#### 1. Introduction

* **Project Overview**: This project aims to build a simple SNMP manager application using the SNMP4J and Mibble libraries in Java. The application will gather SNMP data from servers and network devices, providing an object-oriented interface for interaction.
* **Objectives**: The main objective is to create an SNMP manager that supports SNMPv1 and SNMPv2c, with a design that allows easy extension to SNMPv3 in the future.
* **Scope**: The project focuses on developing only the manager part of the SNMP framework.

#### 2. Background

* **SNMP Protocol Overview**: Simple Network Management Protocol (SNMP) is an Internet-standard protocol for collecting and organizing information about managed devices on IP networks and for modifying that information to change device behavior.
* **SNMP4J Library**: SNMP4J is an open-source Java library for SNMP, providing a comprehensive API for building SNMP applications.
* **Mibble Library**: Mibble is a library for parsing SNMP MIB files in Java, facilitating the extraction of MIB information.

**Goal:**

Get information from network devices

Analyize result

**3. Overview of SNMP**

Simple Network Management Protocol (SNMP) is an application-layer protocol for monitoring and managing network elements. It consists of SNMP Manager, SNMP Agents, Management Information Base (MIB), and Managed Devices.

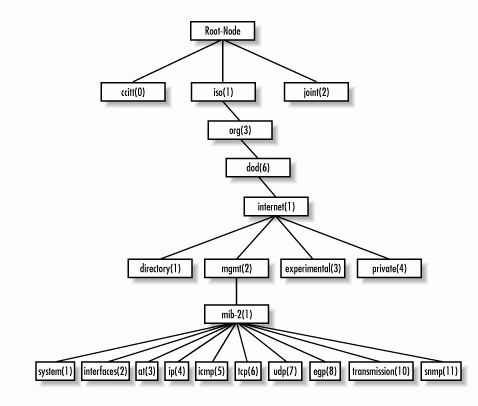
SNMP manages “objects”, each indentified with specific OID, with mappings defined in the MIB provided by agent (device manufacturer)

Communication is facilitated through Protocol Data Unit (PDU):

* get request, get next request, get bulk request: manager queries for specific variables
* response: agents answer to queries
* set request: change agent variables
* trap, also known as alert or alarm: async event triggered at agents
* inform: like traps, but acknowledged by manager

**3.1 SNMP MIB**

The Management Information Base is a hierarchical database of network management information. It can be thought of as a tree structure. Each leaves is a mapping of object id to the actual objects, which in turn are variables set by the agents. This variable can be device name, sensor readings, ip address, statistics, routing table elements, etc...



The MIB provide an easy way for the manager to know what OID to query and interpret agents’ response. MIB maps datapath to meaningful infromation, and is used by both the manager and the agents.

**3.2 SNMP Manager**

The manager is the interface between human and the management system, acting as a central for monitoring and control. A manger may monitor many agents, and an agents may report to many managers. It’s main functionalities are:

* Send request to agents
* Parse and report agents response
* Listen for agents’ async events
* Set agents’ variables at user’s request
* Understand and display information to user.

**3.3 SNMP Agents**

Typically, an SNMP agent is a software module running on a managed device, such as a router, switch, server, or any network-attached device. Each device run its own SNMP agent service, which in turn monitors many arbitrary SNMP objects. The primary responsibilities of SNMP agents include:

* store and manage data about the device in a MIB
* listen for SNMP requests from the manager and provide the requested data. This includes responding to GET, GETNEXT, and GETBULK requests to retrieve data and SET requests to modify data or perform actions like automated tests
* send traps (alerts) to the SNMP manager when certain conditions are met or when specific events occur, such as device reboots, interface failures, or threshold breaches.

**3.4 Protocol Data Unit**

PDUs are fundamental building blocks in the communication in SNMP protocols. As previously mentioned, PDU convey requests, responses and notifications between the manager and agents. Information is sent in the format known as Variable binding (varBind), a combination of data path (OID) and object value.

* GetRequest: sent by the manager to agent to request for one or more object value of managed device.
  + Example: get(7.1.0) → response(7.1.0, 192.168.1.1)
* GetNextRequest: retrieve the value of the next object, useful for objects in table structures, like the routing table of a network device
* GetBulkRequest: retrieve a large amount of data at once, introduced in version 2 of SNMP to address the short comings of version 1
* SetRequest: sent by the manager to modify (inlcuding unset) values and configurations of manged device, also trigger actions such as reboot or turning off interface.
* Response: sent by agent to answer the above PDUs. Returns a VarBind of OID and relevant value. May also contain error information. Version 2 added exceptions to clearer error handling.
* Trap: Also known as alerts, agents automatically notfies manager when a triggering event is occurred, like failures, high CPU usage, etc. However, whether the manager receive the message or not confirmed, hence the unreliability. Manager poll the agents to listen for traps. Agents can also be configured to not send traps or only send to certain managers.
* InformRequest: Introduced in version 2 to address the problem with traps, it require an acknowledgement from the manager.

**4. Design**

**4.1. High level design**

The manager has two component, manager and client. Manager facilitates communication with agents, while client helps end-user easily use manager functions.

**Agent management:**

* The SNMP manager follows the **Facade** structural design pattern, aimed to be an interface to simplify the process of interacting with snmp4j. This pattern abstracts the complexities of the underlying libraries, offering a unified and “easy-to-use” interface. It is a tradeoff between simplicity and functionality.
* In order for snmp4j to be able to send a request, it must perform a variety of tasks, including setting up transport layer communication, specifying target, constructing PDU, sending request and parsing the returning response. This operation contain a lot of moving parts that must be performed sequentially. The **Facade** aims to simplify this into singular function calls.
  + INPUT: OID, IP address
  + OUTPUT: Response string
* Trap/inform listener works differently from the rest. Manager must continuously poll/listen to agents for async events, and as such is implemented as an additional listener Facade class. It work with the client code through an Observer pattern.

**Client code:**

* The application also employs the **Observer** design pattern to manage listener/handler functionality as well as communication with UI elements. In this pattern, the manager acts as the **publisher** that maintains a data structure of “agent” instances encapsulating the state (VarBinds) and registered **subscribers** (front-end elements).
* When an SNMP agent emits a response or event, the target manager updates the state then notifies registered subscribers. This pattern provide a decoupled and scalable way to handle events, allowing the manager to work independently of front-end elements, work with multiple front-end elements at once and keep track of managed agents.
  + Whenever an important event occurs in the publisher, it iterates over its subscribers and calls a specific notification method on each subscriber. This method is part of a common interface that all subscribing front-end elements must implement. This interface ensures that the publisher can notify subscribers without knowing their specifics.
  + Front-ends interact with the manager by calling `subscribe()` to manager-managed agents. Each managed agent’s state is generated from the MIB and received data from the agent. Modifications to the state are optionally emitted to front-end subscribers, enabling real-time updates. Additionally, the front end can call `update()` to retrieve the entire state as needed.
* This **Observer** pattern allows the front-end to be implemented in any format or framework. The manager does not need to be concerned with the inner working of the front-end and simply passing on data. Similary, the front-end does not need to know how the manager work and inner classes, and simply implement the interface to understand the manager. This low coupling and independent allows flexible implementation and extension of elements without affecting others.

**4.2 Class diagram**