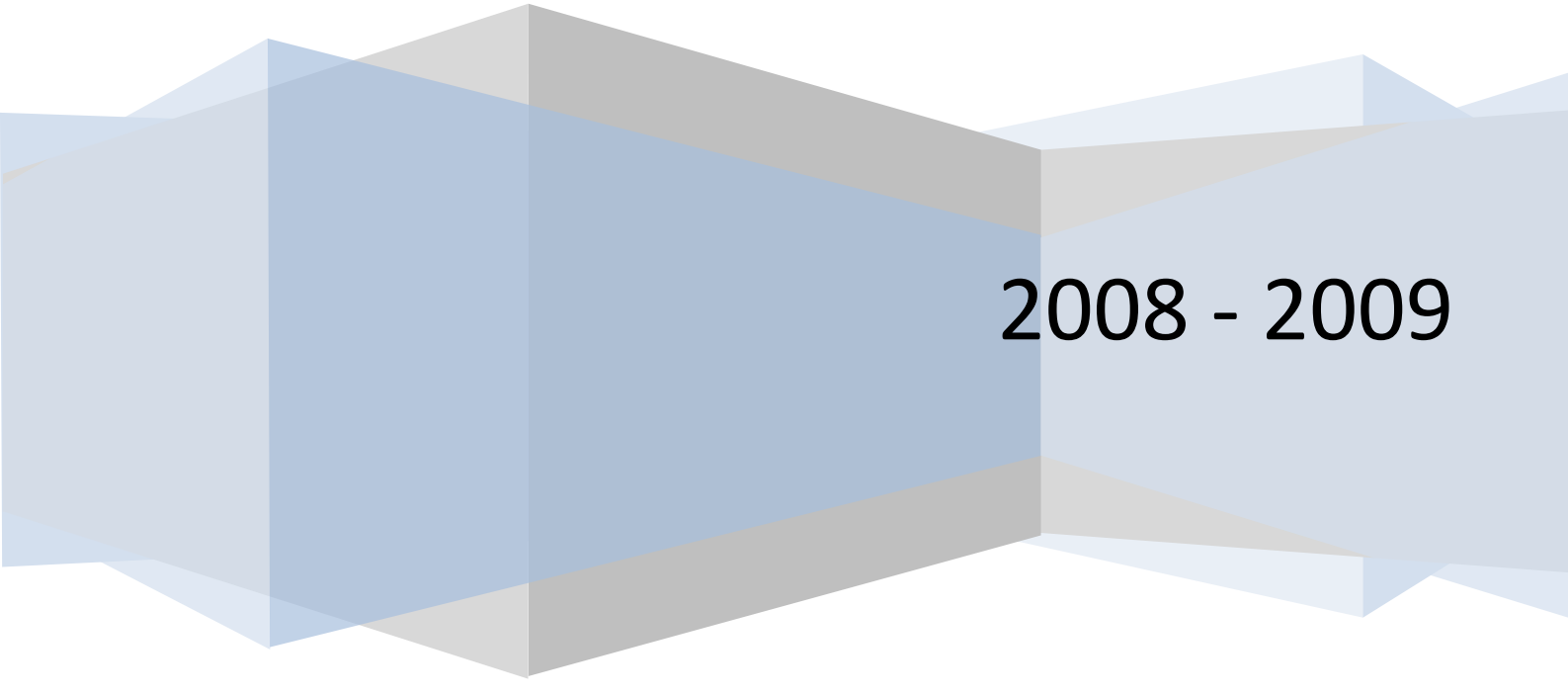
Télécom SudParis – Projet de fin d'étude

Construction d'un proxy XMPP pour un Nabaztag

Clément BEAUSSET et Pierre SOUMOY

Encadré par : M. Sébastien LERICHE



2008 - 2009

SOMMAIRE

1) Introduction

Le Nabaztag est un dispositif communiquant, ressemblant à un lapin, créé par Olivier Mével et Rafi Haladjian, il est produit par la société française Violet. Il mesure 23 centimètres de haut, et se connecte à internet via une borne wifi standard.

Il communique avec son utilisateur via des messages sonores, lumineux ainsi qu'avec ses oreilles. Il peut également diffuser des nombreuses informations comme : la météo, la bourse, la qualité de l'air, le trafic routier du périphérique de Paris, l'arrivée d'emails, …

Toutes les interactions du Nabaztag s'effectuent via internet avec les serveurs de la société Violet. Ce type de fonctionnement restreint considérablement les fonctions du Nabaztag, mais surtout nous sommes dépendant des serveurs de Violet et donc de leurs latences, leurs encombrements, leurs mises à jour… ce qui peut influencer de façon non prévisible sur le bon fonctionnement du Nabaztag, et qui le rend pas très réactif.

Dans ce projet nous avons donc cherché à nous passer des communications avec les serveurs de Violet, pour cela nous avons écouté les communications entre le Nabaztag et le serveur distant, pour ensuite pouvoir créer un serveur que nous pourrons exécuter en local. Grâce à ce proxy, le temps de réaction du lapin est considérablement amélioré, et les fonctionnalités du Nabaztag peuvent donc mieux exploitées.

Sébastien Leriche nous a demandé les fonctions suivantes à inclure sur notre serveur :

- le mouvement des oreilles
- la position des oreilles
- le Text-To-Speech
- la lecture des tags RFID

Nous devions aussi réaliser une API permettant de contrôler facilement le lapin.

2) Gestion de projet

a) Suivi de projet

Tout au long du projet, nous avons eu des réunions de suivi avec M. Leriche, durant lesquelles nous avons pu discuter de l'avancement du projet, du planning pour les semaines à venir, mais également nous avons pu évoquer les différents problèmes techniques que nous avons rencontrés au cours de l'évolution du projet.

b) Plan de charges prévisionnel

	Lancement du projet et tâches annexes	Analyse	Programmation		Rédaction du rapport	Préparation de la soutenance
			développement	tests		
Clément	10%	20%	42%	14%	8%	6%
Pierre	15%	15%	32%	18%	12%	8%

c) Planning prévisionnel

	24/10 au 01/12	01/12 au 15/12	15/12 au 19/01	15/12 au 27/01
Tâche 1	Lancement du projet	Création de l'architecture	Tests	
Tâche 2	Décryptage de la communication		Codage	Rédaction du rapport et préparation de la soutenance

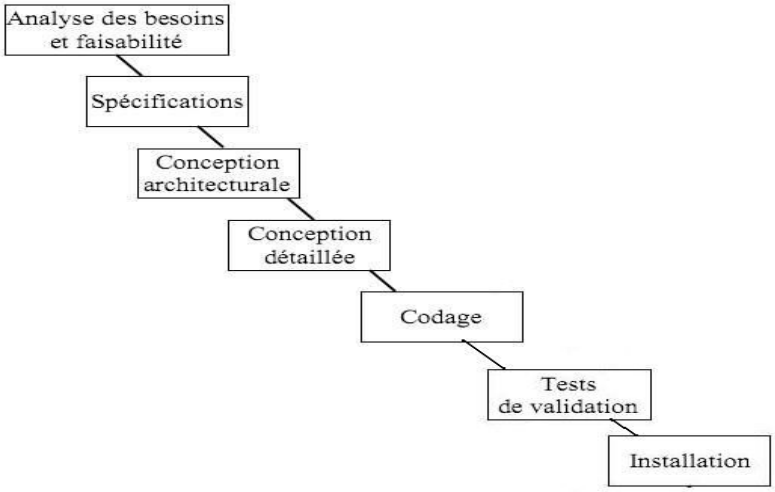
Le planning provisionnel a été bien respecté. Malgré le lancement du projet, qui a été assez long, la suite du projet a été plus rapide, ce qui nous a permis de finir le projet largement dans les temps.

d) Choix du cycle de vie

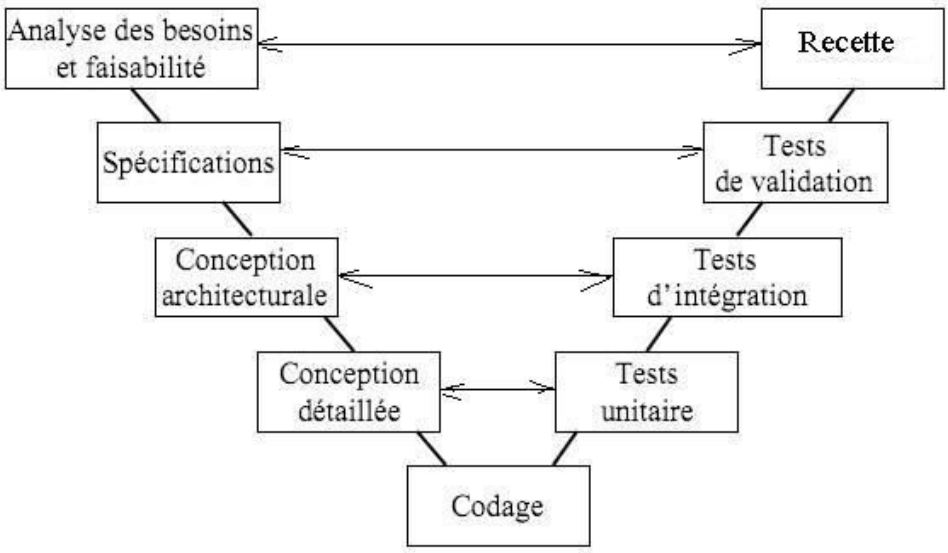
Pour choisir le cycle de vide de notre projet nous avons eu le choix entre plusieurs cycles de développement : mode en cascade, cycle en V, cycle en spirale, extrême programming.

Voici une rapide description des différents cycles :

- Le mode en cascade repose sur la nécessite de terminer une étape pour commencer la suivante. Pour utiliser ce type de cycle de vie il faut avoir une idée très précise du produit final que l'on veut produire.

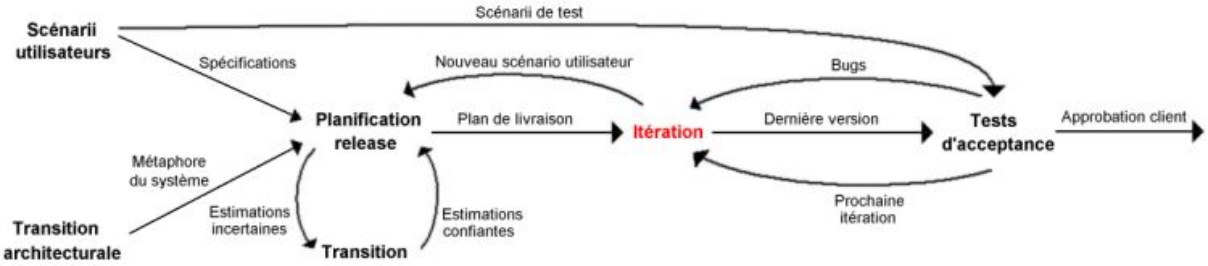


- Le cycle en V a été imaginé pour pallier au problème de réactivité du cycle précédent. Il permet d'anticiper les étapes suivantes, et on prévoit à l'avance ce que les tests doivent donner.



- Le cycle en spirale implémente plusieurs cycles en V successifs. Plus les cycles passent, plus le produit attendu sera complet et dur à réaliser. Le début de chaque itération est précédé d'une étude des risques à venir

- L'extrême programming, quant à lui, repose sur des cycles de développement très rapide dont les étapes sont les suivantes :
    - une phase d'exploration détermine les scénarios clients qui seront fournis pendant cette itération
    - l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels
    - chaque développeur s'attribue des tâches et les réalise avec un binôme
    - lorsque tous les tests fonctionnels passent, le produit est livré



Le cycle en cascade est trop rigide pour ce projet, comme l'est le cycle en V, car nous allons avancer un peu les yeux fermés sur ce projet : nous utilisons des technologies que nous ne connaissons pas du tout (WMPP) ou des technologies que nous ne maîtrisons pas entièrement (Service Web). On peut donc difficilement avoir un cycle de vie aussi planifié.

L'extrême programming semble le meilleur des deux choix restants, car nous aurons besoin de tester à tout moment l'avancement de notre prototype et c'est l'extrême programming qui nous donne le plus de liberté à ce niveau.

Nous allons donc choisir le cycle de vie extrême programming pour notre projet, car il est sans conteste le plus adapté au projet de recherche comme celui que l'on actuellement en train de réaliser.

3) Etude préliminaire

a) écoute des messages

La première difficulté à laquelle nous avons été confronté fut bien l'écoute des messages. Sébastien nous avait fait une démonstration où il arrivait, sans manipulation particulière, à capturer les messages sur le Wifi. En essayant chez nous, aucun message ne transitait. Suite à quelque recherche, il s'est avéré que l'écoute du trafic Wifi n'est possible qu'avec certaines cartes. Elles doivent pouvoir passer en mode *Monitor* : ce sont les cartes typiquement compatibles avec Aircrack. N'en possédant pas, le département informatique de Telecom SudParis nous en a gracieusement prêtée une. Nous avons pourtant rencontré un nouveau problème : la carte n'était pas stable et elle se déconnectait du réseau au bout de quelques secondes, voire minutes dans le meilleur des cas (sans compter qu'elle gelait l'écran de Windows). L'astuce fut d'utiliser la carte juste pour écouter les messages et non pas se voir attribuer une adresse IP. Ceci revenait à taper une commande du genre : *iwconfig eth2 nickname U3NION essid U3NION key 395b732f30ad5z7d285f544b48*

Après avoir résolu ces casse-têtes, l'analyse des messages allait commencer.

b) analyse des messages

i) Initialisation

Maintenant, analysons les paquets à partir du tout premier :

- **Le premier est une requête HTTP :**

```
GET
/vl/bc.jsp?v=0.0.0.10&m=00:19:db:9e:d0:17&l=00:00:00:00:00:00&p=00:00:00:
00:00:00&h=4 HTTP/1.0

User-Agent: MTL

Pragma: no-cache

Host: r.nabaztag.com
```

Nous récupérons ce fichier bc.jsp. Nous remarquons que ça a tout l'air d'être une classe Java. Donc il n'y a pas vraiment d'espoir pour nous de pouvoir modifier ceci. Il s'agit en fait du programme chargé sur le Nabaztag, une sorte de mini-OS. Pour faire tourner un serveur en local, nous gardons donc ce fichier quelque part dans un coin.

Dans les paramètres de l'URL, il est facile de deviner que le *m* est égal à l'adresse mac du Nabaztag qui est aussi son serial.

- **La deuxième requête est aussi une requête HTTP :**

```
GET /vl/locate.jsp?sn=0019db9ed017&h=4&v=18005 HTTP/1.0

User-Agent: MTL

Pragma: no-cache

Icy-MetaData:1

Host: r.nabaztag.com
```

Après édition du fichier locate.jsp, nous avons ce que nous voulions avoir : les paramètres de configuration des différents serveurs :

```
ping tagtag.nabaztag.objects.violet.net

broad broad.violet.net

xmpp_domain xmpp.nabaztag.com
```

Nous ne savons pas encore à quoi correspond chacun des serveurs. Nous récupérons donc leur adresse IP et continuons notre investigation.

Il s'en suit une phrase de connexion du Nabaztag au serveur xmpp.nabaztag.com sur le port 5222. Voici un serveur identifié : il s'agit du serveur jabber avec lequel le Nabaztag communiquera le plus souvent. La phrase d'initialisation est la suivante (en fond bleu le Nabaztag, en fond rouge, le serveur) :

```
<?xml version='1.0' encoding='UTF-8'?>

<stream:stream to='xmpp.nabaztag.com' xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
```

```
<?xml version='1.0'?><stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='1549628791'
from='xmpp.nabaztag.com' version='1.0' xml:lang='en'>
```

Aucun problème. Le serveur répond en clair au lapin.

```
<stream:features>

        <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

                <mechanism>

                 DIGEST-MD5

                </mechanism>

         <mechanism>

                 PLAIN

         </mechanism>

         </mechanisms>

<register xmlns='http://violet.net/features/violet-register'/>

</stream:features>
```

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='DIGEST-MD5'/>
```

Ah, ici, DIGEST-MD5 semble de mauvais augure. Poursuivons.

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

bm9uY2U9IjE4MDk0ODIwIixxb3A9ImF1dGgiLGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNz

</challenge>
```

Voilà, les ennuis commencent. Les plus aguerris auront reconnu le codage utilisé dans certains messages de type MIME. Il s'agit en effet du codage en base 64. Continuons jusqu'à ce qu'il n'y ait plus de cryptage.

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

dXNlcm5hbWU9IjAwMTlkYjllZDAxNyIsbm9uY2U9IjE4MDk0ODIwIixjbm9uY2U9Ij
Q4NDA1NjAwNTk0NzQAIixuYz0wMDAwMDAwMSxxb3A9YXV0aCxkaWdlc3QtdX
JpPSJ4bXBwL3htcHAubmFiYXp0YWcuY29tIixyZXNwb25zZT01NjZhY2IyNjI2OTZkN
2NjZGI3NTIzZjBlNWU4NTEwYixjaGFyc2V0PXV0Zi04

</response>
```

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>

cnNwYXV0aD05NWFhMGFiZDc0Nzc0Y2Q1NTFjMTQ2MDIwZjdkMmNlOA==

</challenge>
```

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Success ! Le nabaztag vient de s'identifier auprès du serveur. Retournons sur les messages codés en base 64.

```
nonce="18094820",qop="auth",charset=utf-8,algorithm=md5-sess
```

```
username="0019db9ed017",

nonce="18094820",cnonce="4840560059474",

nc=00000001,qop=auth,digest-uri="xmpp/xmpp.nabaztag.com",

response=566acb262696d7ccdb7523f0e5e8510b,

charset=utf-8
```

```
rspauth=95aa0abd74774cd551c146020f7d2ce8
```

Il n'est pas difficile de trouver avec ces informations que nous avons affaire à une authentification basée sur la méthode DIGEST (http://fr.wikipedia.org/wiki/HTTP_Authentification). Après avoir lu la page de wikipédia, il nous manque deux paramètres cruciaux pour calculer le *rspauth* : le *realm* et le *password*. Le *realm* est-il produit en fonction de l'adresse du serveur ? A-t-il une valeur fixe ? Il nous semble évident qu'il est stocké coté lapin et serveur, nous l'apercevons d'ailleurs dans le fichier bc.jsp mais nous ne parvenons pas à le trouver. Les mêmes questions se posent pour le paramètre *password*. Le password vaut-il le token ou correspond-il au mot de passe du site web ? Nous n'en savons rien.

Nous consultons alors les autres projets qui ont déjà été faits. Rien. Ils sont tous basés soit sur le protocole de la version 1 (exclusivement sur le HTTP et non sur jabber) et ils n'ont pas eu affaire à une séquence d'identification. Le seul projet version 2 fut un proxy qui relaie les informations du Nabaztag au serveur Violet mais nous, nous voulons pouvoir faire tourner le lapin en réseau local, donc sans internet. Nous sommes donc bloqués. Il n'y a plus de projet de serveur local depuis que la version 2 est sortie. Peut-être est-ce du à cause de cette identification ? Doit-on abandonner notre projet de serveur local ?

Que nenni ! Réfléchissons un peu. Les premières versions basées sur le protocole HTTP souffraient d'une extrême latence : Violet n'avait pas prévu le succès de leur produit et les serveurs ne suivaient pas. Les services fournis par Violet sont très pauvres. Le Nabaztag en lui même est en effet inutile. Dès que le lapin envoie le premier message *<challenge>*, nous lui envoyons le message *<success>*, et le nabaztag croit qu'il s'est identifié !

Une fois ceci fait, le nabaztag doit se mettre en *idle* (i.e. en attente de message). La séquence suivante correspond à cette action.

```
<?xml version='1.0' encoding='UTF-8'?>

<stream:stream to='xmpp.nabaztag.com' xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' version='1.0'>
```

```
<?xml version='1.0'?>

<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='1208618374'
from='xmpp.nabaztag.com' version='1.0' xml:lang='en'>
```

```
<stream:features>

        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>

                <required/>

        </bind>

        <unbind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>

        <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>

</stream:features>
```

```
<iq from="0019db9ed017@xmpp.nabaztag.com/" to="xmpp.nabaztag.com"
type='set' id='1'>

        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>

                <resource>

                        Boot

                </resource>

        </bind>

</iq>
```

```
<iq id='1' type='result'>
        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
                <jid>
                        0019db9ed017@xmpp.nabaztag.com/boot
                </jid>
        </bind>
</iq>
```

```
<iq from='0019db9ed017@xmpp.nabaztag.com/boot' to='xmpp.nabaztag.com'
type='set' id='2'>
        <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

```
<iq type='result' to='0019db9ed017@xmpp.nabaztag.com/boot'
from='xmpp.nabaztag.com' id='2'>
        <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

```
<iq from='0019db9ed017@xmpp.nabaztag.com/boot'
to='net.violet.platform@xmpp.nabaztag.com/sources' type='get' id='3'>
        <query xmlns="violet:iq:sources">
                <packet xmlns="violet:packet" format="1.0"/>
        </query>
</iq>
```

```
<iq from='net.violet.platform@xmpp.nabaztag.com/sources'
to='0019db9ed017@xmpp.nabaztag.com/boot' id='3' type='result'>
        <query xmlns='violet:iq:sources'>
```

```
                <packet xmlns='violet:packet' format='1.0' ttl='604800'>
                        fwQAAAx////+BAAFAA7/CAALAAABAP8=
                </packet>
        </query>
</iq>
```

```
<iq from="0019db9ed017@xmpp.nabaztag.com/boot"
to="xmpp.nabaztag.com" type='set' id='4'>
        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
                <resource>
                        Idle
                </resource>
        </bind>
</iq>
```

```
<iq id='4' type='result'>
        <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
                <jid>
                        0019db9ed017@xmpp.nabaztag.com/idle
                </jid>
        </bind>
</iq>
```

```
<iq from='0019db9ed017@xmpp.nabaztag.com/idle' to='xmpp.nabaztag.com'
type='set' id='5'>
        <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

```
<iq type='result' to='0019db9ed017@xmpp.nabaztag.com/idle'
from='xmpp.nabaztag.com' id='5'>
        <session xmlns='urn:ietf:params:xml:ns:xmpp-session'/>
</iq>
```

```
<presence from='0019db9ed017@xmpp.nabaztag.com/idle' id='6'>
</presence>
```

```
<presence from='0019db9ed017@xmpp.nabaztag.com/idle'
to='0019db9ed017@xmpp.nabaztag.com/idle' id='6'/>
```

```
<presence from='0019db9ed017@xmpp.nabaztag.com/idle'
to='0019db9ed017@xmpp.nabaztag.com/idle' id='6'/>
```

```
<iq from='0019db9ed017@xmpp.nabaztag.com/boot' to='xmpp.nabaztag.com'
type='set' id='7'>
        <unbind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
                <resource>
                        boot
                </resource>
        </unbind>
</iq>
```

```
<iq id='7' type='result'/>
```

Rien de spécial. Les messages peuvent être facilement modifiés. Il y a juste un message contenant du codage en base 64 mais qui se révélera être un message contrôlant la vitesse de clignotement de la LED violette sous le nabaztag.

ii) Paquets jabber

Il est temps de s'attaquer aux messages transitant.

Remarquons que le Nabaztag envoie des messages ayant pour seul octet 0x20 (un caractère espace) toutes les 9 à 10 secondes sur le port 5222. Il teste ainsi si le serveur XMPP est toujours joignable.

Nous nous sommes d'abord intéressés à ce qui nous semblait le plus facile : les messages contrôlant le mouvement des oreilles. Ils sont de ce type :

```
<message from='beveren-tomcat8009@xmpp.platform.violet.net/beveren-tomcat8009:10.100.1.194' to='0019db9ed017@xmpp.nabaztag.com/sources' id='aTePH-7055'>
    <amp xmlns='http://jabber.org/protocol/amp' from='beveren-tomcat8009@xmpp.platform.violet.net/beveren-tomcat8009' to='0019db9ed017@xmpp.nabaztag.com/sources'>
        <rule action='drop' condition='expire-at' value='2009-01-10T13:52:03Z'/>
        <rule action='drop' condition='deliver' value='stored'/>
    </amp>
    <x xmlns='jabber:x:expire' seconds='300'/>
    <packet xmlns='violet:packet' format='1.0' ttl='300'>
        fwQAAAh////+BAYFAf8=
    </packet>
</message>
```

De l'XML avec un bout de base 64. Ensuite, nous voulions lire un son.

```
<message from='beveren-tomcat8009@xmpp.platform.violet.net/beveren-tomcat8009:10.100.1.194' to='0019db9ed017@xmpp.nabaztag.com/idle' id='aTePH-18465'>
```

```
<amp xmlns='http://jabber.org/protocol/amp' from='beveren-
tomcat8009@xmpp.platform.violet.net/beveren-tomcat8009'
to='0019db9ed017@xmpp.nabaztag.com/idle'>

        <rule action='drop' condition='expire-at' value='2009-01-
                11T16:22:22Z'/>

        <rule action='store' condition='match-resource' value='other'/>

</amp>

<x xmlns='jabber:x:expire' seconds='600'/>

<packet xmlns='violet:packet' format='1.0' ttl='600'>

fwoAACUAxLtPl2ODH6nCYBgoKOHCkKCoA5z5zWmyO0womy2Af8K
ViCRF/w==

</packet>

</message>
```

De même. Une partie en clair, une partie cryptée. Les formes des paquets XML sont légèrement différentes. Ceci étant dit, nous n'avions pas perdu notre temps en regardant les anciens projets et nous savions à quoi nous attendre concernant la partie en base 64. Il s'agit à peu de chose près aux paquets de la version 1. Nous distinguerons les paquets des blocs. Un bloc est la partie décodée en base 64 d'un paquet. Nous avons identifiés deux types de blocs (en fait, il y en a quatre mais les autres ne sont pas très intéressants) : les AmbientBlock et les MessageBlock.

Pour tous les blocs, il y a un format qui ne change pas.

7F	04	00	00	04	XX	XX	XX	XX	FF

Les cases en blanc sont la tête et la queue du paquet, ils ne changent jamais.

En rouge, il s'agit du type de bloc :

- 0x04 : Ambientblock
- 0x0A : Messageblock

En jaune, il s'agit de la taille des données sur 3 octets.

Les cases grises sont justement les données.

(1) Les Ambientblock

Il s'agit des messages qui permettent de faire bouger les oreilles du lapin et de faire autres choses (clignotements des LED en mode idle...). Nous n'avons identifié clairement que les messages correspondant au contrôle des oreilles (les autres n'étant que peu intéressants).

Pour les oreilles, le paquet se construit de cette façon :

7F	04	00	00	08	7F	FF	FF	FE	04	LL	05	RR	FF

LL : position de l'oreille gauche entre 0 et 16

RR : position de l'oreille droite entre 0 et 16

(2) Les Messageblock

Ils contiennent des commandes chacune séparées par un caractère de retour à la ligne ('\n').

Les commandes connues sont:

ID : l'ID du message : ne sert absolument à rien.

CL change la couleur des LEDs. Le format est de ce type : *CL 0xRRVVBBh* où RR, VV et BB sont les couleurs de rouge, vert et bleu.

PL choix de la palette. Le format est : *PL X* où X est un numéro entre 0 et 7. Ceci est à associé avec les chorégraphies.

CH joue une chorégraphie. Le format est *CH url*.

MU joue un son. Le format est *MU url*.

ST joue un son en streaming. Format : *ST url*. Non implémenté sur le serveur.

MW attend que les précédentes commandes soient exécutées**.**

Ce message est ensuite crypté selon un algorithme récursif :

```
C[i] = (B[i] - 0x2F) * (1 + 2 * C[i-1])
```

où C[-1] = 35, C est le tableau d'octets crypté et B le décrypté

(Nous remercions ici les anciens projets pour nous avoir fourni l'algorithme)

7F	0A	00	00	79	...	...	...	...	...	...	...	...	FF

Sans plus tarder, levons le voile sur le serveur *broad*. C'est un serveur HTTP sur lequel le Nabaztag va chercher ses sons et chorégraphies. Par exemple, pour un Messageblock contenant *CH broadcast/broad/toto.*mp3, le Nabaztag ira chercher le son sur l'URL http://broad.violet.net:80/broad/toto.mp3

(3) Autre message

La partie message difficile est terminée. Voici la partie la plus facile.

```
<message from='0019db9ed017@192.168.1.4/idle'to='int@xmpp.objects.violet.net/int' id='8'>
        <button xmlns="violet:nabaztag:button">
                <clic>
                        1
                </clic>
        </button>
</message>
```

```
<message from='0019db9ed017@192.168.1.4/idle'to='int@xmpp.objects.violet.net/int' id='8'>
        <button xmlns="violet:nabaztag:button">
                <clic>
                        2
                </clic>
       </button>
</message>
```

Ce sont les messages envoyés du nabaztag au serveur pour lui signaler qu'il y a eu un clic ou un double clic sur sa tête.

Dès qu'on bouge une oreille, le nabaztag envoie la nouvelle position des oreilles au serveur :

```
<message from='0019db9ed017@192.168.1.4/idle'to='int@xmpp.objects.violet.net/int' id='8'>
```

```
        <ears xmlns="violet:nabaztag:ears">
                <left>
                        6
                </left>
                <right>
                        0
                </right>
        </ears>
</message>
```

Nous avons ici fini avec la partie Jabber.

iii) Paquets HTTP

Il nous a été demandé de récupérer le paquet concernant le tag RFID. Dès que nous passons le tag devant le nez du lapin, nous captons ceci :

```
GET /vl/rfid.jsp?sn=0019db9ed017&v=18005&h=4&t=d0021a0353031f79 HTTP/1.0

User-Agent: MTL

Pragma: no-cache

Icy-MetaData:1

Host: tagtag.nabaztag.objects.violet.net
```

Nous avons maintenant l'information sur l'utilité du serveur *ping* : c'est le serveur web qui gère les tags. Concernant la requête, elle est assez claire.

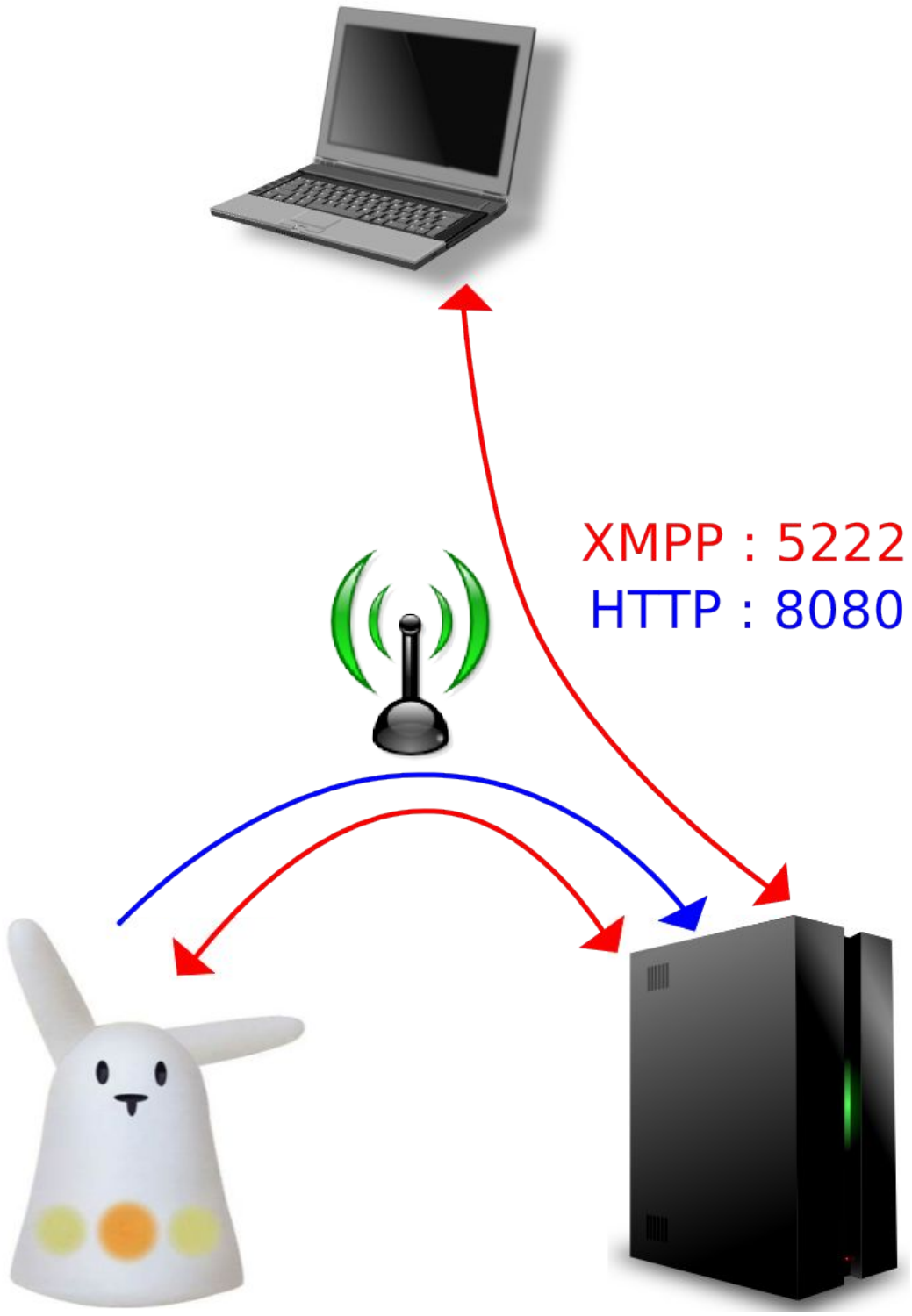
Nous ne nous sommes pas arrêtés là. Il nous manquait encore quelque chose : le microphone du Nabaztag. Il suffit de faire une capture supplémentaire :

```
POST /vl/record.jsp?sn=0019db9ed017&v=18005&h=4&m=0 HTTP/1.0

User-Agent: MTL

Pragma: no-cache

Icy-MetaData:1

Host: tagtag.nabaztag.objects.violet.net

Content-length: 2876
```

Là encore, le serveur *ping* est sollicité. Et juste à la suite, il y a le fichier son enregistré en format WAV dont la taille en octets est de 2876.

4) Choix de l'architecture

Nous avons besoin d'un serveur jabber et d'un serveur web voire de deux serveurs web dont un gérant le JSP. On nous avait demandé de faire une solution simple, rapide à installer. Nous nous sommes donc penchés sur la réalisation d'un serveur web tout simple, ne contenant que le strict minimum au lieu de l'installation d'un tomcat gérant le JSP. De plus, nous avons choisi de le lancer sur le port 8080 et non sur le 80 pour éviter le passage en mode super-utilisateur. Nous avons donc réuni le serveur *broad* et le serveur *ping* en un seul serveur.



5) Codage de la solution

a) Serveur web

Après une petite recherche Google sur la possible réalisation d'un serveur web en Java, nous sommes tombés sur SimpleWebServer (http://www.jibble.org/). Nous l'avons donc adapté à nos besoins.

b) Serveur Jabber

Pour ce serveur, nous sommes partis de 0. Il nous fallait un serveur qui distingue si un client est un Nabaztag ou si c'est une personne désirant contrôler un Nabaztag. Ceci se fait en analysant la première requête du client. Nous n'avons pas rencontré de difficulté ici. De part la nature du langage objet, il nous est possible de connecter plusieurs lapins sur le même serveur et de pouvoir les contrôler.

c) API client

Une fois connecté au serveur par l'intermédiaire d'un socket sur le port 5222, on peut interagir avec un Nabaztag comme ceci :

Message à envoyer	Description
<nabaztag>serial</nabaztag>	permet de contrôler le Nabaztag demandé
<ear side='l'>pos</ear>	bouge l'oreille gauche (-1 < pos < 17)
<ear side='r'>pos</ear>	bouge l'oreille droite (-1 < pos < 17)
<getearspos />	renvoie la position des oreilles sous cette forme : <ear side='l'>posleft</ear>

	<ear side='r'>posright</ear>
<sound>url_broad</sound>	joue un son
<tts>text</tts>	lit le texte demandé

Il y a aussi la possibilité d'envoyer des informations pour former un MessageBlock :

Message à envoyer	Description
<message>	
<sound>url</sound>	ajoute un son
<chor>url</chor>	ajoute une chorégraphie
<cl>instruction</cl>	permet un choix dans la couleur des LEDs
<pl>nombre</pl>	choisit la palette couleur pour la chorégraphie (entre 0 et 7)
<w />	attend la fin des commandes précédentes
</message>	

Nous avons développé deux classes Java permettant un contrôle facilité du Nabaztag en se basant sur la classe fournie par Sébastien Leriche, qui utilisait l'API web du Nabaztag.

d) Difficultés rencontrées

La première fut bien le multi-thread en Java. Nous n'en avions jamais fait. Mais cette lacune fut aisément combler avec dix minutes de recherche sur Internet.

La seconde fut la capture du WAVE enregistré par le Nabaztag. Il y avait des problèmes au niveau de la taille des paquets reçus et nous ne pouvions pas utiliser de String (encodage des caractères, caractères de fin de ligne '\n' et de fin de chaîne de caractères '\0').

A part ça, c'était de la manipulation de socket, de tableau et de String. Nous n'avons pas eu à utiliser une librairie pour le XML vu que le Nabaztag n'est pas très rigoureux sur les attributs dans les messages.

De manière plus pratique, il nous manquait un Nabaztag pour être vraiment efficace. Nous ne pouvions tester le code que chez une seule personne, ce qui nous a fait énormément de temps.

Sur certains routeurs, il faut un nom de domaine pour le serveur XMPP, dans le cas contraire, il envoie sans cesse des requêtes DNS.

6) Installation, configuration et compilation

a) Installation

Pour installer le serveur, décompressez l'archive dans un répertoire adapté. Vous devez y trouver un dossier *jNabNabServer* à la racine. Dans *bin/* il y a un répertoire *boot/* qui contient les fichiers nécessaire au démarrage du lapin et un dossier *broad/* contenant diverses ressources (dont le TTS et le son enregistré avec le microphone) Le *src/* contient les sources du projet.

b) Configuration

Il faut configurer le Nabaztag pour qu'il se connecte à notre serveur. Pour cela, connectez-vous sur le Nabaztag et réglez dans les paramètres avancés l'adresse de la plate-forme Violet comme ceci (adaptez selon l'IP du serveur) :

Ensuite, du côté serveur, allez dans le répertoire *bin/boot/* qui contient les fichiers nécessaire au démarrage du Nabaztag. Éditez le fichier *server.conf* et adaptez en respectant les retours à la ligne et les espaces. Par exemple, pour un serveur ayant l'IP 192.168.1.4 :

ping 192.168.1.4:8080

broad 192.168.1.4:8080

xmpp_domain 192.168.1.4

Voilà, vous êtes prêts à taper *java -jar jNabNabServer.jar* pour lancer le serveur !

c) Compilation

Le projet utilise la librairie freetts pour le Text-To-Speech, le restant étant plus standard. Pour compiler le projet, tapez simplement *ant build* à la racine du répertoire jNabNabServer.

7) Conclusion

Même si nous avons respecté le contrat initial, il y a beaucoup de chose perfectible. Ainsi, nous donnons quelques idées pour la suite du projet. A l'heure actuelle, tous les messages (ou presque) sont captés et traités mais n'ont pas d'actions propres. C'est à dire qu'ils ont une action par défaut. Par exemple, un simple clic sur la tête du lapin met les oreilles en position 14, et ceci est valable pour tous les lapins connectés. Il faudrait donc établir un système de configuration où l'association à ces fonctions se fait plus aisément que passer par les sources, en passant par exemple par des fichiers.

Il manque aussi un éditeur de chorégraphie. Il n'existe pas dans tous les projets que nous avons parcouru, un éditeur disposant d'une interface graphique intuitive. C'est même assez difficile d'en créer une sans passer quelques heures dessus.

L'API client ne permet que de contrôler qu'un seul lapin à la fois. Il serait peut-être bon de la modifier afin de prendre un compte un groupe de Nabaztag.

Dans tous les cas, le choix de ce projet fut facile, la réalisation beaucoup moins. Nous avons passé la plupart de notre temps sur l'analyse des messages et le choix de l'architecture. Une fois tout ceci éclairci, le développement fut plutôt facile et s'est déroulé sans difficulté majeure pour finalement être expédié en quelques jours.

Nous pensons que le fait d'avoir passé beaucoup de temps sur l'analyse et l'architecture a relayé le développement au rang de tâche secondaire. Nous avons même rajouté tout naturellement des fonctionnalités qui n'étaient pas demandées, car nous avions tous les éléments pour. Par ailleurs nous n'avons pas réalisé un proxy XMPP comme initialement demandé, mais un vrai serveur, que nous pouvons même commandé via un téléphone mobile (testé avec un Samsung Player Addict), grâce à une applet J2ME développée pour la présentation orale.

ANNEXES

Classe MessageBlock :

```
public class MessageBlock {

	String message;
	MessageBlock() {
		this.message = new String();
	}

	public void addSound(String url) {
		this.message += "<sound>" + url + "</sound>";
	}

	public void addChoregraphy(String url) {
		this.message += "<chor>" + url + "</chor>";
	}

	public void addWait(String url) {
		this.message += "<w />";
	}

	public void addPresetColor(int r, int g, int b) {
		if (r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255 ) return;
		this.message += "<cl>" + (b + (g * 255) + (r * 255 * 255)) +"</cl>";
	}

	public void addPalette(int pl) {
		if (pl < 1 || pl > 7) return;
		this.message += "<pl>" + pl + "</pl>";
	}

	public String generate(){
		return "<message>"+this.message+"</message>";
	}

}
```

Classe Nabaztag :

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

public class Nabaztag {

	private String serial;
	private Socket s;
	private InputStream in;
	private PrintWriter out;

	Nabaztag(String serial) {
		this.serial = serial;
	}

	public void connect(String ip) {
		try {
			s = new Socket(ip,5222);
			in = this.s.getInputStream();
			out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(s.getOutputStream())),true);
		} catch (Exception e) {
			e.printStackTrace();
			System.exit(1);
		}

		out.print("<nabaztag>"+serial+"</nabaztag>");
		out.flush();
	}


	public void setRightEarPos(int pos) {
		if(pos < 17 && pos > -1) {
			out.print("<ear side='r'>"+pos+"</ear>");
			out.flush();
		}
	}

	public void setLeftEarPos(int pos) {
		if(pos < 17 && pos > -1) {
			out.print("<ear side='l'>"+pos+"</ear>");
			out.flush();
		}
	}

	public void play(String url) {
```

```
		out.print("<sound>"+url+"</sound>");
		out.flush();
	}

	public void speak(String text) {
		out.print("<tts>"+text+"</tts>");
		out.flush();
	}

	public int[] getEarsPos() {

		out.print("<getearspos />");
		out.flush();
		byte[] buf = new byte[128];

		try {
			in.read(buf);
		} catch (IOException e) {
			e.printStackTrace();
		}

		String posi = new String(buf);
		int tmp = posi.indexOf("</ear>");
		int[] pos = {Integer.parseInt(posi.substring(14,
tmp)),Integer.parseInt(posi.substring(tmp+20, posi.lastIndexOf("</ear>")))};

		return pos;

	}


	public void sendMessageBlock(MessageBlock msg) {
		out.print(msg.generate());
		out.flush();
	}


	public void disconnect() {
		try {
			s.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Bibliographie

Nabaztag (site officiel) : http://www.nabaztag.com/fr/index.html

Nabaztag (wikipédia) : http://fr.wikipedia.org/wiki/Nabaztag

Violet : http://fr.wikipedia.org/wiki/Violet_(entreprise)

API du Nabaztag : http://api.nabaztag.com

Projet d'un serveur local : http://www.cs.uta.fi/hci/spi/jnabserver/

Base 64 : http://fr.wikipedia.org/wiki/Base_64