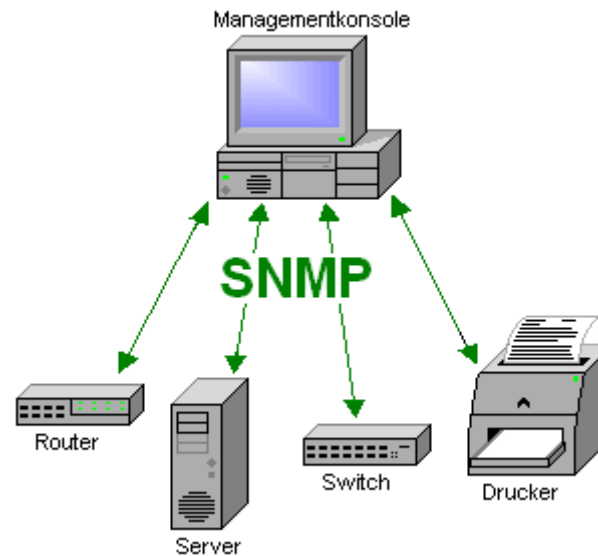


Project SNMP Client



By : Mohammed Akram MECHERI

Professeur: M. VERLAN

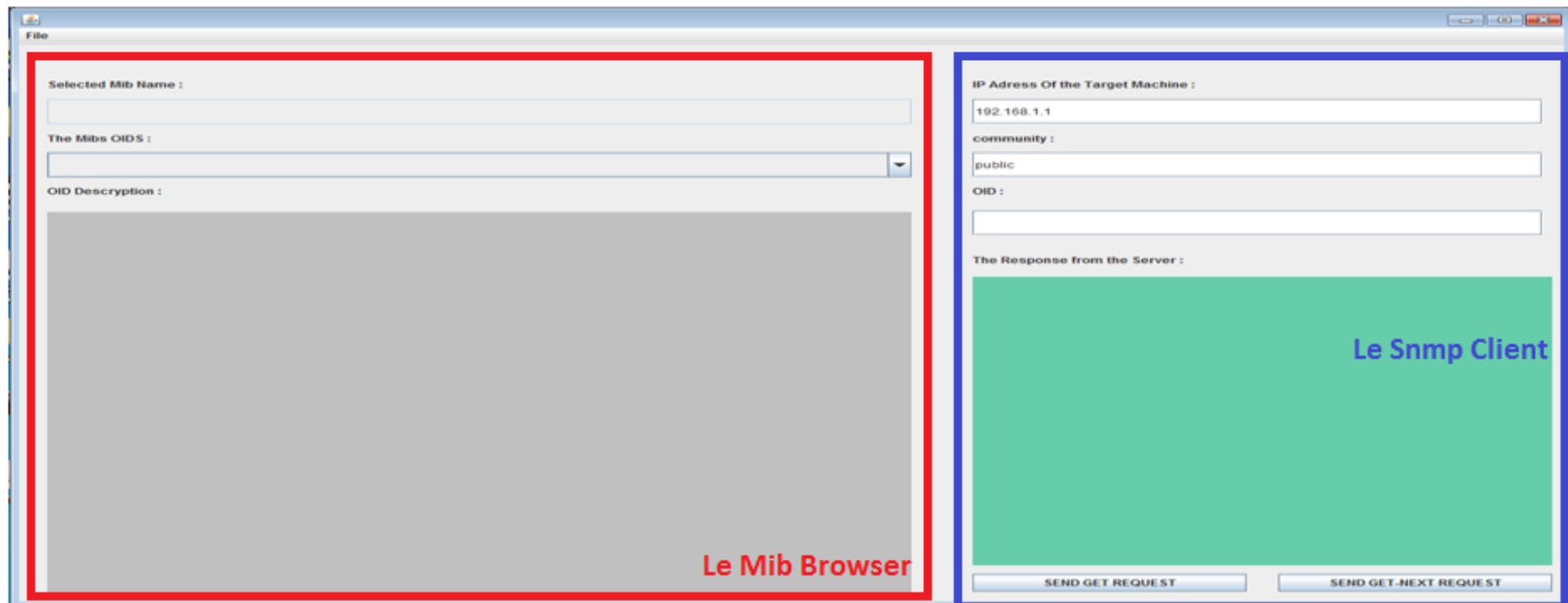
SNMP Client :

Ce projet est un client SNMP qui permet de construire et d'envoyer deux types de requêtes :

- *Get*
- *Get-Next* (*Fonctionnalité Bonus*)

et qui reçoit des réponses et les traduit. en format compréhensible.

Ce projet est aussi un navigateur de Fichiers Mib (*Fonctionnalité Bonus2*) qui lit les Fichier.mib et en extrait les Oid (Object Identifier) et leurs descriptions.



Le Mib Browser(Partie Gauche) et le constructeur de requêtes(Partie Droite) peuvent fonctionner indépendamment l'un de l'autre mais on peut aussi auto-remplir le champ Oid à partir du Mib Browser et voir sa description dans le champs Oid Description et sa valeur dans la target machine dans le champs The response from the server.

Difficultés Rencontrées :

- Ce projet est le fruit de deux mois de recherche et de programmation.
- La difficulté de sa réalisation a été principalement l'inexistence de projet similaire sur le net (Manque de références) et la difficulté de lire les RFC officiels.
- Une autre difficulté est due au fait que Snmp est un protocole qui fonctionne en UDP donc il nous a fallu construire les requêtes avec des bits précis.
- L'interprétation des réponses du serveur est aussi compliqué à cause du nombre de variables que le serveur peut envoyer et leur codage compliqué.
- Au début du projet on est parti plusieurs fois sur de mauvaises pistes, mais ça nous a permis de mieux comprendre le protocole.
- Au début du projet on ne comprenait rien dans la notion Serveur/Client et on ne savait même pas de quel côté était située notre application, mais après quelques Td et Tp et avec l'aide du M Verlan on a mieux compris tout ça.

Bibliothèques :(Hors JDK)

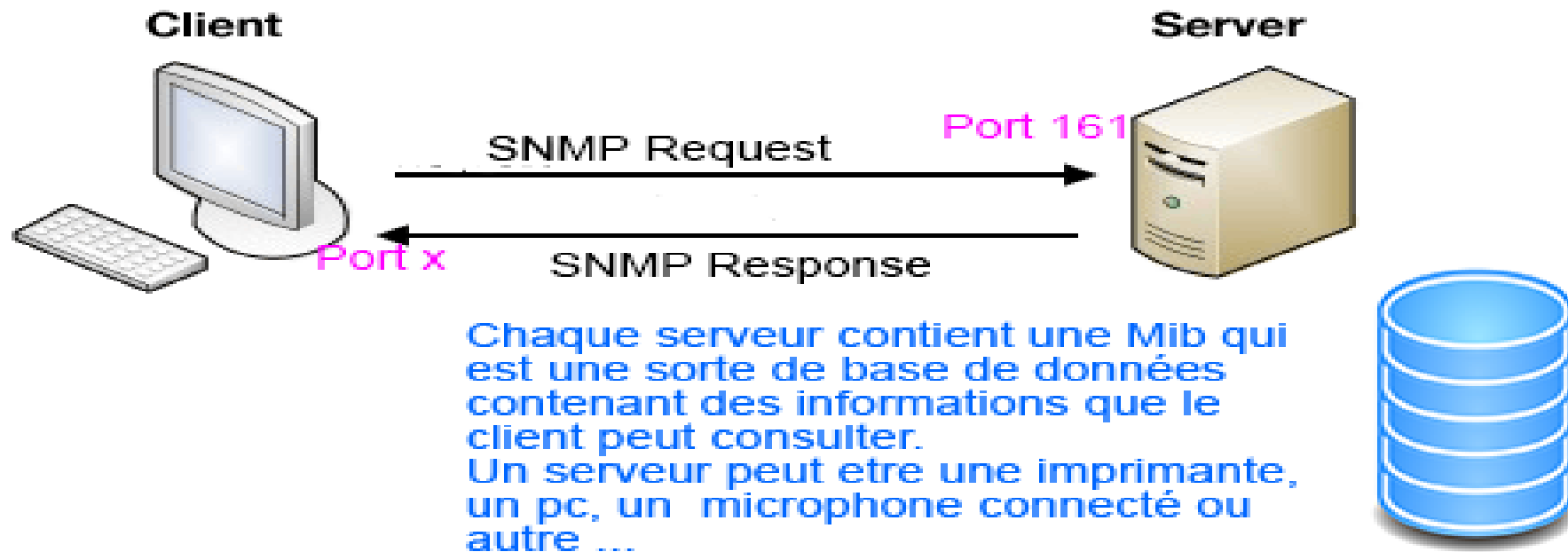
Pour interpréter les variables on a créé nos propres algorithmes pour les String et Int et Oid object. et on a utilisé une Bibliothèque **Snmp4j** pour interpréter le type **ip** et le type **TimeTick**.

On a utilisé la bibliothèque **mibbe** pour la lecture des fichiers Mib et on a créé nos propres algorithmes pour en extraire les informations des Oid.

Le Protocol Snmp :

Ce protocole qui fonctionne sur le port Udp 161 est un protocole Binaire donc la première réflexion est de communiquer en DatagramSocket pour atteindre la cible.

La problématique est de contruire le bon paquet de bytes pour que le serveur Snmp puisse comprendre la requête et envoyer le résultat, comme dans le schéma ci-dessous :



Pour que la connexion soit possible il faut que le Serveur Soit à l'écoute sur le port 161 ce qui n'est pas toujours le cas par défaut selon l'équipement par exemple mon routeur Bouygues n'a jamais voulu répondre à mes requêtes ou encore mon deuxième ordinateur qui est sous windows qui ne lance pas le serveur Snmp automatiquement au démarrage ce qui nous faisait croire au début que le problème venait de notre application vu que 2 équipements ne répondait pas à nos requêtes...

Mais ceci nous a permis de bien comprendre le principe Serveur/Client en Snmp :

Serveur	Client
à l'écoute sur le port 161	Crée une DatagramSocket sur n'importe quel port x disponible et envoi un DatagramPacket au serveur au port 161.
Le serveur reçoit le DatagramPacket du Client sur son port 161 et de source port x du client. Il envoi sa réponse au client sur le port x	

La Forme d'un DatagramPacket:

On utilisant le logiciel WireShark on peut voir à quoi ressemble un DatagramPacket envoyé à mon deuxième ordinateur qui tourne un serveur Snmp (net-Snmp) :

Quelque info avant de commencer :

- Ip client : 192.168.1.2
- Ip Serveur(Deuxième pc): 192.168.1.3
- communauté (une sorte de Nom d'utilisateur on en parlera plus loin) : ALLO
- Oid demandé (System description) :1.3.6.1.2.1.1.1.0

DatagramHacking ;)

En tête du Paquet contient des infos comme Destination Ip: 192.168.1.3, Destination Port: 161,

Source ip: 192.168.1.2 Source Port: 52982 type de protocole: UDP...

Le message Snmp Qu'on va expliquer byte par byte !

60 d8 19 0e a5 f6 74 27 ea be 46 b1 08 00 45 00 00 42 cb 4d 40 00 40 11 ec 07 c0 a8 01 02 c0 a8
01 03 ce f6 00 a1 00 2e 83 95 30 24 02 01 01 04 04 41 4c 4c 4f a0 19 02 01 01 02 01 00 02 01 00
30 0e 30 0c 06 08 2b 06 01 02 01 01 01 00 05 00

Avant d'expliquer le message Snmp on va voir sur quel model de codage repose ce dernier, Le codage du message Snmp est inspiré du model ASN.1 BER de télécommunication les règles d'encodage de ce model est utilisé pour les communication de téléphonie portable et dans d'autres protocoles de télécommunication..

La règle la plus fondamentale de ce model est que chaque champ du message est codé sur trois entités comme suivant :

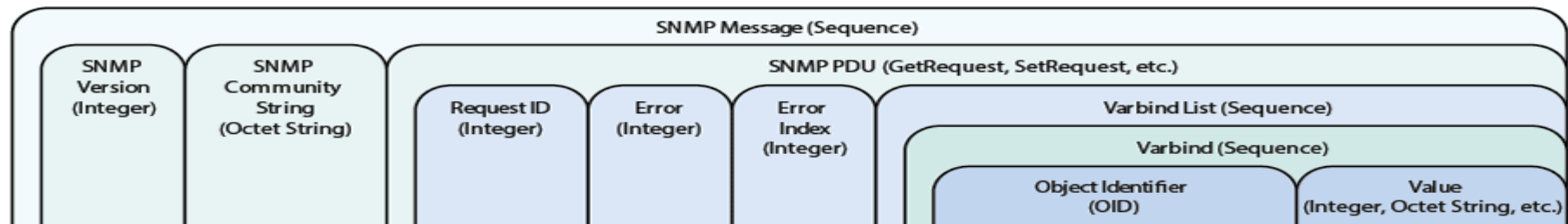
Type de la donnée	Longueur de donnée	La donnée
-------------------	--------------------	-----------

il existe des données composée qui contiennent elles mêmes d'autres données avec leurs types et leurs longueurs...

Exemple :

Type de donnée Composée	Longueur de donnée Composée	Type de donnée	Longueur de donnée	La donnée	Type de donnée	Longueur de donnée	La donnée
		La donnée Composée					

Voici les champs et les champs composés qui composent un message Snmp:

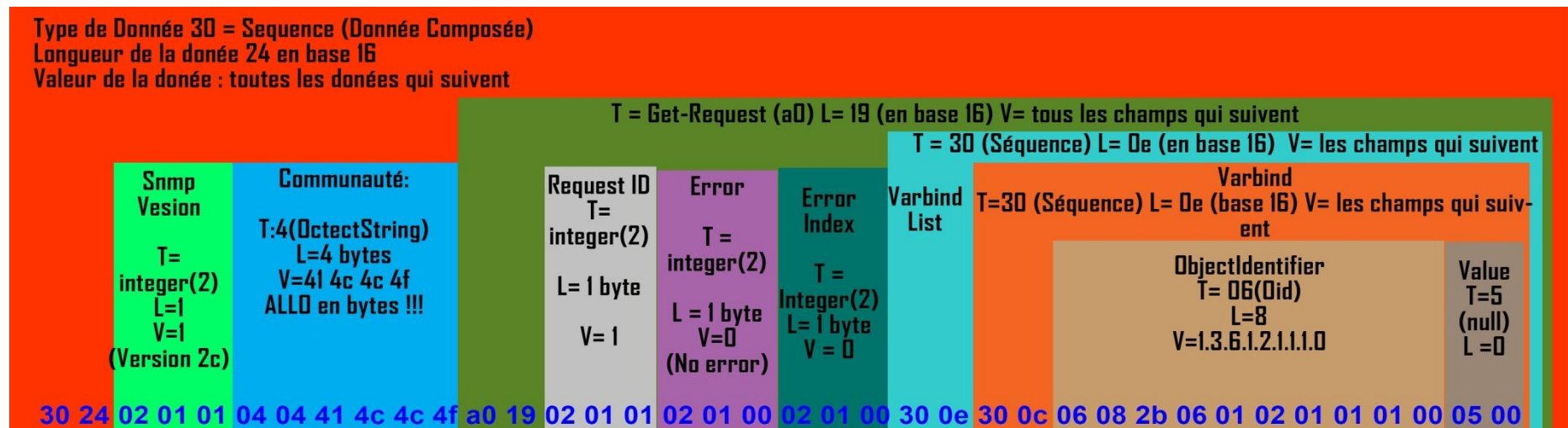


Explication des bytes du message envoyé:

Ces bytes representent une Get-Request du Oid 1.3.6.1.2.1.1.1.0 pour la communauté ALLO de mon Serveur Snmp, une telle requête contient des **bytes constants** et des **bytes variables** :

30 24 02 01 01 04 04 41 4c 4c 4f a0 19 02 01 01 02 01 00 02 01 00 | 30 0e 30 0c 06 08 2b 06 01 02 01 01 01 00 05 00

les bytes variables varient selon le nom de la communauté et le Oid demandé, la variation de ces deux données simples fait varier les données Composées qui les contiennent il faut prendre ça en considération lors de la création du DatagramPacket !!!



T = Type de la donnée
L = Longueur de la donnée
V = Valeur de la donnée

Description des Champs de tout les messages Snmp :

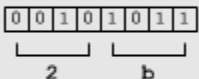
Champ	Description
SNMP message	Une séquence(0x30) qui représente tout le message snmp (version, communauté....
SNMP version	Un Integer qui represente la version snmp du message v1=0x00 v2c=0x01 ...
Communauté	Une chaine de caractères qui représente la communauté qui est un nom d'utilisateur sur le serveur Snmp, par défaut sur la plupart des périphériques ce nom d'utilisateur est "public" sur mon serveur c'est "ALLO"(modif perso).
SNMP PDU	Le Type de la requête get-Request = 0xA0 get-Next-Request=0xA1
Request-ID	Un Id qui permet le client Snmp d'identifier une reponse du serveur
Error	Un integer qui permet de dire si le serveur a rencontrer un problème ou pas 0x00 Pas de probleme 0x01 Message très long pour le transport 0x02 Oid introuvable 0x03 Un type de donnés dans la requête envoyé par le client n'est pas reconnu par le serveur . 0x04 Le client a essayé modifier a un paramètre avec écriture non-autorisée 0x05 Général error
Error Index	Si il y a erreur l'index prend un pointeur vers l'objet qui a causer l'erreur
Varbind List	Une séquence de Varbinds
Varbind	Une séquence de deux champs: Oid et la valeur de ce Oid
Oid	Un Id qui pointe vers un paramètre précis dans le serveur

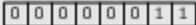
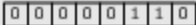
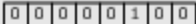
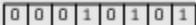
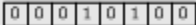
Value	La valeur retournée elle est nulle dans une requête Get et elle peut prendre plusieurs types de valeur dans un get-response (String , Integer, Ip, Oid, TimeTick..).
-------	--

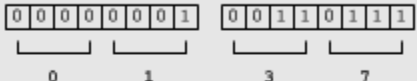
Pour une Get-Request la difficulté de codage réside dans le codage du Oid si on ne veut pas utiliser la classe Oid qui existe depuis le JDK 1.4 !

son codage est le suivant :

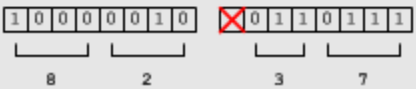
OID:
1.3.6.1.4.1.311.21.20 (ClientId Attribute)

1) Encoding the First Two Nodes:
 $1 \times 40 + 3 = 43d = 0x2B =$



2) Single byte encoding of all remaining nodes other than 311:
 $3 = 0x03 =$ 
 $6 = 0x06 =$ 
 $4 = 0x04 =$ 
 $21 = 0x15 =$ 
 $20 = 0x14 =$ 
 $=$

3) Multiple byte encoding of 311:
 $311d = 0x0137 =$


This is encoded to 0x82 0x37 by:
 1) Setting bit 7 of the leftmost byte to 1.
 2) Ignoring bit 7 of the rightmost byte.
 3) Shifting the right nibble of the leftmost byte to the left by 1 bit.



When decoded, the bits are assembled in the following manner:



3) Summary encoding of OID : 1.3.6.1.4.1.311.21.20

0x2B 0x06 0x01 0x04 0x01 0x82 0x37 0x15 0x14

Nous avons essayé de coder nos Oid sans utiliser la classe en Oid du JDK 1.4 en codant le Oid bit par bit en utilisant la classe BitSet mais nous avons rencontré beaucoup de problèmes quand un champs du Oid est supérieur à 127 et qu'il est codé sur 2 octets ou plus nous étions alors obligé d'utiliser la classe Oid.

On a vu donc comment est codé une requête Get et on a donné assez d'indices pour construire d'autres requêtes (Get-Next , Set ...)

Réponse Du serveur :

Le serveur nous répond sur le même port avec une Get-Response dans un DatagramPacket sur le même port qu'on a utilisé pour lui envoyé une requête Get-Request, pour décoder ce paquet on fait l'opération inverse du codage ...

En tête Du DatagramPacket

Message Snmp

Valeur retournée

```
74 27 ea be 46 b1 60 d8 19 0e a5 f6 08 00 45 00 00 c2 00 2e 00 00 80 11 b6 a7 c0 a8 01 03 c0 a8 01 02 00
a1 ce f6 00 ae 1b c4 30 81 a3 02 01 01 04 04 41 4c 4c 4f a2 81 97 02 01 01 02 01 00 02 01 00 30 81 8b 30
81 88 06 08 2b 06 01 02 01 01 01 00 04 7c 48 61 72 64 77 61 72 65 3a 20 78 38 36 20 46 61 6d 69 6c 79 20
32 30 20 4d 6f 64 65 6c 20 32 20 53 74 65 70 70 69 6e 67 20 30 20 41 54 2f 41 54 20 43 4f 4d 50 41 54 49
42 4c 45 20 2d 20 53 6f 66 74 77 61 72 65 3a 20 57 69 6e 64 6f 77 73 20 56 65 72 73 69 6f 6e 20 36 2e 31
20 28 42 75 69 6c 64 20 37 36 30 30 20 4d 75 6c 74 69 70 72 6f 63 65 73 73 6f 72 20 46 72 65 65 29
```

Avant les octets du message retourné on voit le octet qui représente sa longueur (0x7c) et le octet qui représente le type de la réponse (0x04) qui est OctetString en le traduisant on obtient :

Hardware: x86 Family 20 Model 2 Stepping 0 AT/AT COMPATIBLE - Software: Windows Version 6.1
(Build 7600 Multiprocessor Free)

Qui correspond a la description de mon serveur SNMP.

Activity Diagram

