reliability because it is easier to recover from failures.

The REST architecture style provides equivalent functionality in comparison with the alternative approaches such as SOAP or XML-RPC in terms of communication power between client and server. However, REST Web service requires less client-side software to be written than the other approaches because a single browser can access any application and any resource. Therefore, any distributed system committing to REST constraints depends less on vendor software and mechanisms which layer additional messaging frameworks on top of HTTP.

Due to the capability of document types like HTML to evolve without breaking backwards- or forwards-compatibility, REST provides better long-term compatibility and abilities to be evolved than XML-RPC or SOAP.

Finally, other benefits of REST can be named as network effects due to ubiquitous resource identification and uniform operation semantics, scalability through layering, representation caching, and stateless interaction.

2.2 SNMP Technology

This section is dedicated to illustrate most of fundamental aspects of SNMP technology.

2.2.1 SNMP Background

To alleviate the exchange of management information between network devices, the Simple Network Management Protocol (SNMP) as an application-layer protocol forms part of the Internet protocol suite which has been defined by the Internet Engineering Task Force (IETF). SNMP is used in network management systems to monitor network-attached devices, manage network performance, find and solve network problems, as well as plan for network growth. SNMP is a part of a larger architecture called the Internet Network Management Framework (NMF), which is defined on Internet documents called Requests For Comments (RFCs).

There are three versions of SNMP: version 1, version 2 and version 3. SNMP version 1 (SNMPv1) is the oldest standard version of the SNMP protocol. It is defined on RFC 1157 [19] as a full IETF standard. SNMPv1’s security is based on community strings which play roles as passwords while community strings allow any SNMP-based application knowing the strings to gain access to a device's management information. There are three kinds of community strings in SNMPv1 including Read-Only, Read-Write,
SNMP version 2 is technically called SNMPv2c, but it is referred throughout this section simply as SNMPv2. It is defined in RFCs [1905-1907] [17]. Even it is only at the experimental state, some vendors have started supporting it in practice. SNMP version 3 (SNMPv3) will be the next version of the protocol to reach the full IETF status. It is currently a proposed standard, defined in RFCs [1905-2575] [17]. It adds supports for strong authentication and private communication between managed entities.

The core of SNMP is a simple set of operations and the information collected by these operations enables network administrators to change the state of some SNMP-based devices. For example, administrators can use SNMP to shut down an interface on a router or check the speed at which an Ethernet interface is working on.

Because it is a simple solution, requiring little code to implement, vendors can easily build SNMP agents to their products. SNMP is extensible, allowing vendors to easily add network management functions to their existing products. SNMP also separates the management architecture from the architecture of the hardware devices, which broadens the base of multi vendor support. Perhaps most important, unlike other so called standards, SNMP is not a mere paper specification, but an implementation that is widely available today. This simplicity has led directly to the widespread use of SNMP, specifically in the Internet Network Management Framework. Within this framework, it is considered to be robust because of the independence of the SNMP managers from the agents, for instance, if an SNMP agent fails, the SNMP manager will continue to function or vice versa.

2.2.2 Internet Management Model

A network management system comprises:

- **Network elements**: Sometimes called managed devices. Bridges, hubs, routers or network servers are examples of managed devices that contain managed objects.
- **Agents**: Agents are the entities that offer interfaces to the actual device being managed. Agents reside in network elements. They collect and store management information such as the number of error packets received by a network element.
- **Managed objects**: These managed objects might be hardware, configuration parameters, performance statistics, and so on, that directly relate to the current operation of the device in question. In other words, managed objects are a characteristic of something that can be managed. Managed objects differ from variables, which are particular object instances. Managed objects can be scalar (defining a single object instance) or tabular (defining multiple, related instances).
- **Management information base (MIB)**: The managed objects are arranged in what is known as a virtual information database, called a management information base, or MIB. A group of related managed objects are defined in specific MIB modules.
- **Syntax notation**: A syntax notation is a language used to describe a MIB’s managed objects in a machine-independent format. Consistent use of a syntax notation allows different types of computers to share information. Internet management systems use
a subset of the International Organization for Standardization's (ISO's), Open System Interconnection (OSI) and Abstract Syntax Notation 1 (ASN.1) to define both packets exchanged by the management protocol and the objects that are to be managed.

- Structure of Management Information (SMI): The SMI defines the rules for describing management information. The SMI is defined using ASN.1.
- Network management station (NMS): Sometimes called console. These devices execute management applications that monitor and control network elements. Physically, NMSs are usually engineering workstation-caliber computers with fast CPUs, megapixel color displays, substantial memory, and abundant disk space.
- Management protocol: A management protocol is used to convey management information between agents and NMSs. SNMP is in practice the Internet community's standard management protocol.

The most basic elements of the Internet management model are graphically shown in Figure 2.1 below.

![Figure 2.1: Internet Management Model](image)

A typical agent usually implements full SNMP protocol, stores and retrieves management data as defined by the Management Information Base. An SNMP agent can asynchronously signal event to the manager. A manager is implemented as a Network Management Station (NMS). A manager is able to query agents, get responses from agents, set variables in agents and acknowledge asynchronous events from agents.

2.2.3 Layered Communication Model

In this section, the Simple Network Management Protocol (SNMP) focusing specifically on the layered communication model used to exchange information will be
SNMP domiciles in the Application layer, UDP resides in the Transport layer and IP domiciles in the Internet layer. The fourth layer is the Network Interface layer where the assembled packet is actually interfaced to some kind of transport media.

To depict the function of this layered model, a single SNMP request from the agent's perspective will be described. For example, an SNMP manager desires to know what one MIB managed object is. First of all, it prepares a Get message for an appropriate OID. It then hands the message to the UDP layer. The UDP layer appends a data block that keys out a manager port to which the response packet should be sent and a port on which it expects an SNMP agent to be listening for messages. After that, the formed packet is handed to the IP layer. Before the entire assembled packet gets passed to the Network Interface layer, a data block containing the IP and Media Access addresses of the manager and the agent is added at the IP layer. The Network Interface layer checks media access and availability and locates the packet on the media for transport. The packet finally reaches the agent after working its way across bridges and through routers based on the IP information.

It goes through the same four layers in exactly the reverse order as it did at the manager. Firstly, it is drawn off the media by the Network Interface layer. After sustaining that the packet is inviolate and valid, the Network Interface layer simply gives it to the IP layer. After the IP address block of the packet is verified, the IP layer passes it to the UDP layer. At the UDP layer, the target port is checked for connected applications. If the application is the agent listening at the target port, the packet is passed to the agent. After the agent processes the request, its response then follows the indistinguishable path in opposite to reach the manager.

Understanding this layered communication model allows readers to get to know why UDP was chosen over the TCP because it is connectionless; that is, no end-to-end connection is made between SNMP agents and NMS (Network Management Station) when datagrams sent back and forth. UDP is unreliable since there is no acknowledgment.
of lost datagrams at the protocol level. The upside of this unreliable nature of UDP is that it requires low overhead, so that the overall network performance is increased. On other respects, SNMP over TCP is a bad idea because it causes a heavily congestion in the network. Hence, it is important for developers to select UDP over TCP while carrying out programming tasks with SNMP. It is also worth to know that SNMP uses UDP port 161 for sending and receiving requests and port 162 for receiving Traps from managed devices.

2.2.4 MIBs and Object Identifiers

A MIB can be depicted as an abstract tree with an unnamed root. Individual data items make up the leaves of the tree. Object Identifier (OID) uniquely identifies or names MIB objects in the tree. An Object Identifier is an administratively assigned name. Each type of MIB objects has a name, a syntax, and an encoding.

MIB objects or managed objects are organized into a treelike hierarchy. This structure is the basis for SNMP's naming scheme. An object ID is made up of a series of integers based on the nodes in the tree, separated by a dot ("."). Although there is also a human-readable form that is friendlier than a string of numbers, this form is nothing more than a series of names separated by dots, each representing a node of a tree. SMIv2 extends the SMI Object Tree defined in SMIv1 by adding the snmpV2 branch to the internet subtree and by adding several new data types and making a number of other changes.

In the object tree, the node at the top is called the root, anything with children is called a subtree, anything without children is called a leaf node. For example, in figure 2.3, ccitt(0) and joint(2) are the leaf nodes. Conversely, iso(1) is the subtree. For the remainder of this chapter, the iso(1).org(3).dod(6).internet(1) subtree, which is represented in OID form as 1.3.6.1 or as iso.org.dod.internet, will be discussed. As it is mentioned, each managed object has a numerical OID and an associated textual name. The dotted-decimal notation describes how a managed object is represented internally within an agent. The textual name, like an IP domain name, saves humans from having to remember long, tedious strings of integers. The directory branch currently is not used. The management branch, or mgmt, defines a standard set of Internet management objects. The experimental branch is reversed for testing and research purposes. Objects under the private branch are defined unilaterally, which means that individuals and organizations are responsible for defining the object under this branch. Here is the definition of the internet subtree, as well as all four of its subtrees [20]:

```plaintext
internet OBJECT IDENTIFIER ::= {iso org (3) dod (6) 1}
directory OBJECT IDENTIFIER ::= {internet 1}
mgmt OBJECT IDENTIFIER ::= {internet 2}
experimental OBJECT IDENTIFIER ::= {internet 3}
private OBJECT IDENTIFIER ::= {internet 4}
```
The first line declares `internet` with the OID 1.3.6.1, that is defined as a subtree of `iso.org.dod`, or 1.3.6. The last four declarations are similar, but they define the other branches that belong to `internet`. For the `directory` branch, the notation `{internet 1}` tells us that it is part of the `internet` subtree and its OID is 1.3.6.1.1. The OID for `mgmt` is 1.3.6.1.2 and so on. The `mib-2` branch represents MIB-2. MIB-2 is very important management group because every device that supports SNMP must also support MIB-2. Mib-2 is defined as `iso.org.dod.internet.mgmt.1`, or 1.3.6.1.2.1. The mib-2 subtree looks like this [20]:

```
mib-2  OBJECT IDENTIFIER ::= {mgmt 1}
system  OBJECT IDENTIFIER ::= {mib-2 1}
interfaces OBJECT IDENTIFIER ::= {mib-2 2}
at  OBJECT IDENTIFIER ::= {mib-2 3}
ip  OBJECT IDENTIFIER ::= {mib-2 4}
icmp  OBJECT IDENTIFIER ::= {mib-2 5}
tcp  OBJECT IDENTIFIER ::= {mib-2 6}
udp  OBJECT IDENTIFIER ::= {mib-2 7}
egp  OBJECT IDENTIFIER ::= {mib-2 8}
transmission OBJECT IDENTIFIER ::= {mib-2 10}
smnp  OBJECT IDENTIFIER ::= {mib-2 11}
```

There is at the moment one branch under the `private` subtree. It is used to give hardware and software vendors the ability to define their own private objects for any type of hardware and software they want to manage by SNMP. Its SMI definition is:

```
enterprises OBJECT IDENTIFIER ::= {private 1}
```
2.2.5 SMI Definitions

The SMI specifies that all managed objects should have a name, a syntax, and an encoding. The name is the object ID, which was discussed in the preceding section. The syntax defines the object's data type (for example Integer or String). A subset of ASN.1 definitions are used for the SMI syntax. The encoding, which describing how the information associated with the managed object, is formatted as a series of data items for transmission on the network. Another ISO specification, called the Basic Encoding Rules (BERs), details SMI encodings. SNMP data types will be summarized in the table 2.1.

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integers</td>
<td>Unique values that are positive or negative numbers, including zero.</td>
</tr>
<tr>
<td>Octet strings</td>
<td>Unique values that are an ordered sequence of zero or more octets.</td>
</tr>
<tr>
<td>Counters</td>
<td>Non-negative integers that increment by positive one until they reach a maximum value. The total number of bytes received on an interface is an example of a counter. In SNMPv1, counter size was not specified. In SNMPv2, 32-bit and 64-bit counters are defined.</td>
</tr>
<tr>
<td>Object IDs</td>
<td>Unique values from the set of all object identifiers allocated according to the rules specified in ASN.1.</td>
</tr>
<tr>
<td>Sequence</td>
<td>Defines lists that contain zero or more other ASN.1 data types.</td>
</tr>
<tr>
<td>Sequence of</td>
<td>Defines a managed object that is made up of a sequence of ASN.1 types.</td>
</tr>
<tr>
<td>Network Addresses</td>
<td>Represent an address from a particular protocol family.</td>
</tr>
<tr>
<td>Gauges</td>
<td>Non-negative integers that can increase or decrease, but latch at a maximum value. The length of an output packet queue (in packets) is an example of a Gauge.</td>
</tr>
<tr>
<td>Time ticks</td>
<td>Represents hundreds of a second since an event occurs. The time since an interface entered its current state is an example of a Time ticks.</td>
</tr>
<tr>
<td>Opaque</td>
<td>Represents an arbitrary encoding. This data type is used to pass arbitrary information strings that do not conform to the strict data typing used by the SMI.</td>
</tr>
<tr>
<td>Unsigned32</td>
<td>Represents unsigned integer-valued information. It is useful when values are always non-negative. This data type redefines the ASN.1 Integer simple data type, which has arbitrary precision in ASN.1 but bounded precision in the SMI.</td>
</tr>
<tr>
<td>Counter64</td>
<td>Similar to Counter32. Counter64 is ideal for situations in which a Counter32 may wrap back to 0 in a short amount of time.</td>
</tr>
<tr>
<td>Bit strings</td>
<td>New in SNMPv2, these comprise zero or more named bits that specify a value.</td>
</tr>
</tbody>
</table>

Table 2.1: SNMP Data types [20].
2.2.6 SNMP Operations

In the previous section, it has been discussed how SNMP organizes information. However, that how the management information can be gathered has been left out. In this section, a closer look under the hood to see how SNMP does its things will take place.

The Protocol Data Unit (PDU) is the message format that managers and agents use to send and receive information. There are some standard PDU formats: Get, Get-Next, Get-Bulk (SNMPv2c and SNMPv3), Set, Get-Response, Trap, Notification (SNMPv2c and SNMPv3), Inform and Report (SNMPv2c and SNMPv3). The SNMP operations can be named according to its PDU formats.

An SNMP Get request is generated by a NMS, which sends the request to an agent. The agent receives the request and processes it. If the agent is successfully in gathering the requested information, it sends a Get-Response back to the NMS, where it will be processed. One of the items in the Get request is a variable binding. A variable binding, or Varbind for short, is a list of MIB objects that enables a request's recipient like agents to understand what the NMS desires to know. Variable bindings can be thought of as OID=value pairs. The NMS can pick out the information that it needs from variable bindings when the agent sends back a response to it.

The Get-Next operation lets network administrators issue a sequence of commands to retrieve a group of values from the MIB. To retrieve each MIB object, a separate Get-Next request and Get-Response are generated. The Get-Next command traverses a subtree in lexicographic order. It is easy for an agent to begin at the root of its SMI Object Tree and work its way down until it finds the OID which is looking for because an OID is a sequence of integers. NMS issues another Get-Next command when it receives a response from the agent for the Get-Next command which is just issued. It continues this operation until the agent returns an error, denoting that the end of the MIB has been reached and there are no more objects left to get.

SNMPv2c and SNMPv3 define the new Get-Bulk operation. The Get-Bulk operation is normally used for retrieving large amount of data, particularly from large tables. A Get-Bulk request is made by giving an OID list along with a Max-Repetitions value and a Non-repeaters value as input parameters. The Get-Bulk operation performs a continuous Get-Next operation based on the Max-Repetitions value. The Non-repeaters value determines the number of variables in the variable list for which a simple Get-Next operation has to be done. For the remaining variables, the continuous Get-Next operation is done based on the Max-Repetitions value. The Get-Bulk operation has many advantages over the standard Get operation. The standard Get operation can try to get more than one MIB object at once, but message sizes are restricted by an agent's capabilities. If the agent cannot return all the requested responses, it returns an error message with no data. The Get-Bulk operation orders the agent to send as much of the response back as it is capable to do. This means that incomplete responses are possible.

The SNMP Set operation is used by the NMS to modify values of managed
objects. Most of the managed objects have a default value maintained by agents. Sometimes the applications might want to modify one or more MIB variables by using the SNMP Set operation. The application typically performs an SNMP Set operation by providing the host name of the agent, one or more OIDs along with their instances, and the new values. The agent processes the request and assigns the new values to the MIB variables. If an error occurs, the new values are not assigned.

The agent, upon receiving a Get or Get-Next message, will issue a Get-Response message to the SNMP manager with either the information requested or an error indicating why the request cannot be processed.

A Trap message allows an agent to spontaneously inform an SNMP manager of important events. The agent initiates a Trap message and sends it to the Trap destination which is configured within the agent itself. In practice, the IP address of the NMS is normally configured as a Trap destination. The agent has no way of realizing whether the Trap reaches the NMS because no acknowledgment is sent from the NMS to the agent. As SNMP uses UDP and since Traps are designed to report problems within a network, Traps are especially prone to getting lost and in some cases, Trap messages cannot reach their destinations. SNMP Inform is a new operation for SNMPv2. The Inform operation was added to allow one NMS to send Trap information to another NMS.

### 2.2.7 SNMP Message Format

In this section, the structure of an SNMP message will be described. An SNMP message format is shown in Figure 2.4.

![Figure 2.4: SNMP Message Format](20)

The first part contains a version of SNMP and a community name that acts as a password. The second part contains the actual SNMP Protocol Data Unit (PDU) specifying the operation to be performed (e.g., Get, Set and so on) and the object instances involved in the operation. Table 2.2 elaborates descriptions of all fields of an SNMP packet.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP Version</td>
<td>An Integer that identifies the version of SNMP.</td>
</tr>
<tr>
<td>SNMP Community String</td>
<td>An Octet String that may contain a string used to add security to SNMP devices.</td>
</tr>
<tr>
<td>SNMP PDU</td>
<td>An SNMP PDU contains the body of the SNMP message. There are several types of PDUs. Three common PDUs are Get-Request, Get-Response and Set-Request.</td>
</tr>
<tr>
<td>Request ID</td>
<td>An Integer that identifies a particular SNMP request. This index is echoed back in the response from the SNMP agent, that allows the SNMP manager to match an incoming response to the appropriate request.</td>
</tr>
</tbody>
</table>
| Error Status          | An Integer set to 0x00 in the request sent by the SNMP manager. The SNMP agent places an error code in this field in the response message if an error occurred processing the request. Some error codes include:  
  • 0x00 -- No error occurred  
  • 0x01 -- Response message too large to transport  
  • 0x02 -- The name of the requested object was not found  
  • 0x03 -- A data type in the request did not match the data type in the SNMP agent  
  • 0x04 -- The SNMP manager attempted to set a read-only parameter  
  • 0x05 -- General error (some error other than the ones listed above) |
| Error Index           | If an error occurs, the error index will hold a pointer to the object that caused the error. Otherwise, the error index will be 0x00. |
| Variables bindings    | A sequence of Varbinds.                                                                                |
| Varbind               | A sequence of two fields, an Object ID and the value for/from that Object ID.                          |
| Object Identifier (OID) | An Object Identifier that points to a particular parameter in the SNMP agent.                         |
| Value                 | Set-Request PDU: Value is applied to the specified OID of the SNMP agent.  
                        | Get-Request PDU: Value is a Null that acts as a placeholder for the returned data.  
                        | Get-Response PDU: The returned value from the specified OID of the SNMP agent. |

Table 2.2: Fields in the SNMP message [20]

### 2.3 Ajax

Ajax is an acronym of Asynchronous Javascript and XML. It can be thought of as a group of Web development techniques used for creating interactive Web applications or rich Internet applications. In this section, general mechanisms of Ajax as well as some well
known Ajax toolkits and frameworks will be elaborated.

2.3.1 Mechanisms

Under a close look at the technical details, Ajax refers to the technique of using Javascript (specifically, the XMLHttpRequest object) to request data asynchronously, then dynamically updating a Web page with the requested data. In fact, Ajax is not a single technology. Rather, it comprises four distinct technologies that complement each other. Table 2.3 summarizes these technologies and the role that each one has to play.

<table>
<thead>
<tr>
<th>Technologies</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Javascript</td>
<td>Javascript is a general-purpose scripting language designed to be embedded inside applications. The Javascript interpreter in a Web browser allows programmatic interaction with many of the browser’s inbuilt capabilities. Ajax applications are written in Javascript.</td>
</tr>
<tr>
<td>Cascading Style Sheets (CSS)</td>
<td>CSS offers a way of defining reusable visual styles for Web page elements. It offers a simple and powerful way of defining and applying visual styling consistently. In an Ajax application, the styling of a user interface may be modified interactively through CSS.</td>
</tr>
<tr>
<td>Document Object Model (DOM)</td>
<td>The DOM presents the structure of Web pages as a set of programmable objects that can be manipulated with Javascript. Scripting the DOM allows an Ajax application to modify the user interface on the fly, effectively redrawing parts of the page.</td>
</tr>
<tr>
<td>XMLHttpRequest object</td>
<td>The XMLHttpRequest object allows Web programmers to retrieve data from the Web server as a background activity. The data format is typically XML, but it works well with any text-based data. While XMLHttpRequest is the most flexible general-purpose tool for this job, there are some other ways of retrieving data from the server which however will not be covered in this chapter.</td>
</tr>
</tbody>
</table>

Table 2.3: Summary of Ajax technologies [33]

The following graphic shows how these technologies work together to update a piece of a page with new data from the server.
1. A Javascript call is a result of an event that can be generated when the user clicks a button.

2. The browser client implements a Javascript HTTP client library called XMLHttpRequest. To build an HTTP request, the XMLHttpRequest object is created and configured with a request parameter that includes the ID (Identification) of the component that generated the event and any value that the user might have entered. This simple task is actually one of the major differences between the Web browsers. The constructor works in Mozilla-family browsers like Firefox: request = new XMLHttpRequest().

3. The next step is to call the open method of XMLHttpRequest with the request's information: request.open([HTTP method], [URI], true, [Basic authentication user name], [Basic authentication password]). A server-side object, like a Java Servlet, receives the request, processes it, and stores any data in the request to the database.

4. The server-side object that processed the request returns an XML document to the browser client. At this point, the XMLHttpRequest instance gains some information that can be used to update the browser client:
   - The status property contains the numeric status code for the request.
   - The responseXML property contains a pre-parsed DOM object representing the response document, assuming that it was served as XML and the browser can parse it. HTML, even XHTML, will not be parsed into responseXML, unless the document was served as an XML media type like application/xml or application/xhtml+xml. The responseText property instead contains the response document as a raw string which is useful in JSON (Javascript Object Notation) and some other non-XML formats.

5. The XMLHttpRequest object eventually processes returned data in XML or non-
2.3.2 Ajax Frameworks

Ajax frameworks are browser-side frameworks very commonly used in Ajax development. Nowadays, hundreds of Ajax frameworks are available in the market. However, in this section, only the most long-familiar Ajax frameworks will be introduced.

2.3.2.1 Layers of Ajax Frameworks

Many people consider the words such as framework, library, and toolkit as synonymous. This is true in the sense that they are all descendants of the same parent. However, the obvious difference existing between these concepts needs to be clarified. A framework term is defined originally in an area of software development as a support structure in which other projects can be organized and developed. A framework typically comprises several smaller components, namely supporting programs, libraries, and a scripting language. A library is determined as a set of related functions and procedures used to develop software. A toolkit is generally used in reference to graphical user interface (GUI) toolkits. In practice, a toolkit is a library that is mainly focused on creating a GUI. However, there also exist remoting toolkits that focus on providing communication functionalities. Therefore, the toolkits might also define constraints for the server-side.

Figure 2.6 clarifies properly the distinct layers of Ajax. The lowest level of the Ajax helpers is a remoting toolkit. The remoting toolkit wraps XMLHttpRequest or XHR for short with its own API to make a life easier. Good remoting toolkits should at least enable developers to avoid writing tedious codes, for instance an ugly statement try/catch or an XHR instantiation code. Additionally, the remoting toolkits should provide possible ways to fake an XHR support, for example by using a hidden iframe, in case browser clients do not support XMLHttpRequest. Figure 2.6 lists a handful of such remoting toolkits, namely DWR (Direct Web Remoting) [36], JSON-RPC [37] and Dojo [13]. DWR couples a
Javascript client library with a server-side listener that allows Javascript in a browser to interact with a Java program on a server. JSON-RPC itself has various bindings for many back-end languages. JSON-RPC-Java is a dynamic JSON-RPC implementation in Java. It allows a browser client to transparently call server-side Java code from Javascript with an included lightweight JSON-RPC Javascript client. Dojo is the very well-known toolkit and the fundamental functions of Dojo will be elaborated in subsection 2.3.2.2.

The UI (User Interface) toolkits that place above remoting toolkits give Web developers abilities to use rich widgets like trees, tabbed panes and menus. These are self-contained, instantiable UI components that can be used to compose rich Web pages. Additionally, some UI toolkit like Smartclient [38] aims to provide widgets that can be used to build a UI which looks and feels the same as a native application on Windows or Mac OS [36]. Some well-known UI toolkits can be listed here such as Dojo, Smartclient, Backcase [40], Prototype [28] and Yahoo! UI library [32].

At the top of the tower, there are various Web frameworks, namely Rails [41], Tapestry [42], Webworks [43] and ASP.Net [44]. This is a growing group which covers all of the platforms such as Java, .NET, Ruby, PHP, Python and Perl. The various frameworks offer different models about how Web developers can work with them to build Ajax-based Web applications.

2.3.2.2 Dojo Toolkit

Dojo is an open source toolkit written in Javascript that makes it easier to use Ajax to build great web applications. It was started by Alex Russell, Dylan Schiemann, David Schontzler, and others in 2004. It “allows you to easily build dynamic capabilities into web pages and any other environment that supports Javascript. Dojo provides components that let you make your sites more usable, responsive, and functional. With Dojo you can build degradable user interfaces more easily, prototype interactive widgets quickly, animate transitions, and build Ajax-based requests simply.” [13].

The Dojo toolkit is organized as a rich set of libraries, components, widgets and extensions. It is a set of three major layers: Dojo Core, Dijit and DojoX. This section serves as a guide to these layers, introducing some fundamental concepts of the Dojo toolkit which Web developers need to get familiar. Dojo Core comprises many packages that are layered in many levels. The bottom most layer is the Packaging System that enables Web developers to customize the distribution of Dojo for their applications. The language library like the Dojo Event System and the Language utilities reside on the top of Dojo Core's Package System. Table 2.4 shows functions of some main packages of Dojo Core that are most useful for Web developers.
<table>
<thead>
<tr>
<th>Package</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>dojo.io</td>
<td>Additional I/O transports. XMLHttpRequest (XHR) object is encapsulated in this package to make Ajax request easier.</td>
</tr>
<tr>
<td>dojo.event</td>
<td>Browser compatible event system.</td>
</tr>
<tr>
<td>dojo.lang</td>
<td>Language utilities.</td>
</tr>
<tr>
<td>dojo.widget</td>
<td>Contains all widgets that are available within the Dojo Core's namespace.</td>
</tr>
<tr>
<td>dojo.dnd</td>
<td>Drag and drop support.</td>
</tr>
<tr>
<td>dojo.animation</td>
<td>Create animation effects.</td>
</tr>
<tr>
<td>Dojo.rpc</td>
<td>Dojo remote-procedure-call resources.</td>
</tr>
<tr>
<td>Dojo.string</td>
<td>String utilities for Dojo.</td>
</tr>
<tr>
<td>Dojo.parser</td>
<td>The Dom/Widget parsing package.</td>
</tr>
<tr>
<td>Dojo.colors</td>
<td>Color utilities.</td>
</tr>
<tr>
<td>Dojo.data</td>
<td>A uniform data access layer.</td>
</tr>
<tr>
<td>Dojo.fx</td>
<td>Effect library on top of Base animations.</td>
</tr>
<tr>
<td>Dojo.i18n</td>
<td>Utility classes to enable loading of resources for internationalization.</td>
</tr>
<tr>
<td>Dojo.regexp</td>
<td>Regular expressions and Builder resources.</td>
</tr>
</tbody>
</table>

Table 2.4: Summary of the main Dojo Core's packages

Dijit is a widget system layered on top of the Dojo toolkit. Dijit comes bundled with its own theme like Tundra, which brings a common design and color scheme to all the widgets. Everything in Dijit is designed to be globally accessible to accommodate users with different languages and cultures as well as those with different abilities. Language translation, bi-directional text, and cultural representation of things like numbers and dates are all encapsulated within the widgets. Server interactions are done in a way that makes no assumption about local conventions. All widgets are keyboard accessible and using the standard Dijit theme, usable in high-contrast mode as well as by screen readers. As these features are baked in so that, all users are treated equally as much as possible. Users can use Dijit in one of two ways: declaratively by using special attributes inside of regular HTML tags, and programmatically through Javascript.

DojoX is an area for development of extensions to the Dojo toolkit. DojoX is managed by sub-projects which may have dependencies on Dojo and Dijit code or other sub-projects in DojoX. Since DojoX plays a role as an incubator for new ideas, one DojoX sub-project's status can vary from immature to stable. There are four different levels of DojoX sub-project's status such as experimental, alpha, beta and production. If a sub-project status is experimental, it may be new or a proof of concept. Hence, it can be removed without notice if its functions are highly unstable. Conversely, a production level indicates
that the sub-project is at a state of major release. A sub-project at a production level must follow the same conventions as Dojo Core and Dijit. The well-known library DojoX.Cometd is an example of DojoX sub-project that is coined by Alex Russel and Greg Wilkins[2]. The essential domain of its application is to transfer low-latency data from servers to clients. This library implements a Bayeux [2] protocol's client which is preferred to use in connecting with an existing Cometd server like Jetty [22].

### 2.3.3 Advantages of Ajax

The first advantage of Ajax is lower bandwidth usage. Some Web hosts have low bandwidth limitations per month, therefore decreasing the amount used for each application or each request is an essential target of many Web masters. Ajax application can diminish monthly bandwidth usage significantly because it only employs bandwidth to load the first request. Unlike normal applications, Ajax does not require each page to be refreshed or re-loaded each time when a user requests another process. As contents on each page are loaded in the background, the user can have fluid, effortless communication with the site.

The second Ajax is cross-browser and cross-platform compatibility. Ajax-based applications can be hosted on any browser, such as IE, Firefox, Safari and Opera and on any operating system, namely Windows, Mac and Linux. This property of Ajax enables it to overcome drawbacks of many other Web hosting technologies that are only available for some operating systems and some Web browsers.

The Ajax's third advantage is speed. Because Ajax generates HTML pages locally and it does not request useless data, pages built with Ajax load quicker than normal ones.

### 2.3.4 Disadvantages of Ajax

The first drawback of Ajax technology is the lack of a good browser integration. For example, clicking on the “forward/back” buttons to go to the next page or return to the previous page might not be allowed on a user’s browser. Furthermore, dynamic Web page updates, which are parts and parcels of the Ajax technology, also make the bookmarking to become difficult. Because Web pages are automatically updated without clicking on “refresh” button, it might not contain the desired information when a user return to a previous page.

Although it improves the user's experience, Ajax still uses the standard HTTP model, the Pull architecture. In the Pull architecture, a communication begins when a client makes a request for a resource to a server. The communication ends when the request is answered by a response from the server with the requested resources. If one client-server system used the Pull architecture, its performance would be suffered. The reason is that the network traffics arise significantly when the client repeatedly polls to see whether new data is available. To get rid off this problem, Ajax applications often use polling to detect new information on the server, but it often results in unacceptable loads on the server in case Ajax-based Web
clients have to poll data from the server repeatedly.

2.4 Comet

In this section, principal aspects of Comet will be shortly described.

2.4.1 Comet Background

To overcome the disadvantages of the traditional Pull architecture in Ajax, Alex Russell coined the neologism Comet to describe the evolving Push architecture of the Web [1]. The Push architecture enables a server to deliver data asynchronously to a client without the need for the client to explicitly request the data. Hence, the data is being sent as soon as it is available and it results a fast communication. The Comet term is also known as many other names such as Ajax Push, Reverse Ajax, HTTP Streaming and HTTP Server Push. Given the speed and usability as advantages of the Push architecture, Web developers investigate means by which the same functionality can be realized on the Web.

2.4.2 Comet Mechanism

The connection between a client and a server in a Comet application is established by using long-lived HTTP or HTTPS connections, through which the server can push new data to the client as it becomes available. A Comet server must keep an uninterrupted connection to any client for a specific duration of a session. The defined attributes of Comet differs its server from traditional Web servers, which deliver a single payload and then immediately close the connection to clients. The figure 2.7 describes the Comet data transfer mechanism. If the client has initiated a connection, the server may send updates as soon as they appear, without issuing further requests.

![Comet mechanism](image.png)

Figure 2.7: Comet mechanism

2.4.3 Long Polling Mechanisms

One variant of the HTTP Streaming technology can also be known as the Long Polling or Asynchronous Polling. This method is a hybrid of two models such as the pure Server Push and Client Pull. It is coined based on a specification of the Bayeux protocol.
Bayeux is a protocol for routing JSON encoded events between clients and servers in a publish/subscribe model. The protocol is designed to overcome the client/server nature of the internet in general and specifically of HTTP to allow asynchronous messaging between all participants [15]. Publish/subscribe, or Pub/Sub for short, is said to be an asynchronous message paradigm where senders (publishers) of messages are not programmed to send their messages to specific receivers (subscribers). Published messages instead are categorized into different classes without knowledge of existence of any subscriber. A subscriber expresses its interest in one or more classes and only receives messages within its interest, without knowing whether there exist publishers.

According to the Long Polling mechanism, after a subscription to a channel is made, the connection between a client and a server is kept openly for a defined amount of time. If no event occurs on the server side and a timeout occurs, the server will ask the client to reconnect asynchronously. If an event occurs, the server may send data to the client. Consequently, the client will reconnect to the server.

### 2.5 Related Works

In this section, the pros and cons of some prominent related works in the research area of integrating the SNMP infrastructure to an immediate gateway by using different communication technologies will be analyzed. On a basis of this analyzing, proposed improvements to overcome these disadvantages will be demonstrated.

Yoon-Jung Oh, Mi-Jung Choi, James W.Hong and Hong-Taek Ju [5] have proposed three interaction translation methods between an XML/SNMP gateway and the XML-based manager based on a specific translation algorithm. First of all, they proposed the DOM-based translation method that gives the manager a possibility to exchange management information with SNMP agents through a standard DOM interfaces. Afterward, they offered the HTTP-based translation using URI extension with XPath and XQuery. Finally, they proposed SOAP-based translation method. The manager can access management information provided by SNMP agents standing behind the gateway using SOAP RPC. However, each of this approach exposes its own drawback. The DOM-based approach imposes a burden on the manager of invoking a series of interfaces for request processing. The HTTP-based translation requires both the manager and the gateway must include the XPath and XQuery enabled module. The main limitation of the SOAP-based one is the overhead of packaging SOAP messages. In addition, the XML-based manager has not implemented with the cutting-edge technology such as Web 2.0. Therefore, it lacks of a variety of its nice features. The suggested solution in this thesis should overcome all drawbacks which are just claimed and should enhance quality of the User Interface (UI) of the Web-based management application supposed that advanced features of Web 2.0 are utilized.

Using Java Servlets and a library to access SMI MIB Information called “libsml” [6], Jens Müller [7] carried out an implementation of a XML/SNMP gateway that enables
retrieving MIB information addressed through XPath expression encoded in URL. However, the big issue of this work is a poor performance of the gateway. In addition, his implementation is not completed because the gateway lacks a module that receives and forwards Trap message to Web clients. With the implementation of the REST-based SNMP gateway in this thesis, the vital issue like performance will be avoided. Additionally, Web clients have a possibility to receive Trap messages via the gateway right after they occur.

Mazumdar Subrata at Avaya Labs [8] proposed concepts for an XML-based interface to interchange management information with SNMP agents. An XML-RPC protocol has been specified and developed in order to allow retrieving and modifying MIB information on SNMP agents. XPath is used in this protocol to name MIB variables within the agents. Unfortunately, his proposed concept has not been realized with an implementation.

Zhixia Zhao, Ziheng Liu, Yu Bai and Debao Xiao [11] designed and implemented an Universal Gateway for XML-based Network Management (XNM). Through the Universal Gateway, the XNM can manage networks equipped with SNMP agents and Non-SNMP Agents, namely TELNET, HTTP and so forth. The Universal Gateway in fact contains two parts such as a XML-SNMP gateway and XML-non-SNMP gateway. Among these two gateways, the XML-SNMP gateway which is implemented based on SOAP is used to translate XML form message from Web manager to SNMP form message. REST is claimed to be simpler and more lightweight than SOAP, therefore the purport of this thesis is to emphasize on improving the simplicity and efficiency of the suggested system.

CMIP, which was designed to compete with SNMP, has more features than SNMP. For example, SNMP defines only the Set action to modify the state of an SNMP-managed device but CMIP allows the definition of any type of actions. CMIP provides powerful capabilities that allow management applications to accomplish more with a single request. However, on the Internet, most TCP/IP devices support SNMP and not CMIP. Micheal Langer [10] in his Master thesis designed and implemented a CMIP/SNMP gateway. The management gateway integrates the SNMP management information to the OSI management architecture. Therefore, it provides an OSI-Manager the ability to access SNMP resources and improves the capability of the network management system. However, CMIP is very complicated to develop programs. Additionally, the cost of deploying, maintaining and operating a CMIP based network management system is expensive. In contrast, the REST-based network management solution proposed in this thesis accentuates the simplicity of developing, deploying and maintaining the REST-based Web service.

Robert McMillan [9] has proposed a REST-full approach to Web Services. He claimed that Thomsom Publishing Asia Pacific Group before deciding to apply REST for its Web-based typesetting service had considered SOAP-based solution. Finally, they chose REST-based approach because it offers superior performance, dependability and scalability to SOAP-based one. However, their approach just results a poor REST-full Web
service that does not offer any possibility to access any legacy application via the Web service. Taking the advantages of Web service based REST over the SOAP-based approach, the Master thesis aims to integrate Web-based network management application to the legacy SNMP infrastructure via a gateway acting as REST-like Web service.

Gerd Aschemann, Thomas Mohr and Mechthild Ruppert [18] presented preliminary ideas to integrate both Web-based and CORBA-based techniques into management platforms SNMP. They implemented a prototype of CORBA-SNMP gateway which covers the translation of the information model as well as the communication model. The transformation of information model requires a tool to generate a CORBA IDL specification from a given SNMP MIB. The Web interface used to access SNMP objects is implemented as a Java applet component embedded in an HTML page which interacts with the CORBA-SNMP gateway via ORB (Object Request Broker). However, the Web client and the gateway are tightly coupled. With REST-based approach, it is very practical to keep off such a very complex component to transform SNMP MIB into an intermediate language such as IDL. Moreover, REST-based Web service is loosely coupled with its Web clients. Therefore, they can operate independently of each other with no problems. This is an excellent fit when designing Web-based network management solution for accessing SNMP infrastructure in the World Wide Web, when all components are not always available.

In summary, this thesis aims to prove that it is possible to integrate any legacy application like SNMP infrastructure into a modern Web-based application via an intermediate gateway. In addition, the Web-based network management solution proposed in the thesis purposes to provide enhancements in comparison with similar proposed approaches in other related works in terms of the following aspects: flexibility, scalability, interoperability and efficiency.
Chapter 3 Architecture Overview

A detailed analysis of system requirements, in respect of the above-mentioned problem of integrating a Web-based network management application into the SNMP infrastructure through an intermediate gateway, is presented in the first section of this chapter. The next section demonstrates the general approach to the intermediate gateway's implementation in accordance with constraints of REST. Finally, the overall architecture of the whole system will be depicted in the last section.

3.1 Contribution and Requirements Analysis

As mentioned in section 1.1, the proposed solution in the Master thesis has to offer a possibility of integrating a legacy application like SNMP into the modern Web technology like Web 2.0 in order to take its various advantages. In order to manage network resources using HTTP, it is necessary to have an application which speaks both HTTP and SNMP. This can be achieved in two ways as follows:

1. Extending standard HTTP servers.
2. Creating a proxy application or a gateway which allows to issue SNMP protocol requests by using HTTP.

Regarding the first solution, HTTP servers offer possibilities to extend their functionalities by building add-ons or extensions. However, it is not trivial to implement HTTP server's add-ons because it requires a deep understanding of an implementation of each HTTP server. Since an implementation of an one particular HTTP server's add-on cannot be used in another HTTP server, this approach is not flexible. Therefore, a better possibility to extend HTTP servers is to use standard interfaces like CGI (Common Gateway Interface), which allows external applications to be executed when a certain URL (Uniform Resource Locater) pointing to a retrievable object by using the HTTP protocol is requested.

The second solution requires an implementation of HTTP protocol, but offers a better performance than the first solution. In terms of the first solution, since launching external applications requires some system resources, the performance is degraded proportionally to a complexity of a CGI-based application which must be executed. Apart from offering a better performance, the second solution allows SNMP Traps to be handled without a need to rely on any other application. The reason is that the gateway can receive SNMP Traps and forward them to Web clients. In case of a CGI-based solution, an external application has to receive SNMP Traps which can then be retrieved through a CGI application. In addition, since SNMP resources such as scalar objects and columnar objects must be handled transparently, it must be able to access MIB information to determine an MIB object's type the server-side regardless of the nature of an application, when an HTTP server is extended with CGI applications or a gateway. As a CGI-based application cannot directly access MIB information, an external application is required. In other respects, a gateway can access all MIB information easily by means of its modules. According to the above-mentioned
arguments, the second solution would be considered as the best choice of an SNMP's integration into Web 2.0.

In order to integrate the SNMP infrastructure into a Web-based network management application, the gateway must provide an automatic translation between HTTP methods (e.g. Get, Post, Delete and Put) and SNMP operations (e.g. Get, Get-Next and Get-Bulk). The translation between the same primitives is not difficult because their formats are similar, while the vital problem is how to deal with the different primitives of the HTTP protocol and the SNMP protocol.

In principle, the proposed gateway should support a complete set of SNMP operations which can be either operations to retrieve data from SNMP agents or operations to modify network resources managed by SNMP-enabled devices. The gateway should also provide a possibility to receive asynchronously Trap messages from SNMP agents and then forward them to registered Web clients.

The proposed system should be very efficient and not generate huge overhead. Hence, the gateway will be able to handle many parameters with just a single request from a Web client. Consequently, network administrators can simultaneously access multiple SNMP relevant data items by using the Web-based manager application. For example, networks administrators are allowed to query a big list of SNMP managed objects with just one single request or to retrieve a large table object. In terms of network utilization, it is thus much more efficient than sending a bulk of single requests continuously to retrieve each single object of the list or the table. Traps are normally generated unexpectedly and dynamically by SNMP agents and need be propagated to registered Web clients as soon as possible. Some Web 2.0 technology as Ajax still suffers from the limitations of the Web’s request/response architecture which prevents servers from pushing dynamic Web data. The gateway should be consequently enforced to take advantages of the Long-Polling approach mentioned in section 2.4.2. It can reduce high network traffic and avoid the missing data updates of the traditional HTTP Pull approach. In terms of enhancing the the proposed system's efficiency, some heavy transport protocol such as SOAP or XML-RPC should be avoided when exchanging data between Web clients and the gateway. Some lightweight transport protocol as HTTP should be employed instead.

User interface (UI) is a key factor to an application's usability. Hence, UI of a Web-based network management application should be designed smartly and friendly to enable network administrators to monitor and configure network devices and services in an intuitive manner. Fortunately, a Web 2.0 technology as Ajax can offer a richer set of UI controls than simple HTML or PHP, so that Ajax can make a smart-looked UI of a Web client. Moreover, a Web-based network management application should be worked properly in the most well-known Web browsers like Internet Explorer, Firefox, Safari and Opera.

Security issues of the system are important that it should be considered seriously to ensure that SNMP relevant data can be secured while being transferred between Web clients and the gateway.
Other very important requirements of the proposed system should be named such as interoperability, integrability, scalability and robustness. With respect to the proposed system, interoperability refers to the capability of the system to work with most SNMP devices. Integrability refers to an ability to easily integrate the proposed gateway to many legacy applications. Robustness of the system is the quality of being able to cope well with unpredictable event like damage of SNMP devices. Scalability of the proposed system indicates its ability to handle growing amounts of Web clients or SNMP devices.

Last but not least, operating system is one of the requirements mentioned when defining system requirements. Thus, the proposed system should not be limited to particular operating systems running on particular architectures.

Finally, the following list summarizes the mentioned requirements to the proposed solution:

- The implementation of a network management application based on Web technologies
- Providing automatic translation between HTTP requests and SNMP operations via a REST-based gateway
- Supporting a complete set of SNMPv2c operations
- A smart and friendly designed UI of the Web-based management application
- Providing an universal access to SNMP resources
- Interoperability, integrability, robustness, scalability
- Efficiency
- Security
- Platform independence.

### 3.2 General Communication Model

From the view of a Web-based management application, the gateway acts as an intermediate agent. In other respects, the gateway which represents a management application sends requests to and receives responses from SNMP agents over SNMP protocol. The Web-based management application sends requests to the gateway based on the fundamental HTTP methods. The reason why HTTP protocol is selected for delivering communication data between Web clients and the gateway will be described elaborately in section 3.3.1. The gateway is responsible for translating each HTTP-based request into a corresponding SNMP operation and forwarding the commands to remote SNMP agents. Afterwards, the gateway encapsulates responses from SNMP agents into XML-based messages and sends them back to the Web client.

REST is very abstract and general. Particularly, it is not bound to the Web. However, in a scope of the Master thesis, the REST concepts are explicitly tied to the existing technologies of the Web. Therefore, from the view of the Web client, the gateway is implemented as Web services.

Because of the other REST's benefits such as an ease of use, flexibility and
simplicity, REST constraints are enforced upon a specification of the gateway. From now on, we can name the gateway as REST-based SNMP gateway. Obviously, the architecture of a proposed communication model is built on a basis of the client-server constraint of REST. Figure 3.1 shows the schematic communication model of the involved components.

![Figure 3.1: Gateway communication.](image)

The gateway connects the two different networks protocols, namely HTTP and SNMP. Because of the translation purpose, the gateway requires an Information-Base. Thus, the MIB repository stands for the one-to-one mapping between object identifiers (OIDs) in the SNMP framework with corresponding elements in a slash (“/”) format of URI.

Simple Network Management Protocol version 2 or SNMPv2c is used for communication channel between the gateway and SNMP agents because SNMPv2c revises version 1 and includes improvements in the areas of performance, security, confidentiality and manager-to-manager communications. Especially, it introduces a new PDU (Protocol Data Unit) Get-Bulk as an alternative Get-Next for retrieving large amounts of management data in a single request, that enables us to realize not only simple SNMP operation but also the SNMP complex bulk transfer. Other PDUs also included in SNMPv2c are Trap and Inform which is known as an acknowledged Trap.

### 3.3 General Approach

In this section, general approaches to the aforementioned problem will be illustrated elaborately.

#### 3.3.1 Protocol

In principle, it is possible to apply REST concepts to many network protocols. However, HTTP (Hypertext Transport Protocol) is the most REST-full protocol because it was designed to conform to REST. In order to provide a secured transmission channel between a Web client and the Gateway, the well known HTTPS protocol is used instead of a
pure HTTP to gain proper encryption and authentication of the communication channel. By the way, this functionality can be offered in the Internet in a secure style.

On the other hand, SNMPv2c predetermined in the last chapter 3.2 is used to manage SNMP agents through the REST-based gateway. The requirement interoperability with many SNMP devices leads to the selection of SNMPv2c. Similarly to HTTP, SNMPv2c operates at the application layer (layer 7) of the OSI model.

The main difference between semantic meanings of HTTP methods and SNMP operations shows that the gateway not only is responsible for simply converting PDU (Packet Data Unit) to corresponding HTTP relevant formatted message but also enables the Web-based management application to access SNMP resources in the similar manner such as other ordinary Web resources, namely static HTML Web sites, text documents or images. Therefore, the gateway should create a REST-based view of SNMP resources that provides a way to translate each SNMP operation into a corresponding HTTP method.

### 3.3.2 Resource Design

The proposed gateway considered as a Web service exposes all of its data through resources. The Web service should start with a data set or at least an idea for one. Thus, the MIB (Management Information Base) can be thought of as a data set of the gateway. The next step is to decide how to expose this data set to HTTP resources. A resource is defined as a target of a hypertext link. The MIB can be considered basically as a collection of managed objects which can be considered as logical resources. These resources can be viewed, modified and deleted.

### 3.3.3 URI Design

Since a resource and its URI ought to have an intuitive correspondence, a URI should be descriptive and should have a predictable structure. It is one important rule of a good Web design. The reason is that if a Web client knows a structure of the resource address URI, it can create its own entry points in the resource and make possible for the client to get directly to any specific resource without traversing a bunch of intermediate resources.

As it is mentioned, the MIB elaborates the structure of the management data. The MIB's hierarchy can be depicted as a tree with a nameless root, the levels of which are assigned by different organizations. It is also said that URI should be designed based on some basic rules. First, path variables can be used to separate elements of hierarchy by using a notation slash “/” and the URI looks like a hierarchical structure of a directory. There is a trivial similarity between a MIB treelike-hierarchy and a structure of URI using path variables because path variables look like we are traversing a tree. Therefore, it only needs to make a slight change from dot (.) format of OID to slash format of URI in order to make an appropriate URI addressing the “sysDescr” managed object. Example:
Besides that usage of query variables can be utilized instead of path variables if URI points to multiple resources. Example:

```
http://[Gateway-IP]/[Agent-IP]/[Community-String]/List?OID1=1.3.6.1.2.1.1.1
&OID2=1.3.6.1.2.1.1.2.
```

Query variables like we are passing arguments to an algorithm. “List” sounds like a name of an searching algorithm.

### 3.3.4 Resource Representations

The main purpose of any representation is to convey a state of a resource. In the paragraphs 3.3.1 and 3.3.2, which resources should be exposed by the gateway and what their URI will look like have been declared. Now there is a need to decide which data to send when a Web client requests a resource and what data format to use.

**Addressability**

In the end-users perspective, URI addressability is the most important thing of any Web site or application. In particular, URI addressability is sufficient for Web clients to use services provided by the gateway in simple ways. The term URI addressability gives the proposed gateway and Web clients a chance to take many benefits of REST. Thus, representations are a kind of things that should be addressable. A URI alone is expected to designate a particular representation for a resource. This seems to be a natural way on that Web applications should work. Unfortunately, Ajax-based Web applications do not work on this way. In the end-users perspective of an arbitrary Ajax-based application, there is only one URI. Whatever interaction end users carry out, whatever pieces of information they retrieve from or send to the gateway, they will not see any different URI. The reason is that an Ajax-based application is a sophisticated program which lives within a single Web page. While running the Ajax-based Web applications, the Web browser's location bar stays exactly the same when end users select different functions to change the Web application's state. Furthermore, if users click the “Back” button of a Web browser to undo the previous action, they will get surprised that the browser completely leaves the Ajax-based application's Web page. Chapter 4.2 is devoted to designate an Ajax-based Web application as a Web client of the gateway.

**Representation Formats**

REST does not specify how results of requests should be represented. However, it would be very nice to decide a format or a set of formats that can meet the goals of REST-full representation as conveying the current state of a resource and linking to possible new application and resources. Two arisen questions are: how resources should be represented and what representations format of resources the gateway should send to Web clients. The gateway should pick a representation format that says something about the
semantics of the resources, so that Web clients can process representations easily. A resource state is claimed to be just any information of underlying resources. It was already mentioned that Varbind or Variable Binding is a sequence of two specific fields. The first field is an OID which addresses a specific parameter. The second field contains a value of the specified parameter. In this case, the resource state is going to answer the question of “what a Varbind looks like in a distributed environment.”

Extensible Markup Language (XML) is a simple, very flexible text format which facilitates a sharing of structured data between machine-to-machine in a distributed system. Therefore, XML is the best choice to represent resources and MIB managed object as well as to carry current states of resources. If a Web client sends a request to query a value of a managed scalar object addressed by the OID of 1.3.6.1.2.1.1.1, the gateway encapsulates a response in the following simple XML representation.

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>XML Representation</th>
</tr>
</thead>
</table>
| Get a value of managed object addressed by OID 1.3.6.1.2.1.1.1 | `<?xml version='1.0' encoding='ISO-8859-1'?>
<Message>
  <Object>
    <OID> 1.3.6.1.2.1.1.1.0 </OID>
    <Name>sysDescr.0</Name>
    <Value>Windows PC 6.0.6000 Professional x86 Family Model</Value>
  </Object>
</Message>` |

In the SNMP world, it is easy for users to use command line tools to query a list of managed objects or a complex managed object like a table. Therefore, it is essential to discuss what the good format for a representation of a list of managed objects is or how to convey the current state of a columnar object like a table. The paragraph 3.4.1.4 is devoted to present detailed discussion on this issue.
This paragraph of the thesis is dedicated to describe the proposed network management architecture that can fulfills the thesis goal. Figure 3.2 shows in detail an extension of the gateway communication model that has already depicted in figure 3.1. General functionalities of all system components will also be described in this part.

First of all, to enable administrators to monitor and configure network devices and services in an intuitive manner, a Web-based management application should be designed and developed. Additionally, network administrators can be allowed to load both standard and vendor proprietary MIBs on the fly. The MIB OIDs grouped under specific MIB modules are browsed as a treelike view that facilitates administrators to choose OIDs of managed objects in order to make requests to the gateway.

As previously mentioned in chapter 3.2, the Web-based management application can access SNMP resources via the gateway. According to REST constraints, all resources are accessed via a generic interface. In other words, everything which is modeled in terms of resources like MIB managed objects is accessed through a URI, and only four CRUD HTTP operations of Get, Post, Put, and Delete. In end-users perspective, it is very counterintuitive to express object interfaces in terms of these four methods. The REST Uniform
Interface as the core component of the gateway is made up from several aforementioned REST interface constraints, namely identification of resources, manipulation of resources through representations and hypermedia as the engine of an application state.

The **URI Parser** component can be specified to provide many actions for the gateway, namely extracting SNMP related informations like OID of a managed object or an IP address of a SNMP agent. Besides, it identifies which corresponding SNMP operation should be executed upon receiving one specific HTTP request.

The **SMI Parser** or **SNMP MIB Parser** components can be used by the gateway to read and load SNMP MIB files as well as simple ASN.1 files. The gateway stores the data in memory to allow quick access and lookup. It is easy for the gateway to access to all the information in the MIB file, including OIDs, data types and descriptions as well as to perform some simple actions, namely translation from OID to a symbolic name and vice versa or to determine a type of managed object.

The **SNMP Adapter** component is made up by many sub components. Two of them are named **Command Generator** and **Command Responder**. In the SNMP Adapter, the task of translating an HTTP relevant information into an SNMP PDU object is delegated to its **Command Generator** subcomponent. The **Command Generator** subcomponent initiates SNMP requests, such as Get, Get-Next, Get-Bulk and Set. Conversely, the **Command Responder** subcomponent processes a response to a request which the **Command Generator** subcomponent generated. The **Command Responder** subcomponent will perform the appropriate protocol operation and coordinates with the **XML Generator** subcomponent to generate an encoded response message in XML format which is sent to the request originator.

The **HTTP Streaming** component is responsible for sending data from the gateway to registered Web clients in response to an event raised by the **Trap Receiver and Event Generator** component as soon as it receives a Trap message from SNMP agents. The **HTTP Streaming** component can be achieved by applying the Push technology or Server Push. It is said that the Push technology is contrasted with the Pull technology because its request for the transmission of information originates from the receiver, but not from clients.

### 3.4 Specification of REST Uniform Interface

All interaction between Web clients and resources is mediated through some basic HTTP methods. Any resource will expose some or all of these methods which do the same thing on every resource supporting it. The core of SNMP is a simple set of operations that gives administrators an ability to query or change state of some SNMP-enable devices. In this chapter, mappings from each SNMP operation to a corresponding regular HTTP method will be presented.

#### 3.4.1 SNMP Read Request

One category of the SNMP operations is named as retrieving data including Get,
Get-Next and Get-Bulk. These operations focus only on retrieving values of management information. On the other hand, the HTTP Get method fetches a representation of the target resource. For example, a Web client manipulates a resource by connecting with the hosting gateway that hosts it and sending the gateway a method of Get and path to the resource. An important rule of thumb is that a Get operation is safe. It can be done repeatedly without changing visibly a state of a resource. This property is very important for various reasons. It can be argued that there is similarity between SNMP Get, Get-Next and Get-Bulk operation with HTTP Get method regarding their operational functionalities. To be part of a uniform interface, these operations must not be specific to a particular type of resource. Being resource-independent requires that these SNMP operations should be defined in terms of Web resources and in this specific case as the verb Get. Therefore, to retrieve data via the HTTP Get method, translations from SNMP operations will be demonstrated in detail in this section of the thesis.

**SNMP Get**

To fetch a resource, a Web client sends an HTTP Get request to its URI which is constructed based on the format mentioned in section 3.3.3. In principle, the SNMP Get operation allows the gateway acting as a Network Management Station (NMS) to retrieve object instances from agents. Consequently, the gateway sends back a representation of the resource which is encoded in a response's entity-body to Web clients.

Again, thinking of the gateway as a Web service, a Web client can get three kind of resources including a scalar managed object, a tabular managed object and a list of scalar managed objects. If a Web client wants to receive a value of a single scalar MIB object from an SNMP agent, it must send an HTTP Get request to the following URI.

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the value of single MIB variable</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/[OID].</td>
</tr>
</tbody>
</table>

In other cases, if a Web client wants to obtain not only a single value but also the values of a list of MIB managed objects addressed by a range of OIDs from [v_oid1] to [v_oidn], the URI will be constructed as following:

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get values of multiple MIB objects</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/List?OID1=[v_oid1] &amp; OID2=[v_oid2] &amp; ... &amp;OIDn=[v_oidn].</td>
</tr>
</tbody>
</table>

In principle, a URI can be truncated if its length is longer than 255 characters if a Web client accesses resources via an HTTP Get method. Therefore, it must detect a length of the URI. If the URI has more than 255 characters, a Web client automatically divides the original list of OIDs into a number of sublists. Afterward, it sends multiple HTTP Get requests to the gateway while passing elements of these sublists like URI-encoded parameters. The truncation operation of one specific URI must be completely transparent.
to end users.

**SNMP Get-Next**

The Get-Next operation allows the gateway to retrieve the next object instance from a table or a list within an agent. For example, when the gateway wants to retrieve all elements of a table from an agent, it initiates a Get operation which is followed by a series of Get-Next operations.

If a Web client wants to get the value of the next scalar object in the MIB tree, the mapping of the SNMP Get-Next operation to a corresponding HTTP method can also be done similarly with the SNMP Get one. Therefore, a URI can be modified a bit in an under manner:

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the value of the next scalar object in the MIB tree</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/[OID]/Next.</td>
</tr>
</tbody>
</table>

Similarly, a request to fetch a content of a particular columnar MIB object as a table is produced by passing an OID of the table object together with a range of columns indexes as URI-encoded parameters. An example of how to retrieve a partial content of the table object from column $m$ to column $n$ is illustrated below. The URI is given in the same form above to get the value of a single scalar MIB object.

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a table object</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/[OID]?min_column=m&amp;max_column=n.</td>
</tr>
</tbody>
</table>

Without passing input parameters, $min\_column$ and $max\_column$, the content of the whole table will be returned as a result. Obviously, the types of managed objects, scalar or tabular, are completely transparent in a view of end users. The gateway is responsible for determining the type of a specific MIB object based on its OID.

In case a Web client uses SNMP Get-Next requests to query a network entity for a subtree under one specific OID, it sends a request to the following URI:

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNMP-Walk</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/[OID]/SubTree</td>
</tr>
</tbody>
</table>

This OID specifies which portion of the object identifier space will be searched using Get-Next requests. All variables in the subtree under the given OID are queried and their values presented in a XML format are sent to the Web client.

**SNMP Get-Bulk**

The Get-Bulk operation was added to make it easier for acquiring a large amount of related information without initiating repeated Get-Next operations.

Complex SNMP operations, namely Get-Bulk, can be also realized in the same
URI-based manner as simple ones like Get and Get-Next. For instance, when a Web-based network management application desires to get a large amount of data, its request is drawn by giving a list of OIDs ranging from \([v\_oid1]\) to \([v\_oidn]\) along with the \textit{Max\_Repetitions} and \textit{Non\_Repeaters} values. The URI standing for the matched resources can be formulated in the next favorable fashion:

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get-Bulk values of a list of OIDs</td>
<td>\texttt{http://[gateway_address]/[agent_address]/[community_string]/Bulk?OID1=[v_oid1] &amp;...&amp;OIDn=[v_oidn] &amp;max_repetitions=[max_repetitions] &amp; non_repeaters=[non_repeaters].}</td>
</tr>
</tbody>
</table>

\textbf{Handling of HTTP Get in The Gateway}

Upon receiving an HTTP Get request through the \textit{REST Uniform Interface} component, a URI is parsed by invoking some appropriate methods of the \textit{URI Parser} component. Consequently, the gateway obtains all SNMP essential information such OIDs of managed objects, an IP address of an SNMP agent, a name of the SNMP operation which should be performed. Since a mechanism to make a request to retrieve a table object completely differs from the one to fetch a scalar object, the type of a queried MIB managed object, such as scalar or tabular, must be distinguished in advance by the gateway thanks to functionalities of the \textit{SMI Parser} component. The identified SNMP operation's input parameters can also be figured out in case the operation is Get-Bulk or in case a table object must be retrieved. In the succeeding step, the \textit{SNMP Adapter} component generates and sends the proper SNMP request, namely Get, Get-Next, Get-Bulk or Tree-Walk respectively, to an SNMP agent as well as processes a response to its request. Finally, the \textit{SNMP Adapter} component encapsulates the response message into a simple XML representation which will be sent to a Web client later. In case a Web client requests to fetch a list of managed objects, it is possible to extend the XML representation of a single scalar MIB object that has been depicted out in the section 3.3.4.2 to present an XML representation of a list of managed MIB objects.
XML Representation of a list of scalar MIB objects

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<Message>
  <Object>
    <OID> [OID_1] </OID>
    <Name> [Name_1] </Name>
    <Value> [Value_1] </Value>
  </Object>
  <Object>
    <OID> [OID_2] </OID>
    <Name> [Name_2] </Name>
    <Value> [Value_2] </Value>
  </Object>
  ..........................................................
  ..........................................................
  <Object>
    <OID> [OID_n] </OID>
    <Name> [Name_n] </Name>
    <Value> [Value_n] </Value>
  </Object>
</Message>
```

If a Web client sends an HTTP GET request to fetch a table object with \( n \) rows and \( m \) columns, its XML representation can be described as follows:

XML Representation of a table object

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<Message>
  <Table>
    <Row>
      <Column> [Value_11] </Column>
      <Column> [Value_12] </Column>
      <Column> [Value_13] </Column>
      ......................................................
      <Column> [Value_1m] </Column>
    </Row>
    ......................................................
    ......................................................
    ......................................................
    <Row>
      <Column> [Value_n1] </Column>
      <Column> [Value_n2] </Column>
      <Column> [Value_n3] </Column>
      ......................................................
      <Column> [Value_nm] </Column>
    </Row>
  </Table>
</Message>
```

### 3.4.2 SNMP Modify Request

The HTTP PUT method replaces the target resource with another. The client usually sends a representation along with a PUT request, and the server tries to create or change the
resource so that its state matches what the representation says. The SNMP Set operation is used to change value of a managed object or to create a new row in a table. Due to a matched semantic of the HTTP Put method, it is mapped to the SNMP Set operation to enable a Web client to alter managed objects defined with an access privilege such as read-write or write-only in the MIB.

**Syntactical and Semantical Specification**

To modify a value of a resource like a MIB scalar object, a Web client sends an HTTP Put request to the gateway that usually includes an entity-body containing its new value. The URI can be formulated as following:

<table>
<thead>
<tr>
<th>SNMP Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set a new value for the MIB variable</td>
<td>http://[gateway_address]/[agent_address]/[community_string]/[OID]</td>
</tr>
</tbody>
</table>

**Handling of HTTP Put on Gateway**

The HTTP Put request can be treated in a similar manner with the HTTP Get request. Firstly, the gateway attempts to pull out an OID of a managed object and an IP address of an SNMP agent directly from a URI. A new value of the managed object can be drawn out from a body content of the HTTP Put request. In the consecutive step, an appropriate method of the Command Generator component is invoked to deliver an SNMP Set operation to the SNMP agent. Finally, the gateway sends an XML message back to the Web client to notify whether the requested operation has been successfully executed or not.

**3.4.3 Trap Asynchronous Operation**

This chapter discusses how the gateway can receive Trap messages from SNMP agents and forward Trap messages encoded in a predetermined format to registered Web clients.

**Syntactical and Semantical Specification**

From the view of SNMP agents, the gateway in general acts as a Network Management Station (NMS). Therefore, Trap messages are delivered directly to the gateway from SNMP agents once they occur. However, from the view of its Web clients, the gateway plays a role as a Trap generator because Traps from SNMP agents must be finally delivered to all Web clients which express their interests in the Traps. Web clients should have a possibility to register its interest in a particular Trap bound up to an OID with the gateway. The gateway upon receiving and processing any Traps will hand them off as soon as possible to a few number of Web clients. It is possible to see that the gateway's Trap notification feature can be implemented by using the Long Polling model mentioned in section 2.4 because this approach offers all functions which are needed. Under the view of Pub-Sub model, Web clients can be considered as subscribers. For example, if a Web client
wants to receive a Trap message called *Authentication Failure* generated when there is a request coming from an unauthorized manager, it sends an HTTP Get request to the gateway with the following URI. In this case, the OID value of 1.3.6.1.6.3.1.5.5 is bound to the above-mentioned Trap.

<table>
<thead>
<tr>
<th>Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Web client register its interestingness with the gateway to receive Trap that are bound to event</td>
<td>http://[gateway_address]/Subscribers?channel=[OID]</td>
</tr>
</tbody>
</table>

**Handling of Trap Operation on Gateway**

At present, an interesting idea is raised up: “How are Trap messages transmitted to Web clients?”. Let's start by discussing how to deal with incoming traps from SNMP agents. Handling incoming Traps is the responsibility of the REST-based SNMP gateway. To realize the idea, subcomponents of the gateway, *Trap Receiver and Event Generator*, are specified.

The *Trap Receiver* component listens for Traps generated by devices in the network and hands them off to the *Event Generator* component. In turns, the *Event Generator* component triggers what the gateway calls as an event. The outputted event will later be detected and captured by another subcomponent of the gateway, *HTTP Streaming*. The *HTTP Streaming* subcomponent, in turn, converts the original Trap to a corresponding XML formatted message and sends it to all registered Web clients by publishing the Trap message to one named logical channel or one specific topic. Consequently, all Web clients or subscribers interested in the Trap will receive messages published to the topic. For example, if an SNMP agent sends a Trap corresponding to system reboots (*coldStart*) to the gateway, a message in an XML format conveying to Web clients that are interested in the Trap can be described as follows.

<table>
<thead>
<tr>
<th>Event</th>
<th>XML representation of trap message</th>
</tr>
</thead>
</table>
| The Trap message received when a system is rebooted. | `<xml version='1.0' encoding='ISO-8859-1'?>
<Message>
  <Source>192.168.0.4</Source>
  <TrapOID>coldStart</TrapOID>
  <Object>
    <OID>sysUpTime</OID>
    <Value>0</Value>
  </Object>
</Message>` |

**3.4.4 Get List of MIB Nodes**

As already mentioned in section 3.3.5, the Web-based clients allows network administrators to view the hierarchy of SNMP MIB variables in the form of a tree. In order to enable the content of a tree view browsing MIB OIDs to be generated dynamically, the
gateway should provide Web clients a way to fetch a list of child nodes of one specific node in a tree. When a Web client requires a list of child nodes of the MIB managed object in a tree, it sends a Get request to the undermentioned URI:

<table>
<thead>
<tr>
<th>Command</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a list of child nodes</td>
<td>http://[gateway_address]/[OID]/Children</td>
</tr>
</tbody>
</table>

In turns, the task of looking up the in-memory MIB repository to find a list of child nodes of the parent one is delegated to the SMI Parser component. Representation of one single child node should contain at least its symbolic name, its OID as well as the type of the node, inner node or a leaf node. An inner node has at least one other child node. Conversely, a leaf node has no child node. A simple XML representation of a list of child nodes with \( n \) elements of one specific parent node can be drawn out below.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<Message>
  <Node>
    <Type>[Inner]or [Leaf] </Type>
    <Name> [Symbolic name of MIB node 1-st] </Name>
    <OID>[OID of a node 1-st]</OID>
  </Node>
  .......................................................................................
  <Node>
    <Type>[Inner]or [Leaf] </Type>
    <Name> [Symbolic name of MIB node n-th] </Name>
    <OID>[OID of a node n-th]</OID>
  </Node>
  </Message>
```
Chapter 4 Design and Implementation

In this chapter, to get a closer look at what the proposed system’s functions can offer network administrators prospects of monitoring and configuring SNMP-enabled network devices, all use cases under a view of a network administrator will be generally elaborated in section 4.1. In the next section 4.2, the reasons of selecting some specific frameworks, libraries to implement the REST-based SNMP gateway will be clarified. Afterwards, the implementation of the gateway as well as the implementation of the Ajax-based network management application will be detailed in section 4.3 and 4.4.

4.1 Used Technologies and Libraries

The implementation of the gateway is followed in Java. The simple reason is that there are many open-source libraries and frameworks written in Java that can alleviate the implementation of core components of the REST-based SNMP gateway, namely REST Uniform Interface, SMI Parser, SNMP Adapter and HTTP Streaming. Additionally, Java is claimed to be an OS (Operating System) independent programming language, thence the gateway can be deployed on arbitrary platforms. Although the Restlet framework coined by Jérôme Louvel and Thierry Boileau [14] is claimed as a lightweight REST framework for Java to embrace the architecture of the Web and get benefits from its simplicity and scalability. Restlet started as an API built on top of the Servlet API. Unsurprisingly, it turns out the Restlet API which is quite similar to Java Servlet API but more complicated. Restlet would be the first choice if a server-side application has to handle complicated a content negotiation between multiple representations of a resource that would not be touched in the scope of this Master thesis. It is especially beneficial to use Restlet to develop and deploy standard-alone Restlet-based applications. However, it is not trivial to deploy a Restlet-based application inside a standard Servlet container using an adapter provided by the Restlet framework. In other respects, the Java Servlet API is perfectly capable of creating a REST-like Web service and easy to use. By definition, REST implementations are constrained to very few verbs such as Get, Post, Put, Delete. In fact, the Java Servlet API already has corresponding methods defined in the HttpServlet class, namely doGet, doPost, doGet, doPostDelete. Based on these arguments, Java Servlet API is used to implement the core component like REST Uniform Interface.

The SNMP Adapter component is realized on a basis of the library SNMP4J [3]. SNMP4J is the free open-source and state-of-the-art SNMP implementation for Java. SNMP4J is the alternative written in Java of SNMP++, which is a very well-known SNMP API for C++ in the industry for many years. Thus, SNMP4J provides a rich set of features expected for the implementation of the SNMP Adapter component. It, for example, supports all PDU types of SNMPv1, SNMPv2c and SNMPv3 as well as provides pluggable transport mappings, so that UDP and TCP are both supported. Additionally, SNMP4J API contains supports for a multi-threading model, command generator as well as responder. It also enables us to send synchronous and asynchronous requests as well as to retrieve a table
object very efficiently with Get-Bulk.

The *SMI Parser* component is realized by using the open-source SNMP MIB parser written in Java called Mibble [4]. Mibble is the good choice because it is the open-source and stable product. It is very easy to integrate Mibble to SNMP4J. Moreover, Mibble facilitates quick accessibility to all information in both SNMP MIB files and simple ASN.1 files. Mibble is well-documented and distributed as a Java library, so that it is easy for developers to understand Mibble API as well as to include it in any Java-based application.

It is claimed in section 3.4.3, the gateway's feature of delivering Trap messages to registered Web clients can be implemented by using an approach on a basis of the Bayeux protocol. Cometd is known as a Dojo Foundation project to provide implementations of the Bayeux protocol in Javascript for client side, in Java for server side, and in some other languages such as Perl, Python. Some other organizations (e.g. Sun, IBM and BEA) also have their commercial implementations of the Bayeux protocol. However, Cometd is the best choice because it is an open-source product that is originally developed by the same working team who coined the Comet neologism. It is absolutely unsurprised, that Bayeux and Cometd are two complimentary parts of a plan to tackle the complexity around building and deploying Comet applications that becomes easier in the Bayeux-compliant servers.

Bayeux protocol has recently been implemented and included in a number of Web servers including Jetty [21] and IBM Websphere [22]. Jetty is selected as a server to deploy the gateway because it can act as a Servlet container and offer a free license. In addition, the Bayeux protocol currently provides a connection type called Long Polling for HTTP Push, which is implemented in the Jetty’s Cometd library.

In chapter two, it is explained that the Ajax/REST architectural style is superior to the traditional client/server Web architecture in terms of its run-time characteristics, its simplicity to personalize user interfaces and its scalability. These are main reasons why Ajax is selected to develop the Web-based network management application.

The Dojo toolkit is simply chosen among a lot of commercial and free license frameworks in the market, such as Prototype [26], jQuery [27], Ext [28], MochiKit [29], GWT [30] and YUI [31], even though choosing an appropriate toolkit to develop Ajax-based Web applications is not a trivial task. The primary reason is that the Dojo toolkit is a Dojo Foundation open source project like Cometd, therefore there is no doubt of integrating the Jetty's Cometd library to an Ajax-based Web application using the Dojo toolkit. The Dojo toolkit is backed by an open community containing many individuals and companies who commit to continuously develop and build new tools that benefit everyone. Thus, the Dojo toolkit can offer many fascinating features that can be named as follows. Firstly, Dojo includes high-performance implementations of common utilities into its core. Particularly, the rebuild of Dojo for version 0.9 takes a focus especially on its performance and its reduced code footprint. Thus, the Dojo toolkit offer an ability to implement Ajax-based Web sites with a great performance. Consequently, Dojo is used to develop high-
profile, high-traffic sites of well-known organizations and companies, such as AOL [22] and Nexaweb [23]. Secondly, it is easy to manage large-scale UI development projects thanks to the small footprint of the Dojo’s package system. Dojo is a solid infrastructure for delivering great experiences because it allows each component of a Web application to be built on a trusted set of high-quality building blocks by providing an integrated infrastructure and a variety of optional modules. Finally, it is very beneficial to use Dojo as it provides a uniform API that not only hide the differences between browsers when it comes to XMLHttpRequest, but also hide the difference between XMLHttpRequest and other ways of getting the browser to send an HTTP request. Hence, it enables a simple cross-browser solution for the Web.

4.2 Use Case

It is said that the use case technique is used in software and system engineering to capture concretely the functional requirements of a system. Use cases help readers can get an better understanding of how a network administrator can interact with the proposed system to achieve his goals. The following graphic 4.1 shows all main use cases under a view of a network operator which demonstrate how interactions take place.

![Use Cases of Network Administrator](image)

Figure 4.1: Use cases of network administrator

Many coders will begin the design and implementation phase by thinking about the functionality of the Login use case: “We need a Login screen since we will authenticate our users' credentials”. Although the concept of Login is not originally built into the Web, HTTP is a stateless protocol and Login which implies maintenance of state information must be added on the top of it. In other respects, it is argued that without the Login page, all Web browsers can request any sensitive and important SNMP data from the
gateway. This fact is obviously very harmful. In respect of protecting HTTP traffic between the gateway and Web clients from adversary, using of HTTPS is more preferable than HTTP. Hence, in case of the Web-based network management application, the Login is the first use case that should be considered seriously.

The use case named Configure Input Parameters is used to depict an user's action to enter into the specific gateway attributes such as the IP of the gateway, values of read/write community strings and the value of the HTTP connection timeout.

A set of analogous use cases such as Execute SNMP-Get, Execute SNMP-GetNext, Execute SNMP-GetBulk, Execute SNMP-TreeWalk, Execute SNMP-Set and Execute SNMP-TableView illustrate abilities of network operators to execute not only SNMP basic operations such as SNMP Get, Set, Get-Next and Get-Bulk but also complex ones to retrieve a table object or to walk through a subtree.

The use case called View Traps describes a circumstance when any Trap message generated at the SNMP agents' side is passed through the gateway to registered Web clients. In the next step, detailed information of the Trap message is displayed in the Web browser.

4.3 Design and Implementation of Gateway

4.3.1 Classes Design

The following UML class diagram gives an overview of the most essential classes which constitute the implementation of the gateway.

The SNMP Adapter component described in figure 3.2 can be implemented as a class with the analogous named SNMPAdapter. According to principles of OO (Object-
oriented) design, association relations between the SNMP Adapter component and its subcomponents such as Command Generator, Command Responder, XML Generator and Trap Receiver&Event Generator can be realized by using either the design concept of Interface or Composition. However, the utilization of Interface is more preferable as it leads to dynamic binding and polymorphism which play very important roles to the OO design and programming. Hence, it enables us to add or remove easily and flexibly new functions and behaviors to or from the SNMPAdapter class in case this class is extended because of new requirements.

The LoginServlet and the AuthenticationServlet classes work together in order to realize the use case Login mentioned in section 4.1.

The SMI Parser and URI Parser components are realized respectively by the SMIParser and URIParser classes which are shown in figure 4.2.

The RestServlet class as the Java specific implementation of the Rest Uniform Interface component is also depicted in figure 4.2.

The PubSubServlet class that is created to operate as a component of receiving subscriptions from different Web clients. The Web client can express their interests only in some particular Trap messages. The HTTPStreaming class is the Comet-based implementation of the abstract HTTP Streaming component which has been depicted in figure 3.2. As the HTTPStreaming class associates with the SNMPBayeuxService class through the Composition association relation, one instance of the HTTPStreaming class might contain multiple instances of the SNMPBayeuxService class which is responsible for delivering Traps to registered Web clients as soon as they occurs.

The MainClass class is dedicated for setting up the initial configuration parameters of Jetty-embedded server as well as for initializing Servlet holders of aforementioned important Java Servlet-based classes such as LoginServlet, AuthenticationServlet, RestServlet and PubSubServlet.

4.3.2 SNMP Adapter Class

In the sub section 4.3.1, it is mentioned that the SNMPAdapter class implements a variety of different interfaces. The implementation of this class is accomplished on a basis of a utilization of the SNMP4J library. It is said that an interface practically represents a contract in which a class implementing an interface must implement every method of it. Therefore, the SNMPAdapter class must provide details of implementations of all methods which are declared in the bodies of these interfaces. According to the general class diagram showed in the figure 4.2, the SNMPAdapter class implements the ICommandGenerator interface. Thus, the methods with the following signatures must be implemented.
The method named `createSnmpTarget` is responsible for setting up initial input parameters such as a version of SNMP protocol, an IP Address of an SNMP agent and a security relevant input parameters such as read/write community strings. The SNMP4J library is supposed to provide a pluggable transport mappings. This means that both UDP and TCP are supported out of the box. Hence, the `createSnmpSession` method is given to work on input parameters which determine a type of a used transport protocol. This method is also accountable for initializing an SNMP session. The `sendPDURequest` method is responsible for sending different simple PDU types such as Get, Get-Next and Set to SNMP agents. It is acknowledged that it is complicated to issue and perform repeated SNMP Get-Next and SNMP Get-Bulk operations to retrieve a table object or to get a content of a subtree in the SMI Object Tree. Therefore, the two methods of `getTable` and `doTreeWalk` are designed to alleviate these complex issues respectively.

The `PDUFactory` interface that is provided as a part of the SNMP4J library is also implemented by the `SNMPAdapter` class. The `PDUFactory` interface designates a method called `createPDU` with the purport to create the proper PDU. The type of PDU can be set by one of the following values: Get, GetNext, Set and GetBulk. The `SNMPAdapter` class must implement this method. The signature of the aforementioned `createPDU` method looks like:

```java
PDU createPDU(int pduType);
```

Since, the `SNMPAdapter` class also implements the `ICommandResponder` interface, the most important methods with the following signatures must be implemented.

```java
void processPDU(CommandResponderEvent e);
StringBuilder sendProcessSNMPResponse();
```

These methods called `processPDU` and `sendProcessSnmpReponse` collaborate with each other to handle result messages of incoming requests to SNMP agents.

As the two methods named `getPDUResponse` and `getPostPDUReport` are declared in a scope of the `IXmlGenerator` interface, their implementations must be made by the `SNMPAdapter` class due to a contract of the Interface association relation. Their signatures can be depicted as following.

```java
StringBuilder getPDUResponse(PDU response);
StringBuilder getPDUReport(PDU report);
```

The purpose of these two methods are to generate messages in XML format which are later returned to Web clients. The `getPDUResponse` method is used particularly in case a command request is executed successfully without any error. Conversely, the `getPDUReport` method is answerable for producing XML-based error messages in case some errors occur while performing SNMP requests.

SNMP agents are capable of sending either SNMP Trap or Inform messages that
depends on the PDU type which is set to `PDU.TRAP` or `PDU.INFORM` severally. It is pointed out that Trap and Inform messages are mainly similar to each other. Therefore, for a sake of simplicity, the `SNMPAdapter` class that implements the `ITrapGenerator` interface only provide methods for handling with SNMP Traps. The two methods with signatures listed below enable us to initialize input parameters and start up an instance object of the `Trap Receiver and Event Generator` class.

```java
void initializeTrapHandler(String host);
void startTrapHandler();
```

### 4.3.4 SMI Parser Class

The helper `SMIParser` class provides facilities to read SNMP MIB files as well as ASN.1 files that are located at the gateway side. MIB files can loaded by an instance of the `MIBLoader` class that is contained inside the Mibble library. The method named `loadMIBToMemory` is used to load MIB files into memory to allow a quick access and lookup. The signature of the method can be shortly described in the following manner.

```java
void loadMIBToMemory(ArrayList<String> listOfMIB);`

The method searches a path for locating MIB files. The search path directories can be either normal system files or directories.

```java
Iterator<String> iter = listOfMIB.iterator();
while(iter.hasNext())
{
    File file = new File(iter.next());
    _mibloader.addDir(file.getParentFile());
    _mibloader.load(file);
}
```

Eventually, it stores the data of MIB files in the memory through the global variable `_mibs` for the later accessing purpose.

```java
_mibs = _mibloader.getAllMibs();
```

The `SMIParser` class contains methods for an easy access to all the information such as OIDs, type data and description in the loaded MIB files. For example, the `getMIBType` method returns a MIB type such as Object Identifier, String, Integer and Boolean of an arbitrary MIB object if its OID is passed as an input parameter.

```java
String getMIBType(String oid);
```

In the scope of the `getMIBType` method, an instance of the `MibValueSymbol` class that holds information relevant to a MIB value assignment, such as a type and a value, is instantiated. Normally the value is an Object Identifier.

```java
MibValueSymbol valueSymbol = this.getValueSymbol(oid);
```

Afterward, the method must checks whether the type is an instance of the `SnmpObjectType` class by using the `instanceof` Java operator.

```java
If (valueSymbol.getType() instanceof SnmpObjectType)
```

If the `if-then` statement is true, the type is extracted from an instance object of the `MibValueSymbol` class through the `getType` method.
MibName mibName = (SnmpObjectType)valueSymbol.getType();

By a mean of the instanceof operator, the type object can be finally determined.

if (mibName instanceof IntegerType)
{
    //Do something ..........
}
else if (mibName instanceof BooleanType)
{
    //Do something ..........
}
else if (mibName instanceof ObjectIdentifierType)
{
    //Do something ..........
}

Within the scope of this Master thesis, other remaining methods of the SMIParser class will not be described in detail. However, it is possible to summarize signatures as well as general meanings of the most important ones in the following table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MibNameValueSymbol[] GetChildrens(String oid)</td>
<td>Returning a set of child nodes of one particular node in the SMI Object Tree.</td>
</tr>
<tr>
<td>Boolean isScalar(String oid)</td>
<td>Determining whether a type of an MIB object is scalar or not.</td>
</tr>
<tr>
<td>MibNameValueSymbol getValueSymbol(String oid)</td>
<td>Returning an object that holds information relevant to an MIB value assignment such as a type and a value,</td>
</tr>
<tr>
<td>Boolean isTable(String oid)</td>
<td>Determining whether a type of an MIB object is tabular.</td>
</tr>
<tr>
<td>String getFullSymbolicName(String oid)</td>
<td>Returning a full symbolic name of an MIB object identified by a corresponding OID.</td>
</tr>
</tbody>
</table>

### 4.3.5 URI Parser Class

The main purpose of creating the helper URIParser class is to parse URI schemes which are passed as a part of HTTP request's body when Web clients desire to retrieve or modify SNMP resources standing behind the gateway. Therefore, two methods of the URIParser class called extractSnmpParameters and extractOID provide possibilities to extract all SNMP specific parameters from the provided URI schemes. These methods signatures can be described as follows:

```java
boolean extractSnmpParameters(HashMap<String, String> result,
                            String[] tokens, int startPos);
StringBuffer extractOID(String[] tokens, int startPos);
```

The other two methods named parseURL and parsePath with the following signatures provide us facilities to collect all URI specific elements that are separated by the slash delimiter “/” from the URI or its relative path. The parseQuery method is used to return a list of query variables from the URI's query string.
4.3.6 RestServlet Class

As it is already mentioned, Java Servlet API is a perfect selection to implement REST-like Web service. One might see an analogy in terms of semantics between four basic methods of HttpServlet such as doGet, doPost, doPut and doDelete to four universal verbs Get, Post, Put, Delete that are widely used for developing REST-like Web service. However, as mentioned in chapter 3, only two verbs such as Get and Post would be used to identify which SNMP operation is performed. Thus, the RestServlet class, the implementation of the abstract component named REST Uniform Interface, should extend the HttpServlet base class in order to provide elaborated implementations of the two methods doGet and doPost. These two methods with their analogous signatures can be described in the following manner.

```java
void doGet(HttpServletRequest req, HttpServletResponse resp);
void doPost(HttpServletRequest req, HttpServletResponse resp);
```

The RestServlet class also provides some helper methods such as doSnmpGet, doSnmpGetNext, doSnmpGetBulk, doSnmpSet, doSnmpWalk and doSnmpGetTable which are accountable for invoking its corresponding method of the SNMPAdapter class to retrieve or alter values of SNMP objects. The doSnmpSet method is only invoked in the scope of doPost method in case Web clients send requests to modify MIB scalar objects. In the other cases, when Web clients send HTTP requests to the gateway in order to retrieve SNMP objects, which method should be invoked in the scope of the doGet method depending on one specific element extracted from the passed URI scheme which identifies the name of the corresponding SNMP operation to be performed. For the sake of simplicity, only the implementation of the doSnmpGet method will be exemplified to detail how the Rest Uniform Interface component and the SNMPAdapter component collaborate to perform an SNMP Get operation. An insight look at the implementation of other helper methods of the SNMPAdapter class are not carried out within the scope of this Master thesis due to their similarities to the implementation of the doSnmpGet method.

```java
void doSnmpGet(String agent, String communityString, String oid, HttpServletResponse resp) {
    SnmpAdapter snmpAdapter = SnmpAdapter.createSnmpAdapter(agent,
            new String[] {oid + ".0"}, false,0);
    snmpAdapter.setOperation(PDU.GET);
    snmpAdapter.setCommunityString(communityString);
    StringBuilder resultString =
            snmpAdapter.sendProcessSnmpResponse();
    returnHttpResponse(resp, resultString);
}
```

First of all, an object instance of the SNMPAdapter class is instantiated. Values of
input parameters which are SNMP specific such as an IP address of an SNMP agent, a name of SNMP operation, a community string must be passed either to the SNMPAdapter class constructor or its Set helper methods. Afterwards, the sendProcessSnmpResponse method of the SNMPAdapter class is invoked to deliver a request with its parameters to an SNMP agent which is already identified by its IP address. Finally, XML-based returned messages will be flushed to the output stream of the gateway through an invocation of the helper returnHttpResponse method of the RestServlet class. The body of this helper method can be listed out as follows.

```java
resp.setContentType("text/xml");
ServletOutputStream out = resp.getOutputStream();
BufferedReader buf = new BufferedReader(
    new StringReader(stringResult.toString())
);
String line = buf.readLine();
while (line != null) {
    out.println(line);
    line = buf.readLine();
    out.flush();
}
```

### 4.3.7 Login Servlet and Authentication Servlet Classes

The LoginServlet and AuthenticationServlet classes are the important realizations of the mechanism HTTP Authentication which is built into the gateway to prevent unauthorized users from accessing SNMP resources. Every user of the system is required to enter his credential such as user name and password in the Login page which is generated by the LoginServlet object. In the next step, the browser sends the encrypted credential to the gateway by issuing an HTTP Post request. The AuthenticationServlet object validates the inputed credential by calling the checkAccessRight method. In turn, this method matches the supplied credential with a collection of pairs <user name, password> stored in a simple database which can be accessed by the gateway. In a respect of implementing the gateway prototype, a very simple text file is used as a storage of credentials of authorized users.

```java
boolean checkAccessRight(String username, String password) {
    //validates user name and pass word
}
```

If the credential is acceptable, the AuthenticationServlet object returns a main HTML page of the Web-based network management application to the Web client. The main page provides authorized users facilities to carry out actions to fetch or modify SNMP resources.

### 4.3.8 HTTP Streaming Class

It is mentioned that the Trap Receiver&Event Generator component generates an external event to notify the HTTP Streaming component if Trap occurs. Thus, one might see that an interaction between these components can be realized by using the Observer
pattern. This pattern is used to notify and update observers automatically when an observable object changes its state. According to this pattern, since the `HTTPStreaming` class playing a role as an observer implements `ITrapObserver` interface, its `updateState` method with the following signature must be compulsorily implemented.

```java
void updateState(HashMap<String, String> trapMessage);
```

After processing an event raised by the `Trap Receiver & Event Generator` component notifying about an occurrence of a Trap message, the `sendTrapToCometdClients` method of the `HTTPStreaming` class is called to deliver the Trap message encoded in XML to all registered Web clients. In the scope of this method, all instance objects of the server-side class extending the base one, `ClientImpl`, are collected into an array. Note that the `ClientImpl` class provides services to remote objects located at the client side.

```java
Set<String> ids = _bayeux.getClientIDs();
for(String id: ids)
{
   clients[i] = (ClientImpl)_bayeux.getClient(id);
}
```

Afterward, each of these instance objects will be checked if it is an instance object of the `ContinuationClient` class by using the `instanceof` operator. If the operator returns a true value, the object has a reference to one remote object at the client side. Consequently, the Trap message will then published to a predefined channel identified by a specific name.

```java
if(clients[i] instanceof ContinuationClient)
{
   _bayeux.getChannel(channelName).publish(clients[i],
       trapMessage, null);
}
```

Typically, a special server-side service called `SnmpBayeuxService` has subscribed itself to the channel in advance. Messages published to the predefined channel will be delivered to types of objects like this service. This service in turn will delivers the Trap message to every registered Bayeux object located at the client side by invoking the `send` method with the following signature.

```java
send(Client client, String channel, Message object, String id);
```

### 4.4 Ajax as REST Client

The focus of this section is to explain how to create the Web-based network management application based on an Ajax framework.

#### 4.4.1 Structure of Web-based Network Management Application

The `Login` page is constructed as a simple HTML Web page. The preliminary purpose of creating the `Login` page is to prevent unauthorized users from retrieving or altering critical SNMP resources. First, an arbitrary user is asked for entering his user name and password. If the user is authorized, the `AuthenticationServlet` object makes the resource available by redirecting the user to the main page.
After the user's credential is verified, the main page of the Web-based network management application is displayed. The main page is organized with different panels. The primary panel is used to display results returned from the gateway when users send HTTP requests to retrieve SNMP resources. A tree view browsing the content of the SMI Object Tree is located in the second panel. The content of this tree view can be generated on the fly. When an user clicks on one particular node, a set of its child nodes will be returned to the Web browser and they will be added dynamically to the tree view. It is worth to acknowledge that, in order to reduce communication traffic between the gateway and Web clients, a list of child nodes of one specific node are only returned one time during a whole working session. Additionally, the main page provides a menu that enables users to execute different SNMP operations. In terms of executing a complicated SNMP operation like Get-Bulk and Table-View, a different panel is provided for entering the operation's parameters, such as `max-repetitions`, `non-repeaters`. The last panel is dedicated for integrating a Trap receiver application to the main page. A detailed information of a Trap message will be displayed in this panel in case the Trap occurs.

### 4.4.2 Implementation of Dojo-based Web Client

This subsection is devoted to provide a look at technical details of the implementation of the Web client UI by utilizing the Dojo toolkit. One would begin the implementation by including references to the Dojo library and the required name spaces:

```javascript
<script type="text/javascript" src="dojo/dojo.js"></script>
<script type="text/javascript" src="dojox/cometd.js"></script>
dojo.require("dojo.parser");
dojo.require("dijit.Tree");
dojo.require("dijit.Menu");
dojo.require("dojox.cometd");
//more including statements...
```

The core subroutines of the Dojo toolkit allow Web clients to send HTTP Get requests asynchronously to the gateway to grab SNMP relevant information. In order to illustrate out how a Web client sends an HTTP Get request to retrieve a value of a scalar object, the implementation of the `SnmpGetSingleOID` method will be exemplified in the following part. Basically, an OID of the scalar object must be passed as a the method's input parameter. The implementation of this method is self-explanatory for the other analogous methods which can be called to retrieve different kinds of SNMP resources, for example, a set of child nodes of one particular node in the SMI Object Tree, or a set of multiple scalar objects, or a table object. Furthermore, for the sake of simplicity, the implementations of these methods will not be exemplified in the scope of this subsection. The signature and the body of the `SnmpGetSingleOID` method will be described in the following manner.
function SnmpGetSingleOID(oid)
{
    var uri = "http://"+dojo.byId("uGatewayIP") + dojo.byId("uAddress")
    + "/"+ dojo.byId("uCommunityString")+'/'+
    + replaceChars(oid, ".", "/");

    //more code here....
}

First, the value of the variable named uri is assigned on the basis of the HTTP Get request's URI scheme which is elaborated in section 3.4.1. Note that, every dot character “.” appearing in an original OID is replaced by the alternative character called slash “/” by invoking the replaceChars method.

dojo['xhrGet']({"url": url,handleAs: "text",
            load: function(xmlText){
            //do something
            },
            error: function(response){
            //do something
            },
            timeout: 10000
            });

Subsequently, a name of a desired method like HTTP Get is passed as a part of a function's name called dojo['xhrGet']. In this case, the key word called xhrGet refers to the name of the HTTP Get method. In principle, the function name is equivalent to the method call dojo.xhrGet which is provided as a part of the Dojo library. The assigned value of the variable uri is passed as the first argument of this function. The second argument handleAs determines a data type of a response text. In case of the dojo.xhrGet method, it asks the Dojo library to treat data returned from the gateway as a text string. The third argument of the method is the load function which is called on a successful response. The next argument is the callback function called error that is invoked if any error occurs. The final argument called timeout is given in milliseconds. The default value of this argument is set to 0, which means “wait forever”. In this particular case, the request command is expected to expire after 10000 milliseconds.

load: function(xmlText)
{
    dojo.byId("PanelDisplay").innerHTML = parseSnmpGet(xmlText);}

If the request will be completely performed, the browser will call the load handler function. In fact, two different arguments are passed as input parameters to this function. These arguments create a handy way of passing information to the load function. The first one called xmlText contains a response text. The second one passed implicitly to this function is the XMLHttpRequest object that is used to make a remote invocation. The gateway will normally return a response in XML format. The response will be parsed to HTML fragments by invoking the method parseSnmpGet. The innerHTML attribute of one object sets its inner text to the result of this method.
placeholder called Panel1Display is a right place to display the HTML fragments.

```javascript
try //Internet Explorer
{
    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async = "false";
    xmlDoc.loadXML(xmlText);
}
catch (e) //Firefox, Mozilla, Opera, etc{
    parser = new DOMParser();
    xmlDoc = parser.parseFromString(xmlText,"text/xml");
}
```

In principle, in order to manipulate an XML document, we need an XML parser which is normally integrated into any normal Web browser. As it is already mentioned, Web clients should be able to get rid off the critical cross-browser problem. Therefore, an XML document object must be properly instantiated in many different Web browsers, such as Internet Explorer, Firefox and Opera. Subsequently, an XML text will be loaded into this XML document object. At the final step, the text values of all elements of the XML document object will be extracted by traversing its DOM tree.

```javascript
var x = xmlDoc.getElementsByTagName("Object");
for (i = 0; i < x.length; i++)
{
    //Extract text values such as pairs of <OID,value>
}
```

In the case of the error handler function, the error message is also parsed in the analogous manner as the message indicating the successful request. Since the request is normally successfully executed, the detailed implementation of this function in Javascript will not be depicted in this section. In order to enable the Web-based client receiving Trap messages from the gateway, the setupComet method must be invoked when the main page is loaded.

```javascript
function setupComet()
{
    dojox.cometd.init('/cometd');
}
```

In case a network administrator desires to get a Trap that is bound to a specific OID, he clicks a button to invoke the subscribe function.

```javascript
function subscribe()
{
    var channel = "service/" + dojo.byId("uTrapOID");
    dojox.cometd.subscribe("/" + channel, trapCallBack);
}
```

The name of a callback function which is invoked when a Trap occurs is passed as an input argument to the subscribe method. The callback function is responsible for parsing the Trap message encoded in XML format into appropriate HTML fragments and displaying it in the Web browser.

```javascript
function trapCallBack(Msg){
    dojo.byId("TrapPanel").innerHTML = parseTrapMessage(Msg); }
```
Chapter 5 Evaluation

The following chapter gives an evaluation of the developed prototype for the proposed REST-based network management system under the criterion such as the fulfillment of the system requirements in terms of user centric perspectives (e.g. its supported functionalities, its Web user interface) or technological perspectives (e.g. its scalability, flexibility and its efficiency).

5.1 Evaluation System Requirements

5.1.1 User Centric Perspectives

The prototype implementations of the proposed REST-based SNMP gateway and the Ajax-based Web management tool constitute a complete SNMP network management utility. This tool enables network operators to perform not only the basic SNMP operations such as Get, Get-Next, Get-Bulk and Set but also the extended ones such as Table View or Tree Walk. In addition, it supports multi-varbind requests to allow grouping of multiple OIDs in a single request of SNMP operations such as Get and Get-Bulk. Users can therefore request to retrieve values of hundreds of MIB scalar objects by performing single SNMP request. Thanks to the support of the Cometd library, the desired function of the proposed solution to push Trap messages to registered Web clients right after they occur has been realized with the prototype implementation. The prototype implementation, therefore, meets the aforementioned essential system prerequisites regarding its full support for the SNMP infrastructure.

The Ajax-based Web network management application has a simple and intuitive graphic user interface that allows users to view the hierarchy of SNMP MIB variables in the form of a tree. This Web user interface allows system users to perform various functions in a mouse-click. For example, SNMP operations like Get and Get-Next can be performed by first clicking a node of this MIB tree to add its OID to a list. The next step is to select the appropriate operation to be executed from a drop-down menu. The results of such operations are displayed in one panel in an intuitive form of table with the two columns, OID and value. More complex SNMP operations such as Get-Bulk, Table View or Tree Walk can be performed in the similar manners like the above-mentioned simple ones, except that users must insert these complex operations specific parameters in another separated panel. It also provides a user-friendly view of SNMP table data. Moreover, it enables network administrators to view incoming Traps in a single panel called Trap Viewer after a message box is popped up. Because a Trap message in XML format can be parsed at the client side, so that users can look at its all details, notably IP of the original source that generates a Trap. A type of a Trap indicates the reason why the Trap is generated and a time stamp refers to the point in time when the Trap occurs will also be shown. Because of a diversity of the Dojo toolkit interesting features, the network management based on Web
2.0 results finally a smart and friendly designed Administrators GUI. By this Web application, all wining goals concerning substantial attributes of the presentation layer which has been introduced in section 3.1 are fulfilled. Figure 5.1 shows the Ajax-based Web client displaying the result received from the Gateway as a response to a request to get a content of the MIB table object, iftable, addressed by the OID of 1.3.6.1.2.1.2.2.

Figure 5.1: Execute SNMP-TableView

Web-based network management application has been implemented by using Dojo toolkit which supports a wide variety of browsers, notably IE 5.5+, Firefox 1.0+, Safari, Opera, Konqueror. Hence, cross-browser is known as the ability for a Website, Web application to support all Web browsers is not an issue anymore with the prototype implementation.

5.1.2 Technological Centric Perspectives

**Mapping of HTTP Methods to SNMP Operations**

The implementation of the gateway accomplishes its first prerequisite when it is required to bridge over the two different networks protocols, namely HTTP and SNMP. The reason is that the gateway works as a protocol converter that maps each HTTP method such as Get and Post to a corresponding SNMP operations based on some specific elements of
the HTTP request's URI. For example, it is previously said, that it is possible to map the HTTP Get method to the simple SNMP Get operation. Therefore, the REST-based gateway can act as the main access point for the Web-based network management application to SNMP objects. In its multi-threaded version, this gateway enables concurrent requests from many Web-based managers to SNMP managed objects.

**Flexibility**

Flexibility of the proposed solution is achieved by assigning each resource like an MIB object by its own URI. These assignments essentially give us an unlimited number of "Nouns", therefore it makes the resources addressing to be handled in the variability. Consequently, the proposed REST-based SNMP gateway can handle with a variety of MIB objects. In addition, it is possible to easily extend the gateway with additional SNMP MIBs by simply saving them as files in a specific directory that is managed by the gateway. No recompilation or restart of the gateway is necessary. Hence, the gateway can communicate and interact with any SNMP-enabled device assumed that MIB files provided by a vendor for this device can be loaded by the gateway.

**Interoperability**

The interoperability of the proposed system can be gained for the following reasons. Firstly, a URI which addresses one specific resource allows many Web applications can share the same link to this resource. Secondly, the universal interface implemented on a basis of REST constraints allows these applications to fetch the resource just with simple HTTP Get requests. Thirdly, these applications connect with the gateway via asynchronous communication channels, therefore temporal couplings between the gateway and its clients are avoidable.

**Scalability**

Since the HTTP protocol is stateless, the scalability of the whole system is increased because the gateway does not have to store state between different requests but quickly free resources. It results a capability to serve for requests of multiple Web clients simultaneously. In addition, the scalability of the system can be considered in terms of interactions between different components. Obviously, the interaction between Web clients and the gateway is enhanced by having limited number of operations or verbs. There are just two verbs, namely Get and Post, are applied for the implementation of the gateway's prototype implementation. In other respects, with a XML-RPC-based architecture, there is no limit to the number of verbs.

**Simplicity**

The stateless constraint enforced in the implementation of the REST-based SNMP gateway further simplifies its implementation because the gateway does not have to manage
resource usage across requests. Therefore, the implementation of the REST Uniform Component is kept very compact because it does not contain any functions or methods serving for maintaining states across different requests. The simplicity of the REST-based approach in the Master thesis is consequently improved in terms of complexity of the implementation. The generic interface of the gateway as described is what is believed to make the REST-based approach used for implementing the gateway much simpler than the similar SOAP-based and XML-RPC approaches mentioned in the paper[5] because it facilitates communications between Web clients and the gateway. The reason is that the same verb can be applied to different nouns which are just like the same operation as the HTTP Get method can be applied to different resources (e.g. MIB columnar objects or scalar objects). In contrast, with the SOAP-based approach or XML-RPC-based approach, all the method names, addressing model and procedural conventions of a particular service must be known in advance. Furthermore, HTTP clients as the Ajax-based network management applications can communicate with any HTTP server like Jetty without any further configuration. The reason for these arguments is clear because HTTP is an application protocol whereas SOAP and XML-RPC are framework's protocols.

**Security**

It is mentioned that the security issues of the system has to be considered seriously. It is worth to remember that the prototype implementation supports only SNMPv2c. However, the SNMPv2c lacks of security features, notably authentication and privacy, which are added in the recent set of RFCs, known collectively as SNMPv3. Nevertheless, the prototype implementation offers the alternative to these security features of SNMPv3 by providing Login page to authenticate system users. In addition, the communication channels between the gateway and Web clients are secured by using HTTPS. In other respects, many SNMP agents can support only SNMPv1/v2c. Therefore any support for SNMPv3 might drive the final solution to become unnecessarily more complicated because it is necessary to provide a component to translate SNMPv3 requests for SNMPv1/v2cagents and reconstruct SNMPv1/v2c responses back to SNMPv3 ones.

**Operating System Independent**

Note that many components which constitute REST-based gateway are developed by using Java programming language. In addition, Java is the OS (Operating System) independent programming language. Thus, the gateway can be deployed in a various number of OS platforms as it is required in subsection 3.1.

**Efficiency**

There were a lot of online articles [24, 25, 26] advocating a better efficiency of the REST over SOAP, XML-based RPC and so forth. According to the experiment results which will be elaborated in the next section 5.2, once again, the efficiency of the REST-
based approach over the SOAP-based and the XML RPC-based methods to the specific problem of the Master thesis will be confirmed.

5.2 Performance Analyze

In this section, the result of the experiment to analyze the efficiency of the proposed solution will be introduced.

5.2.1 Goals and Setup

The goals of the experiment consist of exploring the actual performance trade-offs of the implementation of the proposed solution integrating REST-based SNMP gateway with the legacy SNMP infrastructure and compare it to a pure approach by executing SNMP requests with an SNMP command line tool. The experiment has to done in case a big set of SNMP MIB objects should be fetched simultaneously with and without the involvement of the intermediate gateway. The experiment should be repeated many times and the final results should be calculated as an average value. This experiment aims to prove that the proposed solution in the Master thesis is efficient.

5.2.2 Testing Environment and Testing Tools

The experiments will be conducted in the LAN network that comprises two personal computers. The LAN network is connected with 100 Mbit/s Ethernet connection. These first personal computer has the CPU Intel Core Duo T2450 2 GHz, 895 MB RAM and Windows Vista Business as its Operating System. The second PC is equipped with the CPU AMD Athlon 1,6 GHz, 512 MB RAM and its Operating System is Windows XP. The JREs (Java Runtime Environment) which have been installed in these two computers have the version 1.6.0 and 1.5.1 respectively. The Web browsers used for the experiment are Internet explorer and Firefox because these two Web browsers are the most popular ones in the market at the moment. Jetty is used as an HTTP server because it is the only open-source Java application server that currently implements the Comet Bayeux protocol.

Wireshark [16] which is the free packet sniffer computer application is used to capture and analyze packets sent between two computers. Wireshark is selected among a lot of tools supporting for sniffer network because it is the open-source, cross-platform that is an extension of the well-known network analyzer Ethereal. Especially, Wireshark supports a variety of network protocols including SNMP. With the support of Wireshark tool, the round-trip time of any request based on HTTP protocol or a proprietary such as SNMP can be measured.

Net-SNMP [17] is the open-source suite of applications used to implement SNMPv1, SNMPv2c and SNMPv3. A command-line application of Net-SNMP seemed to be a good option, providing a tool to retrieve information from SNMP, either using single requests or multiple requests. In order to simulate SNMP-enable devices, the extensible SNMP
agent is provided as a part of this suite which should be installed in one computer for a testing purpose. The agent is accountable for answering queries to retrieve or modify SNMP management information. Moreover, the Net-SNMP agent includes built-in support for a wide range of MIB information modules.

5.2.3 Procedures of Performance Analysis

The routine test run consists of several steps. The first test run should be done with the SNMP command line tool such as Net-SNMP. The command-line application is installed in a different machine where a simulation SNMP agent is running. The retrieval of the complex MIB columnar object with a huge number of columns and rows is performed. The round-trip time of this request and response is calculated on a basis of the average value of 5 times performing the same request.

![Net-SNMP command line](image1)

Figure 5.2: Test case with SNMP command line tool

The second test run will be conducted with the involvement of the REST-based SNMP gateway. Requests to retrieve the same sample MIB object will be sent from Web browsers, Internet Explorer and Firefox. Regarding the second test run, two separated evaluation scenarios have been carried out with an involvement of the intermediate gateway. In the first case, an efficiency of the pure SNMP-based communication between the gateway and SNMP agents is measured. Therefore, the simulation SNMP agent software is installed in one computer. The REST-based SNMP gateway and the Ajax-based network management run one the different one. Wireshark is used to capture data exchanged on a basis of the SNMP protocol between the gateway and the SNMP agent.

![REST-based SNMP Gateway](image2)

Figure 5.3: Test case 1 with the participation of the gateway.
In the second scenario with the participation of the gateway, the Web-based network management application sends HTTP request over the LAN network to the gateway which is located in the same machine with the simulation SNMP agent software. This scenario is depicted out in the following figure 5.4. In this case, the access to SNMP infrastructure is built on the end-to-end communication based on the HTTP protocol between the Ajax-based Web client and the gateway.

Figure 5.4: Test case 2 with the participation of the gateway

5.2.4 Results

The complete experiments to retrieve a content of the sample MIB object, *ifTable*, have been realized on a basis of command line SNMP tools and with the participation of the REST-based gateway implementation. This table contains 24 columns and 19 rows. If a request is realized by SNMP command line tools without an intermediate gateway, it takes 1,725 seconds on average.

In the first case, an efficiency of the pure SNMP-based communication between the gateway and SNMP agents is measured. The round-trip time required to receive a response of a command to obtain the *ifTable* object over the SNMP link between SNMP agent and the gateway is approximately 1,776 seconds on average.

In the second case, a Web client directly sends a Get-Table request to the gateway in order to query a content of the aforementioned MIB tabular object from SNMP agents standing behind the gateway. It takes totally only 3,182 seconds on average.

5.2.5 Discussion Performance Analysis

According to the results of the performance analysis exposed the previous subsection 5.2.4, it has shown that the implementation of the SNMP Adapter component of the gateway in Java offers a very slightly worse performance in comparison with the implementation of the command line tool in C++ such as Net-SNMP. The speed of communication is only fallen back with the factor of 1.03 times. This experiment has also
shown that whilst the results of the SNMP communication channel between SNMP agent and gateway are evaluated and encapsulated into an XML representation and access is based on HTTP, the end-to-end communication is only slowed down only 1.844 times. The performance analysis showed that the REST-based SNMP gateway approach does not generate an unacceptable processing overhead compared with the existing management paradigm, the pure SNMP clients/managers model.
Chapter 6 Conclusions and Future Works

In the scope of the Master thesis, the Web-based network management system has been proposed to meet the needs of modern NWM tools. The suggested system is based on SNMP which is used in network management systems to monitor network-attached devices. However, the main drawback of SNMP is identified as lacking of easy integration into other tools and 3rd party software. The own gateway approach for managing the existing legacy agents based on constraints of REST (Representation State Transfer) architecture style has been explained in order to eliminate this shortcoming. For validation, the REST-based SNMP gateway and Ajax-based network management application have been designed and implemented based on the proposed Web-based client and gateway architecture. The design and implementation of the REST-based SNMP gateway partially follow REST’s architectural principles and constraints such as resource, resource representation, uniform interface, stateless interaction. In terms of the implementation of the gateway, REST is chosen due to its many advantages such as simplicity, scalability, interoperability and flexibility. The Web-based network management application based on Ajax technologies is also designed and developed as a part of the overall network management solution to prove that loose integration of tools is possible. With the smart and friendly designed UI (User Interface) of the Web-based client, authorized users are enabled to easily carry out requests to fetch or alter MIB objects.

Auto discovery can be thought as a mechanism that enables new devices discovered in the network to be polled automatically. Since the suggested REST-based SNMP gateway does not manage any information about the potential communication partners such as SNMP agents, it will be a target for future researches the profitableness provided by the auto discovery feature which can be implemented as a part of the gateway.

As it is mentioned above, SNMP is only one typical example of legacy applications which can be integrated to the REST-based gateway. In general, the REST-based gateway provides a universal interface based on REST constraints while the design of CMIP (Common Management Information Protocol) is similar to SNMP. Therefore, it is possible to integrate another legacy infrastructure like CMIP into a Web-based management application via the gateway which might be extended to meet requirements of this legacy application. Regarding of a possibility to integrate a Web-based client into CMIP, two most essential works have to be done such as specifying and implementing the mapping from HTTP methods (e.g. Get, Post, Delete, Put) into CMIP operations (e.g. Action, Cancel, Create, Delete, Get, Set) and assigning URIs to managed objects of CMIP management agents. For the future work, a research on extending the REST-based gateway which integrates a Web-based network management application into the other network management infrastructure like CMIP can be done in a scope of another Master thesis.

WADL (Web Application Description Language) is designed to provide a simple alternative of WSDL in XML/HTTP Web application. WADL is claimed to program easily a Web-based network management application because a WADL file, which can be
complied to a target programing language by using an appropriate WADL parser, describes the following essential things: which resources exposed by the uniform interface, which URIs the Web clients can visit, what data those URIs expect the Web clients to send and what data the REST uniform interface serve in return. Therefore, in terms of many benefits given by WADL, the future goal of this thesis is to extend the proposed solution by describing the uniform interface which uses WADL.
References


72
23. AOL, WWW Page, www.aol.com
37. JSON-RPC, WWW Page, www.json-rpc.org
38. SmarClient, WWW Page, www.smartclient.com
40. Backcase, WWW Page, www.backcase.org
41. Ruby on Rails, WWW Page, www.rubyonrails.org
42. Tapestry, WWW Page, www.tapestry.apache.org
44. ASP.Net, WWW Page, www.asp.net

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASN.1</td>
<td>Abstract Syntax Notation 1</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous Javascript And XML</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>NWM</td>
<td>Commercial Network Management</td>
</tr>
<tr>
<td>CGI</td>
<td>Common Gateway Interface</td>
</tr>
<tr>
<td>CMIP</td>
<td>Common Management Information Protocol</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IE</td>
<td>Internet Explorer</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>MIB</td>
<td>Management Information Base</td>
</tr>
<tr>
<td>MB</td>
<td>Mega Byte</td>
</tr>
<tr>
<td>NMF</td>
<td>Network Management Framework</td>
</tr>
<tr>
<td>NMS</td>
<td>Network Management Station</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PDU</td>
<td>Protocol Data Unit</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedural Call</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>SNMP</td>
<td>Simple Network Management Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SMI</td>
<td>Structure of Management Information</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>URI</td>
<td>Universal Resource Identifier</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
</tbody>
</table>
1. Changing the value of the *mib.dir* variable in the file *mib.properties* with an appropriate file path.

2. Copying all files in the directory Dojo to one directory and changing the value of the *web.dir* variable in the file *server.properties*.

3. Changing default values of the *user name* and *password* variables in the file *server.properties* if there is a need. Default values are *admin* and *test* respectively.

4. Changing the value of the *trapreceiver* variable to the value of an IP address of a PC where the gateway will run.

5. The Jetty server can be started on the port 80 as an embedded server by running the *MainClass* class.

6. Typing in a Web browser's address box [http://localhost/Login.html](http://localhost/Login.html)

7. Inserting the correct values of user name and password in the Login page.

8. The main page will be displayed after the user name and the password is verified.
1. Executing the SNMP Get operation.

<table>
<thead>
<tr>
<th>IP of SNMP Agent</th>
<th>List of OIDs</th>
<th>Result</th>
<th>Select Get operation</th>
</tr>
</thead>
</table>

[Image showing a window with a list of OIDs and a table with results]
1. Setting parameters, such as Max-Repetition and Non-Repeaters of the SNMP Get-Bulk. The Max-Repetition is set to 10 and the Non-Repeaters is set to 2. The number of variable bindings is 3.

2. Executing the SNMP Get-Bulk operation.
1. Clicking on the *Subscribe* button to receive the Trap *Link Down*.

2. Popping up the message box to notify about the occurrence of the Trap. The Trap will be displayed below.