Création d'un module de communication

C:\Users\Leo\Desktop\logo_newel.png

Léo PERRIN

Création d'un module de communication sécurisé utilisant le protocole HTTPS entre une application android et un serveur J2EE

Contenu

[1. Introduction 2](#_Toc345882978)

[1.1. Objectifs initiaux : 2](#_Toc345882979)

[1.2. Environnement de travail. 2](#_Toc345882980)

[1.3. Méthodologie 2](#_Toc345882981)

[2. Développement de l’application JAVA. 4](#_Toc345882982)

[2.1. Utilisation de la librairie HAPI seule. 4](#_Toc345882983)

[2.1.1. SSL et JAVA. 4](#_Toc345882984)

[2.1.2. Des problèmes au sein de HAPI 4](#_Toc345882985)

[2.1.3. Problèmes persistants 4](#_Toc345882986)

[2.2. Utilisation de l’extension HL7 over HTTP 5](#_Toc345882987)

[2.2.1. Mise en place du keystore 5](#_Toc345882988)

[2.2.2. Mise en place du truststore 5](#_Toc345882989)

[2.3. L’application finale 5](#_Toc345882990)

[2.3.1. Trace de l’exécution coté serveur 6](#_Toc345882991)

[2.3.2. Trace de l’exécution coté client 7](#_Toc345882992)

[3. Portabilité du code sous android 8](#_Toc345882993)

[3.1. Utilisation de la librairie HAPI Lite 9](#_Toc345882994)

[3.2. Utilisation de la libraire HAPI officiel 9](#_Toc345882995)

[4. Conclusion 10](#_Toc345882996)

# Introduction

## Objectifs initiaux :

Développement d’une application android capable de communiquer des données au format HL7 à l’aide de l’API HAPI minimisé par Mr Nicolas Maillard de la société Newel.

Cette application sera la base de travail pour un module de communication. Ce module sera utilisé dans le cadre d’une application médicale.

Les données de type médical sont des données d’ordre sensible et doivent donc être protégées à l’aide d’un protocole sécurisé.

Dans le cadre de l’application la société Newel (en se basant sur les recommendation de continua) a opté pour le protocole HTTPS.

## Environnement de travail.

Pour coller au mieux à l’environnement de travail utilisé chez Newel, j’ai opté pour une base de travail basée sur le système d’exploitation Ubuntu, et l’IDE éclipse.

Ne connaissant pas le serveur utilisé pour le J2EE, j’ai opté pour un serveur tomcat.

## Méthodologie

J’ai d’abord mis une place une application java utilisant la version complète de HAPI. C’est seulement une fois cette version fonctionnelle que je passerais à un environnement android et à la version simplifiée de HAPI.

La librairie HAPI propose nativement un protocole de communication sécurisé par la technologie SSL au même titre que le HTTPS, j’ai donc cherché à utiliser ce qui existait déjà.

Apres des échecs répétés avec HAPI, je me suis tourné (sur conseil de Nicolas Maillard) vers le module HL7 over HTTP.

C’est cette dernière solution qui a donné les meilleurs résultats.

## Architecture des trames HAPI

Les trames HAPI sont definit par la norme HL7. Les trames (aussi appeler messages) sont composées de plusieurs sous partie, d’un type et d’un sous type. En premier lieu les segments, se sont des containers formaliser, parmis les segments remarquables on notera le segment d’entete (MSH) et le champ d’observation (OBX). Chacun de ses segments sont subdiviser en champs. La norme HL7 definit précisément les champs pour chacun des segments ainsi que deux regles :

* Une regle pour definir si un champ est obligatoire
* Une regle pour definir si un champ peut etre présent plusieurs fois dans le segment

Pour les 2 segments cité ci-dessus, la definition de champ est disponible en annexe.

Dans notre cas, nous utilison un message ORU\_R01 ou ORU (Observational Report – Unsolicited) est le type et R01, le sous type.

# Développement de l’application JAVA.

## Utilisation de la librairie HAPI seule.

Mes 1eres recherches m’ont orienté vers l’utilisation de la surcouche SSL native à HAPI.

Dans ce cadre je me suis heurté à plusieurs soucis.

### SSL et JAVA.

La mise en place de SSL sous JAVA n’est pas chose simple, tout d’abord la génération des truststore et keystore

N’étant pas familier aves les technologies de sécurité, j’ai mis beaucoup de temps à trouver les méthodes de génération des certificats nécessaires pour une communication sécurisée sans action de l’utilisateur.

Finalement mes recherches m’ont amené à trouver la solution suivante :

keytool -genkey -alias tomcat -keyalg rsa -validity 3660 -keystore .keystore –keysize 2048 –storepass e-Care

keytool -export -alias tomcat -keystore .keystore -rfc -file public.cert

keytool -import -alias tomcat -file public.cert -keystore .truststore

rm public.cert

N’ayant découvert le processus qu’après le passage à HL7 over HTTP je n’ai pas pu tester la mise en place.

Cependant en plus de la création des store, le code java nécessite quelques aménagements pour utiliser les codes serveurs.

Notamment la déclaration des keystore pour le serveur :

System.setProperty("javax.net.ssl.keyStore", "[path]/.keystore");

System.setProperty("javax.net.ssl.keyStorePassword", "[password]");

System.setProperty("javax.net.ssl.keyStoreType", "pkcs12");

### Des problèmes au sein de HAPI

En plus de mon manque de connaissances, j’ai été confronté à des problèmes liés à la librairie HAPI. Dans le package ConnectionData on peut observer le code suivant :

public [ConnectionData](http://hl7api.sourceforge.net/xref/ca/uhn/hl7v2/app/ConnectionData.html)(String host, int port, [Parser](http://hl7api.sourceforge.net/xref/ca/uhn/hl7v2/parser/Parser.html) parser, LowerLayerProtocol protocol, boolean tls) {

this(host, port, 0, parser, protocol, false);

}

Ce code désactive l’option TLS si l’on spécifie un seul port (le port sortant) au lieu de 2 pour le client. Cette piste reste exploitable mais necessite de lourde configuration pour fonctionner sur l’infrastructure 3G

### Problèmes persistants

Apres ces différents échecs, le seul souci subsistant est le dernier énoncé, je n’ai pas réussi à ce que le client encode ses requêtes, je me suis donc naturellement tourné vers la solution HL7 over HTTP.

## Utilisation de l’extension HL7 over HTTP

A l’origine du projet j’étais réticent à utiliser l’extension HL7 over HTTP, car elle alourdie l’application android et ajoute une source d’erreur ou de bug.

Cependant, étant donné les faiblesses de HAPI seul, je me suis tourné vers ce module pour mener à bien le projet.

### Mise en place du keystore

La mise en place du keystore, coté serveur, est la partie qui m’as posé le plus de difficultés. Sur l’aspect des conaissances.

Ne maitrisant pas l’outil tomcat, j’ai fait de recherches qui m’ont amené à modifier la configuration du serveur comme suit :

<Connector

SSLEnabled = "true"

acceptCount = "100"

clientAuth = "false"

debug = "0"

disableUploadTimeout = "true"

enableLookups = "true"

keystoreFile = "[path]/.keystore"

kaystorePass = "e-Care"

maxSpareThreads = "75"

maxThreads = "25"

port = "8443"

scheme = "https"

secure = "true"

sslProtocol = "TLS"

/>

Le keystore contient une clé (auto signée dans mon cas) qui authentifie le serveur. La configuration ne se fait pas dans le code mais dans la configuration du serveur tomcat (fichier serveur.xml).

Cette configuration permet aux servlet d'accepter les connexions sur le port HTTPS (8443 dans mon cas) et donc d'accéder aux requêtes du client. La configuration ne change pas, quelque soit le client utiliser (application java ou android ou bien même d’autre client communiquant avec HL7 over HTTPS)

### Mise en place du truststore

Le truststore permet de sécuriser la connexion entre client et serveur sans action du client.

Le fichier définit les serveurs de confiance avec lesquels le client peut communiquer.

CustomCertificateTlsSocketFactory clientSocketFactory = new CustomCertificateTlsSocketFactory();

clientSocketFactory.setKeystoreFilename("truststore.jks");

clientSocketFactory.setKeystorePassphrase("trustpassword");

cli.setSocketFactory(clientSocketFactory);

Dans cette configuration le truststore.jsk se trouve dans le dossier src. Sinon il faut adapter le Path.

Une fois ces deux mesures de sécurité mises en place, le client et le serveur discutent sur une connexion cryptée par SSL/TLS la sécurité de la communication dépend de 2 facteurs, La taille du certificat, et l’etat du reseau (si le reseau presente des faille de type Homme du Milieu, le codage SSL/TLS deviens inutile.

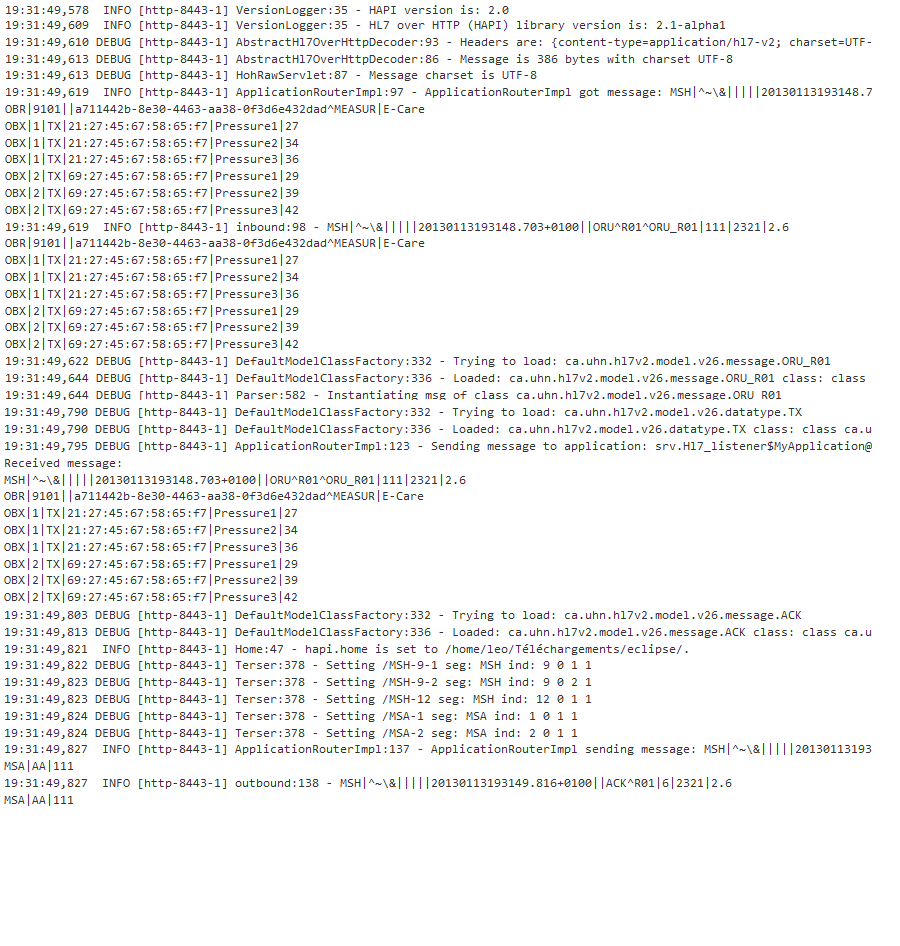
## L’application finale

Une fois tous les éléments mis en place, l’application java fonctionne parfaitement.

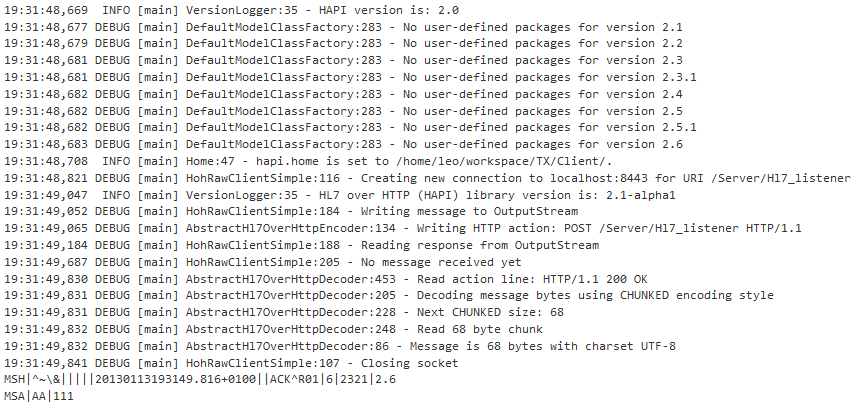
On constate avec wireshark que la communication est effectivement cryptée (non lisible par un tiers) et que le message arrive bien à destination sans deformation.

L’objectif principal est donc atteint.

### **Trace de l’exécution coté serveur**



### Trace de l’exécution coté client

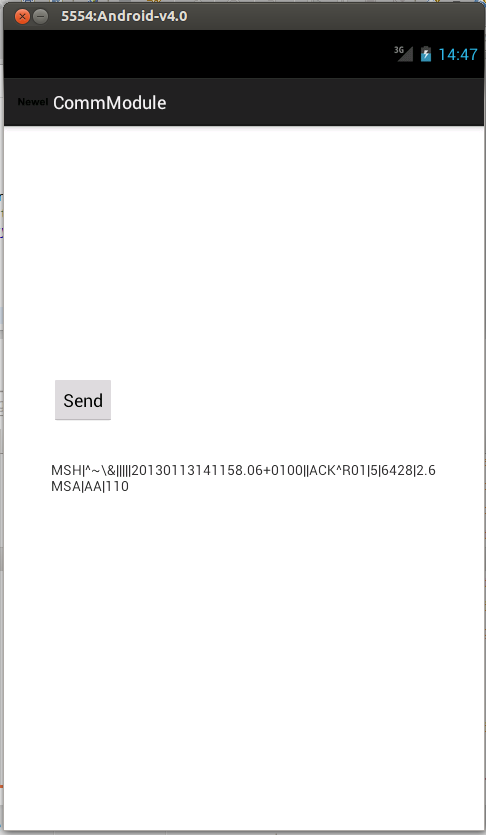


En fin de trace on a l’accusé de réception du message envoyé au serveur.

Les traces ne montrent aucune erreur et le message arrive bien a destination (reception de l’accusé).

# Portabilité du code sous android

Pour l'utilisation de l'application sous android, j'ai confectionné une interface simple permettant d'envoyer un message généré automatiquement et d'afficher l'accusé de réception du message. Pour cette capture j’ai renseigner en statique la reponse afin de simuler un fonctionnement normal.



La portabilité de l’applicaions vers android ne s’est pas faite sans heurts. Pour la portabilité j’ai procéder en 2 étapes.

## Utilisation de la libraire HAPI officiel

J’ai dans un premier temps essayer de travailler avec la librairie officiel de HAPI afin de limite les eventuels problème causé par la librairie minifié. La mise en place de cette libraire ne s’est pas faite sans soucis, le principale problème que j’ai rencontrer concerne la class MessageIDGenerator cette librairie créé un fichier temporaire dans un endroit définit de manière statique.

public final static String DEFAULT\_ID\_FILE = Home.getHomeDirectory().getAbsolutePath() + "/id\_file";

Etant donné que Newel a travaillé sur une simplification de l’API HAPI, j’ai fait la demande pour récupérer et modifier les sources de la version Lite.

## Utilisation de la librairie HAPI Lite

Une fois les sources recu, je me suis mis au travail. Cette librairie presente l’avantage d’etre completement modifiable et bien plus legere que l’API complete. Cependant, il manquais quelques element a cette librairie pour fonctionner avec HOH, j’ai donc du rajouter les class suivantes :

* ca.uhn.hl7v2.parser.DefaultXMLParser
* ca.uhn.hl7v2.parser.XMLParser

Une fois tout cela modifier j’ai lancé l’application, mais un autre problème se pose.

Les applications android ne supportent qu’un encodage BKS hors HAPI est concu pour fonctionner avec un keystore au format JKS, les deux etant bien entendu incompatible, il faut donc modifier l’API pour supporter le format BSK. N’etant pas au courrant des differences entre ces 2 format, j’ai demander de l’aide sur de nombreux sites et forum sans réponse.

# Conclusion

Ce projet est un premier pas vers le module de communication.

Il m’a permis de mettre à jour un certain nombre de contraintes et de limitations liées à l’utilisation de HAPI par exemple.

Je n’ai pas eu le temps de me pencher sur la population des segments, j’ai utilisé les champs qui me semblaient les plus logiques dans le cadre de l’application, cependant une recherche approfondie sur l’utilisation des segments obx est à realiser pour permetre la meilleur compatibilité et portabilité du projet.

Finalement, bien que la portabilité vers android pose des soucis, l’utilisation de HAPI pour le formatage des message remplis parfaitement son rôle. Pour ce qui est de la gestion des message et du support de l’envoit securiser de message, HAPI n’est pas du tout concu pour fonctionner sur un environnement Android. La gestion des fichiers et des keystore pose de réel problème. Ces problèmes depassent mes competence et je n’ai pas su trouver de solution dans le temps imparti.

Il n’est pas invensiageable d’utiliser un autre systeme que HOH pour envoyer les messages. Je ne me suis pas pencher sur les alternatives a ce systeme cependant je pense que des application déjà codé sont disponibles sur internet.

Au final, ce projet m’as permis de poursuivre mon stage dans le dommaine de la securitée mais d’un point de vu plus appliquer. J’ai pu mettre en place un systeme de communication securisé. J’ai aussi pu me familiariser avec l’environnement android. Meme si je n’ai pas pousser de ce coté là j’ai pu effleurer les fondements des applications et je pense continuer ma formation dans cette branche.

Annexes

Annexe 1 : Structure du segment OBX

| **SEQ** | **LEN** | **DT** | **OPT** | **RP/#** | **ELEMENT NAME** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | SI | O |  | Set [*ID*](http://www.corepointhealth.com/resource-center/healthcare-interoperability-glossary#ID) – Obx |
| 2 | 2 | ID | R |  | Value Type |
| 3 | 590 | CE | R |  | Observation Identifier |
| 4 | 20 | ST | O |  | Observation Sub-Id |
| 5 | 65536 | ST | O |  | Observation Value |
| 6 | 60 | CE | O |  | Units |
| 7 | 10 | ST | O |  | Reference Range |
| 8 | 5 | ID | O | Y/5 | Abnormal Flags |
| 9 | 5 | NM | O |  | Probability |
| 10 | 2 | ID | O |  | Nature of Abnormal Test |
| 11 | 1 | ID | R |  | Observ Result Status |
| 12 | 26 | TS | O |  | Data Last Obs Normal Values |
| 13 | 20 | ST | O |  | User Defined Access Checks |
| 14 | 26 | TS | O |  | Date/Time of the Observation |
| 15 | 60 | CE | O |  | Producer’s Id |
| 16 | 80 | XCN | O |  | Responsible Observer |
| 17 | 80 | CE | O | Y | Observation Method |

Legende :

* SEQ : Numero de sequence, ordre des champ dans la séquence
* LEN : Longeur maximum du contenu en octet
* DT : Type de données

|  |  |  |  |
| --- | --- | --- | --- |
| ST | Chaîne de caractère | XPN | Nom de la personne |
| TX | Texte | XAD | Adresse |
| FT | Texte formaté | XTN | Telephone |
| NM | Numerique | ID | Valeure codée |
| D | Date | IS | Valeure codée |
| T | Heure | CM | Composite |
| TS | Horodatage | CN | Identifiant et nom |
| PL | Position du patient | CQ | Quantité et unitée |

Les champs composites sont définis comme la combinaison de plusieurs champs de données. Chaque partie d’un champ composite est appelé un « composant »

* OPT : Requis ou non (O = optionel, R = Requis)
* RP/# : Répététivitée (/# = nombre maximum d’occurences)
* Element Name : Nom du champ

Annexe 2 : Structure du segment MSH

| **SEQ** | **LEN** | **DT** | **OPT** | **RP/#** | **ELEMENT NAME** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | ST | R |  | Field Separator |
| 2 | 4 | ST | R |  | Encoding Characters |
| 3 | 180 | HD | O |  | Sending Application |
| 4 | 180 | HD | O |  | Sending Facility |
| 5 | 180 | HD | O |  | Receiving Application |
| 6 | 180 | HD | O |  | Receiving Facility |
| 7 | 26 | TS | O |  | Date/Time of Message |
| 8 | 40 | ST | O |  | Security |
| 9 | 7 | CM\_MSG | R |  | Message Type |
| 10 | 20 | ST | R |  | Message Control Id |
| 11 | 3 | PT | R |  | Processing Id |
| 12 | 8 | ID | R |  | Version Id |
| 13 | 15 | NM | O |  | Sequence Number |
| 14 | 180 | ST | O |  | Continuation Pointer |
| 15 | 2 | ID | O |  | Accept Acknowledgement Type |
| 16 | 2 | ID | O |  | Application Acknowledgement Type |
| 17 | 2 | ID | O |  | Country Code |
| 18 | 6 | ID | O |  | Character Set |
| 19 | 3 | CE | O |  | Principal Language of Message |