

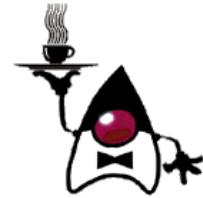
**Java (IV)**  
**Programmation de protocoles applicatifs et**  
**de techniques de sécurité**

**Claude VILVENS**

claude.vilvens@hepl.be



## Sommaire



## Introduction

## Protocoles

### XXI. Les protocoles de messagerie électronique et Java Mail

<b>1. Le format standard des messages Internet</b>	
1.1 Le format général	3
1.2 Les principaux headers de MFS	4
<b>2. Les spécifications MIME</b>	
2.1 Des messages plus évolués	5
2.2 La description du contenu d'un message	6
2.3 Les encodages	7
<b>3. Le fonctionnement d'un système de messagerie Internet</b>	8
<b>4. Le protocole SMTP</b>	
4.1 Une brève présentation	11
4.2 Les commandes de base	11
4.3 Un exemple de transaction SMTP en ligne de commande	12
<b>5. Un MTA classique : Sendmail</b>	
5.1 Une brève présentation	14
5.2 Le fonctionnement de Sendmail	14
5.3 Les principaux fichiers utilisés par Sendmail	14
5.4 Un aperçu du fichier de configuration	15
<b>6. Les e-mails et des packages javax en plus ...</b>	17
<b>7. L'évolution des JavaBeans et le JAF</b>	18
<b>8. Une session mail</b>	19
<b>9. L'interface Part</b>	23
<b>10. Les classes messages</b>	
10.1 La classe de base	23
10.2 La classe e-mail classique	24
<b>11. L'envoi d'un message</b>	25
<b>12. Le protocole POP3</b>	
12.1 Une brève présentation	27
12.2 Les commandes de base	28
12.3 Les commandes de base	29
<b>13. La réception d'un message avec JavaMail</b>	30
<b>14. Du bon emploi des jeux de caractères</b>	33

<b>15. Les headers d'un message reçu</b>	<b>34</b>
<b>16. L'examen des headers avec JavaMail</b>	
16.1 L'envoi et la réception	35
16.2 Les headers du message n°1	37
16.3 Les headers du message n°2	38
<b>17. Les messages avec pièces attachées</b>	<b>40</b>
<b>18. L'envoi d'un e-mail avec attachment</b>	<b>42</b>
<b>19. La réception d'un e-mail avec pièces attachées</b>	<b>46</b>
<b>20. Résumé : classes et interfaces utiles</b>	<b>49</b>
<b>21. Le protocole IMAP</b>	
21.1 Une brève présentation	51
21.2 Les commandes de base de consultation des messages	53
21.3 Un exemple de transaction IMAP en ligne de commande	54
21.4 Les boîtes de réception	56
<b>22. IMAP et Java</b>	<b>59</b>
<b>23. Les événements de type mail</b>	<b>60</b>
<b>24. La suppression des messages</b>	<b>64</b>

## XXII. Le protocole de transfert de fichiers FTP et Java FTP

<b>1. Le protocole FTP</b>	<b>65</b>
<b>2. Le modèle FTP</b>	<b>66</b>
<b>3. L'interface de commande FTP</b>	<b>67</b>
<b>4. Les principales commandes d'accès et de service</b>	<b>69</b>
<b>5. Les principales commandes de paramétrage du transfert</b>	<b>71</b>
<b>6. Un package Java pour un serveur FTP</b>	<b>72</b>
<b>7. Un serveur FTP en Java</b>	
7.1 La classe FtpServer	77
7.2 L'aspect multithread	78
7.3 Une queue moniteur	80
7.4 La classe de connexion FTP	81
7.5 La classe StreamConnector	86
7.6 Un moteur à serveurs	87
7.7 Le diagramme de classes UML	88
<b>8. Un minuscule serveur FTP maison</b>	<b>90</b>
<b>9. Un package Java pour un client FTP</b>	<b>95</b>
<b>10. Un client FTP en Java</b>	
10.1 La classe FTPClient	96
10.2 Les classes sockets	99
10.3 Les classes annexes	101
<b>11. Un petit client FTP maison</b>	<b>102</b>

## **XXIII. Le protocole SNMP et l'administration des réseaux**

<b>1. Les objectifs de l'administration réseau</b>	<b>109</b>
<b>2. Un protocole d'exploration basé sur UDP</b>	<b>109</b>
<b>3. La représentation et l'identification de l'information dans la MIB</b>	<b>110</b>
<b>4. Les messages SNMP v1 et v2</b>	
4.1 Les requêtes et réponses SNMP v1 et v2	113
4.2 Les traps SNMP v1	115
<b>5. Différentes versions de SNMP</b>	
5.1 Les versions 1 et 2	115
5.2 La version 3	116
<b>6. L'architecture SNMP v3</b>	
6.1 Le manager SNMP	117
6.2 L'agent SNMP	118
6.3 Les requêtes et réponses SNMP v3	118
6.4 Les mécanismes de sécurité de SNMP v3	119
<b>7. L'installation et le configuration d'un agent SNMP sous Windows</b>	<b>120</b>
<b>8. Un exemple de manager SNMP</b>	<b>122</b>
<b>9. Le package WareMaker pour SNMP en Java et la commande get</b>	<b>125</b>
<b>10. Analyse du trafic réseau pour une commande get</b>	<b>129</b>
<b>11. La commande set en Java</b>	<b>130</b>
<b>12. La commande get-next en Java</b>	<b>133</b>
<b>13. La réception des traps</b>	<b>135</b>
<b>14. Les particularités des commandes SNMP v3</b>	<b>136</b>
<b>15. La découverte du réseau</b>	
15.1 Programmer un ping	137
15.2 Explorer une range d'adresses connues	140
15.3 Déterminer la topologie du réseau : le principe	144

## *Sécurité et cryptographie*

## **XXIV. Le modèle de sécurité Java**

<b>1. Les chargeurs de classes</b>	
1.1 La mise en mémoire d'une classe dans une machine virtuelle	145
1.2 Le vérificateur de classes	145
1.3 La classe de base des chargeurs complémentaires	146
1.4 Une hiérarchie de base pour les chargeurs de classes	148
1.5 La référence vers le chargeur de classe	149
1.6 Un chargeur de classe personnalisé	150
<b>2. Le Security Manager</b>	
2.1 Le big brother des méthodes sensibles	155
2.2 Les méthodes checkXXX()	156
2.3 Un SecurityManager à mot de passe	157
<b>3. Le contrôleur d'accès</b>	
3.1 Vers un contrôle plus fin	159

3.2 Le code source	159
3.3 Les permissions	159
3.4 Le fichier et la classe Policy	160
3.5 La gestion des Policy	162
3.6 Le retour du Security Manager	163
3.7 L'ange gardien : le contrôleur d'accès	163
3.8 Les domaines de protection	165
3.9 En résumé	165
<b>4. L'ouverture de la sécurité d'une applet</b>	
4.1 Une applet locale qui veut lire un fichier depuis l'EDI	167
4.2 Une applet locale qui veut lire un fichier depuis un browser	170
<b>5. Les permissions avec des signatures et des certificats</b>	<b>172</b>
<b>6. L'ouverture authentifiée de la sécurité d'une applet</b>	
6.1 La signature de l'applet	173
6.2 La préparation de l'utilisation de l'applet signée	175
6.3 La permission avec une signature en local	176
6.4 La permission avec une signature à partir d'un serveur Web	178

## XXV. Le protocole SSL, JSSE et le e-commerce sécurisé

<b>1. Le schéma classique du commerce électronique</b>	<b>180</b>
<b>2. Les protocoles SSL et HTTPS</b>	
2.1 Les objectifs	181
2.2 Un protocole entre deux couches	181
2.3 Un bref historique	181
<b>3. Le dialogue SSL de création d'une session</b>	
3.1 Le principe général	182
3.2 Les sous-protocoles de SSL	183
3.3 Le sous-protocole "SSL Handshake"	183
3.4 Les variables d'état SSL	186
3.5 Le sous-protocole "SSL Alert"	187
3.6 Le sous-protocole "SSL Record"	188
3.7 Les faiblesses de SSL	188
3.8 Le principe de la signature duale	189
<b>4. Les packages Java pour SSL</b>	
4.1 Les classes sockets pour SSL	190
4.2 Obtenir les factories dans un contexte SSL	191
4.3 Keystore : le retour	192
4.4 L'initialisation du contexte SSL	193
4.5 Les sockets et le code du serveur	194
4.6 Une synthèse en diagramme UML	196
4.7 Le code du client	198
4.8 La visualisation du trafic réseau	199
<b>5. Tomcat par rapport à SSL</b>	
5.1 Un keystore pour HTTPS	200
5.2 La création d'un connecteur https	203
5.3 Utilisation du browser en https	204
<b>6. Une communication applet-servlet sécurisée par SSL</b>	<b>207</b>

## XXVI. La génération des certificats

<b>1. Rappels : les outils keytool et keytool IUI</b>	<b>211</b>
<b>2. L'outil OpenSSL</b>	
2.1 Présentation et installation	212
2.2 La création d'un certificat auto-signé pour un CA	215
2.3 La création des keystores	217
2.4 La génération de certificats validés	219
<b>3. La gestion des certificats et keystores par programmation</b>	<b>228</b>

## Ouvrages consultés

## Annexe : Les codes de retour des commandes FTP



## Introduction



Peut-on tout faire en Java ?

Evidemment, on s'attendrait de ma part à une réponse affirmative. Mais, en fait, je serais plus nuancé, quelque chose du genre : "on peut tout faire en Java, pour peu que l'on reste à un niveau d'abstraction suffisant" ...

Mouais ... Mais encore ?

On ne peut pas programmer en Java au niveau du microprocesseur – à moins que l'on ait affaire à un processeur dont le microcode est du bytecode Java (cela existe).

On ne peut pas lancer des appels systèmes ou converser avec le noyau du système d'exploitation – et pour cause, c'est le principe même de la machine virtuelle qui nous en empêche. Ou alors, on paie le prix des méthodes natives ...

Et l'on ne peut pas non plus espérer obtenir une application très performante, économisant la mémoire au bit près – d'abord parce que l'exécution est une interprétation, ensuite, précisément, parce que le langage est abstrait et très typé.

Par contre, on peut programmer en Java les questions concernant la messagerie électronique. La bibliothèque correspondante répond au nom de **Java Mail**. Elle surprend quelque peu. En effet, loin de se contenter d'encapsuler les protocoles **SMTP-POP3-IMAP** (qu'il nous faudra approfondir), elle propose une réflexion plus générale sur la notion de message.

Le protocole **FTP** (que tout le monde croit connaître, mais bon ...) semble échapper à Java : pas de trace du célèbre protocole de transfert de fichier dans les librairies. Mais ce serait mal connaître les ressources du Web, qui ont permis de trouver des packages, œuvres signées par leur auteur, tant pour un serveur que un client. On peut donc vraiment parler de **Java FTP** ...

Dans le même ordre d'idée, pas de trace non plus au sein des librairies standards de **SNMP** (Simple Network Management Protocol), protocole utilisé dans la gestion et l'administration des grands réseaux. Mais, ici encore, une librairie Java offerte par son auteur va nous permettre d'investiguer ...

Voilà qui devrait justifier la première partie du titre du présent travail : nous allons effectivement nous intéresser à des protocoles applicatifs basés sur TCP ou UDP.

Cependant, tous ces protocoles seraient d'un intérêt presque académique s'ils n'étaient pas accompagnés de questions, bien actuelles, de sécurité.

Nous serons donc amenés à nous interroger sur la **sécurité** interne de la plate-forme Java. Seule une compréhension profonde de cette architecture nous permettra d'ouvrir la sécurité de nos applets afin de leur permettre de payer leur caddie virtuel avec une carte Proton ...

---

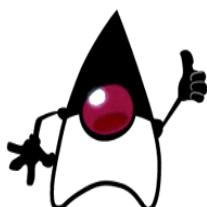
Après la sécurité de la plate-forme, nous passerons à la sécurité des communications réseau en domestiquant le protocole **SSL** (Secure Socket Layer) et la librairie associée **JSSE** (Java Secure Socket Extension). Ceux qui ne savent pas ce qu'est un **certificat** feraient bien de s'abstenir ;-) ... Et d'ailleurs le dernier un chapitre renforcera l'attirail de nos outils permettant de gérer ces derniers.

Ainsi armés, nous pourrons envisager l'étude des deux autres plates-formes Java : la technologie Java Card des cartes à puces et celles de la plate-forme J2ME caractéristique des GSM et Pocket PC – sans parler d'une fierté nationale (ça existe ?) : la carte d'identité électronique belge ! Et puis viendront aussi diverses technologies de conception et de développement, notamment dans le domaine de l'e-commerce : des frameworks comme Struts (ou les Faces) qui facilitent et systématisent la mise en place d'un modèle MVC, la solution Java Web Start qui semble vouloir allier les avantages des applets et des applications sans reprendre leurs inconvénients respectifs, les serveurs d'application comme Tomcat et Glassfish, etc ...

Mais du calme : ce sera pour Java V et VI ;-)

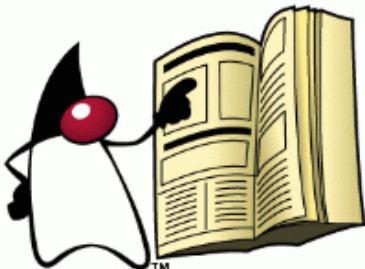
**Claude Vilvens**

*P.S. Les exemples de programmation développés ici l'ont été en utilisant principalement Netbeans 6.\* et 5.\* (ainsi que Sun ONE Studio pour quelques anciens exemples). On utilise, en la version considérée ici, le JDK 1.6.*



Reprenons à nouveau le fil de nos idées ! Les chapitres sont numérotés en poursuivant la séquence du volume III. Nous commençons donc par le chapitre XXI qui va nous fournir des classes incontournables : celles des packages voués à la messagerie électronique ...

## XXI. Les protocoles de messagerie électronique et Java Mail



*Monsieur le Président  
Je vous fais une lettre  
Que vous lirez peut-être  
Si vous avez le temps*

(B. Vian, Le déserteur)

Notre étude des servlets et de la cryptographie nous a emmenés dans le contexte du e-commerce. Celui-ci, du point de vue programmation, utilise encore bien d'autres techniques, comme les Java server Pages, les Enterprise Java Beans et aussi les Java Mail.

Ces derniers vont nous permettre de gérer des e-mails ("courriels" en français) et sont donc des éléments incontournables dans la conception d'un site e-commerce professionnel. L'élément de base est évidemment le concept de message ...

### 1. Le format standard des messages Internet

#### 1.1 Le format général

Dans un contexte aussi hétérogène qu'Internet, il est tout à fait compréhensible qu'est apparue très tôt la nécessité de définir le format commun que toute entité *texte* prétendant être un "message électronique" se devait de respecter. Ce format est le **MFS** (mouais – Internet Message Format Standard), défini à la base dans la RFC 822.

Un message suit la structure suivante (of course, CR désigne un "Carriage Return" - code 13 - et LF désigne "Line Feed" - code 10) :

```
<message> = <en-tête du message [message header fields]>
           CR LF
           <corps du message [message body]>
```

L'en-tête [**headers**] comporte toute une série d'informations sur le message (comme son expéditeur, la date d'émission, etc) tandis que le corps [**body**] représente les données qui constituent effectivement l'information. On dit encore que les headers constituent l'**enveloppe** du message, en ce sens qu'ils représentent toutes les informations nécessaires pour véhiculer le message (comme le nom, l'adresse pour un courrier traditionnel). Cette enveloppe peut cependant être modifiée par les applications d'acheminement du message (tout comme la poste traditionnelle peut ajouter des indications de service sur l'enveloppe).

Les headers (pour être exact, il faudrait dire les "champs de header" [*header fields*] – mais nous utiliserons "header" par abus de langage) suivent la syntaxe simple :

```
<nom du header>: <valeur du header> CR-LF
```

Par exemple :

```
Date: Thu, 21 Mar 2002 12:56:35 +0100
From: vilvens@u2.wildness.loc
Subject: Quand est-ce qu'on mange ?
```

De plus, les headers sont construits en tenant compte des règles suivantes :

- ◆ les seuls caractères utilisables sont les caractères ASCII américain;
- ◆ le nom du header est indifféremment libellé en majuscules ou minuscules – mais il n'en est pas forcément de même pour la valeur;
- ◆ une ligne comporte 998 caractères maximum(sans compter CRLF);
- ◆ un CR doit toujours être suivi d'un LF; ce couple CR-LF marque la fin d'un header, sauf s'il suit un espace (code 32) ou une tabulation (code 9), auquel cas le header se poursuit;
- ◆ un header peut être structuré (il se compose de plusieurs parties, selon une syntaxe bien définie – par exemple, une date) ou non structuré (il comporte une séquence de caractères quelconque – par exemple, le sujet du message).

## 1.2 Les principaux headers de MFS

Deux headers seulement sont absolument nécessaires : il s'agit des headers **From** et **Date**. Bien sûr, bon nombres d'autres headers existent et un certain nombre d'entre eux sont vivement souhaitables. Voyons ici les headers les plus immédiatement utiles.

### A. Origine

#### **From:** <expéditeur>

Spécifie évidemment l'identité de l'expéditeur, avec son adresse e-mail.

#### **Date:** <date de soumission>

Il s'agit de la date qui correspond au moment où l'expéditeur considère que le message a été soumis au service de messagerie pour envoi. La date réelle d'envoi du message sur le réseau peut donc être postérieure (par exemple, parce que le message a été mis en file).

### B. Destination

#### **To:** <destinataire(s)>

#### **Cc:** <destinataire(s)>

#### **Bcc:** <destinataire(s)>

Faut-il vraiment les présenter ? Pour rappel, les destinataires spécifiés dans le header Bcc (Blind Carbon Copy) ne sont pas vus par les autres destinataires des headers To et Cc.

### C. Identification

#### **Message-Id:** <identificateur unique>

A ne pas confondre avec le nom de l'expéditeur, ce header doit contenir un identificateur unique sur Internet : le plus souvent, il comporte plutôt une partie unique sur Internet, comme l'identification de la machine, et une partie unique sur cette machine.

### D. Information

#### **Subject:** <texte sujet du message>

Bien sûr, il s'agit d'une brève description du message.

#### **Content-Type :** <codes MIME>

Pour être en accord avec MFS, ce champ devrait contenir la valeur :

**text/plain; charset=US-ASCII.**

mais nous allons y revenir plus précisément.

#### E. Headers propriétaires

**X-<nom\_propriétaire>: <valeur au sens propriétaire>**

Il s'agit de headers définis par des utilisateurs pour les besoins de leur propre protocole applicatif. Ainsi, dans un contexte de e-commerce, on pourrait imaginer X-Code-Product.

Nous évoquerons d'autres headers en cas de besoin. L'ordre selon lequel ces headers sont déclinés est sans importance. Cependant, l'ordre suivant est celui qui semble le plus souhaitable :

Date:  
From:  
Subject:  
To:  
Cc:  
<autres>

MFS	
Nom complet :	Internet Message Format Standard
Nature :	Protocole de spécification de la nature des données transmises
RFC :	822-2076
Utilisé par :	SMTP, POP3
Champs courants :	Date, From, To, Cc, Subject

## **2. Les spécifications MIME**

### **2.1 Des messages plus évolués**

MFS suppose que les messages considérés sont constitués par du texte encodé au format US-ASCII à 7 bits – on voit les limites inacceptables. L'acronyme **MIME** (Multipurpose Internet Mail Extensions) désigne les spécifications d'un protocole décrivant

comment représenter des données de nature diverse (texte encodé autrement qu'en ASCII de base, flux binaires comme des images ou des sons, messages composites) de manière à ce qu'elles puissent être utilisées par un système de messagerie électronique n'utilisant uniquement que des messages à priori de type texte.

Ces spécifications sont détaillées dans les RFCs 2045-2046-2047-2048-2049.

Autrement dit, l'objectif est donc de décrire les données contenues dans le corps du message de manière à ce que le **user-agent** (c'est-à-dire le logiciel derrière lequel se cache l'utilisateur) qui reçoit le message puisse mettre en place les dispositifs nécessaires pour lire ces données ou, à tout le moins, mettre en place un mécanisme quelconque de conversion.

<b>MIME</b>	
Nom complet :	<b>Multipurpose Internet Mail Extensions</b>
Nature :	Protocole de spécification de la nature des données transmises
RFC :	2045-2046-2047-2048-2049
Utilisé par :	HTTP, SMTP, POP3
Types courants :	text/plain, text/html, application/octet-stream
Codages courants :	7bit, quoted-printable, base64

## 2.2 La description du contenu d'un message

L'idée est donc qu'un message e-mail comporte, outre le message proprement dit, un header comportant divers champs décrivant au mieux le type de son contenu. Ainsi, un e-mail des plus simples aura pour en-tête :

From: James Vil < <a href="mailto:civilvens@prov-liege.be">civilvens@prov-liege.be</a> >	
To: Francesca < <a href="mailto:francesca@skynet.be">francesca@skynet.be</a> >	champs du protocole de messagerie
Subject: Libre ce soir ?	
MIME-Version: 1.0	champs MIME
Content-type: text/plain	

Les deux principaux champs du header MIME sont donc :

- ♦ **le champ de version** (MIME-Version) :

<b>MIME-Version</b> : <version de MIME>
---

Par exemple :

MIME-Version: 1.0

- ♦ **le champ du type du contenu du message** :

<b>Content-Type</b> : <type>/<sous-type> [; informations complémentaires]
---

Ce champ spécifie la nature des données contenues dans le message en fournissant, de manière normalisée, le type et le sous-type, avec éventuellement des informations complémentaires pour les types qui en réclament. Il existe 7 types de base :

<b>type</b>	<b>définition</b>	<b>quelques sous-types courants</b>
text	information textuelle	<ul style="list-style-type: none"> <li>♦ plain : texte non formaté – aucun software complémentaire n'est nécessaire pour appréhender l'entièreté du message – il est possible de spécifier le jeu de caractères par une clause charset :</li> </ul> <p>Content-type: text/plain; charset=us-ascii  Content-type: text/plain; charset=iso-885-5</p> <ul style="list-style-type: none"> <li>♦ html : le texte comporte des tags HTML</li> </ul>

image	bien sûr, il s'agit d'images au format gif ou jp(e)g	<ul style="list-style-type: none"> <li>◆ gif</li> <li>◆ jpeg</li> </ul>
audio	informations nécessitant un dispositif de son	<ul style="list-style-type: none"> <li>◆ basic</li> </ul>
video	à votre avis ?	<ul style="list-style-type: none"> <li>◆ mpeg</li> </ul>
application	informations binaires, à priori destinées à être exécutées par une application basée sur la messagerie	<ul style="list-style-type: none"> <li>◆ octet-stream : données binaires brutes – c'est le sous-type utilisé lorsqu'une application client ne connaît pas le type/sous-type précisé dans le message reçu</li> <li>◆ postscript : pour transporter des informations codées en postscript</li> </ul>
message	message faisant partie d'un autre message et susceptible de contenir des informations d'une autre nature que celle du message principal	<ul style="list-style-type: none"> <li>◆ rfc822 : selon la RFC de base de la messagerie électronique</li> </ul>
multipart	données composées de groupes d'informations de types différents	<ul style="list-style-type: none"> <li>◆ mixed : les différentes composantes sont indépendantes et doivent être fournies dans un ordre précis</li> <li>◆ alternative : les différentes composantes sont des versions différentes de la même information – le choix se fera selon les préférences déclarées du récepteur</li> <li>◆ parallel : l'ordre des différentes parties n'a pas d'importance</li> <li>◆ related : le message est une page WEB consistant en son code HTML, mais aussi en les ressources qu'il référence.</li> </ul>

Le champ Content-Type peut encore utiliser des types propriétaires, référencés par "X-...", encore qu'il soit plutôt recommandé de définir des sous-types des 7 types existants (référencés par "x-..."). Ainsi, par exemple et comme nous l'avons vu, le mécanisme des servlets Java utilise de nouveaux sous-types du type application comme x-www-form-urlencoded ou x-java-serialized-object.

### 2.3 Les encodages

D'autres champs de header ont été définis, dont celui, bien qu'optionnel, qui est l'essence même de MIME : le champ d'encodage

#### Content-Transfer-Encoding: <type d'encodage>

Ce champ permet de spécifier comment l'information du message (ou d'une composante du message) a été codée pour être transférée vers le destinataire et donc aussi comment la décoder. Les valeurs possibles de ce champ sont :

- ◆ **7bit, 8bit et Binary** : il s'agit des représentations en ASCII 7 bits (américain), ASCII 8 bits et chaîne d'octets. En fait, il n'y donc aucun codage particulier (on parle encore

d'identity) et les logiciels de transfert (les MTAs) peuvent donc réaliser leur propre encodage selon les besoins de transport sur le réseau.

- ◆ **Quoted-printable** : il s'agit de données sous forme de texte classique, si ce n'est que
  - les octets sont codés en big-endian;
  - les caractères autres que ceux de code ASCII 33 à 60 et 62 à 126 ainsi que toutes les formes d'espaces (c'est-à-dire 9 et 32) doivent être codés sous forme d'un signe "=" suivi de leur valeur hexadécimale (sur deux digits); bien clairement, ce type d'encodage concerne les messages textes utilisant des caractères non anglophones. Comme le résultat du codage est spécifique (ce n'est plus du texte de base), le message, bien que plus ou moins lisible, ne sera pas modifié par le processus de transport MFS standard.
- ◆ **Base64** : décompose les groupes successifs de 24 bits (3 octets supposés être en big-endian) en 4 groupes de 6 bits. Chacun de ces groupes est alors envoyé en tant que caractère ASCII, selon la convention que "A" correspond à 0, "B" à 1, ..., "Z" à 25, "a" à 26, ..., "z" à 51, "0" à 52, "9" à 61, "+" à 62 et "/" à 63. On utilise donc un jeu de 64 caractères. Comme la fin du message peut contenir des octets vides, ceux-ci sont remplis (*padding*) avec "==" (il n'y a plus que deux groupes de 6 bits) et "=" (il n'y a plus que trois groupes de 6 bits). Il s'agit évidemment du format destiné à tout ce qui n'est pas texte devant être compris tel quel par le destinataire (comme une image, un programme compilé, etc). A nouveau, le résultat du codage est spécifique (ce n'est plus du texte de base) et le message, cette fois totalement illisible, ne sera pas modifié par le processus de transport MFS standard.

Maintenant que la structure de base d'un message, et sa représentation, nous sont connues, voyons selon quels principes un tel message est véhiculé sur Internet.

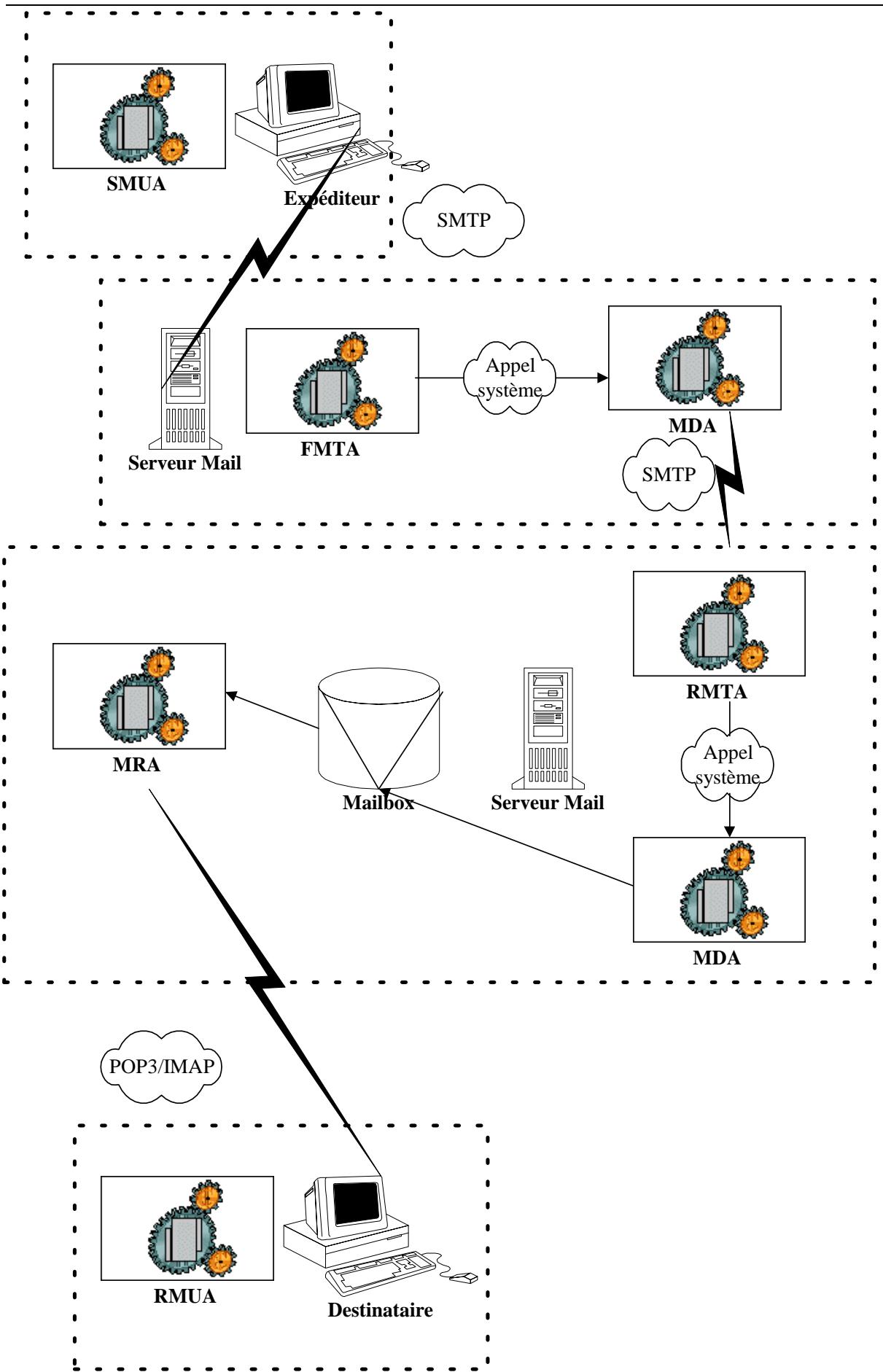
### **3. Le fonctionnement d'un système de messagerie Internet**

Pour qu'un message soit acheminé depuis son expéditeur jusqu'à son destinataire, il faut l'intervention de plusieurs acteurs (concrètement, des applications), que la littérature appelle des "agents". Selon le scénario le plus simple :

- ◆ l'expéditeur rédige son message, puis demande à une application (MS-Outlook, Netscape Messenger, ...) de l'envoyer : celle-ci est un **MUA** (**Mail User Agent**) et même un **SMUA** (**Sending MUA**);
- ◆ le MUA contacte le serveur de mail (Sendmail, MS-Exchange, ...) qui est une application dont le rôle est de préparer le transfert du message de la machine vers une autre en décodant l'en-tête du message et en déterminant la meilleure méthode de transfert : il constitue un **MTA** (**Mail Transfer Agent**) et même plus précisément un **FMTA** (**Forwarding MTA**);
- ◆ le MTA utilise alors un programme appelé un **MDA** (**Mail Delivery Agent**) ; ce programme reçoit un message d'un MTA et le délivre
  - soit à un autre MTA, auquel le MDA se connecte, si le message n'est pas destiné à un utilisateur de la machine sur laquelle le MDA fonctionne;
  - dans la boîte aux lettres du destinataire (en fait, dans un fichier) si celui-ci est connu sur la machine locale;

- ◆ le message passe donc par le réseau pour atteindre un autre MTA qui gère le compte courrier du destinataire; mais il est possible qu'il soit passé par des MTAs intermédiaires parce que le destinataire n'a pas de compte sur les premiers MTAs atteints; les MDAs déterminent dans l'adresse le domaine de destination et retrouvant le MTA visé au moyen d'un DNS; un MTA peut aussi faire office de passerelle pour transférer le message d'un réseau à un autre de nature différente;
- ◆ le MTA atteint finalement, qui est un **RMTA** (Receiving MTA), utilise un MDA qui écrit le message dans la boîte aux lettres du destinataire;
- ◆ le destinataire va lire les messages de sa boîte aux lettres au moyen d'une application qui est aussi un MUA, plus précisément un **RMUA** (Receiving MUA); si la boîte aux lettres n'est pas locale au destinataire mais distante (cas le plus fréquent), le MUA appellera un **MRA** (Mail Retrieval Agent) dont le rôle est de lire la boîte aux lettres distante.

Schématiquement :



On peut constater que le transfert des messages de l'expéditeur jusqu'à la boîte aux lettres du destinataire utilise le protocole SMTP tandis que la lecture des messages de la boîte aux lettres par son propriétaire se fait en utilisant les protocoles POP3 ou IMAP.

Commençons donc par nous préoccuper de l'envoi d'un message et abordons donc le protocole SMTP ...

## 4. Le protocole SMTP

### 4.1 Une brève présentation

Protocole de la couche application, **SMTP** (Simple Mail Transfer Protocol) est un protocole simple permettant le transfert du courrier électronique sur Internet. Pour faire court :

<b>SMTP</b>	
Nom complet :	<b>Simple Mail Transfer Protocol</b>
Nature :	Protocole applicatif de transfert de messages électroniques
RFC :	821-2821
Port par défaut :	25
Protocole de transport :	TCP
Principales commandes :	helô, mail from, rcpt to, data, quit

Les commandes de ce protocole ne sont pas très nombreuses. Elles sont toujours formées de 4 lettres, suivies éventuellement de paramètres de type "from:" ou "to:".

### 4.2 Les commandes de base

Pour parvenir à envoyer un message en utilisant SMTP, après s'être connecté au serveur, les commandes de base suivantes sont nécessaires :

Commande	Description
<b>helô &lt;adresse IP machine expéditeur&gt;</b>	Permet de s'identifier en tant que machine émettrice
<b>helô &lt;nom machine expéditeur&gt;</b>	Permet de spécifier l'adresse e-mail de l'émetteur
<b>mail from: &lt;adresse e-mail expéditeur&gt;</b>	Permet de spécifier l'adresse e-mail du récepteur
<b>rcpt to: &lt;adresse e-mail destinataire&gt;</b>	Représente le corps du message – il se termine par un point isolé sur une ligne
<b>data &lt;message&gt;</b>	A votre avis ? En fait, envoie tous les messages mis en attente dans la file d'envoi
<b>quit</b>	Permet d'annuler le message en cours de transaction.
<b>rset</b>	Opération vide = "j'existe" ...
<b>noop</b>	Pour obtenir la liste des commandes SMTP supportées par le serveur
<b>help</b>	



**SMTP**

*Quelques codes de retour*

<b>Code</b>	<b>Signification</b>
214	Message d'aide
220	Service disponible
221	Canal de transmission en cours de fermeture
250	Commande reçue exécutée avec succès
251	Utilisateur non local avec relais automatique – le message est réémis vers l'adresse
354	Début de l'encodage d'un message
450	Erreur : boîte aux lettres non disponible (l'accès est impossible ou bloqué)
451	Erreur : erreur dans le traitement local du message
452	Erreur : pas assez de ressources système
500	Erreur : faute de syntaxe dans la commande
501	Erreur : faute de syntaxe dans les paramètres de la commande
502	Erreur : commande non implémentée
504	Erreur : paramètre de commande non implémenté
550	Erreur : boîte aux lettres inaccessible
551	Erreur : utilisateur non local sans relais automatique
552	Erreur : manque de ressource de stockage
553	Erreur : nom de boîte aux lettres non autorisé

On remarquera les codes 251 et 551 qui correspondent au cas où l'adresse du destinataire est incorrecte, mais avec un récepteur SMTP connaissant l'adresse correcte : celui-ci peut prendre la responsabilité d'envoyer le message à l'adresse correcte (251) ou pas (551 + spécification de l'adresse correcte).

#### **4.3 Un exemple de transaction SMTP en ligne de commande**

Point n'est besoin de programmer classiquement en TCP/IP pour tester ces commandes. Nous allons utiliser telnet pour nous connecter à un serveur de messagerie et envoyer à celui-ci nos commandes SMTP. Pour rappel :

<b>TELNET</b>	
Nom complet :	TERminaL NETwork
Nature :	Emulation d'un terminal local sur le réseau
RFC :	435 – 495 – 854 – 861 - très nombreuses RFC s complémentaires
Port par défaut :	23
Protocole de transport :	TCP

Concrètement, nous allons nous connecter depuis une machine Windows à un serveur courrier de Skynet :

| C:\VILVENS>**telnet relay.skynet.be 25**

On peut vérifier que la connexion est bien établie, avec une adresse IP attribuée dynamiquement :

```
C:\VILVENS>netstat -an
```

#### Connexions actives

Proto	Adresse locale	Adresse distante	État
TCP	0.0.0.0:1052	0.0.0.0:0	LISTENING
...			
TCP	127.0.0.1:1025	0.0.0.0:0	LISTENING
...			
TCP	192.168.2.2:137	0.0.0.0:0	LISTENING
...			
TCP	195.238.12.152:1052	149.174.38.142:80	ESTABLISHED
TCP	195.238.12.152:1053	149.174.38.142:80	ESTABLISHED
TCP	195.238.12.152:1071	<b>195.238.3.128:25</b>	<b>ESTABLISHED</b>
...			
UDP	192.168.2.2:137	*:*	
...			

On peut dès lors imaginer le dialogue suivant :

```
220 vador.skynet.be ESMTP Sendmail 8.11.6/8.11.6/Skynet-OUT-2.20; Wed, 7 Aug 200
2 11:03:28 +0200 (MET DST)
helo 195.238.12.152
250 vador.skynet.be Hello 152.12-238-195.dialup.skynet.be [195.238.12.152], pleased to
meet you
mail from: vilvens.claude@skynet.be
250 2.1.0 vilvens.claude@skynet.be... Sender ok
rcpt to: civlvens@prov-liege.be
250 2.1.5 civlvens@prov-liege.be... Recipient ok
data
354 Please start mail input.
Bonjour !
Essai SMTP avec Telnet
.
250 Mail queued for delivery.
quit
221 Closing connection. Good bye.
```

On peut vérifier dans la boîte aux lettres du destinataire que le message est bien arrivé :



Il faudra alors lire ce message et interpréter ses headers - mais ceci est du ressort d'un autre protocole (POP3 ou IMAP) : patience donc ...

## 5. Un MTA classique : Sendmail



### 5.1 Une brève présentation

Sendmail est ce que l'on a coutume de désigner sous le nom de "serveur de messagerie", c'est-à-dire qu'il s'agit d'un **MTA** : son rôle est donc d'envoyer les messages transmis par un **MUA**, de les stocker temporairement si nécessaire et surtout de les acheminer vers le **MTA** du destinataire.

Sendmail a été créé par Eric Allman en 1980 à Berkeley et reste l'un des plus populaires sur Internet (<http://www.sendmail.org>). Le logiciel a connu de nombreuses versions, parfois parallèles. Les dernières en date ont une version de niveau 8.14.\* au moins. Sendmail est reconnu comme un serveur puissant, mais compliqué à cause précisément de ses nombreuses configurations possibles et de la syntaxe lourde de son fichier de configuration (on y utilise, par exemple, des macros certes puissantes mais rendant la lecture obscure ...).

Pour notre part à l'InPrES, nous utiliserons un serveur Sendmail installé sur une machine Linux appelé u2 (plus précisément u2.wildness.loc). Sendmail a été installé à l'aide du kit Jussieu, du nom de l'université française dont les scientifiques créateurs du kit proviennent; ce kit permet de générer un fichier de configuration très rapidement et très facilement.

### 5.2 Le fonctionnement de Sendmail

Sendmail fonctionne en tant que démon. Il scrute le port 25 en attente d'une nouvelle connexion. Lorsque celle-ci se produit, un processus fils est créé et se voit chargé de la gestion de celle-ci tandis que le processus père peut se remettre en attente.

C'est donc le processus fils qui reçoit le message. Il le place dans une file d'attente. Ensuite :

- ◆ si le message peut être immédiatement envoyé, il est envoyé par le réseau au destinataire (Sendmail est donc un MTA avec un MDA intégré); le message est ensuite retiré de la file;
- ◆ si le message ne peut pas être immédiatement envoyé (par exemple parce que la machine hôte n'est pas connectée à Internet à ce moment), le processus fils se termine; le message reste donc dans la file.

Ceci implique que le processus père principal de Sendmail doit lancer, à intervalles réguliers, un processus fils chargé de parcourir la file pour tenter d'envoyer les messages qui s'y trouvent encore.

### 5.3 Les principaux fichiers utilisés par Sendmail

On s'en doute dans un monde Unix, le fichier de base pour Sendmail est son fichier de configuration **/etc/sendmail.cf**. Celui-ci est relu à chaque démarrage et il a donc été taillé pour que l'opération de décodage par Sendmail soit la plus rapide possible. Evidemment, cela a impliqué une syntaxe délicate à interpréter pour un humain (normal) : c'est le langage **M4**.

Le fichier **/etc/aliases** mémorise les différents comptes sur sendmail, avec leur identification. Deux alias doivent impérativement exister :

```
postmaster:    root
mailer-daemon: postmaster
```

Le répertoire **/var/spool/mqueue** est évidemment celui où sont stockés les messages qui n'ont pu être acheminés.

---

Le fichier **/etc/password** est bien connu et et utilisé par Sendmail pour déterminer les comptes locaux. Par conséquent, tout utilisateur connu est automatiquement possesseur d'une boîte aux lettres sur la machine.

Enfin, les utilisateurs peuvent définir dans leur répertoire des fichiers **.forward** permettant de spécifier des règles de transfert des courriers entrants.

#### 5.4 Un aperçu du fichier de configuration

Inutile de dire que nous n'avons pas l'intention de décortiquer tout sendmail.cf. Nous allons donc simplement en observer les grandes lignes.

**1)** Une commande fondamentale est la commande **D** (on était prévenu, on fait dans le court ...). Elle permet de définir une **macro** (on pourrait aussi dire une variable) selon la syntaxe :

**D<nom de la macro><valeur affectée à la macro>**

Par exemple,

```
...
#
# Le host
#
DHu2

#
# Le domaine :
# - la variable M contient le nom de domaine officiel
# - la classe M (différente de la variable M) contient les noms abrégés
#
DMwildness.loc
...
```

définit les macros H et M, avec pour valeurs respectives "u2" et "wildness.loc". La valeur d'une macro est invoquée par son nom précédé de '\$'. Ainsi :

```
Dj$H.$M
```

définit la macro j dont la valeur est le nom complet "u2.wildness.loc".

Toute une série de macros sont des macros standards de sendmail et doivent être définies. Citons :

V	sert d'identification de la version de ce sendmail.cf
m	domaine
j	host+domaine
e	message de bienvenue SMTP
l	format de la ligne "From"

Par exemple : DVjtpda-5.3.3

L'écriture des macros peut être plus complexe grâce à la possibilité d'insérer des conditions (testant l'existence d'une macro) dans les commandes de définitions selon les syntaxes :

```
$?<macro><texteV>$.
$?<macro><texteV>$|<texteF>$.
```

Par exemple : Dq\${\$?x (\$x)\$}.

Décryptons : la macro q comporte la valeur de la macro g suivie, si la macro x a été définie, de la valeur de cette macro x entre parenthèses. Palpitant, non ? ;-)

**2)** Une classe de Sendmail n'a rien à voir avec la P.O.O. – il s'agit en fait d'une liste de valeurs à même valeur sémantique. Les commandes permettant de les définir sont **C** (si l'on cite les valeurs) et **F** (si les valeurs proviennent d'un fichier). Par exemple :

CM wildness.loc

CX LOCAL INTERNE EXTERNE

Une convention est que les noms en minuscules sont réservés aux noms internes à Sendmail – les classes utilisateur ont donc toujours un nom en majuscules.

**3)** Une option de l'environnement Sendmail se définit par la commande **O** selon la syntaxe :

O<nom option><valeur>

Par exemple :

OA/etc/mail/aliases	chemin du fichier des alias
OPPostmaster	nom du PostMaster
OQ/var/spool/mqueue	répertoire de la file d'attente
Or5m	time out
Os	force Sendmail à placer les messages dans la file
Ox20	limite à partir de laquelle on met les courriers en attente
OX30	limite à partir de laquelle on refuse les connexions SMTP
Oh17	un message est considéré comme tournant en rond après 17 traitements (version 8)

**4)** Le format des en-têtes que Sendmail inclut dans les messages peut être défini très précisément par la commande **H** :

```
...
H?P?Return-Path: <$g>
HReceived: $?sfrom $s$?_($_)$.$.|$?_from ($_)$.|.
    by $j ($v/$V)$?r with $r$. id $i
    $?ufor $u$.; $b
H?D?Resent-Date: $a
H?D?Date: $a
```

```
H?F?Resent-From: $q
H?F?From: $q
H?x?Full-Name: $x
HSubject:
H?M?Resent-Message-Id: <$t.$i@$j>
H?M?Message-Id: <$t.$i@$j>
...
...
```

Bon amusement pour le décryptage (oui, je sais, c'est lâche ;-)...).

**5)** Il existe encore d'autres commandes, plus complexes, dont notamment **R**, celle des réécritures des adresses. Les règles ainsi définies permettent d'analyser les adresses électroniques provenant des MUA afin de les récrire sous la forme requise par les programmes de transfert du courrier. Mais, là, cela se complique :-( ...

Maintenant qu'un MTA existe et est en place, voyons comment nous servir de lui pour réaliser une application qui envoie un message. On s'en doute, quelques packages complémentaires seront nécessaires ...

## **6. Les e-mails et des packages javax en plus ...**

On désigne donc sous le terme global de **JavaMail** un ensemble d'APIs, présentées sous forme de classes et d'interfaces, dévolues à la gestion du courrier électronique dans toute sa généralité. JavaMail fait partie du **J2EE** et est à sa version 1.2. Les APIs standards généraux sont en fait répartis dans 4 packages qui sont :

- |                     |   |
|---------------------|---|
| javax.mail          | : Les classes de base qui permettent de réaliser les opérations de base des mails         |
| javax.mail.event    | : Événements associés aux mails (du type arrivée de courrier) et listeners correspondants |
| javax.mail.internet | : Les classes spécifiques aux mails Internet  |
| javax.mail.search   | : Les fonctionnalités de recherche dans les mails   |

Comme toujours en Java, l'ensemble de ces packages propose d'abord une structure de programmation indépendante de la plate-forme utilisée et indépendante des protocoles réseaux sous-jacents. Il s'agit donc d'un ensemble de **composants génériques**, donc d'interfaces et de classes abstraites.

La situation est donc assez similaire à celle des APIs de cryptographie et il appartient donc à des *providers* de fournir les **implémentations** correspondantes – à nouveau, Sun est le provider de base. Les implémentations des différents providers sont décrites dans les fichiers javamail.providers et javamail.default.providers.

Ces implémentations sont sensées comporter

- ♦ un protocole de transfert [**transport**] de courrier électronique - habituellement, il s'agit de SMTP;
- ♦ un protocole de stockage et de remise du courrier [**store**] - habituellement, il s'agit de POP3 ou IMAP.

On trouve dans le fichier mail.jar un fichier javamail.default.providers :

---

### **javamail.default.providers**

```
# JavaMail IMAP provider Sun Microsystems, Inc  
protocol=imap; type=store; class=com.sun.mail.imap.IMAPStore; vendor=Sun  
Microsystems, Inc;  
# JavaMail SMTP provider Sun Microsystems, Inc  
protocol=smtp; type=transport; class=com.sun.mail.smtp.SMTPTransport; vendor=Sun  
Microsystems, Inc;  
# JavaMail POP3 provider Sun Microsystems, Inc  
protocol=pop3; type=store; class=com.sun.mail.pop3.POP3Store; vendor=Sun  
Microsystems, Inc;
```

Les classes qui implémentent les trois protocoles que nous utiliserons sont donc celles se trouvant dans le package com.sun.mail.

Cependant, un autre groupe d'APIs est nécessaire à la programmation des JavaMail : le JavaBeans Activation Framework (JAF) ...

## **7. L'évolution des JavaBeans et le JAF**

Sun a marqué très nettement sa volonté de faire évoluer les JavaBeans<sup>1</sup> en créant un nouvelle spécification pour le modèle des beans. Le nom de code du projet est **Glasgow**. L'objectif annoncé est de permettre une intégration plus aisée des beans dans l'environnement d'exécution, qu'il s'agisse d'un bureau, d'un browser, d'un traitement de textes ou d'autre chose ... Pour cela, Glasgow ajoute au modèle des Beans trois éléments, qui sont :

### **1) l'extension de la notion de container** (Extensible Runtime Containment and Services Protocol)

Il s'agit pour un objet container de fournir à un bean un environnement d'exécution avec des moyens standards permettant de découvrir et d'utiliser les fonctionnalités que le bean va apporter en plus à son container.

### **2) le JAF (JavaBeans Activation Framework)**

Face à des données quelconques, il s'agit de fournir ici des services standards permettant :

- ◆ de déterminer le type de ces données;
- ◆ d'en découvrir les fonctionnalités disponibles;
- ◆ d'instancier le composant approprié pour un type d'opération donnée.

L'exemple classique :

- ◆ un browser voit arriver des données;
- ◆ il faut d'abord qu'il se rende compte qu'il s'agit, par exemple, d'une séquence vidéo MP3;
- ◆ ensuite, il faut qu'il trouve une classe qui est capable de manipuler des MP3;
- ◆ et finalement, il lui reste à instancier un objet qui va effectivement réaliser le travail.

### **3) un couper-coller étendu pour les JFC**

L'idée est de permettre des couper/coller entre des applications graphiques natives et des applications purement Java.

---

<sup>1</sup> voir "Langage Java (I) : Programmation de base" – chapitre IX.

L'objectif global est donc bien de fournir aux développeurs *des moyens standards de diffuser leurs beans sans devoir se préoccuper des environnements sur lesquels ces composants devront être utilisés*. Un certain nombre d'interfaces et de classes du package **javax.activation** réalisent cet objectif ambitieux. Ainsi, pour nous limiter à ce qui nous intéresse directement, l'interface **DataSource** représente une collection quelconque de données qui seront accédées comme des InputStreams et OutputStreams. Ses méthodes sont assez prévisibles :

```
public java.lang.String getContentType()  
    qui renvoie le type MIME de la donnée  
  
public java.lang.String getName()  
    qui renvoie le nom de l'objet qui représente la donnée  
  
public java.io.InputStream getInputStream() throws java.io.IOException  
public java.io.OutputStream getOutputStream() throws java.io.IOException  
    qui fournit les flux représentant les données en entrée ou en sortie, si c'est possible  
(une exception est lancée dans le cas contraire).
```

Revenons-en à présent à nos e-mails ... Et précisons encore que le contexte de développement est :

- ◆ un serveur de courrier Sendmail installé sur la machine Linux u2 dans le domaine wildness.loc;
- ◆ une IDE de développement qui est Netbeans 6.\*.

## 8. Une session mail

Un objet instanciant la classe **Session**, qui hérite directement d'Object et qui est définie dans le package javax.mail, est absolument nécessaire pour pouvoir envoyer ou recevoir des mails. En effet, il comporte toutes les caractéristiques, notamment les valeurs par défaut, auxquelles les APIs de mail vont se référer. A priori, l'objet session est partageable entre les diverses applications tournant sur la machine virtuelle considérée; il est cependant possible de modifier ce partage implicite.

Comme on peut s'y attendre d'après le chapitre consacré à la cryptographie, ces caractéristiques sont mémorisées dans un objet instanciant la classe **Properties** (package java.util) qui est une HashTable spécialisée. Les entrées auxquelles les APIs JavaMails feront référence sont :

mail.transport.protocol	protocole de transport pour la session
mail.store.protocol	protocole de store pour la session
mail.host	serveur par défaut pour les deux protocoles transport et store
mail.<protocole>.host	serveur pour un protocole spécifique – donc, par exemple, mail.smtp.host
mail.user	nom d'utilisateur par défaut pour les deux protocoles transport et store
mail.<protocole>.user	nom d'utilisateur pour un protocole spécifique – donc, par exemple, mail.smtp.user
mail.from	adresse pour la réponse à l'utilisateur courant
mail.debug	true ou false selon le mode utilisé

On peut obtenir une session initialisée avec les valeurs par défaut au moyen de la méthode factory polymorphe :

```
public static Session getDefaultInstance(java.util.Properties props)
public static Session getDefaultInstance(java.util.Properties props, Authenticator authenticator)
```

La session est effectivement créée si elle n'existe pas encore, sinon l'application obtient l'accès à la session existante (du moins en l'absence d'authentificateur). Quant aux paramètres :

- ◆ le premier permet au programmeur de spécifier les valeurs de certains propriétés pour lesquelles les valeurs par défaut ne peuvent probablement pas convenir : mail.host, mail.user et mail.from, éventuellement mail.store.protocol et mail.transport.protocol;
- ◆ le second paramètre éventuel permet de vérifier, dans un but de sécurité, que l'accès à la session est permis (on peut envisager de prendre ses précautions pour protéger, par exemple, un mot de passe). Si ce paramètre existe, il est associé à la nouvelle session créée; tout nouvel appel à la méthode vérifiera que l'objet authentificateur est le même : si ce n'est pas le cas, l'accès sera refusé.

Créons une simple session avec ses valeurs par défaut. Nous utiliserons la liste de propriétés du système sur lequel nous travaillons, liste que l'on obtient par la méthode de System

```
public static Properties getProperties()
```

(comme vu en son temps au paragraphe 3 du chapitre VI du volume I). Nous ajouterons simplement à cette liste, au moyen de la méthode put (héritée de Dictionary) une entrée de plus ("mail.smtp.host"), qui représentera le nom de la machine serveur de mails (soit ici u2.wildness.loc ou u2 en abrégé) :

### JMailSimplePart.java

```
/*
 * JMailSimplePart.java
 * Created on 18 mars 2002, 12:51
 */

import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

import java.util.*;

/**
 * @author VILVENS
 * @version
 */

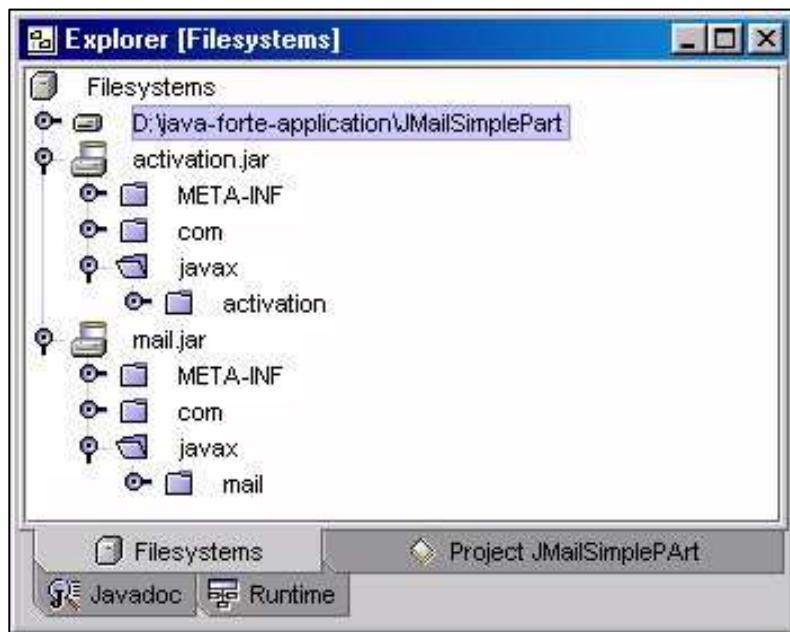
public class JMailSimplePart
{
    static String host = "u2.wildness.loc";
```

```
/** Creates new JMailSimplePart */
public JMailSimplePart() {}

/**
 * @param args the command line arguments
 */
public static void main (String args[])
{
    Properties prop = System.getProperties();
    prop.put("mail.smtp.host", host);

    System.out.println("Création d'une session mail");
    Session sess = Session.getDefaultInstance(prop, null);
    prop.list(System.out);
}
}
```

Bien sûr, on a monté dans le projet les fichiers jar nécessaires, soit activation.jar et mail.jar :



Le résultat de l'exécution sur la console est (c'est un peu long, mais exhaustif) :

```
Création d'une session mail
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
sun.boot.library.path=D:\JDK16~1.1_0\jre\bin
java.vm.version=1.6.0_13-b03
java.vm.vendor=Sun Microsystems Inc.
java.vendor.url=http://java.sun.com/
path.separator=;
java.vm.name=Java HotSpot(TM) Client VM
file.encoding.pkg=sun.io
java.vm.specification.name=Java Virtual Machine Specification
```

```
user.dir=c:\forte4j\bin
java.runtime.version=1.6.0_13-b03
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
os.arch=x86
java.io.tmpdir=C:\DOCUME~1\VILVEN~1.ADM\LOCALS~1\Temp\
line.separator=

java.vm.specification.vendor=Sun Microsystems Inc.
java.awt.fonts=
os.name=Windows 2000
java.library.path=D:\JDK16~1.1_0\jre\bin;;C:\WINNT\Sys...
java.specification.name=Java Platform API Specification
java.class.version=47.0
os.version=5.0
user.home=C:\Documents and Settings\VILVENS.ADM...
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding=Cp1252
java.specification.version=1.6
user.name=VILVENS
java.class.path=c:\forte4j\bin\java-application\syste...
java.vm.specification.version=1.0
java.home=D:\JDK16~1.1_0\jre
user.language=fr
java.specification.vendor=Sun Microsystems Inc.
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode
java.version=1.6.0_10
java.ext.dirs=D:\JDK16~1.1_0\jre\lib\ext
sun.boot.class.path=D:\JDK16~1.1_0\jre\lib\rt.jar;D:\JDK1...
java.vendor=Sun Microsystems Inc.
file.separator=\
java.vendor.url.bug=http://java.sun.com/cgi-bin/bugreport...
mail.smtp.host=u2.wildness.loc
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
user.region=FR
sun.cpu.isalist=pentium i486 i386
```

Venons-en maintenant à la confection d'un message. Un message comporte classiquement un header (qui comporte un certain nombre d'**attributs**) et un **contenu** [body part]. Donc, en général ...

## **9. L'interface Part**

L'interface **Part** (défini sans héritage dans le package javax.mail) représente tout ce qui caractérise la notion générale de message, soit

**1)** des **attributs** divers, communs aux systèmes de messageries les plus courants, dont le champ Content-Type qui peut être obtenu au moyen de la méthode

```
public java.lang.String getContentType() throws MessagingException
```

Le type est renvoyé selon la codification MIME (on obtient null si le type ne peut être déterminé). D'ailleurs, la méthode

```
public boolean isMimeType(java.lang.String mimeType) throws MessagingException
```

permet de vérifier que l'on a affaire à un type standard, donc pas un type propriétaire.

**2)** un **contenu**, dont le type sera décrit au moyen de la codification MIME. Ce contenu peut être obtenu sous différentes formes :

♦ un **DataHandler** (classe définie dans javax.activation), obtenu par la méthode

```
public javax.activation.DataHandler getDataHandler() throws MessagingException
```

- un tel objet permet de déterminer les opérations possibles sur le contenu et de manipuler effectivement les données contenues en instanciant les objets appropriés; on a coutume de dire qu'un tel objet "encapsule" les données véritables; quant à l'aspect "découverte des opérations", il fait clairement allusion aux JavaBeans ...

♦ la méthode de DataHandler

**DataSource** **getDataSource()**

fournit un objet **DataSource** qui dissimule les ressources à attacher;

♦ un **flux d'entrée**, obtenu au moyen de la méthode prévisible de DataSource

```
public java.io.InputStream getInputStream()
```

```
throws java.io.IOException, MessagingException
```

qui ne fait d'ailleurs qu'appeler la méthode du même nom du DataHandler évoqué plus haut;

♦ un **objet Java**, instance d'une classe quelconque et qui pourra fournir son contenu; encore une fois, le DataHandler travaille en tâche de fond pour fournir cet objet. Cette dernière opération est possible avec la méthode :

```
public java.lang.Object getContent() throws java.io.IOException, MessagingException.
```

## **10. Les classes messages**

### **10.1 La classe de base**

La classe abstraite attendue **Message** (bien sûr définie dans javax.mail) est dérivée directement d'Object et implémente l'interface Part. Elle comporte donc un ensemble d'attributs et un contenu. Elle implémente ainsi :

```
public void setText(java.lang.String text) throws MessagingException
```

de l'interface Part.

---

En fait, elle ajoute des attributs supplémentaires à ceux prévus par Part :

- ◆ les adresses de l'expéditeur et du destinataire;
- ◆ le sujet;
- ◆ les dates de réception et de fin.

Elle propose donc aussi les méthodes prévisibles :

```
public abstract void setFrom(Address address) throws MessagingException
public void setRecipient(Message.RecipientType type, Address address)
    throws MessagingException
public abstract void setSubject(java.lang.String subject) throws MessagingException
public abstract void setSentDate(java.util.Date date) throws MessagingException
```

où l'on voit apparaître deux classes :

- ◆ la classe **Address** qui apparaît pour l'expéditeur et le(s) destinataires(s) est une classe abstraite qui sert de base à tout ce qui est une adresse de mail. Ce sont ses filles, **InternetAddress** (du package javax.mail.internet) et **NewsAddress** (même package) qui sont effectivement utilisées (on devine dans quel contexte – elles sont conformes respectivement aux RFC 822 et 1036).
- ◆ la classe **RecipientType** est une *classe imbriquée* dans Message : elle représente le statut du destinataire, c'est-à-dire destinataire direct ou destinataire pour information ("carbon copy", caché ou non) et propose d'ailleurs des variables de classe très explicites :

```
public static final Message.RecipientType TO
public static final Message.RecipientType CC
public static final Message.RecipientType BCC
```

On peut donc voir un objet Message (non composite – on dit encore "**Simple part**", par opposition à "Multipart") sous cette forme :

<b>Header</b>	
	Attributs définis dans Part, dont Content-Type
	Attributs ajoutés par Message, dont From, To, Subject
<b>Corps [Content body]</b>	
	Objet DataHandler donnant accès au contenu du message

## 10.2 La classe e-mail classique

On l'a dit, cette classe Message est une classe abstraite. Une classe dérivée **MimeMessage** (définie dans le package javax.mail.internet) implémente effectivement les messages classiques utilisant la codification MIME. Elle implémente d'ailleurs l'interface **MimePart** qui définit l'interface d'une entité conforme aux spécifications MIME (comme définie dans la RFC2045, Section 2.4). Outre des méthodes comme

```
public String getContentType() throws MessagingException
public String getEncoding() throws MessagingException
public void setContentMD5(java.lang.String md5) throws MessagingException
```

elle ajoute des attributs complémentaires, comme des flags divers si le message est conservé dans un folder. Ceux-ci correspondent à une variable membre

**protected Flags flags**

où la classe Flags encapsule une série de flags qui instancient la *classe imbriquée Flag* (sans le 's'). Une série de variables de classes éclairent sur le rôle de ces flags :

```
public static final Flags.Flag ANSWERED
public static final Flags.Flag DELETED
public static final Flags.Flag DRAFT
public static final Flags.Flag FLAGGED
public static final Flags.Flag RECENT
public static final Flags.Flag SEEN
public static final Flags.Flag USER
```

On peut donc voir un objet MimeMessage sous la forme :

Header	
	Attributs définis dans Part, dont Content-Type
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
Corps [Content body]	
	Objet DataHandler donnant accès au contenu du message

Un constructeur de cette classe qui nous sera utile est :

```
public MimeMessage(Session session)
```

qui crée au sein de la session un message vide dont il s'agit alors de remplir les champs principaux au moyen des méthodes setXXX et addXXX. Restera alors à l'envoyer ...

## 11. L'envoi d'un message

La classe **Transport** est une classe dérivée de la classe **Service**, classe abstraite qui définit les services généraux de type transport et store. Elle porte bien son nom, puisqu'elle comporte notamment une méthode statique :

```
public static void send(Message msg) throws MessagingException
```

qui permet évidemment d'envoyer le message préalablement construit au sein de la session mail définie préalablement.

Le programme suivant envoie un message de l'utilisateur vilvens à l'utilisateur dalpont :

### JMailSimplePart.java

```
/*
 * JMailSimplePart.java
 */
```

```
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import java.util.*;

/**
 * @author VILVENS
 */
public class JMailSimplePart
{
    static String host = "u2.wildness.loc";

    public JMailSimplePart() {}

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        prop.put("mail.smtp.host", host);

        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
        prop.list(System.out);

        try
        {
            System.out.println("Création du message");
            String exp = "vilvens@u2.wildness.loc";
            String dest = "dalpont@u2.wildness.loc";
            String sujet = "Premier essai";
            String texte = "Essai avec JavaMail - Peux-tu me répondre ? - Merci - CV";
            MimeMessage msg = new MimeMessage (sess);
            msg.setFrom (new InternetAddress (exp));
            msg.setRecipient (Message.RecipientType.TO, new InternetAddress (dest));
            msg.setSubject(sujet);
            msg.setText (texte);

            System.out.println("Envoi du message");
            Transport.send(msg);
            System.out.println("Message envoyé");
        }
        catch (MessagingException e)
        {
            System.out.println("Erreur sur message : " + e.getMessage());
        }
        catch (Exception e)
        {
            System.out.println("Erreur sur message : " + e.getMessage());
        }
    }
}
```

Le résultat de l'exécution du programme donne :

**Création d'une session mail**

```
-- listing properties --
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...
...
mail.smtp.host=u2.wildness.loc
...
sun.cpu.isalist=pentium i486 i386
```

**Création du message**

**Envoi du message**

**Message envoyé**

Maintenant que nous savons envoyer des messages au moyen de SMTP tant de manière interactive qu'en programmation Java, sans doute serait-il temps de se demander comment lire ces messages qui ont atterri dans la boîte aux lettres ...

## 12. Le protocole POP3

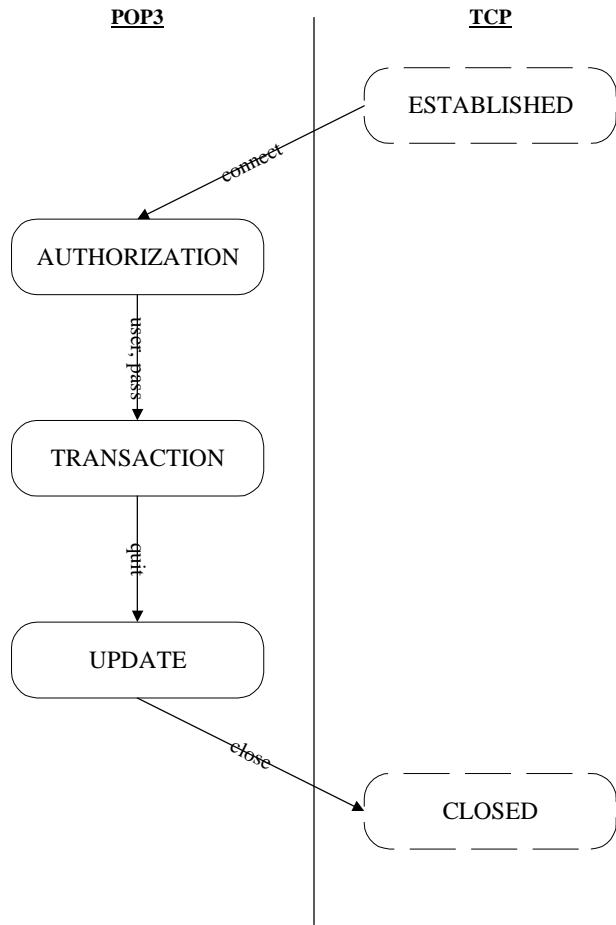
### 12.1 Une brève présentation

Le protocole **POP3** (Post Office Protocol version 3) est donc celui qui permet de récupérer les messages réceptionnés. Pour faire court :

<b>POP3</b>	
Nom complet :	<b>Post Office Protocol version 3</b>
Nature :	Protocole applicatif de récupération des messages électroniques
RFC :	1725
Port par défaut :	110
Protocole de transport :	TCP
Principales commandes :	user, pass, retr, dele

En fait, POP3 est un protocole à états, même si sa machine à états est sans commune mesure avec celle de TCP. On dispose de 3 états :

<b>état</b>	<b>state</b>	<b>signification</b>
AUTORISATION	AUTHORIZATION	La connexion TCP étant acceptée, le client doit à présent s'authentifier
TRANSACTION	TRANSACTION	La boîte aux lettres de l'utilisateur authentifié est verrouillée. Le client peut consulter, supprimer, etc.
MISE_A_JOUR	UPDATE	Toutes les modifications à la boîte aux lettres sont enregistrées; celle-ci est ensuite déverrouillée et la connexion TCP est fermée.



Les commandes de ce protocole ne sont pas très nombreuses. Elles sont toujours formées de 4 lettres, suivies éventuellement de paramètres.

## 12.2 Les commandes de base

Pour parvenir à lire un message en utilisant POP3, après s'être connecté au serveur, les commandes de base suivantes sont nécessaires :

Commande	Description
<b>user &lt;nom utilisateur&gt;</b>	Permet de s'identifier comme utilisateur
<b>pass &lt;mot de passe utilisateur&gt;</b>	Permet de confirmer son identité par introduction du mot de passe
<b>retr &lt;numéro de message&gt;</b>	Permet de récupérer le message du numéro précisé
<b>dele &lt;numéro de message&gt;</b>	Permet de supprimer le message du numéro précisé (plus exactement, le message est marqué logiquement pour la suppression)
<b>stat</b>	Donne le nombre et la taille cumulée des messages non lus
<b>quit</b>	A votre avis ? On passe dans l'état UPDATE et les messages marqués pour la suppression le sont alors effectivement.
<b>rset</b>	Permet d'annuler la suppression de tous les messages



Pop3 ne renvoie pas de code de retour. Il se contente, à chaque requête client, de donner une réponse sous forme d'un indicateur ("+OK" ou "-ERR") suivi d'un message.

### **12.3 Un exemple de transaction POP3 en ligne de commande**

A nouveau, point n'est besoin de programmer classiquement en TCP/IP pour tester ces commandes. Nous allons utiliser telnet pour nous connecter à un serveur de relevé de messagerie et envoyer à celui-ci nos commandes POP3. Concrètement, nous allons nous connecter depuis une machine Windows à un serveur courrier entrant de Skynet :

```
C:\VILVENS>telnet pop.skynet.be 110
```

On peut vérifier que la connexion est bien établie, avec une adresse IP attribuée dynamiquement :

```
C:\VILVENS>netstat -an
```

Connexions actives

Proto	Adresse locale	Adresse distante	État
TCP	0.0.0.0:1052	0.0.0.0:0	LISTENING
...			
TCP	127.0.0.1:1025	0.0.0.0:0	LISTENING
...			
TCP	192.168.2.2:137	0.0.0.0:0	LISTENING
...			
TCP	195.238.12.152:1052	149.174.38.142:80	ESTABLISHED
TCP	195.238.12.152:1053	149.174.38.142:80	ESTABLISHED
TCP	195.238.12.46:1064	195.238.3.116:110	ESTABLISHED
...			
UDP	192.168.2.2:137	*:*	
...			

On peut dès lors imaginer le dialogue suivant :

```
+OK Skynet POP3 Ready pop3pool1.skynet.be
user vilvens.claude
+OK USER vilvens.claude set
pass etpuisquoientcore
+OK You are so in
stat
+OK 2 2930
retr 1
+OK 1452 octets
...
```

### **13. La réception d'un message avec JavaMail**

Le message que nous avons envoyé par programmation ci-dessus (paragraphe 11), si il est bien arrivé, a atterri dans un folder. Pour aller le chercher, l'utilisateur va utiliser le protocole POP3 (ou IMAP). C'est le rôle de l'autre classe dérivée de **Service**, soit la classe **Store**, de fournir les outils permettant de gérer les folders et d'y récupérer les messages entrants.

Cette fois, il ne suffira pas d'utiliser une méthode de classe de Store et il faudra donc tout d'abord se procurer un tel objet au moyen de la méthode de l'objet Session préalablement obtenu :

```
public Store getStore(java.lang.String protocol) throws NoSuchProviderException
```

On aura compris que le paramètre permet d'exprimer le protocole utilisé. Une fois l'objet Store obtenu, il faut le mettre en rapport avec le serveur (POP3 en l'occurrence) visé. C'est le rôle de la méthode (héritée de Service) :

```
public void connect(java.lang.String host, java.lang.String user, java.lang.String password)  
throws MessagingException
```

On peut remarquer qu'une tentative de connexion à un service déjà connecté provoque une erreur. Si tout s'est bien passé, on peut alors se procurer une objet instanciant la classe **Folder** au moyen de la méthode factory de Store :

```
public abstract Folder getFolder(java.lang.String name) throws MessagingException
```

On constate qu'il faut fournir le nom du Folder. Le plus souvent (mais ce n'est pas sûr), il existe un folder de base appelé "INBOX". C'est le cas avec notre serveur Sendmail. Au départ, le folder est fermé : il faut donc l'ouvrir avec la méthode de Folder

```
public abstract void open(int mode) throws MessagingException
```

Le mode est spécifié en utilisant l'une des variables de classe de Folder :

```
public static final int READ_ONLY  
public static final int READ_WRITE
```

Après cette ouverture, on peut obtenir la liste des messages se trouvant dans le folder par :

```
public Message[] getMessages() throws MessagingException
```

comme d'ailleurs le nombre de nouveaux messages (c'est-à-dire ceux dont le flag RECENT est positionné) ou le nombre total de messages résidant dans le folder par :

```
public abstract int getMessageCount() throws MessagingException  
public int getNewMessageCount() throws MessagingException
```

On peut évidemment parcourir la liste comme toute liste et obtenir pour chaque message ce que l'on souhaite au moyen de méthodes de Message comme :

```
public abstract Address[] getFrom() throws MessagingException  
public abstract java.lang.String getSubject() throws MessagingException
```

et bien sûr la fameuse méthode de l'interface Part :

```
public java.lang.Object getContent() throws java.io.IOException, MessagingException.
```

Le programme suivant résume la démarche :

### JMailSimplePartRecv.java

```
/*  
  
 * JMailSimplePartRecv.java  
 * Created on 19 mars 2002, 16:36  
 */  
  
import javax.mail.*;  
import javax.mail.internet.*;  
import javax.activation.*;  
import java.util.*;  
import java.io.*;  
  
/**  
 * @author VILVENS  
 */  
  
public class JMailSimplePartRecv  
{  
    static String host = "u2.wildness.loc";  
    public JMailSimplePartRecv() {}  
  
    public static void main (String args[])  
    {  
        Properties prop = System.getProperties();  
        System.out.println("Création d'une session mail");  
        Session sess = Session.getDefaultInstance(prop, null);  
        prop.list(System.out);  
  
        try  
        {  
            String user = "vilvens";  
            String pwd = "beauGosse";  
            System.out.println("Obtention d'un objet store");  
            Store st = sess.getStore("pop3");  
            st.connect(host, user, pwd);  
  
            System.out.println("Obtention d'un objet folder");  
            Folder f = st.getFolder("INBOX");  
            f.open(Folder.READ_ONLY);  
            System.out.println("Obtention des messages");  
        }  
    }  
}
```

```
Message msg[] = f.getMessages();
System.out.println("Nombre de messages : " + f.getMessageCount());
System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());

System.out.println("Liste des messages : ");
for (int i=0; i<msg.length; i++)
{
    if (msg[i].isMimeType("text/plain"))
    {
        System.out.println("Expéditeur : " + msg[i].getFrom() [0]);
        System.out.println("Sujet = " + msg[i].getSubject());
        System.out.println("Texte : " + (String)msg[i].getContent());
    }
}
System.out.println("Fin des messages");
}

catch (NoSuchProviderException e)
{
    System.out.println("Erreur sur provider : " + e.getMessage());
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (IOException e)
{
    System.out.println("Erreur sur I/O : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur indéterminée : " + e.getMessage());
}
}
```

Le résultat, si l'utilisateur dalpont a envoyé un mail à l'utilisateur vilvens, sera :

#### Création d'une session mail

-- listing properties --

...

#### Obtention d'un objet store

#### Obtention d'un objet folder

#### Obtention des messages

Nombre de messages : 1

Nombre de nouveaux messages : 0

Liste des messages :

Expéditeur : dalpont@u2.wildness.loc

Sujet = reponse au test

Texte : Oui, j'ai bien recu votre mail...

Fin des messages

## 14. Du bon emploi des jeux de caractères

Soucieux de vérifier la portabilité de nos applications, nous allons évidemment sortir du contexte purement Sendmail qui est le nôtre jusqu'à présent et utiliser un deuxième serveur de messagerie, soit un serveur Microsoft Exchange situé, dans le contexte de l'Institut, sur mesepl.epl.prov-liege.be. Donc, dans le programme JMailSimplePart :

- ◆ l'objet host est "mesepl.epl.prov-liege.be";
- ◆ l'objet dest est "claude.vilvens@prov-liege.be";
- ◆ l'objet texte est " Il faudrait voir aussi comment on gère la liste des providers (comme avec Cryptix) – CV".

Le résultat est assez surprenant. En effet, le message reçu ressemble à ceci :

Ce message utilise un jeu de caractères qui n'est pas pris en charge par le service Messagerie Internet. Pour visualiser le contenu du message original, ouvrez le message joint. Si le texte n'est pas affiché correctement, enregistrez la pièce jointe sur le disque, et ouvrez-le avec une application capable d'afficher le jeu de caractères original.

<fichier.txt>

Autrement dit, le message reçu ne se présente pas comme un simple message simple part, mais comme un message multipart, c'est-à-dire avec un fichier (texte) attaché ! Dans ce dernier on trouve :

```
Received: from u2.wildness.loc (10.59.5.219 [10.59.5.219]) by mesepl.epl.prov-liege.be with
SMTP (Microsoft Exchange Internet Mail Service Version 5.5.2650.21)
      id G09DRQ9Y; Thu, 21 Mar 2002 12:32:51 +0100
Received: from ulyssse (ulyssse.thalassa.inpres.prov-liege.be [10.59.5.13])
      by u2.wildness.loc (8.11.3/jtpda-5.3.3) with ESMTP id g2LBuUt04450
      for <cvilvens@prov-liege.be>; Thu, 21 Mar 2002 12:56:35 +0100
Date: Thu, 21 Mar 2002 12:56:35 +0100
Message-ID: <425609.1016711321891.JavaMail.VILVENS@ulyssse>
From: vilvens@u2.wildness.loc
To: claude.vilvens@prov-liege.be
Subject: Essai vers Exchange
Mime-Version: 1.0
Content-Type: text/plain; charset=Cp1252
Content-Transfer-Encoding: quoted-printable
```

Il faudrait voir aussi comment on g=E8re la liste des providers (comme avec= Cryptix) – CV

En fait, le problème provient de l'emploi de caractères accentués, comme on peut le voir dans la phrase qui aurait du, normalement, constituer le seul message. Une analyse plus précise nous montre en effet que le champ Content-Type, s'il est bien du type MIME text/plain, emploi un jeu de caractères [charset] qui n'est pas celui qui est habituel dans nos contrées, à savoir ISO-8859-1. Une lecture plus attentive des Properties affichées au paragraphe 2 nous le montre d'ailleurs :

---

```
...
file.encoding=Cp1252
...
```

La solution est évidemment de fixer le jeu de caractères qui nous intéresse à l'émission, en modifiant le champs file.encoding dans la liste des Properties :

```
static String charset = "iso-8859-1";
...
Properties prop = System.getProperties();
...
prop.put("file.encoding", charset);
...
```

Un nouvel essai est, cette fois, parfaitement concluant.

## 15. Les headers d'un message reçu

SMTP utilise deux champs de header pour "tracer" le message qu'il véhicule, c'est-à-dire enregistrer toutes les informations qui permettront de reconstituer le chemin suivi :

**Return-Path:** <adresse>

Il s'agit de l'adresse de retour à utiliser en cas d'erreur.

**Received:** <description du chemin>

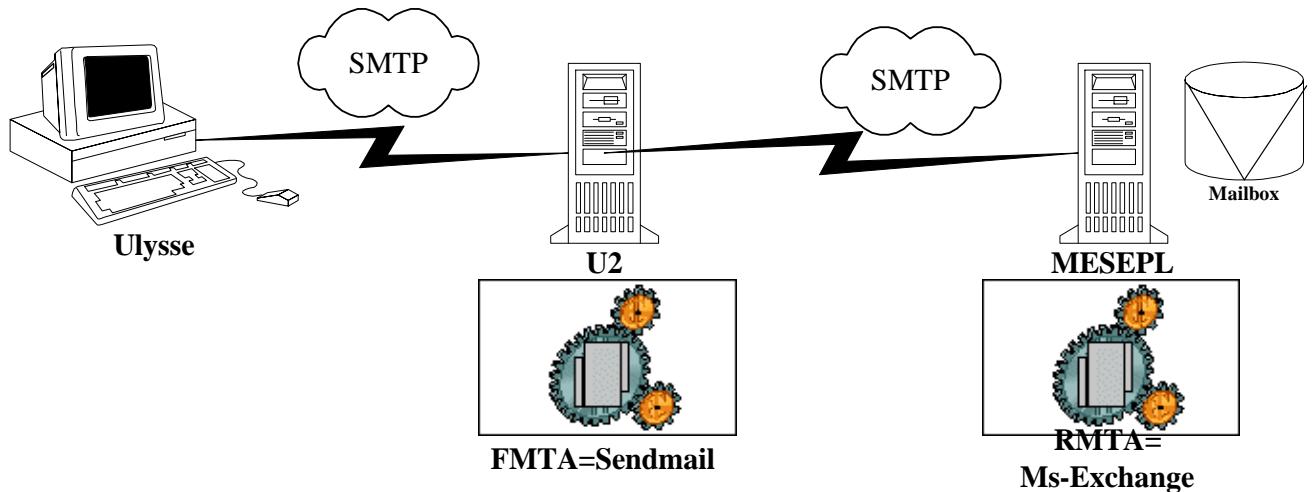
Un tel champ est ajouté par chaque agent qui intervient; le chemin suivi pour atteindre cet agent est décrit au moyen des clauses suivantes :

Clause	Signification
from	site émetteur
by	site récepteur
via	chemin physique
with	protocole utilisé
id	identificateur du message pour le récepteur
for	destinataire

Ainsi, dans l'exemple précédent réalisé dans la configuration de l'INPRES, on trouvait deux champs "received" :

**Received:** *from u2.wildness.loc (10.59.5.219 [10.59.5.219]) by mesepl.epl.prov-liege.be with*  
SMTP (Microsoft Exchange Internet Mail Service Version 5.5.2650.21)  
*id G09DRQ9Y; Thu, 21 Mar 2002 12:32:51 +0100*  
**Received:** *from ulyssse (ulyssse.thalassa.inpres.prov-liege.be [10.59.5.13])*  
*by u2.wildness.loc (8.11.3/jtpda-5.3.3) with ESMTP id g2LBuUt04450*  
*for <cvilvens@prov-liege.be>; Thu, 21 Mar 2002 12:56:35 +0100*

ce qui correspond à la situation suivante :



- ◆ Le deuxième champ received est le premier à avoir été créé<sup>2</sup> : le serveur Sendmail sur U2 a reçu le message provenant de la machine Ulysse et destiné à `civilvens@prov-liege.be`; nous avons donc ici un comportement classique de MTA de type forward : son MDA (dans le cas précis de Sendmail, c'est lui-même) détermine la cible (la machine mesepl) en utilisant le DNS.
- ◆ Le premier champ received est celui construit par le serveur Exchange sur Mesepl, qui trouve sur son hôte la boîte aux lettres du destinataire – la clause for a disparu.

## 16. L'examen des headers avec JavaMail

### 16.1 L'envoi et la réception

On s'en doute, les méthodes comme `getFrom()`, `getSubject()` ou `getContent()` ne sont que la pointe d'un iceberg correspondant à la possibilité de scanner tout l'entête d'un message (tout comme pour le protocole HTTP pour les servlets).

On ne sera donc pas étonné de découvrir l'existence de la méthode de Part :

```
public java.util.Enumeration getAllHeaders() throws MessagingException
```

Elle fournit une énumération d'objets **Header** (classe du package `javax.mail`, héritée d'`Object`), qui ne sont en fait que des paires (<nom de header>, <valeur de ce header>); les méthodes en sont donc bien prévisibles :

```
public java.lang.String getName()
public java.lang.String getValue()
```

Les parcourir relève du jardin d'enfants. Si donc nous complétons notre programme précédent, cette fois avec `vilvens.claude@skynet.be` qui écrit à `civilvens@prov-liege.be`, cela donne pour lire les messages avec les headers :

---

<sup>2</sup> manifestement, les horloges des machines intervenantes ne sont pas synchronisées ;-)

### JMailSimplePart.java (version skynet-prov-liege)

```
/*
 * JMailSimplePart.java
 * Created on 18 mars 2002, 12:51
 */
...
public class JMailSimplePart
{
    static String host = "relay.skynet.be";

    public JMailSimplePart() {}

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        prop.put("mail.smtp.host", host);

        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);

        try
        {
            System.out.println("Création du message");
            String exp = "vilvens.claude@skynet.be";
            String dest = "civilvens@prov-liege.be";
            String sujet = "T'as de beaux headers ;-)";
            String texte = "Coucou Mimi3 - c'est juste pour voir tes headers ;-) - CV";
            MimeMessage msg = new MimeMessage (sess);
            msg.setFrom (new InternetAddress (exp));
            msg.setRecipient (Message.RecipientType.TO, new InternetAddress (dest));
            msg.setSubject(sujet);
            msg.setText (texte);

            System.out.println("Envoi du message");
            Transport.send(msg);
            System.out.println("Message envoyé");
        }
        ...
    }
}
```

La réception du message se fera en lisant les headers en question.

---

<sup>3</sup> du calme : mon épouse se nomme Myriam, diminutif : Mimi – ouf !

### JMailSimplePartRecv.java (avec lecture des headers)

```
...
    System.out.println("Liste des messages : ");
    for (int i=0; i<msg.length; i++)
    {
        System.out.println("\n<nHeaders du message n°" + (i+1));
        Enumeration e = msg[i].getAllHeaders();
        Header h = (Header)e.nextElement();
        while (e.hasMoreElements())
        {
            System.out.println(h.getName() + " --> " + h.getValue());
            h = (Header)e.nextElement();
        }
        System.out.println("Texte : " + (String)msg[i].getContent());
    }
...
}
```

Au moment du test (2 minutes plus tard), quelqu'un avait envoyé un autre message ;-) ! :

Résultat :

Création d'une session mail

Obtention des messages

**Nombre de messages : 2**

Nombre de nouveaux messages : 0

Liste des messages :

Examinons posément les résultats.

#### 16.2 Les headers du message n°1

On a obtenu, pour le message non prévu :

**Return-Path** --> <lucien.nicolay@swing.be>

**Received** --> **from** kuzko.skynet.be (kuzko.skynet.be [195.238.3.136])

    by leia.skynet.be (8.11.6/8.11.6/Skynet-MAILSTORE-2.10) **with** ESMTP **id**

    g4KJG1T14744; Mon, 20 May 2002 21:16:01 +0200 (MET DST)

    (envelope-from <lucien.nicolay@swing.be>)

**Received** --> **from** excalibur.skynet.be (excalibur.skynet.be [195.238.3.90])

    by kuzko.skynet.be (8.11.6/8.11.6/Skynet-IN-2.27) **with** ESMTP **id** g4KJFri01977;

    Mon, 20 May 2002 21:15:53 +0200 (MET DST)

    (envelope-from <lucien.nicolay@swing.be>)

**Received** --> **from** lucien ([62.4.212.158])

    by excalibur.skynet.be (8.11.6/8.11.6/Skynet-OUT-2.19) **with** SMTP **id** g4KJFlH03254;

    Mon, 20 May 2002 21:15:47 +0200 (MET DST)

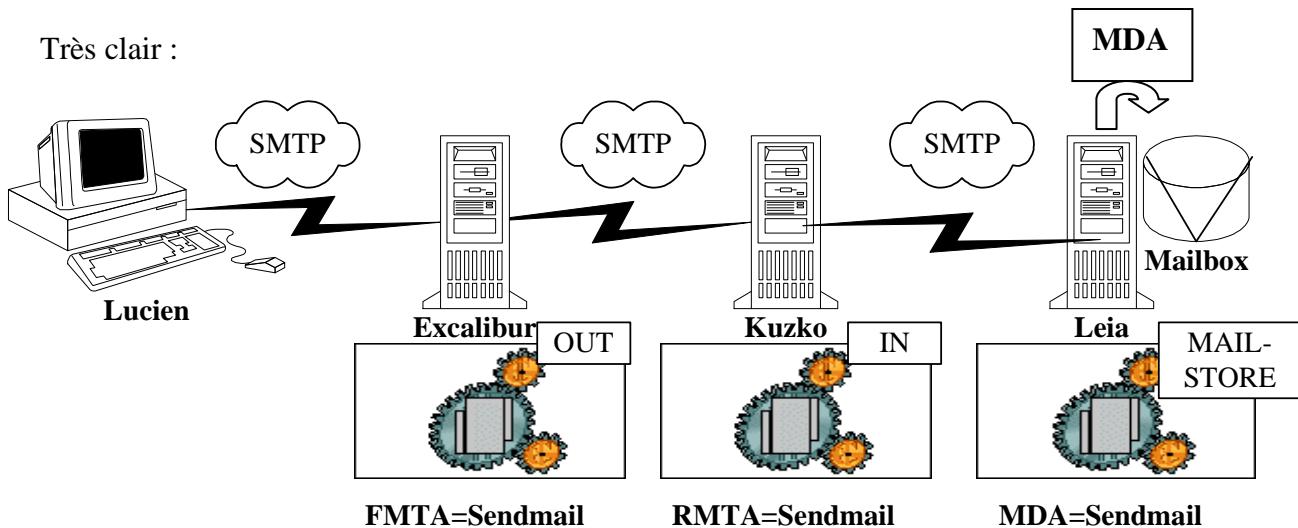
    (envelope-from <lucien.nicolay@swing.be>)

**Message-ID** --> <001701c20032\$ba909ec0\$9ed4043e@nicolay>

**From** --> "lucien.nicolay" <lucien.nicolay@swing.be>

**To** --> "Nicolay Claude" <nicolay.claude@swing.be>, <lecloux.jm@skynet.be>, <vilvens.claude@skynet.be>  
**References** --> <002501c1fec3\$3bc61680\$f6b1fea9@skynet>  
**Subject** --> Champion de tennis?  
**Date** --> Mon, 20 May 2002 21:15:38 +0200  
**MIME-Version** --> 1.0  
**Content-Type** --> text/plain;  
  charset="Windows-1252"  
**Content-Transfer-Encoding** --> 8bit  
**X-Priority** --> 3  
**X-MSMail-Priority** --> Normal  
**X-Mailer** --> Microsoft Outlook Express 5.00.2919.6600  
**X-MIMEOLE** --> Produced By Microsoft MimeOLE V5.00.2919.6600  
Texte : Bonsoir Maître – comment vont les mollusques ?

Très clair :



L'expéditeur est hébergé sur swing.be tandis que le destinataire l'est sur skynet.be – tout reste en fait dans la même maison ...

### 16.3 Les headers du message n°2

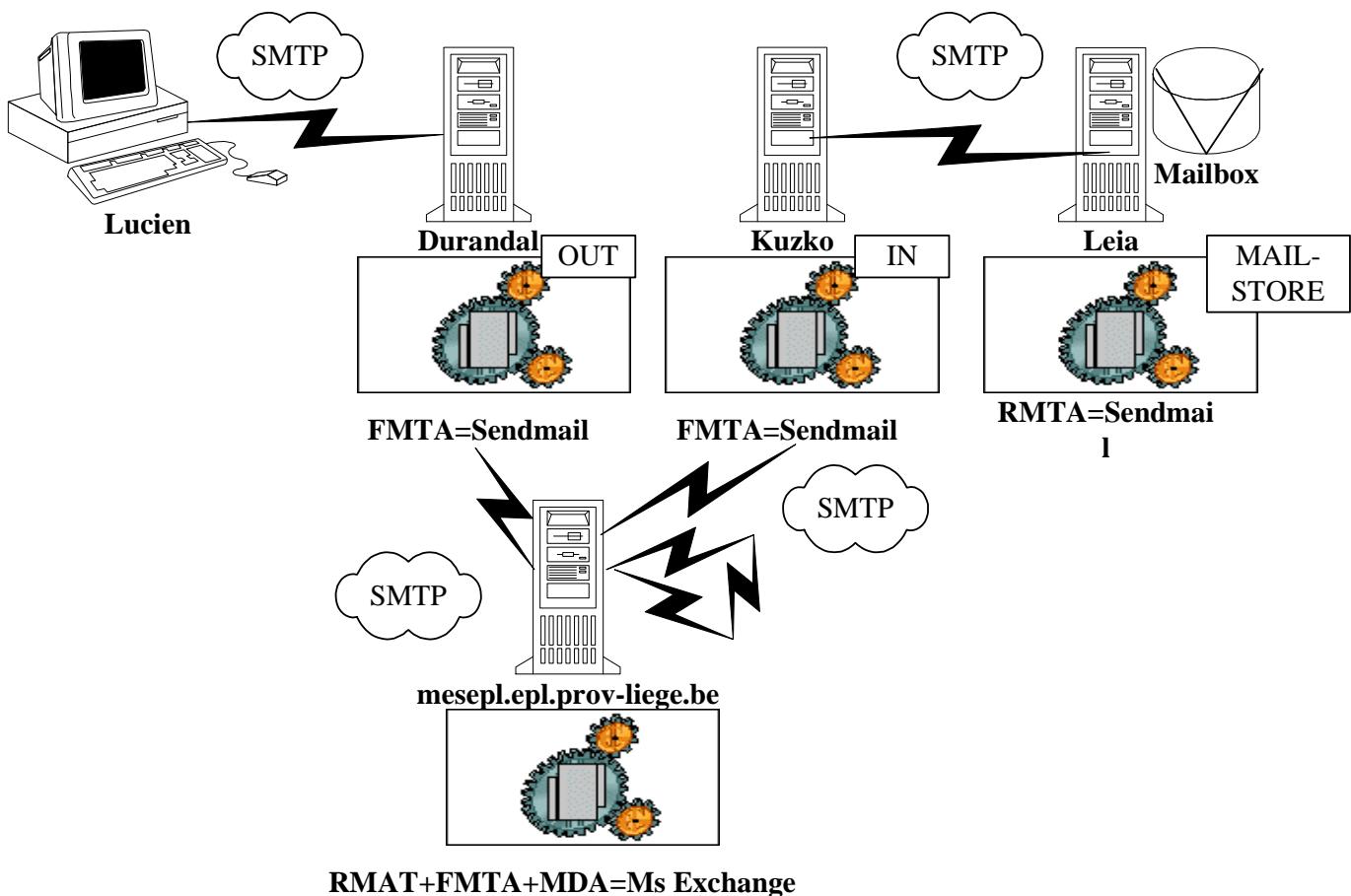
Le message que nous avons envoyé par programme va être traité d'une manière un peu particulière. En effet, il faut savoir que le serveur de courrier de prov-liege.be est configuré pour renvoyer systématiquement à vilvens.claude@skynet.be les messages reçus pour vilvens@prov-liege.be. Ceci se constate dans les champs de header :

**Return-Path** --> <vilvens.claude@skynet.be>  
**Received** --> from kuzko.skynet.be (kuzko.skynet.be [195.238.3.136])  
  by leia.skynet.be (8.11.6/8.11.6/Skynet-MAILSTORE-2.10) **with** ESMTP **id** g4KJRMT01090  
  for <xyz54321@leia.skynet.be>; Mon, 20 May 2002 21:27:22 +0200 (MET DST)  
  (envelope-from <vilvens.claude@skynet.be>)  
**Received** --> **from** mesepl.epl.prov-liege.be (vilvens.prov-liege.be [193.190.122.60])  
  by kuzko.skynet.be (8.11.6/8.11.6/Skynet-IN-2.27) **with** ESMTP **id** g4KJREi09341  
  for <vilvens.claude@skynet.be>; Mon, 20 May 2002 21:27:14 +0200 (MET DST)  
  (envelope-from <vilvens.claude@skynet.be>)  
**Received** --> **by** mesepl.epl.prov-liege.be **with** Internet Mail Service (5.5.2650.21)  
  **id** <K0SH3T7V>; Mon, 20 May 2002 21:28:16 +0200

**Received** --> *from* durendal.skynet.be ([195.238.3.91]) *by* mesopl.epl.prov-liege.be *with* SMTP (Microsoft Exchange Internet Mail Service Version 5.5.2650.21)  
*id* K0SH3T74; Mon, 20 May 2002 21:28:13 +0200  
**Received** --> *from* claude (195.238.20.245.dialup.skynet.be [195.238.20.245])  
*by* durendal.skynet.be (8.11.6/8.11.6/Skynet-OUT-2.19) *with* ESMTP *id* g4KJQhU16043  
*for* <civilvens@prov-liege.be>; Mon, 20 May 2002 21:26:43 +0200 (MET DST)  
(envelope-from <vilvens.claude@skynet.be>)  
**Date** --> Mon, 20 May 2002 21:26:43 +0200 (MET DST)  
**Message-ID** --> <8010433.1021923215920.JavaMail.Vilvens@claude>  
**From** --> vilvens.claude@skynet.be  
**To** --> civilvens@prov-liege.be  
**Subject** --> Premier essai  
**Mime-Version** --> 1.0  
**Content-Type** --> text/plain; charset=us-ascii  
**Content-Transfer-Encoding** --> 7bit  
Texte : Coucou Mimi - c'est juste pour voir tes headers ;-) - CV

Fin des messages

Donc, en fait, très clair aussi :



## 17. Les messages avec pièces attachées

n

<b>Header</b>	
	Attributs définis dans Part, dont <b>Content-Type vaut Multipart</b>
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
<b>Corps [Content body]</b>	Objet DataHandler <b>donnant accès à un objet Multipart</b>

La classe abstraite **Multipart**, directement dérivée d'Object et dont le type MIME est par défaut multipart/mixed, est un container d'objets BodyPart. Plus précisément, la classe Multipart possède une variable membre

```
protected java.util.Vector parts
```

dont le rôle est bien clair. La classe abstraite **BodyPart** implémente l'interface Part bien connu. A ce titre, elle présente une structure de message (au sens large), si bien que chaque objet BodyPart peut donner accès, par son DataHandler

- ◆ à des données simples
- ◆ à un autre objet Multipart

On peut donc ainsi créer des hiérarchies ... ce qui explique la présence d'une variable membre de Multipart :

```
protected Part parent
```

Schématiquement, les choses se passent donc, par exemple, comme ceci :

<b>Header</b>	
	Attributs définis dans Part, dont <b>Content-Type vaut Multipart</b>
	Attributs ajoutés par Message, dont From, To, Subject
	Attributs ajoutés par MimeMessage, dont les flags
<b>Corps [Content body]</b>	
	Objet DataHandler <b>donnant accès à un objet Multipart</b>
<b>objet Multipart</b>	
	<b>Objet BodyPart 1</b>
	<b>Header</b>
	Attributs définis dans Part
	<b>Corps</b>
	Objet DataHandler donnant accès au contenu
	<b>Objet BodyPart 2</b>
	<b>Header</b>
	Attributs définis dans Part, dont <b>Content-Type vaut Multipart</b>
	<b>Corps</b>
	Objet DataHandler <b>donnant accès à un objet Multipart</b>
	<b>Objet BodyPart 3</b>
	<b>Header</b>
	Attributs définis dans Part
	<b>Corps</b>
	Objet DataHandler donnant accès au contenu
<b>objet Multipart</b>	
	<b>Objet BodyPart 2.1</b>
	<b>Header</b>
	Attributs définis dans Part
	<b>Corps</b>
	Objet DataHandler donnant accès au contenu
	<b>Objet BodyPart 2.2</b>
	<b>Header</b>
	Attributs définis dans Part
	<b>Corps</b>
	Objet DataHandler donnant accès au contenu

## **18. L'envoi d'un e-mail avec attachment**

Supposons vouloir envoyer un message qui comporte :

- ◆ un texte introductif;
- ◆ un document Word – disons "BienvenueAInpres.doc" :

Bienvenue à l'INPRES !

C'est avec plaisir que nous vous envoyons les informations demandées, ainsi qu'une photo de l'Institut – la photo du Directeur n'est malheureusement plus disponible pour l'instant (trop de succès).



- ◆ une image au format JPG – disons LOGO-INP.jpg :



Concrètement, nous utiliserons les classes dérivées qui implémentent, en utilisant la codification MIME, respectivement MultiPart et BodyPart, soit **MimeMultiPart** et **MimeBodypart**; cette dernière classe implémente l'interface MimePart. La démarche, une fois les objets Session et Message créés, sera alors la suivante. Le contenu du message sera un objet instanciant MimeMultipart; celui-ci sera le container des trois BodyPart correspondant aux trois éléments du futur message; on insérera ces composantes au moyen de la méthode (déclarée dans Multipart) :

```
public void addBodyPart(BodyPart part) throws MessagingException
```

**1)** Le texte sera contenu dans un simple objet BodyPart, au moyen de la méthode **setText** déjà rencontrée pour un message SimplePart et déclarée dans l'interface Part :

```
String texteAcc = "Veuillez trouver ci-joint les documents demandés - CV";  
MimeBodyPart msgBP = new MimeBodyPart();  
msgBP.setText(texteAcc);
```

**2)** Un deuxième BodyPart va être chargé du transport du fichier Word. En fait :

2.1) On définit tout d'abord une source de données à partir de ce fichier Word. Pour cela, on utilisera un objet instanciant la classe **FileDataSource** qui implémente l'interface **DataSource**. L'un des deux constructeurs nous conviendra très bien :

```
public FileDataSource(java.lang.String name)
```

Donc, ici :

```
String nf = "d:\\notes-java-2001\\BienvenueAInpres.doc";  
msgBP = new MimeBodyPart();  
DataSource so = new FileDataSource (nf);
```

2.2) On assigne à l'objet BodyPart un DataHandler (pour rappel, un tel objet encapsule les données véritables et en permet une manipulation simple) spécialement créé pour gérer cette source de données; ceci se fait au moyen de la méthode de BodyPart :

```
public void setDataHandler(javax.activation.DataHandler dh) throws MessagingException
```

Donc, ici :

```
msgBP.setDataHandler (new DataHandler (so));
```

2.3) Accessoirement, on associe au BodyPart un nom de fichier, pris ici comme celui du fichier Word attaché, au moyen de la méthode

```
public void setFileName(java.lang.String filename) throws MessagingException
```

3) On procède de même avec l'image.

En résumé, cela nous donne la petite application suivante :

### JMailMultiplePart.java

```
/*  
 * Main.java  
 * Created on 27 mars 2002, 10:43  
 */  
  
import javax.mail.*;  
import javax.mail.internet.*;  
import javax.activation.*;  
import java.util.*;  
  
/**  
 * @author VILVENS  
 * @version  
 */  
  
public class JMailMultiplePart  
{  
    static String host = "u2.wildness.loc";
```

```
static String charset = "iso-8859-1";

public JMailMultiplePart() {}

public static void main (String args[])
{
    Properties prop = System.getProperties();
    prop.put("mail.smtp.host", host);
    prop.put("file.encoding", charset);

    System.out.println("Création d'une session mail");
    Session sess = Session.getDefaultInstance(prop/*, null*/);

    try
    {
        System.out.println("Création du message");
        String exp = "vilvens@u2.wildness.loc";
        String dest = "cvilvens@prov-liege.be";
        String sujet = "Essai d'attachement";
        String texteAcc = "Veuillez trouver ci-joint les documents demandés - CV";

        MimeMessage msg = new MimeMessage (sess);
        msg.setFrom (new InternetAddress (exp));
        msg.setRecipient (Message.RecipientType.TO, new InternetAddress (dest));
        msg.setSubject(sujet);

        System.out.println("Début construction du multipart");
        Multipart msgMP = new MimeMultipart();

        // 1ère composante : le texte d'accompagnement
        System.out.println("1ère composante");
        MimeBodyPart msgBP = new MimeBodyPart();
        msgBP.setText(texteAcc);
        msgMP.addBodyPart(msgBP);

        // 2ème composante : le fichier Word
        System.out.println("2ème composante");
        String nf = "d:\\notes-java-2001\\BienvenueAInpres.doc";
        msgBP = new MimeBodyPart();
        DataSource so = new FileDataSource (nf);
        msgBP.setDataHandler (new DataHandler (so));
        msgBP.setFileName(nf);
        msgMP.addBodyPart(msgBP);

        // 3ème composante : l'image
        System.out.println("3ème composante");
        nf = "d:\\notes-java-2001\\logo-INPRES.bmp";
        msgBP = new MimeBodyPart();
        so = new FileDataSource (nf);
        msgBP.setDataHandler (new DataHandler (so));
```

```
msgBP.setFileName(nf);
msgMP.addBodyPart(msgBP);

msg.setContent(msgMP);
System.out.println("Envoi du message");
Transport.send(msg);
System.out.println("Message envoyé");
}

catch (AddressException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
}
```

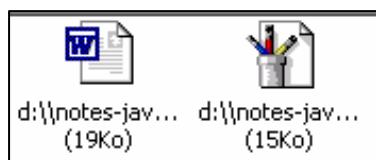
Le résultat dans la fenêtre d'output sera simplement :

Création d'une session mail  
Création du message  
Début construction du multipart  
**1ère composante**  
**2ème composante**  
**3ème composante**  
Envoi du message  
Message envoyé

Allons à présent du côté du destinataire, qui utilise Outlook :



Le message est donc bien arrivé. Dans la fenêtre, on observe en bas les pièces attachées :



## **19. La réception d'un e-mail avec pièces attachées**

Du point de vue du destinataire, tout commence classiquement par l'accès au folder et au scan des différents messages. Il s'agit ensuite de savoir si un message est d'une seule pièce ou, au contraire, comporte des pièces attachées. La méthode `isMimeType()` convient évidemment parfaitement à ce travail.

Ensuite, il s'agit en priorité de récupérer l'objet `MultiPart`, puisqu'il contient les différentes composantes du message. De par l'implémentation de l'interface `Part`, on peut utiliser la méthode `getContent()` pour cela. La méthode (déclarée dans l'interface `MultiPart`)

`public int getCount() throws MessagingException`

permet de savoir combien d'objets `BodyPart` sont inclus tandis que

`public BodyPart getBodyPart(int index) throws MessagingException`

permet d'extraire le `BodyPart` dont le numéro d'ordre est précisé (le premier a le numéro 0).

Chacun de ces objets nécessite un traitement spécifique, puisque différents types sont possibles. Le cas le plus simple est celui du texte accompagnateur (qui déclare un type MIME "text/plain"). Pour les autres, il s'agit de voir, au moyen de la méthode de `Part` :

`public java.lang.String getDisposition() throws MessagingException`

si l'on affaire à une pièce attachée ("ATTACHMENT") ou à une donnée distante ("INLINE"), ce que l'on tester en utilisant les deux constantes de classe :

`public static final java.lang.String ATTACHMENT`  
`public static final java.lang.String INLINE`

La RFC 2183 précise cette "disposition" peut indifféremment utiliser des minuscules ou des majuscules.

Si l'on a détecté une pièce attachée, on peut alors se souvenir que le `BodyPart` correspondant, implémentant `Part`, doit donc, par `DataHandler` interposé, fournir un flux de lecture permettant d'acquérir les bytes constituant le fichier attaché : la méthode

`public java.io.InputStream getInputStream()`  
`throws java.io.IOException, MessagingException`

Il nous reste dès lors à recopier ces bytes dans un fichier local. Comme le nombre de bytes est inconnu, le plus simple est d'utiliser un `ByteArrayOutputStream` pour la récupération, puis d'écrire celui-ci dans un `FileOutputStream`. Il est assez logique de donner à ce fichier local le même nom que celui du fichier attaché, nom que l'on récupérera au moyen de la méthode (toujours issue de `Part`) :

`public java.lang.String getFileName() throws MessagingException`

L'application de réception des messages devient :

### JMailMultiplePartRecv.java

```
/*
 * Main.java
 * Created on 27 mars 2002, 17:48
 */

import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;
import java.util.*;
import java.io.*;

/**
 * @author Vilvens
 * @version
 */

public class JMailMultiplePartRecv
{
    static String host = "pop.skynet.be"; //"2.wildness.loc";

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
        try
        {
            String user = "xyz54321";
            String pwd = "abc123";
            System.out.println("Obtention d'un objet store");
            Store st = sess.getStore("pop3");
            st.connect(host, user, pwd);

            System.out.println("Obtention d'un objet folder");
            Folder f = st.getFolder("INBOX");
            f.open(Folder.READ_ONLY);
            System.out.println("Obtention des messages");
            Message msg[] = f.getMessages();
            System.out.println("Nombre de messages : " + f.getMessageCount());
            System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());

            System.out.println("Liste des messages : ");
            for (int i=0; i<msg.length; i++)
            {
                System.out.println("Message n° " + i);
                System.out.println("Expéditeur : " + msg[i].getFrom() [0]);
                System.out.println("Sujet = " + msg[i].getSubject());
            }
        }
    }
}
```

```
System.out.println("Date : " + msg[i].getSentDate());

// Récupération du conteneur Multipart
Multipart msgMP = (Multipart)msg[i].getContent();
int np = msgMP.getCount();
System.out.println("-- Nombre de composantes = " + np);

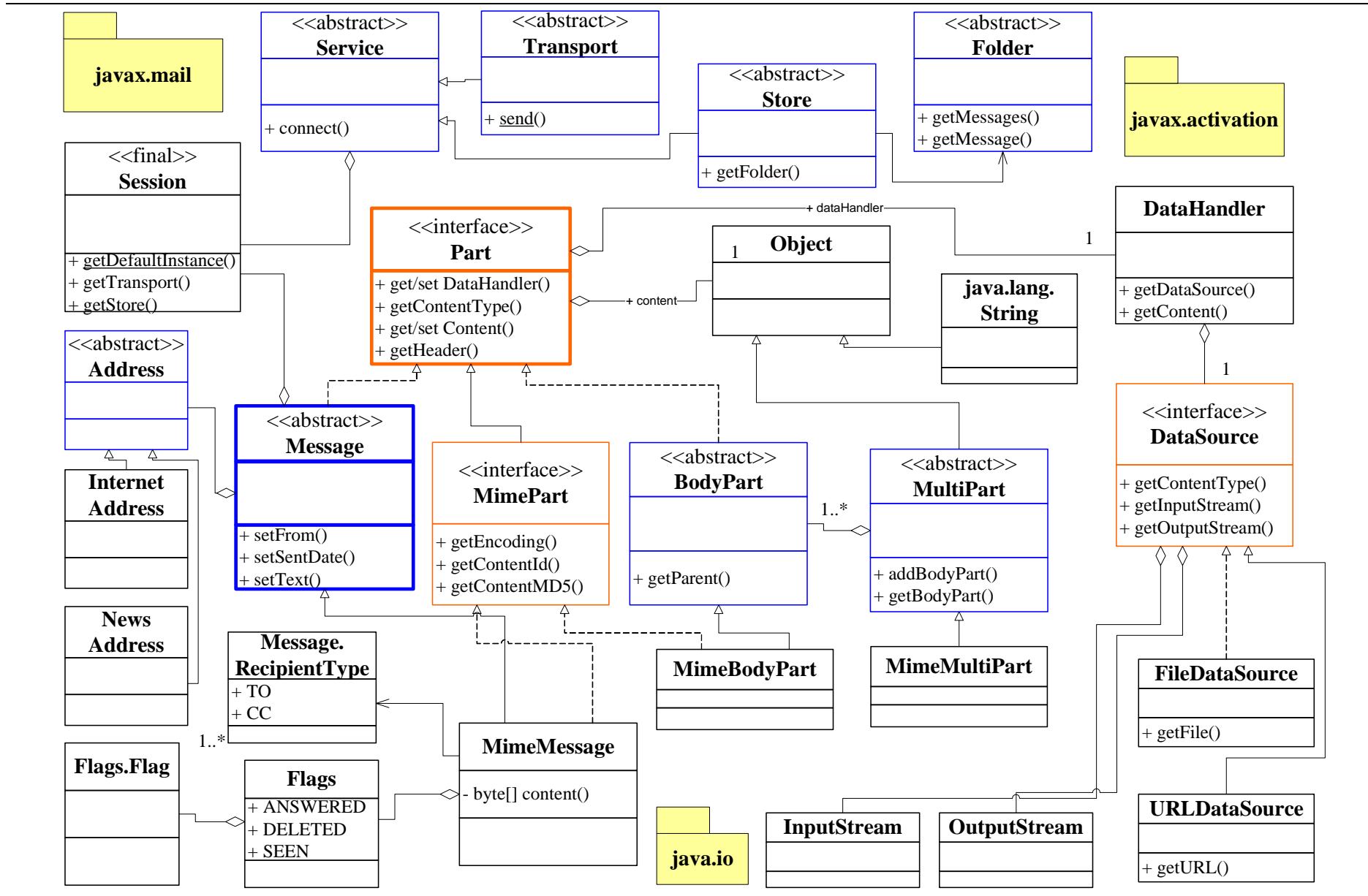
// Scan des BodyPart
for (int j=0; j<np; j++)
{
    System.out.println("--Composante n° " + j);
    Part p = msgMP.getBodyPart(j);
    String d = p.getDisposition();
    if (p.isMimeType("text/plain"))
        System.out.println("Texte : " + (String)p.getContent());
    if (d!=null && d.equalsIgnoreCase(Part.ATTACHMENT))
    {
        InputStream is = p.getInputStream();
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        int c;
        while ((c = is.read()) != -1) baos.write(c);
        baos.flush();
        String nf = p.getFileName();
        FileOutputStream fos =new FileOutputStream(nf);
        baos.writeTo(fos);
        fos.close();
        System.out.println("Pièce attachée " + nf + " récupérée");
    }
}
} // fin for j
} // fin for i
System.out.println("Fin des messages");
}
catch (NoSuchProviderException e)
{
    System.out.println("Erreur sur provider : " + e.getMessage());
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (IOException e)
{
    System.out.println("Erreur sur I/O : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur indéterminée : " + e.getMessage());
}
}
```

On obtient, suite à l'envoi expliqué aux paragraphes précédents :

```
Création d'une session mail
Obtention d'un objet store
Obtention d'un objet folder
Obtention des messages
Nombre de messages : 1
Nombre de nouveaux messages : 0
Liste des messages :
Message n° 0
Expéditeur : vilvens.claude@skynet.be
Sujet = 2ème Essai d'attachement par Skynet
Date : Wed Mar 27 18:30:41 CET 2002
-- Nombre de composantes = 3
--Composante n° 0
Texte : Veuillez trouver ci-joint les documents demandés - CV
--Composante n° 1
Pièce attachée d:\notes-java\notes-java-2001\BienvenueAInpres.doc récupérée
--Composante n° 2
Pièce attachée d:\notes-java\notes-java-2001\logo-INPRES.bmp récupérée
Fin des messages
```

## **20. Résumé : classes et interfaces utiles**

A ce stade, et avant de dire encore quelques mots du troisième protocole de messagerie (IMAP), nous pouvons dresser le tableau suivant des principales classes et interfaces utilisés dans la programmation classique de la messagerie électronique :



## 21. Le protocole IMAP

### 21.1 Une brève présentation

Le protocole **IMAP** (Internet Message Access Protocol) est présenté comme le successeur de POP3. Il s'agit d'un protocole qui permet de traiter les messages réceptionnés, mais de manière beaucoup plus puissante que ne le fait POP3.

En effet, POP3 ne propose aux utilisateurs connectés seulement par intermittence qu'une seule solution : ils téléchargent leurs mails sur leur machine locale à chaque connexion, les mails pouvant être détruits ou conservés sur le serveur (dans les limites de l'espace autorisé). Au contraire, IMAP propose à ces mêmes utilisateurs d'**accéder à leur boîte aux lettres distante comme si elle était locale**, c'est-à-dire *sans download immédiat* : à eux de choisir quels messages il faut détruire, conserver dans un folder ou effectivement télécharger sur la machine locale. De plus, ces messages peuvent être consultés par plusieurs clients simultanément et des folders peuvent être partagés : ceci constitue une alternative intéressante aux mailing-lists, puisqu'un message avec des annexes volumineuses pourra être consulté de manière beaucoup plus légère avec l'assurance d'en avoir la dernière version.

Evidemment, ce confort, auquel les utilisateurs de portables notamment seront sensibles, se paie par un trafic réseau plus important en termes de messages applicatifs.

IMAP en est à la version 4, révision 1, soit IMAP4rev1 en abrégé. Pour faire court :

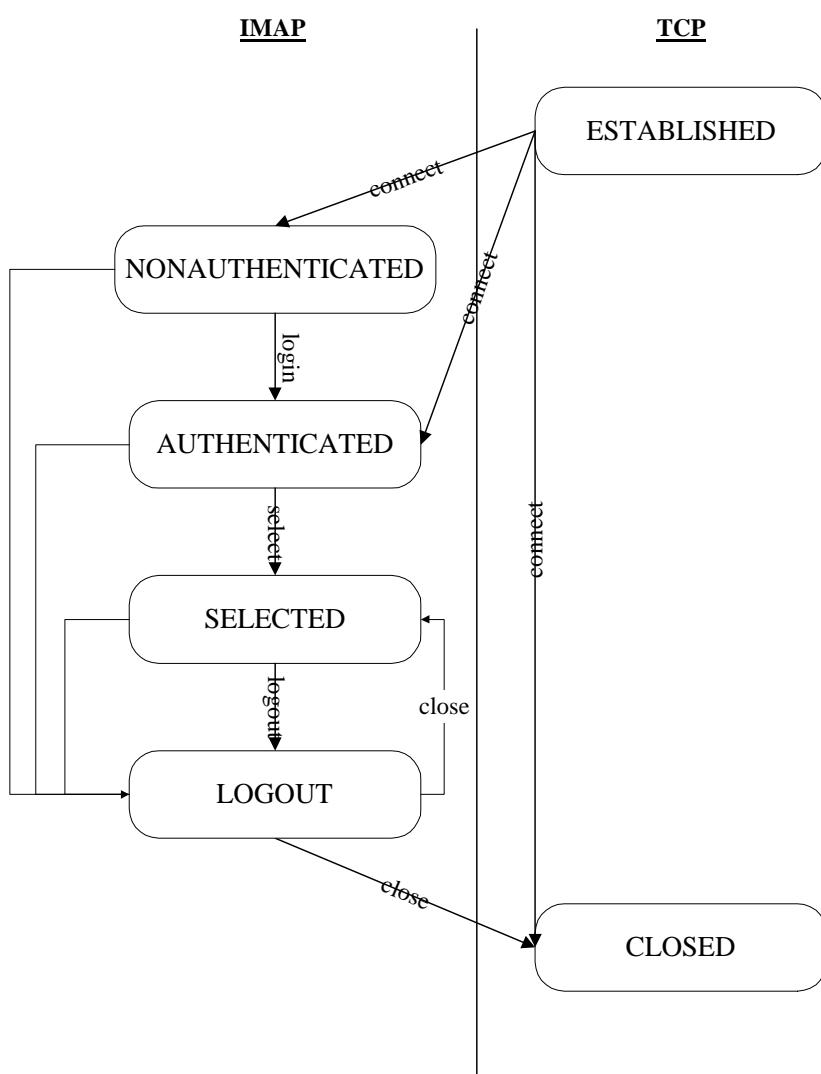
<b>IMAP</b>	
Nom complet :	<b>Internet Message Access Protocol</b>
Nature :	Protocole applicatif de récupération des messages électroniques
RFC :	2060, 1733 + 2086
Port par défaut :	143
Protocole de transport :	TCP
Principales commandes :	login, select, search, fetch, store, expunge, logout

IMAP fonctionne côté serveur comme un démon, indépendant ou au sein d'**inetd** ("super-serveur" qui attend les connexions réseaux et les redirige vers une processus créé pour la circonstance – intéressant pour les serveurs peu utilisés) sous UNIX et comme un service sous Windows. Il attend qu'un client se connecte sur son port d'écoute; lorsque c'est le cas, un nouveau sous-processus (ou un thread) serveur IMAP le prend en charge sur un autre port et attend les commandes d'accès à la boîte aux lettres visée.

De plus, *un client IMAP est sensé être un MUA multithread* : il peut envoyer plusieurs commandes simultanément et recevoir une réponse d'un serveur à tout moment, même s'il est occupé simultanément à autre chose. Dans ces conditions, il peut donc y avoir plusieurs "conversations" simultanées. Pour déterminer à laquelle une commande ou une réponse appartient, le client ajoute à chaque commande une chaîne de caractères alphanumérique qui sert d'identifiant au couple "commande-réponse" – typiquement : client → "TAG52 ..." et serveur → "TAG52 ...").

A nouveau, comme POP3, IMAP est un protocole à états, certaines commandes ne pouvant être exécutées que dans un état donné :

état	state	signification
NON_AUTHENTIFIE	NONAUTHENTICATED	La connexion TCP étant acceptée, le client doit à présent s'authentifier
AUTHENTIFIE	AUTHENTICATED	Le client s'est connecté et fait reconnaître par le serveur.
SELECTIONNE	SELECTED	Le client a sélectionné la boîte aux lettres sur laquelle il va effectuer diverses opérations.
FERME	LOGOUT	Dès que la connexion au serveur est perdue, soit parce que le client a terminé, soit parce que le serveur refuse ses services, ou encore parce que la connexion a été coupée.



Contrairement à POP3, ce protocole possède de nombreuses commandes, paramétrables au moyen de nombreux commutateurs. Cette relative complexité provient de la finesse des opérations réalisables, comme par exemple l'extraction d'une partie bien précise d'un message, la lecture et la modification des flags des messages (précisant si un message est

nouvellement arrivé, a été lu, doit être effacé, a fait l'objet d'une réponse, etc), la gestion des droits d'accès à aux boîtes aux lettres, etc.

## 21.2 Les commandes de base de consultation des messages

Pour parvenir à lire un message en utilisant IMAP, après s'être connecté au serveur, les commandes de base suivantes sont nécessaires. Précisons cependant auparavant qu'un message est désigné au moyen

- ◆ d'un numéro de message au sein de sa mailbox; il peut être modifié car si on efface un message, les numéros des messages suivants sont décrémentés;
- ◆ d'un **UID** (**Unique IDentifier**) qui est attribué définitivement au message à son arrivée dans la mailbox; il ne sera jamais réutilisé dans la mailbox sélectionnée; par contre, des messages se trouvant dans des mailbox différentes peuvent avoir le même UID.

Commande	Etats valides	Description
<b>login</b> <nom utilisateur> <mot de passe>	NONAUTHENTICATED	Permet de passer à l'état authentifié
<b>select</b> <nom de boîte aux lettres>	AUTHENTICATED ou SELECTED	Permet de sélectionner une boîte aux lettres – par exemple : INBOX.
<b>search</b> < <i>all</i>   <i>subject</i> <ch> [ [ <i>not</i> ] <i>body</i> <ch>] >	SELECTED	Permet de rechercher les messages correspondant à tel ou tel critère (la liste des flags est assez longue); on obtient ainsi les numéros des messages (uid) se trouvant dans la mailbox
<b>fetch</b> <numéro de message> < <i>all</i>   <i>body</i>   <i>body</i> [ <i>header</i>   <i>text</i>   <i>mime</i> ] <i>enveloppe</i>   <i>uid</i>   <i>rfc822</i> >	SELECTED	Permet de lire le message du numéro spécifié, en précisant la partie du mail souhaité (la liste des flags est assez longue).
<b>expunge</b>	SELECTED	Supprime les messages dont le flag "deleted" est positionné, ce qui se fait avec la commande suivante
<b>store</b> <numéro de message> + <i>FLAGS</i>   - <i>FLAGS</i> ( \i{answered}   \i{deleted}   \i{seen})	SELECTED	Permet de modifier le flag précisé en fin de commande pour le message du numéro précisé.
<b>logout</b>	tous	A votre avis ?
<b>noop</b>	tous	L'opération vide – elle sert à montrer que le client (comme le canard) est toujours vivant ...
<b>capability</b>	tous	Renseigne sur les disponibilités du serveur.



## IMAP

**Quatre réponses et bien d'autres ...**

Les réponses d'un serveur IMAP sont relativement complexes. Les 4 réponses de base indiquent la fin du traitement de la commande du client en précisant sa réussite ou son échec. Il s'agit de : OK, BAD (erreur, selon le protocole, dans la formulation de la commande), NO (erreur dans la réalisation de la commande) et BYE (le serveur va fermer la connexion). Ces réponses sont dites "taggées" [*tagged*], en référence aux tags identifiant un couple commande-réponse : il y a donc une seule réponse "taggée" pour une commande donnée.

Cependant, le serveur doit aussi spécifier les réponses proprement dites à la commande et peut fournir de nombreuses autres informations utiles. Ces éléments de réponse sont des réponses "non taggées" [*untagged*] ; elles sont précédées par le caractère '\*'.

Enfin, des codes d'erreur complémentaires sont disponibles, comme par exemple ALERT, FLAGS, PERMANENT-FLAGS (indique les flags qui peuvent être modifiés de manière permanente), UNSEEN (numéro du premier message non lu dans la mailbox sélectionnée), PARSE (le header du message ne peut être correctement analysé), etc.

### 21.3 Un exemple de transaction IMAP en ligne de commande

Concrètement, voici un dialogue interactif entre un client "vilvens" et un serveur IMAP. Deux messages (qu'il s'est en fait envoyé à lui-même) l'attendent dans sa boîte aux lettres. Après être passé dans l'état authentifié, il va lire ses messages et finalement effacer le premier. Cela donne :

```
vilvens@cure:~ > telnet u2 143
Trying 10.59.5.219...
Connected to u2.
Escape character is '^].
* OK [CAPABILITY IMAP4 IMAP4REV1 STARTTLS LOGIN-REFERRALS
AUTH=LOGIN] u2.wildne)
tag1 capability
* CAPABILITY IMAP4 IMAP4REV1 STARTTLS NAMESPACE IDLE MAILBOX-
REFERRALS SCAN SORN
tag1 OK CAPABILITY completed
tag2 login vilvens Etpuisquoientcore
* CAPABILITY IMAP4 IMAP4REV1 STARTTLS NAMESPACE IDLE MAILBOX-
REFERRALS SCAN SORD
tag2 OK LOGIN completed
tag3 select inbox
* 2 EXISTS
* 2 RECENT
* OK [UIDVALIDITY 1031664763] UID validity status
* OK [UIDNEXT 3] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent fs
* OK [UNSEEN 1] first unseen message in /var/spool/mail/vilvens
tag3 OK [READ-WRITE] SELECT completed
tag4 search all
* SEARCH 1 2
```

```
tag4 OK SEARCH completed
tag5 fetch 2 all
* 2 FETCH (FLAGS (\Recent INTERNALDATE "10-Sep-2002 15:26:27 +0200"
RFC822.SIZE)
tag5 OK FETCH completed
tag6 fetch 2 body[header]
* 2 FETCH (BODY[HEADER] {397}
X-UIDL: +>2!!><O!!7+3"!-j="!
Return-Path: <vilvens@u2.wildness.loc>
Received: from ulysses (vilvens@[10.59.5.224])
    by u2.wildness.loc (8.11.3/jtpda-5.3.3) with SMTP id g8ADOrS25724
    for <vilvens@u2.wildness.loc>; Tue, 10 Sep 2002 15:25:12 +0200
Date: Tue, 10 Sep 2002 15:25:12 +0200
From: vilvens@u2.wildness.loc
Message-Id: <200209101325.g8ADOrS25724@u2.wildness.loc>

)
* 2 FETCH (FLAGS (\Recent \Seen))
tag6 OK FETCH completed
tag7 fetch 1 text
tag7 BAD Bogus attribute list in FETCH
tag7 fetch 1 body[text]
* 1 FETCH (BODY[TEXT] {70}
Aujourd'hui est un grand jour :
Je suis papa pour la 15eme fois !!!
)
* 1 FETCH (FLAGS (\Recent \Seen))
tag7 OK FETCH completed
tag8 fetch 1 body "papa"
tag8 BAD Excessively complex FETCH attribute list
tag9 store 1 +flags (\deleted)
* 1 FETCH (FLAGS (\Recent \Seen \Deleted))
tag9 OK STORE completed
search all
search BAD Command unrecognized: ALL
tag10 search all
* SEARCH 1 2
tag10 OK SEARCH completed
expunge
expunge BAD Missing command
tag11 expunge
* 1 EXPUNGE
* 1 EXISTS
* 1 RECENT
tag11 OK Expunged 1 messages
tag12 logout
* BYE u2.wildness.loc IMAP4rev1 server terminating connection
tag12 OK LOGOUT completed
Connection closed by foreign host.
vilvens@cure:>
```

## 21.4 Les boîtes de réception

IMAP propose encore un certain nombre de commandes permettant de manipuler les boîtes de réception :

Commande	Etats valides	Description
<b>create</b> <nom de boîte aux lettres>	AUTHENTICA-TED ou SELECTED	Permet de créer une nouvelle boîte aux lettres.
<b>list</b> <nom de folder> < nom de boîte aux lettres>	AUTHENTICA-TED ou SELECTED	Fournit la liste des boîtes aux lettres accessibles à l'utilisateur – les caractères jokers '*' et '%' peuvent être utilisés.
<b>rename</b> <nom actuel de boîte aux lettres> <nouveau nom de boîte aux lettres>	AUTHENTICA-TED ou SELECTED	Permet de modifier le nom d'une boîte aux lettres
<b>close</b>	SELECTED	Ferme la boîte aux lettres en cours d'utilisation et valide l'effacement des messages marqués pour la destruction – le client revient à l'état AUTHENTICATED.
<b>delete</b> <nom de boîte aux lettres>	AUTHENTICA-TED ou SELECTED	A votre avis ?
<b>append</b> <nom de boîte aux lettres> {<nombre de caractères>}	AUTHENTICA-TED ou SELECTED	Placera le message qui va être défini par après à la fin de la boîte aux lettres spécifiée.

On peut ainsi imaginer ceci :

```

vilvens@cure:~ > telnet u2 143
Trying 10.59.5.219...
Connected to u2.
Escape character is '^'.
* OK [CAPABILITY IMAP4 IMAP4REV1 STARTTLS LOGIN-REFERRALS
AUTH=LOGIN] u2.wildne)
tag1 login vilvens EtPuisQuoiEncore
* CAPABILITY IMAP4 IMAP4REV1 STARTTLS NAMESPACE IDLE MAILBOX-
REFERRALS SCAN SORD
tag1 OK LOGIN completed
tag2 select inbox
* 1 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1031664763] UID validity status
* OK [UIDNEXT 3] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent fs
tag2 OK [READ-WRITE] SELECT completed

```

```
tag4 create inbox.secret
tag4 OK CREATE completed
tag4 list """
* LIST (\NoInferiors \UnMarked) "/" compile.sh
* LIST (\NoSelect) "/" java
* LIST (\NoSelect) "/" webdev
* LIST (\NoSelect) "/" webdev/WEB-INF
* LIST (\NoSelect) "/" webdev/WEB-INF/classes
* LIST (\NoSelect) "/" webdev/WEB-INF/classes/Preferences
* LIST (\NoInferiors \UnMarked) "/" webdev/WEB-INF/classes/FFServlet.class
* LIST (\NoInferiors \UnMarked) "/" webdev/WEB-INF/classes/FFServlet.java
* LIST (\NoSelect) "/" webdev/WEB-INF/lib
* LIST (\NoInferiors \UnMarked) "/" webdev/WEB-INF/lib/Preferences.jar
* LIST (\NoInferiors \UnMarked) "/" webdev/WEB-INF/web.xml
* LIST (\NoSelect) "/" webdev/public_html
* LIST (\NoInferiors \UnMarked) "/" webdev/public_html/useBean.html
* LIST (\NoSelect) "/" webdev/jsp
* LIST (\NoInferiors \UnMarked) "/" webdev/jsp/Identification.jsp
* LIST (\NoInferiors \UnMarked) "/" webdev/jsp/jspuser.jsp
* LIST (\NoInferiors) "/" inbox.secret
* LIST (\NoInferiors) NIL INBOX
tag4 OK LIST completed
tag6 rename inbox.secret inbox.ultra-secret
tag6 OK RENAME completed
tag7 list """
* LIST (\NoInferiors \UnMarked) "/" compile.sh
...
* LIST (\NoInferiors \Marked) "/" inbox.ultra-secret
* LIST (\NoInferiors) NIL INBOX
tag7 OK LIST completed
tag8 select inbox
* 1 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1031664763] UID validity status
* OK [UIDNEXT 3] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent fs
tag8 OK [READ-WRITE] SELECT completed
tag9 search all
* SEARCH 1
tag9 OK SEARCH completed
tag10 fetch 1 body[etxt]
tag10 BAD Unknown section text specifier
tag10 fetch 1 body[text]
* 1 FETCH (BODY[TEXT] {85}
Je viens de passer mon examen chez vil+char
Gloups - ce sont de vrais tueurs !?????
)
tag10 OK FETCH completed
tag13 append inbox.ultra-secret {120}
```

---

```
+ Ready for argument
from: <vilvens@u2.wildness.loc>
to: <cvilvens@prov-liege.be>
subject: etat du terrain
Zut : aucun prof absent !!!
tag13 OK APPEND completed
tag14 select inbox.ultra-secret
* 1 EXISTS
* 1 RECENT
* OK [UIDVALIDITY 1031721488] UID validity status
* OK [UIDNEXT 2] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)] Permanent fs
* OK [UNSEEN 1] first unseen message in /home/vilvens/inbox.ultra-secret
tag14 OK [READ-WRITE] SELECT completed
tag15 search all
* SEARCH 1
tag15 OK SEARCH completed
tag16 fetch 1 body
* 1 FETCH (BODY ("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 29 1))
tag16 OK FETCH completed
tag16 fetch 1 body[text]
* 1 FETCH (BODY[TEXT] {29}
Zut : aucun prof absent !!!
)
* 1 FETCH (FLAGS (\Recent \Seen))
tag16 OK FETCH completed
tag18 close inbox.ultra-secret
tag BAD Command unrecognized: 18
tag18 close
tag18 OK CLOSE completed
tag19 search all
tag19 BAD Command unrecognized: SEARCH
tag22 logout

* BYE u2.wildness.loc IMAP4rev1 server terminating connection

tag22 OK LOGOUT completed
Connection closed by foreign host.
vilvens@cure:~ >
```

Une extension du protocole IMAP (définie dans la RFC 2086) permet même de gérer des droits sur ces boîtes de réception en utilisant une ACL (Access Control List).

## 22. IMAP et Java

On s'en doute, l'encapsulation poussée des classes **Transport** et surtout **Store** laisse espérer qu'il suffit, pour récupérer un message en usant du protocole IMAP, de remplacer "pop3" par "imap". Effectivement, le programme suivant accède à une boîte aux lettres sur U2, machine qui possède un serveur IMAP :

### JMailSimplePartRecvImap.java

```
/*
 * JMailSimplePartRecvImap.java
 * Created on 19 mars 2002, 16:36
 */
...

public class JMailSimplePartRecvImap
{
    static String host = "u2.wildness.loc";
    public JMailSimplePartRecvImap() { }

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);
        prop.list(System.out);

        try
        {
            String user = "vilvens";
            String pwd = "EtPuisQuoiEncore";
            System.out.println("Obtention d'un objet store");
            Store st = sess.getStore("imap");
            st.connect(host, user, pwd);

            System.out.println("Obtention d'un objet folder");
            Folder f = st.getFolder("INBOX");
            f.open(Folder.READ_ONLY);

            System.out.println("Obtention des messages");
            Message msg[] = f.getMessages();
            System.out.println("Nombre de messages : " + f.getMessageCount());
            System.out.println("Nombre de nouveaux messages : " + f.getNewMessageCount());

            System.out.println("Liste des messages : ");
            for (int i=0; i<msg.length; i++)
            {
                System.out.println("\n<nHeaders du message n°" + (i+1));
                Enumeration e = msg[i].getAllHeaders();
                Header h = (Header)e.nextElement();
            }
        }
    }
}
```

```
while (e.hasMoreElements())
{
    System.out.println(h.getName() + " --> " + h.getValue());
    h = (Header)e.nextElement();
}
System.out.println("Texte : " + (String)msg[i].getContent());
}
System.out.println("Fin des messages");
}
...
}
```

Le résultat est tout à fait conforme à ce que l'on en attend :

Création d'une session mail

Obtention d'un objet store

Obtention d'un objet folder

Obtention des messages

**Nombre de messages : 1**

**Nombre de nouveaux messages : 0**

**Liste des messages :**

<nHeaders du message n°1

X-UIDL --> +>2!!><O!!7+3"!-j="!

Return-Path --> <vilvens@u2.wildness.loc>

Received --> from ulyssse (vilvens@[10.59.5.224])

    by u2.wildness.loc (8.11.3/jtpda-5.3.3) with SMTP id g8ADOrS25724

    for <vilvens@u2.wildness.loc>; Tue, 10 Sep 2002 15:25:12 +0200

Date --> Tue, 10 Sep 2002 15:25:12 +0200

From --> vilvens@u2.wildness.loc

Texte : Je viens de passer mon examen chez vil+char

Gloups - ce sont de vrais tueurs !????

Fin des messages

## 23. Les événements de type mail

Les couples event-listener nous sont familiers depuis longtemps, puisque toute la gestion des GUI selon l'AWT et selon Swing utilise ce paradigme événementiel. Les Java beans nous ont conforté dans l'idée que l'on pouvait, en fait, associer à une classe (ou à un groupe de classes) un ensemble d'événements permettant une programmation de type asynchrone.

Les Java mails ne font pas exception, loin de là. En effet, l'API JavaMail permet de désigner un certain nombre de listeners chargés de gérer des événements concernant la messagerie électronique : connexion à un serveur, envoi ou réception d'un message, modifications apportées à un message ou à un folder. Plus précisément, la classe **MailEvent** du package javax.mail.event dérive de java.util.EventObject et sert de classe de base aux classes événements de type mail. Elle possède donc un certain nombre de filles :

classe événement	sémantique
ConnectionEvent	connexion à un serveur de messagerie
TransportEvent	tout ce qui concerne l'envoi d'un message : message correctement délivré, ou incomplètement, ou non délivré
StoreEvent	ce qui concerne la réception d'un message, c'est-à-dire l'alerte correspondant à la réception d'un nouveau message
MessageCountEvent	comme le nom l'indique, l'événement consiste en un ajout ou un retrait d'un message d'une boîte de réception
MessageChangedEvent	modifications des messages d'une boîte de réception
FolderEvent	modifications apportées à un folder (fonctionnalité qui n'est pas forcément implémentée pour un service de messagerie)

Inutile sans doute de préciser qu'à chaque événement correspond un listener, ainsi d'ailleurs qu'une adapter fournissant une implémentation minimale de cet interface listener. Le développeur procèdera donc comme d'habitude, c'est-à-dire en implémentant directement le listener considéré ou en dérivant de l'adapter. Et, bien sûr, chaque classe concernée par un type d'événement une méthode addXXXListener pour prendre en compte un objet listener donné. Donc, par exemple, on va trouver dans les librairies :

classe / méthodes	package
FolderEvent	javax.mail.event
FolderListener	javax.mail.event
FolderAdapter	javax.mail.event
Folder public void addFolderListener(FolderListener l) public void addConnectionListener(ConnectionListener l) public void addMessageChangedListener(MessageChangedListener l) public void addMessageCountListener(MessageCountListener l)	javax.mail

ou encore (l'adapter n'est pas fourni ;-) ...) :

classe / méthodes	package
StoreEvent	javax.mail.event
StoreListener	javax.mail.event
Store public void addFolderListener(FolderListener l) public void addStoreListener(StoreListener l)	javax.mail

Le mécanisme est assez analogue à celui des événements attachés aux Java Beans : chaque classe événement possède une méthode

`public void dispatch(java.lang.Object listener)`

dont le rôle est bien clair.

En pratique, il faut cependant bien admettre que l'absence habituelle de connexion permanente diminue considérablement l'intérêt de ce système événements-listeners. A titre purement illustratif, voici :

- ♦ un listener chargé de surveiller le changement du nombre de messages dans un folder; cette classe nommée **MailboxCountListener** est dérivée de l'adapter **MessageCountAdapter**. Elle se contente de redéfinir la méthode

```
public void messagesAdded(MessageCountEvent e)
```

- ♦ un listener chargé de surveiller les alertes sur une boîte de réception; cette classe appelée class **MailboxStoreListener** implémente l'interface **StoreListener** qui ne comporte que la méthode

```
public void notification(StoreEvent e).
```

### JMailSimplePartRecv.java

```
/*
 * JMailSimplePartRecv.java
 * Created on 1 octobre 2002, 12:08
 */
import javax.mail.*;
import javax.mail.internet.*;
import javax.mail.event.*;
import javax.activation.*;
import java.util.*;
import java.io.*;

/**
 * @author VILVENS
 * @version
 */
public class JMailRecvEvents
{
    static String host = "pop.skynet.be"; //wildness.loc";
    public JMailRecvEvents() { }

    public static void main (String args[])
    {
        Properties prop = System.getProperties();
        System.out.println("Création d'une session mail");
        Session sess = Session.getDefaultInstance(prop, null);

        try
        {
            String user = "vilvens";
            String pwd = "EtPuisQuoiEncore";
            System.out.println("Obtention d'un objet store");
            Store st = sess.getStore("imap");
            st.connect(host, user, pwd);
            st.addStoreListener(new MailboxStoreListener());
        }
    }
}
```

```
System.out.println("Obtention d'un objet folder");
Folder f = st.getFolder("INBOX");
f.open(Folder.READ_ONLY);
f.addMessageCountListener(new MailboxCountListener());
System.out.println("Listeners enregistrés");

int np=-1;
boolean fini=false;
int cpt=0;
while (!fini)
{
    Thread.sleep(2000);
    cpt++;
    int nm = f.getMessageCount();
    if (np != nm)System.out.println ("nm = " + nm);
    else System.out.print("-");
    if (cpt%80==0) System.out.println();
    np = nm;
}
}
catch (NoSuchProviderException e)
{
    System.out.println("Erreur sur provider : " + e.getMessage());
}
catch (MessagingException e)
{
    System.out.println("Erreur sur message : " + e.getMessage());
}
catch (Exception e)
{
    System.out.println("Erreur indéterminée : " + e.getMessage());
}
}

// -----
class MailboxCountListener extends MessageCountAdapter
{
    MailboxCountListener () { /*super();*/ }

    public void messagesAdded(MessageCountEvent e)
    {
        System.out.println("* MessageCountEvent *");
        Message msg[] = e.getMessages();
        System.out.println(msg.length + " nouveau(x) message(s) !");
    }
}
```

```
class MailboxStoreListener implements StoreListener
{
    MailboxStoreListener() {}

    public void notification(StoreEvent e)
    {
        System.out.println("* StoreEvent *");
        System.out.println("-->" + e.getMessage());
    }
}
```

## 24. La suppression des messages

Supprimer un message logiquement ne pose guère de problèmes : il suffit de positionner le flag **DELETED** pour ce message au moyen de la méthode

```
public void setFlags(Flags flag, boolean set) throws MessagingException  
de la classe MimeMessage (qui redéfinit la méthode abstraite du même nom de la classe mère Message). Dans le cas qui nous occupe, c'est bien sûr le flag DELETED qu'il faudra positionner en utilisant la constante de classe  
public static final Flags.Flag DELETED
```

Cependant, pour que ceci soit possible, il faudra ouvrir le folder en **READ\_WRITE**. Donc :

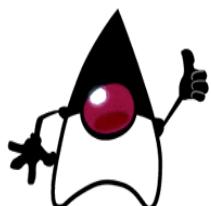
```
Folder f = st.getFolder("INBOX");
f.open(Folder.READ_WRITE);
```

La suppression physique des messages se réalise,

- ◆ dans le cas où le protocole de store est **POP**, par l'appel de la méthode de la classe Folder :  
public abstract void **close**(boolean expunge) throws MessagingException  
dont le flag placé à true indique évidemment qu'il faut supprimer les messages marqués pour la destruction;
- ◆ dans le cas où le **protocole** de store est **IMAP**, par l'appel de la méthode de la classe Folder :

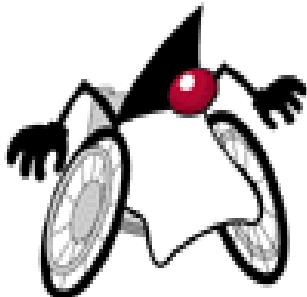
```
public abstract Message[] expunge()throws MessagingException
```

Bien sûr, celle-ci supprime les messages dont le flag **DELETED** est positionné et les place dans le tableau retourné par la méthode. Bien sûr, les messages survivants sont ensuite renumérotés au sein du folder.



Changeons à présent de protocole applicatif et venons-en à l'un des plus célèbres protocoles construits sur TCP, juste après HTTP ...

## XXII. Le protocole de transfert de fichiers FTP et Java FTP



*Il y a des gens qui ont une bibliothèque comme les eunuques ont un harem.*

(V. Hugo, Fragments)

A priori, tout le monde connaît FTP, le protocole de transfert de fichiers. Ou, plus exactement, tout le monde, chez les informaticiens, s'en sert tous les jours au moyen d'un logiciel ou, parfois, au moyen de quelques instructions en ligne de commandes du type get ou put. Mais ce qui se passe "en-dessous" est quelque peu plus complexe. C'est ce que nous allons voir ici, en étudiant également ce qui a été fait dans le domaine en programmation Java.

### 1. Le protocole FTP

FTP (File Transfer Protocol) est un protocole de la couche applicative dédié au transfert de fichiers entre machines distantes, selon un schéma client-serveur. Pour faire court :

FTP	
Nom complet :	File Transfer Protocol
Nature :	Protocole applicatif de transfert de fichiers
RFC :	959
Port par défaut :	21 + 20
Protocole de transport :	TCP
Principales commandes :	user, pass, pasv, type, retr

Les commandes de ce protocole sont assez nombreuses. Elles sont le plus souvent formées de 4 lettres, suivies éventuellement d'un paramètre. On peut distinguer :

- ◆ les commandes d'**accès**;
- ◆ les commandes de **service**;
- ◆ les commandes de **paramétrage**.

Si l'objectif du protocole semble clair, il n'est pas aussi simple à atteindre qu'il y paraît :

- ◆ les formats de stockage ne sont pas forcément les mêmes sur les machines intervenantes;
- ◆ le transfert doit être effectué de la manière la plus rapide et la plus fiable possible;
- ◆ il faut déterminer qui peut transférer quoi;
- ◆ les problèmes de sécurité classiques en réseau existent également ici.

## 2. Le modèle FTP

Dans un scénario FTP typique, **le client** qui désire transférer un fichier depuis un serveur [*download*] va utiliser ce que l'on appelle un processus **USER-FTP**. Celui-ci comporte

- ◆ un interface utilisateur, qui met à la disposition de l'utilisateur un langage local de commande (comme les célèbres commandes "get" ou "put");
- ◆ un interpréteur de protocole **USER-PI** (**Protocol Interpreter**);
- ◆ un processus de transfert de données **USER-DTP** (**Data Transfer Processus**) qui établit et gère la connexion.

Le client va donc se connecter au serveur sur le port prévu (par défaut, 21).

**Le serveur** est matérialisé par un processus **SERVER-FTP** (sous UNIX, c'est un démon `ftpd` du super démon `/etc/inetd`) qui comporte essentiellement :

- ◆ un interpréteur de protocole **SERVER-PI**;
- ◆ un processus de transfert de données **SERVER-DTP**.

Le SERVER-PI est à l'écoute sur le port 21. Lorsqu'un client se connecte, le SERVER-PI va établir une connexion que l'on appelle le **canal de contrôle**. C'est par ce canal qu'il recevra les commandes du USER-PI, qu'il y répondra et qu'il pilotera ensuite le SERVER-DTP. *Le dialogue du canal de contrôle s'effectue en utilisant TELNET*. A priori, on peut supposer qu'il est fait appel au service Telnet des machines impliquées – il est probable qu'il existe. Cependant, on peut aussi envisager que les USER-PI et SERVER-PI soient écrits en suivant directement les règles de Telnet.

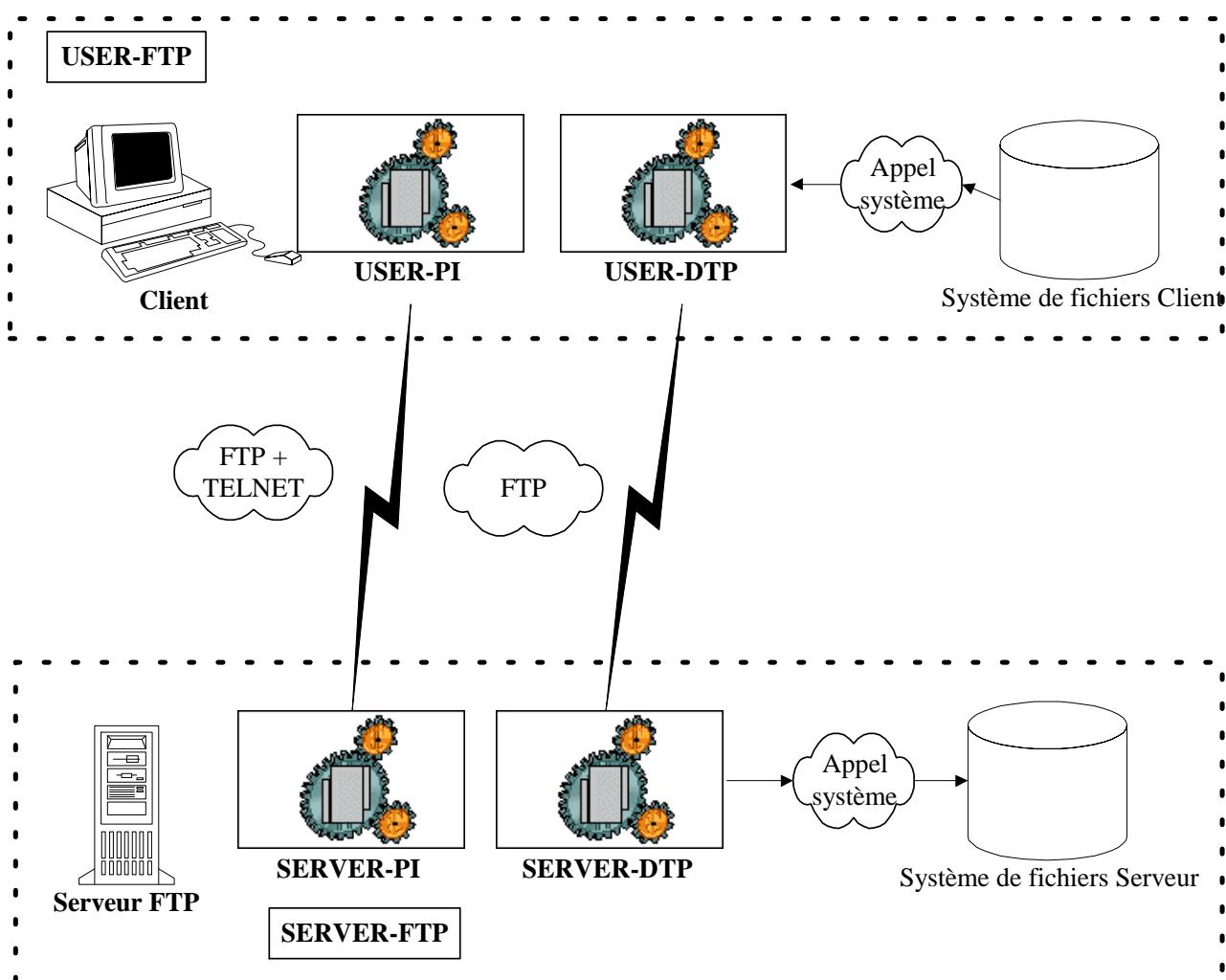
Le client, par son USER-PI, peut à présent s'identifier (nom et mot de passe – sauf s'il est anonyme) puis spécifier, outre le nom du fichier à retrouver, le type d'encodage et de représentation, le mode de transfert, le port de données utilisé, ... tout cela à destination du SERVER-PI. Le serveur répond par la connexion établie.

Le USER-PI pilote alors le USER-DTP. Lorsqu'il est question du transfert lui-même, il va falloir *initialiser une nouvelle connexion*, dédicacée au transfert, entre la socket client et une socket serveur attachée au port de données qui a été spécifié. Dans le mode de fonctionnement de base, ce USER-DTP se met en écoute sur ce port de données spécifié : il est "*passif*". Le SERVER-DTP va se connecter sur ce même port de données : il est "*actif*".

Mais les rôles peuvent être inversés, et c'est le cas le plus fréquent avec les machines dont nous nous servons : le serveur peut passer en mode "*passif*" (il se met en écoute) tandis que le client devient actif (c'est lui qui va se connecter sur le port d'écoute du serveur). Pour ce faire, le serveur va utiliser pour l'écoute

- ◆ soit, par défaut, le *port adjacent* de numéro immédiatement inférieur à celui qui est utilisé par le canal de connexion, soit donc le port 20 ( $21-1=20$ );
- ◆ soit un port qu'il précise dans le retour de la commande qui l'a mis en mode passif.

Il crée ainsi un deuxième canal, le **canal de données**, par lequel le transfert de fichier va être effectué. En fait, comme on peut le voir avec une commande `netstat`, ce canal de données correspond à deux connexions réseaux : une pour le contrôle et une pour le transfert lui-même. Schématiquement :



### 3. L'interface de commande FTP

Il est probable que chaque lecteur a déjà utilisé la commande `ftp` pour transférer un fichier dans un sens ou dans un autre. Il convient cependant de remarquer que les commandes utilisées classiquement sont bien des commandes d'un interface standardisé et PAS des commandes du protocole FTP proprement dit. L'analyse d'un téléchargement FTP classique va éclairer le mécanisme réel.

Supposons qu'un utilisateur en session sur une machine Linux appelée `Cure` veuille télécharger un fichier qui se trouve sur une autre machine Linux appelée `U2`. Au départ, la commande "`netstat -an`" sur `U2` nous indique que le démon `ftp` est en attente :

<code>tcp</code>	<code>0</code>	<code>0 0.0.0.0:21</code>	<code>0.0.0.0:*</code>	<code>LISTEN</code>
------------------	----------------	---------------------------	------------------------	---------------------

L'utilisateur se connecte donc sur ce port 21 :

```
vilvens@cure:~ > ftp u2
Connected to 10.59.5.219.
220 u2.wildness.loc FTP server (Version 6.5/OpenBSD, linux port 0.3.2) ready.
Name (u2:vilvens): vilvens
331 Password required for vilvens.
Password: ...
```

```
230- Have a lot of fun...
230 User vilvens logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

La connexion est visiblement établie :

tcp	0	0	10.59.5.219: <b>21</b>	10.59.5.224:4229	<b>ESTABLISHED</b>
-----	---	---	------------------------	------------------	--------------------

L'utilisateur peut alors vaquer à ses occupations, usant des commandes de l'interface standard :

```
ftp> ls
229 Entering Extended Passive Mode (|||32995|)
200 EPRT command successful.
150 Opening BINARY mode data connection for 'file list'.
.
..
compile.sh
java
webdev
inbox.ultra-secret
226 Transfer complete.
WARNING! 6 bare linefeeds received in ASCII mode.
File may not have transferred correctly.
ftp> cd webdev
250 CWD command successful.
ftp> ls
200 EPRT command successful.
150 Opening BINARY mode data connection for 'file list'.
.
..
WEB-INF
public_html
jsp
226 Transfer complete.
WARNING! 5 bare linefeeds received in ASCII mode.
File may not have transferred correctly.
ftp> cd public_html
250 CWD command successful.
ftp> ls
200 EPRT command successful.
150 Opening ASCII mode data connection for 'file list'.
.
..
useBean.html
226 Transfer complete.
ftp> get useBean.html
local: useBean.html remote: useBean.html
```

---

```

200 EPRT command successful.
150 Opening BINARY mode data connection for 'useBean.html' (598 bytes).
100% |*****| 598 154.69 KB/s 00:00 ETA
226 Transfer complete.
598 bytes received in 00:00 (143.62 KB/s)
ftp> quit
221 Goodbye.
vilvens@cure:~ >

```

Un netstat –an commandé sur U2 juste après l'exécution de la commande get montre qu'un canal de données (soit 2 connexions réseaux) a bien été utilisé au moyen, du côté serveur qui est passé en mode passif, du port adjacent 20 :

tcp	0	0 0.0.0.0:21	0.0.0.0:*	<b>LISTEN</b>
tcp	0	0 10.59.5.219:20	10.59.5.224:4254	<b>TIME_WAIT</b>
tcp	0	0 10.59.5.219:20	10.59.5.224:4252	<b>TIME_WAIT</b>
tcp	0	0 10.59.5.219:21	10.59.5.224:4229	<b>ESTABLISHED</b>

On a évidemment remarqué dans la réponse la présence des codes de retour et aussi, parfois, l'évocation d'une commande comme CWD, commande que l'utilisateur n'a pourtant pas tapée : il s'agit en fait d'une véritable commande FTP, qui a été générée par la commande ls du langage local d'interface. Ce sont évidemment ces véritables commandes qui nous intéressent ...

### **Remarque**

On aura sans doute constaté que, du côté serveur, le même couple (adresse, port) est utilisé pour deux connexions distinctes. On se souviendra que, par défaut, deux bind() sur ce même couple sont refusés. Ceci signifie donc que les sockets en question doivent être paramétrées pour admettre cette double utilisation, ce qui est possible dans le domaine AF\_INET en positionnant l'option correspondante au moyen des constantes SO\_REUSEADDR et SO\_REUSEPORT<sup>4</sup>.

## **4. Les principales commandes d'accès et de service**

Pour parvenir à transférer un fichier en utilisant FTP, il faut d'abord se connecter à ce serveur (user et pass) puis utiliser les commandes de base suivantes :

Commande	Description
<b>user &lt;nom utilisateur&gt;</b>	Le nom attendu est l'identifiant de l'utilisateur que le serveur attend pour permettre l'accès au système de fichiers de la machine hôte.
<b>pass &lt;mot de passe&gt;</b>	Il s'agit du complément de la commande précédente.
<b>cwd &lt;rédertoire&gt;</b>	<b>Change Working Directory</b> : permet de changer le répertoire de travail sur la machine distante; le chemin est exprimé dans le langage du système de fichier local.
<b>pwd</b>	La réponse sera le nom du répertoire courant sur le serveur.
<b>retr &lt;fichier&gt;</b>	<b>Download</b> : pour demander le transfert du fichier du serveur vers le client (ou un autre serveur) – le chemin est exprimé dans le langage du système de fichier du serveur.

<sup>4</sup> voir "Programmation TCP/IP", chapitre VIII, paragraphe 5.2 – du même auteur ...

<b>stor &lt;fichier&gt;</b>	<b>Upload</b> : pour demander le transfert du fichier du client vers le serveur – si un fichier du même nom existe, il est écrasé ...
<b>appe &lt;fichier&gt;</b>	... tandis qu'ici, les données transférées viennent s'ajouter au fichier existant.
<b>list &lt;chemin&gt;</b>	Fournit la liste des fichiers du répertoire spécifié (le répertoire courant en l'absence de paramètre) dont le DTP est passif.
<b>quit</b>	A votre avis ?
<b>rein</b>	<b>REINITIALIZE</b> : permet de supprimer la connexion créée par la dernière commande user – le transfert en cours est cependant achevé normalement.
<b>abor</b>	Pour interrompre une commande de transfert.
<b>help</b>	Permet d'obtenir de l'aide sur l'implémentation du serveur. En particulier, "help site" permet d'obtenir des renseignements sur des services utilisés pour les transferts de fichiers, mais trop peu universels pour justifier une commande reconnue dans le protocole.



FTP

*Quelques codes de retour*

Code	Signification
125	Canal de données déjà ouvert – le transfert est en cours.
150	Ouverture du canal de données.
200	Commande effectuée sans échec.
202	Commande non implémentée.
220	Le service est prêt pour un nouvel utilisateur.
221	Fermeture du canal de contrôle.
225	Canal de données ouvert – pas de transfert en cours.
226	Fermeture du canal de données.
227	Le serveur passe en mode passif – la réponse comporte (h1,h2,h3,h4,p1,p2) : les 4 premiers octets correspondent à l'adresse, les deux derniers au port utilisés.
230	Utilisateur connecté.
250	Requête de fichier réalisée.
331	Nom d'utilisateur reconnu, mais son mot de passe est requis.
421	Service non disponible – fermeture du canal de contrôle.
425	Impossible d'ouvrir un canal de données.
426	Connexion fermée et transfert interrompu.
450	Fichier non disponible.
500	Erreur de syntaxe.
501	Erreur de syntaxe dans les paramètres
502	Commande non implémentée.
503	Mauvaise combinaison de commandes.
504	Commande non implémentée pour un tel paramètre.
530	L'utilisateur ne s'est pas fait reconnaître.
550	Fichier non trouvé ou protégé.

(voir annexe pour une liste complète)

### **Remarque**

On peut évidemment penser à dialoguer avec un serveur FTP en ligne de commande, comme avec un serveur SMTP ou POP3. De fait, on peut imaginer ceci :

```
vilvens@cure:~ > telnet u2 21
Trying 10.59.5.219...
Connected to u2.
Escape character is '^].
220 u2.wildness.loc FTP server (Version 6.5/OpenBSD, linux port 0.3.2) ready.
user vilvens
331 Password required for vilvens.
pass clixpataclix
230- Have a lot of fun...
230 User vilvens logged in.
```

On peut constater que tout va bien :

tcp	0	0 10.59.5.219: <b>21</b>	10.59.5.224:4315	<b>ESTABLISHED</b>
-----	---	--------------------------	------------------	--------------------

Il en sera encore ainsi tant que les commandes resteront de contrôle. Mais l'introduction d'une commande RETR provoquera l'arrêt du processus :

```
retr maxime.txt
425 Can't build data connection: Connection refused.
421 Timeout (900 seconds): closing control connection.
Connection closed by foreign host.
vilvens@cure:~ >
```

la deuxième connexion sera refusée, puisque les sockets ne sont pas configurées en REUSEADDR-REUSEPORT ...

## **5. Les principales commandes de paramétrage du transfert**

La manière dont les données demandées vont effectivement être transférées depuis le système local de fichiers du serveur vers celui du client dépend d'un certain nombre de facteurs. Les sites émetteurs et récepteurs devront en effet effectuer les transformations nécessaires entre la représentation standard et leur propre représentation interne. Les commandes suivantes contribuent à cet effort :

Commande	Description
<b>port &lt;adresse et port&gt;</b>	Permet de spécifier le port de données à utiliser pour l'établissement du canal de données. L'adresse et le port sont à spécifier en octets, sous la forme : a1, a2, a3, a4, p1, p2 où a1 désigne l'octet de poids fort.
<b>pasv</b>	Permet de demander au SERVER-DTP de se mettre à l'écoute d'un port de données et d'attendre une demande de connexion (par défaut, c'est l'inverse). La réponse précise l'adresse et le port sur lesquels le serveur s'est mis à l'écoute.

<b>type &lt;représentation&gt;</b>	Permet de spécifier le type de représentation des données utilisé : ascii ("A" - par défaut), ebcdic, image (ou binaire : suite continue de bytes de 8 bits – "I"), local (emploi de mots logiques dont la taille doit être spécifiée par un 2 <sup>ème</sup> paramètre).
<b>stru &lt;structure&gt;</b>	Pour spécifier la structure du fichier transféré : structure fichier (pas de structure = séquence d'octets – par défaut) → F, structure enregistrement → R ou structure paginée → P.
<b>mode</b>	Pour spécifier le mode de transfert : par flux → S ( <i>stream</i> ) ou par bloc → B

Maintenant que les dessous intimes de ftp nous sont un peu mieux connus, voyons comment l'utiliser en Java.

## 6. Un package Java pour un serveur FTP

On pourrait s'attendre à que Sun propose dans les packages javax tout ce qui faut en termes de classes serveur et client ftp – du genre javax.ftp et javax.ftp.server. Et pourtant, il n'en est rien (pour l'instant) ... ☺

Dans un tel cas, un bon réflexe est de chercher sur Internet si l'un des membres de la communauté Java n'aurait pas déjà fait le travail. Et, si oui, d'espérer qu'en plus, le fruit de son travail est un freeware ;-) ...

Précisément, dans le cas d'un serveur ftp, nous<sup>5</sup> avons trouvé le package espéré sur un site de dévelopeur. Le concepteur se nomme Rana Bhattacharya. Il n'est guère prolix quant à lui-même (modestie ? sécurité ? timidité ? détachement du monde ?). Tout ce que l'on sait, c'est que ce que l'on a téléchargé n'est pas seulement une librairie, mais aussi un serveur ftp utilisable tel quel, serveur qui fait à présent partie du projet Apache<sup>6</sup>. Les concepts de programmation que comportent ce serveur et les packages qui ont servi à l'écrire sont essentiellement les suivants :

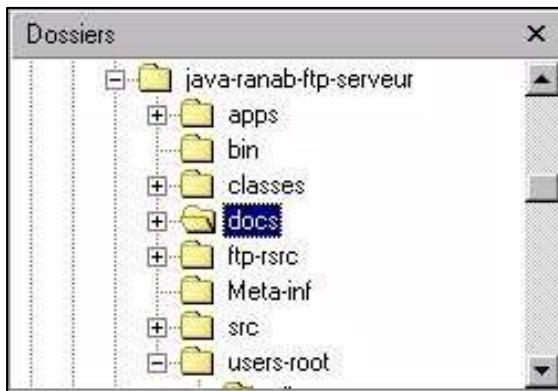


- ◆ 100% Java;
- ◆ multithread;
- ◆ le JDK1.3 au minimum est nécessaire (cela devrait aller, depuis le temps ;-));
- ◆ les fonctionnalités habituelles comme les transferts ASCII ou binaire, les droits, les time-out, les restrictions et adresses rejetées, les utilisateurs anonymes, sont supportées; en fait, la RFC 959 est implémentée;
- ◆ une base de données avec une table FTP\_USER peut être utilisée;
- ◆ les mots de passe des utilisateurs peuvent être hashés avec un algorithme MD5.

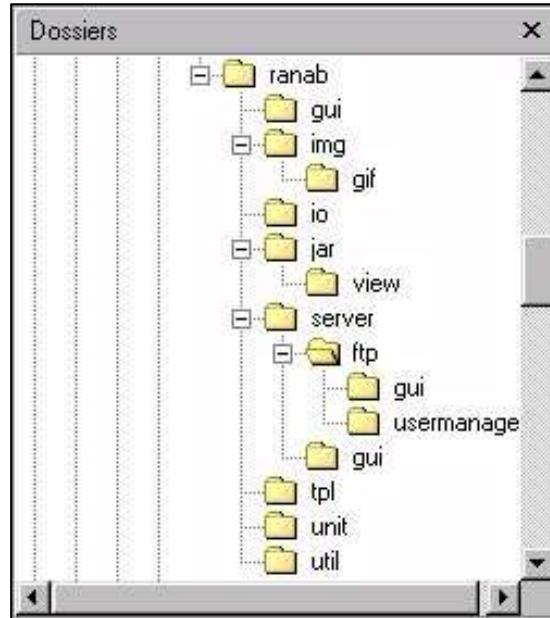
Mouais ... pas mal ;-) ... Pour ce qui nous intéresse, le package obtenu, appelé provisoirement **ranab**, correspond à notre attente et même bien plus. Outre les classes nécessaires à la confection d'un serveur ftp, il comporte bon nombre d'autre utilitaires. Outre les fichiers class, on obtient les sources, l'aide en page html, etc :

<sup>5</sup> merci à Gregory qui a débusqué le site ☺ !

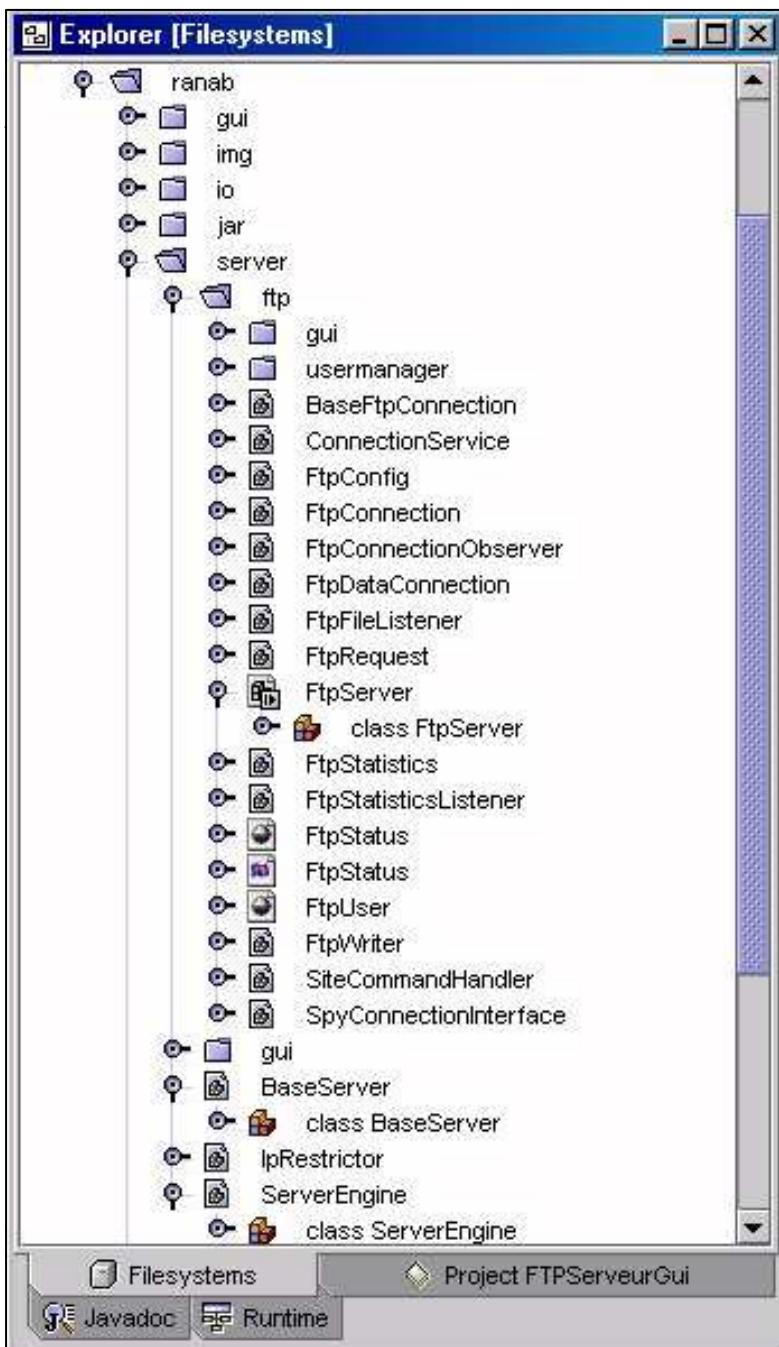
<sup>6</sup> <http://incubator.apache.org/projects/ftpserver.html>



Le package se structure de la manière suivante au sein du sous-répertoire *classes* de java-ranab-ftp-serveur qui est le répertoire où nous avons téléchargé le software (ce nom est bien entendu quelconque) :



Le répertoire classes ayant été monté dans un projet de Netbeans ou Sun ONE, on peut investiguer facilement les classes fournies :



De plus, les classes du package font usage d'un *fichier de configuration ftpd.conf* (ici, dans C:\java-ranab-ftp-serveur\apps\ftp\conf), qui pourra être pris en compte à chaque instantiation du serveur :

```
ftpd.conf
## Ftp server host name
## The default host is the localhost
#FtpServer.server.config.host=localhost

## Ftp server port number.
## Default FTP port is 21.
FtpServer.server.config.port=21
```

```
## Threadpool count.  
## This is the maximum number of simultaneous connections.  
## the default value is 25.  
FtpServer.server.config.thread=25  
  
## Maximum simultaneous logins.  
## Should be less than or equal to the threadpool count.  
## The default value is 20.  
FtpServer.server.config.login=20  
  
## Anonymous login allowed.  
## By default the server allows anonymous connections.  
FtpServer.server.config.anonymous=true  
  
## Maximum anonymous simultaneous logins.  
## Should be less than or equal to the number of server connections.  
## It will not be used if the sever does not allow anonymous login.  
## The default value is 10.  
FtpServer.server.config.anonymous.login=10  
  
...  
  
## Log level.  
## 0 -> Debug  
## 1 -> Information  
## 2 -> Warning  
## 3 -> Error  
## The default log level is 1 (information).  
FtpServer.server.config.log.level=1  
  
## Default root directory.  
## When you start the ftp server for the first time, two users (admin, anonymous)  
## will be created automatically. This directory will be their root directories.  
#FtpServer.server.config.root.dir=C:\\\\  
FtpServer.server.config.root.dir=C:\\\\java-ranab-ftp-serveur\\\\users-root  
  
...  
  
## Ftp resource directory. Ip restrictor file, log files etc. will be  
## stored in this directory or subdirectories.  
#FtpServer.server.config.data=C:\\\\myCode\\\\apps\\\\ftp\\\\resource  
FtpServer.server.config.data=C:\\\\java-ranab-ftp-serveur\\\\ftp-rsrc  
  
...  
## Ftp user manager class. Change it if you want to use a  
## different user manager class. All the user manager classes  
## have to be derived from ranab.server.ftp.usermodeler.UserManager  
## The default.usermodeler class is ranab.server.ftp.usermodeler.PropertiesUserManager.  
FtpServer.server.config.user.manager=ranab.server.ftp.usermodeler.PropertiesUserManager  
FtpServer.server.config.prop.encrypt=false
```

```
## You can add any config param for your user manager customization.  
## Following config params are for the database based user manager.  
## The user manager class ranab.server.ftp.usermodeler.DbUserManager uses these  
## params. This class is tested using MySQL and Oracle databases.  
#FtpServer.server.config.user.manager=ranab.server.ftp.usermodeler.DbUserManager  
## MySql configuration  
#FtpServer.server.config.db.driver=org.gjt.mm.mysql.Driver  
#FtpServer.server.config.db.url=jdbc:mysql://localhost/ftpDB  
#FtpServer.server.config.db.user=root  
#FtpServer.server.config.db.password=admin  
## Oracle configuration  
#FtpServer.server.config.db.driver=oracle.jdbc.driver.OracleDriver  
#FtpServer.server.config.db.url=jdbc:oracle:thin:@ranab:1521:ranab  
#FtpServer.server.config.db.user=user  
#FtpServer.server.config.db.password=password  
...
```

Une lecture rapide de ce fichier nous indique que :

- ◆ comme annoncé et tout serveur qui se respecte, le serveur que l'on peut créer ici est un serveur multithread;
- ◆ il utilise le port par défaut de ftp (21);
- ◆ deux utilisateurs existent dès le démarrage du serveur (admin et anonymous);
- ◆ leur répertoire accessible est C:\java-ranab-ftp-serveur\users-root (pour rappel, java-ranab-ftp-serveur est le répertoire où nous avons téléchargé le software);
- ◆ la gestion des utilisateurs peut se faire en utilisant la classe **PropertiesUserManager**; la consultation de son fichier source :

```
package ranab.server.ftp.usermodeler;  
...  
import ranab.server.ftp.FtpConfig;  
  
public class PropertiesUserManager extends UserManager  
{  
    private final static String PREFIX = "FtpServer.user.";  
    private final static String USER_PROP = "user.properties";  
    ...  
}
```

indique qu'un fichier **user.properties** (ici, dans C:\java-ranab-ftp-serveur\ftp-rsrc) est généré à chaque utilisation et détermine ce qu'un utilisateur peut faire :

<b>user.properties</b>
#Generated file - don't edit (please) #Thu Jul 11 09:53:26 CEST 2002 FtpServer.user.admin.enabled=true <b>FtpServer.user.admin.home=C:/java-ranab-ftp-serveur/users-root/</b> FtpServer.user.anonymous.idle=300

```
FtpServer.user.admin.password=admin
FtpServer.user.anonymous.download=4800
FtpServer.user.anonymous.write=false
FtpServer.user.admin.download=0
FtpServer.user.anonymous.home=C:/java-ranab-ftp-serveur/users-root/
FtpServer.user.admin.write=true
FtpServer.user.anonymous.upload=4800
FtpServer.user.admin.upload=0
FtpServer.user.admin.idle=300
FtpServer.user.anonymous.enabled=true
FtpServer.user.anonymous.password=
```

Il doit donc être possible de créer des utilisateurs taillés sur mesure ... mais il y a plus urgent. En effet, dans ce contexte, analysons de plus près les classes, et leur source java, qui nous seront nécessaires pour mettre un serveur Java-Ftp minimal sur pieds.

## 7. Un serveur FTP en Java

### 7.1 La classe FtpServer

La classe **FtpServer** se trouve dans le package ranab.server.ftp.

Elle dérive d'une classe abstraite (qui est presque un interface) **BaseServer** (du package ranab.server) qui prototypé *les actions classiques de tout serveur* :

```
public abstract int getServerPort()
public abstract java.lang.String getServerName()
public abstract void serveRequest(java.net.Socket dataSoc)
public void dispose()
public java.net.InetAddress getServerAddress()
```

La classe **FtpServer** elle-même a l'aspect suivant :

#### **FtpServer.java** ( © Rana Bhattacharya)

```
...
public class FtpServer extends BaseServer
{
    // server name
    public final static String NAME = "FTP";
    // thread pool
    private ThreadPool mThreadPool;
    // configuration object
    private FtpConfig mConfig;

    public FtpServer(File configFile) throws Exception
    {
        mConfig = new FtpConfig(configFile);
        mThreadPool = new ThreadPool(mConfig.getThreadCount());
        mThreadPool.start();
    }
    ...
}
```

```

public void serveRequest(Socket soc)
{
    BaseFtpConnection ftpCon = new FtpConnection(mConfig, soc);
    mThreadPool.add(ftpCon);
}
...
}

```

Nous pouvons ainsi voir que le constructeur

`public FtpServer(java.io.File configFile) throws java.lang.Exception`

réclame un fichier de configuration dont le chemin est créé bien naturellement au moyen d'un objet File. Ce fichier de configuration permet d'initialiser la variable membre mConfig qui est une instance de la classe **FtpConfig**, qui est en gros une liste de propriétés, donc une fille de la classe `java.util.Properties` du JDK. Dans notre cas, nous prendrons comme fichier de configuration celui qui a été téléchargé, quitte à le modifier ultérieurement.

## 7.2 L'aspect multithread

Ce même constructeur va créer ensuite un groupe de threads (un *pool*), instance d'une classe **ThreadPool** (du package ranab.util). Ces threads, dont le nombre est fixé d'après le fichier de configuration, sont créés au démarrage du serveur et mis en sommeil. L'un d'entre eux sera réveillé pour prendre en charge un client qui se sera connecté. Plus précisément, la classe possède une variable membre

`private PoolThread[] mThreads;`

Il s'agit en fait d'un tableau de threads, instances d'une classe imbriquée, dérivée de la classe **Thread** du JDK, appelée **PoolThread**. Ce qu'elle apporte de plus que sa mère standard, c'est l'existence d'une variable membre qui est une queue d'objets Runnable. On devine la suite : la méthode run() d'un thread va chercher continuellement un objet runnable de la queue et, si cette queue n'est pas vide, exécute la méthode run() de l'objet récupéré. En résumé :

### PoolThread.java ( © Rana Bhattacharya)

```

class PoolThread extends Thread
{
    private boolean mbStopRequest;
    private Queue mRunnableQueue;

    public PoolThread(Queue rnq) { mRunnableQueue = rnq; }

    public void dispose() { mbStopRequest = true; interrupt(); }
    public boolean isStopped() { return mbStopRequest; }
}

```

```

public void run()
{
    while(!mbStopRequest)
    {
        Runnable rn = (Runnable)mRunnableQueue.get();
        if(rn == null) { return; }
        try
        {
            rn.run();
        }
        catch (Throwable ex) { ex.printStackTrace(); }
    }
}

```

A priori, chaque thread possède sa propre queue. Mais, dans le cas de notre serveur FTP, tous les threads seront initialisés sur la même queue, comme on peut le voir dans le constructeur de ThreadPool. En résumé :

### ThreadPool.java ( © Rana Bhattacharya)

```

public class ThreadPool
{
    private int miCount;
    private boolean mbStopRequest;
    private Queue mRunnableQueue;
    private PoolThread[] mThreads;

    public ThreadPool(int count)
    {
        mThreads = new PoolThread[count];
        mRunnableQueue = new Queue(true);
        for(int i=0; i<mThreads.length; i++)
        {
            mThreads[i] = new PoolThread(mRunnableQueue);
        }
    }

    public synchronized void start()
    {
        if(mbStopRequest)
        { throw new IllegalStateException("Pool has been stopped"); }
        for(int i=0; i<mThreads.length; i++)
        {
            mThreads[i].start();
        }
    }
}

```

```

public synchronized void add(Runnable rn)
{
    if(mbStopRequest)
        { throw new IllegalStateException("Pool has been stopped"); }
    mRunnableQueue.put(rn);
}

public synchronized void dispose()
{
    mbStopRequest = true;
    for(int i=0; i<mThreads.length; i++)
    {
        mThreads[i].dispose();
    }
    mRunnableQueue.clear();
}

public boolean isStopped() { return mbStopRequest; }

///////////////////////////// Inner Class - Thread Pool Thread ///////////////////
class PoolThread extends Thread
{
    ...
}

```

### **7.3 Une queue moniteur**

La queue est une instance de la classe **Queue** (du package ranab.util). Bien sûr, elle encapsule un container du JDK :

```
private LinkedList mList = new LinkedList();
```

la classe **LinkedList** étant un container standard de java.util à partir du JDK 1.2. Ce que cette classe apporte de plus, si son constructeur a reçu un paramètre booléen à true, c'est que sa méthode de retrait

```
public java.lang.Object get()
```

met le thread qui l'exécute en sommeil au moyen d'un **wait()** si la file est vide. Bien sûr, la méthode d'ajout

```
public void put(java.lang.Object obj)
```

effectue un **notifyAll()** pour réveiller un thread en attente. En bon programmeurs Java ;-), on se doute que les deux méthodes **get()** et **put()** sont synchronized : l'objet queue est donc un moniteur ! En résumé :

### **Queue.java** ( © Rana Bhattacharya)

```

public class Queue
{
    private LinkedList mList = new LinkedList();
    private boolean mbWait;
    private int miMaxSize = 0;

    public Queue(boolean bWait) { mbWait = bWait; }

    public synchronized Object get()
    {
        if(mbWait)
        {
            while(mList.size() == 0)
            {
                try { wait(); }
                catch(InterruptedException ex) { return null; }
            }
            return mList.removeLast();
        }
        else
        {
            return mList.removeLast();
        }
    }

    public synchronized void put(Object obj)
    {
        if(obj == null)
            { throw new NullPointerException("Queue element cannot be null"); }
        if (miMaxSize <= 0 || mList.size() < miMaxSize)
        {
            mList.addFirst(obj);
            notifyAll();
        }
    }

    ...
}

```

#### **7.4 La classe de connexion FTP**

Jusqu'à présent, tout ce que nous avons découvert s'applique à un serveur quelconque, pas seulement à un serveur FTP. La spécification de ce protocole spécifique apparaît dans le fait que la méthode de FTPServer :

```
public void serveRequest(java.net.Socket soc)
```

va placer dans la queue à destination des threads, en cas de connexion, un objet instance de **FtpConnection** (du package ranab.server.ftp), classe dérivée de **BaseFtpConnection** qui implémente l'interface **Runnable** :

```
public void serveRequest(Socket soc)
{
    BaseFtpConnection ftpCon = new FtpConnection(mConfig, soc);
    mThreadPool.add(ftpCon);
}
```

Comme on l'a vu, cet ajout va réveiller un thread de service qui exécutera la méthode run() de l'objet trouvé dans la queue encapsulée dans l'objet ThreadPool, donc ici de l'objet **FtpConnection**. Que fera cette méthode run() ici ?

1) Examinons tout d'abord la classe de base :

**BaseFtpConnection.java** ( © Rana Bhattacharya)

```
public class BaseFtpConnection implements Runnable, StreamConnectorObserver
{
    protected final static Class[] METHOD_INPUT_SIG =
        new Class[] {FtpRequest.class, FtpWriter.class};
    protected FtpConfig mConfig = null;
    protected FtpDataConnection mDataConnection = null;
    protected Socket mControlSocket = null;
    protected FtpWriter mWriter = null;
    protected boolean mbStopRequest = false;

    public BaseFtpConnection(FtpConfig ftpConfig, Socket soc) {...}

    public void run()
    {
        InetAddress clientAddress = mControlSocket.getInetAddress();
        mDataConnection = new FtpDataConnection(mConfig);
        mConfig.getConnectionService().newConnection(this);
        ...
        do
        {
            // lecture de la requête de transfert
            //dans la chaîne de caractères commandLine
            FtpRequest request = new FtpRequest(commandLine);
            ...
            // execute command
            service(request, mWriter);
        }
        while(!mbStopRequest);
        ...
    }
}
```

```

public void service(FtpRequest request, FtpWriter writer) throws IOException
{
    try
    {
        String metName = "do" + request.getCommand();
        Method actionMet =
            getClass().getDeclaredMethod(metName, METHOD_INPUT_SIG);
        actionMet.invoke(this, new Object[] {request, writer});
    }
    catch(NoSuchMethodException ex) {...}
    catch(InvocationTargetException ex) { ... }
}
...
}

```

On constate donc que la méthode run() appelle, par introspection, une méthode dont le nom est la chaîne de caractère lue dans la commande reçue par réseau, nom précédé de "do".

**2)** La classe **FtpConnection** ne redéfinit pas la méthode run() héritée. En fait, *les méthodes de cette classe matérialisent les commandes FTP proprement dites*. Elles ont toutes un prototype similaire :

do<commande FTP> (FtpRequest requête, FtpWriter sortie) throws IOException.

qui est précisément le format des méthodes appelées par la méthode run() héritée. Ainsi, par exemple :

**doRETR** (FtpRequest request, FtpWriter out)

matérialise la commande : RETR <fichier avec chemin éventuel>  
qui demande au SERVER-DTP de transmettre le fichier demandé au USER-DTP.

a) Le premier paramètre **FtpRequest** ne fait jamais qu'encapsuler la requête FTP en ligne de commande mais en distinguant ses composantes :

### FtpRequest.java ( © Rana Bhattacharya)

```

public class FtpRequest
{
    private String mstLine    = null;
    private String mstCommand = null;
    private String mstArgument = null;

    public FtpRequest(String commandLine)
    {
        mstLine = commandLine.trim();
        parse();
    }
    ...
}

```

b) Le second paramètre n'est jamais qu'un objet Writer qui sera initialisé sur le flux de sortie de la socket :

**FtpWriter.java** ( © Rana Bhattacharya)

```
public class FtpWriter extends Writer
{
    private OutputStreamWriter mOriginalWriter;
    ...
    private FtpConfig mConfig;

    public FtpWriter(Socket soc, FtpConfig config) throws IOException
    {
        mOriginalWriter = new OutputStreamWriter(soc.getOutputStream());
        mConfig = config;
    }
    ...
}
```

On peut alors décoder sans trop de difficulté le code de cette méthode doRETR() :

**FtpConnection.java** ( © Rana Bhattacharya)

```
public class FtpConnection extends BaseFtpConnection
{
    public FtpConnection(FtpConfig cfg, Socket soc)
    {
        super(cfg, soc);
    }
    ...
    public void doRETR(FtpRequest request, FtpWriter out) throws IOException
    {
        String fileName = request.getArgument();
        fileName = mUser.getVirtualDirectory().getAbsoluteName(fileName);
        String physicalName =
            mUser.getVirtualDirectory().getPhysicalName(fileName);
        File requestedFile = new File(physicalName);
        String args[] = {fileName};

        // check permission
        if(!mUser.getVirtualDirectory().hasReadPermission(physicalName, true))
        {
            out.write(mFtpStatus.getResponse(550, request, mUser, args));
            return;
        }

        // now transfer file data
        out.write(mFtpStatus.getResponse(150, request, mUser, null));
        InputStream is = null;
        OutputStream os = null;
```

```

try
{
    Socket dataSoc = mDataConnection.getDataSocket();
    if (dataSoc == null)
    {
        out.write(mFtpStatus.getResponse(550, request, mUser, args));
        return;
    }

    os = mUser.getOutputStream(dataSoc.getOutputStream());

    RandomAccessFile raf = new RandomAccessFile(requestedFile, "r");
    raf.seek(skipLen);
    is = new FileInputStream(raf.getFD());

    StreamConnector msc = new StreamConnector(is, os);
    ...
    msc.connect();
}
catch(IOException ex)
{
    out.write(mFtpStatus.getResponse(425, request, mUser, null));
}
finally
{
    IoUtils.close(is); IoUtils.close(os);
    mDataConnection.reset();
}
}
...
}
}

```

Par héritage, cette classe possède une variable membre

protected ranab.server.ftp.FtpDataConnection ***mDataConnection***;

La classe ainsi instanciée, ***FtpDataConnection***, sert essentiellement à encapsuler les sockets et port nécessaires, comme on peut le voir avec ses variables membres :

```

private FtpConfig mConfig = null;
private Socket mDataSoc = null;
private ServerSocket mServSoc = null;
private InetAddress mAddress = null;
private int miPort = 0;
private boolean mbPort = false;
private boolean mbPasv = false;

```

Mais comment le transfert est-il effectivement réalisé ? Par l'objet ***StreamConnector*** qui, en appelant sa méthode ***connect()***, lance un thread qui va réaliser le transfert en question ...

## 7.5 La classe StreamConnector

Le seul point névralgique est en fait l'utilisation de la classe ***StreamConnector*** (du package ranab.io) qui, en fait, réalise effectivement le transfert :

### StreamConnector.java ( © Rana Bhattacharya)

```
public class StreamConnector implements Runnable
{
    private InputStream mInStream;
    private OutputStream mOutStream;
    ...
    public StreamConnector(InputStream in, OutputStream out)
    {
        mInStream = in; mOutStream = out;
    }
    ...
    public void connect()
    {
        // error test
        ...
        // now connect
        if(mbThreaded) { new Thread(this).start(); }
        else { run(); }
    }

    public void run()
    {
        long startTime = System.currentTimeMillis();
        mConThread = Thread.currentThread();

        InputStream in = mInStream;
        OutputStream out = mOutStream;

        byte[] buff = new byte[4096];

        if(mbBuffered)
        {
            in = IoUtils.getBufferedInputStream(in);
            out = IoUtils.getBufferedOutputStream(out);
        }

        try
        {
            while(! (mbStopRequest || mConThread.isInterrupted()) )
            {
                // check transfer rate
                ...
                // read data
                int count = in.read(buff);
                if(count == -1) { break; }
            }
        }
    }
}
```

```

        // write data
        out.write(buff, 0, count);
        ...
        mTransferSize += count;
    }
    out.flush();
}
catch(Exception ex) {...}
finally
{
    synchronized(this)
    {
        mbStopRequest = true;
        IoUtils.close(in); IoUtils.close(out);
        mConThread = null;
    }
}
}

```

## 7.6 Un moteur à serveurs

Pour terminer, on peut se demander comment la méthode `serveRequest()` est invoquée. Elle l'est par l'intermédiaire d'un objet instanciant la classe ***ServerEngine*** (du package `ranab.server`). Comme son nom l'indique, elle permet de lancer un serveur quelconque – dans notre cas, ce sera un serveur FTP. Il s'agit encore une fois d'un thread, plus précisément d'un objet `Runnable` :

<b>ServerEngine.java</b> ( © Rana Bhattacharya)
<pre> public class ServerEngine implements Runnable {     private Thread mRunner = null;     private ServerSocket mServerSocket = null;     private BaseServer mServer = null;      public ServerEngine(BaseServer server)     {         mServer = server;     }      public synchronized void startServer() throws IOException     {         if (mRunner == null)         {             InetAddress serverAddr = mServer.getServerAddress();             if (serverAddr == null)             {                 mServerSocket =                     new ServerSocket(mServer.getServerPort(), 100);             }         }     } } </pre>

```
        else
        {
            mServerSocket =
                new ServerSocket(mServer.getServerPort(), 100, serverAddr);
        }

        mRunner = new Thread(this);
        mRunner.start();
        System.out.println("Started " + mServer.getServerName());
    }
}

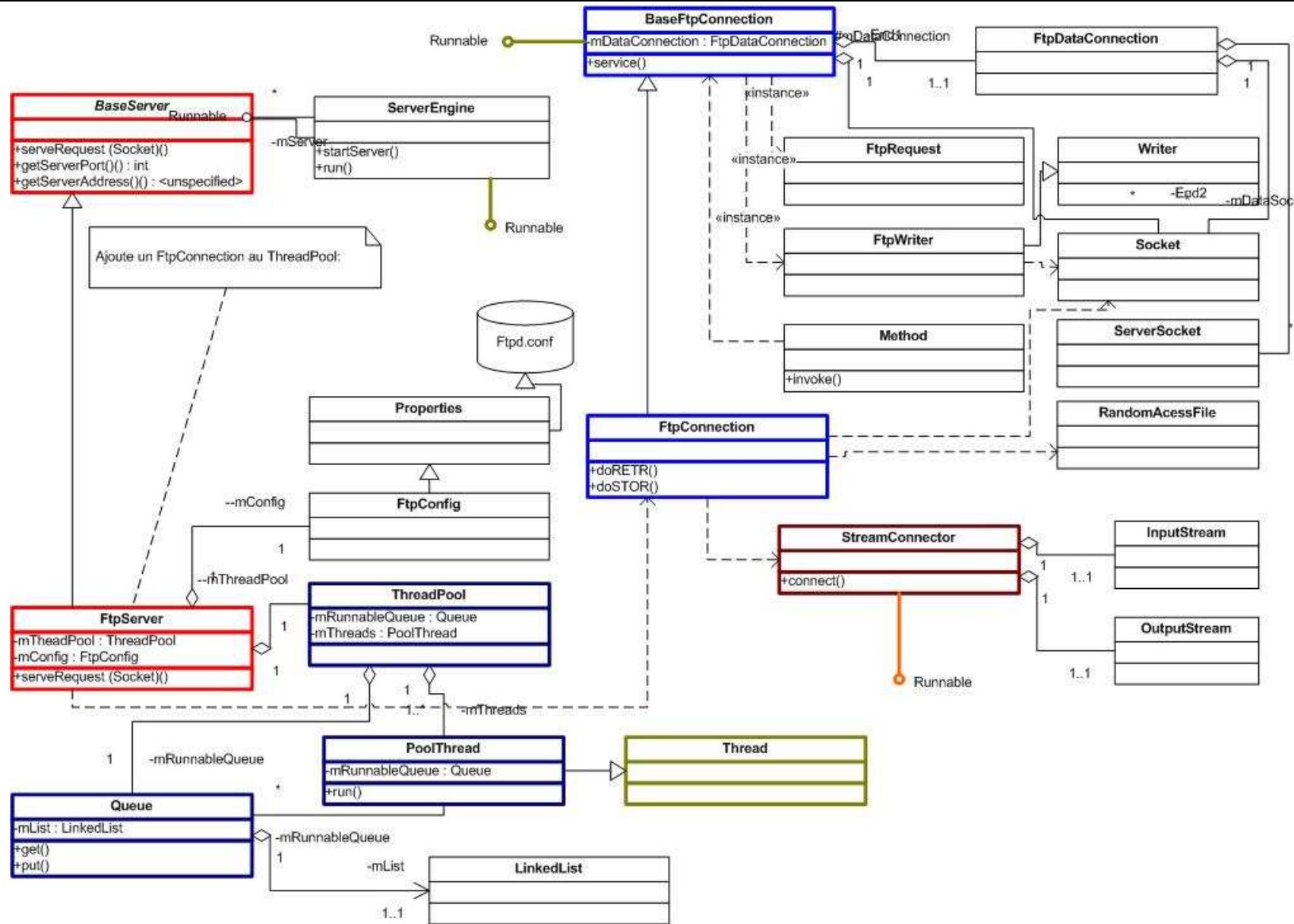
public void run()
{
    while (mRunner != null)
    {
        try
        {
            Socket datasocket = mServerSocket.accept();
            mServer.serveRequest(datasocket);
        }
        catch (Exception ex) {}
    }
}
}
```

On peut constater qu'un thread est créé et que sa méthode run() devra traiter les requêtes adressées au serveur (passé comme paramètre au constructeur) au moyen de la méthode serveRequest qui utilise une socket préalablement créée.

Ouf !

## **7.7 Le diagramme de classes UML**

De manière synthétique et simplifiée, on peut résumer l'architecture selon le diagramme de classes UML suivant :



## 8. Un minuscule serveur FTP maison

Armé de ce package, l'écriture d'un petit serveur ftp pour utilisateur anonyme n'est pas difficile. En effet, il suffit :

- ◆ de déclarer trois variable membres

```
private File fichierConfig;  
private FtpServer ftps;  
private ServerEngine seng;
```

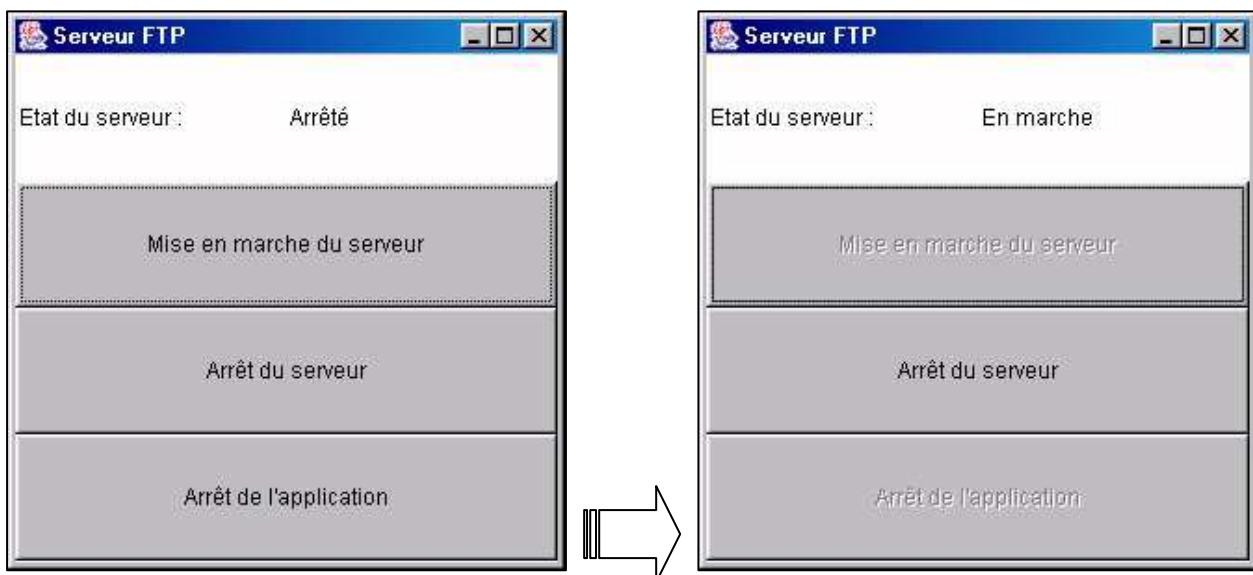
- ◆ de créer une instance d'un FtpServer à qui l'on passe le fichier de configuration (ici, on a pris celui de départ) :

```
fichierConfig = new File("c:\\java-ranab-ftp-serveur\\APPS\\FTP\\CONF\\ftpd.conf");  
ftps = new FtpServer(fichierConfig);
```

- ◆ d'instancier un moteur à serveurs en lui confiant notre serveur FTP :

```
seng = new ServerEngine(ftps);  
seng.startServer();
```

C'est tout ! En ajoutant un interface GUI minimum, l'exécution de notre serveur au démarrage devrait ressembler à ceci :



Donc finalement (le répertoire c:\\java-ranab-ftp-serveur\\classes a été monté dans le projet) :

### FTPServeurGuiFenApp.java

```
/*  
 * FTPServeurGuiFenApp.java  
 *  
 * Created on 10 juillet 2002, 14:40  
 */
```

```
/*
 * @author Vilvens
 */

import java.awt.*;
import java.io.*;
import ranab.server.*;
import ranab.server.ftp.*;

public class FTPServeurGuiFenApp extends java.awt.Frame
{
    private File fichierConfig;
    private FtpServer ftps;
    private ServerEngine seng;

    /** Creates new form FTPServeurGuiFenApp */
    public FTPServeurGuiFenApp()
    {
        initComponents();
    }

    private void initComponents
```

```
BMarche.setLabel("Mise en marche du serveur");
BMarche.setName("BMarche");
BMarche.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BMarcheActionPerformed(evt);
    }
});

add(BMarche);

BArret.setLabel("Arr\u00e9at du serveur");
BArret.setName("BArret");
BArret.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BArretActionPerformed(evt);
    }
});

add(BArret);

BArretApplic.setLabel("Arr\u00e9at de l'application");
BArretApplic.setName("BArretApplic");
BArretApplic.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BArretApplicActionPerformed(evt);
    }
});

add(BArretApplic);

pack();
java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
setSize(new java.awt.Dimension(300, 300));
setLocation((screenSize.width-300)/2,(screenSize.height-300)/2);
}

private void BArretApplicActionPerformed(java.awt.event.ActionEvent evt)
{
    // Arret de l'application
    System.exit(0);
}

private void BArretActionPerformed(java.awt.event.ActionEvent evt)
{
    // Arret du serveur
    BMarche.setEnabled(true);
    BArret.setEnabled(false);
    BArretApplic.setEnabled(true);
    seng.stopServer();
    System.out.println("ServerEngine arr\u00eate");
}
```

```
LEtatValeur.setText("Arrêté");
}

private void BMarcheActionPerformed(java.awt.event.ActionEvent evt)
{
    // Mise en marche du serveur
    BMarche.setEnabled(false);
    BArret.setEnabled(true);
    BArretAplic.setEnabled(false);
    LEtatValeur.setText("En marche");
    try
    {
        fichierConfig = new File("c:\\java-ranab-ftp-serveur\\APPS\\FTP\\CONF\\ftpd.conf");
        System.out.println("Objet File créé");
        ftps = new FtpServer(fichierConfig);
        System.out.println("Objet FTPServer créé");
        seng = new ServerEngine(ftps);
        System.out.println("Objet ServerEngine créé");
        seng.startServer();
        System.out.println("ServerEngine démarré");
    }
    catch (Exception e)
    {
        System.out.println("oh oh - " + e.getMessage());
    }
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) { System.exit(0); }

/**
 * @param args the command line arguments
 */
public static void main(String args[])
{
    new FTPServeurGuiFenApp().show();
}

// Variables declaration - do not modify
private java.awt.Panel panelEtat;
private java.awt.Label LEtatTexte;
private java.awt.Label LEtatValeur;
private java.awt.Button BMarche;
private java.awt.Button BArret;
private java.awt.Button BArretAplic;
// End of variables declaration
}
```

Une fois ce serveur démarré (ici, sur une machine Windows d'adresse 192.168.2.2 dans un réseau local), on peut constater son écoute sur le port 21 :

```
C:\VILVENS>netstat -an
```

Connexions actives

Proto	Adresse locale	Adresse distante	État
TCP	<b>0.0.0.0:21</b>	<b>0.0.0.0:0</b>	<b>LISTENING</b>
TCP	0.0.0.0:8558	0.0.0.0:0	LISTENING
...			
TCP	127.0.0.1:1025	0.0.0.0:0	LISTENING
TCP	192.168.2.2:137	0.0.0.0:0	LISTENING
TCP	192.168.2.2:138	0.0.0.0:0	LISTENING
TCP	192.168.2.2:139	0.0.0.0:0	LISTENING
UDP	0.0.0.0:7318	*:*	
...			

Une fois le client (ici, sur une machine Windows d'adresse 192.168.2.1 dans un réseau local – bien sûr, il nous reste à écrire ce client ...) connecté et le transfert entamé, on peut constater qu'un thread a été créé et détecter l'existence d'une deuxième connexion de transfert :

```
C:\VILVENS>netstat -an
```

Connexions actives

Proto	Adresse locale	Adresse distante	État
TCP	<b>0.0.0.0:21</b>	<b>0.0.0.0:0</b>	<b>LISTENING</b>
TCP	0.0.0.0:8558	0.0.0.0:0	LISTENING
...			
TCP	<b>192.168.2.2:1032</b>	<b>192.168.2.1:1035</b>	<b>TIME_WAIT</b>
TCP	192.168.2.2:1033	0.0.0.0:0	LISTENING
TCP	<b>192.168.2.2:1033</b>	<b>192.168.2.1:1036</b>	<b>ESTABLISHED</b>
TCP	<b>192.168.2.2:21</b>	<b>192.168.2.1:1034</b>	<b>ESTABLISHED</b>
...			

La console Java indique que tout se passe bien :

```
Objet File créé
Objet FTPServer créé
Objet ServerEngine créé
Started FTP
ServerEngine démarré
```

Le serveur accepte un client anonyme (que nous allons détailler dans les paragraphes suivants) et qui souhaite télécharger un fichier fichier.txt se trouvant dans un sous-répertoire "vilvens" de users-root. Dans le *fichier de log* (qui se trouve dans C:\java-ranab-ftp-serveur\ftp-rsrc), on trouve effectivement les signes de la réussite de l'opération :

<b>log</b>
[08/14 19:06:16 (INF)] Loaded user data file C:\java-ranab-ftp-serveur\ftp-rsrc\user.properties
[08/14 19:06:17 (INF)] Configuration loaded c:\java-ranab-ftp-serveur\APPS\FTP\CONF\ftpd.conf
[08/14 19:07:30 (INF)] Handling new request from 192.168.2.1
[08/14 19:07:30 (INF)] New connection from 192.168.2.1
[08/14 19:07:30 (INF)] Anonymous connection - 192.168.2.1 – claud.e.vilvens@prov-liege.be
[08/14 19:07:30 (INF)] User login - 192.168.2.1 - anonymous
[08/14 19:07:31 (INF)] File download : anonymous - C:\java-ranab-ftp-serveur\users-root\vilvens\fichier.txt
[08/14 19:08:25 (INF)] Closing connection service.
[08/14 19:08:25 (INF)] Closing user manager.
[08/14 19:08:25 (INF)] Closing message queue.
[08/14 19:08:25 (INF)] Closing log file.

Il nous reste à écrire un client FTP pour ce beau serveur ...

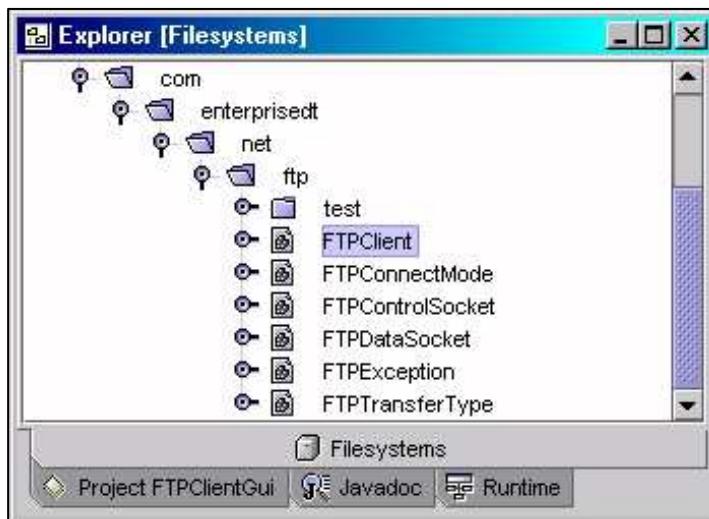
## 9. Un package Java pour un client FTP

A nouveau, on peut avoir le réflexe de chercher sur Internet si l'un des membres de la communauté Java, un autre ou le même que le précédent, n'aurait pas déjà fait le travail de la conception d'un client FTP en Java, avec généreuse offrande en freeware ;-) ...

C'est le cas : au moment où ces lignes sont écrites, on trouve<sup>1</sup> le package espéré sur <http://www.enterprisedt.com>

Le concepteur se nomme Bruce Blackshaw; il appartient à une société de consultance informatique de Londres appelée Enterprise Distributed Technologies Ltd. Cette société travaille dans tous les domaines actuels : Java et J2E, C++, Web, CORBA, ... sur des plates-formes Unix, Linux et Windows.

Le package proposé par cette société est disponible en version 1.1; le JDK1.3 au minimum est nécessaire. Il correspond à notre attente. Outre les fichiers class compactés dans un fichier jar (attention à l'extraction du fichier zip de base qui réclame la version 8.\* de Winzip !), on obtient les sources et l'aide en page html. Le fichier jar ayant été monté dans un projet, on peut investiguer facilement les classes fournies :



<sup>1</sup> encore merci à Gregory qui a également débusqué ce site ☺ !!

## 10. Un client FTP en Java

### 10.1 La classe **FTPClient**

La classe essentielle est ici la classe **FTPClient**. Celle-ci

- ◆ encapsule un certain nombre de variables membres qui
  - permettent le contrôle et le transfert (ce sont donc des sockets, ou plutôt des objets qui encapsulent des sockets);
  - mémorisent les paramètres de la connexion (mode, type, etc);
- ◆ comporte autant de méthodes qu'il y a de commandes FTP supportées, avec des versions dédicacées.

Analysons les éléments principaux :

#### **FTPClient.java** ( © Bruce P. Blackshaw)

```
package com.enterprisedt.net.ftp;

import java.io.IOException;
...
import java.net.InetAddress;
import java.net.Socket;
import java.util.Vector;
import java.util.Properties;

public class FTPClient
{
    ...
    private FTPControlSocket control = null;
    private FTPDataSocket data = null;
    private int timeout = 0;
    private FTPTransferType transferType = FTPTransferType.ASCII;
    private FTPConnectMode connectMode = FTPConnectMode.PASV;

    public FTPClient(String remoteHost) throws IOException, FTPException
    {
        control = new FTPControlSocket(remoteHost);
    }

    // Autres constructeurs
    ...
    public void setTimeout(int millis) throws IOException
    {

        this.timeout = millis;
        control.setTimeout(millis);
    }

    public void setConnectMode(FTPConnectMode mode) {connectMode = mode; }
```

```

public void login(String user, String password) throws IOException, FTPException
{
    String response = control.sendCommand("USER " + user);
    control.validateReply(response, "331");
    response = control.sendCommand("PASS " + password);
    control.validateReply(response, "230");
}

public void user(String user) throws IOException, FTPException
{
    String reply = control.sendCommand("USER " + user);
    String[] validCodes = {"230", "331"};
    control.validateReply(reply, validCodes);
}

public void password(String password) throws IOException, FTPException
{
    String reply = control.sendCommand("PASS " + password);
    String[] validCodes = {"230", "202"};
    control.validateReply(reply, validCodes);
}

public void get(String localPath, String remoteFile) throws IOException, FTPException
{
    if (getType() == FTPTransferType.ASCII)
    {
        getASCII(localPath, remoteFile);
    }
    else
    {
        getBinary(localPath, remoteFile);
    }

    String[] validCodes2 = {"226", "250"};
    String reply = control.readReply();
    control.validateReply(reply, validCodes2);
}

private void initGet(String remoteFile) throws IOException, FTPException
{
    data = control.createDataSocket(connectMode);
    data.setTimeout(timeout);
    String reply = control.sendCommand("RETR " + remoteFile);
    String[] validCodes1 = {"125", "150"};
    control.validateReply(reply, validCodes1);
}

private void getASCII(String localPath, String remoteFile)
    throws IOException, FTPException
{
    BufferedWriter out = new BufferedWriter(new FileWriter(localPath));
}

```

```

initGet(remoteFile);

// get an character input stream to read data from ... AFTER we
// have the ok to go ahead AND AFTER we've successfully opened a
// stream for the local file
LineNumberReader in = new LineNumberReader(
    new InputStreamReader(data.getInputStream()));

// read/write a line at a time
String line = null;
while ((line = in.readLine()) != null)
{
    out.write(line, 0, line.length());
    out.newLine();
}
out.close();

try
{
    in.close(); data.close();
}
catch (IOException ignore) {}
}

private void getBinary(String localPath, String remoteFile)
    throws IOException, FTPException
{
    BufferedOutputStream out =
        new BufferedOutputStream(new FileOutputStream(localPath, false));

    initGet(remoteFile);
    // get an input stream to read data from ... AFTER we have
    // the ok to go ahead AND AFTER we've successfully opened a
    // stream for the local file
    BufferedInputStream in =
        new BufferedInputStream(new DataInputStream(data.getInputStream()));

    int chunksize = 4096;
    byte [] chunk = new byte[chunksize];
    int count;
    while ((count = in.read(chunk, 0, chunksize)) >= 0)
    {
        out.write(chunk, 0, count);
    }
    out.close();

    try
    {
        in.close(); data.close();
    }
}

```

```

        catch (IOException ignore) {} }

    public void setType(FTPTransferType type)
        throws IOException, FTPException
    {
        String typeStr = FTPTransferType.ASCII_CHAR;
        if (type.equals(FTPTransferType.BINARY))
            typeStr = FTPTransferType.BINARY_CHAR;
        String reply = control.sendCommand("TYPE " + typeStr);
        control.validateReply(reply, "200");
        transferType = type;
    }

    public void quit() throws IOException, FTPException
    {
        String reply = control.sendCommand("QUIT" + FTPControlSocket.EOL);
        control.validateReply(reply, "221");
        control.logout();
        control = null;
    }
}

```

Il est clair que tout dépend des objets control et data ...

## 10.2 Les classes sockets

1) La classe **FTPControlSocket** encapsule évidemment un objet Socket et se charge de créer et de gérer la connexion sur le port 21 du serveur :

```
private static final int CONTROL_PORT = 21;
```

Servant de support au contrôle, elle est instanciée dans le constructeur. Cette classe encapsule tout naturellement un objet

```
private Socket controlSock;
```

initialisé dans les constructeurs :

```
public FTPControlSocket(String remoteHost) throws IOException, FTPException
{
    this(remoteHost, CONTROL_PORT);
}
```

```
public FTPControlSocket(String remoteHost, int controlPort)
    throws IOException, FTPException
{
    controlSock = new Socket(remoteHost, controlPort);
    initStreams();
    validateConnection();
}
```

avec une méthode `initStreams()` qui crée classiquement les flux associés à la socket (ici, en l'occurrence, des instances des classes `BufferedReader` et `OutputStreamWriter` du JDK) :

```
private void initStreams() throws IOException
{
    // input stream
    InputStream is = controlSock.getInputStream();
    reader = new BufferedReader(new InputStreamReader(is));

    // output stream
    OutputStream os = controlSock.getOutputStream();
    writer = new OutputStreamWriter(os);
}
```

On peut vérifier le rôle d'intermédiaire du contrôle de l'objet `control`, instance de `FTPControlSocket`, ainsi créé en observant dans, par exemple, la méthode `login` :

```
String response = control.sendCommand("USER " + user);
control.validateReply(response, "331");
```

où, tout naturellement, on constate que la méthode `sendCommand` écrit sur le flux de la socket :

```
String sendCommand(String command) throws IOException
{
    if (debugResponses)
        System.out.println("---> " + command);
    // send it
    writer.write(command + EOL);
    writer.flush();
    // and read the result
    return readReply();
}
```

On remarquera la trace laissée si l'on travail en mode debug (la méthode `debugResponses(true)` enregistre ce fait).

**2) La classe `FTPDataSocket`** correspond bien sûr au canal des données transférées, encapsulant les objets `ServerSocket` et `Socket` (les classes du JDK) nécessaires selon que le serveur est actif ou passif :

```
private ServerSocket activeSocket;
private Socket passiveSocket;
```

Elle est instanciée dans la méthode `initGet()` du client. Elle ne possède aucune méthode publique et est exclusivement utilisée au sein du package. En fait, elle se limite à créer et gérer les flux associés à la socket effectivement utilisée. Ainsi sa méthode :

```
InputStream getInputStream() throws IOException
{
    if(passiveSocket != null)
    {
        return passiveSocket.getInputStream();
    }
    else
    {
        passiveSocket = activeSocket.accept();
        return passiveSocket.getInputStream();
    }
}
```

L'objet `FTPClient` obtient un objet `data`, instance de `FTPDataSocket`, par l'appel dans la méthode `initGet()` qui encapsule un appel de la méthode :

```
FTPDataSocket createDataSocket(FTPConnectMode connectMode)
    throws IOException, FTPException
{
    if (connectMode == FTPConnectMode.ACTIVE)
    {
        return new FTPDataSocket(createDataSocketActive());
    }
    else
    {
        // PASV
        return new FTPDataSocket(createDataSocketPASV());
    }
}
```

où, évidemment, `createDataSocketActive()` retourne un objet `ServerSocket` tandis que `createDataSocketPASV()` renvoie un objet `Socket`.

### **10.3 Les classes annexes**

La classe d'exception `FTPException` est dérivée de la classe `Exception` du JDK. Elle comporte en plus la méthode :

```
public int getReplyCode()
```

qui fournit le code de retour de la dernière commande FTP, si il existe.

Les deux classes `FTPConnectMode` et `FTPTransferType` ne sont que deux hôtes pour constantes de classe désignant respectivement le mode de connexion du serveur :

```
public class FTPConnectMode
{
    private static String cvsId
        = "$Id: FTPConnectMode.java,v 1.1 2001/10/09 20:53:46 bruceb Exp $";
    public static FTPConnectMode ACTIVE = new FTPConnectMode();
    public static FTPConnectMode PASV = new FTPConnectMode();

    private FTPConnectMode() { }
}
```

et la caractérisation du mode de transfert :

```
public class FTPTransferType
{
    private static String cvsId
        = "$Id: FTPTransferType.java,v 1.3 2001/10/09 20:54:08 bruceb Exp $";

    public static FTPTransferType ASCII = new FTPTransferType();
    public static FTPTransferType BINARY = new FTPTransferType();
    static String ASCII_CHAR = "A";
    static String BINARY_CHAR = "I";
    private FTPTransferType() { }
}
```

## 11. Un petit client FTP maison

Sur base du package qui vient d'être étudié, nous allons programmer ici un petit client FTP anonyme de type GUI, avec comme variables membres :

```
String host = "192.168.2.2",
      user = "anonymous",
      password="cvilvens@prov-liege.be";
```

La démarche est simple :

- ♦ on instancie un objet FTPClient pour la machine serveur visée :

```
FTPClient ftpc = null;
ftpc = new FTPClient(host);
```

- ♦ le client s'authentifie sur le serveur en appelant la méthode login :

```
ftpc.login(user, password);
```

- ♦ on paramétrise la connexion :

```
ftpc.setType(FTPTransferType.BINARY);
ftpc.setConnectMode(FTPConnectMode.PASV);
```

- ♦ on récupère dans le GUI le nom du fichier visé et son chemin dans le répertoire de l'utilisateur :

```
String nomFichier = TFNomFichier.getText();
String nomRepertoire = TFNomRepertoire.getText();
String nomFichierOrigine = System.getProperty("file.separator") + nomRepertoire +
System.getProperty("file.separator") + nomFichier;
String nomFichierCible = "copie-" + nomFichier;
ftpc.get(nomFichierCible, nomFichierOrigine);
```

On remarquera l'utilisation du séparateur de fichier propre au système de la machine au moyen de la méthode **getProperty()**.

- ◆ on termine la connexion :

```
ftpc.quit();
```

Le programme complet se contente de demander le nom du fichier visé et le nom du répertoire. Il est donc fort simple :

### FTPClientGuiFenApp.java

```
/*
 * FTPClientGuiFenApp.java
 * Created on 11 juillet 2002, 10:26
 */
/** 
 * @author Vilvens
 */

import com.enterprisedt.net.ftp.*;
import java.io.*;

public class FTPClientGuiFenApp extends java.awt.Frame
{
    String      host = "192.168.2.2",
                user = "anonymous",
                password="claude.vilvens@prov-liege.be";

    public FTPClientGuiFenApp()
    {
        initComponents();
    }

    private void initComponents
```

```
panelFichier.setLayout(new java.awt.GridLayout(1, 2));

LNomFichier.setText("Nom du fichier :");
panelFichier.add(LNomFichier);

panelFichier.add(TFNomFichier);

add(panelFichier);

panelRepertoire.setLayout(new java.awt.GridLayout(1, 2));

LNomRepertoire.setText("Nom du r\u00e9pertoire");
panelRepertoire.add(LNomRepertoire);

panelRepertoire.add(TFNomRepertoire);

add(panelRepertoire);

BDownload.setLabel("T\u00e9l\u00e9charger");
BDownload.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        BDownloadActionPerformed(evt);
    }
});

add(BDownload);

pack();
java.awt.Dimension screenSize = java.awt.Toolkit.getDefaultToolkit().getScreenSize();
setSize(new java.awt.Dimension(400, 200));
 setLocation((screenSize.width-400)/2,(screenSize.height-200)/2);
}

private void BDownloadActionPerformed(java.awt.event.ActionEvent evt)
{
    FTPClient ftpc = null;
    try
    {
        ftpc = new FTPClient(host);
        ftpc.debugResponses(true);
        ftpc.login(user, password);
        ftpc.setTransferType(FTPTransferType.BINARY);
        ftpc.setConnectMode(FTPConnectMode.PASV);

        System.out.println("R\u00e9pertoire courant = " + ftpc.pwd());
        System.out.println("Tentative d'acquisition du fichier");
        String nomFichier = TFNomFichier.getText();
        String nomRepertoire =TFNomRepertoire.getText();
        String nomFichierOrigine = System.getProperty("file.separator") + nomRepertoire +
            System.getProperty("file.separator") + nomFichier;
```

```

String nomFichierCible = "copie-" + nomFichier;
System.out.println("Fichier demandé : " + nomFichierOrigine);
System.out.println("Fichier cible : " + nomFichierCible);
ftpclient.get(nomFichierCible, nomFichierOrigine);
ftpclient.quit();
}
catch (IOException e)
{
    System.out.println("Oh oh IOException : " + e.getMessage());
}
catch (FTPEException e)
{
    System.out.println("Oh oh FTPEexception : " + e.getMessage() + " -- " +
        e.getReplyCode());
}
catch (Exception e)
{
    System.out.println("Oh oh exception : " + e.getMessage());
}
}

/** Exit the Application */
private void exitForm(java.awt.event.WindowEvent evt) {System.exit(0); }

/**
 * @param args the command line arguments
 */
public static void main(String args[])
{
    new FTPClientGuiFenApp().show();
}

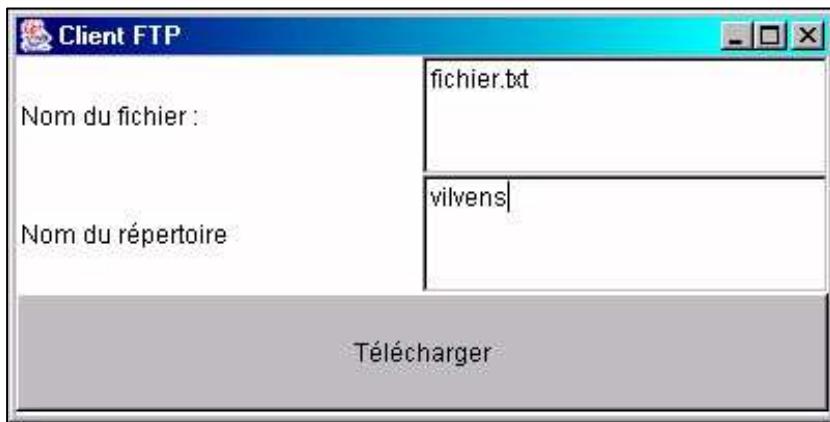
// Variables declaration – do not modify
private java.awt.Panel panelFichier;
private java.awt.Label LNomFichier;
private java.awt.TextField TNomFichier;
private java.awt.Panel panelRepertoire;
private java.awt.Label LNomRepertoire;
private java.awt.TextField TNomRepertoire;
private java.awt.Button BDownload;
// End of variables declaration
}

```

On remarquera encore l'utilisation de la méthode

```
public void debugResponses(boolean on)
```

afin d'afficher les requêtes et réponses sur la console. Un exemple d'exécution sera :



La console nous renseigne sur les commandes FTP envoyées au serveur ainsi que sur les réponses fournies par le serveur :

```
---> USER anonymous
331 Guest login ok, send your complete e-mail address as password
---> PASS claude.vilvens@prov-liege.be
230 User logged in, proceed
---> TYPE I
200 Command TYPE okay
---> PWD
257 "/" is current directory
Répertoire courant = "/" is current directory
---> PASV
227 Entering Passive Mode (192,168,2,2,4,12)
---> NLST *.*
150 File status okay; about to open data connection
226 Closing data connection
0. winzip.log
Tentative d'acquisition du fichier
Fichier demandé : \vilvens\fichier.txt
Fichier cible : copie-fichier.txt
---> PASV
227 Entering Passive Mode (192,168,2,2,4,13)
---> RETR \vilvens\fichier.txt
150 File status okay; about to open data connection
226 Closing data connection
---> QUIT

221 Goodbye
```

Une autre utilisation de notre programme, fonctionnant toujours une machine Windows, avec comme cible la machine Linux U2 et donc avec comme valeur pour les variables membres

```
String host = "u2.wildness.loc",
       user = "vilvens",
       password="EtPuisQuoiEncore ";
```

---

pourrait donner :

```
--> USER vilvens
331 Password required for vilvens.
--> PASS EtPuisQuoiEncore
230- Have a lot of fun...
230 User vilvens logged in.
--> TYPE I
200 Type set to I.
--> PWD
257 "/home/vilvens" is current directory.
Répertoire courant = "/home/vilvens" is current directory.
Contenu du répertoire :
--> PASV
227 Entering Passive Mode (10,59,5,219,128,247)
--> NLST *:*
150 Opening BINARY mode data connection for 'file list'.
226 Transfer complete.
0. compile.sh
1. inbox.ultra-secret
2. maxime.txt
Tentative d'acquisition du fichier
Fichier demandé : maxime.txt
Fichier cible : copie-maxime.txt
--> PASV
227 Entering Passive Mode (10,59,5,219,128,248)
--> RETR maxime.txt
150 Opening BINARY mode data connection for 'maxime.txt' (90 bytes).
226 Transfer complete.
--> QUIT

221 Goodbye.
```

si du moins on a rectifié la gestion des répertoires par :

```
...
System.out.println("Tentative d'acquisition du fichier");
String nomFichier = TFNomFichier.getText();
String nomReperoire = TFNomReperoire.getText();
String nomFichierOrigine;
if (!nomReperoire.equals("")) nomFichierOrigine = nomReperoire + "/" + nomFichier;
else nomFichierOrigine = nomFichier;
String nomFichierCible = "copie-" + nomFichier;
System.out.println("Fichier demandé : " + nomFichierOrigine);
System.out.println("Fichier cible : " + nomFichierCible);
ftpc.get(nomFichierCible, nomFichierOrigine);
ftpc.quit();
...
```

La commande netstat nous renseigne sur les connexions réseau correspondantes :

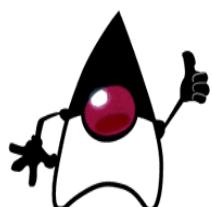
tcp	0	0 0.0.0.0: <b>21</b>	0.0.0.0:*	<b>LISTEN</b>
tcp	0	0 10.59.5.219: <b>33015</b>	10.59.5.13:2539	<b>TIME_WAIT</b>
tcp	0	0 10.59.5.219: <b>33016</b>	10.59.5.13:2540	<b>TIME_WAIT</b>
<i>tcp</i>	<i>0</i>	<i>0 10.59.5.219:<b>21</b></i>	<i>10.59.5.13:2538</i>	<i>TIME_WAIT</i>

Si on analyse la réponse donnée par le serveur à la commande PASV, on constate qu'il compte attendre la connexion sur le port désigné par (128,248) :

128,248 → 1000 0000 1111 1000 = 33016

– ces deux octets identifient bien le port 33016 dont nous voyons l'usage pour la connexion des données (ainsi que le port adjacent 33015 pour la 2ème connexion de données).

Précisons encore que le fichier visé a été recopié, sous le nom de copie-<fichier>.txt dans le répertoire c:\forte4j\bin – simplement parce que c'est le répertoire par défaut de l'IDE utilisé sur la machine cliente !



Qui aurait pu croire que FTP nous occuperait si longtemps ?  
Mais déjà d'autres sirènes "protocoliennes" nous appellent. Les protocoles évoqués dans ces deux derniers chapitres sont tous basés sur TCP – mais il en existe qui sont basés sur UDP, le mal aimé parce que non fiable.  
Par ici ...

## XXIII. Le protocole SNMP et l'administration des réseaux



*L'étude a été pour moi le souverain remède contre les dégoûts de la vie, n'ayant jamais eu de chagrin qu'une heure de lecture ne m'aït ôté.*

(C. Montesquieu, Mes pensées)

### 1. Les objectifs de l'administration réseau

La problématique de la gestion d'un réseau a été de tout temps une espèce de Graal ou de monstre du Loch Ness : gestion des machines, politiques de sécurité, ajout de nouveaux éléments, etc sont les questions fondamentales qui conditionnent le fonctionnement du réseau et, souvent, de l'entreprise elle-même. Pour ne pas tomber dans le piège des pratiques propriétaires trop spécifiques et inutilisables dans d'autres contextes, il est rapidement apparu que la définition de pratiques normalisées était nécessaire.

Sans surprise, c'est donc l'ISO qui a défini les points fondamentaux de l'administration réseau :

- ◆ la gestion des pannes, qui définit une qualité de service;
- ◆ la gestion de la configuration matérielle et logicielle;
- ◆ la gestion des performances, avec les tests d'adéquation des configurations aux résultats attendus;
- ◆ la sécurité;
- ◆ la comptabilité dans le cas des réseaux payants.

Sur cette base, l'ISO a proposé dans les années '80 une norme d'administration de réseaux appelée CMIS/CMIP (Common Management Information Service/Protocol – resp. ISO 9595 et 9596). Provisoirement, l'IAB a approuvé un protocole à utiliser pour l'administration réseau en accord avec cette norme, prévoyant une solution plus évoluée à long terme. Mais le provisoire dure toujours ... et le protocole, qui se nomme **SNMP** (Simple Network Management Protocol), s'est imposé comme le standard !

### 2. Un protocole d'exploration basé sur UDP

L'objectif fondamental de SNMP est donc de rendre accessible le maximum d'informations concernant une machine (au sens large) du réseau, si du moins la politique de sécurité en vigueur le permet. Pour ce faire, il considère que :

- ◆ chaque élément administrable (donc, une machine du réseau) exécute un programme serveur qui gère les informations de cet élément et est susceptible de les fournir en réponse à une requête : ce programme est appelé un **agent SNMP**;
- ◆ la station d'administration, appelée le **manager SNMP**, se comporte comme un client vis-à-vis des différents agents SNMP : elle leur demande des informations, les regroupe et les

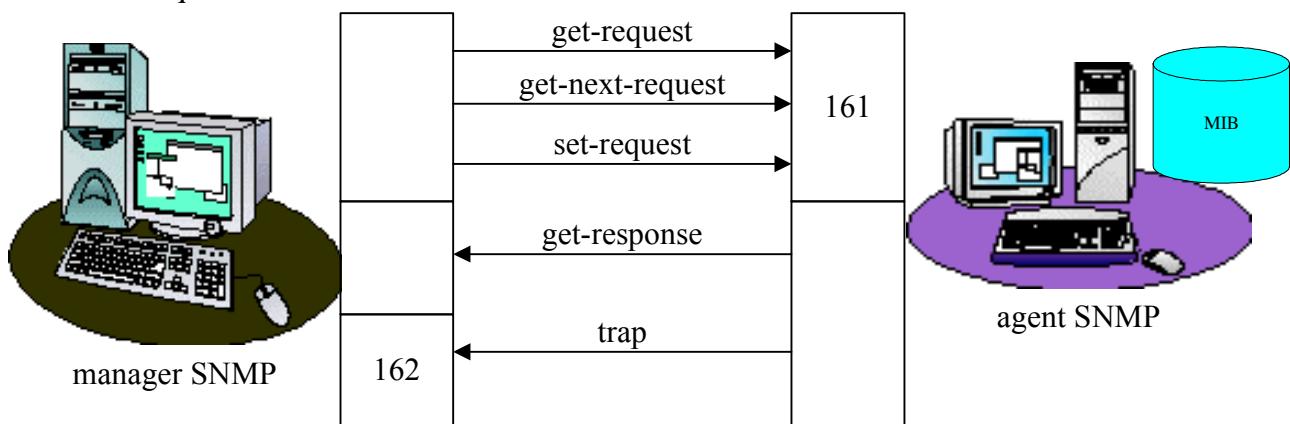
présente de manière claire à l'administrateur réseau qui l'utilise, généralement dans un interface graphique.

Les requêtes et les réponses sont transportées dans des trames **UDP**. Les raisons de préférer ce protocole non fiable au fiable TCP sont que UDP est plus simple, plus rapide et consomme moins de bytes pour son en-tête (8 au lieu de 20) : tout ceci a de l'importance quand on refléchit au nombre important de requêtes qu'implique une surveillance du réseau et à la nécessité de pouvoir communiquer très rapidement une alarme vers l'administrateur en cas de panne.

Pour avoir les moyens de ses ambitions, SNMP a été doté de :

- ◆ 5 types de messages pour les échanges entre le manager et les agents (RFC 1157) :
- 3 **requêtes** d'usage courant (get-request, get-next-request et set-request) envoyées par le manager et la **réponse** de l'agent à celles-ci (get-response), toutes utilisant le port **161**;
- un message de type **alarme** (trap) envoyé par un agent sur le port **162** du manager;
- ◆ un système de représentation normalisé des informations en tous genres que l'on va pouvoir gérer = le **SMI** (Structure of Management Information – RFC 1155);
- ◆ une structure de base de données pour toutes ces informations normalisées que gère chaque agent : la **MIB** (Management Information Base – RFC 1156 et 1213, mais d'autres ont suivi).

Schématiquement :



SNMP	
Nom complet :	Simple Network Management Protocol
Nature :	Protocole applicatif d'administration de réseau
RFC :	1157-1155-1156, 1441→1452, 2571→2575, 2411→3418
Ports par défaut :	161 et 162
Protocole de transport :	UDP
Principales commandes :	get-request, get-next-request, set-request, get-response, trap

SNMP existe en version 1, 2 et 3. La version 1 est assez simple, la version 2 est confuse et la version 3 est la seule à être effectivement sécurisée. Nous y reviendrons.

### 3. La représentation et l'identification de l'information dans la MIB

Tout information est pour SMI un objet caractérisé par

- ◆ un nom qui est son identificateur unique; cette unicité est garantie par le fait que cet identificateur est en fait un chemin dans une arborescence – nous allons y revenir;
- ◆ une syntaxe qui définit le type de cet identificateur (entier [*integer*] ou chaîne de caractères [*octet string*], mais aussi adresse réseau [*network address*] ou compteur [*counter*]);
- ◆ un encodage qui définit comment l'information associée à l'objet est formatée avant sa transmission sur le réseau.

Ainsi, par exemple, l'information "nom de la machine" est connue par

- ◆ le nom : system.sysName.0 ou .1.3.6.1.2.1.1.5.0;

- ◆ la syntaxe : octet string;

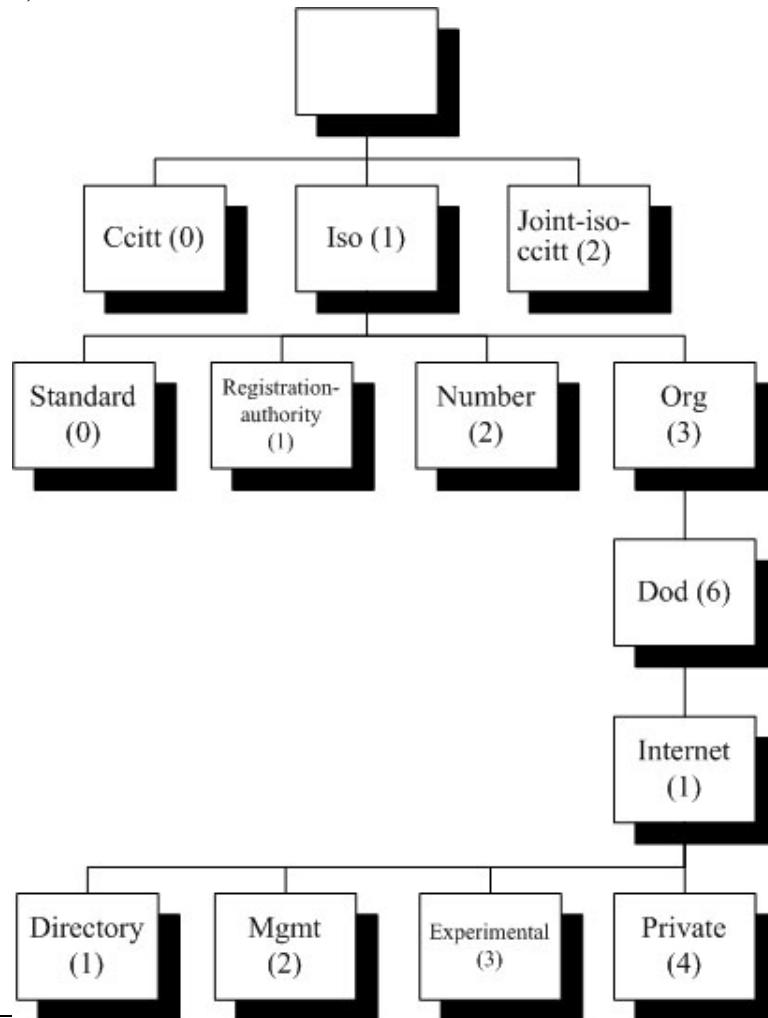
- ◆ l'encodage : par défaut;

sa valeur étant, par exemple, "Ulysse".

La MIB, qui donne donc à toutes ces informations, est une base de données virtuelle, en ce sens qu'elle regroupe des informations éparses sur l'agent SNMP : certaines sont dans des fichiers de configuration, d'autres dans registres particuliers, etc<sup>1</sup>.

Ces informations sont placées dans une structure arborescente standardisée, allant du plus général au plus particulier. Chaque nœud est identifié par un nom symbolique (par exemple, system ou sysDescr) et par un entier correspondant (par exemple, ici, 1 et 1).

Schématiquement, le début de cette hiérarchie est :



<sup>1</sup> en quelque sorte, on dirait en base de données que l'on utilise plutôt une vue des données véritables ...

Comme on peut le voir, la racine de l'arbre n'a pas de nom. Ses trois fils directs correspondent à ce qui est administré par l'ISO (**Iso(1)**), par le CCITT<sup>1</sup> (**Ccitt(0)**) et par les deux organismes conjoints (Joint-iso-ccitt(**2**)). Pour ce qui nous intéresse, nous suivrons le chemin de l'ISO.

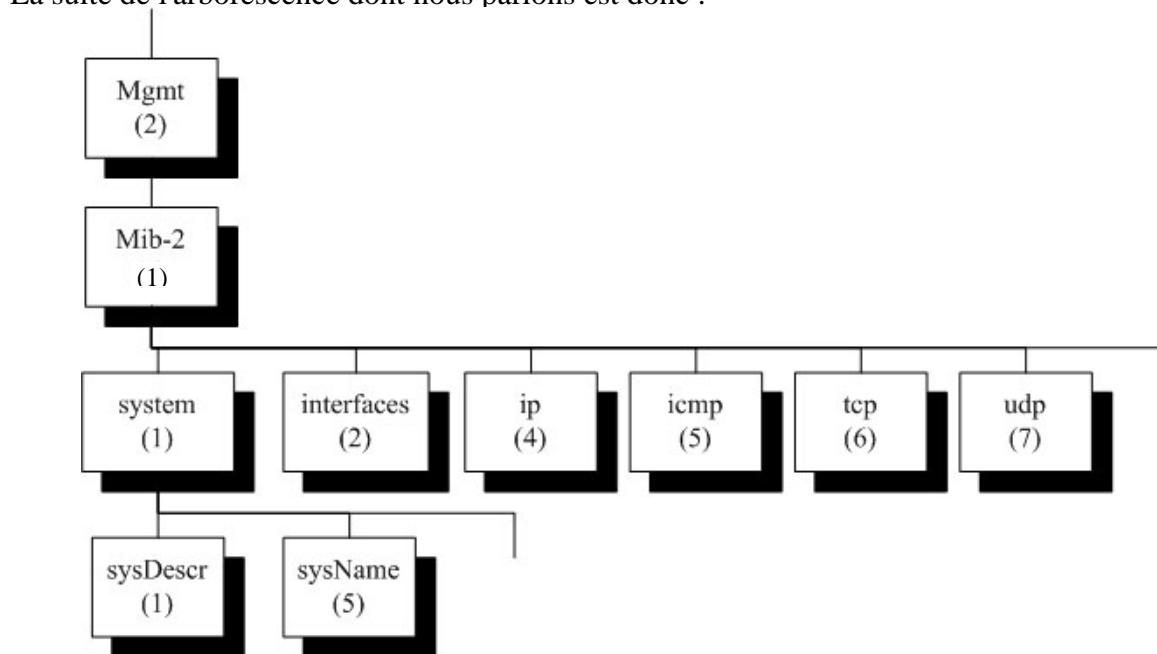
Les fils d'iso(**1**) comportent notamment **Org(3)**, destiné évidemment aux grandes organisations nationales ou internationales – parmi celles-ci, le Département de la Défense des Etats-Unis (**Dod(6)**), qui donne enfin accès au nœud **Internet(1)** – ouf !

Ce n'est pas fini : on découvre ensuite 4 fils, dont deux vont nous intéresser :

- ◆ **Mgmt(2)** : désigne le début du sous-arbre dédié par l'IAB à l'IANA et utilisant des entités normalisées; c'est dans cette arborescence que l'on trouvera la nomenclature de tous les éléments standards d'un système informatique;
- ◆ **Private(4)** : désigne tout ce qui n'est pas défini en standard; on y trouve par exemple, au niveau du nœud Enterprises(**1**), des sous-systèmes spécifiques chacun à un constructeur, qui définit les symboles comme bon lui semble – ce qui sous-entend qu'un constructeur a tout intérêt à demander à l'IANA un nœud à partir duquel il pourra enregistrer ses produits de manière univoque dans une MIB personnelle (ainsi, la MIB.de LAN Manager démarre à 77, celle de Microsoft à 311, etc).

L'information visée est tout simplement désignée par le chemin à suivre dans la MIB pour l'atteindre, chemin en forme de chaîne de noms symboliques (séparés par des points) ou en forme d'une suite d'entiers (également séparés par des points), complétée de ".0" pour signifier que l'on vise une feuille de l'arbre. C'est cette suite d'entiers qui est encore appelée l'**Object Identifier (OID)** de l'objet visé. Ainsi, par exemple, le nœud mgmt (qui n'est pas une feuille) est désigné par : .1.3.6.1.2.

Les premières versions de SNMP utilisaient une seule MIB (dite MIB-I) qui collectait l'ensemble de toutes les variables possibles. Mais devant le volume d'informations, l'IETF est passée à la définition de plusieurs MIB différentes, selon le domaine; il en existe une centaine à l'heure actuelle. Celle qui nous intéressera est la **MIB-II**, qui contient un ensemble cohérent de paramètres communs à tous les équipements : description du type de l'entité, interfaces réseau et adresse IP associée, nombre de datagrammes entrants et sortants, connexions TCP. La suite de l'arborescence dont nous parlons est donc :



<sup>1</sup> Conseil Consultatif International du Télégraphe et du Téléphone – devenu le ITU (International Telecommunication Union)

Donc, pour obtenir

- ◆ une description du système, nous demanderons l'objet .1.3.6.1.2.1.1.1.0 : Hardware: x86 Family 15 Model 2 Stepping 4 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free);
- ◆ le nom du système, nous demanderons l'objet .1.3.6.1.2.1.1.5.0 : Claude.

Evidemment, pour obtenir de telles informations, il faut que l'agent SNMP soit accessible et qu'il soit d'accord de fournir les renseignements demandés – vous avez dit "sécurité" ?

## 4. Les messages SNMP v1 et v2

Un message SNMP, encapsulé au sein d'une trame UDP elle-même enfermée dans une trame IP, va bien entendu comporter les éléments nécessaires à une requête ou une réponse SNMP. Mais d'autres éléments interviennent, variables selon les versions du protocole (on en compte 3 principales).

La structure de ces messages SNMP, comme d'ailleurs le SMI et les MIB, est décrit en **ASN.1** (Abstract Syntax Notation 1). C'est à la fois un langage standard de définition d'objets dont la syntaxe est assez proche des langages évolués tels que C ou Pascal (en réalité, SNMP utilise un sous-ensemble et rajoute quelques définitions) et aussi un système d'encodage optimisé. Nous en verrons l'un ou l'autre exemple ci-dessous.

### 4.1 Les requêtes et réponses SNMP v1 et v2

La structure générale d'une requête (qui n'est pas un trap en version 1) ou d'une réponse est :

ver- sion	<b>commu- nauté</b>	<b>PDU</b>							
		type PDU	id requête	code d'erreur	position erreur	liste arguments			
						nom var.	va- leur	nom var.	va- leur

Détaillons quelque peu les différents champs. Au premier niveau, on compte trois champs :

- ◆ **version** : c'est la version du protocole diminué de 1; donc, ce champ vaut 0 pour SNMP v1.
- ◆ **communauté** [*community*] : une communauté définit en fait un domaine d'administration, une groupe de sécurité : *seuls un manager et un agent de la même communauté pourront s'échanger des informations*; le nom de la communauté agit en fait comme un mot de passe : une requête est donc rejetée si la communauté est incorrecte;
- ◆ **PDU** (**P**acket **D**ata **U**nity) : il s'agit de la requête proprement dite, qui comporte un certain nombre de champs que nous allons détailler.

Auparavant, remarquons que cette structure de base du message sera décrite en ASN.1 de la manière suivante :

```
-- top-level message
Message ::=

SEQUENCE {
    version      -- version-1 for this RFC
        INTEGER {
            version-1(0)
        },
    community    -- community name
        OCTET STRING,
    data          -- e.g., PDUs if trivial
        ANY          -- authentication is being used
}
```

La "charge utile" du message, que l'on appelle donc PDU, comporte :

- ◆ **type de PDU** : un entier désigne le type de requête dont il s'agit selon la spécification ASN.1

```
-- protocol data units
PDUs ::=

CHOICE {
    get-request GetRequest-PDU,
    get-next-request GetNextRequest-PDU,
    get-response GetResponse-PDU,
    set-request SetRequest-PDU,
    trap Trap-PDU
}
-- PDUs
GetRequest-PDU ::= [0] IMPLICIT PDU
GetNextRequest-PDU ::= [1] IMPLICIT PDU
GetResponse-PDU ::= [2] IMPLICIT PDU
SetRequest-PDU ::= [3] IMPLICIT PDU
Trap-PDU ::= [4] IMPLICIT PDU
```

- ◆ **identificateur de requête** : défini par le manager lors de l'envoi d'une requête et utilisé par l'agent dans sa réponse; ceci permet d'associer la réponse à la requête correspondante, car il ne faut pas perdre de vue que le protocole de transport sous-jacent est UDP.
- ◆ **code d'erreur [error status]** : il est fourni par l'agent dans sa réponse; pour une requête, ce champ est nul; les codes utilisables sont :

<b>Statut d'erreur</b>	<b>Nom</b>	<b>Description</b>
0	noError	Pas d'erreurs.
1	tooBig	Réponse de taille trop grande.
2	noSuchName	Variable inexistante.
3	badValue	Écriture d'une valeur invalide
4	readOnly	Essai de modification d'une variable en lecture seule.
5	genErr	Autre erreur

- ◆ **position d'erreur** [*error index*] : indique la variable qui a provoqué l'erreur (uniquement dans les cas *noSuchName*, *badValue* et *readOnly*).
- ◆ **liste des paramètres de la requête**, comportant le nom des variables visées et, dans le cas d'une réponse, les valeurs correspondantes.

## 4.2 Les traps SNMP v1

Dans la version 1, les messages de type trap ont un format particulier (mais dans la version 2, ils prennent le même format que les autres types de messages) :

ver- sion	commu- nauté	PDU										liste arguments			
		type PDU	entre- prise	adresse agent	type trap	code spé- cifi- que	estam- pille	nom var.	va- leur	nom var.	va- leur	...			

Détaillons à nouveau quelque peu les nouveaux champs qui apparaissent :

- ◆ **entreprise** : indique le type d'objet généré;
- ◆ **adresse agent** : indique l'adresse IP de l'agent émetteur;
- ◆ **type trap** : est positionné par l'agent et indique l'évènement selon les codes suivants :

Type de trap	Nom	Description
0	coldStart	Initialisation de l'agent
1	warmStart	Réinitialisation de l'agent
2	linkDown	Passage de l'interface à l'état bas (première variable)
3	linkUp	Passage de l'interface à l'état haut (première variable)
4	authenticationFailure	Emission par le manager d'une communauté invalide
5	egpNeighborLoss	Passage d'un homologue EGP à l'état bas (première variable indiquant l'adresse IP de l'homologue)
6	enterpriseSpecific	cf. champ spécifique pour avoir de l'information

- dans le dernier cas (enterpriseSpecific), il faut regarder le champ suivant;
- ◆ **code spécifique** : il s'agit bien sûr d'un code spécifique à une entreprise (au sens du nœud enterprise de la MIB-II);
- ◆ **estampille horaire** : indique le temps passé depuis l'initialisation de l'agent.

## 5. Différentes versions de SNMP

### 5.1 Les versions 1 et 2

La version 1 de SNMP a été publiée en 1990 (RFCs 1155-6-7). Le principal reproche qui peut lui être adressé est son manque de sécurité : ainsi le nom de la communauté, qui est en fait le "mot de passe" garant de la confidentialité entre l'agent et son manager, passe en clair sur le réseau !

L'IETF a donc mis en chantier une version 2, dont les premières RFCs sont apparues en 1993. Mais il n'a jamais été possible d'accorder les différents membres du groupe de travail, si bien que plusieurs versions 2, incompatibles, ont été proposées. En définitive,

l'IETF a donc du se résoudre à publier en 1996 une version 2 non sécurisée (RFCs 1901→1908). Qu'apportait-elle de plus ? En bref :

- ◆ la communication entre les managers : une machine peut se comporter à la fois comme un agent et comme une station d'administration, si bien qu'un manager d'un niveau supérieur peut commander un agent par l'intermédiaire de ces managers-agents mixtes;
- ◆ l'apparition de deux nouveaux PDU : GetBulkRequest pour réduire le nombre d'échanges réseau en permettant l'utilisation de trames de taille maximale (c'est-à-dire équivalent au MTU du réseau) et InformRequest qui permet à un manager d'envoyer une alarme à un autre manager;
- ◆ un SMI amélioré permettant la définition de nouveaux objets;
- ◆ quelques tentatives de sécurisation, avec des possibilités d'authentification de l'expéditeur du message, de test d'intégrité du contenu et de cryptage.

Cette version 2 s'est cependant révélée plus difficile à implémenter et à mettre en place que la version 1, avec finalement une valeur ajoutée ne justifiant pas ces difficultés. Si bien que lorsque le besoin de sécurité s'est fait sentir de manière accrue, on est passé directement à la version 3 dont on parle depuis 1999.

## 5.2 La version 3

Cette version est en fait une refonte assez importante des concepts des versions précédentes, l'effort consenti se justifiant par la recherche de la sécurité des transactions et l'interopérabilité, avec une ouverture à d'éventuelles modifications ultérieures.

La version 3 définit un nouveau concept : l'**entité SNMP**.

Une **entité** est un ensemble de **modules** dédiés à une tâche précise (notamment la sécurité, le traitement des messages et les contrôles d'accès), choisis parmi une série de modules définis de manière standard. Selon les modules utilisés, on peut construire un **manager SNMP** ou un **agent SNMP**.

Plus précisément, une entité SNMP comporte deux grandes composantes.

1) **un moteur SNMP** : il a pour rôle de mettre en place les services d'envoi et de réception des messages, ceux d'authentification et de cryptage et les services de contrôle d'accès aux objets de la MIB. Un moteur SNMP se compose par conséquent des modules suivants.

- **le dispatcher de PDU** : son rôle est de déterminer à quelle version un message se réfère puis de l'envoyer au sous-système de traitement de messages correspondant.

- **le sous-système de traitement de messages** : il existe actuellement trois modèles, dédiés à SNMP v1, v2c et v3 – mais la porte reste ouverte à des versions futures.

- **le sous-système de sécurité** : à nouveau, il peut exister plusieurs modèles mais qui sont tous définis par rapport à  
-- l'authentification (l'IETF conseille HMAC-MD5-96);  
-- le cryptage (l'IETF conseille DES);  
-- la vérification du temps.

- le sous-système de contrôle d'accès : l'objectif est de définir les droits d'accès aux objets de la MIB; l'idée est de restreindre l'accès d'un client, dûment identifié, aux objets de la MIB qu'il a le droit de voir.

2) **des applications** : ce sont évidemment les processus qui vont être régis par le moteur; on y trouve

- l'application de génération des commandes
- l'application de réponse aux commandes
- l'application de génération des traps asynchrones
- l'application de réception de ces traps
- l'application de relais proxy permettant l'échange de messages entre les entités.

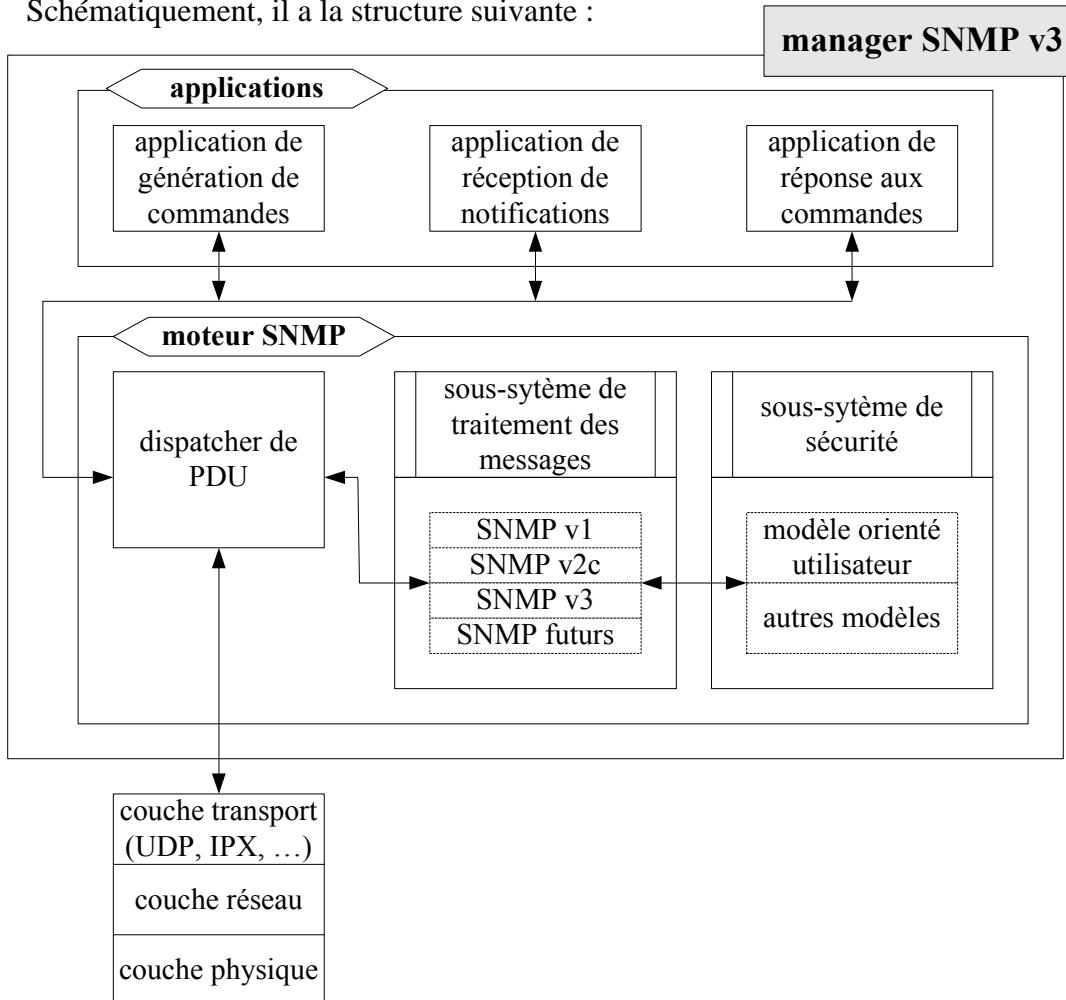
Concrètement, cela donne ceci.

## 6. L'architecture SNMP v3

A la lumière de ce que l'on entend par "entité SNMP", on conçoit qu'un système d'administration SNMP v3 se définit par la description du manager, de l'agent, de la trame et des mécanismes de sécurité mis en œuvre.

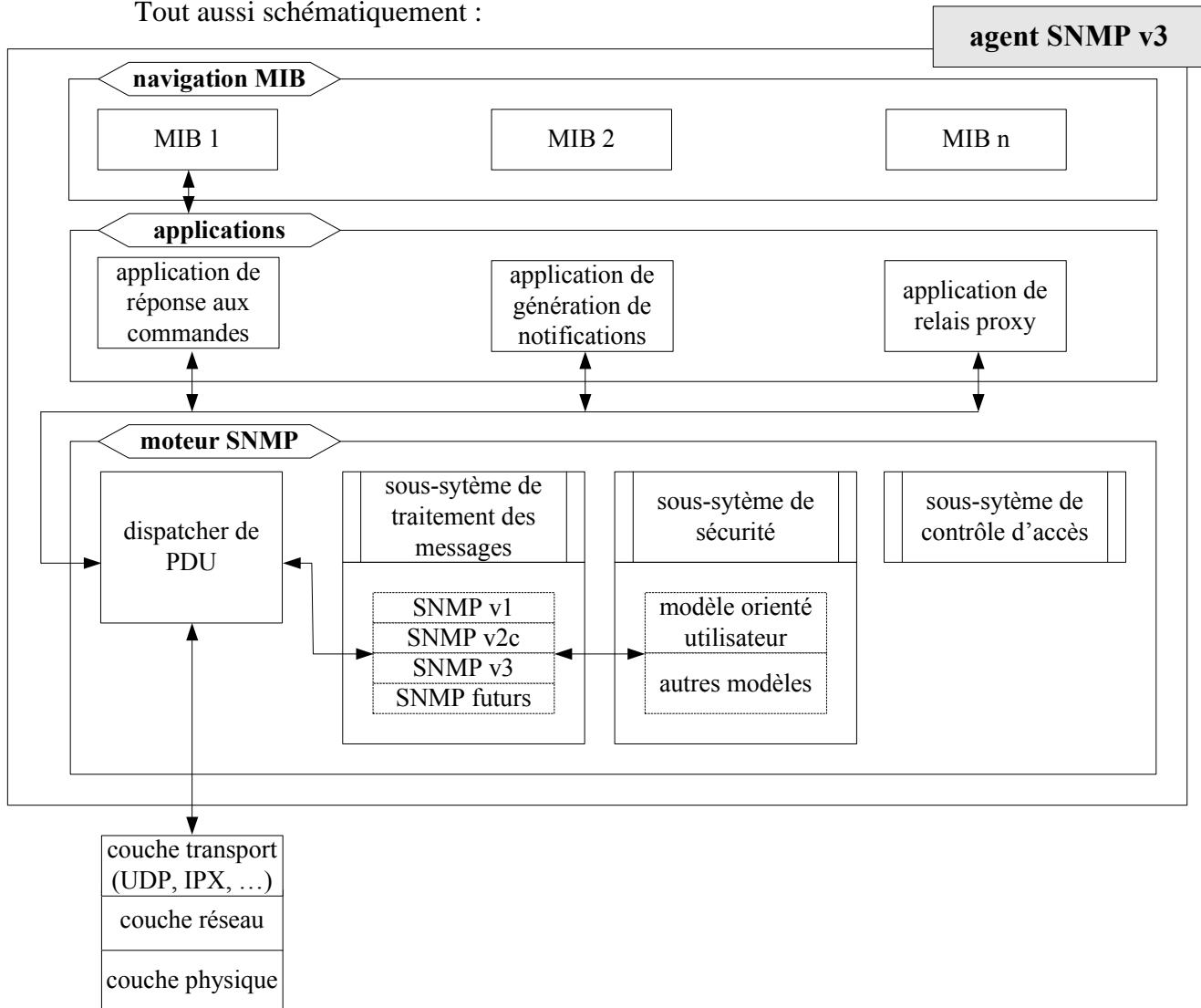
### 6.1 Le manager SNMP

Schématiquement, il a la structure suivante :



## 6.2 L'agent SNMP

Tout aussi schématiquement :



## 6.3 Les requêtes et réponses SNMP v3

La structure générale d'une requête ou d'une réponse selon SNMP v3 est :

version	header des données				paramètres de sécurité	données du PDU		
	msgId	msg-Max-Size	msg-Flags	ms Security-Model		context-Engine-ID	context-Name	PDU

Détaillons à nouveau quelque peu les différents champs. Au premier niveau, on compte quatre champs :

- ◆ **version** : c'est la version du protocole diminué de 1; donc, ce champ vaut 0 pour SNMP v1.
- ◆ **header des données** [*data header*] : on y trouve les informations concernant la nature du message et les caractéristiques de la réponse;
- ◆ **paramètres de sécurité** : on y trouve les informations relatives à l'authentification et au cryptage;

- ♦ **données du PDU** : outre le PDU lui-même (sur lequel nous allons revenir), on y trouve le "contexte", soit une identification de la machine et de son type (par exemple, port de routeur, carte ATM, etc).

Au niveau inférieur, on peut distinguer, sans être exhaustif :

**\*\* dans le header des données** : notamment

- \* msgId : identificateur de la requête à rappeler dans la réponse;
- \* msgFlags : 3 bits sont utilisés
  - le reportable flag : à 1 si une réponse est souhaitée (donc dans le cas d'une requête get, get-next ou set);
  - le **private flag** : à 1 si on utilise un cryptage;
  - l'**authentification flag** : à 1 si une authentification a été effectuée.
- \* msgSecurityModel : informe sur le modèle de sécurité utilisé, donc en fait si on a suivi le modèle de SNMP v1, v2 ou v3.

**\*\* dans les paramètres de sécurité** : notamment

- \* msgUserName : nom de celui qui a envoyé la requête;
- \* msgAuthenticationParameters : données pour l'authentification par **HMAC** (sur base d'un mot de passe);
- \* msgPrivacyParameters : le VI pour le cryptage (du PDU) par **DES/CBC** (avec une clé et le VI créés sur base d'un deuxième mot de passe).

**\*\* dans les données du PDU** : essentiellement le PDU, dont la structure est celle de SNMP v1 et v2, soit pour rappel :

type PDU	id requête	code d'erreur	position erreur	liste arguments				
				nom var.	va- leur	nom var.	va- leur	...

#### 6.4 Les mécanismes de sécurité de SNMP v3

On peut voir dans la structure des messages de SNMP v3 que les questions de sécurité ont été, à tout le moins, envisagées. Deux aspects ne demandent sans doute pas de commentaires supplémentaires : l'authentification (HMAC-MD5-96 et HMAC-SHA-96) et l'encryption symétrique (DES/CBC)<sup>1</sup>. Il faut cependant signaler que les deux clés respectives nécessaires sont générées à partir de deux mots de passe et aussi mentionner deux aménagements propres à SNMP :

- ♦ **la localisation** : l'idée est qu'il est fort risqué d'imaginer que le même mot de passe, ou du moins le même message digest qui sert de base au processus d'authentification, soit valable pour toutes les machines; cela signifierait que sa possession pourrait permettre le contrôle de l'ensemble du réseau administré. On a donc imaginé de faire varier le digest en fonction de la machine : celui-ci est calculé non seulement en fonction du mot de passe général mais aussi du contextEngineID, lequel peut provenir de l'adresse MAC de la carte réseau ou de son adresse IP ou encore d'une chaîne spécifiée par l'administrateur.
- ♦ **l'horodatage** : l'idée est ici d'empêcher qu'un paquet SNMP déjà dûment authentifié, localisé et encrypté ne puisse être récupéré par un tiers qui l'utiliserait plus tard. Donc, lorsqu'un message SNMP est reçu, on compare le temps qu'il contient avec le temps de réception : une différence de plus de 150 secondes fait que le message est tout simplement ignoré.

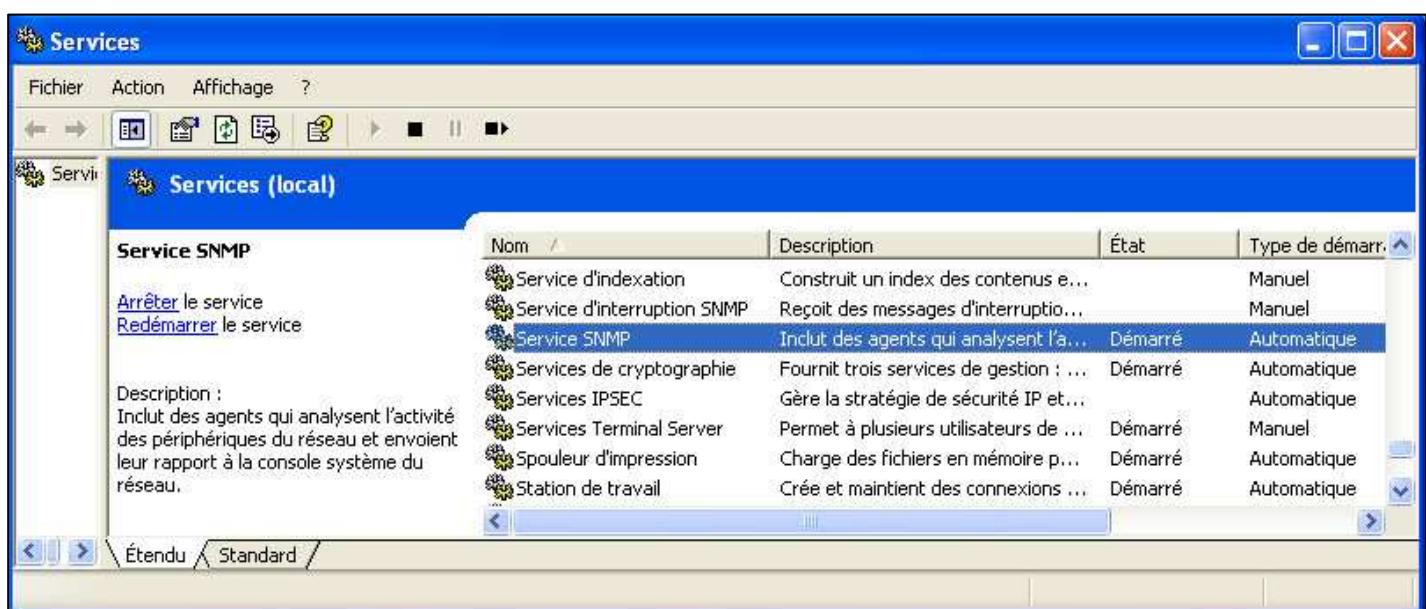
<sup>1</sup> voir " Java (II) : Programmation avancée des applications classiques et cryptographie", chapitre XIV

## 7. L'installation et la configuration d'un agent SNMP sous Windows

Dans le cas où un agent SNMP n'est pas déjà présent en tant que service sur la machine Windows considérée, il faut l'installer en suivant un chemin du type :

Panneau de configuration → Ajout/Suppression de programmes → Ajouter ou supprimer des composants Windows → Outils de gestion et d'analyse → SNMP (Protocole simplifié de gestion de réseau).

Ceci étant réalisé, on configure les services SNMP par :



qui fournit la boîte à onglets suivante, avec des paramétrages fort clairs :

The image displays four windows from the Windows Control Panel showing the properties of the "SNMP" service. Each window has tabs for Général, Connexion, Récupération, and Agent.

**Top Left Window (General Tab):**

- Interruptions:** Checked.
- Sécurité:** Not checked.
- Dépendances:** Not checked.
- Général:**
  - Nom du service:** SNMP
  - Nom complet:** Service SNMP
  - Description:** Inclut des agents qui analysent l'activité des périphériques du réseau et envoient leur rapport à [dropdown menu]
  - Chemin d'accès des fichiers exécutables:** C:\WINDOWS\System32\snmp.exe
  - Type de démarrage:** Automatique
  - Statut du service:** Démarré
  - Buttons:** Démarrer, Arrêter, Suspendre, Reprendre
  - Note:** Vous pouvez spécifier les paramètres qui s'appliquent pour le démarrage du service.
  - Paramètres de démarrage:** [empty input field]
- Buttons at the bottom:** OK, Annuler, Appliquer

**Top Right Window (Security Tab):**

- Interruptions:** Not checked.
- Sécurité:** Checked.
- Dépendances:** Not checked.
- Général:**
  - Contact:** Genius
  - Emplacement:** Liege
  - Service:**
    - Physique
    - Applications
    - Liaison de données et sous-réseau
    - Internet
    - Bout en bout
- Buttons at the bottom:** OK, Annuler, Appliquer

**Bottom Left Window (Connections Tab):**

- Interruptions:** Checked.
- Sécurité:** Not checked.
- Dépendances:** Not checked.
- Général:**
  - Envoyer une interruption d'authentification:** Checked.
  - Noms de communautés acceptés:**

Communauté	Droits
Dupeye	LECTURE SE...

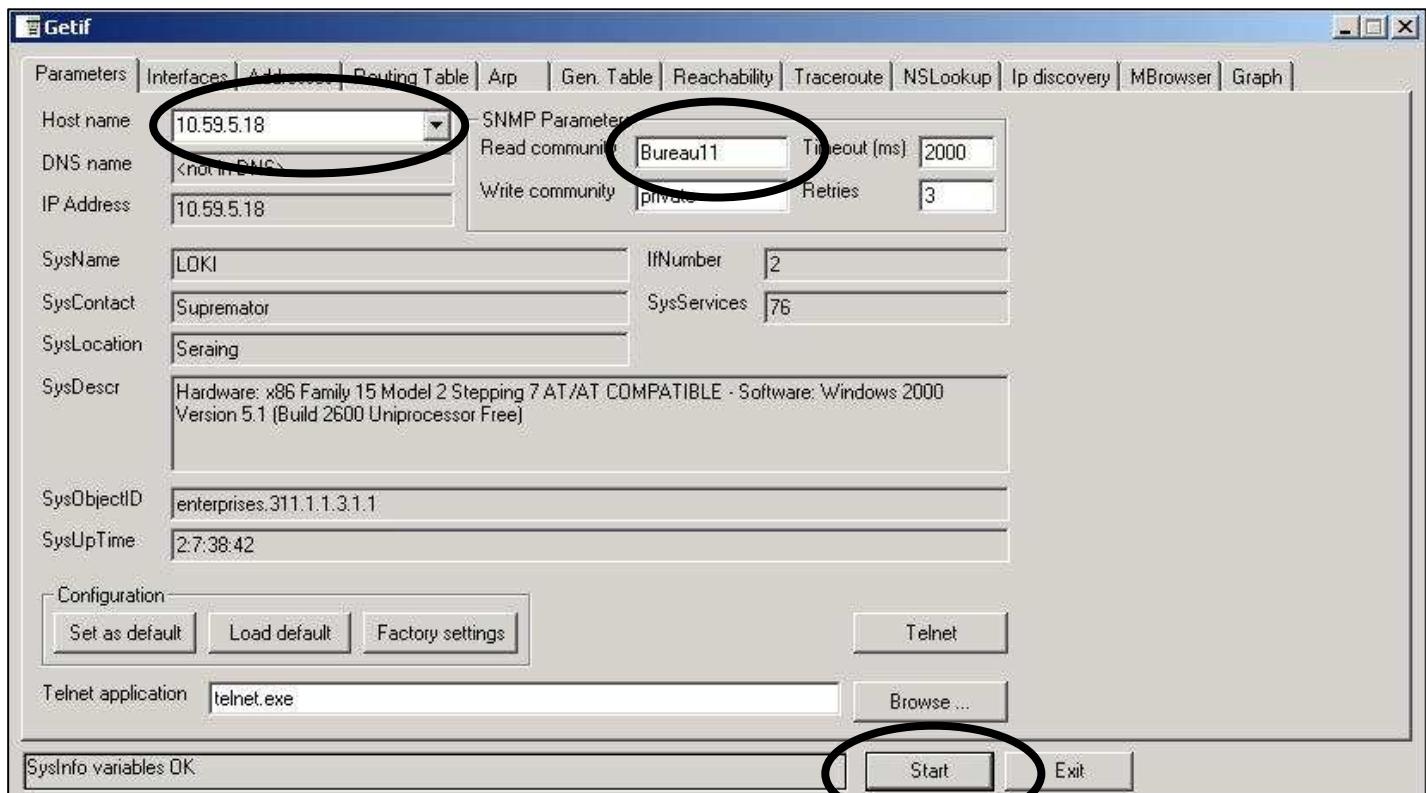
    - Ajouter...**, **Modifier...**, **Supprimer**
  - Connexion:**
    - Accepter les paquets SNMP provenant de n'importe quel hôte
    - Accepter les paquets SNMP provenant de ces hôtes
    - 192.168.2.2
    - 192.168.2.1
    - 127.0.0.1
    - Ajouter...**, **Modifier...**, **Supprimer**- Buttons at the bottom:** OK, Annuler, Appliquer

**Bottom Right Window (Dependencies Tab):**

- Interruptions:** Not checked.
- Sécurité:** Not checked.
- Dépendances:** Not checked.
- Général:**
  - Le service SNMP permet la gestion du réseau via les protocoles TCP/IP et IPX/SPX. Si des interruptions sont requises, un ou plusieurs noms de communautés doivent être spécifiés. Les destinations des interruptions peuvent être des noms d'hôtes, des adresses IP ou IPX.**
  - Nom de la communauté:** Dupeye
  - Ajouter à la liste**
  - Supprimer de la liste**
- Destinations des interruptions:**
  - 192.168.2.2
  - Ajouter...**, **Modifier...**, **Supprimer**
- Buttons at the bottom:** OK, Annuler, Appliquer

## 8. Un exemple de manager SNMP

L'application Getif est un utilitaire disponible gratuitement sur [www.wtcs.org/snmp4tpc/getif.htm](http://www.wtcs.org/snmp4tpc/getif.htm). Il fournit dans un GUI les informations que l'on peut acquérir par SNMP. Ainsi, l'écran du premier onglet ("Parameters") permet de fixer l'adresse IP de l'agent SNMP visé ainsi que la communauté dans laquelle on travaille; ceci étant effectué, l'appui sur le bouton Start permet d'envoyer les requêtes SNMP correspondants aux informations affichées :



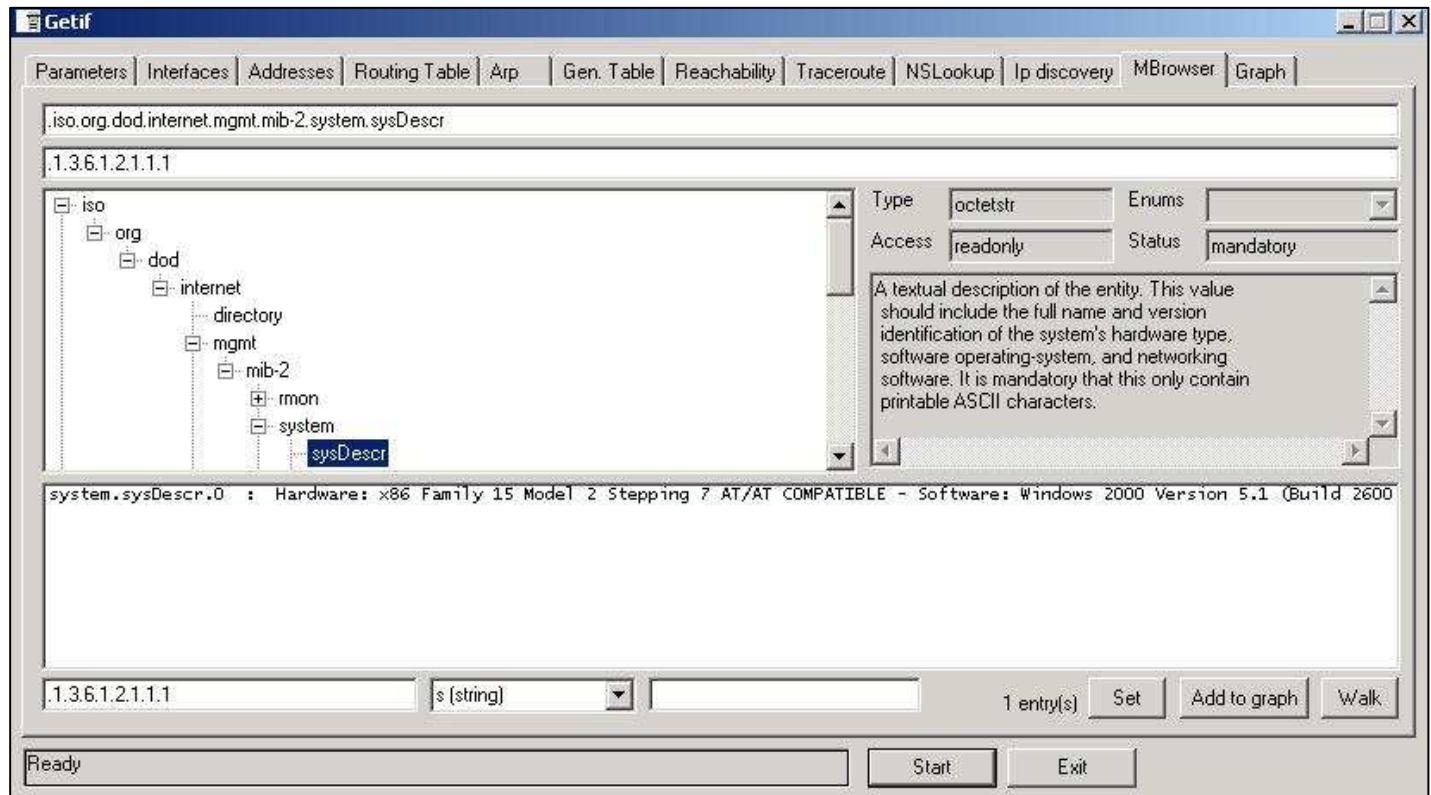
Les autres onglets fonctionnent de la même manière, par exemple pour les tables de routage :

The screenshot shows the 'Getif' application window with the 'Routing Table' tab selected. A table of routing entries is displayed with the following data:

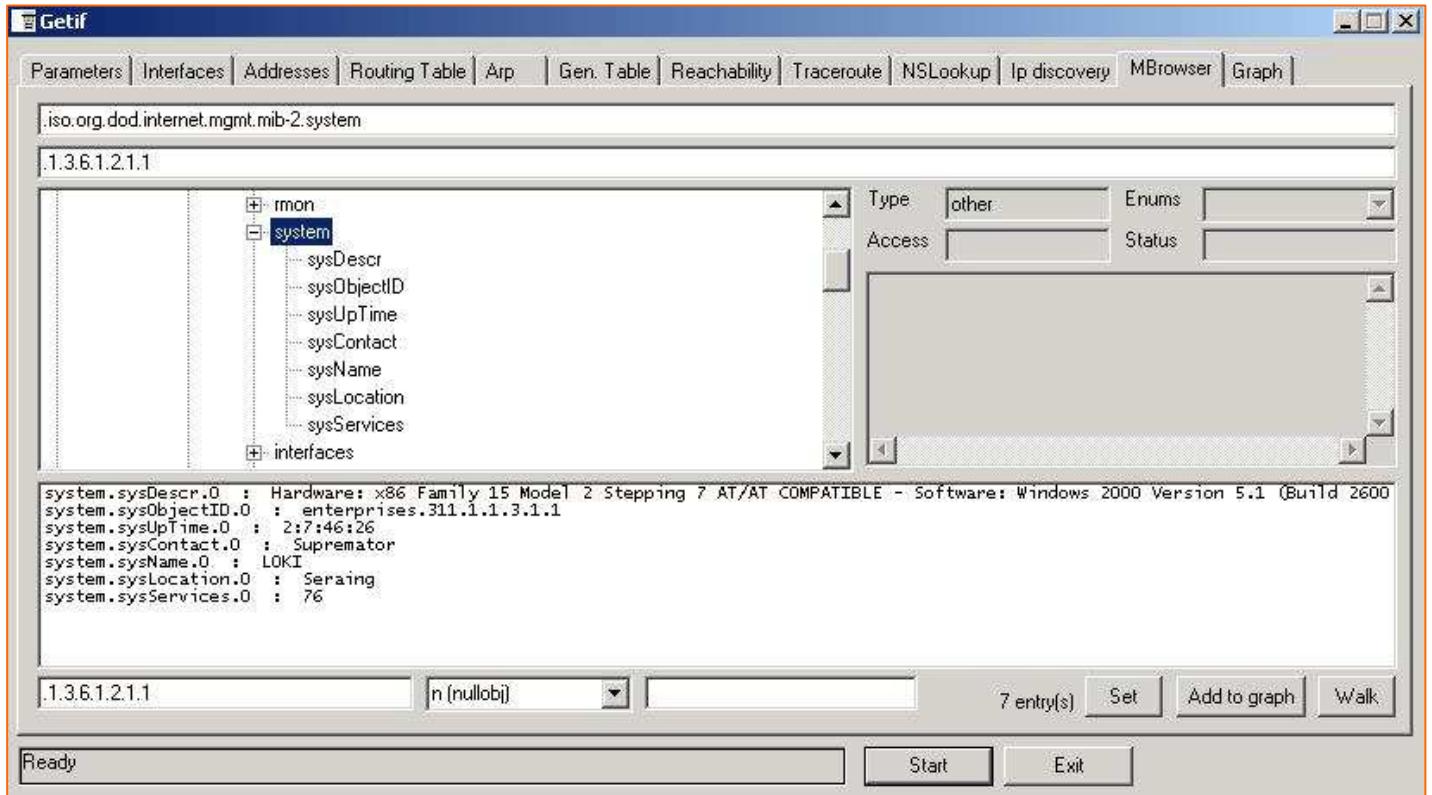
int.	dest	next hop	name	metric	mask	type	proto	age	info
2	000.000.000.000	010.059.004.254		20	000.000.000.000	indirect	netmgmt	201160	.ccitt.nullOID
2	010.059.004.000	010.059.005.018		20	255.255.254.000	direct	local	201160	.ccitt.nullOID
1	010.059.005.018	127.000.000.001		20	255.255.255.255	direct	local	201160	.ccitt.nullOID
2	010.255.255.255	010.059.005.018		20	255.255.255.255	direct	local	201160	.ccitt.nullOID
1	127.000.000.000	127.000.000.001		1	255.000.000.000	direct	local	201160	.ccitt.nullOID
2	224.000.000.000	010.059.005.018		20	240.000.000.000	direct	local	201160	.ccitt.nullOID
2	255.255.255.255	010.059.005.018		1	255.255.255.255	direct	local	201160	.ccitt.nullOID

At the bottom right of the window, there is a 'Start' button which is circled in red.

Mais l'outil le plus intéressant du point de vue didactique est le browser de MIB qu'est l'onglet MBrowser :



On peut donc y parcourir la MIB et obtenir un descriptif succinct d'une feuille (comme ci-dessus) ou de toute une branche (comme ci-dessous):



Il est possible d'enrichir cette MIB standard de MIBs privées, présentées sous forme de fichiers .txt. L'auteur du logiciel fournit sur son site un fichier GETIF-MIBS.ZIP qui comporte des centaines de MIBs privées. Pour les installer, il suffit de

- ◆ décompresser l'archive dans le répertoire C:\Program Files\Getif 2.2\Mibs; on y trouve déjà des fichiers .txt qui décrivent la MIB existante selon le format ASN.1; ainsi, le fichier RFC1155-SMI.txt contient le début :

#### RFC1155-SMI.txt

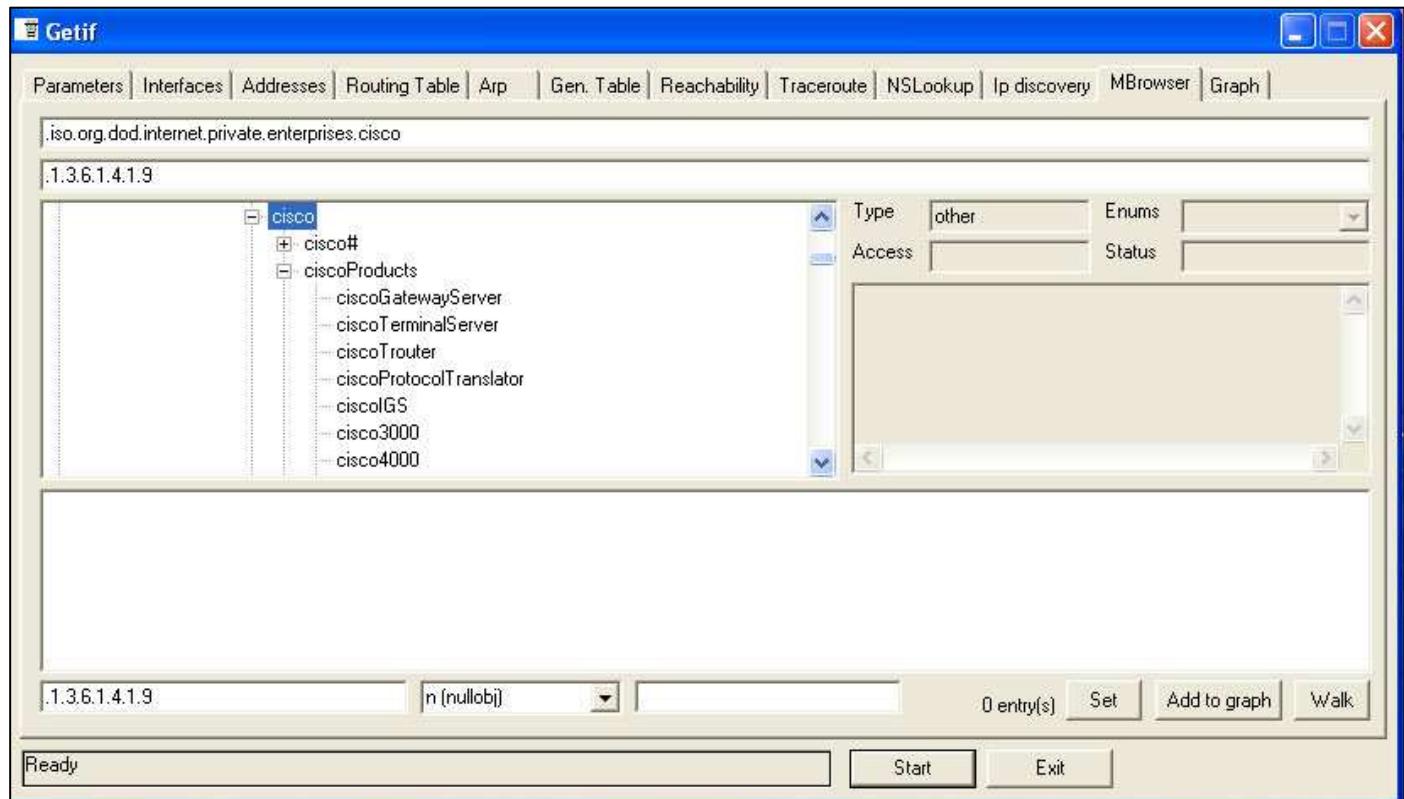
RFC1155-SMI DEFINITIONS ::= BEGIN

```
nullOID      OBJECT IDENTIFIER ::= { ccitt 0 }
internet     OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
directory    OBJECT IDENTIFIER ::= { internet 1 }
mgmt         OBJECT IDENTIFIER ::= { internet 2 }
experimental  OBJECT IDENTIFIER ::= { internet 3 }
private       OBJECT IDENTIFIER ::= { internet 4 }
enterprises  OBJECT IDENTIFIER ::= { private 1 }
```

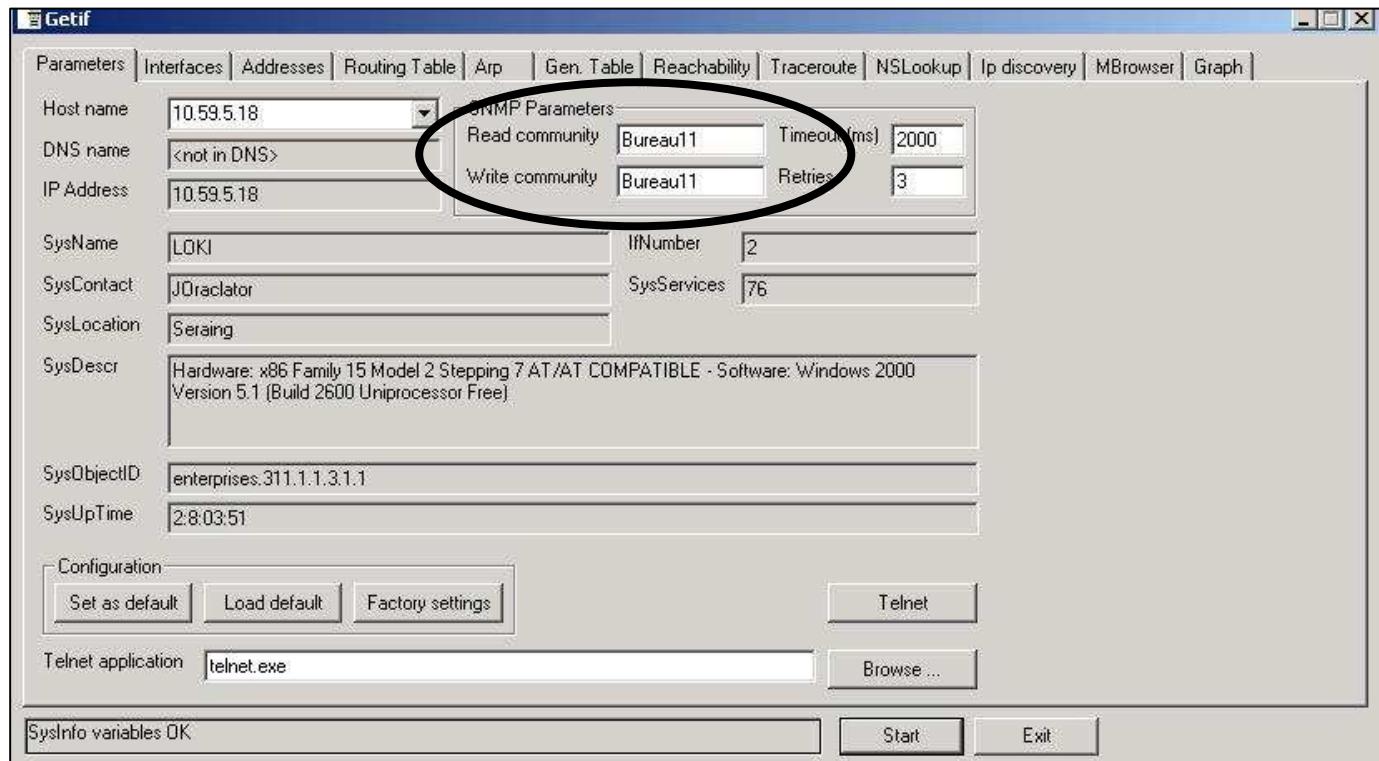
END

- ◆ supprimer le fichier .index qui est utilisé par getif pour retrouver les informations.

On peut ensuite vérifier que ces nouvelles MIBs sont bien présentes dans le browser de MIBs :



Il est également possible d'envoyer des requêtes de type set après avoir fixé une communauté read-write, modifiant ainsi des paramètres de la machine distante qui est d'accord de se laisser faire :



## **9. Le package WareMaker pour SNMP en Java et la commande get**

Ce package, nommé **com.waremaker.snmp** (du nom de la société qui l'a créé) et distribué sous la forme du fichier wmsnmp.jar, comporte une trentaine de classes, suffisantes pour interroger un agent SNMP. Il supporte les versions 1, 2 et 3 de SNMP.

**1)** La classe qui permet de démarrer est **SnmpFactory**. Instanciée au moyen d'un constructeur par défaut, son rôle est de contenir les paramètres de la machine qui demandera des renseignements, typiquement son adresse, précisée au moyen de la méthode

```
public void setHostAddress(String ipAddress)
```

et éventuellement le port qui sera utilisé, défini au moyen de :

```
public void setHostPort(int port)
```

**2)** Par défaut, ces deux paramètres sont 127.0.0.1 et un port dynamique. En quoi cet objet mérite-t-il son nom de factory ? Parce qu'il permet d'obtenir un objet **SnmpCommand** au moyen de

```
public SnmpCommand getCommand() throws SnmpException
```

Bien sûr, cet objet commande matérialise une commande SNMP envoyée par la machine hôte associée à la factory : il comporte la référence d'un objet **SnmpTarget** qui permet de désigner la machine visée par la commande. On peut obtenir cet objet cible, pour le configurer, au moyen de la méthode de **SnmpCommand** :

```
public SnmpTarget getTarget(java.lang.String connectAddress)
```

3) L'objet target ainsi récupéré peut donc être paramétré pour fixer :

- ◆ le nombre d'essais à tenter et le nombre de millisecondes entre ces essais (n'oublions pas que le protocole de transport est ici UDP qui est non fiable), au moyen de la méthode :

```
public void setAttempts(int attempts, int attemptsDelay)
```

- ◆ la communauté dans laquelle on travaille, au moyen de :

```
public void setGetCommunity(java.lang.String getCommunity)
```

- ◆ la version du protocole utilisée, avec :

```
public void setVersion(int version)
```

le paramètre pouvant être l'une des constantes définies dans la classe **SnmpConst** :

```
public static final int VERSION_1  
public static final int VERSION_2  
public static final int VERSION_3
```

4) Il nous reste à présent à construire une requête instance de **SnmpRequest**. Il suffit d'appeler tout d'abord la méthode de SnmpCommand :

```
public SnmpRequest newRequest()
```

qui fournit un tel objet requête initialisé avec les attributs courants de la commande, comme la communauté ou l'adresse de la machine visée. En fait, tout ce qu'il reste à faire est de préciser ce que l'on demande; en utilisant la terminologie MIB, on utilise la méthode de SnmpRequest :

```
public int setOid(int index, java.lang.String oid)
```

le premier paramètre ne servant qu'à préciser une position qui est usuellement 0. Nous pourrions lancer la commande SNMP get au moyen de la méthode SnmpCommand :

```
public SnmpResponse getWait (SnmpTarget target, SnmpRequest request, int msec)
```

le dernier paramètre déterminant un time-out en millisecondes et l'objet renvoyé contenant évidemment l'information demandée. Mais, avant de lancer la commande, on peut plutôt d'abord associer à la requête un handle obtenu au moyen de la méthode SnmpCommand :

```
public int get(SnmpTarget target, SnmpRequest request)
```

- bien que non obligatoire, il est en effet assez logique de penser à attribuer un identificateur univoque à chaque requête si l'on considère que l'on travaille sur UDP et qu'il se pourrait aussi que l'on travaille en multithreads : comme on se trouve en mode non connecté, il est prudent

---

de pouvoir identifier chaque requête par un tel handle, d'autant que la classe SnmpResponse n'est pas "thread-safe". Pour obtenir effectivement la réponse, on utilise l'une des méthodes de SnmpCommand :

```
public SnmpResponse responseWait (int msec)
public SnmpResponse responseWait (int handle, int msec)
```

selon que le handle présente un intérêt ou pas.

Il reste alors à extraire de la réponse ce qu'elle contient au moyen de méthodes comme, par exemple :

```
public java.lang.String getValueAsString (int index)
```

si l'information attendue est une chaîne de caractères. Il existe en fait toute une série de méthodes **getValueAsXXX()** correspondant aux types de données SNMP que l'on peut trouver dans une MIB. Ces méthodes sont en fait définies dans une classe **SnmpPdu** mère de SnmpRequest et de SnmpResponse – logique ... Outre l'information demandée, on peut aussi tirer de l'objet réponse les attributs basiques, comme :

```
public java.lang.String getAddress()
public int getPort()
```

Le programme suivant recherche le nom d'une machine d'adresse 10.59.5.18, et appartenant à la communauté Bureau11 : on utilise donc l'oid .1.3.6.1.2.1.1.5.0.

### **SNMPAgentSimple.java**

```
/*
 * SNMPAgentSimple.java
 */
/**
 * @author vilvens
 */
import com.waremaker.snmp.*;

public class SNMPAgentSimple
{

    public SNMPAgentSimple() {}

    public static void main(String[] args)
    {
        SnmpFactory fac = new SnmpFactory();
        fac.setHostAddress("localhost");

        SnmpCommand com = null;
        try
        {
            com = fac.getCommand();
        }
```

```
catch (SnmpException e)
{
    System.out.println("Erreur d'acquisition de commande : " + e.getMessage());
    System.exit(0);
}

SnmpTarget cible = null;
cible = com.getTarget("10.59.5.18");

cible.setAttempts(2, 2000);
cible.setGetCommunity("Bureau11");
cible.setVersion(SnmpConst.VERSION_1);

SnmpRequest requete = com.newRequest();
requete.setOid(0, ".1.3.6.1.2.1.1.5.0");
String oid = requete.getOidAsString(0);
System.out.println("OID = " + oid);

int handle = com.get(cible, requete);

SnmpResponse reponse = com.responseWait(handle, 5000);

if (reponse !=null)
{
    System.out.println("Reponse = " + reponse.getValueAsString(0));
    System.out.println("Adresse = " + reponse.getAddress());
    System.out.println("Port = " + reponse.getPort());
    System.out.println("Version SNMP = " + reponse.getVersion());
    System.out.println("Communauté = " + reponse.getCommunity());
}
else
{
    System.out.println("Erreur Snmp");
}
com.shutdown();
}
```

L'exécution de ce programme dans le contexte d'un réseau de type Intranet donne, pour la machine hôte :

```
OID = 1.3.6.1.2.1.1.5.0
Reponse = ULYSSE
Adresse = 10.59.5.13
Port = 161
Version SNMP = 0
Communauté = Bureau11
```

et pour une autre machine

---

OID = 1.3.6.1.2.1.1.5.0

Reponse = LOKI

Adresse = 10.59.5.18

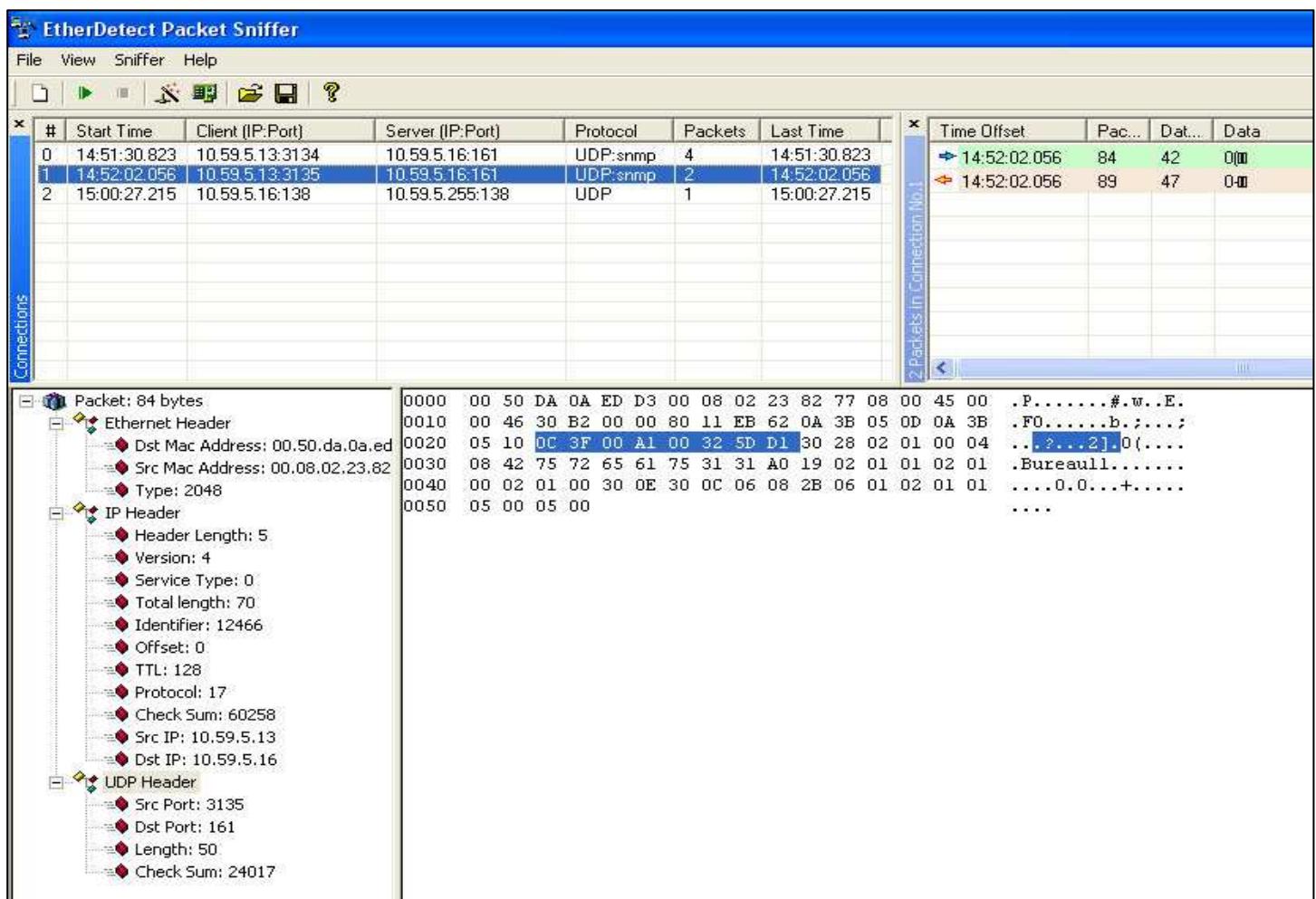
Port = 161

Version SNMP = 0

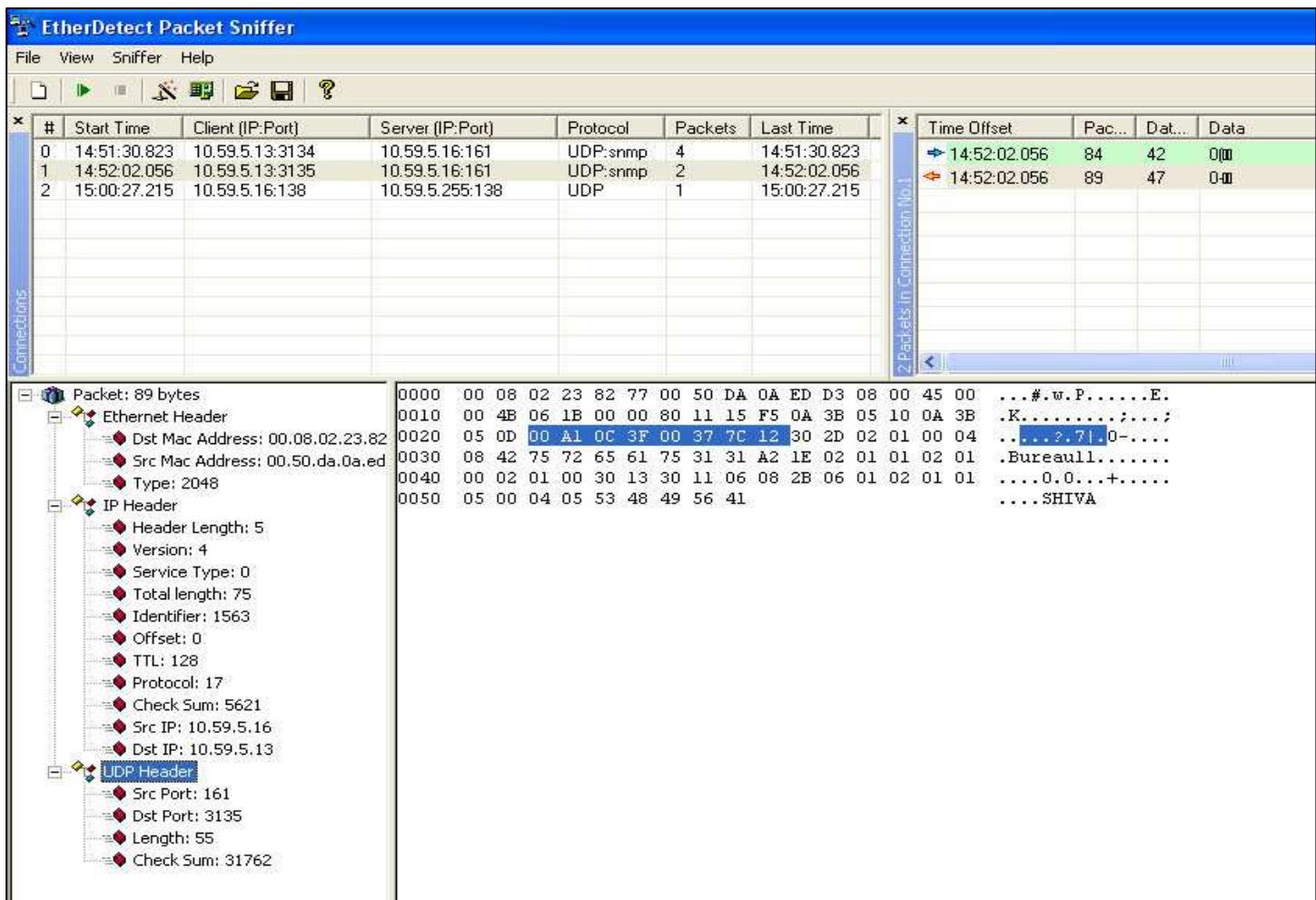
Communauté = Bureau11

## **10. Analyse du trafic réseau pour une commande get**

La requête consiste bien en un paquet UDP dirigée vers le port 161. On remarquera que le nom de la communauté passe en clair sur le réseau !



La réponse comporte bien l'information demandée, soit le nom de la machine :



## 11.La commande set en Java

La programmation d'une commande set de SNMP est très similaire à celle de d'une commande get. Les seules différences

- ♦ il faut préciser la communauté read-write (jusqu'ici, nous n'avions précisé que la communauté read-only) au moyen de la méthode de SnmpTarget :

```
public void setSetCommunity(java.lang.String getCommunity)
```

- ♦ l'objet SnmpRequest doit se voir préciser la nouvelle valeur à donner à l'objet précisé par le setOid(); on dispose pour cela d'autant de méthodes qu'il y a de types de données dans la MIB utilisée, par exemple :

```
public int setInteger (int index, int value)
public int setOctetStr (int index, java.lang.String value)
public int setTimeTicks (int index, int value)
```

...

Comme pour la commande get-request, on peut utiliser un handle ou pas lors de l'envoi de la commande à l'agent SNMP visé, en employant :

public SnmpResponse **setWait**(SnmpTarget target, SnmpRequest request, int msec)

ou

public int **set**(SnmpTarget target, SnmpRequest request)

pour obtenir un handle et, pour obtenir effectivement la réponse, l'une des méthodes de SnmpCommand :

public SnmpResponse **responseWait** (int msec)

public SnmpResponse **responseWait** (int handle, int msec)

selon que le handle présente un intérêt réel ou pas.

Le programme suivant change le nom du contact sur la machine cible :

### SNMPAgentSet.java

```
/*
 * SNMPAgentSet.java
 * Created on 20 septembre 2004, 10:00
 */

/**
 * @author vilvens
 */

import com.waremaker.snmp.*;

public class SNMPAgentSet
{
    public SNMPAgentSet() { }

    public static void main(String[] args)
    {
        SnmpFactory fac = new SnmpFactory();
        fac.setHostAddress("localhost");

        SnmpCommand com = null;
        try
        {
            com = fac.getCommand();
        }
        catch (SnmpException e)
        {
            System.out.println("Erreur d'acquisition de commande : " + e.getMessage());
            System.exit(0);
        }

        SnmpTarget cible = null;
```

```

cible = com.getTarget("10.59.5.36");

cible.setAttempts(2, 2000);
cible.setGetCommunity("Bureau11"); cible.setSetCommunity("Bureau11");
cible.setVersion(SnmpConst.VERSION_1);

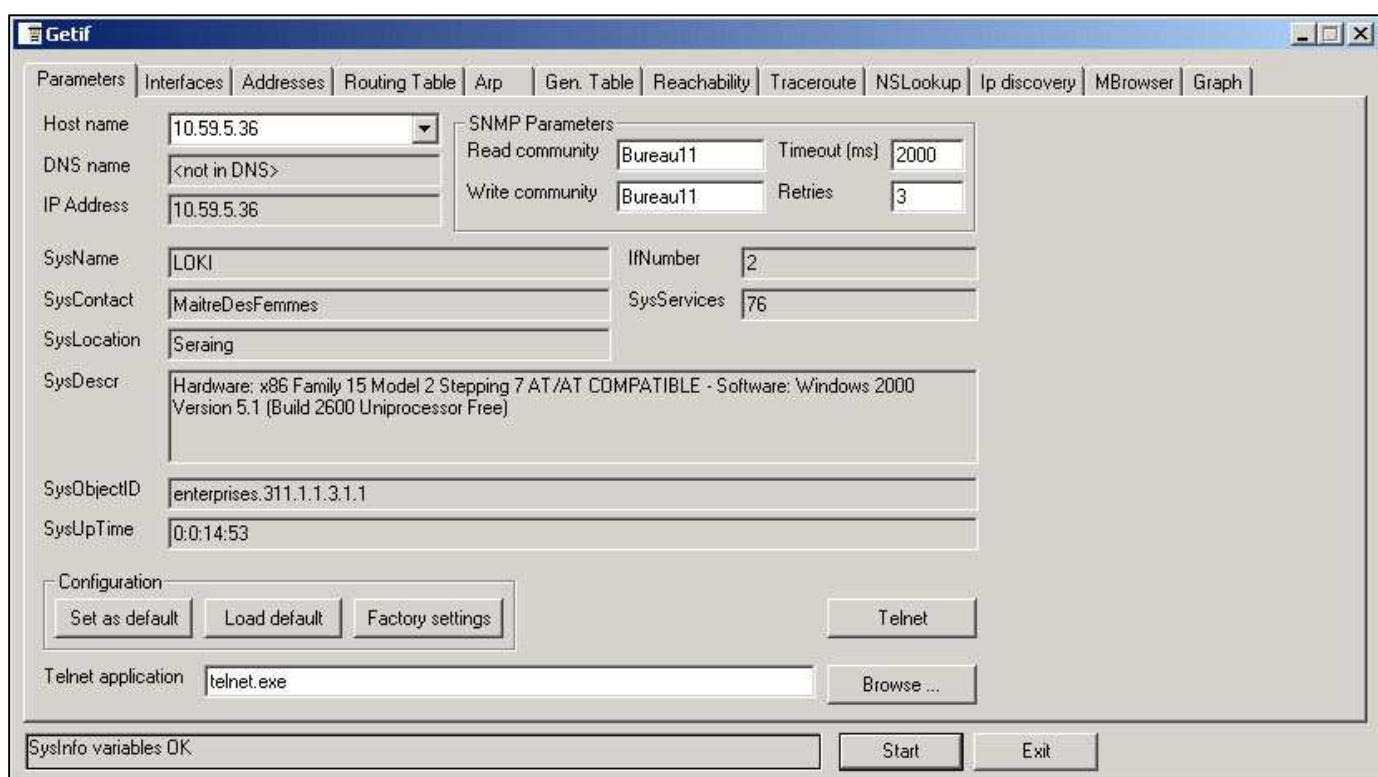
SnmpRequest requete = com.newRequest();
requete.setOid(0, ".1.3.6.1.2.1.1.4.0");
String oid = requete.getOidAsString(0);
System.out.println("OID = " + oid);
requete.setOctetStr(0, "MaitreDesFemmes");

SnmpResponse reponse = com.setWait(cible, requete, 5000);

if (reponse != null)
{
    System.out.println("Reponse = " + reponse.getValueAsString(0));
    System.out.println("Adresse = " + reponse.getAddress());
    System.out.println("Port = " + reponse.getPort());
    System.out.println("Version SNMP = " + reponse.getVersion());
    System.out.println("Communauté = " + reponse.getCommunity());
}
else
{
    System.out.println("Erreur Snmp");
}
com.shutdown();
}
}

```

On peut vérifier, par exemple avec getif, que SysContact a bien été modifié sur la machine cible :



## **12.La commande get-next en Java**

Le package implémente également la commande

```
public SnmpResponse getNextWait (SnmpTarget target, SnmpRequest request, int msec)
```

qui permet de balayer de manière séquentielle toutes les informations d'une partie de la MIB utilisée. Comme pour la commande get, si l'on préfère utiliser un handle de requête, on emploiera :

```
public int getNext(SnmpTarget target, SnmpRequest request)
```

suivi de

```
public SnmpResponse responseWait (int handle, int msec)
```

Il n'y a aucune difficulté particulière à utiliser ces méthodes. Il suffit de remarquer qu'une fois la requête initialisée sur un certain oid (par exemple, .1.3.6.1.2.1.1), il faudra, au sein d'une boucle, faire évoluer cet oid courant sur lequel chaque nouvelle requête sera initialisée. Pour cela, on utilisera la méthode héritée de **SnmpPdu** :

```
public java.lang.String getOidAsString (int index)
```

appliquée à la réponse fournie par `getNextWait()`, ce qui nous donnera l'oid suivant dans la MIB ...

### **SNMPAgentNext.java**

```
/*
 * SNMPAgentNext.java
 * Created on 14 mai 2004, 14:07
 */
/**
 * @author vilvens
 */
import com.waremaker.snmp.*;

public class SNMPAgentNext
{
    public SNMPAgentNext() { }

    public static void main(String[] args)
    {
        SnmpFactory fac = new SnmpFactory();
        fac.setHostAddress("localhost");

        SnmpCommand com = null;
        try
        {
            com = fac.getCommand();
        }
```

```

        catch (SnmpException e)
        {
            System.out.println("Erreur d'acquisition de commande : " + e.getMessage());
            System.exit(0);
        }

        SnmpTarget cible = null;
        cible = com.getTarget("10.59.5.36");

        cible.setAttempts(2, 2000);
        cible.setGetCommunity("Bureau11");
        cible.setVersion(SnmpConst.VERSION_1);

        String oidCourant = ".1.3.6.1.2.1.1";
        System.out.println("OID de base = " + oidCourant);
        SnmpResponse reponse = null;

        SnmpRequest requete;
        requete = com.newRequest();

        do
        {
            requete.setOid(0, oidCourant);
            System.out.println("OID courant = " + oidCourant);

            reponse = com.getNextWait(cible, requete, 5000);

            if (reponse !=null)
            {
                System.out.println("Reponse du next = " + reponse.getValueAsString(0));
                oidCourant = reponse.getOidAsString(0);
            }
            else
            {
                System.out.println("Erreur Snmp");
            }
        }
        while (reponse != null);

        com.shutdown();
    }
}

```

Evidemment, le résultat de l'exécution de ce programme est une avalanche :

OID de base = .1.3.6.1.2.1.1
OID courant = .1.3.6.1.2.1.1
Reponse du next = Hardware: x86 Family 15 Model 2 Stepping 7 AT/AT COMPATIBLE - Software: Windows 2000 Version 5.1 (Build 2600 Uniprocessor Free)
OID courant = 1.3.6.1.2.1.1.0

```
Reponse du next = 1.3.6.1.4.1.311.1.1.3.1.1
OID courant = 1.3.6.1.2.1.1.2.0
Reponse du next = 10428609
OID courant = 1.3.6.1.2.1.1.3.0
Reponse du next = JOraclator
OID courant = 1.3.6.1.2.1.1.4.0
Reponse du next = LOKI
OID courant = 1.3.6.1.2.1.1.5.0
Reponse du next = Seraing
OID courant = 1.3.6.1.2.1.1.6.0
Reponse du next = 76
OID courant = 1.3.6.1.2.1.1.7.0
Reponse du next = 2
OID courant = 1.3.6.1.2.1.2.1.0
Reponse du next = 1
OID courant = 1.3.6.1.2.1.2.2.1.1.1
Reponse du next = 2
OID courant = 1.3.6.1.2.1.2.2.1.1.2
Reponse du next = MS TCP Loopback interface
...
OID courant = 1.3.6.1.2.1.4.21.1.13.224.0.0.0
Reponse du next = 0.0
OID courant = 1.3.6.1.2.1.4.21.1.13.255.255.255.255
Reponse du next = 2
```

Bien évidemment, il faudrait limiter la boucle au champ réellement souhaité. Cela s'appelle un "**Snmp-Walk**" – to do ;-)

### **13.La réception des traps**

Il est possible de réceptionner les traps envoyés par un agent SNMP en procédant comme suit :

- ◆ comme d'habitude, se procurer un objet SnmpCommand
- ◆ démarrer le processus de réception des traps provenant d'une machine donnée au moyen de la méthode de SnmpCommand :

```
public void startTrapReceiver (String adresseOrigine) throws SnmpException
```

- ◆ vérifier périodiquement qu'un trap n'est pas arrivé au bout d'un certain temps au moyen de

```
public SnmpTrap trapWait (int msec)
```

si c'est le cas, le trap est encapsulé dans un objet **SnmpTrap**, classe dérivée de SnmpResponse.

- ◆ analyser le trap; l'objet trap est susceptible de fournir

- le type de trap (v1 ou v2) au moyen de la méthode getCommand() qui sera l'une des deux valeurs de SnmpConst :

public static final int **COM\_TRAP**

➔ l'origine s'obtient par la méthode de SnmpTrap :

public java.lang.String **getAgent()**

public static final int **COM\_V2TRAP**

➔ l'origine s'obtient par la méthode de SnmpPdu :

public java.lang.String **getValueAsString**(int index)

- le code du trap au moyen de la méthode de SnmpTrap :

public int **getGenericTrap()**

## 14. Les particularités des commandes SNMP v3

Dans le cas où l'on utilise SNMP v3, la communauté est remplacée par des procédures d'authentification et de cryptage basées sur les techniques et les algorithmes de cryptographie bien connus. Une requête de type get, par exemple, se programme comme pour les versions 1 et 2, si ce n'est que

♦ l'utilisateur doit être connu de l'agent qui sera sollicité; on spécifie ce **nom d'utilisateur** grâce à la méthode de SnmpTarget

public void **setUserName** (String userName)

♦ cet utilisateur peut être **authentifié** par un HMAC qui est calculé avec une clé construite sur le mot de passe selon un algorithme spécifié dans la version 3; la méthode

public void **setAuthenticationKey** (String authPassword, int authKeyType)

spécifie ce mot de passe, calculé selon l'un des algorithmes désigné par le second paramètre au moyen des constantes de SnmpConst :

public static final int **AUTH\_MD5**

public static final int **AUTH\_SHA1**

l'absence d'authentification étant indiquée par

public static final int **AUTH\_NONE**

♦ le **cryptage** éventuel doit utiliser une clé secrète que l'on créera à partir du mot de passe (à nouveau, la clé est construite sur le mot de passe selon un algorithme spécifié dans la version 3) grâce à

public void **setPrivacyKey** (String privPassword, int privKeyType)

utilisée selon l'un des algorithmes désigné par le second paramètre; la seule constante de SnmpConst disponible est :

public static final int **PRIV\_DES**

qui correspond à un DES à chiffrement de blocs CBC; l'absence de chiffrement est indiquée par

public static final int **PRIV\_NONE**

Une séquence de requête get avec authentification ressemble donc à ceci :

```
// comme un get v1 ou v2
...
SnmpTarget cible = com.getTarget("10.59.5.18");
cible.setAttempts(2, 2000);
cible.setVersion(SnmpConst.VERSION_3);
cible.setUserName("Claude");
cible.setContextName("Domicile");
cible.setAuthenticationKey("CochonDansLePre", SnmpConst.AUTH_MD5);
...
// comme un get v1 ou v2
```

## 15. La découverte du réseau

Ecrire un programme satisfaisant de gestion d'un réseau complexe, donc divisé en sous-réseaux, sort évidemment du contexte de ce cours<sup>1</sup>. Mais on peut néanmoins donner les idées de base qui seraient les chevilles d'une telle application<sup>2</sup>.

### 15.1 Programmer un ping

Avant de se demander si une machine comporte un agent SNMP, il faudrait peut-être mieux se demander si cette machine est au moins accessible par le réseau. En pratique, au niveau du système d'exploitation, nous utiliserions la commande ping. Comment faire au niveau de Java, celui-ci ne manipulant pas les commandes du protocole ICMP qui sous-tend ping ? Le seul moyen est de redescendre au niveau du système d'exploitation de la machine hôte au moyen de la classe **Runtime** de java.lang. Toute application Java possède une seule instance de cette classe, instance qui réalise l'interface avec son environnement d'exécution. On peut obtenir cet objet au moyen de la méthode

public static Runtime **getRuntime()**

de cette classe. Il est alors possible d'exécuter une commande de l'O.S. au moyen de la méthode

public Process **exec(String command)** throws IOException

---

<sup>1</sup> et même des travaux de laboratoire – ouf !

<sup>2</sup> voir le TFE de G. Donnay pour une application de ce type.

qui crée un sous-processus, représenté par une instance de la classe **Process** de `java.lang`, chargé de l'exécution de la commande passée en paramètre (des versions polymorphes permettent d'utiliser des variables d'environnement). Ce sous-processus ne dispose pas de son propre terminal ou de sa propre console : toutes les E/S sont redirigées sur des flux, que l'on peut obtenir au moyen des méthodes de la classe `Process` :

```
public abstract OutputStream getOutputStream()
public abstract InputStream getInputStream()
public abstract InputStream getErrorStream()
```

Comme les résultats d'un ping sont affichés sous forme de lignes de caractères, il nous suffit de construire sur le flux d'entrée de la commande un flux de type `Reader`, que l'on bufférise pour retrouver la notion de ligne. Si la machine visée est inaccessible, on lire une ligne du type :

Paquets : envoyés = 4, reçus = 0, perdus = 4 (perte 100%),

Il suffit donc de détecter la présence de "100%" dans une ligne pour en déduire que la machine visée ne répond pas – dans le cas contraire, elle aura au moins répondu une fois.

Ces considérations nous mènent donc à :

### ping en Java

```
public static int ping (String adresse)
/* -1: erreur ; 0: pas de réponse ; 1: réponse reçue */
{
    System.out.println("Adresse testée = " + adresse);
    Process p = null;
    String commande = "ping -n 4 -w 1000 " + adresse;
    System.out.println("Commande testée = " + commande);
    BufferedReader bfIn = null;
    try
    {
        p=Runtime.getRuntime().exec(commande);
        if (p == null)
        {
            System.out.println("## Erreur d'exécution de la commande ##"); return -1;
        }

        bfIn = new BufferedReader(new InputStreamReader(p.getInputStream()));
        String strLine;
        boolean pasDeReponse = false;
        while ((strLine = bfIn.readLine()) != null)
        {
            System.out.println(strLine); // pour trace
            if (Trouve100(strLine))
            {
                System.out.println("La machine " + adresse + " ne répond pas"); return 0;
            }
        }
    }
```

```

        bfIn.close();
        System.out.println("La machine " + adresse + " a répondu");
        return 1;
    }
    catch(IOException e)
    { System.out.println("Exception IO = " + e.getMessage());  }
    catch(Exception e)
    { System.out.println("Exception = " + e.getMessage());  }
    return -1;
}

static boolean Trouve100 (String s)
/* true : on a trouvé 100% de paquets rejetés
 * false : certains paquets n'ont pas été refusés
 */
{
    boolean trouve = false;
    StringTokenizer scan = new StringTokenizer (s, " ");
    int cpt = 0;
    while (scan.hasMoreTokens())
    {
        String essai = scan.nextToken();
        int pp = essai.indexOf("%");
        if (pp != -1)
        {
            int p100 = essai.indexOf("100");
            trouve = (p100 != -1);
        }
        if (trouve) return true;
    }
    return false;
}

```

Pour tester cette méthode, on peut alors simplement placer dans la méthode main() :

```

public static void main(String[] args)
{
    String adresseCible = "192.168.2.1";
    System.out.println("Essai de ping pour " + adresseCible + " : " + ping(adresseCible));
}

```

pour obtenir par exemple :

```

Adresse testée = 192.168.2.1
Commande testée = ping 192.168.2.1
Envoi d'une requête 'ping' sur 192.168.2.1 avec 32 octets de données:
R, ponse de 192.168.2.1: octets=32 temps<1ms TTL=128

```

Statistiques Ping pour 192.168.2.1:

Paquetsÿ: envoy,s = 4, re us = 4, perdus = 0 (perte 0%),

Dur,e approximative des boucles en millisecondes :

Minimum = 0ms, Maximum = 0ms, Moyenne = 0ms

**La machine 192.168.2.1 a r pondu**

Un d tail mineur : les caract res accentu s ne passent pas tr s bien ce qui est ´videmment du ´ a un charset non appropri .

## 15.2 Explorer un range d'adresses connues

Nous pouvons ´ a pr sent envisager de passer en revue une s rie de machines dont les adresses se trouvent dans une plage d finie. C'est ce que fait le programme suivant, en recherchant au pr alable l'adresse IP et le masque de sous-r seau. de la machine h te qui lance le programme.

### SNMPChercheAgents.java

```
/*
 * SNMPChercheAgents.java
 */
/** 
 * @author vilvens
 */

import java.io.*;
import java.util.*;
import com.waremaker.snmp.*;

public class SNMPChercheAgents
{
    public SNMPChercheAgents() {}

    public static void main(String[] args)
    {
        System.out.println("Recherche des machines");

        /* 1. Recherche des param tres de la machine */
        /* 1.1 Adresse de la machine */
        String adresseMachineHote = "localhost";
        SnmpFactory fac = new SnmpFactory();
        fac.setHostAddress("localhost");

        SnmpCommand com = null;
        try { com = fac.getCommand(); }
        catch (SnmpException e)
        {
            System.out.println("Erreur d'acquisition de commande : " + e.getMessage());
            System.exit(0);
        }
    }
}
```

```

SnmpTarget cible = null;
cible = com.getTarget("localhost");

cible.setAttempts(2, 2000);
cible.setGetCommunity("Oupeye");
cible.setVersion(SnmpConst.VERSION_1);

SnmpRequest requete = com.newRequest();
/* Recherche de l'adresse de la machine qui exécute le programme */
requete.setOid(0, ".1.3.6.1.2.1.4.20.1.1.127.0.0.1");
//ou en passant par la table de routage :
// requete.setOid(0, "1.3.6.1.2.1.4.21.1.11.127.0.0.0");
SnmpResponse reponse = com.getNextWait(cible, requete, 5000);
if (reponse !=null)
{
    System.out.println("Code SNMP : " + reponse.getStatus());
    adresseMachineHote = reponse.getValueAsString(0);
    System.out.println("Reponse pour l'adresse = " + adresseMachineHote);
    System.out.println("Adresse = " + reponse.getAddress());
    System.out.println("Port = " + reponse.getPort());
    System.out.println("Version SNMP = " + reponse.getVersion());
    System.out.println("Communauté = " + reponse.getCommunity());
}
else
{
    System.out.println("Erreur Snmp");
    Exception e = com.getException();
    while (e != null)
    {
        System.out.println("Exception : " + e.getMessage());
        e = com.getException();
    }
}

/* 1.2 Recherche du masque de sous réseau */
requete = com.newRequest();
/* Recherche du masque de la machine qui exécute le programme */
requete.setOid(0, ".1.3.6.1.2.1.4.20.1.3.127.0.0.1");
//ou en passant par la table de routage :
// requete.setOid(0, "1.3.6.1.2.1.4.21.1.11.127.0.0.0");
reponse = com.getNextWait(cible, requete, 5000);
if (reponse !=null)
{
    System.out.println("Code SNMP : " + reponse.getStatus());
    System.out.println("Reponse pour le masque = " + reponse.getValueAsString(0));
}
else
{
    System.out.println("Erreur Snmp");
    Exception e = com.getException();
}

```

```

while (e != null)
{
    System.out.println("Exception : " + e.getMessage()); e = com.getException();
}
}

/* 2. Ping d'une machine */
String baseAdresse = "192.168.2.";
int nbreMachines = 4; // par exemple
for (int i=2; i<=(2+nbreMachines); i++)
{
    String adresseCible = new String
        ( new StringBuffer(baseAdresse).append(new Integer(i).toString()) );
    System.out.println("Essai de ping pour " + adresseCible);
    int resPing = ping(adresseCible);
    switch (resPing)
    {
        case -1 : System.out.println("La commande ping sur " + adresseCible +
            " a donné une erreur"); break;
        case 0 : System.out.println("La machine " + adresseCible + " ne répond pas");
            break;
        case 1 : System.out.println("La machine " + adresseCible + " a répondu"); break;
    }
    if (resPing !=1) continue;
}

/* 3. Acquisitions d'infos de la machine */
cible = com.getTarget(adresseCible);
cible.setAttempts(2, 2000); cible.setGetCommunity("Oupeye");
cible.setVersion(SnmpConst.VERSION_1);

requete = com.newRequest();
requete.setOid(0, ".1.3.6.1.2.1.1.5.0"); // ".1.3.6.1.2.1.1.4.0";
String oid = requete.getOidAsString(0);
System.out.println("OID = " + oid);

reponse = com.getWait(cible, requete, 5000);
if (reponse !=null)
{
    System.out.println("Code SNMP : " + reponse.getStatus());
    System.out.println("Reponse = " + reponse.getValueAsString(0));
    System.out.println("Adresse = " + reponse.getAddress());
    System.out.println("Port = " + reponse.getPort());
    System.out.println("Version SNMP = " + reponse.getVersion());
    System.out.println("Communauté = " + reponse.getCommunity());
}
else
{ System.out.println("Erreur Snmp"); continue; }
} // fin du for
com.shutdown();
}

```

```
public static int ping (String adresse)
/* -1: erreur ; 0:pas de réponse ; 1: réponse reçue */
{ ... }

static boolean Trouve100 (String s)
/* true : on a trouvé 100% de paquets rejetés
 * false : certains paquets n'ont pas été refusés
 */
{ ... }
}
```

Une exécution donne :

```
Recherche des machines
Code SNMP : 0
Reponse pour l'adresse = 192.168.2.2
Adresse = 127.0.0.1
Port = 161
Version SNMP = 0
Communauté = Oupeye
Code SNMP : 0
Reponse pour le masque = 255.255.255.0
Essai de ping pour 192.168.2.2
Adresse testée = 192.168.2.2
Commande testée = ping -n 2 -w 1000 192.168.2.2
La machine 192.168.2.2 a répondu
OID = 1.3.6.1.2.1.1.5.0
Code SNMP : 0
Reponse = CLAUDE
Adresse = 192.168.2.2
Port = 161
Version SNMP = 0
Communauté = Oupeye
Essai de ping pour 192.168.2.3
Adresse testée = 192.168.2.3
Commande testée = ping -n 2 -w 1000 192.168.2.3
La machine 192.168.2.3 ne répond pas
Essai de ping pour 192.168.2.4
Adresse testée = 192.168.2.4
Commande testée = ping -n 2 -w 1000 192.168.2.4
La machine 192.168.2.4 ne répond pas
```

L'expérience montre cependant que certaines exceptions ne sont pas captées en cas d'échec d'accès à une machine – bugs à corriger ?

---

### 15.3 Déterminer la topologie du réseau : le principe

Comme nous l'avons déjà dit, nous nous contenterons ici du principe général de la découverte d'un réseau. Les étapes successives ont :

1) déterminer le sous-réseau de la machine manager et son adresse de broadcast : on trouve ainsi la plage d'adresse à scanner

2) réaliser un ping sur chaque adresse de ce range

3) pour chaque adresse où le ping a donné une réponse positive :

    3.1) rechercher le nom de la machine (sysName) : .1.3.6.1.2.1.1.5.0

    3.2) rechercher le nombre d'interfaces présents sur la machine (interfaces.ifNumber) : .1.3.6.1.2.1.2.1.0 (pour notre exemple : 2 : la carte réseau Ethernet et la carte accès distant); une boucle passe en revue ces interfaces :

        3.2.1) pour chacun d'entre eux, on se demande si cet interface est up (interfaces.ifTable.ifEntry.ifOperStatus) : respectivement .1.3.6.1.2.1.2.2.1.8.1 et .1.3.6.1.2.1.2.2.1.8.2 pour notre exemple

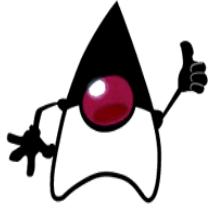
        3.2.2) si il l'est, on récupère l'adresse et le masque de sous-réseau de cette machine par .1.3.6.1.2.1.4.20.1.1.0  
(ip.ipAddrTable.ipAddrEntry.ipAdEntAddr)

        On se demande si l'on se trouve dans le même sous-réseau ou pas

            3.2.2.1) si c'est le même réseau, il reste à rechercher les informations;

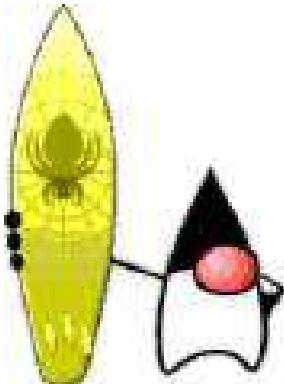
            3.2.2.2) si c'est un sous-réseau différent, on relance toute la procédure.

"Mais, mais ... ce n'est pas fini !?"



Effectivement, il resterait à programmer ces principes, ce qui se révèle demander beaucoup de temps. Or, il y a plus important et plus urgent : comment garantir la sécurité de toutes ces démarches ? SNMPv3 apporte déjà une réponse. Mais il y a autre chose : attaquons à présent ce que tant de gens craignent, à savoir le modèle de sécurité de la plate-forme Java ...

## XXIV. Le modèle de sécurité Java



*La trop grande sécurité des peuples est toujours l'avant-coureur de leur servitude.*

(J.P. Marat, L'Ami du peuple)

Pas de méprise ! Il ne s'agit pas ici des techniques cryptographiques classiques, mais bien de l'étude de l'architecture sécuritaire de Java. Accrochez vos ceintures et montrez patte blanche ;-) ...

### 1. Les chargeurs de classes

#### 1.1 La mise en mémoire d'une classe dans une machine virtuelle

On se souviendra que le chargeur de classes [class loader], comme son nom l'indique, charge dans la mémoire de la machine virtuelle Java le bytecode d'une classe pour le mettre à la disposition de l'interpréteur. Il chargera aussi les autres classes utilisées dans le code exécuté de la première classe dans un espace clos appelé un "espace de nom" [namespace]. En fait, il partitionne ainsi l'ensemble des classes Java utilisées par la machine virtuelle Java en une série de tels namespaces. L'idée sécuritaire sous-jacente est qu'une classe ne peut accéder que des objets qui se trouvent dans le même namespace. On compte un namespace pour les classes du système local (qui réalisent notamment les E/S locales) et un par source réseau (donc correspondant à des URLs différents). La conséquence est donc qu'une classe locale ne pourra jamais interagir avec une classe venue par le réseau, et vice-versa.

Le chargeur de classes de base, encore appelé le "chargeur d'amorce" [bootstrap classloader], fait partie intégrante de la machine virtuelle et est écrit en C. Il est responsable du chargement des classes de base de Java, c'est-à-dire celles du JDK – ces classes sont encore appelées les "classes d'amorce". Mais il peut exister d'autres chargeurs de classes, qui sont alors, comme nous le verrons, des objets Java appartenant à une hiérarchie de chargeurs. Tous ces chargeurs trouveront la définition de la classe à charger en recherchant le bytecode dans un fichier class qui porte à priori le nom de la classe demandée.

Cependant, la transformation du bytecode lu en un objet véritable est soumise à l'assentiment du vérificateur ...

#### 1.2 Le vérificateur de classes

L'action du chargeur est en effet complétée par le vérificateur [Java verifier] qui vérifie le format du bytecode trouvé et y recherche tout élément susceptible d'être dangereux pour la stabilité du système :

- ◆ les variables doivent être initialisées;
- ◆ les règles d'accès aux membres private ou protected doivent être respectées;
- ◆ les variables locales correspondent bien à des zones de la pile locale;
- ◆ la taille de la pile d'exécution n'est pas dépassée.

On constate donc que toute modification malveillante du bytecode sera détectée ... A remarquer que certaines vérifications sont retardées jusqu'à l'exécution : il en est ainsi des accès programmés dans les handlers d'exception (instructions catch), qui ne sont vérifiés que si l'exception en question est effectivement lancée. En cas d'accès illicite, une nouvelle exception `IllegalAccessException` sera lancée.

La vérification est suivie d'une phase de préparation (création des variables de classe, initialisation des variables à leur valeur par défaut) et d'une phase de résolution des références externes contenues dans le bytecode en des références internes à la machine virtuelle.

### **1.3 La classe de base des chargeurs complémentaires**

Il se peut donc très bien qu'un autre chargeur de classe ait été défini, cette fois comme un objet Java. Dans ce cas, son rôle est de charger les classes d'extension au JDK et les classes utilisateurs. Tout chargeur de ce type, défini donc par l'utilisateur, dérive de la classe abstraite **ClassLoader**, héritière directe d'Object et faisant bien sûr partie du package `java.lang`. Sans surprise, on y trouve la méthode

```
protected Class loadClass(String name, boolean resolve) throws ClassNotFoundException  
simplifiée en
```

```
public Class loadClass(String name) throws ClassNotFoundException
```

Mais on trouve aussi des méthodes comme :

- ◆ `protected Class findClass(String name) throws ClassNotFoundException`

dont le rôle est bien clair : trouver la définition de la classe en recherchant le bytecode dans un fichier `class` qui porte à priori le nom de la classe demandée;

- ◆ `protected final Class defineClass(String name, byte[] b, int off, int len)  
throws ClassFormatError`

qui convertit un tableau de bytes en un objet `Class`. Celui-ci ne sera effectivement utilisable qu'après l'appel de la méthode :

- ◆ `protected final void resolveClass(Class c)`

qui réalise le link, c'est-à-dire la résolution des références, si du moins ce n'est pas déjà fait.

Un véritable chargeur de classe est donc une classe héritée de `ClassLoader` (comme par exemple `SecureClassLoader`, du package `java.security`) et qui donnera une implémentation aux méthodes abstraites de sa classe mère.

Evidemment, une nouvelle classe qui est un chargeur de classe doit elle-même être d'abord chargée en mémoire ! Ceci se voit avec les constructeurs disponibles :

```
protected ClassLoader(ClassLoader parent)  
qui instancie un chargeur de classe en désignant le chargeur à utiliser pour réaliser  
cette instanciation
```

---

**protected ClassLoader()**

cas particulier où le chargeur de classe à utiliser est celui retourné par la méthode de classe :

**public static ClassLoader getSystemClassLoader()**

La classe "mère" (attention à la confusion avec la notion d'héritage) ou "parent" est encore appelée la classe "déléguée". En effet, les chargeurs de classes sont basés sur le principe de délégation :

tout chargeur de classes autre que le chargeur d'amorce demande (**délègue**) d'abord le chargement de la classe qui lui est demandé à sa classe mère/parent/déléguée; ce n'est que si celle-ci n'y parvient pas que le chargeur de classes tentera de réaliser le chargement lui-même.

La méthode de base **loadClass()** applique donc l'algorithme :

appel de **findLoadedClass()** afin de vérifier si la classe n'a pas déjà été chargée;  
*si* la classe n'est pas déjà là

*alors* - appel de la méthode **loadClass()** de la classe mère (qui elle-même réapplique le même algorithme – le chargeur d'amorce appelle plutôt **findBootstrapClass()**);  
- *si* la classe n'a pas été trouvée

*alors* appel **findClass(String)**

*si* la classe a été trouvée,

*alors* (si nécessaire) appel de **resolveClass(Class)**

*sinon* lancement de l'exception **ClassNotFoundException**.

En Java, cela donne :

### ClassLoader.java

```
package java.lang;
...
public abstract class ClassLoader
{
    ...
    private ClassLoader parent;

    protected synchronized Class loadClass(String name, boolean resolve)
        throws ClassNotFoundException
    {
        // First, check if the class has already been loaded
        Class c = findLoadedClass(name);
        if (c == null)
        {
            try
            {
                if (parent != null) {c = parent.loadClass(name, false);}
                else {c = findBootstrapClass0(name);}
            }
        }
    }
}
```

```

        catch (ClassNotFoundException e)
        {
            // If still not found, then call findClass in order to find the class.
            c = findClass(name);
        }
    }
    if (resolve) { resolveClass(c); }
    return c;
}

private Class findBootstrapClass0(String name)
    throws ClassNotFoundException
{
    check();
    return findBootstrapClass(name);
}

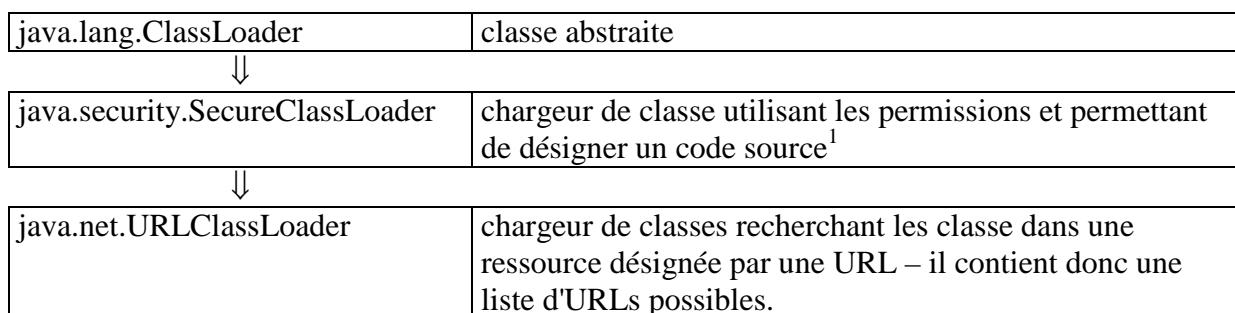
private native Class findBootstrapClass(String name)
    throws ClassNotFoundException;
...
protected Class findClass(String name) throws ClassNotFoundException
{
    throw new ClassNotFoundException(name); // hi hi ;-
}
...
}
```

On remarquera :

- ♦ la méthode native pour le chargeur d'amorce;
- ♦ la méthode **findClass()** qui doit donc être redéfinie pour tout nouveau chargeur de classe que l'utilisateur voudra définir ...

#### 1.4 Une hiérarchie de base pour les chargeurs de classes

Le JDK fournit une hiérarchie de base pour les chargeurs de classe. Cette hiérarchie permet d'introduire les notions de permissions et d'accès réseau (ce dont les applets feront certainement bon usage).



<sup>1</sup> ces termes seront explicités plus loin

Tout développeur qui souhaite écrire un chargeur de classe est invité à redéfinir la méthode **findClass()** plutôt que la méthode `loadClass()` elle-même, afin de respecter l'algorithme fondamental de recherche des chargeurs. C'est le cas ici : on peut en effet constater que la classe `URLClassLoader` redéfinit essentiellement cette méthode.

### 1.5 La référence vers le chargeur de classe

Un objet Java quelconque possède toujours une référence vers son chargeur de classe; il peut obtenir cette référence en utilisant la méthode de la classe `Class` :

```
public ClassLoader getClassLoader()
```

La référence null indique que c'est le chargeur d'amorce qui a été utilisé. Ainsi, si l'on définit une modeste classe :

#### ContratAssurance.java

```
public class ContratAssurance
{
    String numContrat;
    float prime;

    private ContratAssurance() {}

    public ContratAssurance(String n, float p) {numContrat = n; prime = p; }

    void affiche()
    {
        System.out.println("Contrat numero : " + numContrat + " – prime = " + prime);
    }
}
```

on peut instancier cette classe dans une application comme celle-ci :

#### JavaRun.java

```
public class JavaRun
{
    public static void main (String args[])
    {
        ContratAssurance ca = new ContratAssurance("OM524187", 1025);
        System.out.println("Chargeur de classe utilisé : " + ca.getClass().getClassLoader());
    }
}
```

L'exécution sur une machine Windows avec un fichier class généré par l'EDI utilisé donne dans la console :

| Chargeur de classe utilisé : sun.misc.Launcher\$AppClassLoader@71732b

Manifestement, un chargeur de classes a été défini dans le code généré : en fait, ce chargeur est du type URLClassLoader et sa liste d'URLs est celle qui est définie dans le CLASSPATH (sauf ce qui concerne les classes de l'API Java).

## 1.6 Un chargeur de classe personnalisé

Il est donc envisageable de définir son propre chargeur de classes afin de limiter les possibilités d'exécution d'un code Java en ne lui permettant que de charger certaines classes et pas d'autres. Le plus souvent, on dérive une classe chargeur de l'une des classes du JDK, soit SecureClassLoader ou URLClassLoader, ce qui évite de devoir tout implémenter soi-même. Ceci dit, on ne se résout à entreprendre un tel travail que si les chargeurs existants ne répondent pas aux exigences du moment !

Une utilisation possible, où la redéfinition d'un chargeur de classes s'impose, est le chargement de bytecode qui ne se trouve pas dans des fichiers d'extension .class, mais dans des fichiers ayant une autre extension (par exemple .btcd). Une autre est celle où la classe n'est chargée que si l'utilisateur entre un mot de passe au clavier.

A titre d'illustration, voici un chargeur de classes "fait maison".

### chargeurDeClassePersonnel.java

```
/*
 * chargeurDeClassePersonnel.java
 *
 * Created on 19 octobre 2003, 16:08
 * Thanks to Frederic Douette - InPrES
 */

import java.security.*;
import java.io.*;
import java.net.*;

public class chargeurDeClassePersonnel extends SecureClassLoader
{
    /** Creates new Chargeurdeclasse */
    public chargeurDeClassePersonnel() {}

    public chargeurDeClassePersonnel(ClassLoader parent) {super(parent);}

    /** Méthode de chargement de classe */
    public Class loadClass(String Name)throws ClassNotFoundException
    {
        Class c=null;
        ClassLoader parent =null;
        System.out.println("Loading : "+Name +"...");

        System.out.println("recherche dans les classes chargees");
        c =findLoadedClass(Name);
        if(c == null)
            try
            {
                parent = getParent();
            }
            catch(NotSerializableException e)
            {
                System.out.println("La classe "+Name+" n'est pas serialisable");
            }
        else
            System.out.println("La classe "+Name+" a été trouvée");
    }
}
```

```

if(parent != null)
{
    System.out.println("delegation vers le parent");
    c = parent.loadClass(Name);
}
else
{
    System.out.println("recherche dans les classes system");
    c = findSystemClass(Name);
}
}
catch(ClassNotFoundException classNot)
{
    System.out.println("recherche de la classe");
    c = findClass(Name);
}
if( c.getClassLoader() == null)
    System.out.println(Name + " : chargee par -> chargeur d'amorce");
else
{
    System.out.println(Name + " : chargee par -> " + .getClassLoader().getClass().
    getName());
}
System.out.println(" ");
resolveClass(c);
return c;
}

protected Class findClass(String Name) throws ClassNotFoundException
{
    Class c = null;
    String Nom = getNomClasse(Name);
    URL P = getProtocol(Name);
    String nameClass = getNomFichier(Name);
    try
    {
        boolean IO = true;
        URL url = new URL(getProtocol(Name),nameClass);
        InputStream Input = null;
        try
        {
            Input = url.openConnection().getInputStream();
        }
        catch(IOException e){IO = false;}
        if(IO)
        {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            BufferedInputStream BInput = new BufferedInputStream(Input);

            boolean eof = false;
        }
    }
}
```

```

        while(!eof)
        {
            try{
                int i = BInput.read();
                if( i == -1) eof = true;
                else baos.write(i);
            }
            catch(IOException e){ return null;}
        }

        byte[] result = baos.toByteArray();
        CodeSource codesource = new CodeSource(null,null);
        c = defineClass(Nom,result,0,result.length,codesource);
    }
    else c = null;
}
catch( MalformedURLException e){}
if ( c == null) throw new ClassNotFoundException();
return c;
}

/** Construit un tableau de byte qui est le contenu du fichier de la classe */
private byte[] loadClassData(File f) throws IOException
{
    DataInputStream input = new DataInputStream(new FileInputStream(f));
    byte[] content = new byte[(int)f.length()];
    input.readFully(content);
    return content;
}

/** Construit le chemin du fichier à charger. il remplace les '.' par des '/' */
private String getNomFichier(String NameClass){
    int index = NameClass.lastIndexOf('.');
    if(index == -1) return NameClass+".class";
    String name = NameClass.replace('.','File.separatorChar');
    return (name.substring(index)+"class");
}

private URL getProtocol(String Name)
{
    try
    {
        String base = Name;
        if(!(base.endsWith("/"))) base = base+"/";
        return new URL(base);
    }
    catch(MalformedURLException e){}
    return null;
}

```

```
private String getNomClasse(String Name){  
    int index = Name.lastIndexOf('/');  
    return Name.substring(index+1);  
}
```

On remarquera l'utilisation de

- ♦ la classe **CodeSource** (du package `java.security`) qui permet en effet d'encapsuler *l'URL d'un code source* (mais aussi, comme nous le verrons, la(les) clés utilisée(s), sous forme d'un certificat, pour vérifier la signature du code issu de cet URL). Le constructeur de cette classe se prototype donc bien naturellement :

```
public CodeSource(URL url, Certificate[] certs)
```

Pour l'instant, le 2<sup>ème</sup> paramètre sera à null.

- ♦ la méthode héritée de `SecureClassLoader` :

```
protected final Class defineClass(String name, byte[] b, int off, int len, CodeSource cs)
```

qui convertit un tableau de bytes en un objet `Class` en utilisant le code source précisé. L'objet `Class` ne sera effectivement utilisable qu'après l'appel de la méthode `resolveClass()`.

Pour instancier et utiliser notre chargeur de classe, nous utiliserons la classe suivante :

### JavaRun.java

```
/*  
 * JavaRun.java  
 *  
 * Created on 7 mars 2003, 12:17  
 */  
  
public class JavaRun  
{  
    public static void main (String args[])  
    {  
        ClassLoader cdc = (ClassLoader)new chargeurDeClassePersonnel();  
        Class cl=null;  
  
        try  
        {  
            cl = cdc.loadClass("file:e:/java-forte-application/ChargeurClasses/ContratAssurance");  
        }  
        catch (ClassNotFoundException e)  
        {  
            System.out.println ("Classe non trouvée : " + e.getMessage());  
        }  
    }  
}
```

```
Object ob=null;
try
{
    System.out.println("Instanciation par newInstance");
    ob = cl.newInstance();
}
catch(InstantiationException e)
{
    System.out.println("InstantiationException : "+e.getMessage());
    System.exit(0);
}
catch(IllegalAccessException e)
{
    System.out.println("IllegalAccessException : " +e.getMessage());
    System.exit(0);
}

System.out.println("La classe " +cl.getName()+" est chargee");
System.out.println("Nom du classLoader de l'objet instancie = "
+ob.getClass().getClassLoader().toString());

System.out.println("Instanciation par constructeur");
ContratAssurance ca = new ContratAssurance("OM524187", 1025);
System.out.println("Nom du classLoader de l'objet instancie = " +
ca.getClass().getClassLoader());
}
```

Une exécution sur la ligne de commande (mais cela donne la même chose depuis un EDI) donne :

```
E:\java-forte-application\ChargeurClasses>java JavaRun
Loading : file:e:/java-forte-application/ChargeurClasses/ContratAssurance...
recherche dans les classes chargees
delegation vers le parent
recherche de la classe
Loading : java.lang.Object...
recherche dans les classes chargees
delegation vers le parent
java.lang.Object : chargee par -> chargeur d'amorce
file:e:/java-forte-application/ChargeurClasses/ContratAssurance : chargee par ->
chargeurDeClassePersonnel
```

Instanciation par **newInstance**

La classe ContratAssurance est chargee

Nom du classLoader de l'objet instancie = chargeurDeClassePersonnel@3169f8

Instanciation par **constructeur**

Nom du classLoader de l'objet instancie = sun.misc.Launcher\$AppClassLoader@40453

6

---

Mais tout le monde a-t-il le droit de redéfinir ainsi les règles du jeu ? Bien sûr que non !

## 2. Le Security Manager

### 2.1 Le big brother des méthodes sensibles

Toute application Java, et à fortiori toute applet Java, apporte avec elle toute une série de risques potentiels pour le système sur lequel elle fonctionne : lecture ou écriture de fichiers, connexions et trafic réseau, lancement de threads, accès aux données du système, etc. Toutes ces opérations, qui peuvent se révéler dangereuses, doivent évidemment pouvoir être surveillées afin d'éviter des intrusions ou des actions malvenues.

Par défaut,

les méthodes sensibles des classes du package Java et de celles du run-time invoquent l'une ou l'autre des **méthodes de contrôle** d'un objet de surveillance instanciant la classe **SecurityManager** (du package java.lang).

Ces méthodes, telles checkExit(), checkConnect(), checkRead(), checkWrite(), etc, vérifient si l'opération qui leur est associée peut être autorisée ou pas. Selon qu'elles lancent une exception ou pas, les méthodes de librairie seront interrompues ou au contraire menées à bien. Autrement dit, une opération OP sera réalisée selon l'algorithme :

tentative d'accès à l'objet SecurityManager  
*si* accès réussi

*alors* appel de sa méthode **checkOP()**

*si* autorisation accordée

*alors* opération OP poursuivie et menée à bien

*sinon* lancement d'une exception **SecurityException**

*sinon* opération OP poursuivie et menée à bien, puisqu'il n'y a pas de SecurityManager

L'accès à un éventuel SecurityManager peut se faire au moyen de la méthode de la classe System :

```
public static SecurityManager getSecurityManager()
```

Une valeur null signifie évidemment qu'il n'y en a pas. Par défaut :

- ◆ une application n'a pas de SecurityManager;
- ◆ une applet possède toujours un SecurityManager, imposé par le browser ou le viewer d'applets qui l'ont chargé dès leur démarrage.

Ainsi, une application quelconque des chapitres précédents qui exécute le code :

```
SecurityManager sm = System.getSecurityManager();
if (sm == null) System.out.println("Pas de surveillance !");
else System.out.println("Nous sommes surveillés!");
```

constatera toujours qu'elle n'est pas surveillée.

Il est cependant toujours possible de définir un objet de sécurité pour une application. Il suffit pour cela de créer une classe de sécurité dérivée de la classe SecurityManager. Celle-ci est en fait une classe incomplète : elle fournit un interface et une implémentation partielle, qu'il convient évidemment de redéfinir selon ses besoins. On rendra ensuite actif ce nouveau manager en utilisant la méthode de la classe System :

```
public static void setSecurityManager(SecurityManager s)
```

## **2.2 Les méthodes checkXXX()**

Les méthodes qui peuvent être redéfinies sont listées ci-dessous, en fonction du domaine sur lequel elles portent :

domaine concerné	méthodes de contrôle
système de fichier	public void <b>checkRead</b> (FileDescriptor fd)
	public void <b>checkRead</b> (String file)
	public void <b>checkRead</b> (String file, Object context)
	public void <b>checkWrite</b> (FileDescriptor fd)
	public void <b>checkWrite</b> (String file)
threads	public void <b>checkDelete</b> (String file)
	public void <b>checkLink</b> (String lib)
commandes système	public void <b>checkAccess</b> (Thread t)
	public void <b>checkAccess</b> (ThreadGroup g)
sockets	public void <b>checkExec</b> (String cmd)
	public void <b>checkPrintJobAccess</b> ()
	public void <b>checkConnect</b> (String host, int port)
	public void <b>checkConnect</b> (String host, int port, Object context)
	public void <b>checkListen</b> (int port)
chargeur de classe	public void <b>checkAccept</b> (String host, int port)
	public void <b>checkMulticast</b> (InetAddress maddr)
	public void <b>checkMulticast</b> (InetAddress maddr, byte ttl)
interpréteur-machine virtuelle	public void <b>checkSetFactory</b> ()
	public void <b>checkCreateClassLoader</b> ()
	public void <b>checkExit</b> (int status)
packages	public void <b>checkMemberAccess</b> (Class clazz, int which)
	public void <b>checkPackageAccess</b> (String pkg)
properties	public void <b>checkPackageDefinition</b> (String pkg)
	public void <b>checkPropertiesAccess</b> ()
GUIs	public void <b>checkPropertyAccess</b> (String key)
	public boolean <b>checkTopLevelWindow</b> (Object window)
	public void <b>checkSystemClipboardAccess</b> ()
	public void <b>checkAwtEventQueueAccess</b> ()

## 2.3 Un SecurityManager à mot de passe

Un exemple classique est de soumettre les opérations délicates à la délivrance d'un mot de passe. Celui-ci constituera une variable membre du SecurityManager.

On définit ci-dessous ainsi une classe de sécurité qui demande l'introduction d'un mot de passe lorsque l'on tente de lire un fichier sur la machine hôte – l'exemple est classique mais démonstratif. Ceci implique de redéfinir les trois méthodes checkRead() de SecurityManager :

### Une ébauche de Security Manager personnalisé (orienté fichiers)

```
import java.io.*;

class PassswordManager extends SecurityManager
{
    private String password;

    PassswordManager(String motDePasse)
    { super(); password = motDePasse; }

    private boolean Ok()
    {
        DataInputStream entree = new DataInputStream(System.in);
        String passwordProposé;
        boolean autorisé;
        System.out.println("Veuillez entrer le mot de passe autorisant l'accès au fichier : ");
        try
        {
            passwordProposé = entree.readLine();
            if (passwordProposé.equals(password)) autorisé = true;
            else autorisé = false;
        }
        catch (IOException e)
        {
            autorisé = false;
        }
        return autorisé;
    }

    public void checkRead(FileDescriptor fd)
    {
        if (!Ok()) throw new SecurityException("Accès (fd) en lecture refusé !");
    }

    public void checkRead(String nf)
    {
        if (!Ok()) throw new SecurityException("Accès (nf) en lecture refusé !");
    }

    public void checkRead(String nf, Object contexte)
    {
        if (!Ok()) throw new SecurityException("Accès (nf+c) en lecture refusé !");
    }
}
```

On peut tester l'efficacité de SecurityManager "maison" avec :

### FichierSecuriteManager.java

```
import java.io.*;
public class FichierSecuriteManager
{
    public static void main(String args[])
    {
        try
        {
            System.setSecurityManager(new PassswordManager("Alleluia"));
        }
        catch (SecurityException e)
        {
            System.err.println ("Oups ! Il y a déjà un surveillant !");
        }

        try
        {
            FileInputStream fis =
                new FileInputStream("e:\\java-application\\FichierSecuriteManager\\citation.txt");
            DataInputStream dis = new DataInputStream(fis);

            String ligne;
            while ( (ligne=dis.readLine()) != null ) System.out.println(ligne);
            dis.close();
        }
        catch (FileNotFoundException e)
        {
            System.err.println("Erreur ! Fichier non trouvé [" + e + "]");
        }
        catch (IOException e)
        {
            System.err.println("Erreur ! ? [" + e + "]");
        }
    }
}
```

Une exécution avec un mot de passe erroné donne :

Veuillez entrer le mot de passe autorisant l'accès au fichier :

*Tous pour un !*

**java.lang.SecurityException: Accès (nf) en lecture refusé !**

```
at java.lang.Throwable.<init>(Compiled Code)
at java.lang.Exception.<init>(Compiled Code)
at java.lang.RuntimeException.<init>(Compiled Code)
at java.lang.SecurityException.<init>(Compiled Code)
at PassswordManager.checkRead(Compiled Code)
at java.io.FileInputStream.<init>(Compiled Code)
at FichierSecuriteManager.main(Compiled Code)
```

tandis que l'introduction du mot de passe correct donne :

Veuillez entrer le mot de passe autorisant l'accès au fichier :

*Alleluia*

Ce qui se fait par amour  
se fait toujours  
par delà le bien et le mal.

### **3. Le contrôleur d'accès**

#### **3.1 Vers un contrôle plus fin**

Le Security manager qui vient d'être évoqué fournit un dispositif de contrôle au niveau global. En effet, il permet de définir les opérations qu'un code Java peut réaliser ou pas – par exemple, le fait que la lecture des fichiers soit permise ou pas, quel que soit le fichier. Il existe cependant un moyen permettant de définir des droits plus finement pour un code donné – par exemple, permettre la lecture d'un certain fichier et refuser celle d'un autre. Ce moyen est un contrôleur d'accès matérialisé par un objet instanciant la classe **AccessController** du package `java.security`. Pour atteindre un tel objectif, il faut donc pouvoir exprimer très précisément quelles permissions vis à vis de telle ou telle ressource peuvent être octroyées à tel ou tel code. On dispose pour cela d'un certain nombre de classes qui correspondent à ces concepts.

#### **3.2 Le code source**

Bien logiquement, il faut tout d'abord pouvoir désigner le code dont on veut vérifier les droits ou auquel on envisage d'accorder de tels droits. La classe **CodeSource** (du package `java.security`), déjà évoquée, permet certes d'encapsuler l'URL d'un code source, mais aussi de conserver la(les) clés utilisée(s) pour vérifier la signature du code issu de cet URL (les clés sont en fait mémorisées au moyen de certificats). Pour rappel, le constructeur de cette classe se prototype selon :

```
public CodeSource(URL url, Certificate[] certs)
```

et on peut bien sûr en obtenir les deux composantes fondamentales :

```
public final URL getLocation()  
public final Certificate[] getCertificates()
```

#### **3.3 Les permissions**

Une permission quelconque est matérialisée dans un objet instance de la classe abstraite **Permission** du package `java.security` (directement dérivée d'`Object`) ou de l'une de ses classes dérivées. Ainsi, par exemple, on peut y trouver le nom d'un fichier et les types d'actions qu'il est permis de réaliser dessus (comme lire, écrire ou effacer). Une telle permission se construit avec un appel de constructeur du type :

```
FilePermission fp = new FilePermission ("\\budget\\b2006", "read");
```

Le concept de permission encapsule donc :

- ◆ un type, qui est défini par la nature de la classe instanciée;
- ◆ un nom, dont la signification dépend du type de permission;
- ◆ une action, dont l'intitulé dépend également du type de permission.

Il existe 11 classes de permissions standard, dont le nom reflète le type de permission (comme `FilePermission`), qui sont final et qui encapsulent les informations spécifiques nécessaires à cette permission (voir schéma ci-contre).

Permission est une classe abstraite et ses classes dérivées doivent donc implémenter ses méthodes. Parmi celles-ci, remarquons

```
public abstract boolean implies(Permission permission)
```

qui permet de vérifier si la permission courante implique la permission passée comme paramètre ou pas – c'est le fondement même de la logique des permissions.

### Remarques

**1)** On peut définir ses propres permissions en dérivant des classes de `Permission` ou, plus simple, de `BasicPermission`.

**2)** Il existe, au sein du même package `java.security`, une classe abstraite **PermissionCollection** qui n'est rien d'autre qu'un container de permissions (on parle encore d'agrégats de permissions). Les deux méthodes attendues sont :

```
public abstract void add(Permission permission)
public abstract boolean implies(Permission permission)
```

qui permet de vérifier si il existe dans le container une (ou plusieurs permissions) qui implique(nt) la permission passée comme paramètre ou pas.

**3)** La classe **Permissions** implémente cette classe abstraite. Comme elle peut contenir des permissions de divers types (en fait, elle est un container d'objets **PermissionCollection**), elle constitue le siège idéal de toutes les permissions accordées à une application.

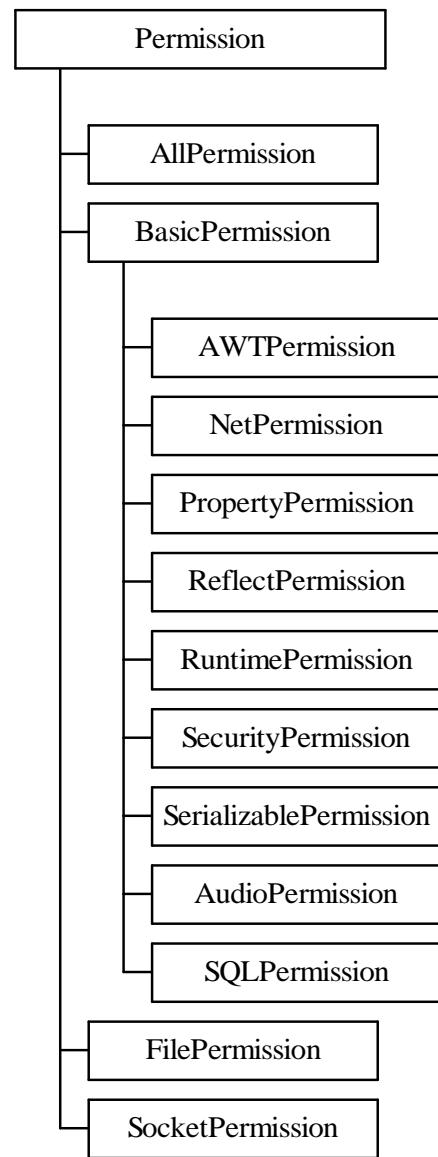
### 3.4 Le fichier et la classe Policy

La mise en correspondance des codes sources avec les permissions qui leur sont attribuées est établie dans un fichier nommé **`java.policy`**, situé dans un répertoire `security` du répertoire lib du JRE. Par défaut, il ressemble à ceci :

```
java.policy
// Standard extensions get all permissions by default

grant codeBase "file:${java.home}/lib/ext/*" {
    permission java.security.AllPermission;
};

// default permissions granted to all domains
```



```

grant {
    // Allows any thread to stop itself using the java.lang.Thread.stop()
    // method that takes no argument.
    // Note that this permission is granted by default only to remain
    // backwards compatible.
    // It is strongly recommended that you either remove this permission
    // from this policy file or further restrict it to code sources
    // that you specify, because Thread.stop() is potentially unsafe.
    // See "http://java.sun.com/notes" for more information.
    permission java.lang.RuntimePermission "stopThread";

    // allows anyone to listen on un-privileged ports
    permission java.net.SocketPermission "localhost:1024-", "listen";

    // "standard" properties that can be read by anyone
    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
    permission java.util.PropertyPermission "java.vendor.url", "read";
    permission java.util.PropertyPermission "java.class.version", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";
    permission java.util.PropertyPermission "os.arch", "read";
    permission java.util.PropertyPermission "file.separator", "read";
    permission java.util.PropertyPermission "path.separator", "read";
    permission java.util.PropertyPermission "line.separator", "read";
    permission java.util.PropertyPermission "java.specification.version", "read";
    permission java.util.PropertyPermission "java.specification.vendor", "read";
    permission java.util.PropertyPermission "java.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.specification.version", "read";
    permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
    permission java.util.PropertyPermission "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read";
};

}

```

La syntaxe des lignes de ce fichier est en fait :

```

grant [signedBy <signataire>], [codeBase <chemin>]
{
    permission <classe de permission> <nom de permission> <action>;
    ...
}

```

Ainsi, il faut comprendre avec la clause :

```

grant codeBase "file:${java.home}/lib/ext/*"
{
    permission java.security.AllPermission;
};

```

que tous les fichiers class ou jar se trouvant dans le répertoire dont le chemin est précisé disposent de toutes les permissions tandis que :

```
grant
{
    permission java.net.SocketPermission "localhost:1024-", "listen";
}
```

permet à tout le monde de se mettre en écoute sur un port de numéro supérieur à 1024.

La classe abstraite **Policy** ne fait que contenir les informations du fichier java.policy et en permet un usage aisément dans du code Java. Ainsi, on trouve la méthode :

```
public abstract PermissionCollection getPermissions(CodeSource codesource)
```

Bien sûr, il ne peut y avoir qu'une seule instance de cette classe au sein de la machine virtuelle et on peut l'obtenir à tout moment au moyen de la méthode de classe :

```
public static Policy getPolicy()
```

Cependant, contrairement à ce qui se passe pour le gestionnaire de sécurité, il est possible de modifier la politique de sécurité au moyen de la méthode de classe :

```
public static void setPolicy(Policy policy)
```

### 3.5 La gestion des Policy

On peut évidemment se demander d'où provient l'objet Policy par défaut. En fait, le fichier java.security, déjà rencontré lors de notre étude de la cryptographie en Java, contient les indications nécessaires :

**jdk1.4.2\_03\jre\lib\security\java.security**  
**ou Program Files\Java\jre1.6.0\_06\lib\security**

```
...
#
# Class to instantiate as the system Policy. This is the name of the class
# that will be used as the Policy object.
#
policy.provider=sun.security.provider.PolicyFile
...

#
# The default is to have a single system-wide policy file,
# and a policy file in the user's home directory.
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

#
# whether or not we expand properties in the policy file
# if this is set to false, properties (${...}) will not be expanded in policy
# files.
policy.expandProperties=true
```

```
# whether or not we allow an extra policy to be passed on the command line
# with -Djava.security.policy=somefile. Comment out this line to disable
# this feature.
policy.allowSystemProperty=true

# whether or not we look into the IdentityScope for trusted Identities
# when encountering a 1.1 signed JAR file. If the identity is found
# and is trusted, we grant it AllPermission.
policy.ignoreIdentityScope=false

...
```

L'entrée "policy.provider" de ce fichier indique la classe à instancier pour obtenir une instance de la classe Policy. Ici, il s'agit d'une classe fournie par Sun nommée **PolicyFile** (dissimulée dans rt.jar) et qui lit les permissions dans les fichiers dont les URLs sont spécifiées par les entrées de type "policy.url.<i>" – ici :

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy
```

On peut encore remarquer qu'un fichier policy peut être spécifié sur la ligne de commande; c'est ce que précise la clause "policy.allowSystemProperty=true". On définit alors cette propriété sur la ligne de commande par

```
java -Djava.security.policy=machin.policy
```

Dans ce cas, ce fichier policy est le seul qui sera lu. Enfin, si aucun fichier policy n'a été lu, d'une manière ou d'une autre, c'est un jeu de permissions statiques internes, semblable à celui du fichier java.policy, qui est utilisé.

### **3.6 Le retour du Security Manager**

A remarquer : avant de réaliser le changement de politique, la méthode setPolicy() évoquée à l'instant commence par appeler la méthode du SecurityManager :

```
public void checkPermission(Permission perm)
```

en lui passant comme argument une référence à un objet SecurityPermission("setPolicy").

En fait, depuis le JDK 1.2, il en est ainsi pour toutes les méthodes checkXXX() afin de déterminer si le code qui souhaite réaliser l'opération demandée possède les droits pour le faire. Par défaut, cette méthode checkPermission() appelle la méthode du même nom de la classe AccessController ...

### **3.7 L'ange gardien : le contrôleur d'accès**

En fait, cet acteur, apparu depuis le JDK 1.2, est le véritable décideur, même si le SecurityManager ne fait pas systématiquement appel à un AccessController. Bien sûr, le SecurityManager doit être conservé pour sauvegarder la compatibilité avec les JDK antérieurs.

**Un contrôleur d'accès permet de déterminer si l'accès à une ressource sensible du système doit être autorisé ou pas, sur base de la politique de sécurité courante.** La classe

qui lui correspond est la classe *final* **AccessController** (du package `java.security`). En réalité, cette n'est pas instanciée car les méthodes qu'elle apporte sont toutes statiques.

**1)** La méthode que l'on remarque immédiatement est :

```
public static void checkPermission(Permission perm) throws AccessControlException
```

Cette méthode vérifie donc si l'on possède la permission correspondant à l'objet permission (qui instancie l'une des classes permissions possibles) passé comme paramètre. Si ce n'est pas le cas, une exception de type **AccessControlException** est lancée. Par exemple, on vérifiera que l'on a le droit de lire le fichier listeEspions par le fait que :

```
FilePermission perm = new FilePermission("/secret/listeEspions", "read");  
AccessController.checkPermission(perm);
```

ne lance pas d'exception et permet donc de poursuivre. Mais comment trouve-t-on la liste de permissions du code ? Par le *domaine de protection* associé à ce code – nous allons en reparler ...

**2)** Signalons auparavant qu'il est aussi possible de permettre un accès à une ressource *de manière temporaire* en accordant au code concerné un **privilège**. C'est le rôle de la méthode polymorphe :

```
public static Object doPrivileged (PrivilegedAction action)
```

où **PrivilegedAction** est un interface qui correspond à un groupe d'instructions devant être exécuté avec des priviléges et sans exception à capter. Si des exceptions sont à prévoir, on emploiera plutôt

```
public static Object doPrivileged(PrivilegedExceptionAction action)  
throws PrivilegedActionException
```

la classe jouant le même rôle. Ces deux interfaces ne réclament qu'une méthode, respectivement

```
public Object run()                et                public Object run() throws Exception,
```

qui représente évidemment le code à exécuter. La séquence classique d'exécution d'un code privilégié est donc :

```
maMethode()  
{  
    ... // code normal  
    AccessController.doPrivileged  
    (  
        new PrivilegedAction()  
        {  
            public Object run()  
            {     ... // code à privilégier  
                return null; // si on ne renvoie rien, un objet sinon  
            }  
        }  
    );  
    ... // code normal  
}
```

### **3.8 Les domaines de protection**

*A chaque classe qu'il charge, le chargeur de classe associe un domaine de protection,* objet instanciant **ProtectionDomain** (du package java.security), classe qui regroupe un ensemble de classes dont les instances fonctionnent avec le même jeu de permissions. Bien clairement, une classe ne peut appartenir qu'à un seul domaine de protection.

De manière plus programmatique, cela signifie qu'un tel objet encapsule un objet CodeSource et un objet PermissionCollection. Son unique constructeur ne demande d'ailleurs rien d'autres :

```
public ProtectionDomain(CodeSource codesource, PermissionCollection permissions)
```

Quand une méthode essaie de réaliser un accès, c'est le domaine de protection de sa classe qui permettra de déterminer si cet accès est autorisé ou pas. Toute classe peut en effet récupérer son domaine de protection au moyen de la méthode de la classe Class :

```
public ProtectionDomain getProtectionDomain()
```

Mais, surtout, le chargeur de classes utilise dans ce contexte (plus précisément dans sa méthode findClass()) la méthode :

```
protected final Class defineClass(String name, byte[] b, int off, int len,  
        ProtectionDomain protectionDomain) throws ClassFormatError
```

ce qui permet d'obtenir les certificats nécessaires éventuels par l'intermédiaire du CodeSource. La version polymorphe

```
protected final Class defineClass(String name, byte[] b, int off, int len)  
        throws ClassFormatError
```

utilise bien entendu le domaine de protection par défaut. On peut encore remarquer que la première classe d'un package détermine ces certificats pour toutes les autres classes du package.

#### **Remarque**

Les domaines de protection existants sont récupérés dans un tableau encapsulé dans un objet instanciant une classe AccessControlContext.

### **3.9 En résumé**

Une tentative d'accès contrôlé par une méthode suit les étapes suivantes :

l'appel de la méthode provoque celui d'une méthode checkXXX() du SecurityManager  
*si* il n'existe pas

*alors* l'accès est permis - FIN

*sinon* - appel de la méthode checkPermission()

- appel de la méthode checkPermission() d'AccessController

- cette méthode recherche les permissions dans le ProtectionDomain associé à la classe de la méthode considérée

- *si* la permission nécessaire existe

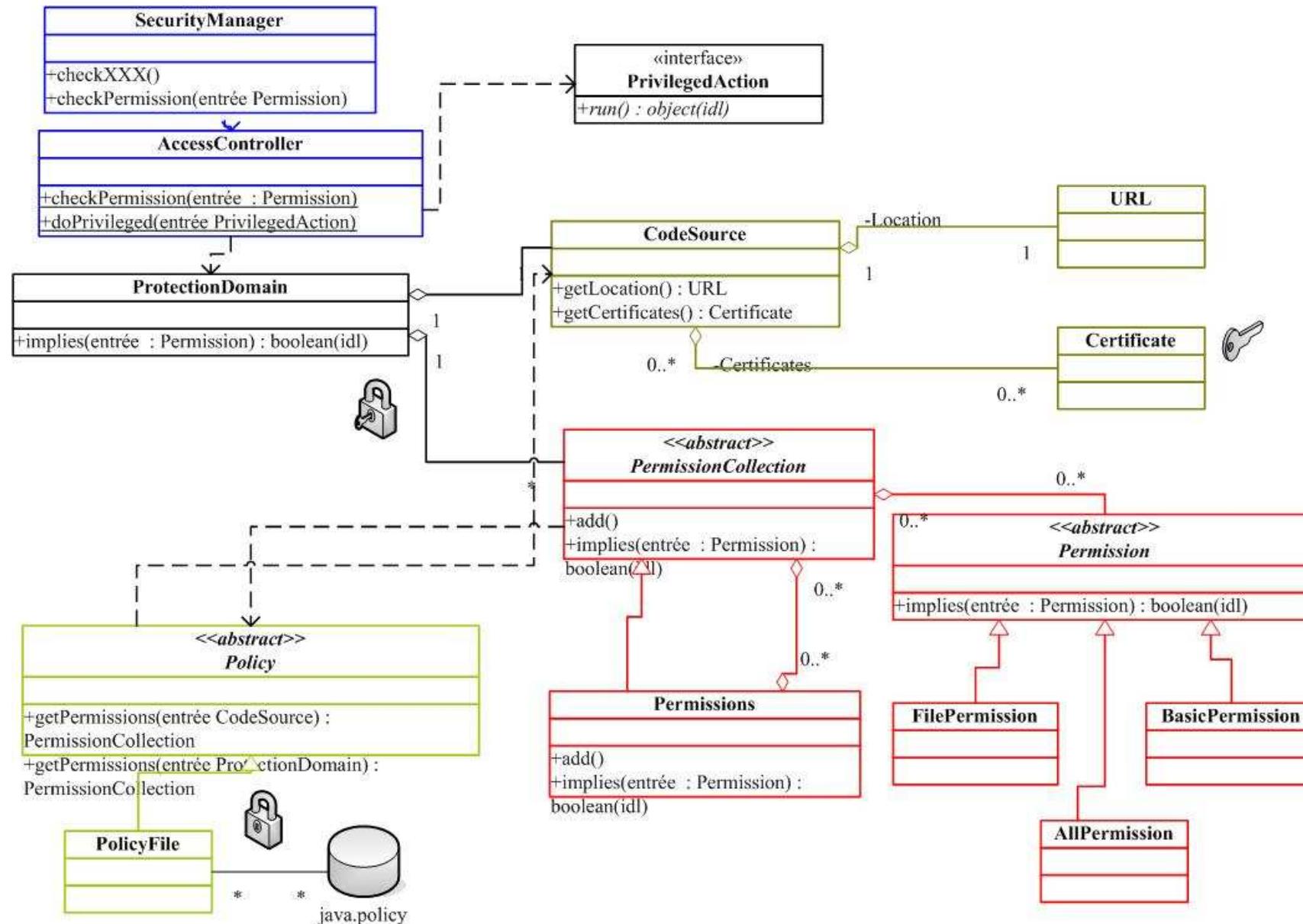
*alors* l'accès est autorisé

*sinon* l'exception AccessControlException est lancée

---

et l'on peut synthétiser avec un diagramme de classe UML :

---

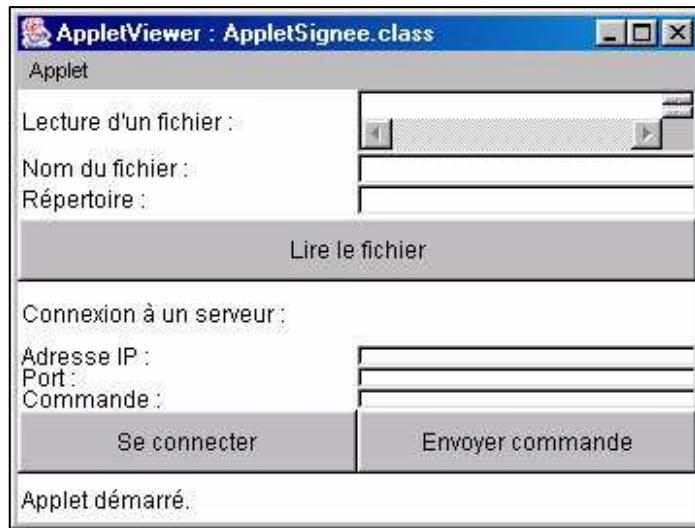


## 4. L'ouverture de la sécurité d'une applet

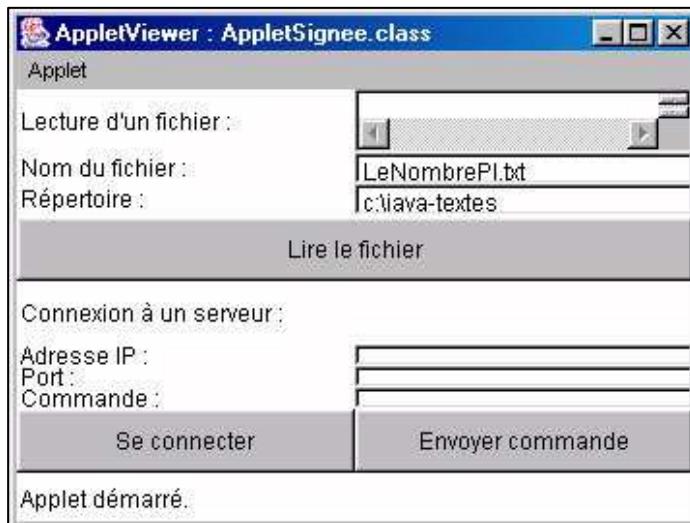
On l'a assez dit, une applet fonctionne, par défaut, dans la sandbox, avec toutes les restrictions que cela suppose. Il se peut cependant que, sans vouloir renier la politique générale d'innocence d'un code téléchargé, on souhaite permettre l'exécution de certaines opérations jugées utiles et sans danger.

### 4.1 Une applet locale qui veut lire un fichier depuis l'EDI

Considérons ainsi l'essai suivant : une applet propose un GUI proposant deux opérations "sensibles", la lecture d'un fichier et la connexion à un serveur réseau :



Son code ne présente guère d'intérêt ici – qu'il nous suffise de savoir qu'elle s'appelle "AppletSignee". L'essai sera d'exécuter cette applet en local et de tenter la lecture d'un fichier texte "LeNombrePI" situé dans le répertoire "java-textes" :

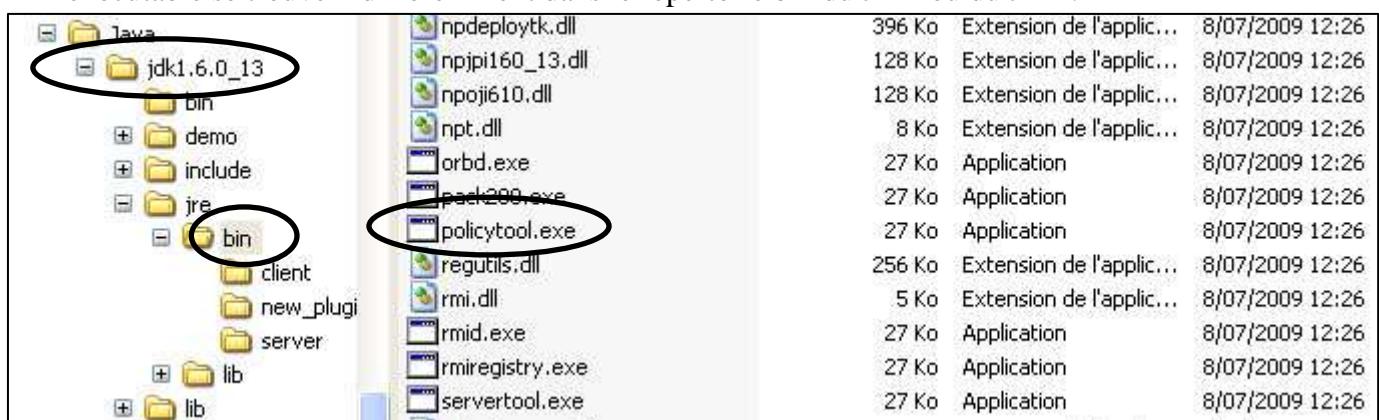


Le résultat ne se fait pas attendre :

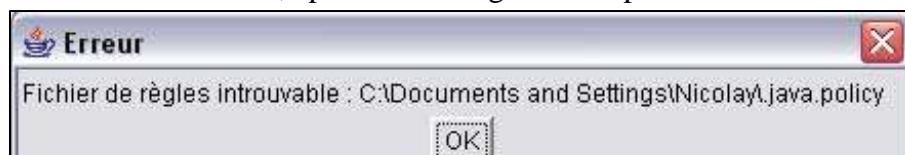
```
Tentative de lecture du fichier : c:\java-textes\LeNombrePI  
Exception occurred during event dispatching:
```

```
java.security.AccessControlException: access denied (java.io.FilePermission
c:\java-textes\LeNombrePI read)
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:272)
    at java.security.AccessController.checkPermission(AccessController.java:399)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:545)
    at java.lang.SecurityManager.checkRead(SecurityManager.java:890)
    at java.io.FileInputStream.<init>(FileInputStream.java:61)
    at AppletSignee.button2ActionPerformed(AppletSignee.java:134)
    ...
    at java.awt.EventDispatchThread.run(EventDispatchThread.java:85)
```

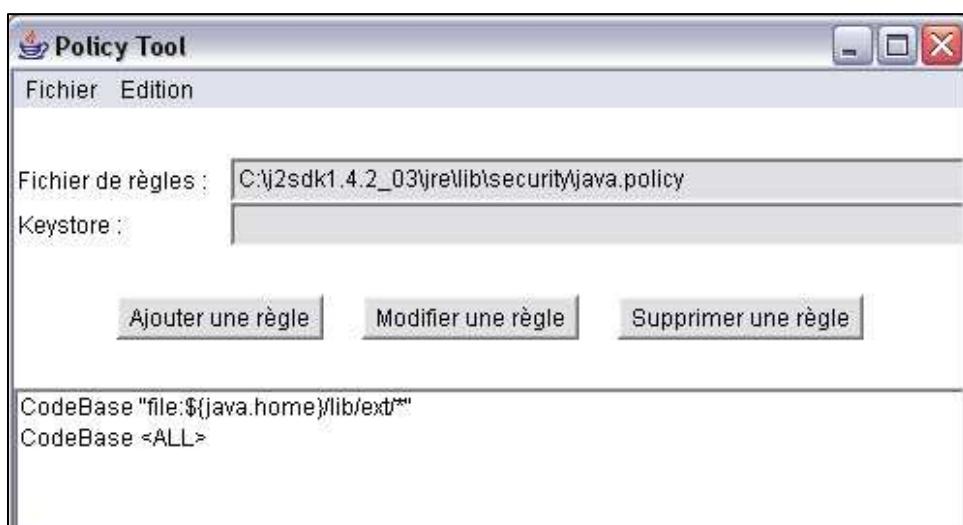
On remarquera la chaîne d'appels des méthodes vérifiant les permissions. Donc, cela ne vas pas ... et il nous faut donner à notre applet la permission ad hoc. Plutôt que de nous égarer dans le fichier **java.policy**, nous allons utiliser un outil de la batterie Java : le **policytool**. Cet exécutable se trouve indifféremment dans le répertoire bin du JDK ou du JRE :



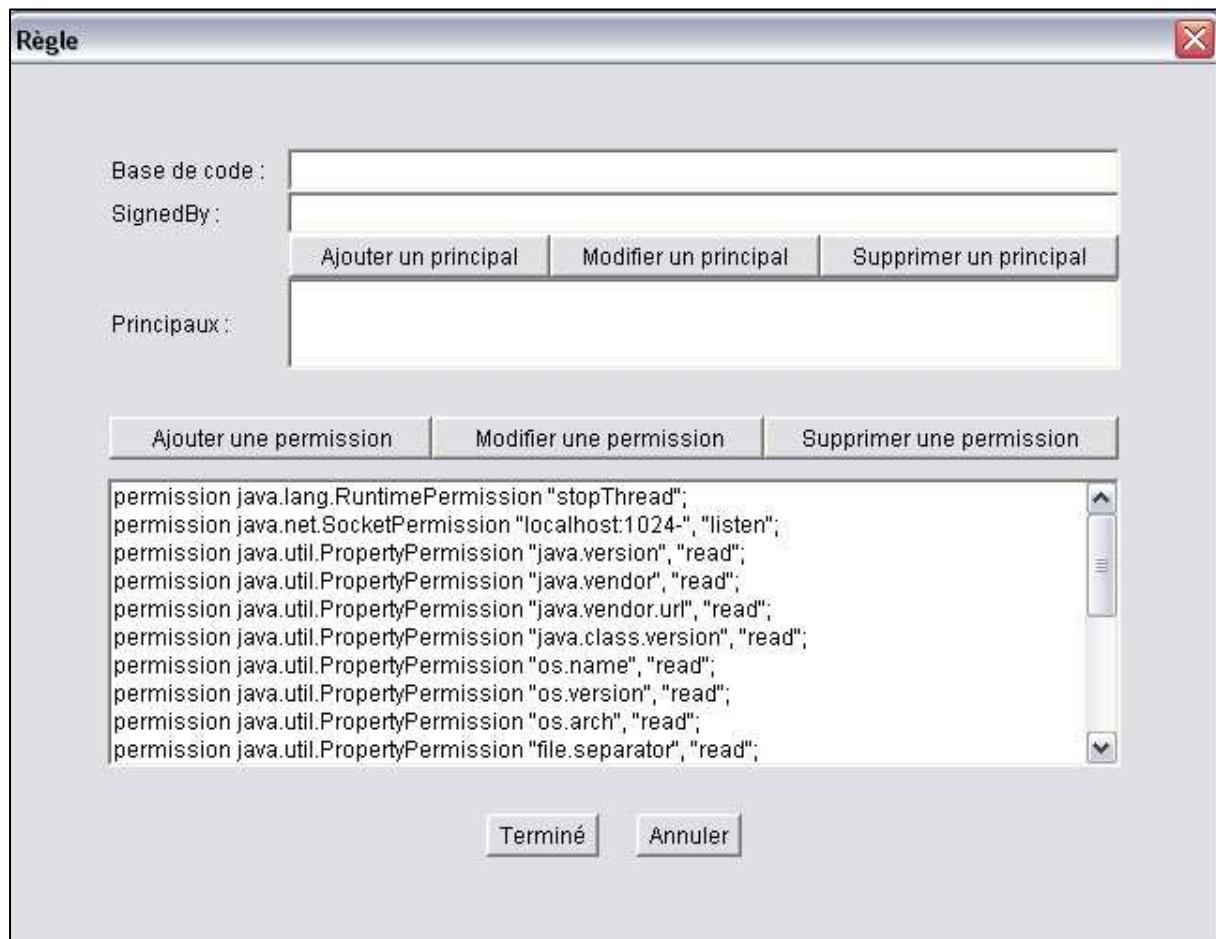
Le lancement de cet exécutable, après un message sans importance :



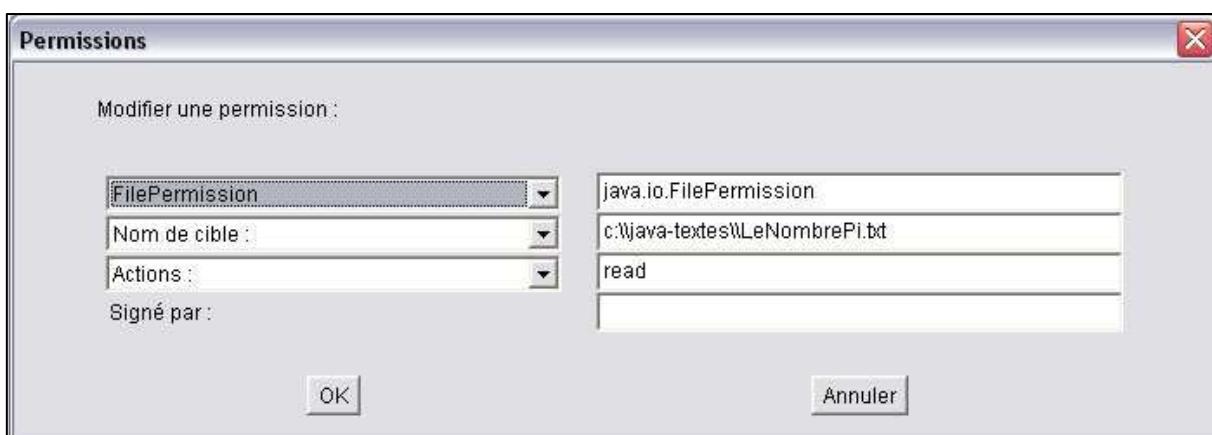
donne un GUI élémentaire avec un menu tout aussi réduit.. Dans ce dernier, on choisi Fichier→Ouvrir pour ouvrir le fichier **java.policy** (des JDK 1.3 , 1.4 ou 1.5) :



La sélection de la ligne "CodeBase <ALL>", par exemple et l'appui sur le bouton "Modifier ..." donne l'explicitation de ce que l'on trouve dans le fichier policy pour les codes quelconques (un "principal" est simplement un utilisateur ou une machine) :



Nous pouvons donc définir une nouvelle permission pour le fichier à faire lire par nos applets :



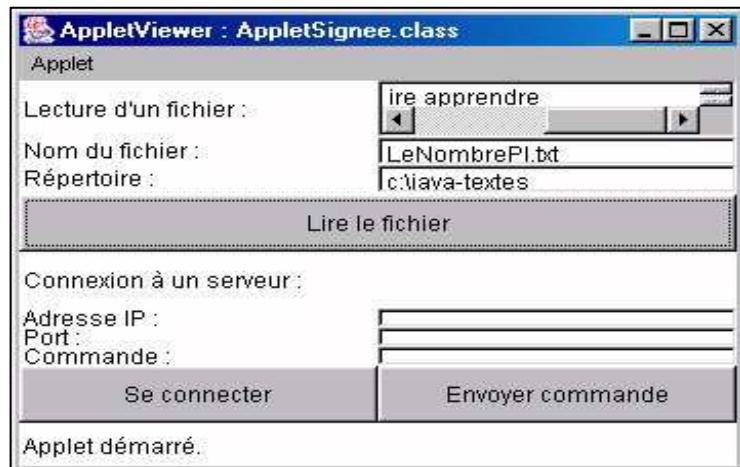
Ceci donner dans le fichier policy :

```

grant {
    permission java.io.FilePermission "c:\\java-textes\\LeNombrePi.txt", "read";
};

```

Une fois le fichier sauvegardé dans sa nouvelle version, une nouvelle tentative donne cette fois :



et sur la console :

Tentative de lecture du fichier : c:\java-textes\LeNombrePI.txt

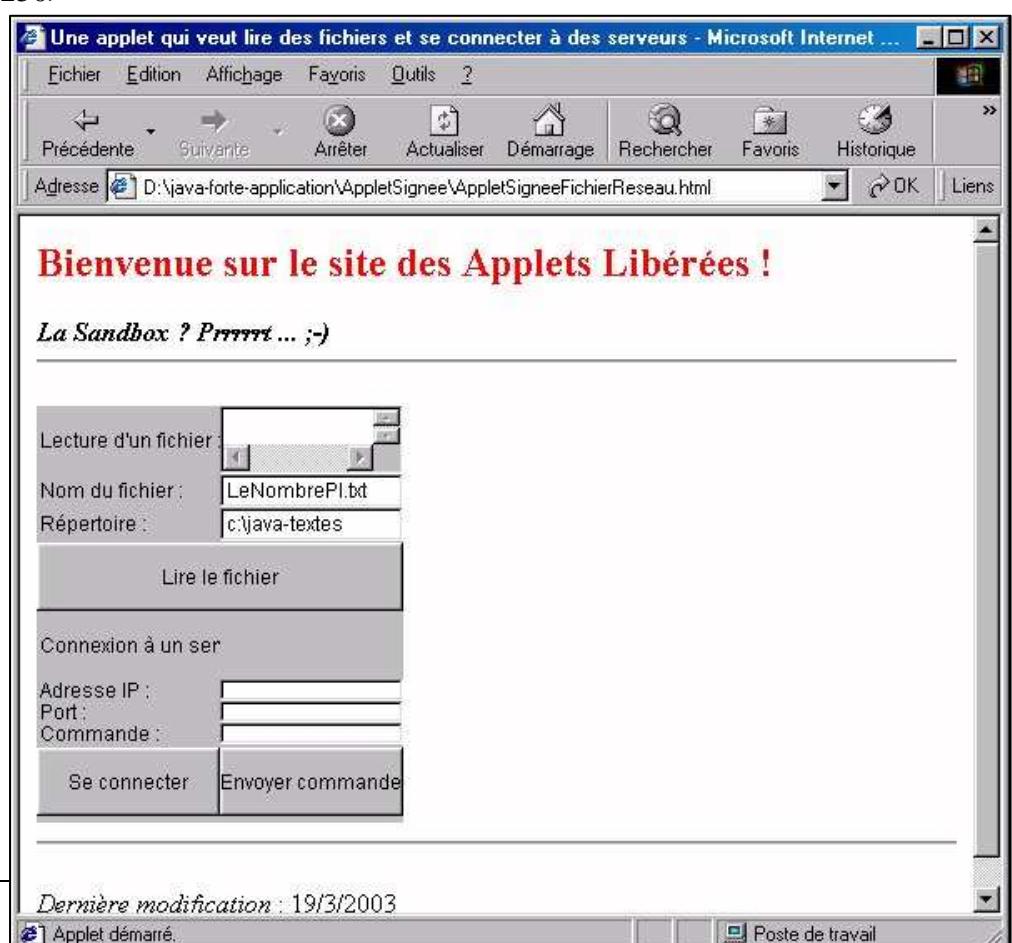
Ligne lue = Que j'aime à faire apprendre

#### **4.2 Une applet locale qui veut lire un fichier depuis un browser**

Nous allons à présent placer notre applet dans une page HTML construite par nos soins. Comme l'EDI (Netbeans ou Sun ONE Studio) a, en réalité, généré deux fichiers class (celui de l'applet AppletSignee proprement dite et celui d'une classe imbriquée AppletSignee\$1 qui implémente les listeners), nous allons les placer dans un fichier jar : appelons-le AppletSigneeFichierReseau.jar. C'est ce jar qui sera appelé dans la page AppletSigneeFichierReseau.html correspondante :

```
<applet code="AppletSignee.class"
        archive="d:\java-forte-application\AppletSignee\AppletSigneeFichierReseau.jar"
        width=220 height=250>
</applet>
```

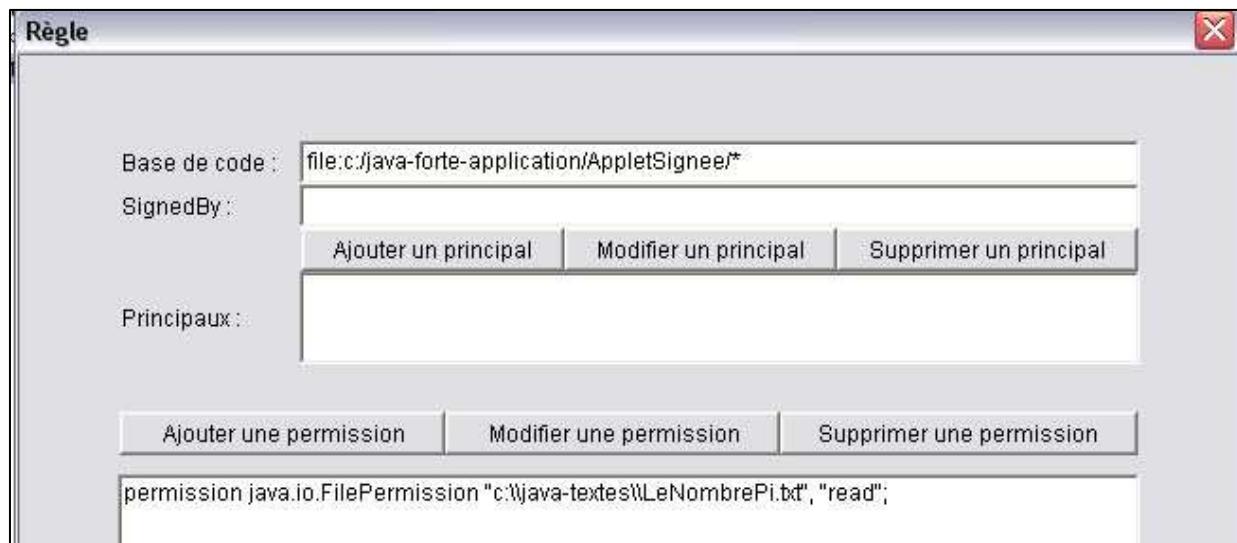
Nous allons consulter cette page HTML avec un browser, disons par exemple Internet Explorer (pour changer un peu ;-)) :



Sans précautions, c'est l'échec, comme on peut le voir sur la console du browser :



Donnons donc à notre applet les droits nécessaires (Internet Explorer utilise le fichier des politiques de sécurité du jdk utilisé) :



Nous avons donc restreint la permission aux seules applets d'un répertoire déterminé :

```
grant codeBase "file:d:/java-forte-application/AppletSignee/*" {
    permission java.io.FilePermission "c:\\java-textes\\LeNombrePI.txt", "read";
};
```

Un essai de consultation de la page conduit cette fois au succès ...

Reste à présent à nous placer dans une situation réelle, celle où l'applet et la page HTML sont sur un serveur WEB. Mais ceci n'a guère de sens sans un minimum de sécurité. En effet, si tout le monde peut permettre n'importe quoi, nous courons à l'anarchie. Il faut donc que l'octroi de permissions soit permis à des gens de toute confiance ...

## 5. Les permissions avec des signatures et des certificats

Nous savons que du code Java peut être signé au moyen d'une signature électronique : il suffit de le placer dans un fichier jar et d'utiliser l'outil jarsigner<sup>1</sup>. Comme la classe **CodeSource** permet en effet d'encapsuler le chemin à suivre pour trouver un code source (au moyen de son URL), il semble assez naturel de la doter des moyens de vérifier l'intégrité et l'authenticité de ce code au moyen des clés publiques correspondant aux clés privées qui ont servi à réaliser les signatures. Un objet CodeSource va donc aussi contenir les clés publiques nécessaires, en fait mémorisées au moyen de certificats. Comme on l'a déjà dit, le constructeur de cette classe se prototype donc bien naturellement selon :

```
public CodeSource(URL url, Certificate[] certs)
```

Ceci implique que la détermination des permissions accordées à un code passe en réalité par une vérification des clés. Ceci explique la présence de la clause signedBy dans la commande grant du fichier policy : la permission concédée peut n'être accordée que pour certains signataires :

```
grant [signedBy <signataire>] [codeBase <chemin>] { ... }
```

L'algorithme de détermination des permissions accordées à un code devient alors :

**si** le code est signé  
    **alors**  
        **si** l'une des clés publiques correspond au signataire d'un grant dans l'objet Policy ou si un grant sans signataire est trouvé  
            **alors**  
                **si** l'une des URL est celle du code  
                    alors utiliser les permissions correspondantes  
                **sinon**  
                    - aucune clé n'est reconnue et le code est considéré comme non signé  
                    - utiliser les permissions par défaut, typiquement celles de la sandbox  
    **sinon**  
        **si** l'une des URL est celle du code  
            **alors** utiliser les permissions correspondantes  
            **sinon** utiliser les permissions par défaut, typiquement celles de la sandbox

---

<sup>1</sup> voir le chapitre consacré à la cryptographie, plus particulièrement le paragraphe consacré aux applets signées, dans "Langage Java (II) : Programmation avancée des applications classiques"

## 6. L'ouverture authentifiée de la sécurité d'une applet

### 6.1 La signature de l'applet

Nous allons générer un certificat auto-certifié dans le keystore (version 1.3 ou 1.4) :

```
C:\jdk1.3.1_02\bin>keytool -genkey -alias Claude -keyalg DSA -keysize 1024 -dname
"CN=Claude Vilvens, O=HEP R Sualem, C=B"
Enter keystore password: beaugosse
Enter key password for <Claude>
(RETURN if same as keystore password): genius
```

On peut vérifier que tout est en ordre :

```
C:\jdk1.3.1_02\bin>keytool -list
Enter keystore password: beaugosse
Keystore type: jks
Keystore provider: SUN
Your keystore contains 1 entry:
claudie, Wed Mar 12 10:41:55 CET 2003, keyEntry,
Certificate fingerprint (MD5): A3:B8:68:DC:05:F7:7B:C5:06:03:1F:BD:F3:5B:06:5C
```

Pour pouvoir faire intervenir ce certificat dans nos problèmes d'applets autorisées, il nous faut l'exporter, c'est-à-dire le placer dans un fichier désigné :

```
C:\jdk1.3.1_02\bin>keytool -export -alias claudie -file james.cer
Enter keystore password: beaugosse
Certificate stored in file <james.cer>

C:\jdk1.3.1_02\bin>dir james.cer
...
JAMES CER      688 12/03/03 10:45 james.cer
    1 fichier(s)      688 octets

C:\jdk1.3.1_02\bin>keytool -printcert -file james.cer
Owner: CN=Claude Vilvens, O=HEP R Sualem, C=B
Issuer: CN=Claude Vilvens, O=HEP R Sualem, C=B
Serial number: 3e6f0099
Valid from: Wed Mar 12 10:40:41 CET 2003 until: Tue Jun 10 11:40:41 CEST 2003
Certificate fingerprints:
    MD5: A3:B8:68:DC:05:F7:7B:C5:06:03:1F:BD:F3:5B:06:5C
    SHA1: 65:13:96:02:F9:D0:5D:C8:8A:F7:1A:B1:7B:9B:08:09:8E:F5:93:13
```

Nous allons ensuite reprendre notre fichier jar contenant l'applet. On peut vérifier son contenu :

```
C:\jdk1.3.1_02\bin>jar -tf d:\java-forte-application\AppletSignee\AppletSigneeFichierReseau.jar
META-INF/MANIFEST.MF
AppletSignee.class
AppletSignee$1.class
```

---

Plaçons-nous dans le répertoire de l'applet et ajustons le PATH pour pouvoir utiliser l'outil jarsigner :

```
| D:\java-forte-application\AppletSignee>set path=c:\jdk1.3.1_02\bin;%path%
```

Signons le jar avec notre clé privée se trouvant dans le keystore :

```
| D:\java-forte-application\AppletSignee>jarsigner -keystore c:\vilvens\keystore  
AppletSigneeFichierReseau.jar claude  
Enter Passphrase for keystore: beaugosse  
Enter key password for claude: genius
```

Le jar a bien été modifié :

```
| D:\java-forte-application\AppletSignee>jar tf AppletSigneeFichierReseau.jar  
META-INF/MANIFEST.MF  
META-INF/CLAUDE.SF  
META-INF/CLAUDE.DSA  
AppletSignee.class  
AppletSignee$1.class
```

Dans le répertoire META-INF, on trouve à présent trois fichiers.

**1)** Le fichier **MF** (ManiFest) contient à présent la valeur des digests des fichiers class cités :

```
Name: AppletSignee$1.class  
SHA1-Digest: CxmtcCTIdVofQW5K3sgwBrB4kLk=
```

```
Name: AppletSignee.class  
SHA1-Digest: pjBOK6rXkaIK2sin1plgYGBSf7E=
```

**2)** Le fichier **SF** est le fichier de signature (Signature File – **SF**); il contient la valeur de hachage du manifeste ainsi que, pour chaque fichier, la valeur de hachage des lignes qui lui sont associées dans le manifeste :

```
Signature-Version: 1.0  
SHA1-Digest-Manifest: SWsC7DuD+/e8AEQr0rndJ1ZNPxU=  
Created-By: 1.3.1_02 (Sun Microsystems Inc.)
```

```
Name: AppletSignee$1.class  
SHA1-Digest: R5JDSBzvfnb6pHbW1XI7HxddDis=
```

```
Name: AppletSignee.class  
SHA1-Digest: Is/9qJzXwlJnpLiI9s0X4+RN5VA=
```

**3)** Le fichier **DSA** (Digital Signature Algorithm – **DSA**) contient la signature du fichier .SF et, sous forme codée, le certificat (ou la chaîne de certificats) du signataire.

## 6.2 La préparation de l'utilisation de l'applet signée

Il faut tout d'abord créer dans le Keystore une entrée de type TrustedCertEntry pour le certificat qui doit servir de sésame pour permettre l'utilisation de l'applet. Mais l'essai n'est pas possible en local :

```
D:\java-forte-application\AppletSignee>keytool -import -alias claude -file  
c:\jdk1.3.1_02\bin\james.cer  
Enter keystore password: beaugosse  
Enter key password for <claudie>: genius  
keytool error: java.lang.Exception: Certificate reply and certificate in keystore are identical
```

Force nous est donc de passer sur une autre machine. Un keystore y existe déjà :

```
C:\jdk1.3.1_02\bin>keytool -list  
Enter keystore password: beaugosse  
Keystore type: jks  
Keystore provider: SUN  
Your keystore contains 2 entries:  
claudie, Thu Oct 24 14:52:39 CEST 2002, keyEntry,  
Certificate fingerprint (MD5): 06:AE:C0:8A:7B:F1:B9:02:FA:A2:DB:30:14:31:7D:71  
denys, Thu Oct 24 15:02:21 CEST 2002, keyEntry,  
Certificate fingerprint (MD5): 37:F2:B0:1E:A7:DE:68:A0:E4:71:D3:80:07:DA:5B:3E
```

Pour ne pas retomber sur le même problème, supprimons l'entrée de l'alias claudie :

```
C:\jdk1.3.1_02\bin>keytool -delete -alias claudie  
Enter keystore password: beaugosse  
C:\jdk1.3.1_02\bin>keytool -list  
Enter keystore password: beaugosse  
Keystore type: jks  
Keystore provider: SUN  
Your keystore contains 1 entry:  
denys, Thu Oct 24 15:02:21 CEST 2002, keyEntry,  
Certificate fingerprint (MD5): 37:F2:B0:1E:A7:DE:68:A0:E4:71:D3:80:07:DA:5B:3E
```

Recopions le fichier certificat qui nous intéresse (donc, james.cer) dans le répertoire courant et importons-le dans le keystore local :

```
C:\jdk1.3.1_02\bin>keytool -import -alias claudie -file james.cer  
Enter keystore password: beaugosse  
Owner: CN=Claude Vilvens, O=HEP R Sualem, C=B  
Issuer: CN=Claude Vilvens, O=HEP R Sualem, C=B  
Serial number: 3e6f0099  
Valid from: Wed Mar 12 10:40:41 CET 2003 until: Tue Jun 10 11:40:41 CEST 2003  
Certificate fingerprints:  
MD5: A3:B8:68:DC:05:F7:7B:C5:06:03:1F:BD:F3:5B:06:5C  
SHA1: 65:13:96:02:F9:D0:5D:C8:8A:F7:1A:B1:7B:9B:08:09:8E:F5:93:13  
Trust this certificate? [no]: y
```

---

Certificate was added to keystore

C:\jdk1.3.1\_02\bin>**keytool -list**

Enter keystore password: *beaugosse*

Keystore type: jks

Keystore provider: SUN

Your keystore contains 2 entries:

claudie, Thu Mar 20 09:35:18 CET 2003, **trustedCertEntry**,

Certificate fingerprint (MD5): A3:B8:68:DC:05:F7:7B:C5:06:03:1F:BD:F3:5B:06:5C

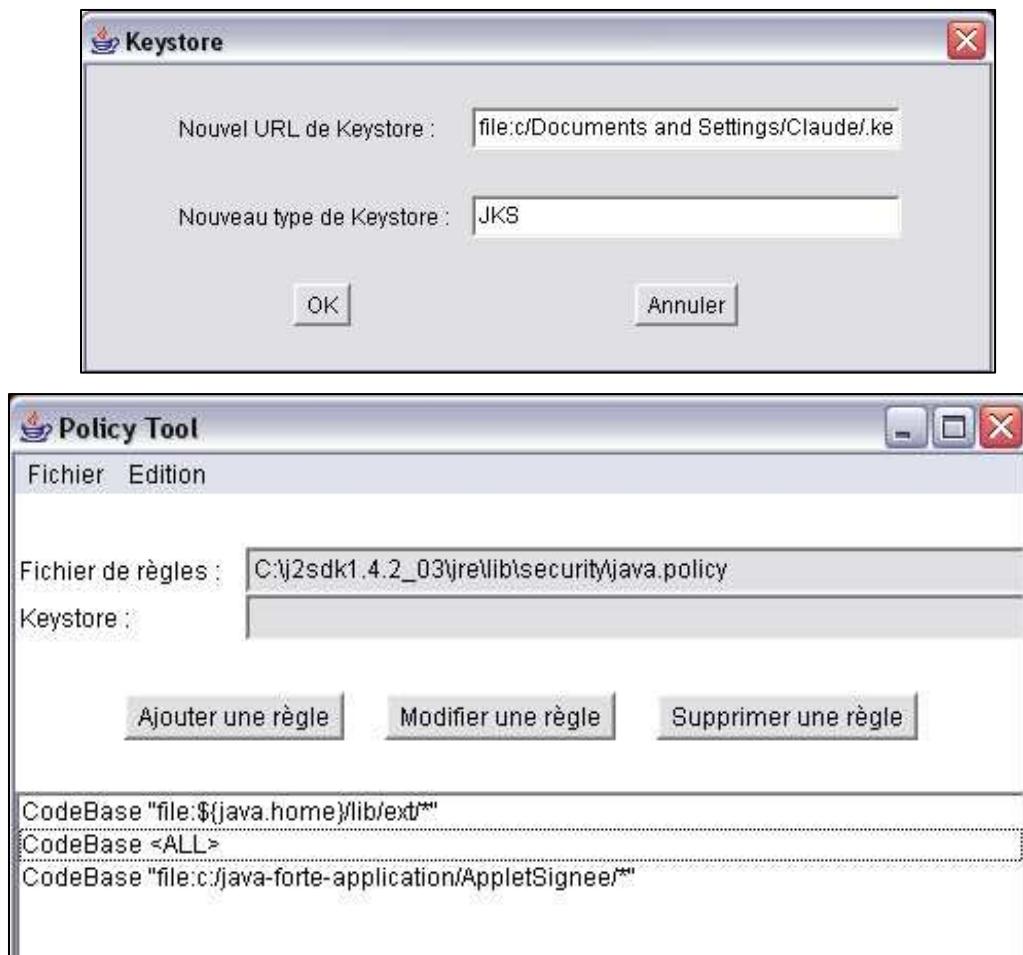
denys, Thu Oct 24 15:02:21 CEST 2002, **keyEntry**,

Certificate fingerprint (MD5): 37:F2:B0:1E:A7:DE:68:A0:E4:71:D3:80:07:DA:5B:3E

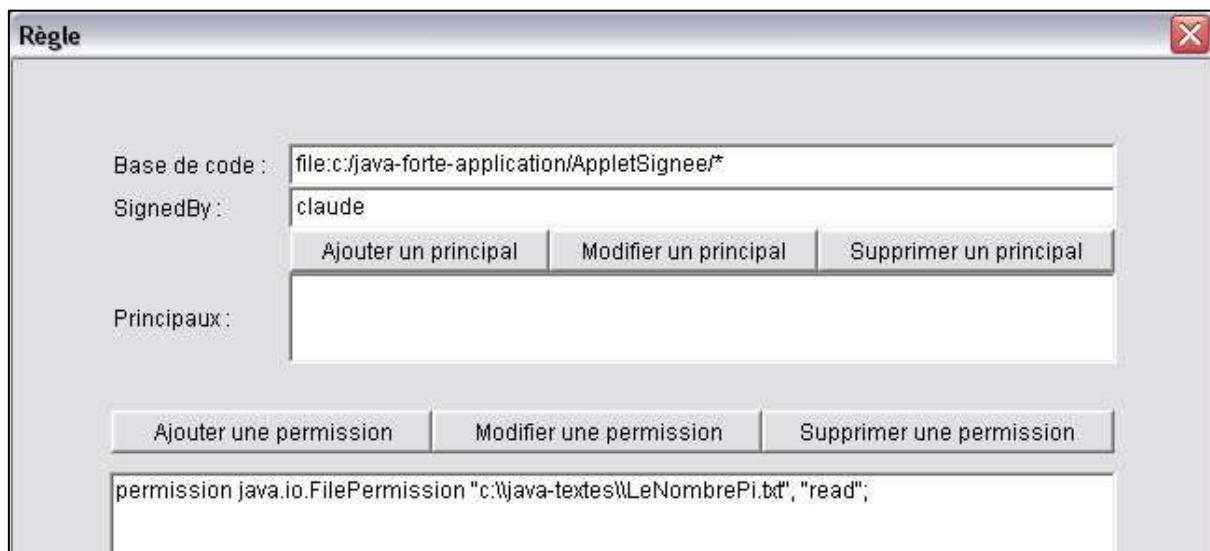
Nous sommes prêts à l'action ...

### **6.3 La permission avec une signature en local**

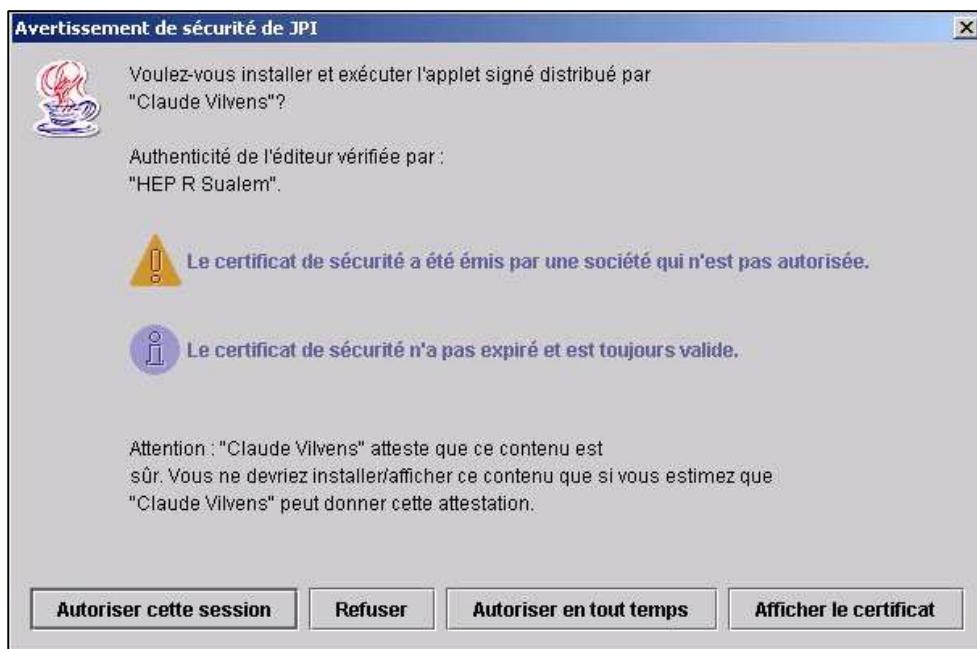
Dans l'outil policytool, nous sélectionnons d'abord dans le menu Edition→Changer de Keystore pour désigner le keystore à utiliser:



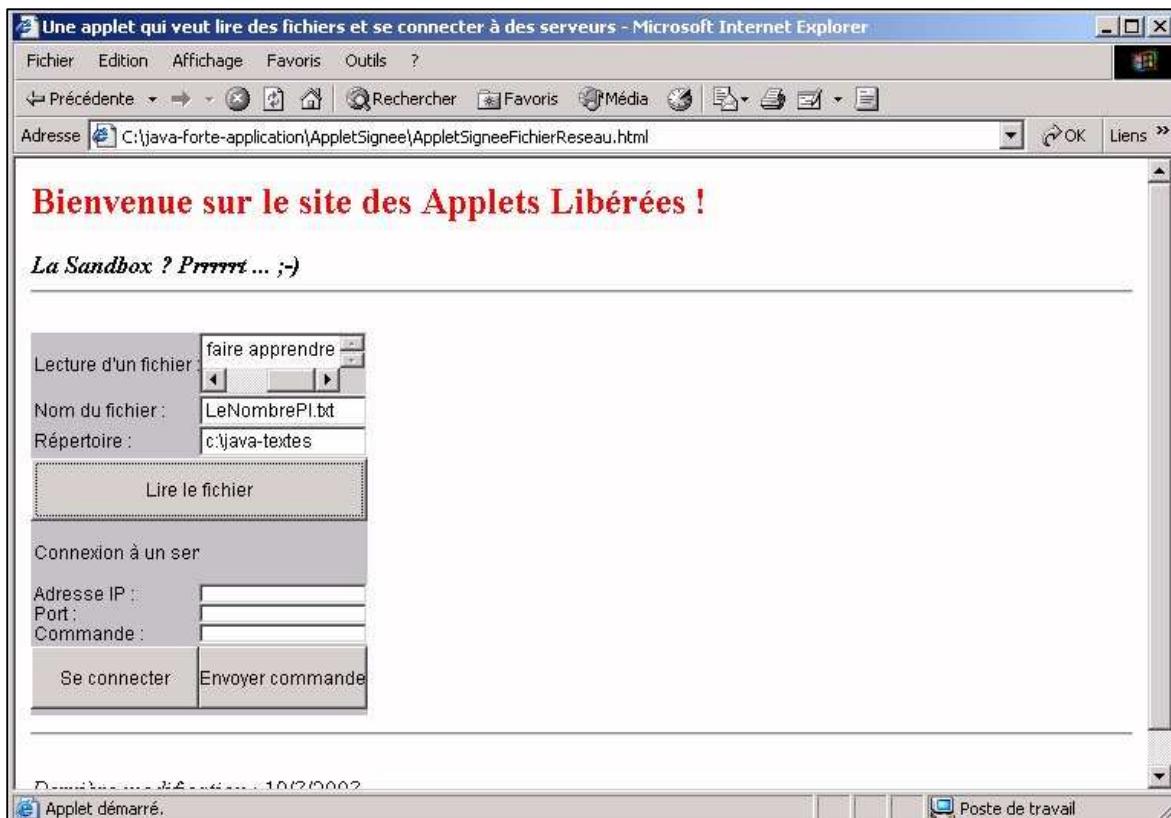
Ensuite, nous spécifions que la permission de lecture est signée par "claude" :



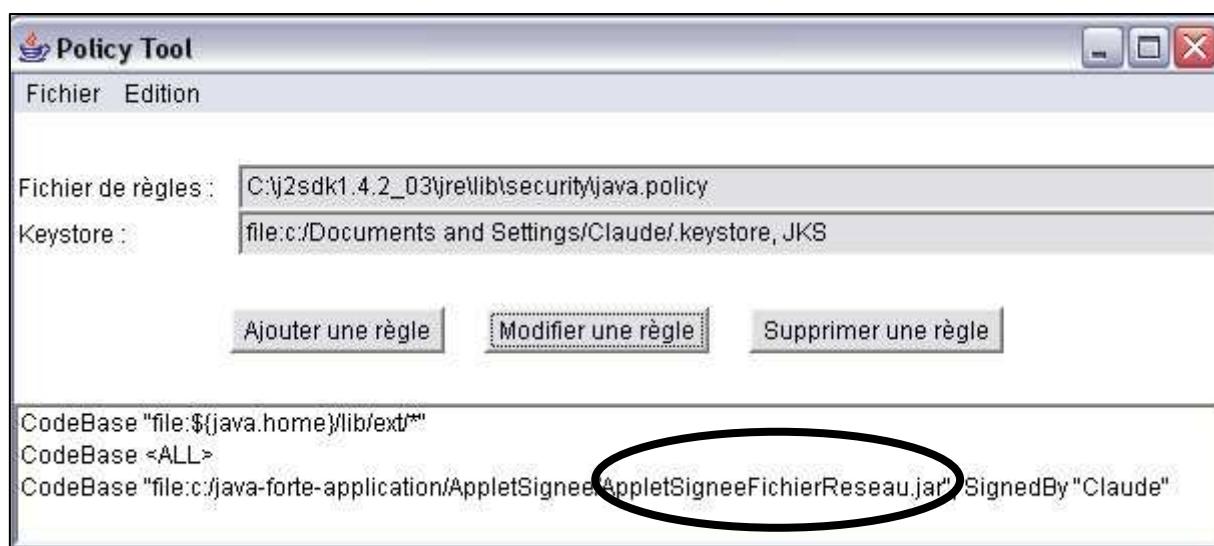
Au chargement de la page HTML, l'applet est chargée et démarrée – on obtient :



Nous choisissons "Refuser" afin de laisser les seules permissions prévues fonctionner. Le reste de l'histoire est connu :

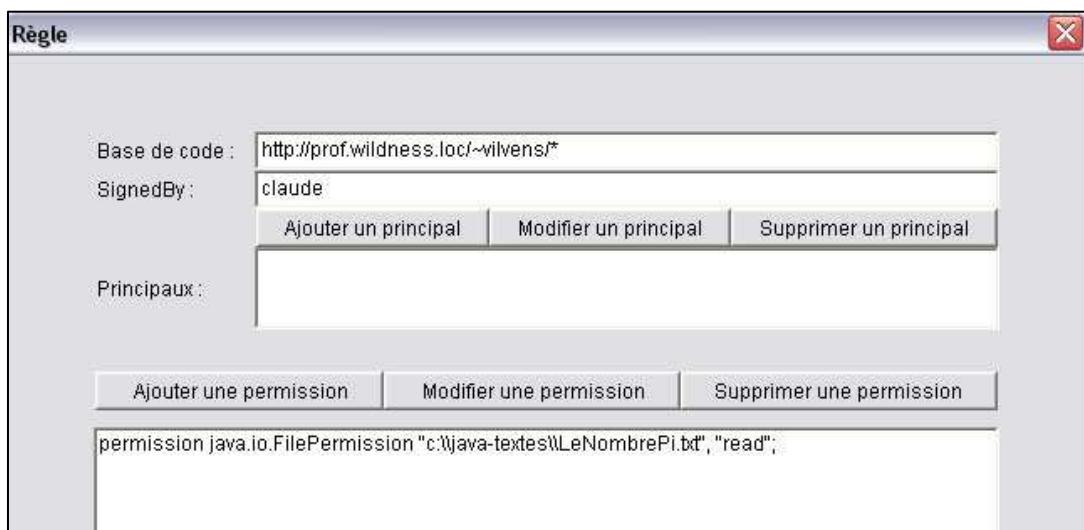


On peut même limiter la permission à la seule applet considérée jusqu'ici : il suffit de compléter le codebase du nom du fichier jar correspondant :

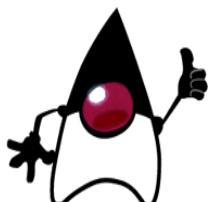


#### 6.4 La permission avec une signature à partir d'un serveur Web

Dans l'outil policytool, nous spécifions cette fois la permission de lecture du fichier par rapport à ce qui provient d'un site Web installé sur une machine Linux :



En ayant aménagé le titre de la page, cela donne, après le refus habituel d'ouverture au tout venant :



A ce stade, nous pouvons donc nous assurer qu'une applet a bien le droit d'effectuer certaines opérations ou pas. Ceci évidemment n'assure pas la confidentialité des échanges. Et, bien sûr, cette technique ne peut s'appliquer aux communications classiques par sockets. Va-t-il falloir tout construire soi-même ? Heureusement non : un protocole va nous aider ...

## XXV. Le protocole SSL, JSSE et le e-commerce sécurisé



*La programmation aujourd'hui, c'est une compétition entre des ingénieurs du logiciel, qui tentent de créer les programmes les plus performants à l'épreuve des idiots, et l'Univers, qui tente de produire des idiots encore plus idiots et encore plus nombreux. Jusqu'ici, c'est toujours l'Univers qui gagne ...*

(R.Cook)

### 1. Le schéma classique du commerce électronique

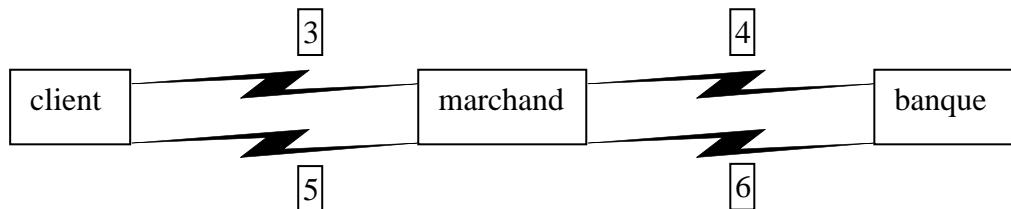
Le scénario à présent classique d'un achat par Internet est le suivant :

1. le client, au moyen de son browser, remplit son caddie virtuel;
2. une fois son caddie rempli, il "passe à la caisse" : pour ce faire, il choisit son mode de paiement, l'institution financière utilisée et ses instructions spécifiques dans un formulaire envoyé par le marchand;
3. le client envoie le formulaire;
4. le marchand demande l'autorisation de paiement auprès de l'institution financière désignée par le client;
5. si il l'obtient, le marchand confirme la transaction au client et envoie effectivement les articles au client;
6. le marchand demande le paiement effectif à l'institution financière du client.

Ce sont essentiellement les communications réseaux qui vont faire l'objet de notre attention. Evidemment, les risques sont clairs :

- ◆ comment le client peut-il être sûr qu'il communique avec le vrai marchand ?
- ◆ le marchand ne risque-t-il pas d'utiliser les informations de paiement dans des buts différents ?
- ◆ les informations de paiement ne risquent-elles pas d'être captées par un hacker quelconque ?

Schématiquement, les étapes névralgiques sont :



Nous avons donc ici un besoin criant d'authentification, d'intégrité et de confidentialité ...

## 2. Les protocoles SSL et HTTPS

### 2.1 Les objectifs

Le protocole **SSL** (Secure Socket Layer) est un protocole qui répond aux besoins décrits ci-dessus puisqu'il vise à trois fonctionnalités principales (déjà évoquées dans un volume précédent) :

- ◆ l'authentification des serveurs : un utilisateur doit pouvoir avoir l'assurance qu'il s'adresse bien au serveur visé (ceci prend évidemment tout son sens lorsque, par exemple, il s'agit d'envoyer à un serveur son numéro de carte de crédit);
- ◆ l'authentification des clients : réciproquement, un serveur doit pouvoir s'assurer qu'un client est bien celui qu'il prétend être;
- ◆ le cryptage et le décryptage des données transmises entre un client et le serveur qu'il accède; encore une fois, l'exemple du numéro de carte de crédit est évocateur.

Pour atteindre ces objectifs, SSL utilisera logiquement :

- ◆ les certificats et donc les signatures digitales (pour les authentifications);
- ◆ les digests et les MACs (pour le contrôle de l'intégrité);
- ◆ les chiffrements symétriques (pour la confidentialité - les cryptages et décryptages ne doivent pas réclamer un temps prohibitif).

### 2.2 Un protocole entre deux couches

Ce qui rend ce protocole intéressant est que SSL se situe entre ce protocole applicatif et la couche transport (soit TCP); *il est donc indépendant des applications qui l'utilisent*. On désigne par le nom d'un protocole applicatif suffixé d'un "s" la version de ce protocole qui utilise SSL. Par exemple, **HTTPS** désigne une "version sécurisée" de HTTP. Donc, en pratique, il suffit de remplacer "http" par "https" dans les URLs concernés. On rencontre de même **FTPS**, **SMTS**, **WTLS** (WAP Security = Wireless Transport Layer Security), etc. Des ports TCP réservés sont attribués à ces protocoles sécurisés :

protocole	port
https	443
smtsp	465
ftps	990
ftps-data	989
telnets	992
pop3s	995
imaps	993

### 2.3 Un bref historique

SSL 1.0 a été conçu par Netscape en 1994, en collaboration avec Mastercard, Bank of America, MCI et Silicon Graphics. Sa deuxième version fut intégrée au célèbre browser Navigator de Netscape. La version 3.0 apparut en 1995, permettant notamment l'authentification du client. L'IETF racheta le brevet à Netscape en 1999 et s'en inspira pour créer un protocole connu sous le nouveau nom de **TLS** (Transport Layer Security Protocol) – version 1. Mais les habitudes sont tenaces et tout le monde parle encore, à l'heure actuelle, de SSL v3.1 ...

---

SSL a donc été au départ développé par Netscape. Mais, à l'heure actuelle, la plupart des navigateurs sont capables de l'utiliser, à commencer par Internet Explorer. Java, de son côté, fournit une extension prenant SSL en compte : elle se nomme **JSSE** et comporte des classes comme `SecureSocket`. A vrai dire, de telles classes ne sont nécessaires que si l'on désire travailler en dessous de la couche application. En effet, au niveau de celle-ci, il suffit d'utiliser "https" dans les constructeurs d'URL !

### **3. Le dialogue SSL de création d'une session**

#### **3.1 Le principe général**

On peut d'emblée remarquer que l'on peut établir des connexions de type différent (c'est-à-dire utilisant des protocoles différents) au sein d'une même session. Une **session SSL** va donc établir la valeur d'un certain nombre de paramètres, valeurs utilisables par diverses **connexions SSL** au sein de la même session (par exemple, une connexion HTTPS et une connexion FTPS).

*Le dialogue de début de session SSL* en fait une combinaison de chiffrement asymétrique et symétrique qui sous-tend le mécanisme :

- ◆ le client contacte le serveur Web sécurisé par une connexion SSL (https, smtps, etc);
- ◆ le client et le serveur s'accordent sur les algorithmes qui seront utilisés (phase de négociation);
- ◆ le serveur envoie au client un **certificat** contenant sa clé **publique**;
- ◆ le client peut vérifier l'authenticité de ce certificat au moyen des certificats qu'il possède (*root certificats*);
- ◆ [optionnel] le client envoie au serveur un **certificat** contenant sa clé **publique**; le serveur peut vérifier l'authenticité de ce certificat au moyen de ses certificats (*server certificats*);
- ◆ [optionnel - suite] le serveur envoie au client un **challenge** (c'est-à-dire une série de bits) que le client doit signer au moyen de sa clé privée et renvoyer au serveur pour s'authentifier;
- ◆ en cas de succès, le client construit un message (*pre-master secret*) qui est ensuite **crypté** au moyen de la clé **publique** du serveur et lui envoie ce message; il génère une **clé de session** (*master secret*) à partir du message;
- ◆ le serveur déchiffre le message au moyen de sa clé **privée**; à partir de ce message, il génère, de la même manière, une **clé de session** (*master secret*) qui est forcément la même que celle générée sur le client;
- ◆ le client et le serveur communiquent par **chiffrement symétrique** en utilisant la clé de session.

Le point intéressant est que **la clé de session**, négociée par les deux intervenants, **n'a de sens qu'au sein de la session ainsi créée** : un hacker équipé du même certificat pourrait certes obtenir la clé publique, mais il se verrait attribuer une autre clé de session, ne lui permettant pas d'intervenir dans une autre communication.

### 3.2 Les sous-protocoles de SSL

SSL est composé de 2 sous-couches :

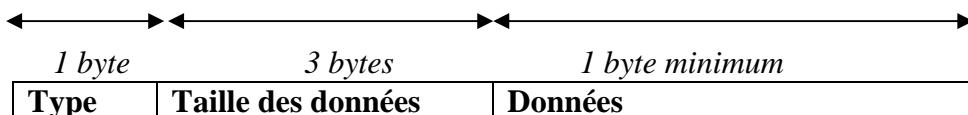
- ◆ une couche qui gère l'établissement de la connexion sécurisée et la négociation des paramètres de la session; on trouve à ce niveau le sous-protocole "**SSL Handshake**", qui est susceptible de lancer
  - \* le sous-protocole "**SSL Change Cipher SPEC**" qui permet notamment au serveur et au client de s'accorder sur les algorithmes de chiffrements asymétrique (en pratique, par exemple, RSA) et symétrique (par exemple, DES) ainsi que sur l'algorithme d'authentification (par exemple, SHA-1 ou MD5);
  - \* le sous protocole "**SSL Alert**" gère les messages d'erreur entre le client et le serveur;
- ◆ une couche, dite "**SSL Record Protocol**", qui définit la façon dont seront traitées les données à chiffrer; en fait, c'est à ce niveau que sont réalisés :
  - \* le chiffrement au moyen de la clé secrète négociée dans le "Handshake protocol", chiffrement précédé de la compression du message [*confidentialité*];
  - \* la création d'un MAC utilisant la clé en question [*intégrité*];
  - en réalité, le message est au préalable fragmenté et ce traitement est appliqué à chaque bloc.

### 3.3 Le sous-protocole "SSL Handshake"

Pierre d'angle de l'établissement de la session, le SSL handshake permet au serveur et au client de :

- ◆ s'authentifier;
- ◆ négocier les algorithmes de chiffrement et de MAC;
- ◆ négocier les clés cryptographiques à employer.

Pour réaliser ces opérations, le client et le serveur échangent des messages ayant toujours un format à 3 champs :



Le sous-protocole SSL Handshake comporte en fait 4 phases d'échanges de messages :

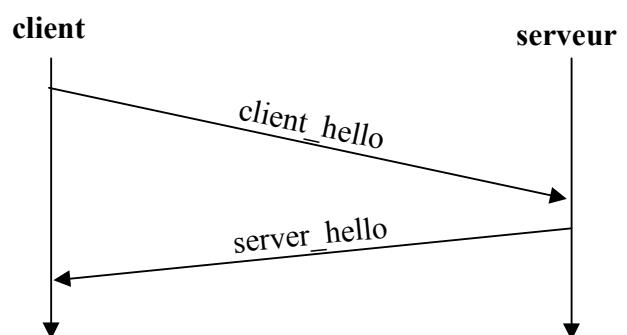
1. établissement des possibilités en matière de sécurité; cette phase détermine entre autres la version, la session, les algorithmes de chiffrement et compression, des nombres aléatoires;
2. authentication du serveur et échange des clés;
3. authentication du client et échange des clés;
4. clôture.

Examinons ces phases de plus près.

#### 1) La détermination des paramètres de sécurité

1.1) Le client envoie un premier message du type *client\_hello*. Ce message comporte :

- ◆ *Version* : la plus haute que le client sait utiliser;
- ◆ *Random* : une représentation de la date-heure sur 4 octets et un nombre aléatoire de 28 bytes;



- ◆ *Session ID* : la valeur 0 signifie que l'on veut créer une nouvelle connexion dans une nouvelle session; une valeur non nulle signifie que l'on veut changer les paramètres existant ou créer une nouvelle connexion dans la session existante;
- ◆ *Cipher Suite* : il s'agit d'une liste des algorithmes d'échanges (RSA, Diffie-Hellman), de chiffrements (DES, RC4, RC2 + par blocs CBC ou par flux) et digests-MACs (MD5, SHA-1) classés par ordre décroissant de préférence, ainsi que de quelques renseignements complémentaires (comme le vecteur IV pour CBC);
- ◆ *Compression Method* : la liste des algorithmes de compression supportés par le client.

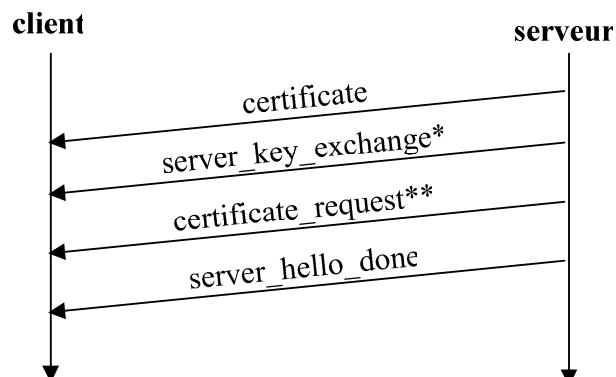
1.2) Le serveur choisit la stratégie optimale en tenant compte des renseignements reçus. Il envoie ensuite au client un message ***server\_hello*** contenant ses choix et un nombre aléatoire (qui servira de numéro de version).

Le client, une fois la réponse reçue du serveur, peut alors mémoriser les paramètres de sécurité ainsi définis dans une structure.

## 2) L'authentification du serveur et l'obtention d'une clé de cryptage

2.1) Le serveur commence par envoyer son certificat dans un message ***certificate***. Celui-ci est à l'origine le plus souvent un certificat X509 niveau 3, mais il sera envoyé au format ASN.1 (Abstract Syntax Notation version 1), assurant ainsi la plus grande compatibilité possible.

Dans le cas d'un certificat classique donnant une clé publique, le client est en état d'authentifier le serveur puis d'envoyer un message "premaster" (destiné à la génération de la future clé de session) en le cryptant avec cette clé publique.



2.2) Par contre,

- ◆ si le serveur n'a pas de certificat,
- ◆ si le certificat envoyé par le serveur n'est qu'un certificat de signature,
- ◆ [pour mémoire : si l'algorithme d'échange de clés retenu lors de la phase 1 est Fortezza<sup>1</sup> (il s'agit d'un ensemble de produits de cryptographie patronnés par le NSA - National Security Agency - américain – ces cryptages sont utilisés pour les e-mails, les cartes à puces, les téléphones cellulaires) – délaissé depuis quelques années],

le serveur doit alors fournir d'autres informations pour que le client soit en mesure de crypter son message premaster. Le serveur envoie donc dans ce cas un message ***server\_key\_exchange*** qui contient

<sup>1</sup> TLS a abandonné Fortezza

- ◆ soit une clé publique d'une paire de clés RSA générée par le serveur; cette clé publique est temporaire : elle n'est valide que durant la session en cours; le message est signé par le serveur selon un algorithme RSA.
- ◆ soit les paramètres d'un algorithme de Diffie –Hellman<sup>1</sup> (donc essentiellement son nombre aléatoire généré X); à nouveau, le message peut être signé par le serveur selon un algorithme RSA.
- ◆ [soit un nombre aléatoire utilisé par Fortezza]..

2.3) Il se peut que le serveur demande au client un certificat – il le fait alors en envoyant un message *certificate\_request*, qui comporte le type d'algorithme utilisé et la liste des CA valides. La réponse viendra en phase 3.

2.4) Le serveur envoie enfin un message *server\_hello\_done* pour signifier la fin de la phase 2 : d'une manière ou d'une autre, l'échange du premaster est possible.

### **3) L'authentification du client et la génération du "master secret"**

3.1) Si le serveur a effectivement demandé un certificat au client, celui-ci commence par envoyer son certificat dans un message *certificate* si il en possède un, un message *no\_certificate* s'il n'en possède pas.

3.2) Le client envoie ensuite un message *client\_key\_exchange* dont la nature dépend de l'algorithme choisi en phase 2 :



- ◆ dans le cas de RSA (basé sur le certificat du serveur ou sur un RSA temporaire), le client génère une séquence de 48 octets, le "*premaster secret*", la crypte avec la clé publique du serveur et la lui envoie;
- ◆ dans le cas de l'algorithme de Diffie-Hellman, le client envoie son nombre aléatoire Y (si celui-ci n'est pas inclus dans le certificat – dans ce cas, le message envoyé sera vide); la clé de session ainsi construite des deux côtés fait office de "premaster secret";
- ◆ [dans le cas de Fortezza, ce sont les paramètres de cet algorithme qui sont envoyé].

Le client et le serveur sont donc, dans tous les cas, en possession du message "premaster secret". Ils génèrent alors un "*master secret*" de 48 bytes en utilisant le "premaster secret" et les nombres aléatoires fournis dans *client\_hello* et *server\_hello*. Cette séquence de bytes, secrète mais partagée par le client et le serveur, leur sert de base pour le calcul des paramètres caractéristiques de la Cipher Suite retenue en phase 1. Ils se retrouvent ainsi tous les deux avec les mêmes éléments cryptographiques, et notamment une **clé de session**.

---

<sup>1</sup> L'algorithme de Diffie-Hellman permet à deux parties A et B de générer la même clé symétrique en échangeant que deux grands entiers premiers entre eux (disons n et g); A (B) choisit un grand entier aléatoire x (y), calcule  $X=g^x \% n$  ( $Y=g^y \% n$ ) et l'envoie à B (A); - A (B) calcule sa clé selon  $Ka = Y^x \% n$  ( $Kb=X^y \% n$ ). Les deux clés sont bien les mêmes.

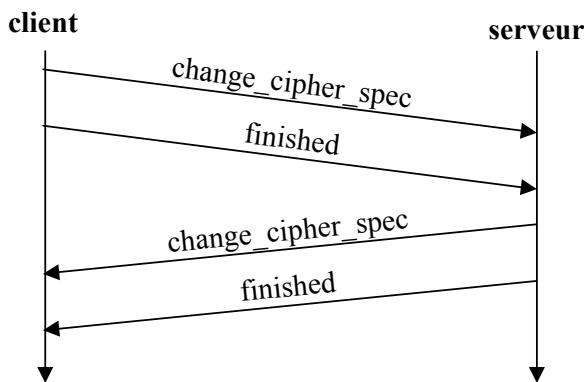
3.3) Si le client a envoyé un certificat, on testera encore qu'il est bien en possession de la clé privée associée à la clé publique faisant l'objet du certificat. Pour ce faire, le client va envoyer un message *certificate\_verify* qui comporte une espèce de digest calculé sur les messages du handshake et sur le "master secret" (donc un "challenge"), le tout signé avec la clé privée du client. Le serveur a manifestement toutes les armes pour vérifier la signature ...

#### 4) L'enregistrement et la validation des spécifications négociées

4.1) Le client envoie un message *change\_cipher\_spec*, dont le but est d'enregistrer la stratégie de sécurité temporaire en stratégie courante. Un tel message est en fait le seul message du sous-protocole "**SSL Change Cipher SPEC**".

4.2) Le client envoie un message *finished* qui est une sorte de digest utilisant l'algorithme négocié (donc MD5 ou SHA) et basé sur les messages du handshake, un identificateur de l'expéditeur et le "master secret". Il s'agit donc d'une espèce de dernière validation du partage d'une clé de session conforme aux étapes précédentes.

4.3 et 4.4) Le serveur réalise les mêmes opérations.



#### 3.4 Les variables d'état SSL

A la lumière de ce qui vient d'être décrit, on peut comprendre qu'un certain nombre de **variables d'état** caractérisent les sessions et les connexions SSL. Ces variables sont :

session SSL	
Session identifier	numéro de session fixé par le serveur (1 byte)
Peer certificate(s)	certificat du serveur + certificat éventuel du client
Cipher spec	algorithme de chiffrement symétrique, de digest, CBC (avec IV) ou stream
Master secret	séquence de bits de 48 bytes, base de la clé de session utilisée pour le chiffrement et de la clé secrète utilisée pour les MACs
Compression method	méthode de compression
Is resumable	pour indiquer si de nouvelles connexions peuvent être créées sur base de cette session

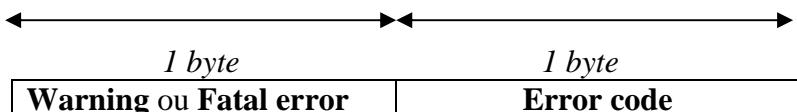
et

connexion SSL	
server et client random	les nombres aléatoires échangés lors du handshake
server et client write	la clé secrète utilisée par le serveur et le client pour générer les

MAC secret	MACs
server et client write key	la clé symétrique utilisée par le serveur et le client pour chiffrer leurs messages
initialization vector	pour les chiffrements CBC
sequence number	numéro de séquence

### 3.5 Le sous-protocole "SSL Alert"

Le rôle de ce protocole est simplement de régitir les messages d'erreur envoyés par le serveur vers le client ou vice versa. Le format de ces messages est des plus simples :



VALEUR	NOM	ERREUR FATALE	COMMENTAIRE
0	Close_Notify	Non	L'émetteur indique qu'il va terminer la connexion
10	Unexpect_Message	Oui	L'émetteur indique qu'il a reçu un message qu'il ne peut interpréter
20	Bad_Record_Mac	Oui	L'émetteur indique que le message reçu semble être altéré (la somme de contrôle n'est pas la bonne)
30	Decompression_Failure	Oui	L'émetteur indique qu'il a reçu des données qu'il ne peut décompresser
40	HandShake_Failure	Oui	L'émetteur indique qu'il n'a pas été capable de négocier un jeu de sécurité qui lui convient pour cette session
41	No_Certificate	Non	Le client indique qu'il ne dispose d'aucun certificat pouvant répondre aux attentes du serveur
42	Bad_Certificate	Non	L'émetteur indique qu'il a reçu un certificat corrompu
43	Unsupported_Certificate	Non	L'émetteur indique qu'il ne peut gérer le type de certificat reçu
44	Certificate_Revoked	Non	L'émetteur indique que le certificat reçu a été révoqué par l'organisme de certification
45	Certificate_Expired	Non	L'émetteur indique que le certificat a expiré.
46	Certificate_Unknown	Non	L'émetteur indique qu'il y a eu un problème avec le certificat reçu
47	Illegal_Parameter	Non	L'émetteur indique qu'il a reçu un handshake contenant des valeurs de paramètre non conformes

En cas d'erreur fatale, la connexion est fermée après l'envoi du message. Les autres connexions de la session concernée ne sont pas coupées, mais il n'est plus possible d'en créer de nouvelles. TLS a introduit de nouveaux codes d'erreur (par exemple, 48 : Unknown\_CA).

### **3.6 Le sous-protocole "SSL Record"**

Une fois la politique de sécurité clairement définie et les éléments techniques mis en place grâce au SSL Handshake, le client et le serveur peuvent échanger des messages et informations de manière sécurisée. C'est le rôle du protocole SSL Record de définir les règles de transmissions assurant

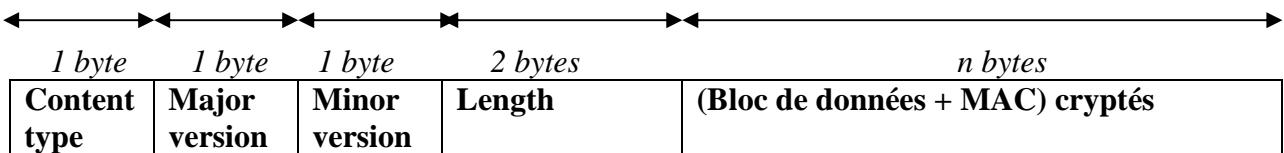
- ◆ la confidentialité : le cryptage symétrique proprement dit, pratiqué au moyen de la clé de session;
- ◆ l'intégrité : l'utilisation de MACs construits au moyen de l'algorithme de digest retenu et des "server write MAC secret" et "client write MAC secret".

Comme on voit, il reçoit donc les informations nécessaires de la couche applicative (les véritables données) et de la couche Handshake-ChangeCipherSpec (les clés, les algorithmes, ...)

Concrètement,

- ◆ les données à transmettre sont découpées en bloc de taille inférieure à 16 Ko;
- ◆ chaque bloc est ensuite éventuellement compressé – si c'est le cas, ce doit être sans pertes et ne doit pas augmenter la taille du contenu de plus de 1024 bytes;
- ◆ on calcule un MAC pour chaque bloc (une formule mêlant hashages et clés "write MAC secret") – TLS utilise plutôt un HMAC, plus difficile à pirater;
- ◆ chaque bloc, complété de son MAC, est crypté selon le cryptage symétrique négocié précédemment;
- ◆ un header SSL est ajouté devant chaque bloc; ce header contient
  - un Content type (1 byte) : le protocole utilisé pour manipuler le fragment (soit change\_cipher\_spec, alert ou handshake, soit un protocole applicatif comme http ou ftp);
  - une Major version (1 byte) : version ssl utilisée : souvent 3;
  - une Minor version (1 byte) : si sslv3 : valeur = 0;
  - une Compressed length (2 bytes) : longueur du fragment de texte clair.

Schématiquement, ce qui est transmis à la couche de transport est donc :



### 3.7 Les faiblesses de SSL

A priori, le protocole décrit ci-dessus est imparable. En fait, indépendamment des limites techniques qui existent malgré tout, il reste que ce protocole est sûr si et seulement si les protagonistes jouent les règles du jeu. Ainsi, dans le contexte du e-commerce :

- ◆ l'authentification du client est optionnelle, si bien que celui-ci peut utiliser, par exemple, un numéro de carte de crédit dérobée;
- ◆ le serveur peut très bien utiliser les informations fournies par le client (par exemple, encore une fois, son numéro de carte de crédit) à des fins malhonnêtes;
- ◆ le client a souvent tendance à ignorer les messages d'avertissement concernant un certificat invalide (autorité de certification non reconnue, validité dépassée).

De plus, si un client s'authentifie à l'aide d'un certificat, il enregistrera probablement ses paramètres de reconnaissance sur son ordinateur : la cible rêvée pour un cheval de Troie ...

Néanmoins, SSL est très largement répandu dans les transactions WEB – il mérite certainement que nous fassions l'effort de l'utiliser dans notre programmation ...



(© Pour la Science N.269 mars 2000)

En fait, les fraudes constatées dans le domaine de l'e-commerce trouvent souvent leur explication dans le fait que les serveurs auxquels les commandes sont envoyées reçoivent et mémorisent les informations de paiement de leurs clients (comme le numéro de carte de crédit et sa date de validité). En effet, ils deviennent ainsi la cible privilégiée des hackers qui tenteront d'accéder à ces données.

### 3.8 Le principe de la signature duale

Le remède semble donc assez clair : il faut séparer

- ◆ la commande (le contenu du caddie virtuel – **OI** : *Order Information*) qui est toujours envoyée au serveur marchand;
- ◆ les informations de paiement (**PI** : *Payment Information*) qui sont envoyées à un serveur de l'institution financière concernée.

**La transaction ne pourra alors s'effectuer que si**

- ◆ le marchand accepte la vente;
- ◆ la banque autorise le paiement.

Cependant, dans un contexte d'authentification, cette manière de faire pose problème : il faut en effet mélanger des informations qui ne sont pas destinées à un seul récepteur. Le problème est contourné au moyen du concept de **signature duale**. Les choses se passent de la façon suivante. Le client :

- ◆ calcule deux digests (fonction de hashage – par exemple, SHA-1), l'un sur l'OI, l'autre sur PI; désignons ces digests par OD et PD;
- ◆ concatène OD et PD et crypte le résultat au moyen de sa clé privée (type RSA) : le résultat constitue ce que l'on appelle la **signature duale** (DS);
- ◆ envoie alors
  - au marchand : DS, OI, PD;
  - à la banque : DS, PI, OD.

Le marchand va pouvoir vérifier l'intégrité de ce qu'il reçoit en :

- ◆ calculant un digest local de OI (LOD);
- ◆ concaténant LOD et PD;
- ◆ décryptant DS au moyen de la clé publique du client (on suppose que le marchand dispose d'un certificat du client)
- ◆ comparant ce qu'il obtient ainsi à LOD+PD.

Bien que ne connaissant pas le PI, il a ainsi l'assurance que ce qu'il a reçu est bien corrélation avec ces informations de paiement.

La banque procèdera évidemment de manière symétrique en :

- ◆ calculant un digest local de PI (LPD);
- ◆ concaténant OD et LPD;
- ◆ décryptant DS au moyen de la clé publique du client (on suppose que la banque dispose d'un certificat du client)
- ◆ comparant ce qu'il obtient ainsi à OD+LPD.

Evidemment, ceci sous-entend malgré tout que le marchand dialogue, pour échanger les accords sur les autorisations de commande et de paiement, avec la banque ... qu'il ne connaît pas forcément (et réciproquement) ! Le modèle proposé n'est donc envisageable que si il existe un intermédiaire réalisant le lien entre les deux protagonistes : il s'agit d'une passerelle, d'un *gateway*. Le protocole **SET** (Secure Electronic Transaction), développé par Visa et Master Card en collaboration avec Microsoft, IBM, HP, etc, travaille selon ce modèle – mais il est peu répandu ... et est à présent remplacé par le protocole 3D-Secure !

Revenons donc à notre programmation utilisant SSL ...

## 4. Les packages Java pour SSL

### 4.1 Les classes sockets pour SSL

**JSSE** (Java Secure Socket Extension) est une extension du J2SE 1.4 qui apporte le support de SSL et TLS en terme de fonctionnalités d'authentification (avec des certificats X.509), cryptage et intégrité. Son package **javax.net.ssl** (intégré au JDK 1.4, à ajouter pour les JDK antérieurs) contient le cadre de référence de l'utilisation de SSL dans le cadre de la programmation Java. Les classes abstraites **SSLocket**, dérivée de la classe **Socket** classique, et **SSLServerSocket**, dérivée évidemment de la classe **ServerSocket** classique, encapsulent

---

donc la manière dont les mécanismes SSL doivent être utilisés. On peut retrouver tous les éléments du protocole SSL en observant les méthodes existantes, comme par exemple :

```
public abstract String[] getEnabledCipherSuites()
```

ou

```
public abstract void setNeedClientAuth(boolean flag)
```

Puisqu'il s'agit de classes abstraites, on aura compris que l'on utilisera des factories pour obtenir les implémentations réelles de ces classes, implémentations fournies par des providers. Au sein de javax.net.ssl, les classes factory en question se nomment, sans surprise, **SSLServerSocketFactory** et **SSLSocketFactory**, dérivées respectivement des classes **ServerSocketFactory** et **SocketFactory** du package javax.net, qui ne contient d'ailleurs que ces deux classes. Le rôle de ces dernières est de fournir

un mécanisme général (c'est-à-dire pas forcément SSL) de création dynamique de sockets paramétrées selon les besoins et la plate-forme utilisée.

Les méthodes statiques de ces factories sont respectivement :

```
public abstract ServerSocket createServerSocket (int port, int backlog,  
InetAddress ifAddress) throws IOException
```

(le 2<sup>ème</sup> paramètre représente le nombre maximum de connexions pendantes) et

```
public abstract Socket createSocket (String host, int port, InetAddress clientHost,  
int clientPort) throws IOException, UnknownHostException
```

mais elles sont disponibles en versions polymorphes se contentant de moins de paramètres.

#### **4.2 Obtenir les factories dans un contexte SSL**

Les deux factories devraient, logiquement, pouvoir se paramétrer en fonction des éléments SSL disponibles sur la machine utilisée : keystore utilisé, mot de passe de celui-ci, certificat d'une entité considérée comme sûre, etc. Il en est bien ainsi, mais indirectement : c'est le rôle de la classe **SSLContext**, directement dérivée d'Object et se trouvant dans le package **com.sun.net.ssl**, de rassembler ces paramètres<sup>1</sup>. Une fois un tel contexte créé, on pourra créer les factories de sockets nécessaires au moyen des méthodes du contexte :

```
public final SSLSocketFactory getSocketFactory()  
public final SSLServerSocketFactory getServerSocketFactory()
```

Cette classe SSLContext s'instancie par une nouvelle factory :

```
public static SSLContext getInstance (String protocol, Provider provider)  
throws NoSuchAlgorithmException, NoSuchProviderException
```

---

<sup>1</sup> il est aussi possible préciser tous ces paramètres sur la ligne de commande l'interpréteur.

une version polymorphe se contentant d'ailleurs d'un provider par défaut. Le premier paramètre est par contre incontournable, puisqu'il désigne le protocole et surtout la version utilisée. Dans notre cas,

```
SSLContext SslC = null;  
SslC = SSLContext.getInstance("SSLv3");
```

Le contexte ainsi obtenu ne contient encore aucune donnée SSL. Où se trouvent les informations SSL nécessaires ? Dans un "magasin à clé" bien sûr ...

#### **4.3 Keystore : le retour**

On se souviendra que le principe et les manipulations de base du keystore et de l'outil keytool ont été évoqués dans un chapitre précédent<sup>1</sup>. Contentons nous de rappeler que de la classe **KeyStore**, est définie dans le package java.security et contient, de manière cryptée, des entrées de type **Key Entry** (une clé privée et une liste de certificats concernant la clé publique correspondante) et **Trusted Certificate Entry** (un certificat d'une personne considérée comme sûre). On obtient une instance de KeyStore au moyen de

```
public static KeyStore getInstance(String type) throws KeyStoreException
```

Si l'on suppose avoir créé (nous verrons en détails comment faire – voir plus loin dans ce chapitre), au moyen de l'outil keytool, deux keystores du type suivant :

**serveur\_keystore** (mot de passe = beaugosser; mot de passe de la clé privé = sexyser)

Type Keystore : jks

Fournisseur Keystore : SUN

**certificauthority**, 23-avr.-2004, trustedCertEntry,

Empreinte du certificat (MD5) : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94

**claudeser**, 23-avr.-2004, keyEntry,

Empreinte du certificat (MD5) : EF:F7:FD:30:61:0C:9B:30:78:3F:26:48:2F:F6:21:41

**client\_keystore** (mot de passe = beaugossecli; mot de passe de la clé privé = sexycli)

Type Keystore : jks

Fournisseur Keystore : SUN

**claudecki**, 20-avr.-2004, keyEntry,

Empreinte du certificat (MD5) : 32:FB:9C:B7:31:A9:6E:3D:39:E5:10:00:03:01:4A:93

**certificauthority**, 20-avr.-2004, trustedCertEntry,

Empreinte du certificat (MD5) : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94

nous pourrons instancier les objets Java Keystore correspondant à partir des fichiers keystore en utilisant

---

◆ <sup>1</sup> voir chapitre "XIV. La cryptographie et Java" dans "Langage Java (II) : Programmation avancée des applications classiques et cryptographie " du même (excellent ;-) auteur

```
public final void load(InputStream stream, char[] password)
    throws IOException, NoSuchAlgorithmException, CertificateException
```

Cette méthode vérifiera l'intégrité du keystore avant de charger effectivement l'objet en mémoire (ce qui explique le 3<sup>ème</sup> type d'exception potentiel). Donc finalement, pour le cas du serveur, les instructions sont (le keystore visé se trouve dans un répertoire makecert) :

```
KeyStore ServerKs = KeyStore.getInstance("JKS");
String FICHIER_KEYSTORE = "c:\\makecert\\serveur_keystore";
char[] PASSWD_KEYSTORE = "beaugosse".toCharArray();
FileInputStream ServerFK = new FileInputStream(FICHIER_KEYSTORE);
ServerKs.load(ServerFK, PASSWD_KEYSTORE);
```

#### **4.4 L'initialisation du contexte SSL**

Il faut à présent placer les informations fournies par le keystore dans l'objet contexte SSL, au moyen de la méthode:

```
public final void init (KeyManager[] km, TrustManager[] tm, SecureRandom random)
    throws KeyManagementException
```

KeyManager et TrustManager sont des *interfaces vides* du package com.sun.net.ssl. Leur nom indique clairement qu'ils correspondent à des objets capables d'utiliser respectivement les Key entries et les Trusted certificate entries. Comment faire pour obtenir ces tableaux d'objets ? Encore une fois, au moyen de factories (ce qui est logique puisque ce l'on attend n'est connu que sous forme d'interface) nommées évidemment **KeyManagerFactory** et **TrustManagerFactory** du même package. On obtient des instances au moyen de

```
public static final KeyManagerFactory getInstance(String algorithm, String provider)
    throws NoSuchAlgorithmException, NoSuchProviderException
public static final TrustManagerFactory getInstance(String algorithm, String provider)
    throws NoSuchAlgorithmException, NoSuchProviderException
```

mais le deuxième paramètre est en fait optionnel – nous utiliserons donc ici la version polymorphe qui se contente de demander l'algorithme utilisé pour la gestion des entrées d'un type ou de l'autre. On initialise ces factories par rapport aux keystores qui vont être utilisés respectivement au moyen de :

```
public void init (KeyStore ks, char[] password)
    throws KeyStoreException, NoSuchAlgorithmException, UnrecoverableKeyException
```

et

```
public void init (KeyStore ks) throws KeyStoreException
```

Elles sont dès lors en état de fournir les KeyManger et TrustManager réclamés par l'initialisateur du contexte SSL, cela au moyen des méthodes respectives :

```
public KeyManager[] getKeyManagers()
```

et

```
public TrustManager[] getTrustManagers()
```

Donc, concrètement :

```
SSLContext SslC = SSLContext.getInstance("SSLv3");
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
char[] PASSWD_KEY = "sexyser".toCharArray();
kmf.init(ServerKs, PASSWD_KEY);
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(ServerKs);
SslC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
```

#### 4.5 Les sockets et le code du serveur

On peut à présent obtenir la factory et la socket au moyen du contexte :

```
SSLSocketFactory SslSFac= SslC.getServerSocketFactory();
SSLSocket = (SSLSocket) SslSFac.createServerSocket(6000);
```

En reprenant le modeste serveur qui servit d'exemple dans le chapitre consacré aux communications réseaux en Java<sup>1</sup>, cela donne :

#### ServerSocketSSL.java

```
import java.io.*;
import java.security.*;
import java.security.cert.*;
import javax.net.ssl.*;

public class ServerSocketSSL
{
    public static void main(String[] args)
    {
        // Version non sécurisée
        // ServerSocket SSocket = null;
        // Socket CSocket = null;
        // *** Version sécurisée ***
        SSLSocketFactory SslSFac= SslC.getServerSocketFactory();
        SSLSocket = (SSLSocket) SslSFac.createServerSocket(6000);
        try
        {
            // Version non sécurisée
            // SSocket = new ServerSocket(6000);
        }
    }
}
```

---

<sup>1</sup> voir "Langage Java (II) : Programmation avancée des applications classiques" – du même (et prodigieux ;-)) auteur

```

// *** Version sécurisée ***
// 1. Keystore
KeyStore ServerKs = KeyStore.getInstance("JKS");
String FICHIER_KEYSTORE = "c:\\makecert\\serveur_keystore";
char[] PASSWD_KEYSTORE = "beaugosser".toCharArray();
FileInputStream ServerFK = new FileInputStream (FICHIER_KEYSTORE);
ServerKs.load(ServerFK, PASSWD_KEYSTORE);
// 2. Contexte
SSLContext SslC = SSLContext.getInstance("SSLv3");
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
char[] PASSWD_KEY = "sexyserv".toCharArray();
kmf.init(ServerKs, PASSWD_KEY);
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(ServerKs);
SslC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
// 3. Factory
SSLServerSocketFactory SslSFac= SslC.getServerSocketFactory();
// 4. Socket
SslSSocket = (SSLServerSocket) SslSFac.createServerSocket(6000);
}
catch (IOException e)
{
    System.err.println("Erreur de fichier E/S ! ? [" + e + "]"); System.exit(1);
}
catch (KeyStoreException e)
{
    System.err.println("Erreur de KeyStore ! ? [" + e + "]"); System.exit(1);
}
catch (NoSuchAlgorithmException e)
{
    System.err.println("Erreur d'algorithme au chargement du KeyStore ! ? [" + e + "]");
    System.exit(1);
}
catch (CertificateException e)
{
    System.err.println("Erreur de certificat au chargement du KeyStore ! ? [" + e + "]");
    System.exit(1);
}
catch (KeyManagementException e)
{
    System.err.println("Erreur pour accéder aux Key managers ! ? [" + e + "]");
    System.exit(1);
}
catch (UnrecoverableKeyException e)
{
    System.err.println("Erreur d'initialisation du KeyManagerFactory ! ? [" + e + "]");
    System.exit(1);
}
System.out.println("Serveur en attente");

```

```

try
{
    // Version non sécurisée
    // CSocket = SSocket.accept();
    // *** Version sécurisée ***
    SslSocket = (SSLSocket)SslSSocket.accept();
}
catch (IOException e)
{
    System.err.println("Erreur d'accept ! ? [" + e + "]");
    System.exit(1);
}

try
{
    // DataInputStream dis = new DataInputStream(new BufferedInputStream
    //     (CSocket.getInputStream()));
    // Deprecated !
    BufferedReader dis = new BufferedReader (new InputStreamReader
        (SslSocket.getInputStream()));
    // DataOutputStream dos = new DataOutputStream
    //     (new BufferedOutputStream (CSocket.getOutputStream()));
    BufferedWriter dos = new BufferedWriter (new OutputStreamWriter
        (SslSocket.getOutputStream()));

    String inLigne, outLigne;
    String noms[] = { "vil", "char", "cler", "del", "FIN." };
    String prenoms[] = { "claude", "christophe", "carine", "pierre", "FIN." };
    int pos=0;
    String rep = null;

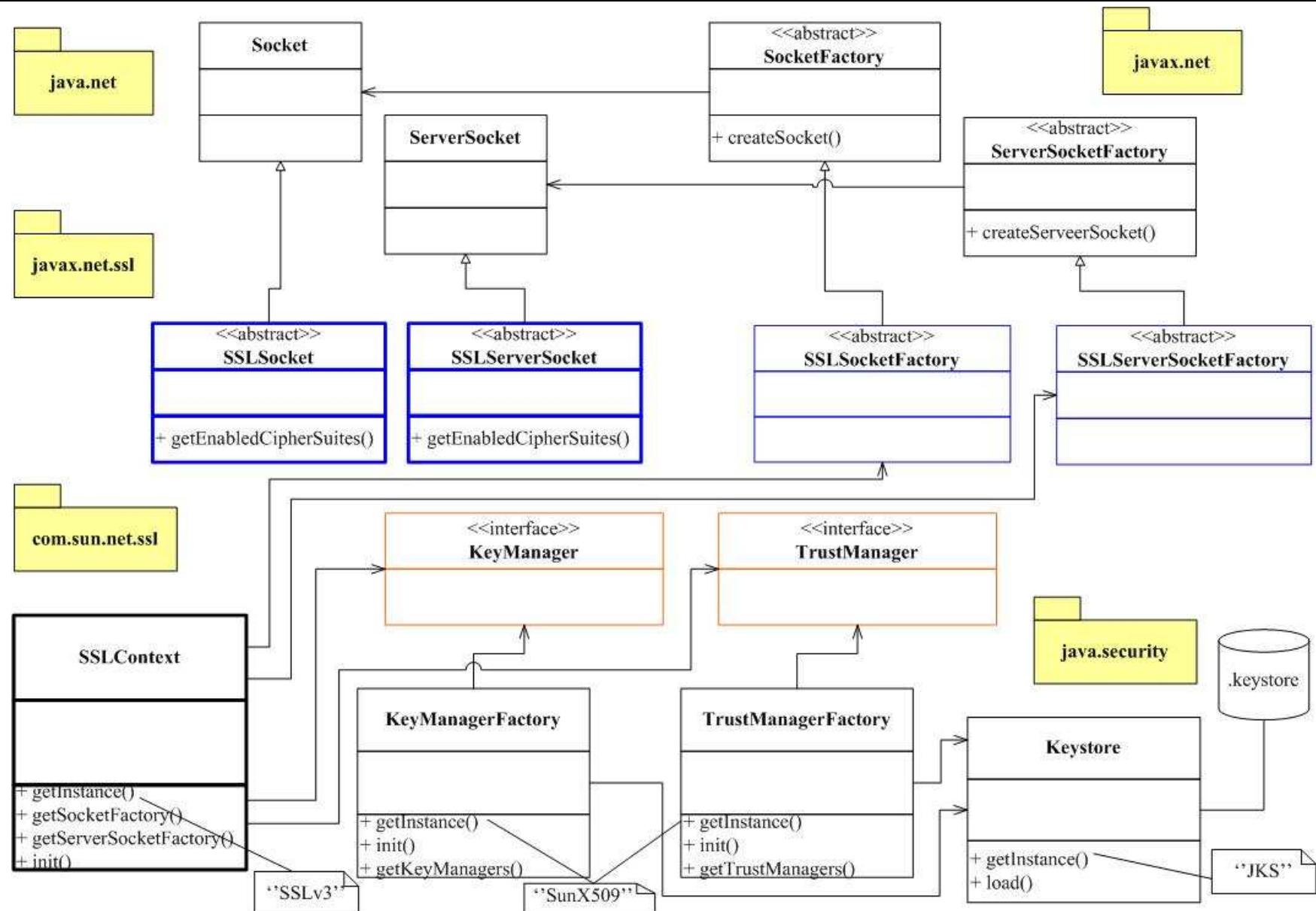
    outLigne = noms[pos];
    dos.write(outLigne + "\n");
    dos.flush();

    while (noms[pos++] != "FIN." && (inLigne=dis.readLine()) != null)
    {
        System.out.println("Réponse reçue = " + inLigne + "(" +
            inLigne.length() + ")");
        if (inLigne.equals(prenoms[pos-1])) rep="OK ";
        else rep="FAUX ";
        System.out.println("Serveur en " + pos);
        outLigne = rep + " -- suite : (" + pos + ")=" + noms[pos];
        dos.write(outLigne + "\n");
        dos.flush();
    }
    dis.close(); dos.close();
    SslSocket.close(); SslSSocket.close();
    System.out.println("Serveur déconnecté");
}
catch (IOException e)
{
    System.err.println("Erreur ! ? [" + e + "]");
}
}
}

```

#### **4.6 Une synthèse en diagramme UML**

Le diagramme de classes UML suivant permet de résumer les interactions :



#### **4.7 Le code du client**

De manière analogue, mais en plus simple, le code du client s'écrit comme ci-dessous et ne réclame sans doute pas de commentaires complémentaires (évidemment, le fichier client\_keystore a remplacé le fichier serveur\_keystore).

##### **ClientSocketSSL.java**

```

import java.io.*;
...
import javax.net.ssl.*;

public class ClientSocketSSL
{
    public static void main(String[] args)
    {
        // Version non sécurisée : Socket cliSock = null;
        // *** Version sécurisée ***
        SSLSocket SslSocket = null;
        BufferedReader dis=null; BufferedWriter dos=null;
        String ligneDuServeur;
        try
        {
            // Version non sécurisée : cliSock = new Socket("claude", 6000);
            // *** Version sécurisée ***
            // 1. Keystore
            KeyStore ServerKs = KeyStore.getInstance("JKS");
            String FICHIER_KEYSTORE = "c:\\makecert\\client_keystore";
            char[] PASSWD_KEYSTORE = "beaugossecli".toCharArray();
            FileInputStream ServerFK = new FileInputStream (FICHIER_KEYSTORE);
            ServerKs.load(ServerFK, PASSWD_KEYSTORE);
            // 2. Contexte
            SSLContext SslC = SSLContext.getInstance("SSLv3");
            KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
            char[] PASSWD_KEY = "sexycli".toCharArray();
            kmf.init(ServerKs, PASSWD_KEY);
            TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
            tmf.init(ServerKs);
            SslC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
            // 3. Factory
            SSLSocketFactory SslSFac= SslC.getSocketFactory();
            // 4. Socket
            SslSocket = (SSLSocket) SslSFac.createSocket("claude", 6000);
            dis = new BufferedReader (new InputStreamReader
                (SslSocket.getInputStream()));
            dos = new BufferedWriter(new OutputStreamWriter
                (SslSocket.getOutputStream()));
        }
        catch (UnknownHostException e)
        { System.err.println("Erreur ! Host non trouvé [" + e + "]"); }
        catch (IOException e)
    }
}

```

```

{ System.err.println("Erreur ! Pas de connexion ? [" + e + "]"); }
catch ...
if (SslSocket==null || dis==null || dos==null) System.exit(1);

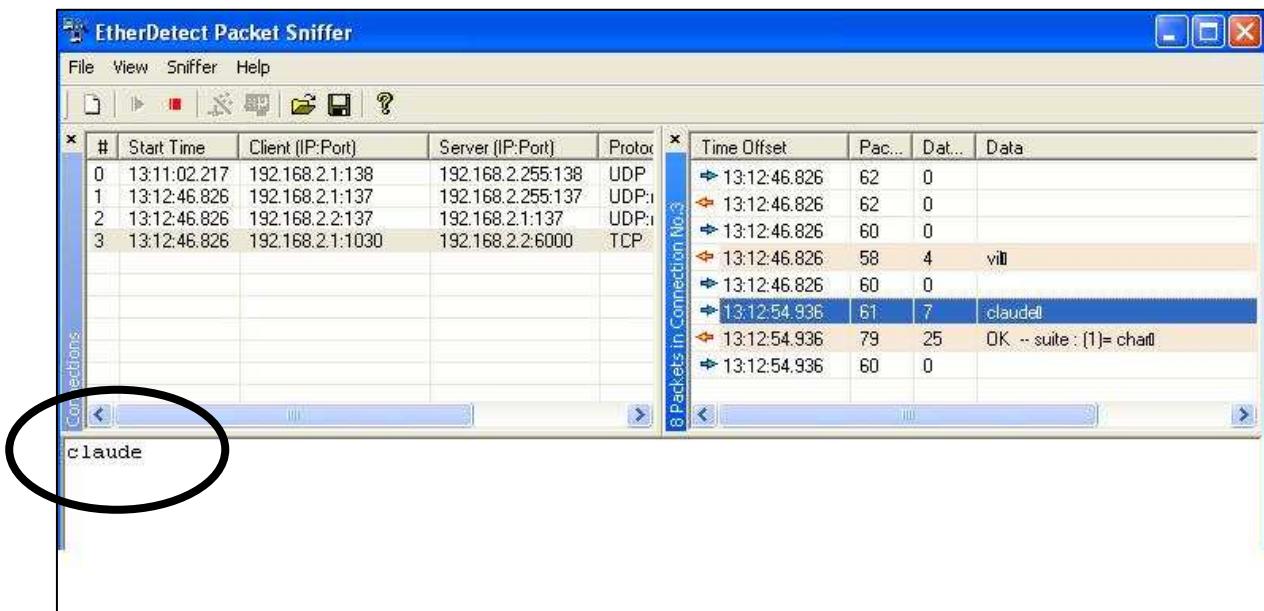
try
{
    StringBuffer buf = new StringBuffer(50);
    int c;
    while ( (ligneDuServeur=dis.readLine()) != null )
    {
        System.out.println("Reçu du serveur : " + ligneDuServeur);
        if (ligneDuServeur.indexOf("FIN.")==-1)
        {
            while((c = System.in.read())!='\n') if (c != '\r') buf.append((char)c);
            System.out.println("Envoyé au serveur : " + buf.toString());
            dos.write(buf.toString()+"\n"); dos.flush(); buf.setLength(0);
        }
    }
    dos.close(); dis.close(); SslSocket.close(); System.out.println("Client déconnecté");
}
catch (UnknownHostException e)
{ System.err.println("Erreur ! Host non trouvé [" + e + "]"); }
catch (IOException e)
{ System.err.println("Erreur ! Pas de connexion ? [" + e + "]"); }
}
}

```

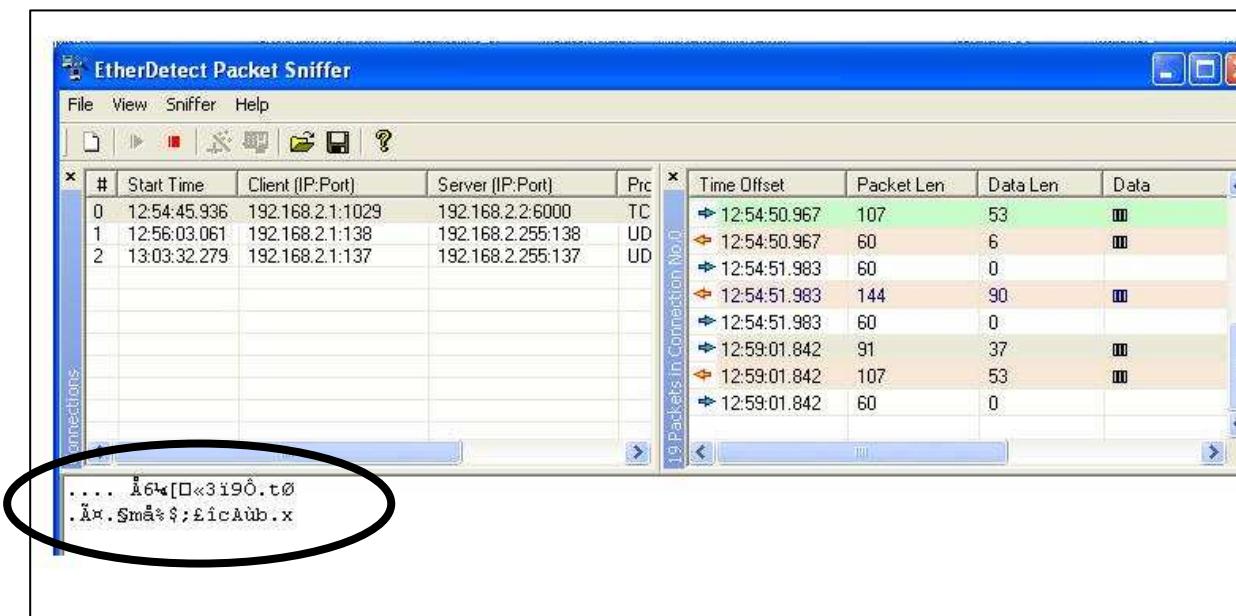
#### **4.8 La visualisation du trafic réseau**

En utilisant un sniffer, on peut visualiser la différence au niveau des données qui transitent sur le réseau :

- en sockets classiques non sécurisées :



- en sockets SSL :



Le cryptage SSL a bien fait son œuvre !

Reste à voir d'où nous viennent nos certificats ... Le chapitre suivant nous permettra de les générer au moyen d'outils (comme **OpenSSL – keytool** et **Keytool IUI** nous sont déjà connus) ou encore par programmation pure.

Mais voyons d'abord ce que pensent de tout cela les applications Webs ...

## 5. Tomcat par rapport à SSL

### 5.1 Un keystore pour HTTPS

A l'image de ci-dessus, supposons disposer d'un keystore dédié à HTTPS (appelons-le `ser_https_keystore`). Celui-ci ne doit posséder que les deux entrées nécessaires, soit :

- ◆ une keyEntry pour le certificat;
- ◆ une trustedCertEntry pour l'autorité qui a certifié ce certificat.

Il s'agit donc de quelque chose de ce genre :

```
C:\cert-factory-https>keytool -list -keystore ser_https_keystore
```

Tapez le mot de passe du Keystore : `bgserhttps`

Type Keystore : jks

Fournisseur Keystore : SUN

Votre Keystore contient 2 entrée(s)

```
claudeli, 11-oct.-2009, PrivateKeyEntry,  

Empreinte du certificat (MD5) : 9C:30:12:1C:DB:C7:01:96:AE:F2:92:A4:61:38:15:7D  

certificauthority, 11-oct.-2009, trustedCertEntry,  

Empreinte du certificat (MD5) : 29:D9:09:7C:8B:FF:F0:75:80:3F:C5:DF:37:74:FA:A2
```

Pour y arriver, nous utilisons par exemple OpenSSL (qui sera exposé en détails dans le chapitre suivant) pour créer les clés et certificats nécessaires et finalement obtenir les fichiers suivants :

```
C:\makecert>dir  
Le volume dans le lecteur C n'a pas de nom.  
Le numéro de série du volume est 6CF8-0DE7  
Répertoire de C:\makecert  
11/10/2009 17:33 <REP> .  
11/10/2009 17:33 <REP> ..  
11/10/2009 17:32 1.024 .rnd  
11/10/2009 17:19 891 ca.key  
11/10/2009 17:31 934 clientHttps.csr  
11/10/2009 17:33 1.191 clientHttps.der  
11/10/2009 17:32 5.203 clientHttps.pem  
11/10/2009 17:32 <REP> demoCA  
11/10/2009 17:27 1.249 ser_https_keystore  
    14 fichier(s) 33.014 octets  
    3 Rép(s) 95.688.167.424 octets libres
```

On peut donc importer les certificats dans le keystore – on veillera à utiliser le même mot de passe pour le keystore et la keyEntry :

```
C:\makecert>keytool -import -v -alias certificauthority -file demoCA\cacert.pem  
-keystore ser_https_keystore  
Tapez le mot de passe du Keystore :  
Propriétaire: EMAILADDRESS=claude.vilvens@provincedeliege.be, CN=VILVENS Claude  
, OU=HEPL, O=HEPL, L=">Oupeye", ST=Liege, C=BE  
Emetteur: EMAILADDRESS=claude.vilvens@provincedeliege.be, CN=VILVENS Claude,  
OU =HEPL, O=HEPL, L=">Oupeye", ST=Liege, C=BE  
Numéro de série: 0  
Valide du: Sun Oct 11 17:21:43 CEST 2009 au: Tue Nov 10 16:21:43 CET 2009  
Empreintes du certificat:  
MD5: 29:D9:09:7C:8B:FF:F0:75:80:3F:C5:DF:37:74:FA:A2  
SHA1: 47:46:A9:83:DA:8C:94:80:06:91:AC:C5:2F:B4:D4:9F:4E:CC:52:20  
...  
Faire confiance à ce certificat ? [non] : oui  
Certificat ajouté au Keystore  
[Stockage de ser_https_keystore]  
et  
C:\makecert>keytool -import -keystore ser_https_keystore -alias ClaudeCli -file  
clientHttps.der  
Tapez le mot de passe du Keystore :  
Réponse de certificat installée dans le Keystore
```

---

## 5.2 La création d'un connecteur https

Pour que Tomcat soit en état, il faut lui ajouter une "connecteur HTTPS" – autrement dit, le configurer pour qu'il écoute également sur un port dédié à HTTPS (typiquement, 443). En fait, il suffit de déclarer ce connecteur dans le fichier server.xml : le tag de base se trouve déjà entre commentaires et il suffit de le compléter avec les informations concernant le keystore :

### server.xml

```
...<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"
    keystoreFile="C:\makecert\ser_https_keystore"
    keystorePass="bgserhttps" />
...
```

On arrête alors Tomcat puis on le relance :

```
C:\makecert>netstat -an | find "8"
TCP 0.0.0.0:8009 0.0.0.0:0 LISTENING
TCP 0.0.0.0:8080 0.0.0.0:0 LISTENING
TCP 0.0.0.0:8443 0.0.0.0:0 LISTENING
...
TCP 127.0.0.1:3105 127.0.0.1:8080 TIME_WAIT
...
TCP 192.168.1.4:2860 72.5.124.55:80 CLOSE_WAIT
TCP 192.168.1.4:2867 194.78.100.32:80 CLOSE_WAIT
TCP 192.168.1.4:3459 194.78.100.10:80 CLOSE_WAIT
```

Le serveur attend donc bien à présent sur deux ports (sans compter le port de shutdown).

### Remarque

Pour Tomcat 5.\*, on disposait d'un *outil d'administration* accédé au moyen de l'URL :

<http://localhost:8090/admin>

ou encore en cliquant sur le lien "admin" dans le Tomcat manager – ce qui appelle manifestement un JSP :

<http://127.0.0.1:8080/admin/frameset.jsp>

On obtient un interface graphique invitant à entrer le nom d'administrateur et son mot de passe :



puis on accède à :

Property	Value
Port Number:	8005
Debug Level:	0
Shutdown:	SHUTDOWN

qui permet en fait d'obtenir dans server.xml ce que nous avons fait directement !

### 5.3 Utilisation du browser en https

Nous pouvons donc à présent accéder à Tomcat sur une URL de type

<https://localhost:8443/>

On obtient l'avertissement :

**Cette connexion n'est pas certifiée**

Vous avez demandé à Firefox de se connecter de manière sécurisée à **localhost:8443**, mais nous ne pouvons pas confirmer que votre connexion est sécurisée.

Normalement, lorsque vous essayez de vous connecter de manière sécurisée, les sites présentent une identification certifiée pour prouver que vous vous trouvez à la bonne adresse. Cependant, l'identité de ce site ne peut pas être vérifiée.

**Que dois-je faire ?**

Si vous vous connectez habituellement à ce site sans problème, cette erreur peut signifier que quelqu'un essaie d'usurper l'identité de ce site et vous ne devriez pas continuer.

[Sortir d'ici !](#)

**Détails techniques**

localhost:8443 utilise un certificat de sécurité invalide.

Le certificat n'est pas sûr car l'autorité délivrant le certificat n'est pas éprouvée.  
Le certificat n'est valide que pour Claude Vilvens.  
(Code d'erreur : sec\_error\_untrusted\_issuer)

**Je comprends les risques**

Si vous comprenez ce qui se passe, vous pouvez indiquer à Firefox de commencer à faire confiance à l'identification de ce site. **Même si vous avez confiance en ce site, cette erreur pourrait signifier que quelqu'un est en train de pirater votre connexion.**

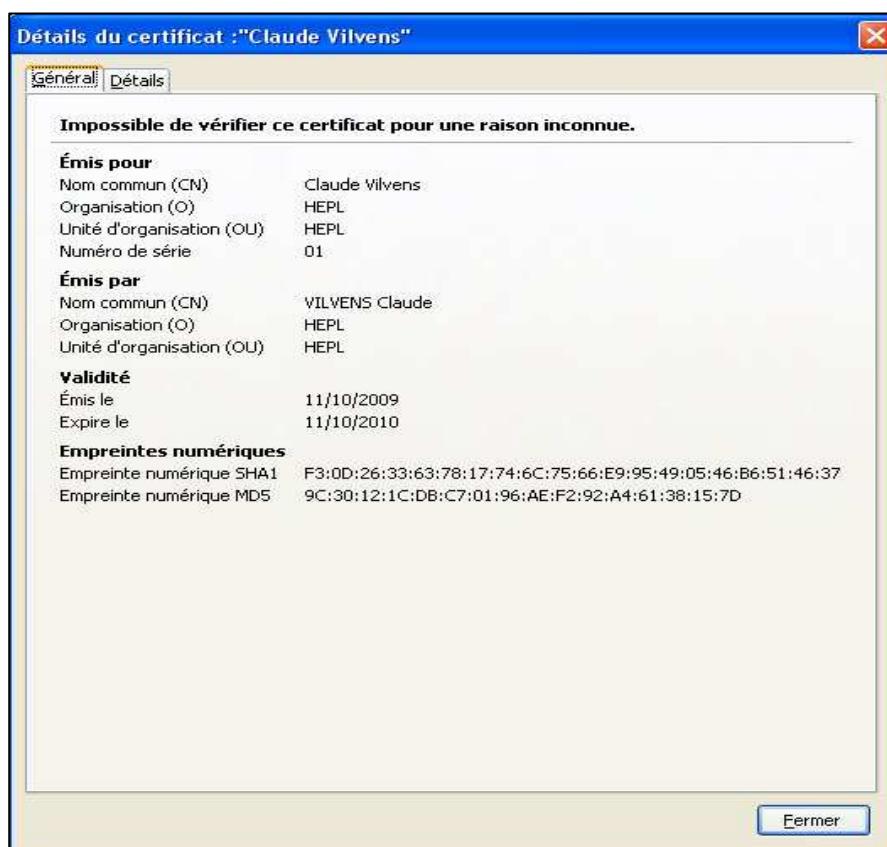
N'ajoutez pas d'exception à moins que vous ne connaissiez une bonne raison pour laquelle ce site n'utilise pas d'identification certifiée.

[Ajouter une exception...](#)

Choisir d'ajouter une exception à la politique de sécurité du browser :



Un appui sur le bouton "Voir" permet de s'assurer que le certificat est bien celui que l'on croit :



On peut donc confirmer (après avoir cependant décoché la case "Conserver cette exception de façon permanente") et arriver sur la page d'accueil de Tomcat :

If you're seeing this page via a web browser, it means you've setup Tomcat successfully.  
Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

`$CATALINA_HOME/webapps/ROOT/index.html`

where "\$CATALINA\_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the [Tomcat Documentation](#) for more detailed setup and administration information than is found in the INSTALL file.

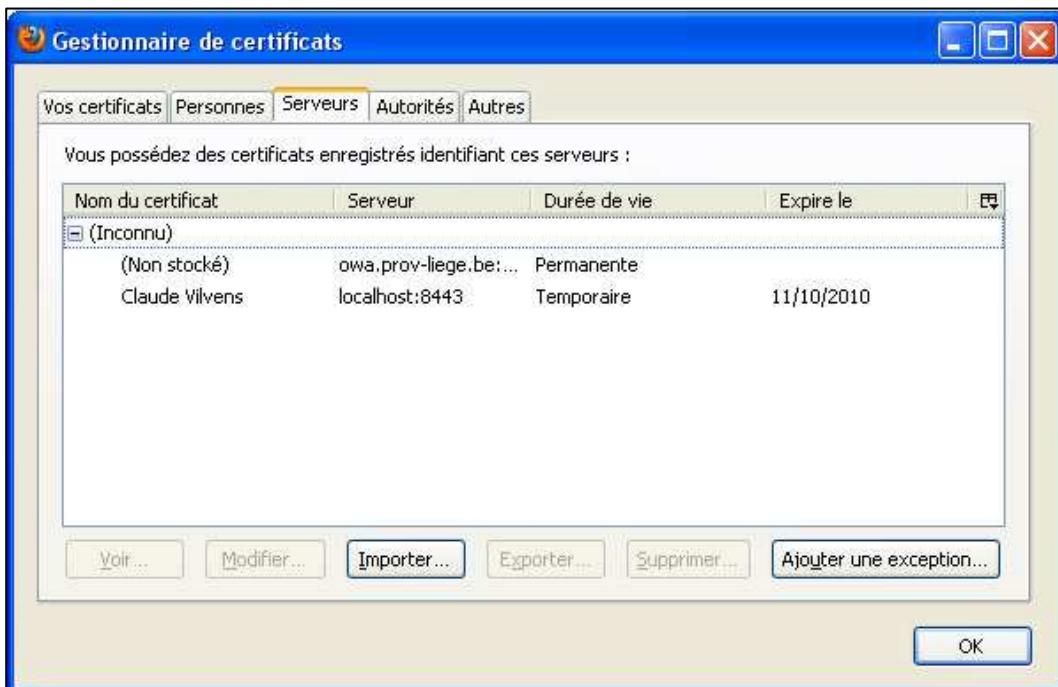
**NOTE: For security reasons, using the manager webapp is restricted to users with role "manager".**  
Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

On peut vérifier l'effet de cette exception dans le browser par

Outils → Options → Avancé → Serveurs :



### **Remarque**

Pour rappel, Tomcat impose l'unicité du mot de passe pour le keystore et les KeyEntry. Si on a négligé ce "détail", on obtient au lancement du serveur quelque chose de ce genre (en faisant glisser l'item de menu "Apache – Start Tomcat" dans une fenêtre DOS) :

```
C:>"C:\Documents and Settings\Vilvens\Menu Démarrer\Programmes\Apache Tomcat  
4.1\Start Tomcat.lnk"
```

```
C:>08-juin-2004 15:59:11 org.apache.coyote.http11.Http11Protocol init  
INFO: Initialisation de Coyote HTTP/1.1 sur le port 8090  
08-juin-2004 15:59:11 org.apache.coyote.http11.Http11Protocol init  
GRAVE: Erreur à l'initialisation du point de contact  
java.io.IOException: Cannot recover key  
    at org.apache.tomcat.util.net.jsse.JSSE14SocketFactory.init(JSSE14Socket  
Factory.java:151)  
...
```

Si tel est le cas, on peut corriger un mot de passe par :

```
C:\cert-factory-https>keytool -storepasswd -v -new bgserhttpscle -keystore  
ser_https_keystore  
Tapez le mot de passe du Keystore : bgserhttps  
[Enregistrement de ser_https_keystore]
```

## **6. Une communication applet-servlet sécurisée par SSL**

Muni de notre Tomcat prêt à accepter des requêtes de type https, nous allons à présent reprendre le grand classique de la communication applet-servlet qu'est le tunnel http, présenté dans le volume consacré aux applications WEB, et le décliner selon un dialogue SSL.

En fait, la servlet de l'époque ne se rendra compte de rien (si on peut donner une conscience à une servlet ;-)) car notre Tomcat est capable à présent de gérer les connexions http sécurisées par SSL.

Pour l'applet, les choses sont par contre différentes. En effet, l'objet URLConnection classique doit être remplacé par une instance de la classe du package com.sun.net.ssl :

```
public abstract class HttpsURLConnection extends HttpURLConnection
```

la classe mère provenant elle-même du package standard java.net. Cette classe vouée manifestement à HTTPS comporte la méthode :

```
public void setSSLSocketFactory (SSLocketFactory sf)
```

qui permet bien sûr, comme dans une application cliente, de désigner la factory qui fournira les sockets sécurisées en fonction du contexte et du keystore utilisé. Notre applet s'écrira donc :

**FormPostAppletSSL.java**

```

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

import javax.net.ssl.*;
import java.security.*;
import java.security.cert.*;

public class FormPostAppletSSL extends java.applet.Applet implements ActionListener
{
    TextField nom, prenom, sexe;
    TextArea resultat;

    public void init()
    {
        initComponents();
        resize(400,200); setLayout(new GridLayout(6,2));
        add(new Label("/** Les joyeux commerçants **"));
        add(new Label("Veuillez entrer vos coordonnées ..."));
        add(new Label ("Nom : ")); nom = new TextField("",20);add(nom);
        add(new Label ("Prenom : ")); prenom = new TextField("",20);add(prenom);
        add(new Label ("Sexe (1 ou 2) : ")); sexe = new TextField("",1);add(sexe);
        add(new Label("A vous !!!"));

        Button seConnecter = new Button ("se connecter"); add(seConnecter);
        add(new Label ("Résultat opération : : "));
        resultat = new TextArea("", 10, 30, TextArea.SCROLLBARS_BOTH);add(resultat);
        seConnecter.addActionListener(this);
    }

    private void initComponents() {
        setLayout(new java.awt.BorderLayout());
    }

    public void actionPerformed (ActionEvent e)
    {
        if (e.getActionCommand() == "se connecter")
        {
            try
            {
                // *** Version sécurisée ***
                // 1. Keystore
                KeyStore ClientKs = KeyStore.getInstance("JKS");
                String FICHIER_KEYSTORE = "c:\\makecert\\client_keystore";
                char[] PASSWD_KEYSTORE = "beaugossecli".toCharArray();
                FileInputStream ClientFK = new FileInputStream (FICHIER_KEYSTORE);
                ClientKs.load(ClientFK, PASSWD_KEYSTORE);
            }
        }
    }
}

```

```

// 2. Contexte
SSLContext SslC = SSLContext.getInstance("SSLv3");
KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
char[] PASSWD_KEY = "sexycli".toCharArray();
kmf.init(ClientKs, PASSWD_KEY);
TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");
tmf.init(ClientKs);
SslC.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
// 3. Factory
SSLSocketFactory SslSFac= SslC.getSocketFactory();

// Pour Tomcat standalone
String adresseServlet = "https://claudie:8090/servlet/FormPostServlet";
URL urlServlet = new URL(adresseServlet);
HttpsURLConnection connexion =
    (HttpsURLConnection) urlServlet.openConnection();
// if needed: connexion.setHostnameVerifier(new HostnameVerifier { ... });
connexion.setSSLSocketFactory(SslSFac);

connexion.setUseCaches(false); connexion.setDefaultUseCaches(false);
connexion.setDoOutput(true);
ByteArrayOutputStream baos = new ByteArrayOutputStream(512);
PrintWriter pw = new PrintWriter(baos, true);

String n = nom.getText(); String p = prenom.getText();
String s = sexe.getText();
String infos = "nom=" + URLEncoder.encode(n) + "&prenom=" +
    URLEncoder.encode(p) + "&sexe=" + URLEncoder.encode(s);
pw.print(infos); pw.flush();

String longueurEnvoyee = String.valueOf(baos.size());
connexion.setRequestProperty("Content-Length", longueurEnvoyee);
connexion.setRequestProperty("Content-Type",
    "application/x-www-form-urlencoded");
baos.writeTo(connexion.getOutputStream());

BufferedReader entree = new BufferedReader(
    new InputStreamReader(connexion.getInputStream()));

String reponse;
while ((reponse=entree.readLine()) != null)
{
    resultat.append(reponse);
    resultat.append("\n");
}
}

catch (MalformedURLException ex)
{System.out.println("Aie aie - URL louche ! : " + ex.getMessage()); }
catch (IOException ex)
{ System.out.println("Aie aie - probleme ouverture connexion ! : " +
    ex.getMessage()); }

```

```
catch (KeyStoreException ex)
{ System.err.println("Erreur de KeyStore ! ? [" + e + "]"); System.exit(1); }
catch (NoSuchAlgorithmException ex)
{ System.err.println("Erreur d'algorithme au chargement du KeyStore ! ? [" +
+ "]); System.exit(1); }
catch (CertificateException ex)
{ System.err.println("Erreur de certificat au chargement du KeyStore ! ? [
+ e + "]"); System.exit(1); }
catch (KeyManagementException ex)
{ System.err.println("Erreur pour accéder aux Key managers ! ? [" + e + "]");
System.exit(1); }
catch (UnrecoverableKeyException ex)
{ System.err.println("Erreur d'initialisation du KeyManagerFactory ! ? [
+ e +
+ "]); System.exit(1); }
}
}
}
```

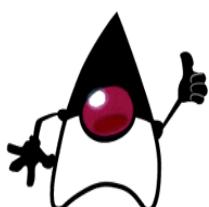
Une difficulté supplémentaire se fait cependant jour : *notre applet doit lire le fichier keystore*. Or, par défaut, une telle opération lui est interdite. Il nous faudra donc la signer pour pouvoir en vérifier l'authenticité et lui donner la permission nécessaire. Après avoir placé notre applet dans un fichier FormPostAppletSignee.jar, nous allons la signer :

```
C:\java-sun-application\TunnelHttp>jarsigner -keystore c:\makecert\serveur_keystore
FormPostAppletSignee.jar claudeser
Enter Passphrase for keystore: beaugosser
Enter key password for claudeser: sexyser
```

et il ne restera plus qu'à la placer dans une page html FormPostAppletSignee.html comportant le tag :

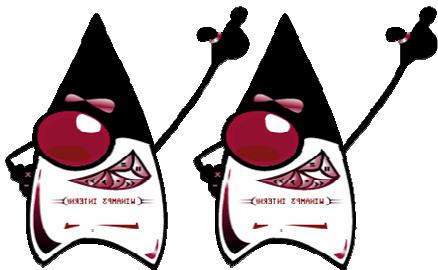
```
<APPLET code="FormPostAppletSSL.class" archive="FormPostAppletSignee.jar"
width=350 height=200></APPLET>
```

Si, au moyen du policytool, nous avons bien donné l'autorisation de lecture de fichier aux applets provenant de notre serveur, le tunnel fonctionnera apparemment comme celui que nous connaissons, mais, en fait, il le fera de manière sécurisée !



Les certificats jouent bien clairement un rôle fondamental dans tout ceci. Mais comment dépasser le stade de certificats basiques auto-signés ?

## XXVI. La génération des certificats



*Un secret, ce n'est pas quelque chose qui ne se raconte pas. Mais c'est une chose qu'on se raconte à voix basse et séparément.*

(M.Pagnol, César)

### 1. Rappels : les outils keytool et keytool IUI

Nous avons dans le chapitre précédent fait un large usage des certificats. On peut évidemment se demander comment se procurer ces certificats (c'est facile, il suffit d'en acheter ;-)) ou comment en fabriquer pour un usage local (problème typique des développeurs et des centres d'enseignement).

Dans tous les cas, ces certificats sont hébergés dans des keystores que l'on peut voir comme des fichiers particuliers particulièrement protégés (et pour cause !). Pour gérer des "magasins à clés", nous connaissons déjà l'outil **keytool** (voir Java II, chapitre "La

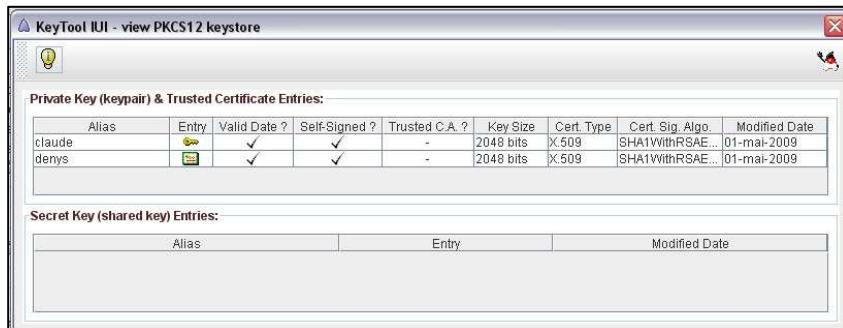
```
invite de commandes
C:\java-netbeans-application>keytool -list -v -keystore java_keystore.jks
Tapez le mot de passe du Keystore :
Type Keystore : JKS
Fournisseur Keystore : SUN
Votre Keystore contient 1 entrée(s)

Nom d'alias : denys
Date de création : 01-mai-2009
Type de trousseau : {0}
Longueur de chaîne du certificat : 1
Certificat[1]:
Propriétaires : CN=Mercenier, OU=HEPL, O=EPL, L=Seraing, ST=Lippe, C=BE, EMAILADDRESS=denys.mercenier@provinceodelige.be, T=Mr
Emetteur : CN=Mercenier, OU=HEPL, O=EPL, L=Seraing, ST=Lippe, C=BE, EMAILADDRESS=denys.mercenier@provinceodelige.be, T=Mr
Numéro de série : 49fb13ea
Valide du : Fri May 01 17:23:22 CEST 2009 au : Mon Apr 30 17:23:22 CEST 2012
Empreintes du certificat :
    MD5: 63:3D:4D:95:8E:E8:01:54:10:47:EA:DE:04:5D:E9:78
    SHA1: C8:0B:D2:16:88:89:EB:BD:97:10:84:93:CA:16:68:69:6F:18:B1:85
    Nom de l'algorithme de signature : {?}
    Version : {?}
```

cryptographie et Java", paragraphe 14) qui fait partie du JDK et dont les possibilités en termes de génération de certificats en standalone sont limitées (il faut impérativement un CA extérieur). De plus, il s'agit d'un outil en ligne de commande qui ne permet une gestion efficace que

des keystores JKS et JCEKS, assez spécifiques au monde Java.

Pour ces raisons, on se tourne bien volontiers vers l'outil graphique **Keytool IUI** (voir à nouveau Java II, chapitre "La



cryptographie et Java", paragraphe 15), écrit en Java (JDK 1.6) et utilisant la librairie BouncyCastle. Outre sa facilité d'utilisation, ce petit logiciel permet une gestion basique mais déjà

assez étendue des certificats stockés dans un keystore.

Mais tout ceci tourne toujours autour des certificats auto-signés. Or, il existe un outil bien plus puissant, indépendant de l'environnement et open source ...

## 2. L'outil OpenSSL

### 2.1 Présentation et installation

OpenSSL (0.9.\* ou 1.0.\*) est un projet Open Source proposant une implémentation de SSL/TLS ainsi qu'une librairie cryptographique<sup>1</sup>. On y trouve notamment un outil qui permet de créer des certificats **X509** tout à fait semblables à ceux que l'on peut demander et obtenir auprès d'une autorité de certification. Une seule limite, mais elle est, bien logiquement, de poids : *le CA et le propriétaire du certificat doivent appartenir à la même organisation (et la même unité de cette organisation)* (champs O et OU du *distinguished name - DN*).

Dans le répertoire OpenSSL 0.9.8i créé au téléchargement, on trouve dans le répertoire Windows (ou Linux) l'exécutable d'installation Win32OpenSSL-v0.9.7d.exe (ou un fichier tar dans le cas de Linux). L'installation en question est immédiate dans un répertoire OpenSSL. On définit les variables d'environnement classiques :

```
C:\makecert\demoCA>set OPENSSL_HOME=c:\OpenSSL
C:\makecert\demoCA>set PATH=%PATH%;%OPENSSL_HOME%\bin
```

Le fichier de configuration est **openssl.cnf**, qui se trouve dans le répertoire bin du répertoire OpenSSL créé lors de l'installation, ainsi qu'indiqué par la variable d'environnement

```
OPENSSL_CONF=C:\OpenSSL\bin\openssl.cnf
```

Son contenu est du type suivant :

<b>openssl.cnf</b> ((OpenSSL\bin))
<pre># # OpenSSL example configuration file. # This is mostly being used for generation of certificate requests.  # This definition stops the following lines choking if HOME isn't # defined. HOME          = . RANDFILE      = \$ENV::HOME/.rnd  # Extra OBJECT IDENTIFIER info: #oid_file      = \$ENV::HOME/.oid oid_section    = new_oids ... [ new_oids ]  # We can add new OIDs in here for use by 'ca' and 'req'. # Add a simple OID like this: # testoid1=1.2.3.4 ...  #####</pre>

<sup>1</sup> <http://www.openssl.org>



```
[ ca ]
default_ca      = CA_default          # The default ca section

#####
[ CA_default ]

dir            = ./demoCA            # Where everything is kept
certs          = $dir/certs          # Where the issued certs are kept
crl_dir        = $dir/crl             # Where the issued crl are kept
database       = $dir/index.txt      # database index file.
#unique_subject = no                # Set to 'no' to allow creation of
# several certificates with same subject.
new_certs_dir = $dir/newcerts       # default place for new certs.

certificate   = $dir/cacert.pem     # The CA certificate
serial         = $dir/serial          # The current serial number
#crlnumber     = $dir/crlnumber      # the current crl number
# must be commented out to leave a V1 CRL
crl            = $dir/crl.pem        # The current CRL
private_key    = $dir/private/cakey.pem# The private key
RANDFILE       = $dir/private/.rand  # private random number file

x509_extensions = usr_cert         # The extention to add to the cert

name_opt       = ca_default         # Subject Name options
cert_opt       = ca_default         # Certificate field options

default_days = 365                # how long to certify for
default_crl_days= 30              # how long before next CRL
default_md   = md5                # which md to use.
preserve       = no                 # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-
policy         = policy_match

# For the CA policy
[ policy_match ]
countryName     = match
stateOrProvinceName= match
organizationName = match
organizationalUnitName = optional
commonName      = supplied
emailAddress    = optional

# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
```

```
[ policyAnything ]
countryName      = optional
stateOrProvinceName= optional
localityName     = optional
organizationName = optional
organizationalUnitName = optional
commonName       = supplied
emailAddress     = optional

#####
[ req ]
default_bits      = 1024
default_keyfile   = privkey.pem
distinguished_name = req_distinguished_name
attributes        = req_attributes
x509_extensions   = v3_ca      # The extentions to add to the self signed cert
...
[ req_distinguished_name ]
countryName          = Country Name (2 letter code)
countryName_default  = AU
...
[ req_attributes ]
challengePassword    = A challenge password
...
[ usr_cert ]
...
```

Comme on peut le constater à la lecture de ce fichier, il faut tout d'abord créer un schéma de répertoires correspondant à ceux définis dans ce fichier de configuration :

```
C:\>md makecert
C:\>cd makecert
C:\makecert>md demoCA
C:\makecert>cd demoCA
C:\makecert\demoCA>md certs
C:\makecert\demoCA>md crl
C:\makecert\demoCA>md newcerts
C:\makecert\demoCA>
```

On crée un fichier index vide :

```
| C:\makecert\demoCA>edit index.txt
```

On initialise les numéros de série :

```
| C:\makecert\demoCA>echo 01 > serial
```

Il reste à créer un répertoire pour les clés privées :

```
C:\makecert\demoCA>md private
C:\makecert\demoCA>dir
Le volume dans le lecteur C s'appelle Programmes
Le numéro de série du volume est C491-02F0
```

Répertoire de C:\makecert\demoCA

```
18/04/2004 17:26 <REP> .
18/04/2004 17:26 <REP> ..
18/04/2004 17:06 <REP> certs
18/04/2004 17:06 <REP> crl
18/04/2004 17:09 0 index.txt
18/04/2004 17:13 <REP> newcerts
18/04/2004 17:26 <REP> private
18/04/2004 17:22 5 serial
      2 fichier(s)      5 octets
      6 Rép(s) 72.779.010.048 octets libres
C:\makecert\demoCA>
```

Tout est prêt pour la confection d'un certificat convenable !

## 2.2 La création d'un certificat auto-signé pour un CA

Pour fabriquer un certificat (qui comporte une clé publique), il nous faut un CA et donc sa signature; une signature étant basée sur une clé privée, il nous faut donc une clé privée. Nous allons générer ces clés avec l'outil Openssl, dont on peut découvrir les possibilités :

```
C:\makecert>openssl
OpenSSL> help
openssl:Error: 'help' is an invalid command.
```

### Standard commands

```
asn1parse   ca        ciphers    crl      crl2pkcs7
dgst       dh        dhparam   dsa      dsaparam
enc        engine   errstr    gendh   gendsa
genrsa     nseq     ocsp      passwd  pkcs12
pkcs7     pkcs8    rand      req     rsa
rsautl    s_client s_server s_time sess_id
smime     speed    spkac    verify  version
x509
```

*Message Digest commands* (see the `dgst' command for more details)

```
md2       md4       md5       mdc2      rmd160
sha       sha1
```

*Cipher commands* (see the `enc' command for more details)

```
aes-128-cbc  aes-128-ecb  aes-192-cbc  aes-192-ecb  aes-256-cbc
aes-256-ecb  base64      bf         bf-cbc    bf-cfb
bf-ecb       bf-ofb     cast      cast-cbc  cast5-cbc
```

```
cast5-cfb    cast5-ecb    cast5-ofb    des        des-cbc  
des-cfb    des-ecb    des-edc    des-edc-cbc    des-edc-cfb  
des-edc-ofb    des-edc3    des-edc3-cbc    des-edc3-cfb    des-edc3-of  
des-ofb    des3        desx        idea        idea-cbc  
idea-cfb    idea-ecb    idea-ofb    rc2        rc2-40-cbc  
rc2-64-cbc    rc2-cbc    rc2-cfb    rc2-ecb    rc2-ofb  
rc4        rc4-40      rc5        rc5-cbc    rc5-efb  
rc5-ecb    rc5-ofb
```

**OpenSSL>**

Sous le prompt d'OpenSSL, générerons donc **une paire de clés** au moyen de la commande genrsa dont la syntaxe est :

```
genrsa [args] [numbits]  
-des      encrypt the generated key with DES in cbc mode  
-des3     encrypt the generated key with DES in ede cbc mode (168 bit key)  
-idea     encrypt the generated key with IDEA in cbc mode  
-aes128, -aes192, -aes256  
          encrypt PEM output with cbc aes  
-out file   output the key to 'file'  
-passout arg  output file pass phrase source  
-f4        use F4 (0x10001) for the E value  
-3         use 3 for the E value  
-engine e   use engine e, possibly a hardware device.  
-rand file;file;...  
          load the file (or the files in the directory) into the random number generator
```

Ici, nous demandons une clé privée de type RSA à 1024 bits, sauvegardée dans un fichier ca.key :

```
OpenSSL> genrsa -out ca.key 1024  
Loading 'screen' into random state - done  
Generating RSA private key, 512 bit long modulus  
.....++++++  
....++++++  
e is 65537 (0x10001)  
OpenSSL>
```

Nous allons à présent nous créer **un certificat X509** au moyen de la commande req :

```
req [options] <infile > outfile  
-inform arg  input format - DER or PEM  
-outform arg  output format - DER or PEM  
-in arg      input file  
-out arg     output file  
-text        text form of request  
-pubkey      output public key  
-noout       do not output REQ
```

---

```
-verify      verify signature on REQ
-modulus    RSA modulus
-nodes       don't encrypt the output key
-engine e   use engine e, possibly a hardware device
-subject     output the request's subject
-passin      private key password source
-key file    use the private key contained in file
-keyform arg key file format
-keyout arg  file to send the key to
-rand file;file;... load the file (or the files in the directory) into the random number generator
-newkey rsa:bits generate a new RSA key of 'bits' in size
-newkey dsa:file generate a new DSA key, parameters taken from CA in 'file'
-[digest]   Digest to sign with (md5, sha1, md2, mdc2, md4)
-config file request template file.
-subj arg    set or modify request subject
-new        new request.
-batch      do not ask anything during request generation
-x509       output a x509 structure instead of a cert. req.
-days       number of days a certificate generated by -x509 is valid for.
-set_serial serial number to use for a certificate generated by -x509.
...
-extentions .. specify certificate extension section (override value in config file)
-reqexts ..  specify request extension section (override value in config file)
-utf8       input characters are UTF8 (default ASCII)
...
...
```

Ce certificat, signé au moyen de la clé privée du CA, contiendra les informations que nous fournirons à la commande :

```
OpenSSL> req -new -x509 -key ca.key -out demoCA/cacert.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BE
State or Province Name (full name) [Some-State]:Liege
Locality Name (eg, city) []:Oupeye
Organization Name (eg, company) [Internet Widgits Pty Ltd]:HEP R Sualem
Organizational Unit Name (eg, section) []:Dept. Informatique
Common Name (eg, YOUR name) []: VILVENS Claude
Email Address []:civilvens@prov-liege.be
OpenSSL>exit
```

## 2.3 La création des keystores

Dans le contexte d'une communication réseau SSL, le serveur et le client auront chacun besoin d'un certificat attestant de leur clé publique. Il nous faut donc au préalable

---

générer pour chacun d'entre eux une coupe clé privée/clé publique (elles seront stockées dans leur "magasin à clés" – leur "keystore") puis exporter (c'est-à-dire rendre public) la clé publique pour générer le certificat correspondant. Nous connaissons l'outil **keytool**, qui fait partie du JDK/JSDK. Pour l'utiliser, une précaution :

```
| C:\makecert>set PATH=C:\Program Files\Java\jdk1.6.0_13\bin;%PATH%
```

Dès lors :

```
C:\makecert>keytool -genkey -alias ClaudeCli -keystore client_keystore
Tapez le mot de passe du Keystore : beaugossecli
Quels sont vos prénom et nom ?
[Unknown] : Claude Vilvens
Quel est le nom de votre unité organisationnelle ?
[Unknown] : Dept. Informatique (clients)
Quelle est le nom de votre organisation ?
[Unknown] : HEP R Sualem
Quel est le nom de votre ville de résidence ?
[Unknown] : Oupeye
Quel est le nom de votre état ou province ?
[Unknown] : Liege
Quel est le code de pays à deux lettres pour cette unité ?
[Unknown] : BE
Est-ce CN=Claude Vilvens, OU=Dept. Informatique (clients), O=HEP R Sualem, L=Oup
eye, ST=Liege, C=BE ?
[non] : oui
Spécifiez le mot de passe de la clé pour <ClaudeCli>
    (appuyez sur Entrée s'il s'agit du mot de passe du Keystore) : sexycli
C:\makecert>
```

Le keystore du client est donc créé dans le répertoire courant. On procède de même pour le serveur :

```
C:\makecert>keytool -genkey -alias ClaudeSer -keystore serveur_keystore
Tapez le mot de passe du Keystore : beaugosser
Quels sont vos prénom et nom ?
[Unknown] : Claude Vilvens
Quel est le nom de votre unité organisationnelle ?
[Unknown] : Dept. Informatique (serveurs)
Quelle est le nom de votre organisation ?
[Unknown] : HEP R SUalem
Quel est le nom de votre ville de résidence ?
[Unknown] : Oupeye
Quel est le nom de votre état ou province ?
[Unknown] : Liege
Quel est le code de pays à deux lettres pour cette unité ?
[Unknown] : BE
Est-ce CN=Claude Vilvens, OU=Dept. Informatique (serveurs), O=HEP R SUalem, L=Ou
peye, ST=Liege, C=BE ?
```

---

[non] : oui

Spécifiez le mot de passe de la clé pour <ClaudeSer>  
(appuyez sur Entrée s'il s'agit du mot de passe du Keystore) : sexyser

C:\makecert>

La situation est donc la suivante :

C:\makecert>dir  
Le volume dans le lecteur C s'appelle Programmes  
Le numéro de série du volume est C491-02F0

Répertoire de C:\makecert

```
18/04/2004 18:28  <REP>      .
18/04/2004 18:28  <REP>      ..
18/04/2004 17:36      887 ca.key
18/04/2004 18:22      1.311 client_keystore
18/04/2004 17:53  <REP>      demoCA
18/04/2004 18:28      1.313 serveur_keystore
            3 fichier(s)      3.511 octets
            3 Rép(s) 72.778.395.648 octets libres
```

C:\makecert>

## 2.4 La génération de certificats validés

On l'a dit en son temps, les certificats ainsi générés sont auto-signés. Il nous faut à présent les faire authentifier par un CA – pour nous, ce sera le CA dont nous avons créé la clé privée ci-dessus. Pour obtenir un tel certificat, on se souviendra qu'il faut tout d'abord construire une demande, un **CSR** (Certificate Signing Request). Ceci se fait, pour le client, par :

C:\makecert>**keytool -certreq -alias ClaudeCli -keystore client\_keystore -file clientJsse.csr -keypass** sexycli **-storepass** beaugossecli -v  
Demande de certification enregistrée dans le fichier <clientJsse.csr>  
Soumettre Ó votre CA  
C:\makecert>

Pour rappel, le fichier csr ainsi généré contient

- ♦ la clé publique du demandeur;
- ♦ la signature construite au moyen de la clé privée du demandeur.

Avec OpenSSL, cette requête CSR sera prise en compte pour donner un certificat signé par notre CA au moyen de la commande ca :

C:\makecert>openssl

**OpenSSL> ca** -in clientJssse.csr -out clientJssse.pem -keyfile ca.key

Using configuration from C:\OpenSSL\bin\openssl.cnf

*Loading 'screen' into random state - done*

*Check that the request matches the signature*

*Signature ok*

Certificate Details:

**Serial Number:** 1 (0x1)

**Validity**

Not Before: Apr 20 15:55:06 2004 GMT

Not After : Apr 20 15:55:06 2005 GMT

**Subject:**

countryName = BE

stateOrProvinceName = Liege

organizationName = HEP R Sualem

organizationalUnitName = Dept. Informatique (clients)

commonName = Claude Vilvens

**X509v3 extensions:**

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

E9:08:70:39:09:D5:4A:A3:2F:FF:2E:56:BA:20:3F:F6:D4:9F:B8:50

X509v3 Authority Key Identifier:

keyid:B5:32:DC:66:D6:E9:BD:7D:35:03:1E:CB:00:5C:1B:37:2B:16:50:8B

DirName:/C=BE/ST=Liege/L=Oupeye/O=HEP R Sualem/OU=Dept.

Informatique/CN= VILVENS Claude/emailAddress=civilvens@prov-liege.be

serial:00

Certificate is to be certified until Apr 20 15:55:06 2005 GMT (365 days)

Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y

Write out database with 1 new entries

Data Base Updated

**OpenSSL>**

On remarquera la vérification de la signature de la requête. En cas de succès, le certificat correspondant à la clé publique est signé au moyen de la clé privée de notre CA.

La situation est à présent celle-ci : un fichier pem (ainsi aussi qu'un fichier .rnd correspondant au nombre aléatoire utilisé), ont été créés par l'opération ca d'OpenSSL.

C:\makecert>dir

Le volume dans le lecteur C s'appelle Programmes

Le numéro de série du volume est C491-02F0

Répertoire de C:\makecert

20/04/2004 17:55 <REP> .

20/04/2004 17:55 <REP> ..

```
20/04/2004 17:55      1.024 .rnd
18/04/2004 17:36      887 ca.key
20/04/2004 17:48      980 clientJsse.csr
20/04/2004 17:55      5.330 clientJsse.pem
18/04/2004 18:22      1.311 client_keystore
20/04/2004 17:55  <REP>     demoCA
18/04/2004 18:28      1.313 serveur_keystore
6 fichier(s)          10.845 octets
3 Rép(s)             72.775.192.576 octets libres
```

De même, le répertoire demoCA contient de nouveaux fichiers :

```
C:\makecert>dir demoCA
Le volume dans le lecteur C s'appelle Programmes
Le numéro de série du volume est C491-02F0
```

Répertoire de C:\makecert\demoCA

```
20/04/2004 17:55  <REP>   .
20/04/2004 17:55  <REP>   ..
18/04/2004 17:53      1.367 cacert.pem
18/04/2004 17:06  <REP>     certs
18/04/2004 17:06  <REP>     crl
20/04/2004 17:55      108 index.txt
20/04/2004 17:55      21 index.txt.attr
18/04/2004 17:09      0 index.txt.old
18/04/2004 17:13  <REP>     newcerts
18/04/2004 17:26  <REP>     private
20/04/2004 17:55      3 serial
18/04/2004 17:22      5 serial.old
6 fichier(s)          1.504 octets
6 Rép(s)             72.775.192.576 octets libres
```

Les deux fichiers old sont les sauvetages de fichiers précédents (ils présentent d'ailleurs la date-heure de l'époque). En fait, il faut les supprimer après chaque opération de signature. Enfin, le répertoire newcerts contient lui aussi un nouveau fichier :

```
C:\makecert>dir demoCA\newcerts
Le volume dans le lecteur C s'appelle Programmes
Le numéro de série du volume est C491-02F0
```

Répertoire de C:\makecert\demoCA\newcerts

```
20/04/2004 17:55  <REP>   .
20/04/2004 17:55  <REP>   ..
20/04/2004 17:55      5.330 01.pem
1 fichier(s)          5.330 octets
2 Rép(s)             72.775.192.576 octets libres
```

---

Les autres répertoires (demoCA\private, demoCA\certs et demoCA\crl) sont vides.

Le certificat doit encore être mis en format d'encodage DER afin d'être conforme à la norme X509 :

```
OpenSSL> x509 -in clientJsse.pem -out clientJsse.der -outform DER  
OpenSSL>
```

Pour rappel, les formats les plus courants sont **DER (Definite Encoding Rule)** avec un encodage en notation ASN.1 (extensions **.der**, **.cer**, **.crt**, **.cert**) et **PEM (Privacy Enhanced Mail - extensions .pem, .cer, .crt, .cert)** qui est un format DER encodé en base64 auquel sont ajoutés des en-têtes en ASCII (comme par exemple "-----BEGIN NEW CERTIFICATE REQUEST-----" ou "-----END CERTIFICATE -----"). On peut encore ajouter aux formats courants le format **PKCS12 (Public-Key Cryptography Standards de RSA)**, dont la portée va au-delà des certificats, puisque ce format concerne aussi le stockage des clés privées; Microsoft, qui ne fait rien comme tout le monde, utilise pour ce format l'extension **.pfx** au lieu de l'extension usuelle **.p12**.

On peut constater que le certificat de sécurité standard a été généré :

```
C:\makecert>dir  
Le volume dans le lecteur C s'appelle Programmes  
Le numéro de série du volume est C491-02F0  
  
Répertoire de C:\makecert  
  
20/04/2004 21:20 <REP> .  
20/04/2004 21:20 <REP> ..  
20/04/2004 17:55 1.024 .rnd  
18/04/2004 17:36 887 ca.key  
20/04/2004 17:48 980 clientJsse.csr  
20/04/2004 21:20 1.245 clientJsse.der  
20/04/2004 17:55 5.330 clientJsse.pem  
18/04/2004 18:22 1.311 client_keystore  
20/04/2004 17:55 <REP> demoCA  
18/04/2004 18:28 1.313 serveur_keystore  
    7 fichier(s) 12.090 octets  
    3 Rép(s) 72.757.981.184 octets libres
```

```
C:\makecert>
```

Le certificat, intimement associé à la clé privée qui a servi à signer le CSR, sera stocké dans le keystore au moyen de la commande d'importation keytool –import. Auparavant, il faudra placer dans le keystore le certificat de notre CA – on remarquera les messages d'erreur si l'on est distrait :

```
C:\makecert>keytool -import -v -alias certificauthority -file demaCA\cacert.pem  
-keystore client_keystore -storepass beaugossecli  
erreur keytool : java.io.IOException: Keystore was tampered with, or password was incorrect
```

```
C:\makecert>keytool -import -v -alias certificauthority -file demaCA\cacert.pem  
-keystore client_keystore -storepass beaugossecli  
erreur keytool : java.io.FileNotFoundException: demaCA\cacert.pem (Le chemin d'accès spécifié est introuvable)
```

```
C:\makecert>keytool -import -v -alias certificauthority -file demoCA\cacert.pem  
-keystore client_keystore -storepass beaugossecli
```

Propriétaire : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",  
OU=Dept. Informatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE

Emetteur : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",  
OU=Dept. Informatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE

Numéro de série : 0

Valide du : Sun Apr 18 17:53:20 CEST 2004 au : Tue May 18 17:53:20 CEST 2004

Empreintes de certificat :

MD5 : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94

SHA1: AA:1B:70:6D:00:A2:6E:F0:C5:09:69:2C:D4:13:96:BF:BE:19:B3:43

Faire confiance à ce certificat ? [non] : **oui**

Certificat ajouté au Keystore

[Enregistrement de client\_keystore]

```
C:\makecert>
```

Il reste donc à placer dans le keystore le certificat proprement dit :

```
C:\makecert>keytool -import -v -keystore client_keystore -alias ClaudeCli -file  
clientJsse.der -storepass beaugossecli
```

Spécifiez le mot de passe de la clé pour <ClaudeCli>**sexycli**

Réponse de certificat installée dans le Keystore

[Enregistrement de client\_keystore]

```
C:\makecert>
```

Puisque l'alias correspond à une entrée dans le keystore, le certificat initial, auto-signé, sera ainsi remplacé par une **chaîne de certificats**. A la fin de cette chaîne se trouvent le certificat reçu en réponse du CA et le certificat qui authentifie la clé publique de ce CA. La preuve :

```
C:\makecert>keytool -list -keystore client_keystore
```

Tapez le mot de passe du Keystore : beaugossecli

Type Keystore : jks

Fournisseur Keystore : SUN

Votre Keystore contient 2 entrée(s)

**claudeli**, 20-avr.-2004, **keyEntry**,

Empreinte du certificat (MD5) : 32:FB:9C:B7:31:A9:6E:3D:39:E5:10:00:03:01:4A:93

**certificauthority**, 20-avr.-2004, **trustedCertEntry**,

Empreinte du certificat (MD5) : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94

```
C:\makecert>
```

Bien sûr, il reste à effectuer le même travail pour le magasin à clé destiné au serveur, que nous avons déjà créé mais qui ne contient pas encore le certificat du CA et la chaîne de certification pour sa clé publique :

```
C:\makecert>keytool -list -v -keystore serveur_keystore  
Tapez le mot de passe du Keystore : beaugosser
```

Type Keystore : jks  
Fournisseur Keystore : SUN

Votre Keystore contient 1 entrée(s)

Nom d'alias : claudeser  
Date de création : 18-avr.-2004  
Type d'entrée : **keyEntry**  
Longueur de chaîne du certificat : 1  
Certificat[1]:  
Propriétaire : CN=Claude Vilvens, OU=Dept. Informatique (serveurs), O=HEP R SUalem, L=Oupeye, ST=Liege, C=BE  
Emetteur : CN=Claude Vilvens, OU=Dept. Informatique (serveurs), O=HEP R SUalem, L=Oupeye, ST=Liege, C=BE  
Numéro de série : 4082ac9e  
Valide du : Sun Apr 18 18:28:14 CEST 2004 au : Sat Jul 17 18:28:14 CEST 2004  
Empreintes de certificat :  
MD5 : 18:DF:89:D1:F7:4A:D8:1D:CD:C2:AD:AE:44:1B:19:49  
SHA1: 88:D4:B9:5E:D1:BE:03:C1:A3:CB:2D:A4:3B:06:82:80:0F:29:A2:6C

```
C:\makecert>
```

Tout comme pour le client :

**1) génération du CSR :**

```
C:\makecert>keytool -certreq -alias ClaudeSer -keystore serveur_keystore -file s  
erveurJsse.csr -keypass sexyser -storepass beaugosser -v  
Demande de certification enregistrée dans le fichier <serveurJsse.csr>  
Soumettre à votre CA
```

```
C:\makecert>
```

**2) génération du certificat brut (attention, il faut effacer au préalable les fichiers old créés par les opérations précédents dans le sous-répertoire demoCA) :**

```
OpenSSL> ca -in serveurJsse.csr -out serveurJsse.pem -keyfile ca.key
```

```
C:\makecert>OpenSSL  
OpenSSL> ca -in serveurJsse.csr -out serveurJsse.pem -keyfile ca.key  
Using configuration from C:\OpenSSL\bin\openssl.cnf  
Loading 'screen' into random state - done  
DEBUG[load_index]: unique_subject = "yes"
```

Check that the request matches the signature

Signature ok

***The organizationName field needed to be the same in the CA certificate (HEP R Sualem) and the request (HEP R SUalem)***

error in ca

OpenSSL>

Eh oui, tout est vérifié ;-) Pour rappel, le CA et le propriétaire du certificat doivent appartenir à la même organisation – et ce n'est pas le cas ici (SUalem et Sualem). Recréons un keystore avec le nom d'organisation requis. Ceci étant fait, un nouvel essai sera couronné de succès :

C:\makecert>OpenSSL

**OpenSSL> ca -in serveurJsse.csr -out serveurJsse.pem -keyfile ca.key**

Using configuration from C:\OpenSSL\bin\openssl.cnf

Loading 'screen' into random state - done

DEBUG[load\_index]: unique\_subject = "yes"

Check that the request matches the signature

Signature ok

Certificate Details:

Serial Number: 2 (0x2)

Validity

Not Before: Apr 23 07:37:04 2004 GMT

Not After : Apr 23 07:37:04 2005 GMT

Subject:

countryName = BE

stateOrProvinceName = Liege

organizationName = HEP R Sualem

organizationalUnitName = Dept. Informatique (serveurs)

commonName = Claude Vilvens

X509v3 extensions:

X509v3 Basic Constraints:

CA:FALSE

Netscape Comment:

OpenSSL Generated Certificate

X509v3 Subject Key Identifier:

3B:FF:5C:7F:B6:1D:4B:AD:BE:C7:A7:1F:D9:F9:21:D9:AA:90:37:5A

X509v3 Authority Key Identifier:

keyid:B5:32:DC:66:D6:E9:BD:7D:35:03:1E:CB:00:5C:1B:37:2B:16:50:8B

DirName:/C=BE/ST=Liege/L=Oupeye/O=HEP R Sualem/OU=Dept. Informatique/CN= VILVENS Claude/emailAddress=civilvens@prov-liege.be  
serial:00

Certificate is to be certified until Apr 23 07:37:04 2005 GMT (365 days)

Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y

Write out database with 1 new entries

Data Base Updated

OpenSSL>

**3) génération d'un certificat au format X509 :**

**OpenSSL> x509 -in serveurJsse.pem -out serveurJsse.der -outform DER**

Le résultat :

```
C:\makecert>dir
Le volume dans le lecteur C s'appelle Programmes
Le numéro de série du volume est C491-02F0

Répertoire de C:\makecert

23/04/2004 09:45 <REP> .
23/04/2004 09:45 <REP> ..
23/04/2004 09:37 1.024 .rnd
18/04/2004 17:36 887 ca.key
20/04/2004 17:48 980 clientJsse.csr
20/04/2004 21:20 1.245 clientJsse.der
20/04/2004 17:55 5.330 clientJsse.pem
20/04/2004 21:49 3.707 client_keystore
23/04/2004 09:37 <REP> demoCA
23/04/2004 09:34 980 serveurJsse.csr
23/04/2004 09:45 1.245 serveurJsse.der
23/04/2004 09:37 5.328 serveurJsse.pem
23/04/2004 09:30 1.311 serveur_keystore
    10 fichier(s)   22.037 octets
    3 Rép(s) 72.764.600.320 octets libres
```

```
C:\makecert>
```

**4) importations dans le keystore :**

```
C:\makecert>keytool -import -v -alias certificauthority -file demoCA\cacert.pem
-keystore serveur_keystore -storepass beaugosser
Propriétaire : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",
OU=Dep
t. Informatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE
Fonctionnaire : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",
OU=Dept. I
nformatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE
Numéro de série : 0
Valide du : Sun Apr 18 17:53:20 CEST 2004 au : Tue May 18 17:53:20 CEST 2004
Empreintes de certificat :
    MD5 : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94
    SHA1: AA:1B:70:6D:00:A2:6E:F0:C5:09:69:2C:D4:13:96:BF:BE:19:B3:43
Faire confiance à ce certificat ? [non] : oui
Certificat ajouté au Keystore
[Enregistrement de serveur_keystore]
```

```
C:\makecert>
```

puis

```
C:\makecert>keytool -import -v -keystore serveur_keystore -alias ClaudeSer -file  
serveurJsse.der -storepass beaugosseser  
Propriétaire : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",  
OU=Dep  
t. Informatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE  
Fmetteur : EMAILADDRESS=cvilvens@prov-liege.be, CN=" VILVENS Claude",  
OU=Dept. I  
nformatique, O=HEP R Sualem, L=Oupeye, ST=Liege, C=BE  
Numéro de série : 0  
Valide du : Sun Apr 18 17:53:20 CEST 2004 au : Tue May 18 17:53:20 CEST 2004  
Empreintes de certificat :  
MD5 : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94  
SHA1: AA:1B:70:6D:00:A2:6E:F0:C5:09:69:2C:D4:13:96:BF:BE:19:B3:43  
Faire confiance Ó ce certificat ? [non] : oui  
Certificat ajouté au Keystore  
[Enregistrement de serveur_keystore]  
  
C:\makecert>keytool -import -v -keystore serveur_keystore -alias ClaudeSer -file  
serveurJsse.der -storepass beaugosseser  
Spécifiez le mot de passe de la clé pour <ClaudeSer>sexyser  
Réponse de certificat installée dans le Keystore  
[Enregistrement de serveur_keystore]  
  
C:\makecert>
```

Résultat final :

```
C:\makecert>keytool -list -keystore serveur_keystore  
Tapez le mot de passe du Keystore : beaugosseser  
  
Type Keystore : jks  
Fournisseur Keystore : SUN  
  
Votre Keystore contient 2 entrée(s)
```

```
certificauthority, 23-avr.-2004, trustedCertEntry,  
Empreinte du certificat (MD5) : 0A:77:22:6F:A8:5A:AE:3B:6F:9D:AB:A8:8E:A0:A4:94  
claudeser, 23-avr.-2004, keyEntry,  
Empreinte du certificat (MD5) : EF:F7:FD:30:61:0C:9B:30:78:3F:26:48:2F:F6:21:41
```

```
C:\makecert>
```

Evidemment, tout ceci est riche en possibilités, mais réclame l'utilisation d'outils extérieurs aux applications. Voyons à présent comment réaliser de telles opérations au sein même de nos applications, donc par programmation.

---

### **3. La gestion des certificats et keystores par programmation**

Grâce à la librairie Bouncy Castle, il est aussi possible de construire un certificat, auto-signé ou non, par programmation pure.

La première démarche consiste à d'abord créer une simple classe InfosCertificat dont le rôle sera de contenir les informations de base d'un certificat :

#### **InfosCertificat.java**

```
package certificatsgenerationpkcs12;

public class InfosCertificat
{
    private String nom;
    private String uniteOrganisation;
    private String organisation;
    private String localite;
    private String etat;
    private String codePays;
    private String adresseEMail;
    private int nbreJoursValidite;

    public InfosCertificat (String n,String u,String o, String l, String e,String c, String a,
                           int njv)
    {   nom = n; uniteOrganisation = u; organisation = o; localite = l; etat = e; codePays = c;
        adresseEMail = a; nbreJoursValidite = njv;   }

    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }
    public String getUniteOrganisation() { return uniteOrganisation; }
    public void setUniteOrganisation(String uniteOrganisation)
    {   this.uniteOrganisation = uniteOrganisation; }
    public String getOrganisation() { return organisation; }
    public void setOrganisation(String organisation) { this.organisation = organisation; }
    public String getLocalite() { return localite; }
    public void setLocalite(String localite) { this.localite = localite; }
    public String getEtat() { return etat; }
    public void setEtat(String etat) { this.etat = etat; }
    public String getCodePays() { return codePays; }
    public void setCodePays(String codePays) { this.codePays = codePays; }
    public String getAdresseEMail() { return adresseEMail; }
    public void setAdresseEMail(String adresseEMail) { this.adresseEMail = adresseEMail; }
    public int getnbreJoursValidite() {return nbreJoursValidite; }
    public void setnbreJoursValidite(int nbreJoursValidite)
    {   this.nbreJoursValidite = nbreJoursValidite; }
    public boolean infosIncompltes()
    {
        return nom==null || uniteOrganisation==null || organisation==null || localite==null
               ||etat==null || codePays==null || adresseEMail==null || nbreJoursValidite <=0;
    }
}
```

Mais le principal du travail sera pris en charge par une classe KeyCertificatePair dont le rôle, comme son nom l'indique, sera de créer un certificat avec les clés qui sont en rapport avec lui (fournies ou générées). Dans un tel contexte, ses trois variables membres sont bien logiquement le certificat (instance de **X509Certificate**), la paire de clés privée-publique associée (instance de **KeyPair**) et le keystore qui les héberge tous (instance de **KeyStore**). Outre un constructeur trivial, la classe possède un second constructeur plus intéressant qui construit un certificat auto-signé à partir d'une paire de clés fournie :

```
public KeyCertificatePair (InfosCertificat infos, KeyPair paireCles, boolean verbose)
// Version 1 : Construction d'un certificat auto-signé à partir d'une paire de clés fournie
```

Ce constructeur commence par créer un objet instance de la classe **X509Principal** (package org.bouncycastle.jce – elle hérite de **X509Name**) dont le rôle consiste à mémoriser les informations du certificat dans une hashtable – l'un de ses constructeurs (le polymorphisme est de mise) est donc :

```
public X509Principal (Hashtable attributes)
```

et il suffit pour l'utiliser de remplir une Hashtable avec les valeurs de l'objet InfosCertificat, les clés correspondantes étant les constantes statiques de X509Principal comme CN, O, OU, etc. Une méthode privée réalise ce travail :

```
static private X509Principal createPrincipal (InfosCertificat infos)
{
    Hashtable att = new Hashtable();
    if ( infos.getNom() != null)att.put(X509Principal.CN, infos.getNom());
    if (infos.getUniteOrganisation() != null) att.put(X509Principal.OU,
        infos.getUniteOrganisation());
    ...
    return new X509Principal(att);
}
```

Le certificat proprement dit, auto-signé, est construit au moyen d'un objet **X509V3CertificateGenerator** (package org.bouncycastle.x509) que l'on instancie avec son constructeur par défaut. Cet objet

- ♦ enregistre les données du propriétaire et du signataire du certificat au moyen des méthodes :

```
public void setSubjectDN (X509Name issuer)
public void setIssuerDN(X509Name issuer)
```

- ♦ crée les dates de validité au moyen des méthodes :

```
public void setNotBefore(Date date)
public void setNotAfter(Date date)
```

- ◆ enregistre la clé publique qui fait l'objet du certificat par

```
public void setPublicKey(PublicKey key) throws java.lang.IllegalArgumentException
```

- ◆ crée un numéro de série aléatoire avec

```
public void setSerialNumber(BigInteger serialNumber)
```

en créant un entier à partir de l'heure courante au moyen de la méthode de System

```
public static long currentTimeMillis()
```

pour finalement en arriver à générer le certificat en le signant avec la clé privée associée au moyen de

```
public X509Certificate generate(PrivateKey key, String provider)
    throws java.security.cert.CertificateEncodingException, java.lang.IllegalStateException,
    java.security.NoSuchProviderException, java.security.NoSuchAlgorithmException,
    java.security.SignatureException, java.security.InvalidKeyException
```

Mais le gros intérêt de notre classe sera de fournir des méthodes factories capables de créer un certificat contenant une clé publique et signé au moyen d'une clé privée, générée ou fournie par un autre certificat :

```
◆ static public KeyCertificatePair getInstance (InfosCertificat infos, KeyPair paireCles,
boolean verbose)
// Version 1 : Construction d'un certificat auto-signé à partir d'une paire de clés fournie
◆ static public KeyCertificatePair getInstance(InfosCertificat infos, boolean verbose)
// Version 2 : Génération d'une paire de clés et construction d'un certificat auto-signé
◆ static public KeyCertificatePair getInstance(InfosCertificat infos, KeyCertificatePair
cleCertifPaire, KeyCertificatePair cleCertifPaireCA, boolean verbose)
// Version 3 : Construction d'un certificat à partir d'un KeyCertificatePair auto-signé
// et signature avec la clé privée associée à un autre certificat
◆ static public KeyCertificatePair getInstance(InfosCertificat infos, KeyCertificatePair
cleCertifPaire, boolean verbose)
// Version 4 : Construction d'un certificat pour une paire de clés générée et signature avec la
clé privée associée à un autre certificat
```

**1)** La version 1 ne fait évidemment rien d'autre que d'appeler le constructeur décrit ci-dessus.

**2)** La version 2 appelle la version 1 en passant comme paire de clés une paire générée au moyen de la méthode generePaireDeCles() (selon le procédé bien connu utilisant un KeyPairGenerator parameter pour RSA).

**3)** La version 3 devient plus intéressante, puisqu'elle va créer un véritable certificat signé par un CA. Dans les paramètres, on distingue un KeyCertificatePair dont la clé publique est l'objet du certificat et un autre KeyCertificatePair dont la clé privée servira à la signature de ce certificat. Comme dans le constructeur décrit ci-dessus, on commence par créer un objet X509Principal pour le propriétaire du certificat (associé à la première paire de clés). Le certificat sera à nouveau construit au moyen d'un objet **X509V3CertificateGenerator** qui va enregistrer les données du propriétaire (méthode setSubjectDN()) et utiliser comme signataire

---

le "Subject" qui se trouve associé au deuxième KeyCertificatePair : ceci peut s'obtenir aisément au moyen de la méthode

```
getSubjectX509Principal(X509Certificate cert)
    throws java.security.cert.CertificateEncodingException
```

de la classe **PrincipalUtil** (toujours du package org.bouncycastle.jce). Il ne reste plus qu'à signer.

4) La version 4 peut évidemment se construire de manière similaire (to do ;-) ...).

Les méthodes de gestion des keystores sont assez immédiates et utilisent des techniques connues – nous n'insisterons donc pas. En résumé :

### KeyCertificatePair.java

```
package certificatsgenerationpkcs12;

/**
 * @author Vilvens
 */
import java.io.*;
import java.math.BigInteger;
import java.security.*;
import java.security.cert.*;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.bouncycastle.jce.PrincipalUtil;
import org.bouncycastle.jce.X509Principal;
import org.bouncycastle.x509.X509V3CertificateGenerator;

public class KeyCertificatePair
{
    private X509Certificate certificat;
    private KeyPair clescertificat;
    private KeyStore keystore;
    static private final String TYPE_SIGNATURE = "SHA1withRSA";
    static private String codeProvider = "BC";

    public KeyCertificatePair (X509Certificate c, KeyPair kp)
    {
        certificat =c;
        clescertificat = kp;
        keystore = null;
    }
}
```

```

public KeyCertificatePair (InfosCertificat infos, KeyPair paireCles, boolean verbose)
// Version 1 : Construction d'un certificat auto-signé à partir d'une paire de clés fournie
{
    clescertificat = paireCles;
    if (verbose) System.out.println("*** Cr ation d'un certificat auto-sign  pour " +
        infos.getNom());
    if (infos.infosIncompl es())
        System.out.println("x Informations incompl es pour cr er un certificat valide");
    if (verbose) System.out.println("- Cr ation du principal");

X509Principal p = createPrincipal(infos);

if (verbose) System.out.println("- Cr ation et initilisation du g n rateur");
X509V3CertificateGenerator certGen = new X509V3CertificateGenerator();
certGen.setNotBefore(new Date(System.currentTimeMillis()));
certGen.setNotAfter(new Date(System.currentTimeMillis() + ((long)
    infos.getnbreJoursValidite() * 24 * 60 * 60 * 1000)));
certGen.setSubjectDN(p);
certGen.setIssuerDN(p);
certGen.setPublicKey(paireCles.getPublic());
paireCles.getPrivate();
certGen.setSignatureAlgorithm(TYPE_SIGNATURE);
certGen.setSerialNumber(genX509SerialNumber());

if (verbose) System.out.println("- G n ration du certificat");
try
{
    certificat = certGen.generate(clescertificat.getPrivate(), codeProvider);
}
catch (GeneralSecurityException ex)
{ System.err.println("Erreur: "+ex.toString()); }

if (verbose) afficheCertificat();
}

// Les factories
static public KeyCertificatePair getInstance (InfosCertificat infos, KeyPair paireCles,
    boolean verbose)
// Version 1 : Construction d'un certificat auto-sign  à partir d'une paire de cl s fournie
{
    return new KeyCertificatePair (infos, paireCles, verbose);
}

static public KeyCertificatePair getInstance(InfosCertificat infos, boolean verbose)
// Version 2 : G n ration d'une paire de cl s et construction d'un certificat auto-sign 
{
    return getInstance (infos, generePaireDeCles(verbose), verbose);
}

```

```

static public KeyCertificatePair getInstance(InfosCertificat infos, KeyCertificatePair
    cleCertifPaire, KeyCertificatePair cleCertifPaireCA, boolean verbose)
// Version 3 : Construction d'un certificat à partir d'un KeyCertificatePair auto-signé
// et signature avec la clé privée associée à un autre certificat
{
    if (verbose) System.out.println("*** Crédation d'un certificat signé par " +
        cleCertifPaireCA.getcertificat().getSubjectDN().getName() + " pour " +
        infos.getNom());
    if (infos.infosIncompltes())
        System.out.println("x Informations incompltes pour créer un certificat valide");
    if (verbose) System.out.println("- Crédation du principal");

X509Principal p = createPrincipal(infos);

    if (verbose) System.out.println("- Crédation et initialisation du génératuer");
X509V3CertificateGenerator certGen = new X509V3CertificateGenerator();
certGen.setNotBefore(new Date(System.currentTimeMillis()));
certGen.setNotAfter(new Date(System.currentTimeMillis() + ((long)
    infos.getnbreJoursValidite()* 24 * 60 * 60 * 1000)));
certGen.setSubjectDN(p);
try
{
    certGen.setIssuerDN(PrincipalUtil.getSubjectX509Principal(cleCertifPaireCA.
        getcertificat()));
}
catch (CertificateEncodingException ex)
{ Logger.getLogger(KeyCertificatePair.class.getName()).log(Level.SEVERE, null, ex); }
PublicKey clePublique = cleCertifPaire.getcertificat().getPublicKey();
certGen.setPublicKey(clePublique);
certGen.setSignatureAlgorithm(TYPE_SIGNATURE);
certGen.setSerialNumber(genX509SerialNumber());
if (verbose) System.out.println("- Génération du certificat");

X509Certificate certificatGenere = null;
try
{
    certificatGenere = certGen.generate(cleCertifPaireCA.getClePriveeAssociee(),
        codeProvider);
}
catch (GeneralSecurityException ex)
{ System.err.println("Erreur: "+ex.toString()); }

KeyCertificatePair kp = new KeyCertificatePair (certificatGenere,cleCertifPaire.
    getClécertificat());
if (verbose) kp.afficheCertificat();
return kp;
}

```

```

static public KeyCertificatePair getInstance(InfosCertificat infos,
    KeyCertificatePair cleCertifPaire, boolean verbose)
// Version 4 : Construction d'un certificat pour une paire de clés générée et signature
// avec la clé privée
// associée à un autre certificat
{
    // TODO ;)
    return null;
}

public X509Certificate getcertificat() { return certificat; }
public PrivateKey getClePriveeAssociee() { return getClescertificat().getPrivate(); }
public KeyPair getClescertificat() { return clescertificat; }

private void afficheCertificat()
{
    System.out.println("- Contenu du certificat");
    System.out.println("Type de certificat : " + certificat.getType());
    System.out.println("Nom du propriétaire du certificat : " +
        certificat.getSubjectDN().getName());
    PublicKey cléPublique = certificat.getPublicKey();
    System.out.println("... sa clé publique : " + cléPublique.toString());
    System.out.println("... la classe instanciée par celle-ci : " +
        cléPublique.getClass().getName());
    System.out.println("Dates limites de validité : [" + certificat.getNotBefore() + " - " +
        certificat.getNotAfter() + "]");
    System.out.println("Signataire du certificat : " + certificat.getIssuerDN().getName());
    System.out.println("Algo de signature : " + certificat.getSigAlgName());
    System.out.println("Signature : " + certificat.getSignature());
    System.out.println("Numéro de série : " + certificat.getSerialNumber());
}

private void getKeystore(String nomKeystore, boolean newKeystore, String motDePasse)
{
    keystore = null;
    char[] pwdKeystore = motDePasse.toCharArray();
    try
    {
        keystore = KeyStore.getInstance("PKCS12", codeProvider);
        if (newKeystore)keystore.load(null, pwdKeystore);
        else keystore.load( new FileInputStream(
            new File(System.getProperty("user.dir")).getParent().toString()
            + System.getProperty("file.separator") + nomKeystore), pwdKeystore);
    }
    catch (Exception ex) {
        Logger.getLogger(KeyCertificatePair.class.getName()).log(Level.SEVERE,
            ll, ex);
    }
}

```

```

private void updateKeystore(String nomKeystore, String motDePasse)
{
    try
    {
        char[] pwdKeystore = motDePasse.toCharArray();
        keystore.store(new FileOutputStream(new
            File(System.getProperty("user.dir")).getParent().toString() +
            System.getProperty("file.separator") + nomKeystore), pwdKeystore);
    }
    catch (Exception ex)
    { Logger.getLogger(KeyCertificatePair.class.getName()).log(Level.SEVERE,null,ex); }
}

public void ajouteCertificatKeystore(String nomKeystore, String alias,
    boolean newKeystore, String motDePasse)
{
    System.out.println("- Keystore : "+ nomKeystore);
    try
    {
        getKeystore(nomKeystore, newKeystore, motDePasse);
        if (keystore != null)
        {
            keystore.setCertificateEntry(alias, certificat);
            updateKeystore(nomKeystore, motDePasse);
        }
    }
    catch (KeyStoreException ex)
    {Logger.getLogger(KeyCertificatePair.class.getName()).log(Level.SEVERE, null, ex); }

public void ajouteCertificatEtCleKeystore(String nomKeystore, String alias,
    String password, X509Certificate certCA, boolean newKeystore)
{
    try
    {
        getKeystore(nomKeystore, newKeystore, password);
        X509Certificate[] cert = null;
        cert = new X509Certificate[1];
        cert[0] = certCA;
        if (keystore != null)
        {
            keystore.setKeyEntry(alias, this.getCléPriveeAssociee(), password.toCharArray(),
                cert);
            updateKeystore(nomKeystore, password);
        }
    }
    catch (KeyStoreException ex)
    { Logger.getLogger(KeyCertificatePair.class.getName()).log(Level.SEVERE, null, ex); }
}

```

```

static public KeyPair generePaireDeCles(boolean verbose)
{
    if (verbose) System.out.println("- Génération des clés du certificat");
    KeyPair paireClesGeneree = null;
    try
    {
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("RSA");
        SecureRandom rand = SecureRandom.getInstance("SHA1PRNG");
        keyPairGen.initialize(2048, rand);
        paireClesGeneree = keyPairGen.generateKeyPair();
    }
    catch(Exception ex)
    { System.err.println("Erreur: "+ex.toString()); }
    return paireClesGeneree;
}

static private BigInteger genX509SerialNumber()
{
    return new BigInteger(Long.toString(System.currentTimeMillis())/// 1000));
}

static private X509Principal createPrincipal (InfosCertificat infos)
{
    Hashtable att = new Hashtable();

    if (infos.getNom() != null)att.put(X509Principal.CN, infos.getNom());
    if (infos.getUniteOrganisation() != null) att.put(X509Principal.OU,
        infos.getUniteOrganisation());
    if (infos.getOrganisation() != null) att.put(X509Principal.O, infos.getOrganisation());
    if (infos.getLocalite() != null)att.put(X509Principal.L, infos.getLocalite());
    if (infos.getEtat() != null)att.put(X509Principal.ST, infos.getEtat());
    if (infos.getCodePays() != null) att.put(X509Principal.C, infos.getCodePays());
    if (infos.getAdresseEMail() != null)att.put(X509Principal.E, infos.getAdresseEMail());

    return new X509Principal(att);
}
}

```

On peut à présent executer un programme de test :

### TestGestionCertificats.java

```
package certificatsgenerationpkcs12;
```

```
/**  
 * @author Vilvens  
 */
```

```
public class TestGestionCertificats  
{
```

```
public static void main(String[] args)
{
    InfosCertificat icCA = new InfosCertificat("HEPL-Root", "HEPL", "HEPL",
                                                "Seraing", "Liege", "BE", "autorite.dieu@ciel.be", 100);
    KeyCertificatePair gcCA = KeyCertificatePair.getInstance(icCA,true);
    InfosCertificat icVil = new InfosCertificat("Vilvens", "HEPL", "HEPL",
                                                "Oupeye", "Liege", "BE", "vilvens.claude@skynet.be", 100);
    KeyCertificatePair gcVil = KeyCertificatePair.getInstance(icVil,true);
    KeyCertificatePair gc = KeyCertificatePair.getInstance(icVil,gcVil,gcCA,true);
    gc.ajouteCertificatEtCleKeystore("keystore-simpleminds.p12", "Claude", "pwdpwd",
                                     gc.getcertificat(), true);
    gcCA.ajouteCertificatEtCleKeystore("keystore-simpleminds.p12", "HEPL-root",
                                       "pwdpwd", gcCA.getcertificat(), false);
    gcCA.ajouteCertificatKeystore("keystore-simpleminds.p12", "HEPL", false,
                                  "pwdpwd");
}
}
```

Ce qui donne par exemple :

- Génération des clés du certificat

**\*\*\* Crédation d'un certificat auto-signé pour HEPL-Root**

- Création du principal

- Création et initialisation du générateur

- Génération du certificat

- Contenu du certificat

Type de certificat : X.509

Nom du propriétaire du certificat :

O=HEPL,E=autorite.dieu@ciel.be,ST=Liege,L=Seraing,C=BE,CN=HEPL-Root,OU=HEPL

... sa clé publique : RSA Public Key

modulus:

87bec110deeed7d93b42b538e321d2cea5b2f6cc2895e0d0372e2057087342fe3e2ce7a130b573  
c0a0af169bcd0fe24da6098032165fef19ea50ee3d4a368cd8ed25a4f12cc9955820edb8477083  
e3483d7ec8787b47ac9119b5550898611264a3f5c48f0390480fdfc32818a73fcda5dc6e991ba  
9fcf7d88deae7f91e77584b72a17934bb2faf886ccf9ac7937fcf44400bdd88400645b5400c4279  
7206c0c97d526c7ba738cd2f0bf06e38f3f7926da9636e477c85bb0fffcfead84963a4ef048cc8e7  
42c93d677589a2cf89050b4f1f4992fea032ef3c217433d436dc571290cd61d6047a6cc36f41a87  
7dc7fd91c659d52a40b3f920e283e868ddaae03

public exponent: 10001

... la classe instanciée par celle-ci : org.bouncycastle.jce.provider.JCERSAPublicKey

Dates limites de validité : [Sun Aug 16 14:15:09 CEST 2009 - Tue Nov 24 13:15:09 CET  
2009]

Signataire du certificat :

O=HEPL,E=autorite.dieu@ciel.be,ST=Liege,L=Seraing,C=BE,CN=HEPL-Root,OU=HEPL

Algo de signature : SHA1WithRSAEncryption

Signature : [B@30c221

Numéro de série : 1250424909343

- Génération des clés du certificat

**\*\*\* Crédation d'un certificat auto-signé pour Vilvens**

- Création du principal
- Création et initialisation du générateur
- Génération du certificat
- Contenu du certificat

Type de certificat : X.509

Nom du propriétaire du certificat :

O=HEPL,E=vilvens.claude@skynet.be,ST=Liege,L=Oupeye,C=BE,CN=Vilvens,OU=HEPL

... sa clé publique : RSA Public Key

modulus:

a34eba4dd62d364f31e615479b127e3c0d9b851bef98ab3230c2e7d5970104b31b30ae97febadb  
a8b72ea520526d77d5754a8f0f7e4a7c42770235d4e4023cf2a612f0ec2a0ac7908e2cb9f7adc31  
3072a2531275269b43e0595b8c83df0fb142a642ac89fd6c3a3f508c2ff173bf6e4d1a90945cc69  
bb15c861de03cf1760b1961c48f8b7c09cd4e4934cb726ee66f34a876371db9c0e7074dfb561ea  
9273f9275d2c970220243e54e340bd2861990e29a15e265ece029b7e520647390efc07dada305  
b89d8876a18c018d579a8876d8bab697b0bbae4179c6d2a5e5765a9dd97442859aa152dd6a897  
f78cbf92b382379108093b74c0103c946c57a63907d1

public exponent: 10001

... la classe instanciée par celle-ci : org.bouncycastle.jce.provider.JCERSAPublicKey

Dates limites de validité : [Sun Aug 16 14:15:10 CEST 2009 - Tue Nov 24 13:15:10 CET  
2009]

Signataire du certificat :

O=HEPL,E=vilvens.claude@skynet.be,ST=Liege,L=Oupeye,C=BE,CN=Vilvens,OU=HEPL

Algo de signature : SHA1WithRSAEncryption

Signature : [B@1f33675

Numéro de série : 1250424910968

#### \*\*\* Crédit d'un certificat signé par

**O=HEPL,E=autorite.dieu@ciel.be,ST=Liege,L=Seraing,C=BE,CN=HEPL-**

**Root,OU=HEPL pour Vilvens**

- Création du principal
- Création et initialisation du générateur
- Génération du certificat
- Contenu du certificat

Type de certificat : X.509

Nom du propriétaire du certificat :

O=HEPL,E=vilvens.claude@skynet.be,ST=Liege,L=Oupeye,C=BE,CN=Vilvens,OU=HEPL

... sa clé publique : RSA Public Key

modulus:

a34eba4dd62d364f31e615479b127e3c0d9b851bef98ab3230c2e7d5970104b31b30ae97febadb  
a8b72ea520526d77d5754a8f0f7e4a7c42770235d4e4023cf2a612f0ec2a0ac7908e2cb9f7adc31  
3072a2531275269b43e0595b8c83df0fb142a642ac89fd6c3a3f508c2ff173bf6e4d1a90945cc69  
bb15c861de03cf1760b1961c48f8b7c09cd4e4934cb726ee66f34a876371db9c0e7074dfb561ea  
9273f9275d2c970220243e54e340bd2861990e29a15e265ece029b7e520647390efc07dada305  
b89d8876a18c018d579a8876d8bab697b0bbae4179c6d2a5e5765a9dd97442859aa152dd6a897  
f78cbf92b382379108093b74c0103c946c57a63907d1

public exponent: 10001

... la classe instanciée par celle-ci : org.bouncycastle.jce.provider.JCERSAPublicKey

Dates limites de validité : [Sun Aug 16 14:15:11 CEST 2009 - Tue Nov 24 13:15:11 CET  
2009]

---

Signataire du certificat :

O=HEPL,E=autorite.dieu@ciel.be,ST=Liege,L=Seraing,C=BE,CN=HEPL-Root,OU=HEPL

Algo de signature : SHA1WithRSAEncryption

Signature : [B@1befab0

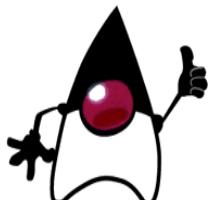
Numero de série : 1250424911031

- Keystore : keystore-simpleminds.p12

Une petite vérification par Keytool IUI peut nous convaincre que tout s'est bien passé :

**Private Key (keypair) & Trusted Certificate Entries:**

Alias	Entry	Valid Date ?	Self-Signed ?	Trusted C.A. ?	Key Size	Cert. Type	Cert. Sig. Algo.	Modified Date
Claude		✓	-	-	2048 bits	X.509	SHA1WithRSAE...	16-août-2009
HEPL		✓	✓	-	2048 bits	X.509	SHA1WithRSAE...	16-août-2009
HEPL-root		✓	✓	-	2048 bits	X.509	SHA1WithRSAE...	16-août-2009



**FIN  
provisoire ?**

Bien sûr aussi qu'il y a encore des sujets passionnants comme les deux autres plate-formes de Java : J2ME et Java Card, sans parler des serveurs d'applications – pour ne citer qu'eux !

Sans doute : mais à chaque cours suffit sa peine - bye !

## Ouvrages consultés

### Ouvrages imprimés et électroniques

**Allamaraju, S, et al.** Professional Java E-Commerce. Birmingham, United Kingdom. Wrox Press Ltd. 2001.

**Belflamme, M.** Etude et programmation de librairies spécifiques SNMP en C. Seraing, Belgique. TFE In.Pr.E.S. 2004.

**Benitez, E.** Etude de nouvelles technologies intégrées dans Java Software Development Kit 1.4 (JDBC & JSSE). Seraing, Belgique. TFE In.Pr.E.S. 2004.

**Dal Pont, M.** Etude des APIs d'extension Java : Les APIs JavaMail et le serveur de messagerie Sendmail. TFE In.Pr.E.S. 2002.

**Donnay, G.** Etude et programmation de librairies spécifiques SNMP en Java. Seraing, Belgique. TFE In.Pr.E.S. 2004.

**Flanagan, D.** Java in a nutshell. Sebastopol, California, U.S.A. O'Reilly and Associates, Inc. 1999.

**Flanagan, D., Farley, J., Crawford, W. & Magnusson, K.** Java Enterprise in a nutshell. Sebastopol, California, U.S.A. O'Reilly and Associates, Inc. 1999.

**Herbiet, L.** Introduction à la cryptographie (cours). Liège, Belgique. Université de Liège. 2003.

**Jeunesse, F.** Etude synthétique de techniques de sécurité utilisées par les applications réseaux. Seraing, Belgique. TFE Haute Ecole de la Province de Liège (In.Pr.E.S.). 2009.

**Leburton, C.** Utilisation des technologies Java et XML dans le cadre d'un projet européen : Onto-Logging. TFE In.Pr.E.S. 2002.

**Lennertz, S.** Système de messagerie électronique sécurisé basé sur le serveur SMTP Postfix avec interface Web de configuration. TFE In.Pr.E.S. 2003.

**Licata, J.** Etude et conception d'une architecture pour le commerce électronique. TFE U.L.G. 2004.

**Loshin, P.** Essential Email Standards. NewYork, U.S.A. John Wiley & Son, Inc. 2000.

**Simon, G.** Construction d'un prototype de site d'e-commerce à technologie Java pour la commercialisation de publications sur Internet. TFE In.Pr.E.S. 2002.

**Vilvens, C.** Java (I) : Programmation de base. Seraing, Belgique. A.S.B.L. DEFI. 2011.

**Vilvens, C.** Java (II) : Programmation avancée des applications classiques et cryptographie. Seraing, Belgique. A.S.B.L. DEFI. 2011.

---

**Vilvens, C.** Java (III) : Programmation des applications WEB. Seraing, Belgique. A.S.B.L. DEFI. 2011.

**Vilvens, C.** Programmation TCP/IP. Seraing, Belgique. A.S.B.L. DEFI. 2009.

**White, S., Fischer, M., Catell, R., Hamilton, G. & Hapner, M.** JDBC API Tutorial and Reference, Second Edition / The Java Series. Reading, Massachussettes, U.S.A. Addison-Wesley Publishing Company. 1999.

**Wood, D.** Programming Internet Email. Sebastopol, California, U.S.A. O'Reilly and Associates, Inc. 1999.

\*\*\*\*\*

### Sites Internet

<http://java.sun.com/>

avec en particulier :

<http://developer.java.sun.com/>

<http://developer.java.sun.com/developer/onlineTraining>

<http://developers.sun.com/mobility/midp/articles/deploy/>

<http://www.bejug.org/>

Site du Belgian Java User Group



## Annexe : Les codes de retour des commandes FTP

(Postel & Reynolds - **RFC 959** - October 1985)

<b><u>Connection Establishment</u></b>	PASV 227 500, 501, 502, 421, 530
120	
220	
220	
421	
<b><u>Login</u></b>	MODE 200 500, 501, 504, 421, 530
USER	TYPE 200 500, 501, 504, 421, 530
230	
530	
500, 501, 421	STRU 200 500, 501, 504, 421, 530
331, 332	
PASS	
230	
202	
530	
500, 501, 503, 421	
332	
ACCT	
230	
202	
530	
500, 501, 503, 421	
CWD	
250	
500, 501, 502, 421, 530, 550	
CDUP	
200	
500, 501, 502, 421, 530, 550	
SMNT	
202, 250	
500, 501, 502, 421, 530, 550	
<b><u>Logout</u></b>	
REIN	
120	
220	
220	
421	
500, 502	
QUIT	
221	
500	
<b><u>Transfer parameters</u></b>	
PORT	
200	
500, 501, 421, 530	
	PASV 227 500, 501, 502, 421, 530
	MODE 200 500, 501, 504, 421, 530
	TYPE 200 500, 501, 504, 421, 530
	STRU 200 500, 501, 504, 421, 530
	ALLO 200 202 500, 501, 504, 421, 530
	REST 500, 501, 502, 421, 530 350
	STOR 125, 150 (110) 226, 250 425, 426, 451, 551, 552 532, 450, 452, 553 500, 501, 421, 530
	STOU 125, 150 (110) 226, 250 425, 426, 451, 551, 552 532, 450, 452, 553 500, 501, 421, 530
	RETR 125, 150 (110) 226, 250 425, 426, 451 450, 550 500, 501, 421, 530
	LIST 125, 150 226, 250 425, 426, 451

450	257
500, 501, 502, 421, 530	500, 501, 502, 421, 530, 550
NLST	PWD
125, 150	257
226, 250	500, 501, 502, 421, 550
425, 426, 451	ABOR
450	225, 226
500, 501, 502, 421, 530	500, 501, 502, 421
APPE	<b>Informational commands</b>
125, 150	SYST
(110)	215
226, 250	500, 501, 502, 421
425, 426, 451, 551, 552	STAT
532, 450, 550, 452, 553	211, 212, 213
500, 501, 502, 421, 530	450
RNFR	500, 501, 502, 421, 530
450, 550	HELP
500, 501, 502, 421, 530	211, 214
350	500, 501, 502, 421
RNTO	<b>Miscellaneous commands</b>
250	SITE
532, 553	200
500, 501, 502, 503, 421, 530	202
DELE	500, 501, 530
250	NOOP
450, 550	200
500, 501, 502, 421, 530	500 421
RMD	
250	
500, 501, 502, 421, 530, 550	
MKD	