
TCP java.net

Patron MVC & Procuration

jean-michel Douin, douin au cnam point fr
version : 29 Janvier 2008

Notes de cours

Sommaire pour les patrons

- **Une illustration du modèle de conception MVC**
 - Un Chat sur le web
- **Proxy :**
 - le retour ...

Les Patrons

- **Classification habituelle**

- **Créateurs**

- **Abstract Factory, Builder, Factory Method Prototype Singleton**

- **Structurels**

- **Adapter Bridge Composite Decorator Facade Flyweight Proxy**

- **Comportementaux**

- **Chain of Responsibility. Command Interpreter Iterator**

- **Mediator Memento Observer State**

- **Strategy Template Method Visitor**

Sommaire

- **TCP/IP** (*uniquement, (UDP : un autre support)*)
 - Serveur et clients, `java.net.ServerSocket`, `java.net.Socket`
 - **Architectures respectives**
 - Protocole « java », (Serializable)
 - Protocole « maison », (propriétaire)
 - Protocole HTTP
- **Serveur Web**
 - Usage du patron « PoolThread »
 - Applette
- **MVC distribué ?**
 - une esquisse
- **Annexes**
 - Java Web Start
 - Junithttp
 - ..

Bibliographie utilisée

- Design Patterns, catalogue de modèles de conception réutilisables de Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides [Gof95]
International thomson publishing France

Java et les réseaux

<http://java.sun.com/docs/books/tutorial/networking/>
<http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/>

En savoir plus

Doug Lea : <http://gee.cs.oswego.edu/dl/cpjslides/nio.pdf>

java.nio

<http://javanio.info/>

Architecture of a Highly Scalable NIO-Based Server de G.Roth

<http://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html>

Pré-requis

- **Notion**
 - TCP/IP
- **Notion des patrons**
 - Adaptateur
 - Procuration
 - Observateur & MVC

Contexte, vocable

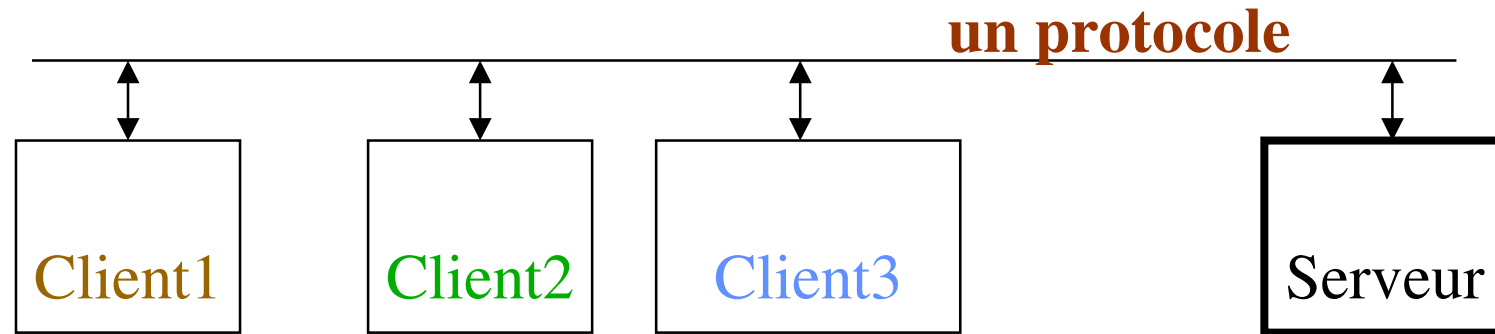
- **Appels distants en mode TCP/IP**
 - Point à point avec accusé de réception
 - En détail ici <http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/tcp.html>
 - TCP comme telnet, ftp, http, ...

- **URL *Uniform Resource Locator*** *une adresse sur internet*
 - <http://jfod.cnam.fr>

 - http le protocole
 - [//jfod.cnam.fr](http://jfod.cnam.fr) le nom de la ressource

 - <http://jfod.cnam.fr:8999/ds2438/mesures.html>

Exemples clients / serveurs



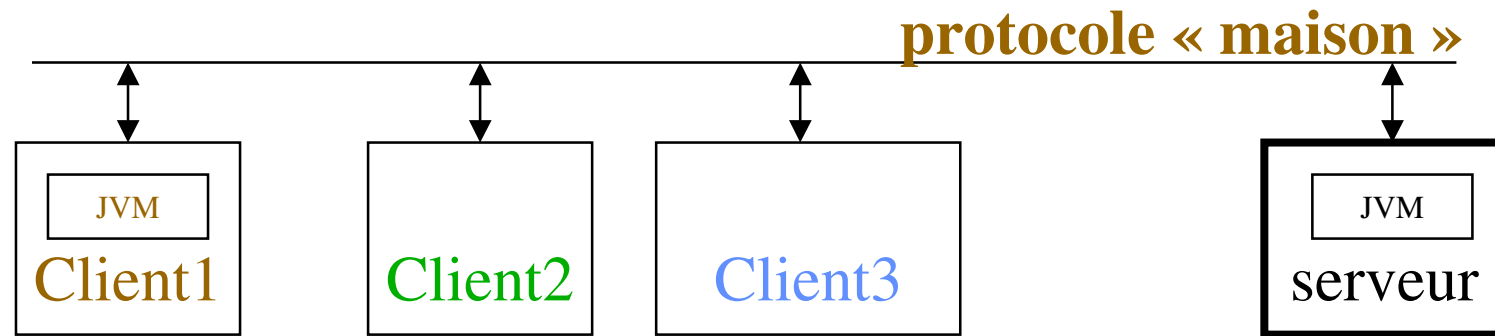
1. Le client s'adresse au serveur

- Établit une connexion, à son initiative

2. Le serveur satisfait ses clients

- Mode synchrone, analogue à l'appel d'une méthode locale

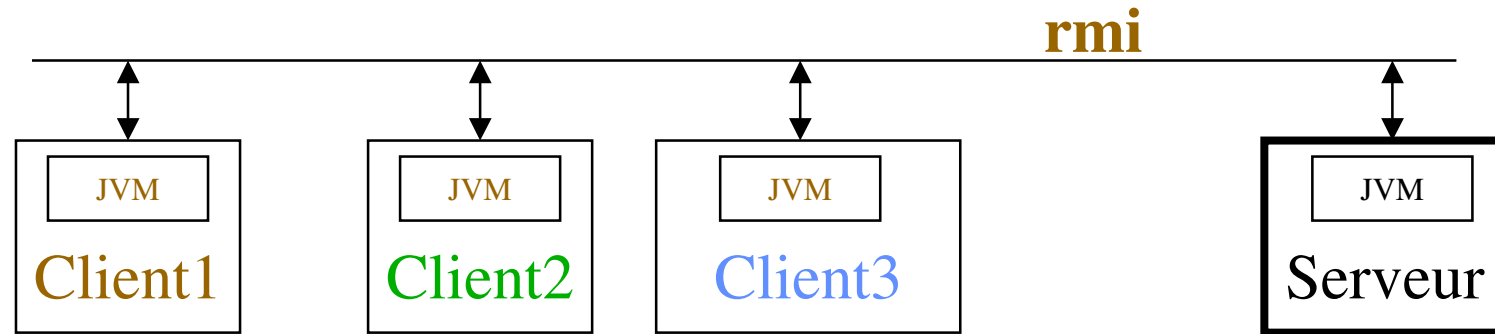
Appels distants protocole « maison » propriétaire



- **Le contexte**

- Client Java, ou autres
- Serveur en java ou autre
- maison : //serveur/....

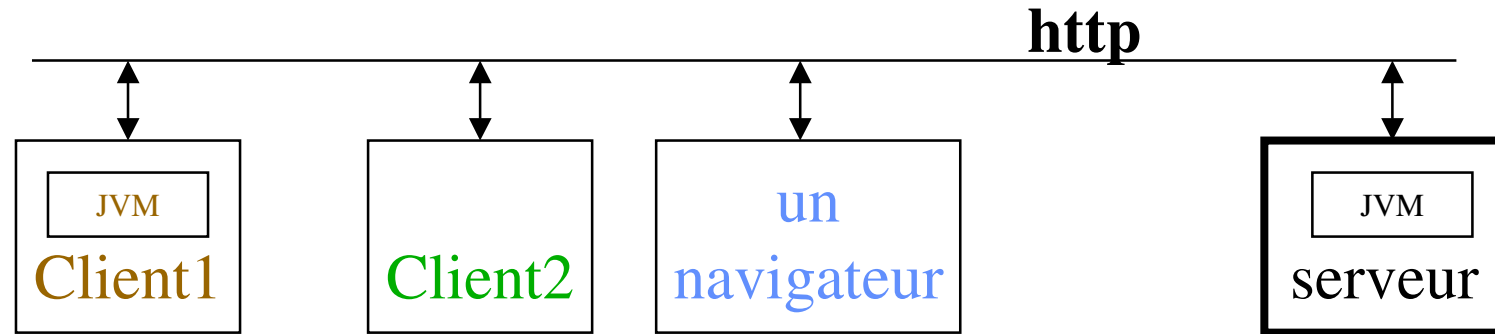
Appels distants protocole JRMP (rmi)



- **Le contexte**

- Clients Java uniquement
- Serveur en java
 - **rmi://serveurDeNoms/service**
 - **+ une JVM avec le service rmi**
- **JRMP Java Remote Method Protocol**

Appels distants protocole http



- **Le contexte**

- Client Java(application comme applette), **ou autres**
- Un navigateur

- Serveur en java , **ou autres**
 - **http: //serveur/index.html**
 - **Standard, universel ...**

Implémentations en Java

- **Paquetage java.net**

- **Principales classes**

- **ServerSocket**
 - **Socket**
 - **InetAddress**
 - **URLConnection**
 - ...

- **Quelques lignes de sources suffisent ...**

- **Paquetages java.rmi et java.rmi.server**

- **Une solution tout java, autre support**

usage de java.net TCP/IP



- 2 classes essentielles

Côté Serveur

- **java.net.ServerSocket**

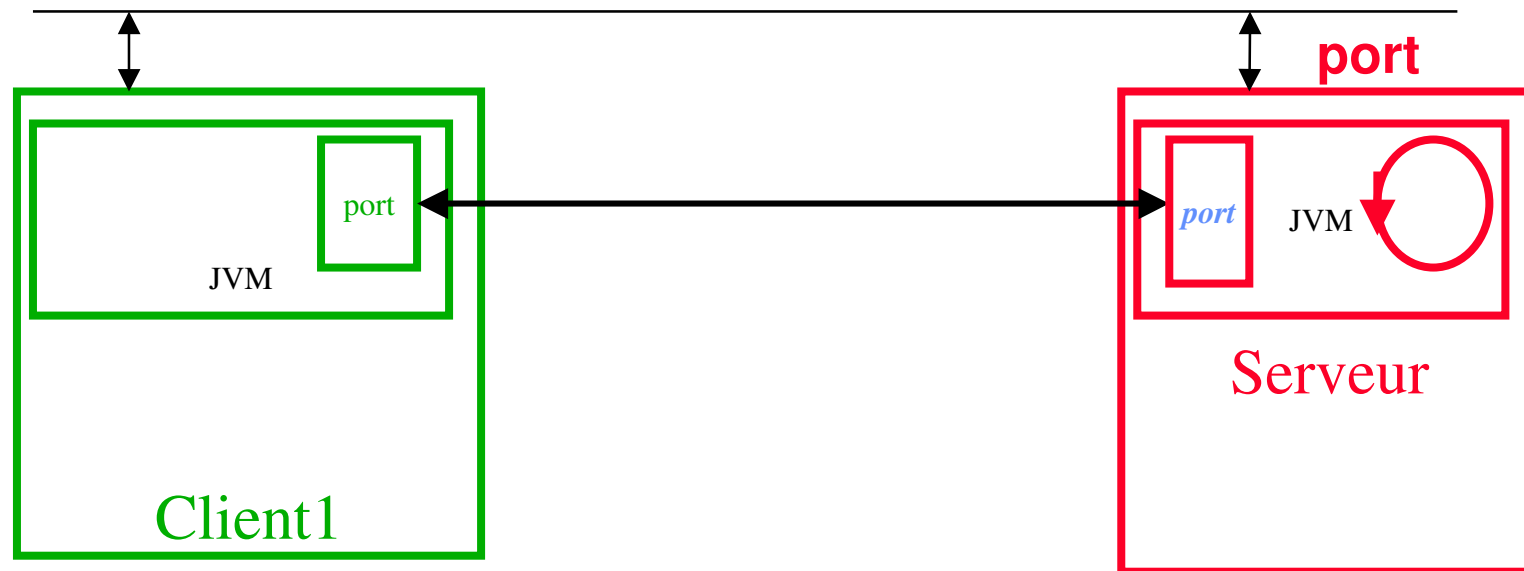
- Méthode **accept()** sur une instance de la classe ServerSocket

Côté Client

- **java.net.Socket**

- Envoi sur une instance de la classe Socket de données

Connexion / Principes



- **Le Serveur attend une requête sur son port**
 - `ServerSocket server = new ServerSocket(port)`
 - `Socket socket = server.accept();`
- **Dès la connexion établie,**
 - une instance de la classe `Socket` est engendrée sur un *port temporaire*
- **Établir une connexion par le client est effectuée par**
 - `Socket s = new Socket(Serveur, port)`

3 exemples

- **Serveur et clients**

1. **Au protocole « java »**

- **les instances transmises implementent `java.io.Serializable`**

2. **Au protocole « maison »**

- **Le serveur ne connaît que la commande « parle » et répond « bonjour »**
- **Tout autre commande est ignorée !**

3. **Au protocole http**

- **Seule la méthode GET /index.html HTTP1.0 est possible**
- **Un sous-ensemble donc ...**

En TCP un serveur (jfod.cnam.fr), protocole « java »

```
public class Serveur{

    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);

        while(true) {
            Socket socket = serveur.accept(); // attente active d'un client

// analyse de la requête
            ObjectInputStream ois= new ObjectInputStream(socket.getInputStream());
            Object obj = ois.readObject();

// réponse
            ObjectOutputStream oos= new ObjectOutputStream(socket.getOutputStream());
            oos.writeObject(obj.toString());

            socket.close();
        }
    }
}
```


En TCP le Client, protocole « java » (le serveur:jfod)

```
public class Client{

    public static void main(String[] args) throws Exception{
        // ouverture d'une connexion TCP
        Socket socket = new Socket("jfod.cnam.fr", 5000);
        ObjectOutputStream oos= new ObjectOutputStream( socket.getOutputStream());

        // envoi vers le serveur de cette « requête »
        SortedSet<String> l = new TreeSet<String>();
        l.add("TCP");l.add("essai");
        oos.writeObject( l);

        // lecture de la réponse retournée
        ObjectInputStream ois= new ObjectInputStream( socket.getInputStream());
        System.out.println("le serveur retourne : " + ois.readObject());

        socket.close();
    }
}
```

Discussion

- **Simple**
- **Appels synchrones**
- **Les paramètres doivent implémenter « `java.io.Serializable` »**
 - Une formalité : l'interface est un marqueur (vide)
 - Quelque soit la complexité de la structure !
 - La machine distante doit posséder tous les `.class` nécessaires
- **Dédié java : une JVM côté client et serveur**

Sérialisation : principes (rappels ?)

- Le paramètre est une instance de **java.io.Serializable**

```
public class XXXX implements java.io.Serializable{...}
```

Opérations internes : - **écriture** par copie de l'instance en profondeur
- **lecture** de l'instance

- **Ecriture de l'instance :**

```
OutputStream out = ...
```

```
ObjectOutputStream oos = new ObjectOutputStream( out );
```

```
oos.writeObject( obj );
```

*Les données d'instance sont copiées sauf les champs "**static**" et "**transient**"*

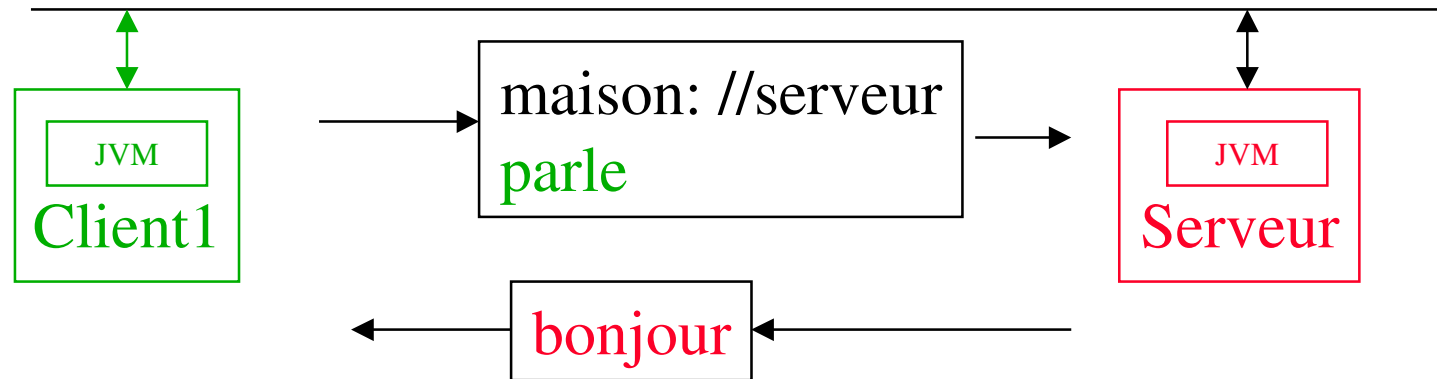
- **Lecture de l'instance :**

```
InputStream in = ...
```

```
ObjectInputStream ois = new ObjectInputStream( in );
```

```
Object obj = ois.readObject();
```

Exemple 2



- **Au protocole « maison »**
 - **Le serveur ne connaît que la commande « parle » et répond « bonjour »**
 - **Tout autre commande est ignorée !**
- **Client java ou autre**

Un serveur avec un protocole « maison »

```
public class Serveur{

    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);
        while(true) {
            Socket socket = serveur.accept();

            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String cmd = in.readLine();

            // traitement de la commande reçue
            DataOutputStream out = new DataOutputStream( socket.getOutputStream());
            if(cmd.equals("parle")){
                out.write("bonjour\n".getBytes());
            }else{
                out.write("commande inconnue ?\n".getBytes());
            }
            socket.close();
        }
    }
}
```

Le client « maison »

```
public class Client{

    public static void main(String[] args) throws Exception{
        Socket socket = new Socket("vivaldi.cnam.fr", 5000);
        DataOutputStream out= new DataOutputStream( socket.getOutputStream());
        out.write(args[0].getBytes());
        out.write("\n".getBytes());

        BufferedReader in = new BufferedReader(
            new InputStreamReader(socket.getInputStream()));

        System.out.println(in.readLine());

        socket.close();
    }
}
```

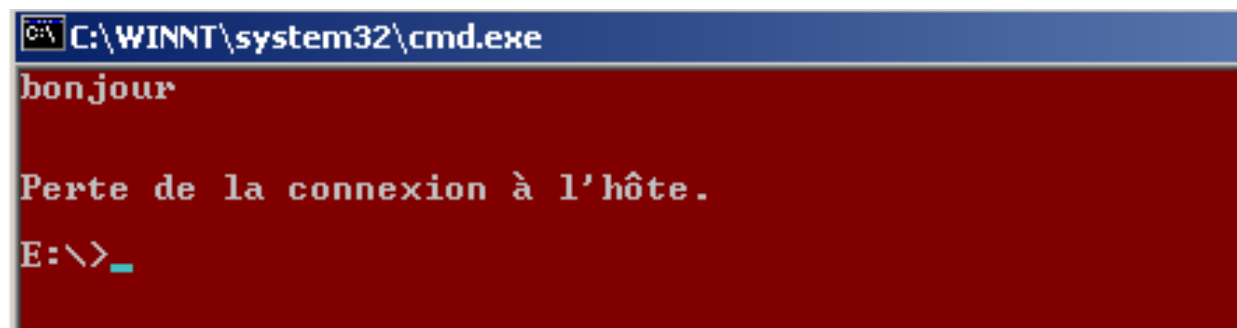
```
H:\NSY102\tp_pattern_correction>java -cp . question3.Client parle
bonjour
H:\NSY102\tp_pattern_correction>java -cp . question3.Client écris
commande inconnue ?
```

Un client « maison », telnet

- **telnet localhost 5000**
 - **parle** // frappe des touches sans écho...



```
C:\WINNT\system32\cmd.exe - telnet localhost 5000
```



```
C:\WINNT\system32\cmd.exe
bonjour
Perte de la connexion à l'hôte.
E:\>_
```

- petit outil utile : tcpview sous windows

<http://www.microsoft.com/technet/sysinternals/utilities/tcpview.msp>

Discussion

- **Simple**
 - 10 lignes
- **Appel synchrone**
 - Le contenu de la requête respecte une grammaire, un protocole
- **Clients comme serveur : en java ou autres**
- **telnet** comme outil de mise au point ?
- **tcpview** ou autres utiles ?

Exemple 3

- **Le protocole HTTP**
 - Les méthodes GET, POST,

- **Mise en œuvre / démo**
 - Usage d'un client telnet sur un site existant

 - Une application Java cliente

 - Un serveur en java
 - Un navigateur comme client
 - Une application cliente en java

Protocole HTTP

- ***HyperText Transfer Protocol***
 - Au dessus de TCP
- **Les Méthodes**
 - GET /index.html HTTP/1.0
 - HEAD
 - POST

 - PUT
 - DELETE
 - TRACE
 - CONNECT

 - Voir <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

Côté serveur, accept

```
ServerSocket listen = new ServerSocket (HTTP_PORT) ;

while (!stopped()) {
    try{
        new this.Connexion(listen.accept()); // traitement
    }catch(Exception e) {
    }
}

listen.close();
}
```

Ici chaque requête engendre

la création d'une instance de la classe interne **Connexion**

Chaque instance créée engendre à son tour un « Thread »

La méthode accept est bloquante

Côté serveur, à chaque Connexion un Thread

```
public class Connexion extends Thread{
...
public Connexion(Socket s) {
    this.s = s; start();
}

public void run() {
    try{
        BufferedReader is = new BufferedReader(
            new InputStreamReader(s.getInputStream()));
        DataOutputStream os =
            new DataOutputStream(s.getOutputStream());

        // analyse du contenu au bon protocole HTTP

        // envoi du document
    }
}
```

Schéma avec Un Pool de Thread

```
class WebServer { // 2004 JavaOneSM Conference | Session 1358
    Executor pool = Executors.newFixedThreadPool(7);

    public static void main(String[] args) {
        ServerSocket socket = new ServerSocket(80);
        while (true) {
            final Socket s = socket.accept();
            Runnable r = new Runnable() {
                public void run() {
                    BufferedReader is = new BufferedReader(
                        new InputStreamReader(s.getInputStream()));
                    DataOutputStream os =
                        new DataOutputStream(s.getOutputStream());
                    // analyse du contenu au bon protocole HTTP
                    // envoi du document
                }
            };
            pool.execute(r);
        }
    }
}
```

Côté serveur, accept « peut-être »

```
ServerSocket listen = new ServerSocket (HTTP_PORT) ;
listen.setSoTimeout (TIME_OUT) ;
while (!stopped()) {
    try{
        new Connexion(listen.accept()) ;
    }catch (SocketTimeoutException e) {
        // délai de garde échu, ou le délai a chu
    }catch (Exception e) {
    }
}
listen.close() ;
}
```

Méthode accept avec délai de garde

exception `SocketTimeoutException` à l'échéance

Requête GET avec telnet

- Un client telnet et un site du Cnam

- telnet jfod.cnam.fr 80
 - GET /index.html HTTP/1.0 (*frappe sans écho*)

HTTP/1.0 200 OK

Last-Modified: Thu, 08 Feb 2007 14:55:29 GMT

Date: Thu, 08 Mar 2007 10:33:55 GMT

Server: Brazil/1.0

Content-Length: 7624

Content-Type: text/html

Connection: close

<HTML>

<HEAD>

<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">

.....

Le résultat est retourné, le source du fichier index.html précédé de quelques informations...

Démonstration/discussion

- **Extrait du Web**

- OneShotHttpd
- SimpleHttpd

- Les sources ici <http://jod.cnam.free.fr/serveurs/>

OneShot Httpd by Hendrik

```
public class OneShotHttpd {
    protected static File docRoot;
    public final static int HTTP_PORT = 8080;

    public static void main(String argv[]){
        try{
            docRoot = new File(".");
            ServerSocket listen = new ServerSocket(HTTP_PORT);
            Socket client = listen.accept();
            BufferedReader is = new BufferedReader(new
                InputStreamReader(client.getInputStream()));
            DataOutputStream os = new DataOutputStream(client.getOutputStream());
            String request = is.readLine();
            StringTokenizer st = new StringTokenizer(request);
            if((st.countTokens() == 3) && st.nextToken().equals("GET")){
                String filename = docRoot.getPath() + st.nextToken();
                if(filename.endsWith("/") || filename.equals(""))
                    filename += "index.html";
                File file = new File(filename);
                sendDocument(os, file);
            } else
                System.err.println("400 Bad Request");
            is.close(); os.close(); client.close();
        }catch(IOException ioe){
            System.err.println("Error: " + ioe.toString());
        }
    }
}
```

OneShot « envoi du document »

```
public static void sendDocument(DataOutputStream out, File file) throws
    IOException{
    try{
        BufferedInputStream in = new BufferedInputStream(new
        FileInputStream(file));
        byte[] buf = new byte[1024];
        int len;
        while((len = in.read(buf,0,1024)) != -1)
        {
            out.write(buf,0,len);
        }
        in.close();
    }
    catch(FileNotFoundException fnfe)
    {
        System.err.println("404 Not Found");
    }
}
}
```

Requête GET en Java

- **L'essentiel**

- Créer une URL
- Ouvrir une connexion
 - Écrire et lire sur les flots associés

- **Classe java.net.URL**

- **Classe java.net.URLConnection**

- `URL url = new URL("http://jfod.cnam.fr/index.html");`
- `URLConnection connection = url.openConnection();`

Requête GET au complet

```
public void testGET()throws Exception{
    URL url = new URL("http://jfod.cnam.fr/index.html" );
    URLConnection connection = url.openConnection();

    BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));

    String inputLine = in.readLine();
    while(inputLine != null){
        System.out.println(inputLine);
        inputLine = in.readLine();
    }
    in.close();
}
```

Requête GET avec paramètres

```
public void testGET() throws Exception{
    URL url =
        new URL("http://jfod.cnam.fr:8999/ds2438/?listAll=on" );
    URLConnection connection = url.openConnection();
    connection.setDoInput(true);

    BufferedReader in = new BufferedReader(
        new InputStreamReader(connection.getInputStream()));

    String inputLine = in.readLine();
    while(inputLine != null){
        System.out.println(inputLine);
        inputLine = in.readLine();
    }
    in.close();
}
```

Requête POST

```
URL url = new URL ("http://jfod.cnam.fr/index.html");  
URLConnection connection = url.openConnection();
```

```
connection.setDoInput(true);  
connection.setDoOutput(true);
```

```
PrintWriter out = new PrintWriter(connection.getOutputStream());  
out.print("listAll=on");
```

```
out.close();
```

```
BufferedReader in = new BufferedReader( new  
InputStreamReader(connection.getInputStream()));  
String inputLine = in.readLine();  
while(inputLine != null){  
    System.out.println(inputLine);  
    inputLine = in.readLine();  
}  
in.close();
```

Classes utiles

- **InetAddress**
 - Adresse IP en « clair »
- **URL**
 - Pour Uniform Resource Locator, sur le www
- **URLConnection**
 - Une classe abstraite, super classe de toutes les classes établissant un lien entre une application et une URL
 - Sous-classes
 - **HttpURLConnection, JarURLConnection**
 - Patron Fabrique afin d'écrire son propre gestionnaire de protocole
 - Voir <http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/java.url2.html>
 - Méthode `URLConnection.setContentHandlerFactory(...);`

Patron Fabrique, au bon protocole ...

- **Un protocole propriétaire**

- <http://java.sun.com/developer/onlineTraining/protocolhandlers/>
- <http://monge.univ-mlv.fr/~rousseau/RESEAUJAVA/java.url2.html>

java.net.InetAddress

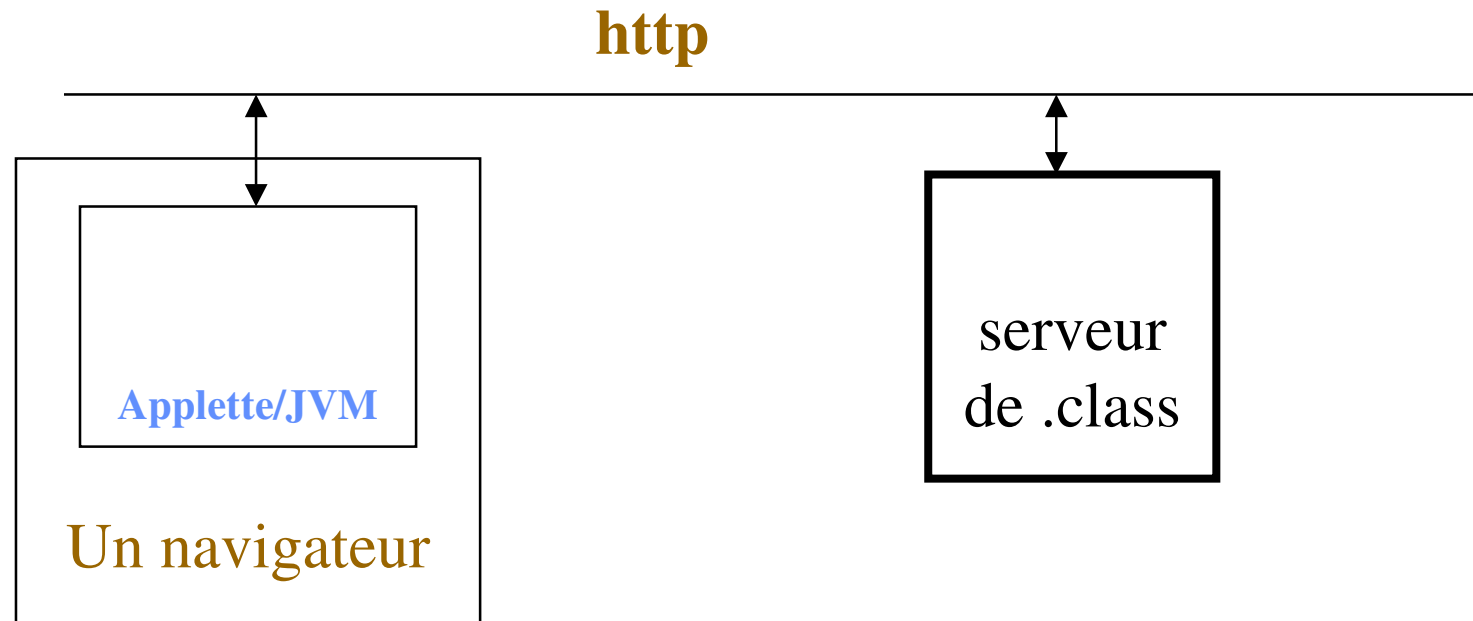
- **L'adresse IP de la machine locale**
 - `InetAddress.getLocalHost().getHostAddress()`

- **Un client sur la même machine que le serveur**
 - `new Socket(InetAddress.getLocalHost(), 8999);`

Architecture, mise en oeuvre, répartition, téléchargement, maintenance, ...

- **Un client du Web**
- **À l'aide d'un navigateur**
 - Applet & serveur ?
 - Application Java ?
- **En application**
 - **URLClassLoader** (*voir en annexe*)
 - **Java Web Start** (*un exemple en annexe*)

Applette : rappels



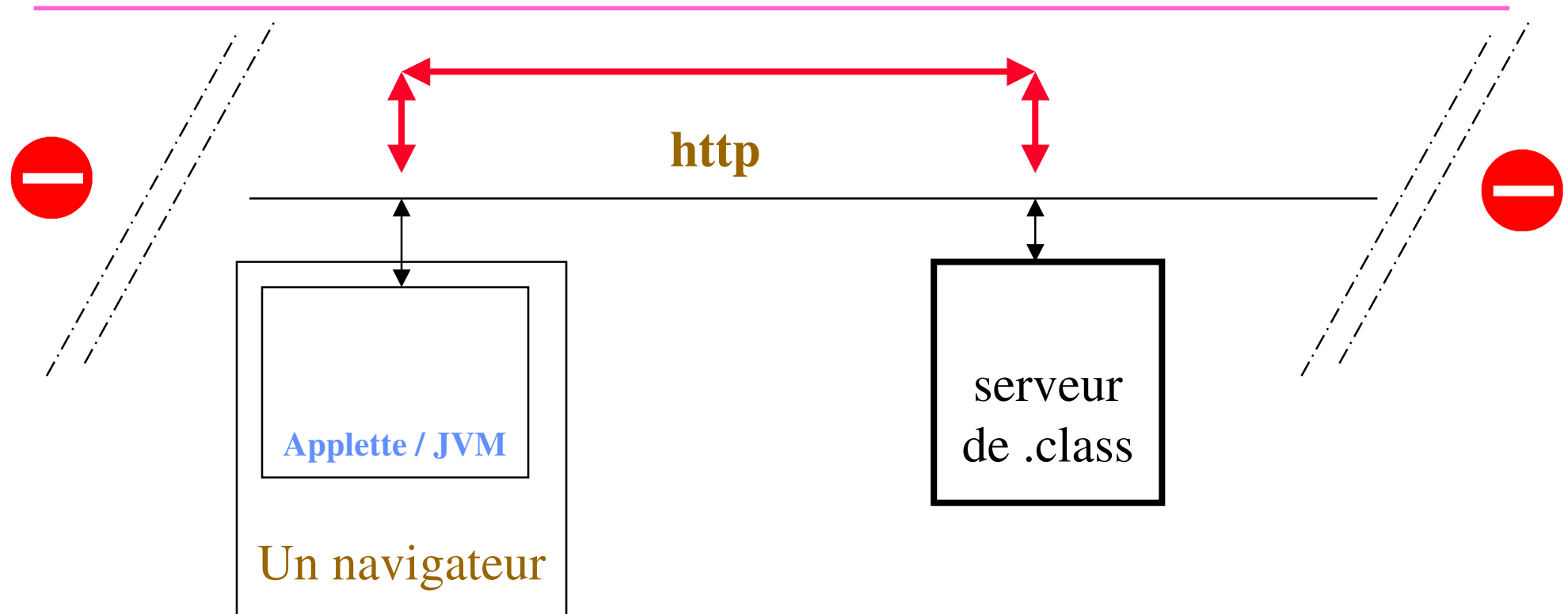
1. L'applette est téléchargée

- Les fichiers `.class` sont demandés au fur et à mesure
- Une archive peut être téléchargée

2. Des contraintes de sécurité sont installées par défaut

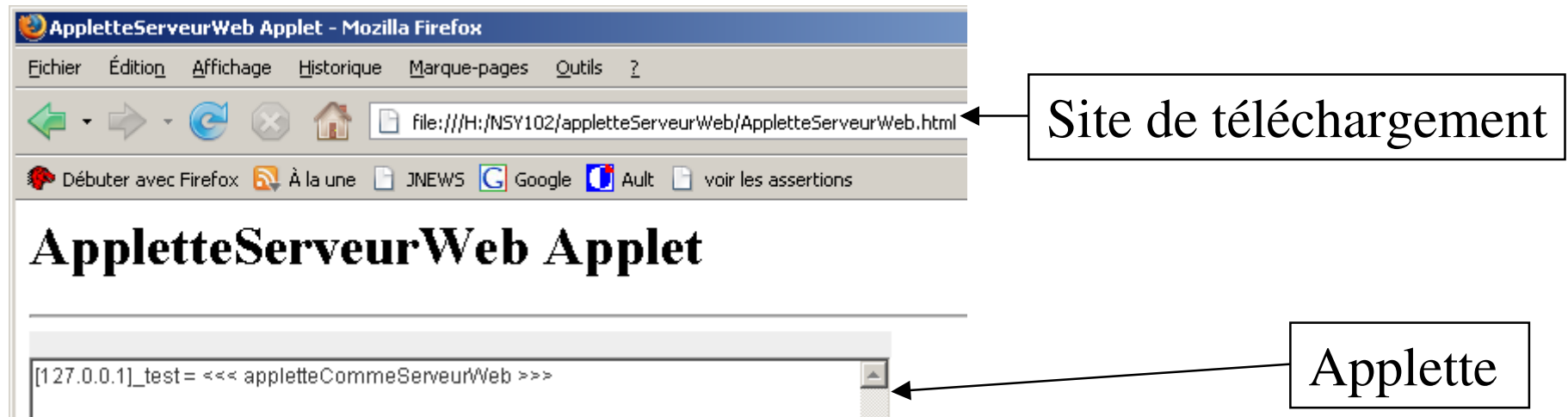
- Modifiables, mais peu recommandé !
- Un navigateur ou bien l'outil `appletviewer`
 - `appletviewer http://machine_distante/tp/tp1.html`

L'Applette contient un serveur Web



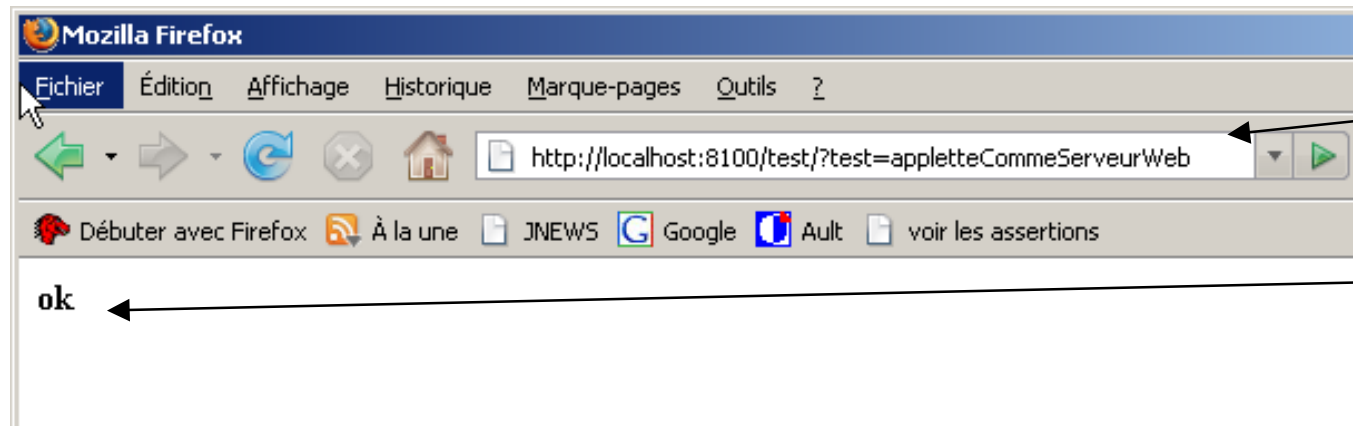
1. L'applette téléchargée contient un serveur HTTP
 2. Le navigateur héberge donc un « serveur Web » (côté client)...
 3. **Attention : un seul client est possible ***
 - Le serveur depuis lequel l'applette est issue (et en localhost ...)
- * (avec la stratégie de sécurité standard, qu'il n'est pas recommandé de modifier(rappel))

L'applette en démo, ici port 8100



- **Cette applette Serveur Web**
 - affiche l'adresse IP du client et la valeur du paramètre test
 - retourne au client **ok**

L'applette en démo, suite



requête

résultat

– Le résultat de la requête **ok**

Le source ici <http://jfod.cnam.free.fr/apletteServeurWeb/>

L'applette est ici <http://jfod.cnam.free.fr/apletteServeurWeb/AppletteServeurWeb.html>

Une requête possible : <http://localhost:8100/test/?test=succes>

Source de l'applette : un extrait

```
public class AppletteServeurWeb extends JApplet implements Runnable{
    public static final int TIME_OUT = 500;

    private Thread    local;
    private String    urlPrefix  = "test";
    private int       port       = 8100;

    public void init(){
        JRootPane rootPane = this.getRootPane();
        // IHM et paramètres

        local = new Thread(this);
        local.start();
    }

    public void run(){
        ServerSocket listen = new ServerSocket(this.port);
        listen.setSoTimeout(TIME_OUT);
        while(!local.isInterrupted()){
            Socket          socket = null;
            BufferedReader   in     = null;
            DataOutputStream out    = null;
            try{
                socket = listen.accept();
                in     = new BufferedReader(new InputStreamReader(socket.getInputStream()));
                out    = new DataOutputStream(socket.getOutputStream());

                // analyse du contenu au bon protocole HTTP
                // envoi du document
            }
        }
    }
}
```

Discussion : Applette et serveur /TCP

- **Pourquoi faire ?**
- **Un simple navigateur et sa JVM suffisent**
 - Stratégie de sécurité standard
- **Transparence assurée envers le client**
 - http : protocole standard
- **Notification asynchrone sur le web**
 - Un navigateur et son applette sont notifiés
- **Attention aux pare-feux**
 - ...

En conséquence

- **Toute machine équipée d'un navigateur acceptant des applettes peut devenir un serveur Web**

Mais

- **Un seul client possible de ce serveur web !** *C'est peu ...*
 - Avec la stratégie de sécurité par défaut (qui doit être conservée)

Et alors... et alors ...

- **Et, MVC est arrivé,**
 - Les vues seront des applettes/serveur web
 - **Leur seul client sera le Modèle**
 - Les contrôleurs adresseront le Modèle

Architecture distribuée et HTTP

- **Hypothèse**

- **Toute machine connectée possède un serveur Web/HTTP**
 - **En application autonome / et ou téléchargée**
 - **Dans un navigateur comme une applette**

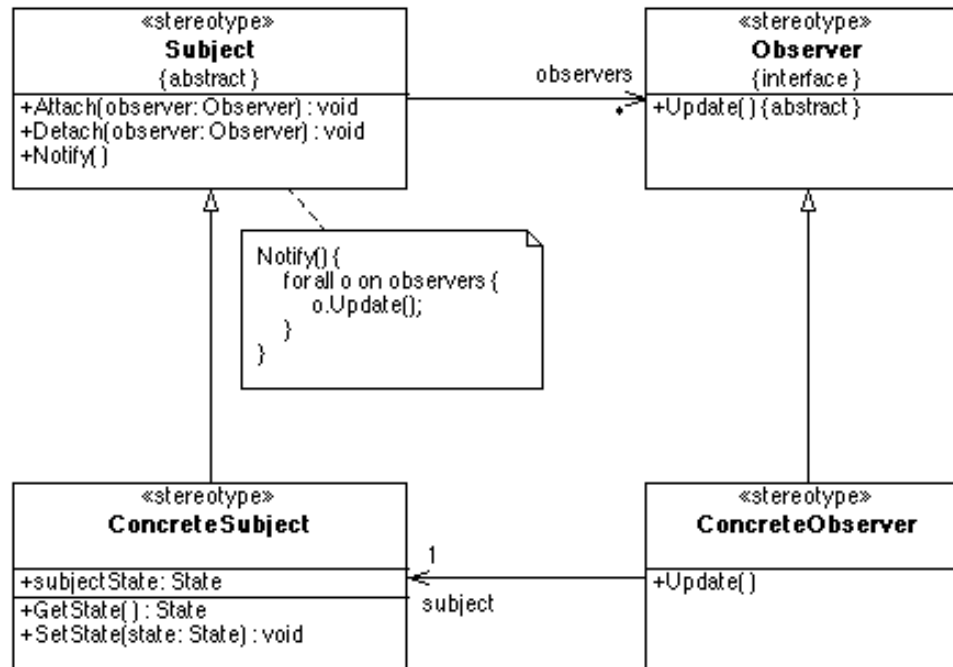
- **Premier essai d'architecture**

- **Patron Observateur/Observé**
 - **Lors d'un changement d'état, notification aux observateurs inscrits**

- **Deuxième essai**

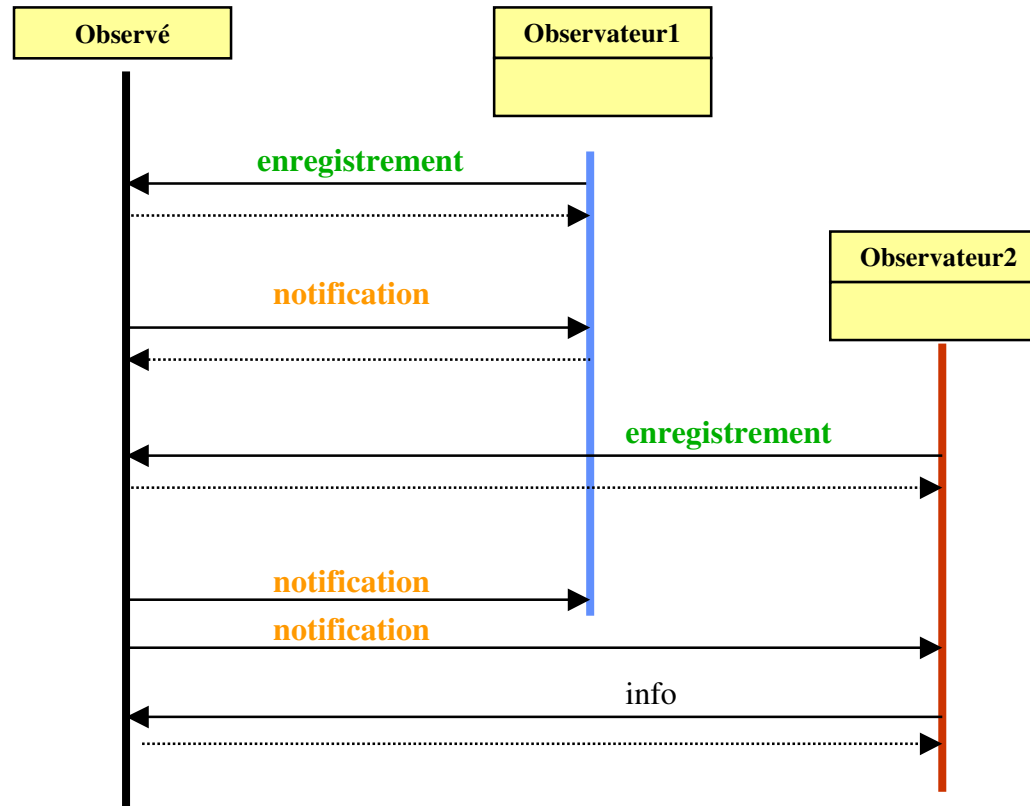
- **Patron Modèle Vue Contrôleur**

Observateur/Observé : l'original



- notification

Diagramme de séquence



Adéquation ?

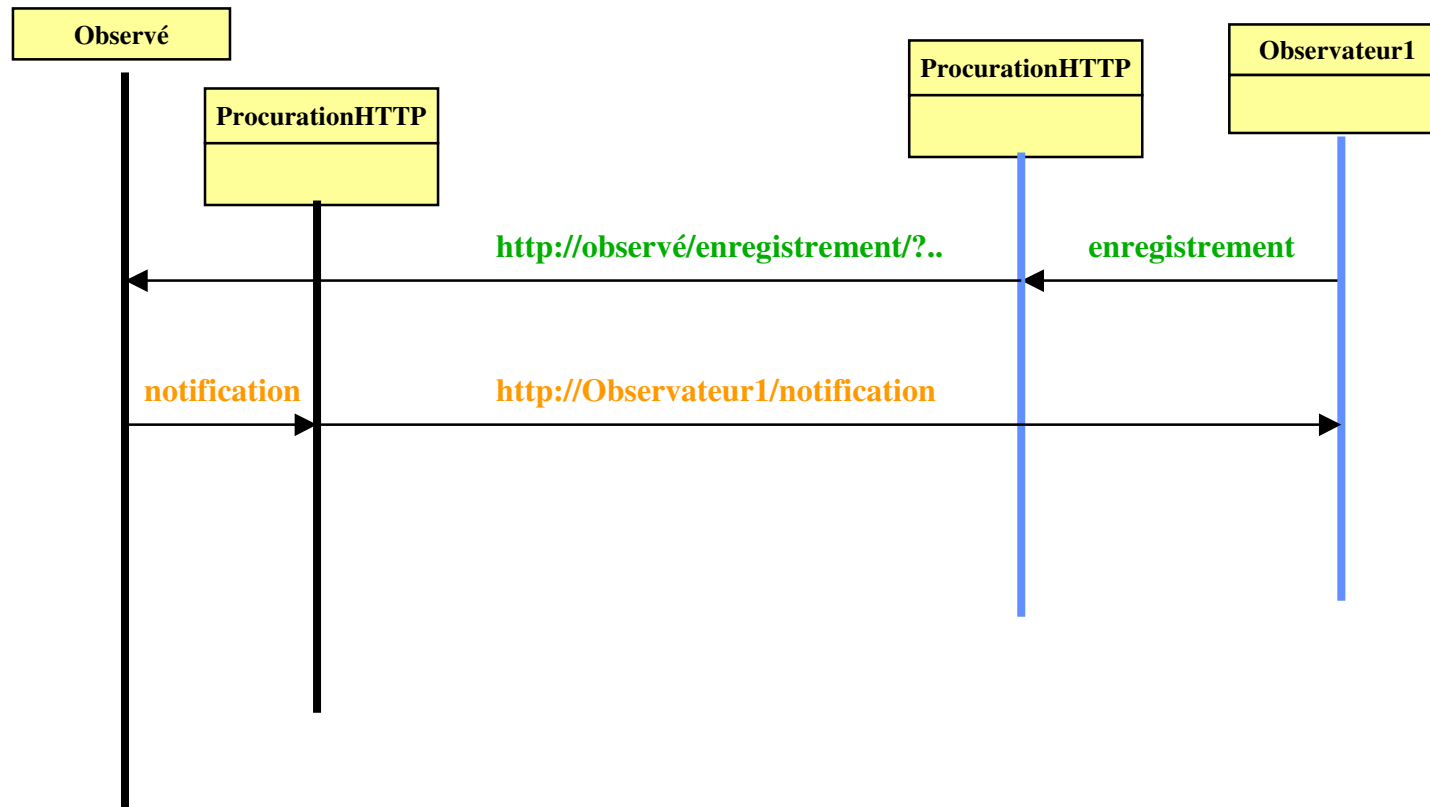
enregistrement

<http://observé/addObserver?url=http://observateur1/update/>

notification

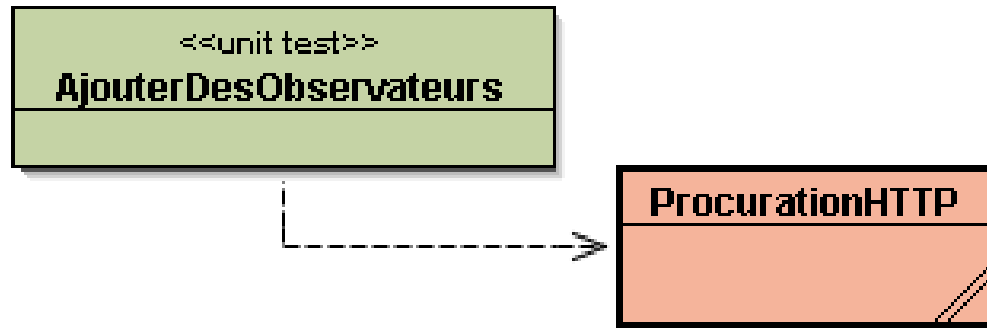
<http://observateur1/update/?arg=evt>

Patron Procuration



- **Procuration à l'émission**
 - Ou l'adéquation appel de méthodes/ requêtes HTTP

Exemple de mise en oeuvre



- **Adéquation**

- **addObserver** → **http://site_observable/...**

Un Exemple de procuration pour addObserver

```
public class ProcurationHTTP{
    private String urlObservé;
    private final ExecutorService executor;

    public ProcurationHTTP (String urlObservé) {
        this.urlObservé = urlObservé;
        this.executor    = Executors.newCachedThreadPool ();
    }

    public void addObserver (String urlObservateur) {
        Future<String> res = executor.submit (
            new RequeteHTTP ("commande=addObserver&url=" + urlObservateur));
    }
}
```

RequêteHTTP en Java, un classique Callable ...

```
private class RequeteHTTP implements Callable<String>{
    private String result = "";
    private String paramètres;

    public RequeteHTTP(String paramètres){
        this.paramètres = paramètres;
    }

    public String call(){
        try{
            URL url = new URL(urlObservé + "?" + paramètres);
            URLConnection connection = url.openConnection();

            connection.setDoInput(true);

            BufferedReader in = new BufferedReader( new InputStreamReader(connection.getInputStream()));

            String inputLine = in.readLine();
            while(inputLine != null){
                result = result + inputLine;
                inputLine = in.readLine();
            }
            in.close();
        }catch(Exception e){}
        return result;
    }
}
```

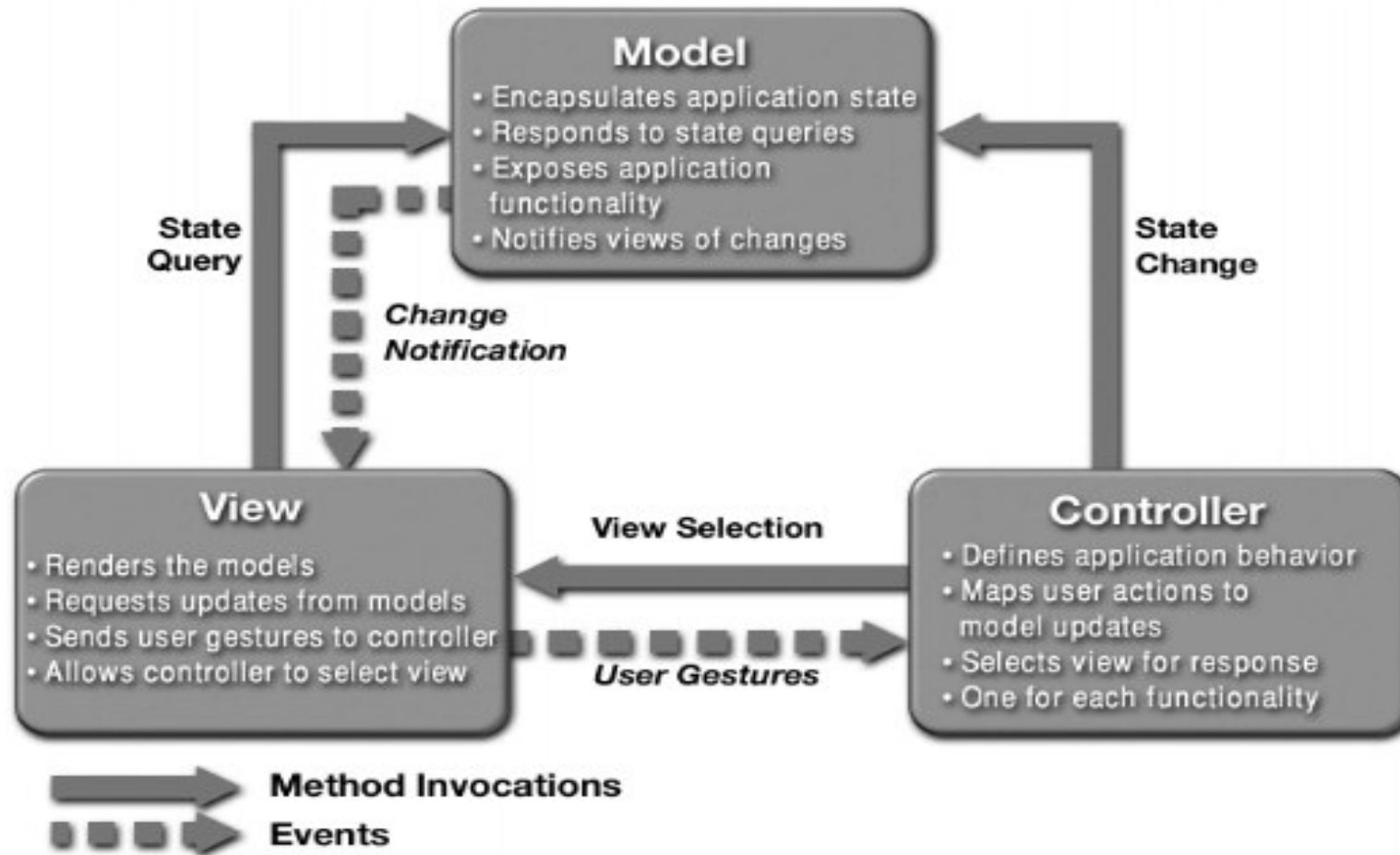

Un usage : un test

- Deux applettes serveurs Web deviennent des observateurs

```
public void testDeuxObservateurs() {
    ProcurationHTTP proxy = new
    ProcurationHTTP("http://jfod.cnam.fr:8799/observer.html");

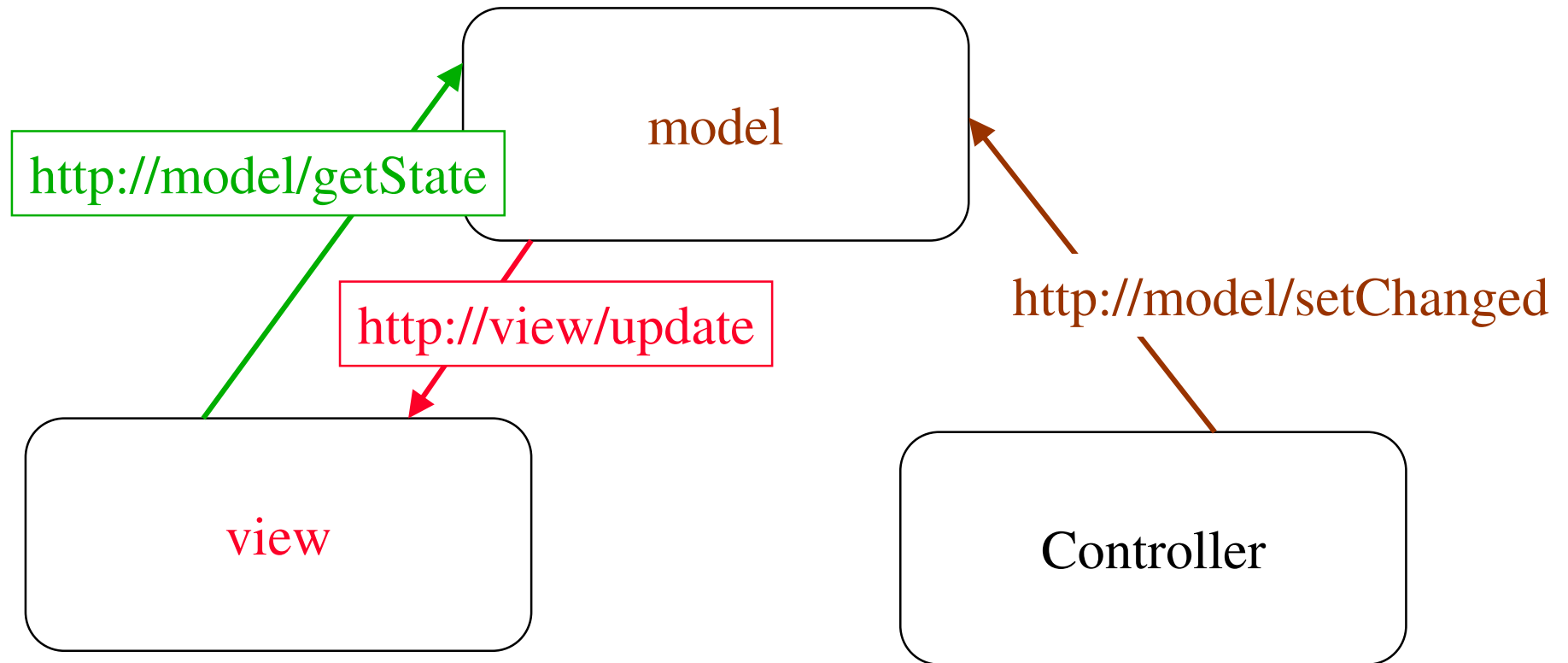
    proxy.addObserver("http://localhost:8100/tests/?test=obs1");
    proxy.addObserver("http://localhost:8200/tests/?test=obs2");
}
```

MVC rappel



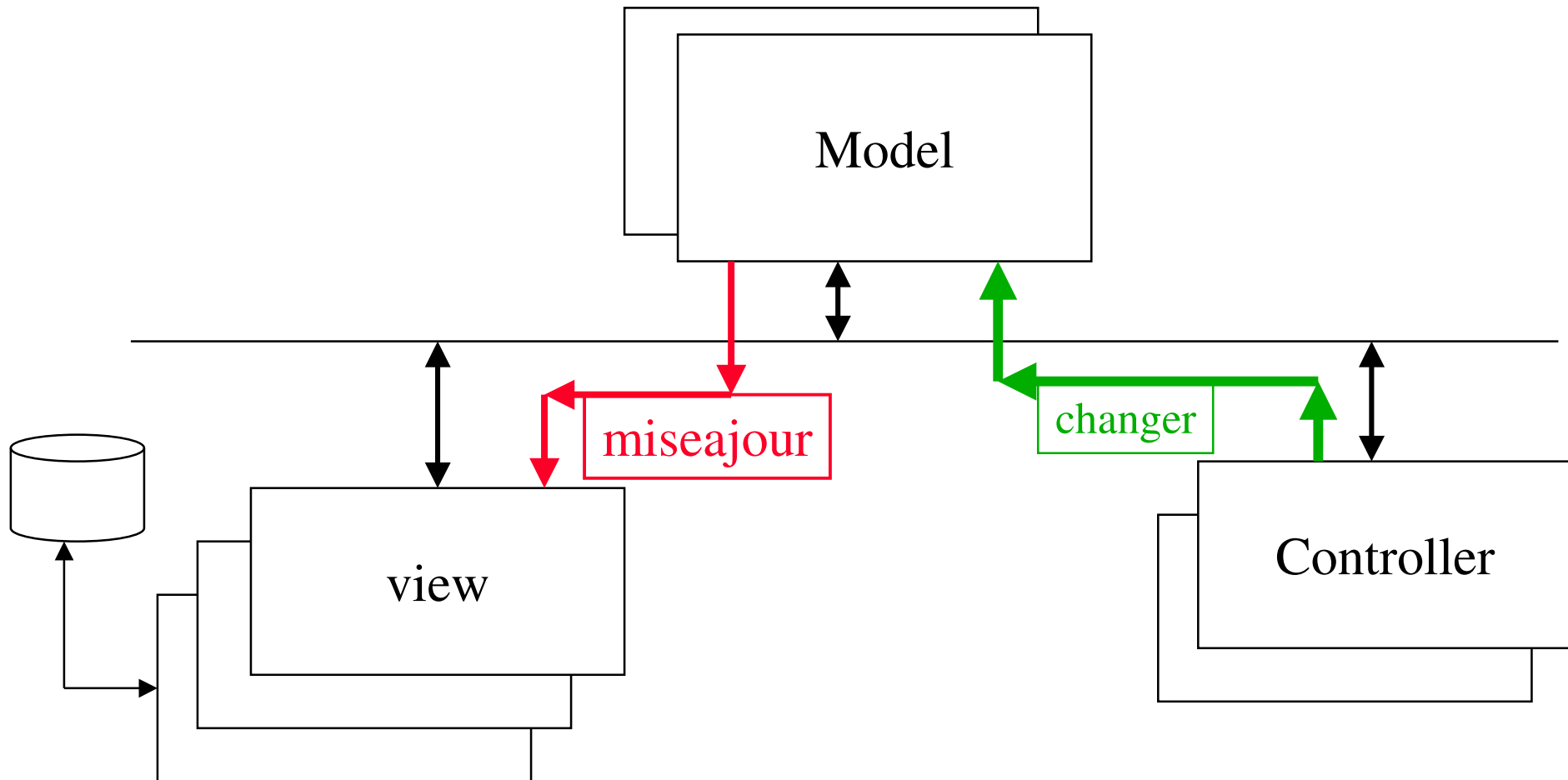
- <http://java.sun.com/blueprints/patterns/MVC-detailed.html>

MVC distribué



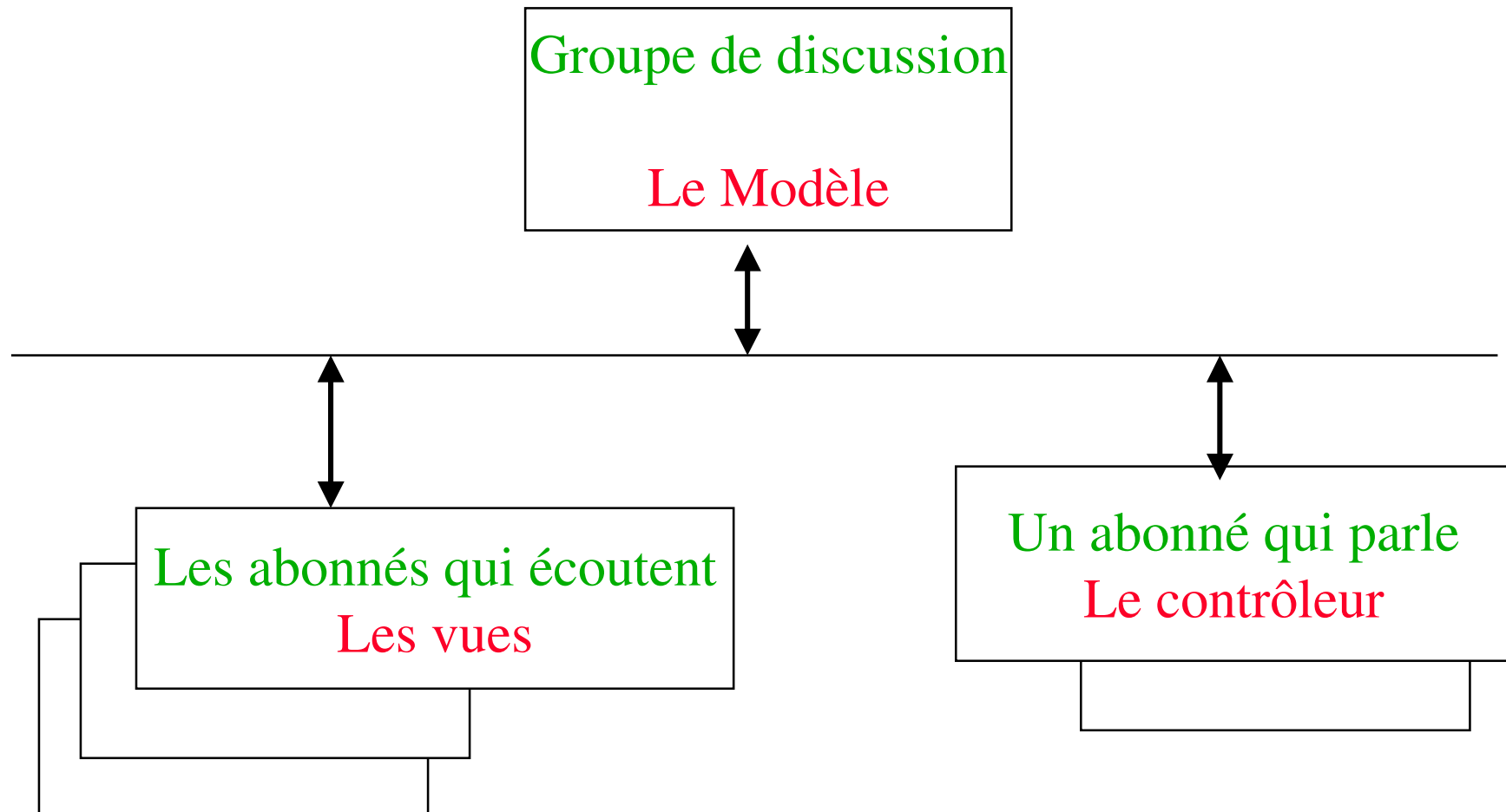
- **Adéquation appels de méthodes / requêtes HTTP**
 - Application existante + Procurations

3 Vues, « 2 modèles », 2 contrôleurs



- Un deuxième Modèle éventuel pour la redondance
- Une Vue peut assurer la persistance

Un exemple parmi d'autres : un « chat »



- Une instance de MVC

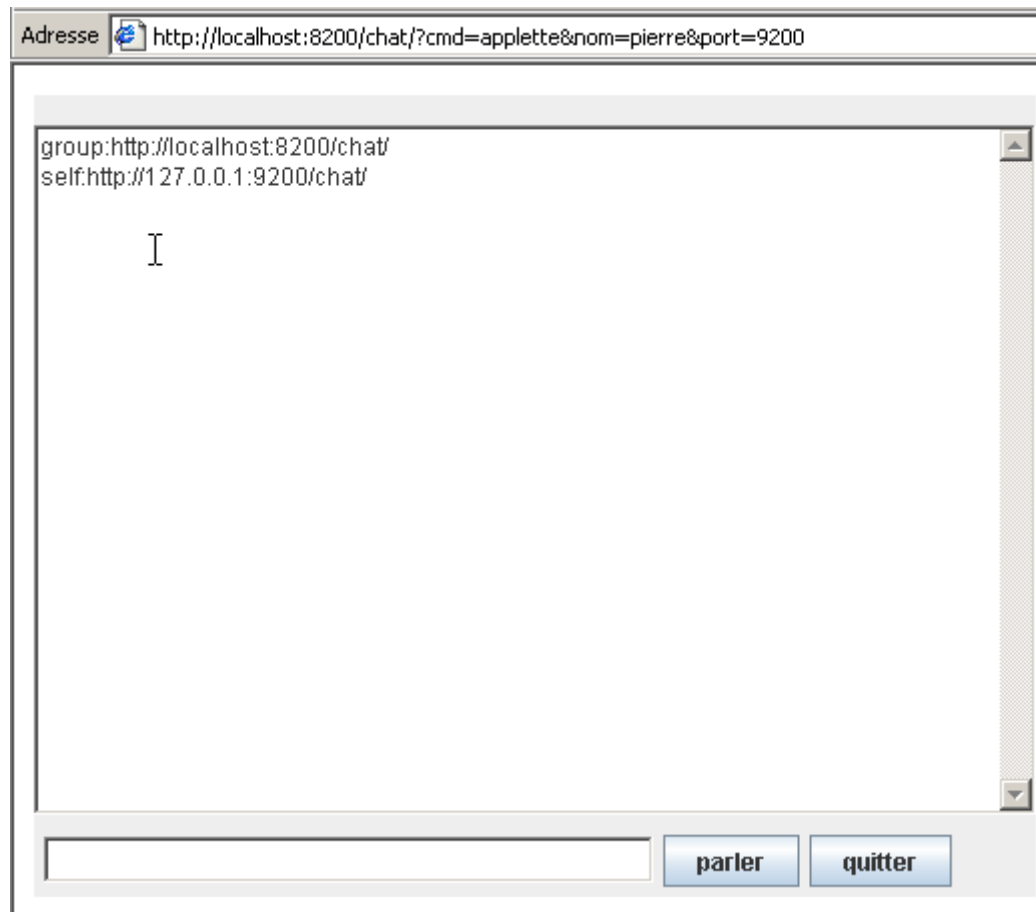
Une démonstration en ligne ou en local...

- **Un logiciel de causerie**
 - Généralement en intranet seulement
 - Contraintes des pare-feux (port, protocoles, ...)
- **Un gestionnaire du groupe de discussion est déjà en**
 - <http://pc5357c.esiee.fr:8200/chat/>
- **Ajouter un client**
 - Depuis votre navigateur
 - [http:// pc5357c.esiee.fr:8200 /chat/?cmd=applette&nom=pierre&port=9100](http://pc5357c.esiee.fr:8200/chat/?cmd=applette&nom=pierre&port=9100)
 - Ou bien depuis une console avec appletviewer
 - appletviewer [http:// pc5357c.esiee.fr:8200 /chat/?cmd=applette&nom=paul&port=9100](http://pc5357c.esiee.fr:8200/chat/?cmd=applette&nom=paul&port=9100)

Démonstration/ discussion

Démonstration deux copies d'écran

```
H:\IN413_IN4A21\IN413_0708\tp6_correction>java -cp . question3.ChatGroup 8200  
chat groupe en http://163.173.228.59:8200/chat/
```



- **Pierre s'inscrit au chat**
- *L'applette devient un observateur*

Démonstration suite, paul et pierre causent

Adresse  <http://localhost:8200/chat/?cmd=aplette&nom=pierre&port=9200>

```
group:http://localhost:8200/chat/  
self:http://127.0.0.1:9200/chat/  
paul_ <<< bonjour+pierre >>>  
pierre_ <<< bonjour+paul >>>
```

```
group:http://localhost:8200/chat/  
self:http://127.0.0.1:9100/chat/  
paul_ <<< bonjour+pierre >>>  
pierre_ <<< bonjour+paul >>>
```

Java Web Start

- **JavaWebStart**

- Téléchargement d'applications Java
 - **Un descripteur au format JNLP Java Network Launch Protocol**
- Une archive Java signée
- Recherche automatique de la dernière version
- **Console> javaws**
- **Ou depuis un navigateur**
- **Voir** http://www.java.com/fr/download/faq/java_webstart.xml

- **Toute application java, comme par exemple un serveur Web**

- **Démonstration suite ...**
 - [http:// pc5357c.esiee.fr:8200 /chatclient.jnlp](http://pc5357c.esiee.fr:8200/chatclient.jnlp)**
ici vous vous appellerez alfred sur le port 9500... non mais

Conclusion intermédiaire

- **MVC, web**
 - Déjà vu
 - JMS, publish-subscribe en standard ...
- **Classes essentielles**
 - **ServerSocket**
 - Méthode **accept** bloquante ou avec un délai de garde
 - **Socket**
 - Un thread à chaque connexion,
 - Usage d'un d'un pool de Thread
- **Un Serveur en quelques lignes de Java**
- ***Bien mais***

Le patron HeartBeat

- **Le serveur est-il en état de fonctionner ?**
- **Requête régulière et décision en conséquence**

ou

- **Si la réponse dépasse une certaine durée, le serveur est considéré comme en panne**
- **Ou bien un bail est alloué à la ressource ...**

Un source de HeartBeat : le serveur est-il actif ?

```
public class HeartBeat implements Callable<Void>{
    private Map<String,Future<String>> observable;
    private Map<String,Future<String>> unaccessible;
    private boolean stopped;
    private final ExecutorService executor;
    private final int period;

    public HeartBeat(int period){
        this.observable = new HashMap<String,Future<String>> ();
        this.unaccessible = new HashMap<String,Future<String>> ();
        this.executor = Executors.newCachedThreadPool ();
        this.period = period;
        executor.submit(this);
    }

    public synchronized void addObserver(String url){
        this.observable.put(url, null);
    }
}
```

Un source suite ...

```
public synchronized void call(){
    while(!stopped){
        try{
            // attente de la période
            Thread.sleep(period); count++;
            for(String obs : observable.keySet()){
                observable.put(obs, executor.submit(new RequeteHTTP(obs)));
            }

            // Retrait de tous les « inaccessibles »
            Iterator<String> it = observable.keySet().iterator();
            while(it.hasNext()){
                String obs = null;
                try{
                    obs = it.next();
                    observable.get(obs).get();
                }catch(Exception e){ it.remove(); inaccessible.put(obs, null);
                }
            }
            // retenter les accès toutes les k*period
        }catch(InterruptedException ie){}
    }
    return null;}
}
```

Conclusion

- **Programmation Réseau avec Java**
 - Simple
- **MVC distribué**
- **À regarder de plus près**
 - Patrons Reactor et Acceptor
 - **java.nio Depuis le j2se 1.4**
 - rmi ? Remote Method Invocation
 - **Une solution tout java, performante : un autre support**

Annexes

- 1. Derrière un proxy**
- 2. Un exemple d'analyse du bon protocole**
- 3. Junit & http = httpjunit**
- 4. Java Web Start**
- 5. URLClassLoader**

Annexe 1 : derrière un proxy

```
java -cp . -Dhttp.proxyHost=cache.esiee.fr  
        -Dhttp.proxyPort=3128 ClientHTTP
```

ou

```
public static  
    void setHttpProxy(String proxyHost, int proxyPort) {  
        Properties prop = System.getProperties();  
        prop.put("proxySet", "true");  
        prop.put("http.proxyHost", proxyHost);  
        prop.put("http.proxyPort", Integer.toString(proxyPort));  
    }
```

Annexe 2 : analyse du contenu (*simplifié*)

A chaque requête

```
String request = in.readLine();
StringTokenizer st = new StringTokenizer(request);
String token = st.nextToken();
if(token.equals("GET")){
    String paramUrl = st.nextToken();
    File file = new File(".") + paramUrl);
    send(out, file);
} else {
    ...}
```

Annexe 2 : envoi du document (*simplifié*)

```
private void send(DataOutputStream os, File file) throws Exception{
    try{
        BufferedInputStream in = new BufferedInputStream(
            new FileInputStream(file));
        os.write("HTTP/1.0 200 OK\r\n".getBytes());
        os.write(new String("Content-Length" + new Long(file.length()) +
            "\r\n").getBytes());

        os.write(new String("Content-Type: " + guessType(file.getPath())
            + "\r\n\r\n").getBytes());

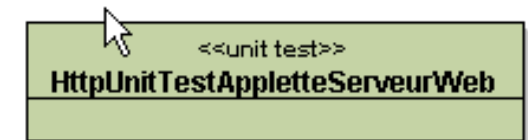
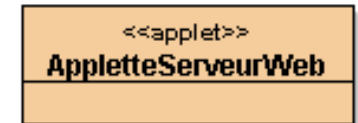
        byte[] buf = new byte[1024];
        int len;
        while((len = in.read(buf,0,1024)) != -1){
            os.write(buf,0,len);
        }
        in.close();
    }catch(FileNotFoundException fnfe){
        ...
    }
}
```

Annexe 3 : http Test unitaires

- **HttpUnit** <http://httpunit.sourceforge.net/>
- **Exemple ici intégré à BlueJ*** :

```
public void testApplette() throws Exception {  
    WebConversation conversation = new WebConversation();  
    WebRequest request = new GetMethodWebRequest(  
        "http://localhost:8100/test/?test=truc" );  
  
    WebResponse response = conversation.getResponse( request );  
    assertEquals(" réponse ???", "<b>ok</b>", response.getText());  
}
```

- * **httpunit.jar** et **Tidy.jar** sont dans `<BlueJ_HOME>/lib/userlib/` ou bien dans le répertoire `+libs` de votre projet
- Voir aussi **jwebunit** <http://jwebunit.sourceforge.net>



Annexe 4 : Java Web Start

- Téléchargement depuis le Web d'applications Java certifiées
 - Assurance de toujours disposer de la dernière version
 - Principe de mise en œuvre
 - Côté serveur
 - Signer une archive Java (.jar)
 - Proposer le fichier JNLP (Java Network Launch Protocol)
 - Installer ce fichier et cette archive sur un serveur
 - Côté client
 - Depuis un navigateur télécharger le fichier JNLP
 - Ou depuis une console exécuter l'outil >javaws
 - Une icône/raccouci sous windows
- Une comparaison des dates entre la version locale et distante est effectuée*

Exemple le serveur au protocole « maison »

```
import java.net.Socket;
import java.net.ServerSocket;
import java.io.*;
public class Serveur{
    public static void main(String[] args) throws Exception{
        ServerSocket serveur = new ServerSocket(5000);
        while(true) {
            Socket socket = serveur.accept();

            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String cmd = in.readLine();
            // parle !!!
            DataOutputStream out = new DataOutputStream( socket.getOutputStream());
            if(cmd.equals("parle")){
                out.write("bonjour\n".getBytes());
            }else{
                out.write("commande inconnue ?\n".getBytes());
            }
            socket.close();
        }
    }
}
```

1. Génération de l'archive serveur.jar

- Le fichier MANIFEST.MF contient ces 3 lignes
Manifest-Version: 1.0
Class-Path:
Main-Class: Serveur

Exemple suite

- Créer une signature
 - `keytool -genkey -alias jmd -keypass nsy102`

- Signer cette archive
 - `jarsigner server.jar jmd`

- Proposer le fichier JNLP, [ici serveur.jnlp](#)

```
<jnlp spec="1.0+«  
  codebase="http://jfod.cnam.fr/NSY102/serveurs/"  
  href="serveur.jnlp">  
  <information>  
    <title>Serveur Maison NSY102</title>  
    <vendor>Cnam NSY102</vendor>  
    <description>Serveur maison</description>  
    <description kind="short">dis bonjour</description>  
    <offline-allowed/>  
  </information>  
  <security>  
    <all-permissions/>  
  </security>  
  <resources>  
    <j2se version="1.5+"/>  
    <jar href="serveur.jar"/>  
  </resources>  
  <application-desc main-class="Serveur"/>  
</jnlp>
```

Exemple fin

Il suffit de cliquer ici

<http://jfod.cnam.fr/NSY102/serveurs/serveur.jnlp>

Ou bien depuis une console

javaws <http://jfod.cnam.fr/NSY102/serveurs/serveur.jnlp>

Tests avec un client telnet par exemple

telnet localhost 5000

En résumé Java Web Start

Voir le mode d'emploi

http://jfod.cnam.fr/tp_cdi/jnlp/

Et aussi

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/keytool.html>

<http://ragingcat.developpez.com/java/outils/keytool/ui/>

<http://java.sun.com/j2se/1.5.0/docs/tooldocs/windows/jarsigner.html>

Annexe 5 : URLClassLoader

```
// son propre chargeur de classe
```

```
ClassLoader loader = new MyClassLoader();
```

```
URLClassLoader urlLoader =  
    URLClassLoader.newInstance(urls, loader);
```

```
Class c = urlLoader.loadClass(program);
```

```
// par introspection recherche de la méthode main
```

```
Method m = c.getMethod("main", new Class[] {  
    args.getClass() });
```

```
m.invoke(null, new Object[] { args });
```