



L .BALBIANI F .EL JAMAL M.TEIKITUHAAHAA

Projet Java: SS Manager

Rapport de la version 3

Ce rapport présente les spécifications pour la version 1 du projet SNMP Java de la 2ème année d’ingénieure STRI.

# Cahier des charges

Voici ce qui est demandé pour la version 1 :

*Les agents doivent maintenant pouvoir prévenir les managers au travers d’évènements sur le modèle des*

*traps SNMP. Il faudra aussi, à cette étape, prévoir une opération d’abonnement des agents aux traps.*

On programmera donc un Agent SNMP qui enverra des TRAP à un Manager, et un Manager qui recevoir des traps et qui signaleront à un utilisateur via une IHM.

# Architecture du programme

Pour la version 3, on respecte le même modèle en couche que dans la version 1.

## Modèle en couche

Ce modèle en couche permet de représenté plus facile les ensembles de fonctionnalités de chaque classe. Elle sera valide tout au long du projet.

### IHM

Cette couche représente l’interface graphique du Manager/Agent et a les fonctionnalités suivantes :

* Affichages des résultats des requêtes ;
* Affichages des notifications des agents (TRAP SNMP) ;
* Envoi de requêtes ponctuelles.

### SNMP Kernel

C’est le noyau du programme et il est constitué soit du Manager ou de l’Agent SNMP pour cette version. Dans les prochaines versions, on tentera du fusionner ces entités qui devront partager les ressources communes comme les sockets, et faire en sorte de bien router les messages SNMP vers les bonnes entités.

### SNMP Message Handler

Cette couche a les fonctionnalités suivantes :

* Conversion des Messages SNMP en *DatagramPacket* et ou en *SNMPMessage* (voir chapitre 3 sur les Diagrammes) ;
* Gestion des retransmissions des Datagrammes (pour les versions suivantes).

### SNMP Socket Handler

Cette couche a les fonctionnalités suivantes :

* Récupération ou envoi des Datagrammes vers la couche Transport (UDP). Elle permettra de gérer le taux d’envoi et le taux de réception des requêtes.
* Gestion des sockets.

## Architecture du Manager version 3

File d’attente

File d’attente

Transport Layer

*SNMPSocketHandler InputStream*

Q

*SNMP Message HandlerOutputStream*

*SNMP Message HandlerInputStream*

*SNMPSocketHandler OutputStream*

IHM

SNMP Handler

Messages entrants

Messages sortant

TRAP SNMPv2

*SNMPTrapListener*

*Port 162*

Description :

On identifie bien les couches présentées précédemment. Elles sont liées entre elles avec des files d’attentes de type FIFO. Lorsqu’une PDU SNMP arrive, elle est réceptionnée par la couche *SNMPSocketHandler* puis transférer à la couche supérieure via une première file d’attente, pour y être converti en *SNMPMessage.* Une fois la conversion terminée, elle ce dernier est placé dans une seconde file d’attente pour être traité par un Manager.

Le même processus se passe pour les messages sortants.

Le Manager enverra ses messages sur un numéro de port quelconque et écoutera les TRAP SNMP sur le port 162.

### Architecture de l’Agent SNMP

File d’attente

File d’attente

Q

SNMP Message Handler OutputStream

SNMP Message Handler InputStream

SNMPSocketHandler OutputStream

SNMP Message Handler InputStream

Transport Layer

AFFICHAGE CONSOLE

SNMP Agent

Messages entrants

Messages sortants

TRAP SNMPv2

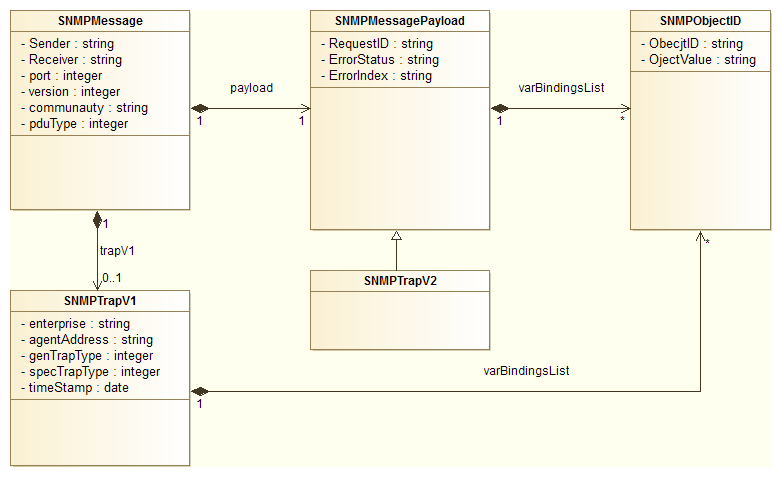
Description :

L’agent utilise suit le même processus de traitement des messages que le manager. Il écoutera les requêtes SNMP sur le port numéro 161, et émettra des TRAP SNMP à destination de son Manager sur le port 162.

# Diagrammes

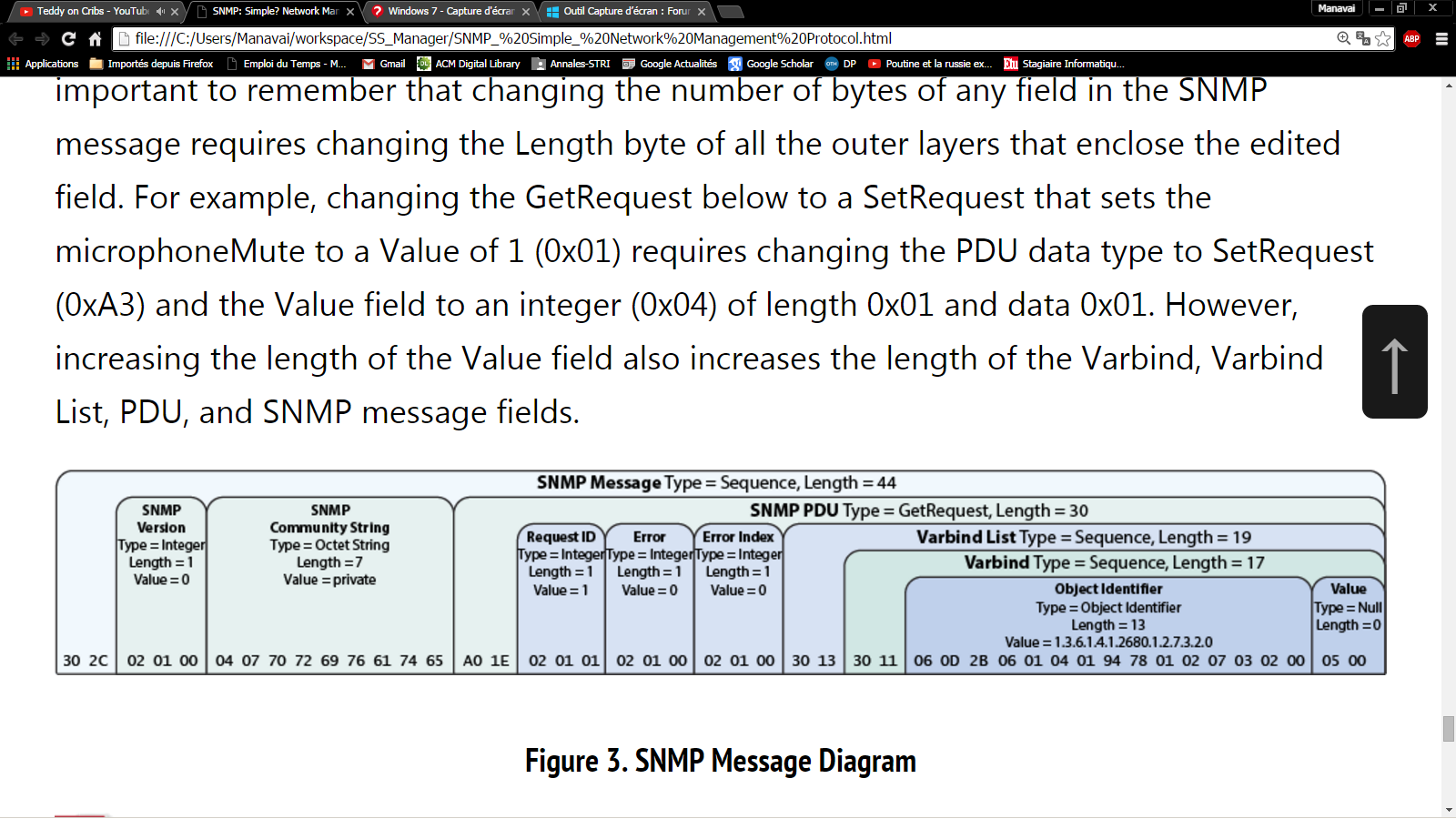
Dans les diagrammes de classe suivant, on ne mettra que les attributs importants. Les méthodes seront développées par chacun selon ses besoins.

## SNMP Messages



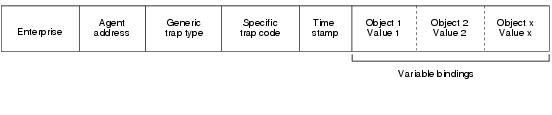
Il s’agit de l’une des parties les plus difficiles du programme : les gestions des messages SNMP. En effet ces derniers sont au format BER (Type length Value). Il s’agit de pouvoir extraire à partir d’un Datagramme, les informations dont nous avons besoins, et de pouvoir les constituer.

Voici un exemple de PDU SNMP :



Les PDU SNMP de la version 1 et version 2c sont identiques. Seul les TRAP SNMP changent.

La TRAP SNMP v1 est de la forme suivante :

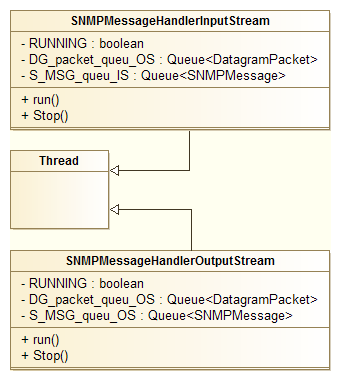


La TRAP SNMP v2 est de la forme d’un SNMP Message normale avec un PDU Type différent et ayant comme Variable *sysUpTime.0* et *snmpTrapOID.0.* (<https://tools.ietf.org/html/rfc3416#page-22>)

Attention : Dans cette version 3, seule des messages SNMPv2 seront envoyés.

**NB :** les types des PDU sont disponibles en annexe. De plus les Traps ne seront pas géré dans cette version. Néanmoins, les classes seront programmées.

## SNMP Message Handler



Les *SNMPMessageHandler* sont des Thread qui ne gèrent que la conversion des messages SNMP et des *DatagramPacket*, dans cette première version. La méthode **run ()** contient le code qui sera exécuté par le Thread, et la méthode **Stop ()** permet de stopper le Thread en mettent l’attribut RUNNING à false permettant ainsi au Thread de quitté la boucle infini et de se terminer.

Voici l’algorithme général d’un Thread :

// méthode run()

DEBUT THREAD

WHILE(RUNNING)

DEBUT

SI LA FILE D’ATTENTE A LIRE N’EST PAS VIDE ALORS

DEBUT SI

ON RECUPERE LE MESSAGE A TRAITER

S’IL LE MESSAGE EST UN TRAP ALORS

ON NOTIFIE L’IHM ;

FIN

ON LE CONVERTI

ON LE TRANSMET A L’AUTRE FILE D’ATTENTE

FIN SI

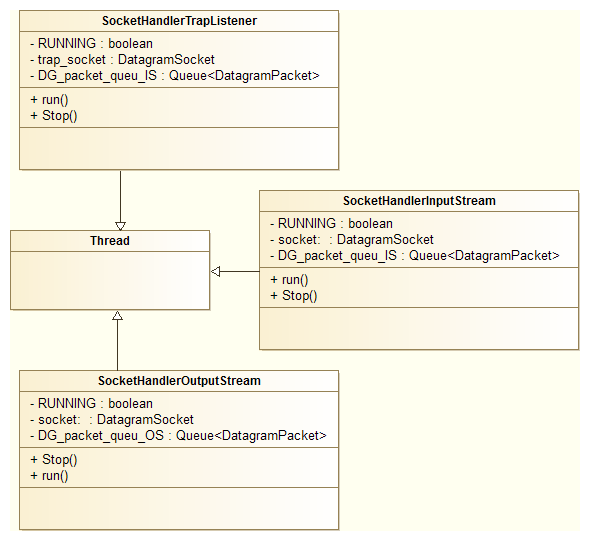
SINON

ON ENDORS LE THREAD PENDANT 100ms

FIN

FIN THREAD

## SNMP Socket Handler

Les *SNMPSocketHandler* sont des Thread qui ne gèrent l’envoi et la réception des DatagramPacket. La méthode **run()** contient le code qui sera exécuté par le Thread, et la méthode **Stop()** permet de stopper le Thread en mettent l’attribut RUNNING à false permettant ainsi au Thread de quitté la boucle infini et de se terminer.

Voici l’algorithme général d’un Thread :

// méthode run() pour SocketHandlerOutputStream

DEBUT THREAD

WHILE(RUNNING)

DEBUT

SI LA FILE D’ATTENTE A LIRE N’EST PAS VIDE ALORS

ON RECUPERE LE DATAGRAMPACKET

ON LE TRANSMET SUR LE SOCKET

SINON

ON ENDORT LE THREAD PENDANT 100ms

FIN

FIN THREAD

// méthode run()

// Pour SocketHandlerInputStream et SocketHandlerTrapListener

DEBUT THREAD

WHILE(RUNNING)

DEBUT

LIRESOCKET () // Fonction Bloquante

ON RECUPERE LE DATAGRAMPACKET

ON LE TRANSMET A LA FILE D’ATTENTE

FIN

FIN THREAD

## SNMP Kernel (version 1)

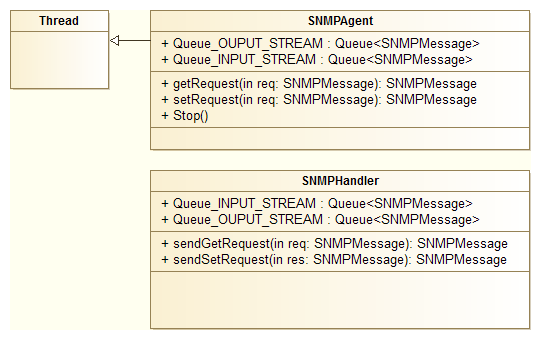
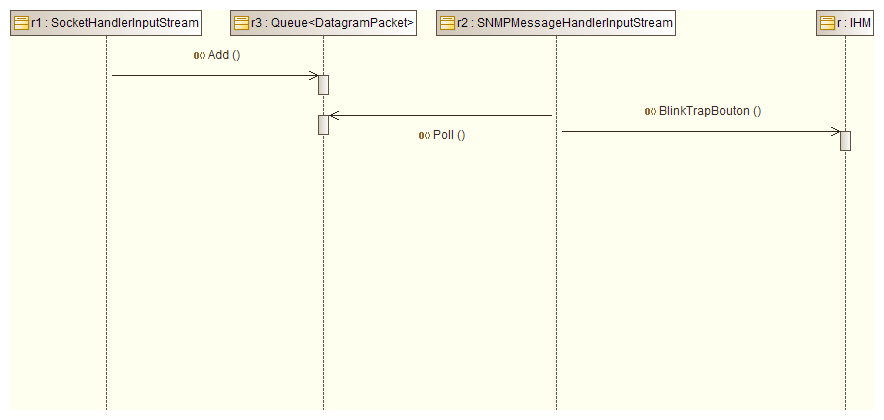
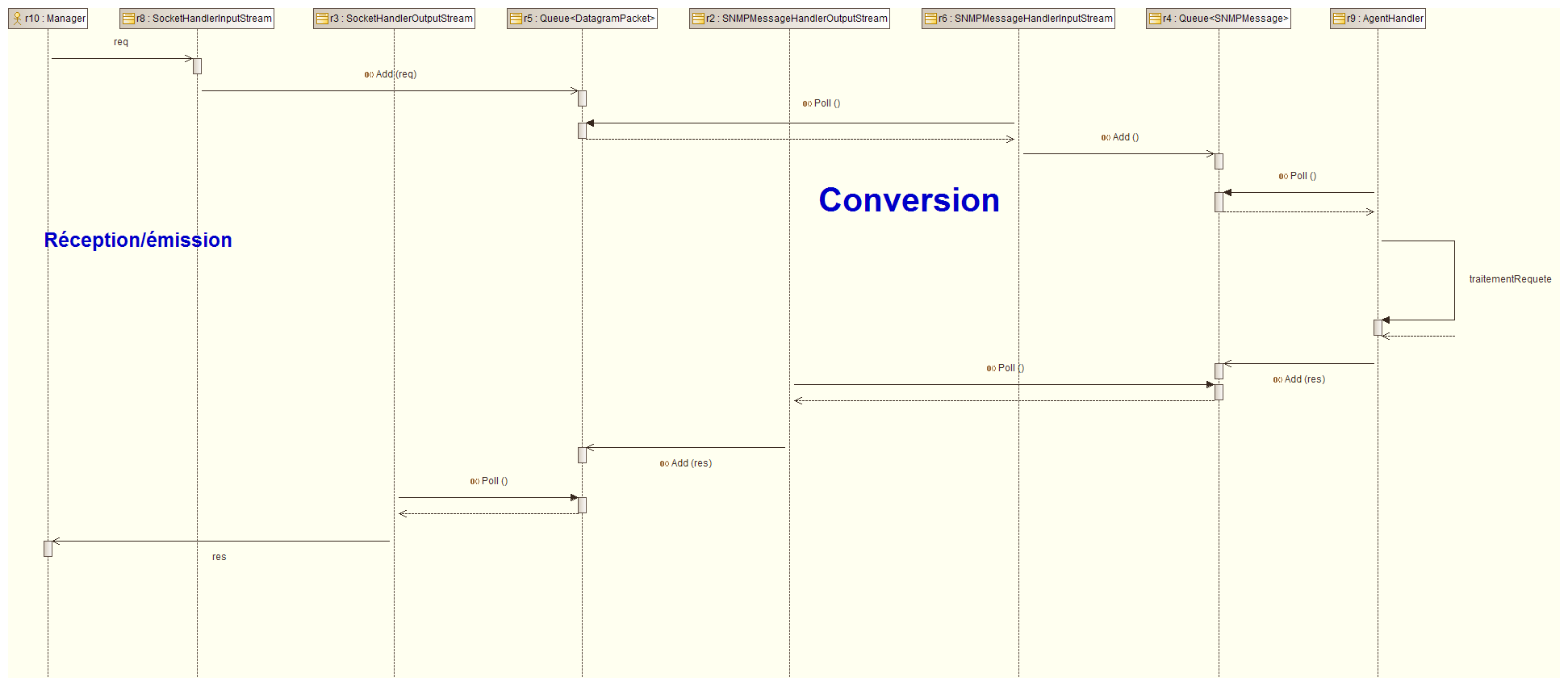


Diagramme de Séquence – Manager : Réception d’un TRAP SNMPv2



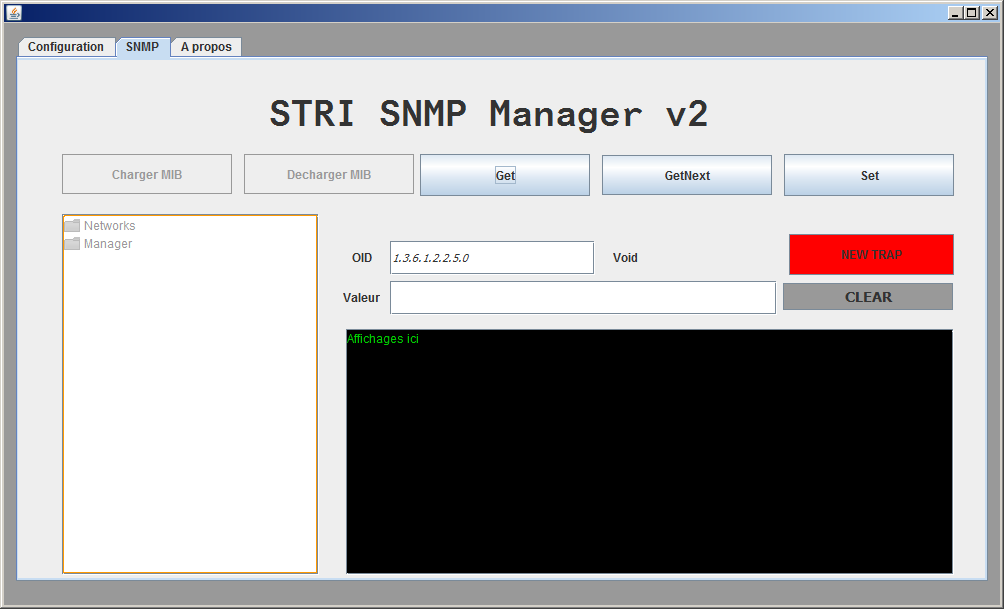
Seul, 3 entités interviennent dans la réception d’une TRAP. le SNMPMessageHandlerInputStream envoie notifie directement l’IHM s’il s’agit d’un TRAP SNMP. sinon il passera le message au SNMPKernel.

Diagramme de Séquence – Agent: Réception d’une requête GET, SET et GET-NEXT, envoi d’un TRAP SNMP.



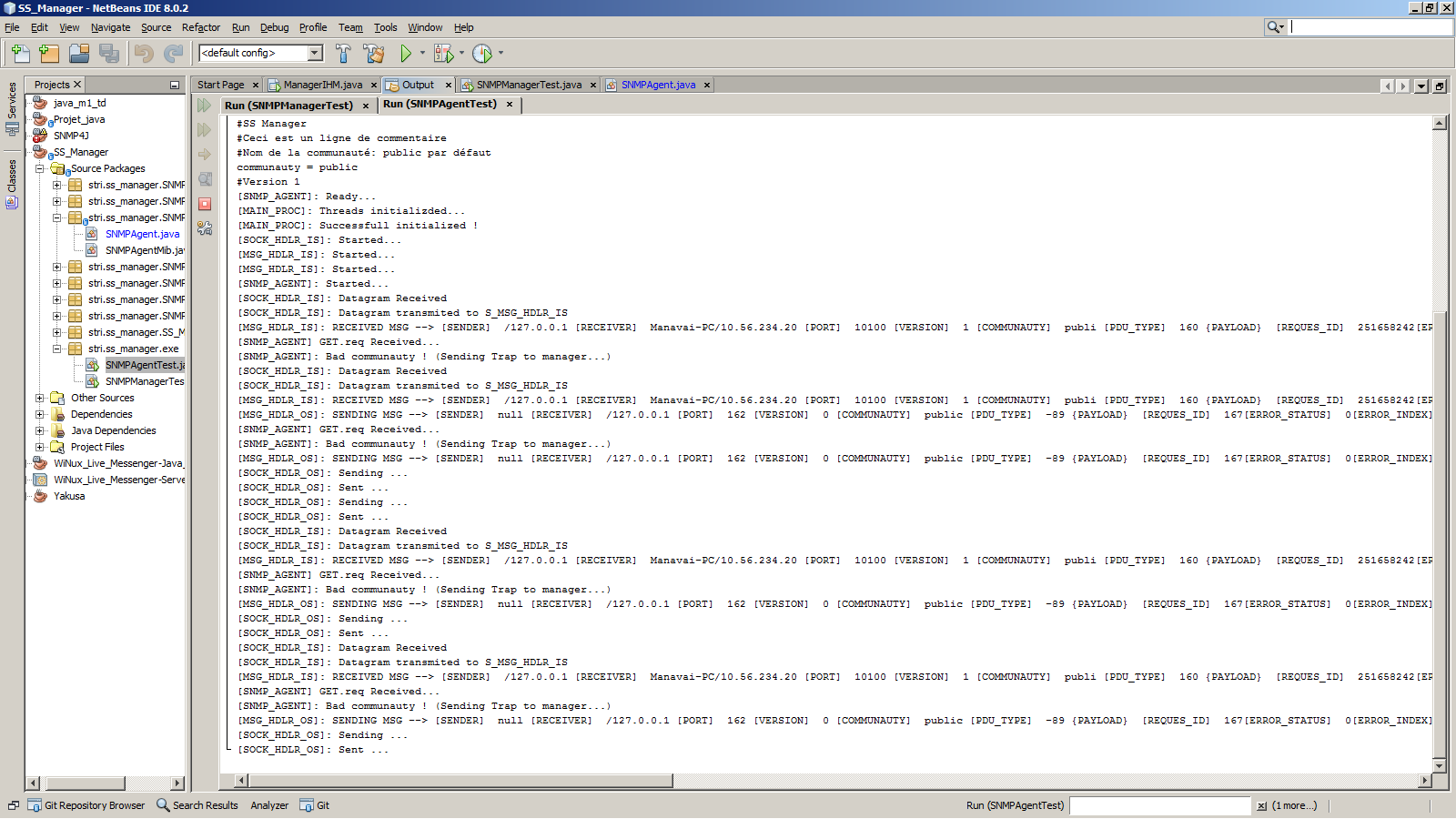
Lorsque l’agent reçoit une requête, il vérifie la communauté dans traitementRequete(), si ce n’est pas la bonne, le message renvoyé sera un trap SNMP.

Description de l’IHM du Manager de la version 3 (implémentons la notification de TRAP):



Lorsque le Manager reçoit un trap SNMPv2, ce dernier prévient l’utilisateur via le bouton « New TRAP » qui se met à clignoter en rouge. Lorsque l’utilisateur clique sur ce bouton, ce dernier affiche le contenu du message.

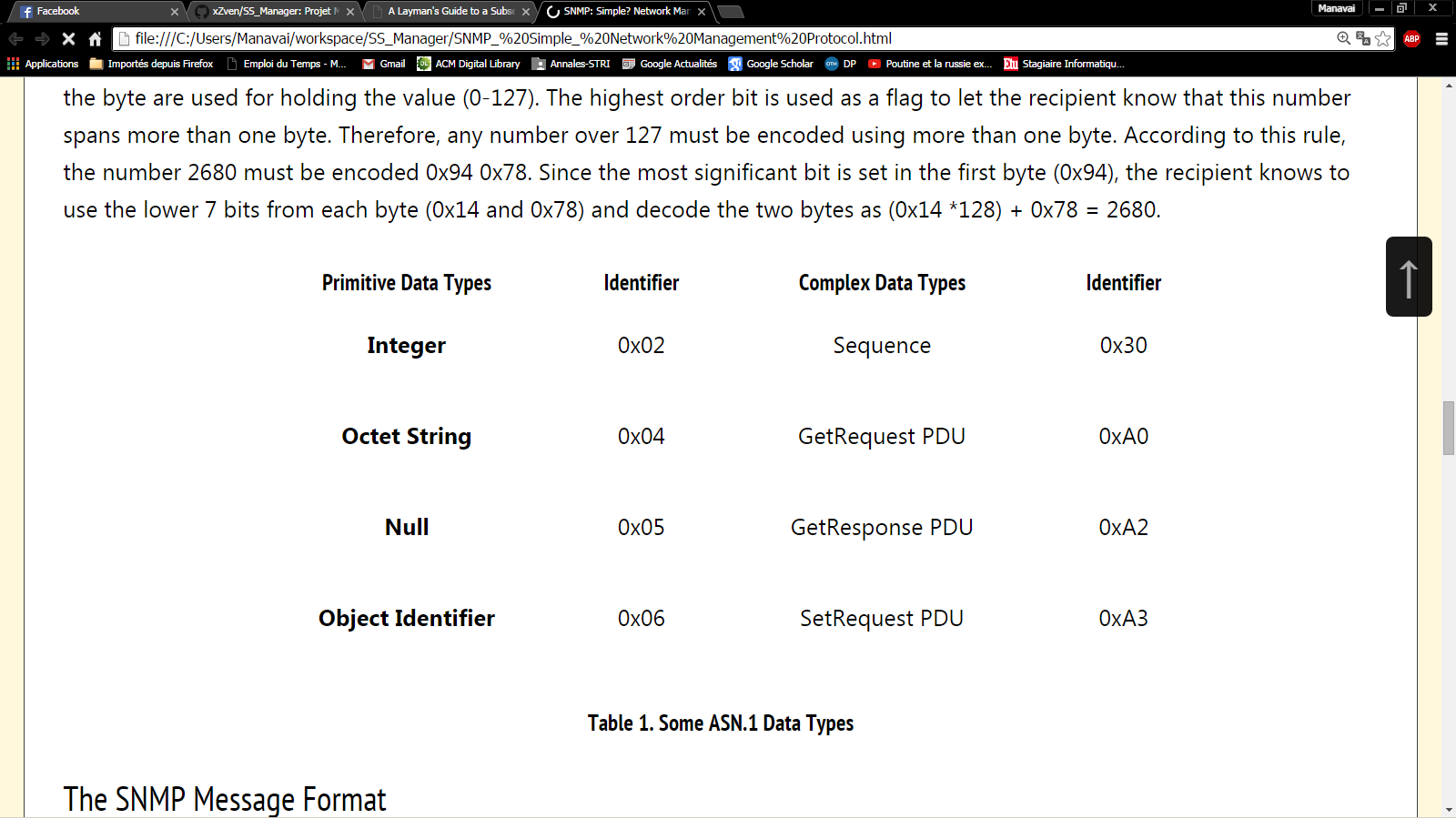
L’agent lui envoi à un seul Manager dont l’adresse IP est entrée en dur dans le code. Dans cette version, seule les erreurs d’authentification sont gérées (mauvaise communauté). Donc l’Agent, enverra à son Manager un trap sur son port 162, comme le montre la capture d’écran ci-dessous :



# Annexe

Liens git hub du projet : <https://github.com/xZven/SS_Manager>

**Type BER :**



Fichier annexe :

STRI SNMP Manager (à ouvrir avec modelio)

Code source dans un Projet Maven.