Les architectures à composants

Jean-Marc Farinone

Les architectures réparties : Du client-serveur aux N-tiers

- Années 80 :
  - un programme demandeur : le client,
  - un programme qui reçoit la requête et y répond : le serveur.
- Mais, le serveur (ou le client) devient trop gros
  - voire parfois, ne remplit plus sa fonction initiale mais d'autres fonction beaucoup plus lourde.
    - Exemples : serveur web qui interroge par CGI des BD (années 93 …), client trop lourd qui traite le format d'affichage des résultats
- D'où architecture 3 tiers (client, niveau intermédiaire, le système d'information de l'entreprise = SIE)
- Puis architecture N-tiers

Mais

- Dans une architecture simple client-serveur, c'est à l'application à gérer des problèmes comme :
  - la confidentialité, l'intégrité des données, l'authentification des entités bref la sécurité
  - les accès concurrents à des données communes
  - les transactions
  - ...
- Alors que ces problèmes sont connus et bien résolus depuis longtemps (moteurs transactionnels, transferts cryptés, ...)

Les technologies de composants

- Construire des entités qui répondent à certaines (petites ?!) contraintes qui vont leur permettre :
  - de bien s'intégrer à un environnement d'exécution
  - de bien s'intégrer avec d'autres entités de même "type"
- environnement d'exécution =
  - ce qui permet de traiter (exécuter ?) ces entités +
  - tout ce que l'environnement peut amener de services connus (gestion des mises à jour, un contexte commun à toutes ces entités, transaction, sécurité, …)
- entités = composants (souvent des instances ou des classes ou ...)
- environnement d'exécution = conteneurs
- Exemple de conteneurs :
  - conteneur web (pour servlets, JSP)
  - conteneur EJB (EJB)
  - conteneur de programmes pour téléphones portables (MIDlet de Java ME)

Comment utiliser tous ces outils ?

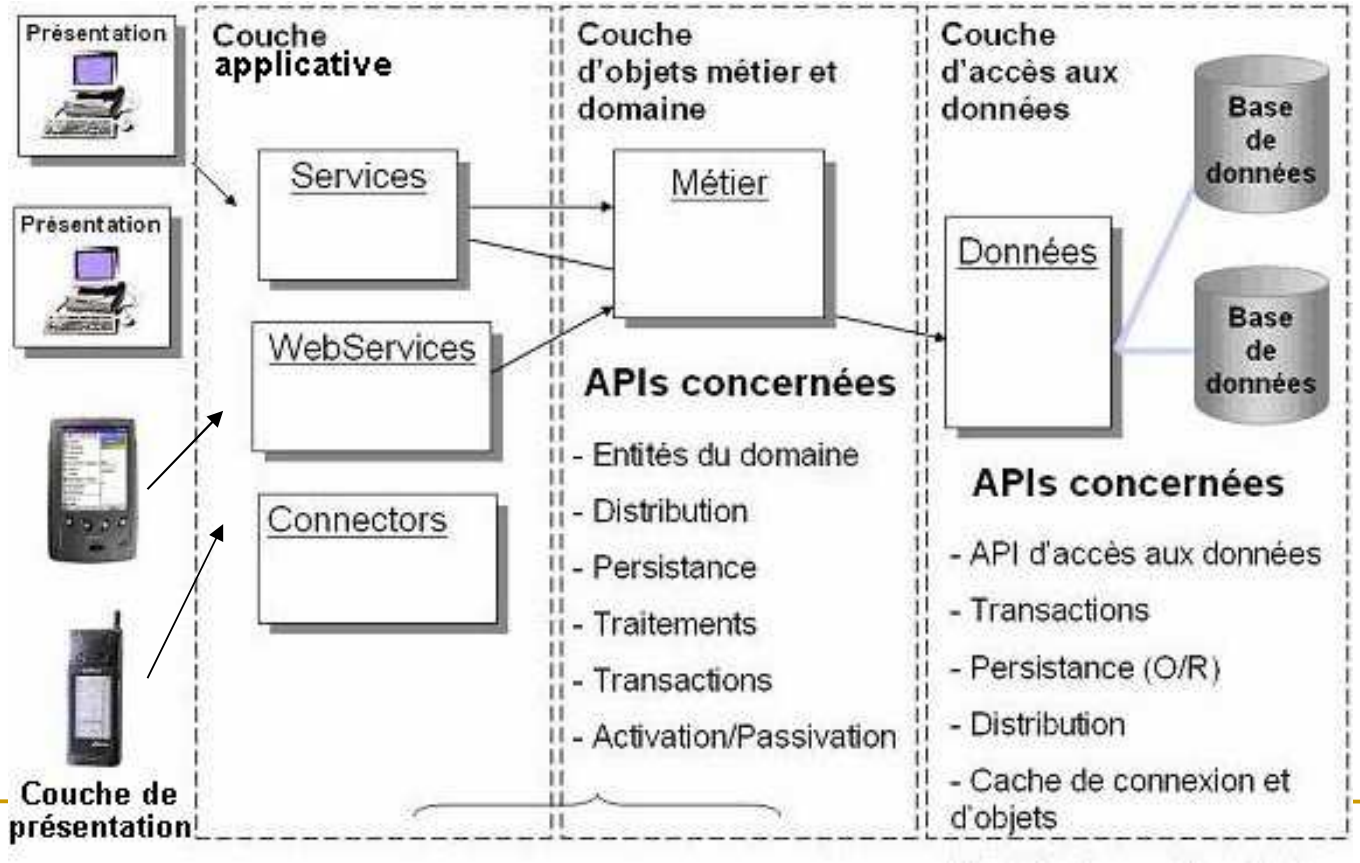
- Des règles de bons fonctionnements (bonnes écritures, bonnes utilisations, ...) ont été érigées
- Ce sont des « règles de conception » ou design pattern
- Design pattern = une solution à un problème (connu et récurrent) dans un certain contexte
- Cf. Gof Book Design Pattern Erich Gamma et al.

- Premières règles
  - Si l'interaction est simple (questions réponses) et nécessite pas d'architecture sécurisée importante (cryptage, transaction, …) utiliser seulement servlet + JSP
  - Si l'interaction est intrinsèquement complexe et nécessite obligatoirement un service d'authentification, de confidentialité, transactionel, utiliser les EJB (avec éventuellement servlet et JSP)

Résumé de l'épisode précédent

INTRODUCTION AUX ARCHITECTURES N-TIER

- **Les différentes couches d'une architecture 4-tier :**

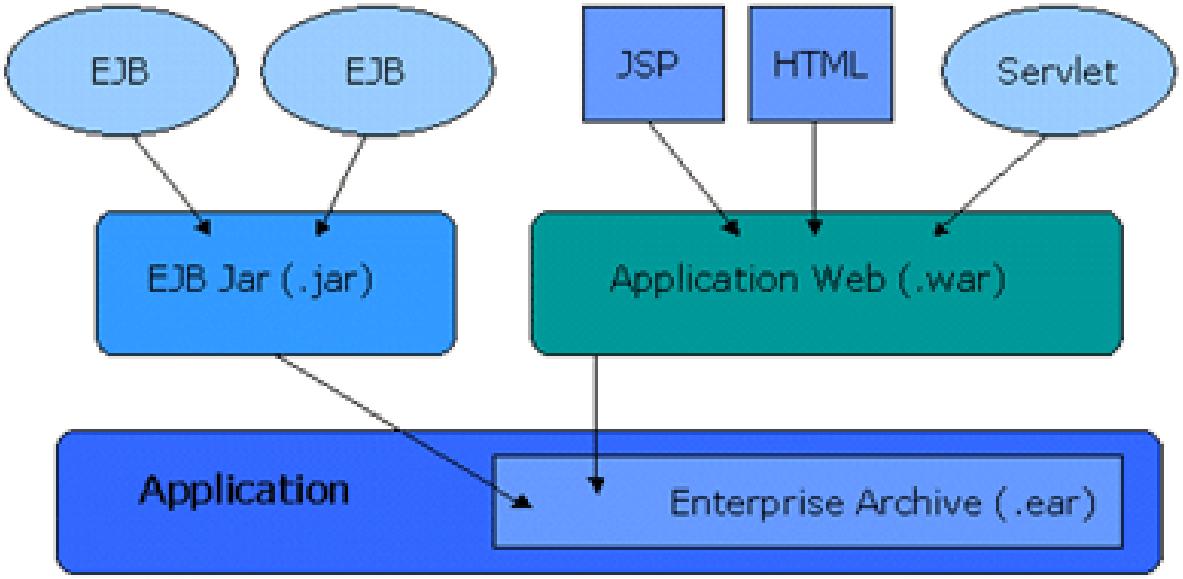


Les composants d'après Fabrice

- Objectif des composants : avoir des briques de bases réutilisables.
- Définition d'un composant :
  - module logiciel,
  - exporte différents attributs, propriétés et méthodes,
  - est prévu pour être configuré,
  - est prévu pour être installé,
  - fournit un mécanisme lui permettant de s'auto-décrire.
- Composant = objet + configurateur + installateur.

Un peu de technique

- **Architecture d'une application Java EE :**



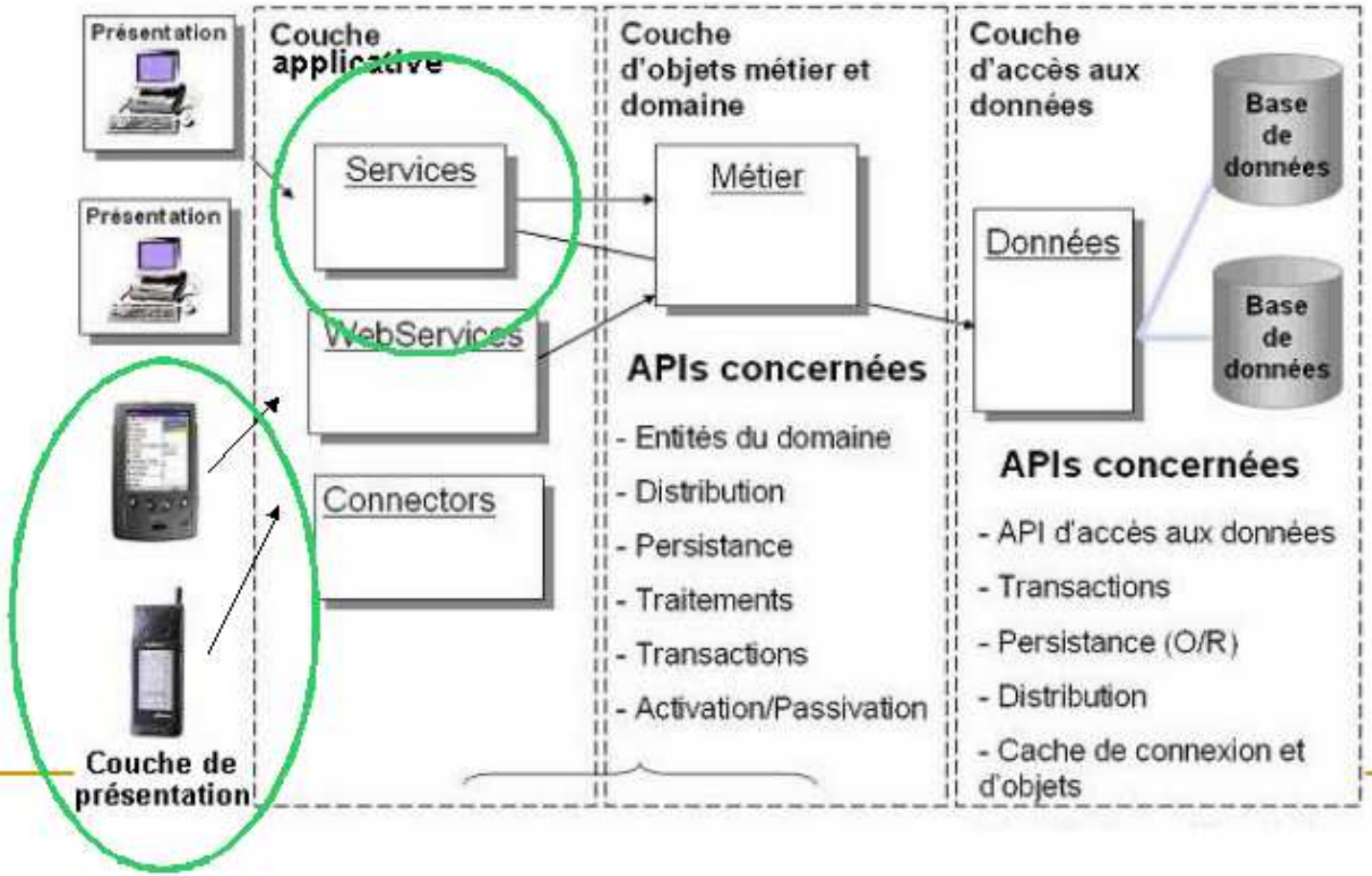
- 3 couches :
  - *Les composants.*
  - *Les modules regroupant les composants*
  - *Les applications regroupant les modules*

- Les modules et les applications correspondent physiquement à des fichiers d'archives : archive EJB JAR (.jar) pour un module EJB, archive WAR pour un module web, archive EAR pour une application.

Fin du résumé de l'épisode précédent

Le sujet de cet exposé

- **Les différentes couches d'une architecture 4-tier :**



Les composants de cet exposé

- On va faire des composants coté client MIDlet et coté serveur (application web)
- Puis architecture d'une application web avec ses composants : servlets et JSP
- Arborescence d'une appli web et son web.xml
- Architecture MVC d'une application web
- Connexion entre un téléphone portable et un serveur (application web)

Le monde des mobiles

Jean-Marc Farinone

Les téléphones mobiles au 14.01.11

- Il y a plusieurs "mondes", quasi-incompatibles entre eux :
  - l'IPhone (Objective C, environnement Apple, ...)
  - Androïd de Google (Java, notions et type de programmation différente de Java ME)
  - Symbian
  - Windows CE
- Et Java ME avec ses sous catégories :
  - MIDP
  - DoCoMo
- Et cela ne se simplifie pas au fil des ans !

Java ME : une présentation

Jean-Marc Farinone

But de cette partie

- Comprendre, définir, situer les termes :
  - Java ME, J2ME, CDC, CLDC, Configuration, Profiles, MIDP (1.0, 2.0), MIDlet, etc.
  - Donner des références
  - Donner des exemples de programmes

Les concepts fondamentaux de Java ME

Java ME = ?

- Java ME = Java Micro Edition
- Site de référence = `http://java.sun.com/javame/index.jsp`
- S'appelait anciennement J2ME : terme encore très souvent employé.
- Famille de spécifications pour développer des logiciels pour des objets électroniques (device = périphérique) comme

  - les téléphones portables,
  - les assistants personnels (PDA)
  - « téléphones intelligents » (smart phones)

Java ME = Configuration et Profiles

- Le monde des périphériques électroniques est vaste, divers et varié.
- => Pas de possibilités d'avoir un seul environnement uniforme pour tous (!= Java SE)
- => Architecture en couche :
  - Bibliothèques de base : les configurations
  - Les ajouts à ces bibliothèques : les profiles

Configuration

- = Spécifications pour un ensemble de périphériques ayant des caractéristiques similaires comme :
  - Le type et la quantité mémoire disponible
  - Le type de processeur (vitesse, etc.)
  - Le type de réseau disponible pour ce périphérique
- Configuration = plate-forme minimale pour cet ensemble. Pas d'extension ni de retrait possible
- => portabilité

Les deux configurations fondamentales

- CLDC (Connected Limited Device Configuration), CDC (Connected Device Configuration)
- CLDC ~ wireless Java.
  - Pour téléphone cellulaire, PDA ayant 192 Ko de mémoire minimum (CLDC 1.1) pour la JVM
  - Téléchargement de programmes Java
  - 2 versions 1.0 (JSR-30 Mai 2000), 1.1 (JSR-139 Mars 2003)

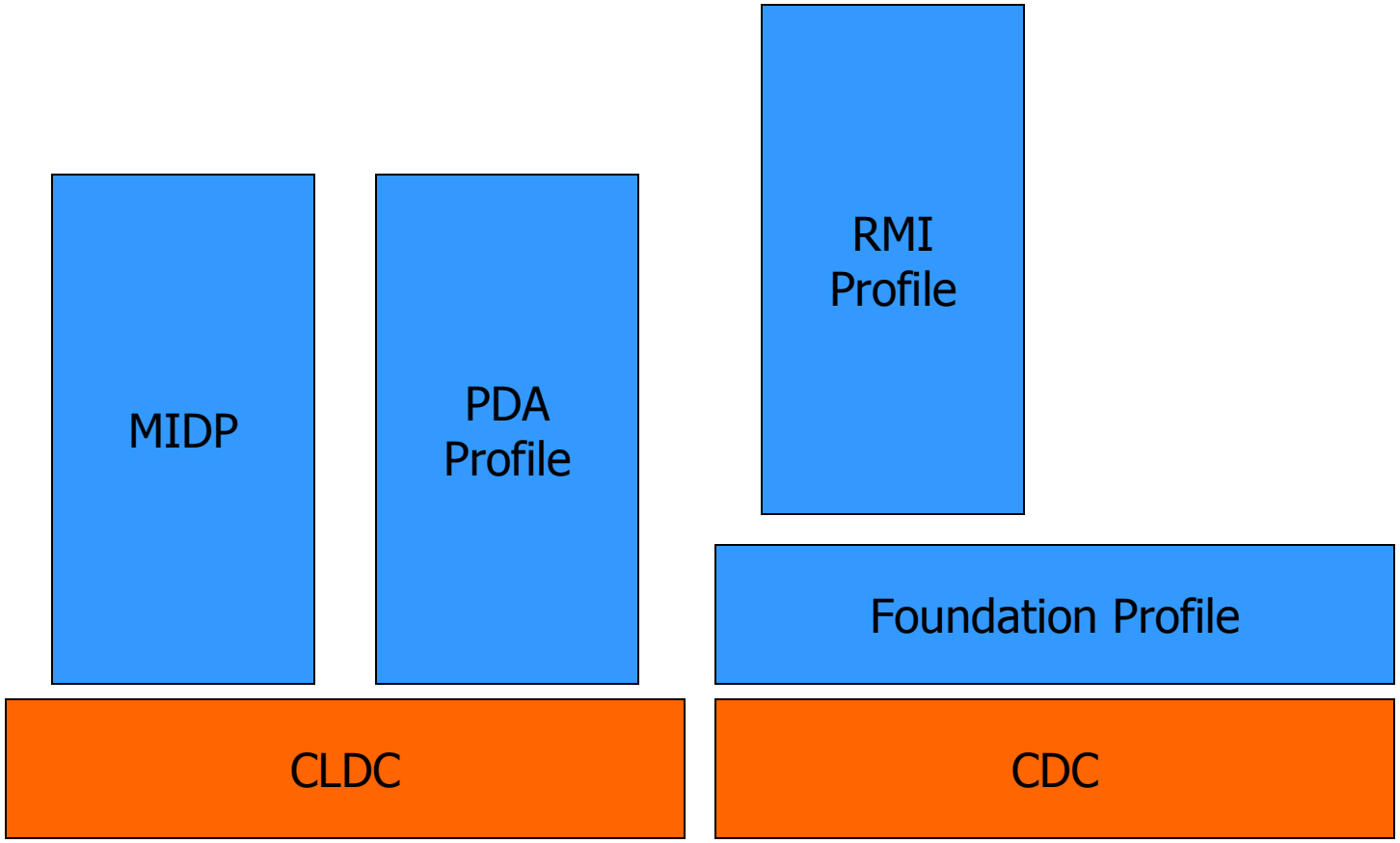
Les deux configurations fondamentales (suite)

- CDC = entre CLDC et Java SE
  - Périphériques ayant 2Mo ou plus de mémoire : smart phone, web téléphone, boitier TV (set-top boxes).

Profile

- = un complément à une configuration.
- Apporte des classes supplémentaires pour un domaine ou un marché spécifique
- Les profiles au dessus de CLDC :
  - MIDP (Mobile Information Device Profile)
  - PDA Profile
- Les profiles au dessus de CDC :
  - Foundation Profile
  - RMI Profile

Configuration et Profile : conclusion



MIDP =

- Mobile Information Device Profile
- Amène :
  - la partie réseau (+ HTTP 1.1)
  - des composants d'IHM
  - le stockage local

MIDP

Introduction

- Pas d'APIs d'interaction utilisateur, de stockage, de réseau, dans CLDC
- d'où MIDP
- applications MIDP = MIDlets
- réseau par HTTP 1.1 au moins (pas forcément TCP/IP)

MIDlet

- Dérive de la classe abstraite `javax.microedition.midlet.MIDlet`
- Doit avoir un constructeur par défaut (éventuellement donné par le compilateur)
- La MIDlet minimale :

```
// pour la classe abstraite MIDlet
import javax.microedition.midlet.*;

public class TrameJMFMIDlet extends MIDlet {
   // les 3 méthodes abstraites de MIDlet
   public void destroyApp(boolean condition)
        throws MIDletStateChangeException {}
   public void pauseApp() { }
   public void startApp() throws MIDletStateChangeException {}
   public TrameJMFMIDlet(){ }
}
```

Développer une MIDlet

- Installer Java SE.
- Charger gratuitement l'environnement "Wireless toolkit" à partir de `http://java.sun.com/products/j2mewtoolkit/index.html`
- Eventuellement être inscrit au Download Center.

Développer une MIDlet (suite)

- Lancer la Ktoolbar (soit par windows soit par des commandes en ligne)
- Créer un projet (New Project). Donner un nom de projet, le nom de la classe MIDlet (ici `PremiereJMFMIDlet`, cf. diapo suivante) Cliquer "Create Project". =>
  - 1°) Les infos du `.jad` sont affichées.
  - 2°) un répertoire du nom du projet a été créé sous l'environnement wireless toolkit.
- On placera sources, ressources, etc. dans ce répertoire.
- Début de la demo

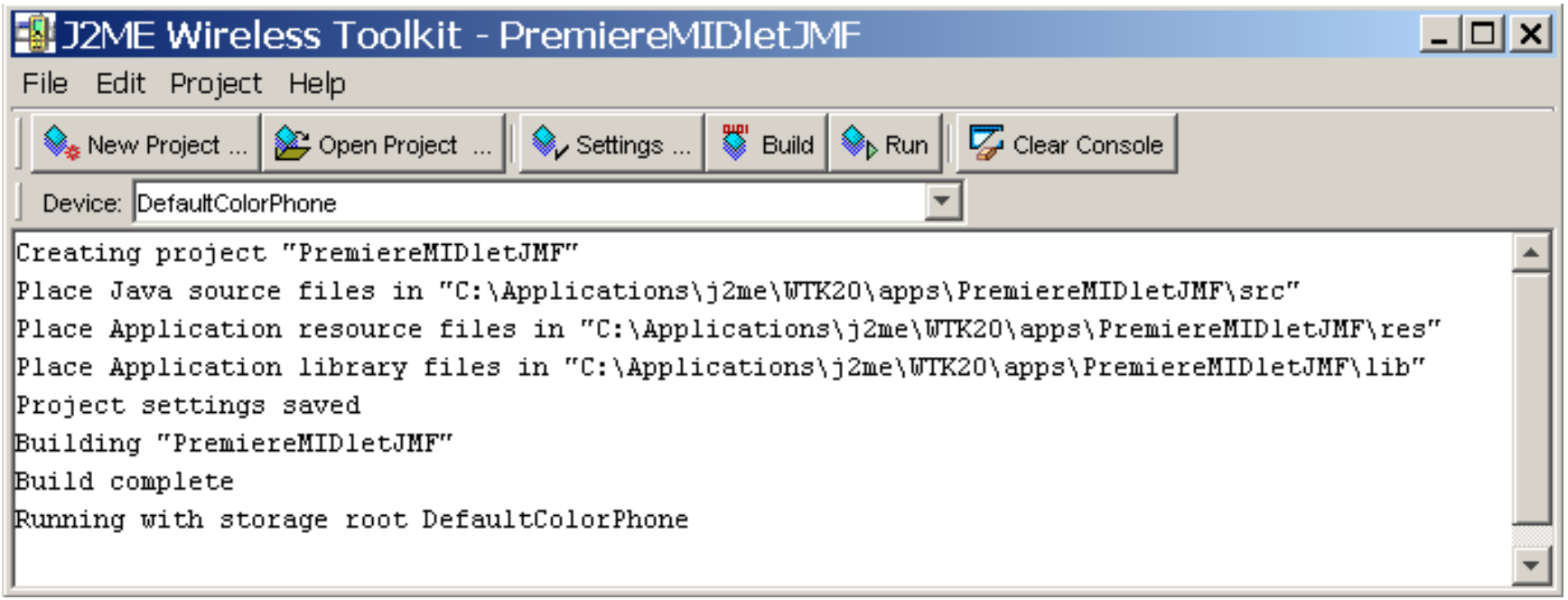
Développer une MIDlet (suite)

- Code de la MIDlet à sauvegarder dans `src`

```
import javax.microedition.midlet.*;
// pour CommandListener
import javax.microedition.lcdui.*;
public class PremiereJMFMIDlet extends MIDlet implements CommandListener {
   // les 3 méthodes abstraites de MIDlet
   public void destroyApp(boolean condition){}
   public void pauseApp(){}
   public void startApp(){
        Display.getDisplay(this).setCurrent(mMainForm);
   }
   // La methode de l'interface CommandListener
   public void commandAction(Command c, Displayable d) {}
   public PremiereJMFMIDlet() {
        mMainForm = new Form("Ma Premiere MIDlet JMF");
        mMainForm.append(new StringItem(null, "Bonjour à tous"));
        mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
        mMainForm.setCommandListener(this);
   }
   private Form mMainForm;
}
```

Développer une MIDlet (fin)

- Cliquez Build. L'environnement a :
  - Créer les répertoires `classes`, `tmpclasses`.
  - Compiler les sources Java, résultat dans `tmpclasses`
  - Prévérifier ces `.class` et mis dans `classes`
  - Construit les `.jar` et ajuste le `.jad`



Exécuter la MIDlet

- Cliquer "Run"
- Changer de périphérique par Device (QwertyDevice)
- Une demo : OK !
- Plus de code ...
- … au chapitre suivant (programmation réseau avec MIDP)

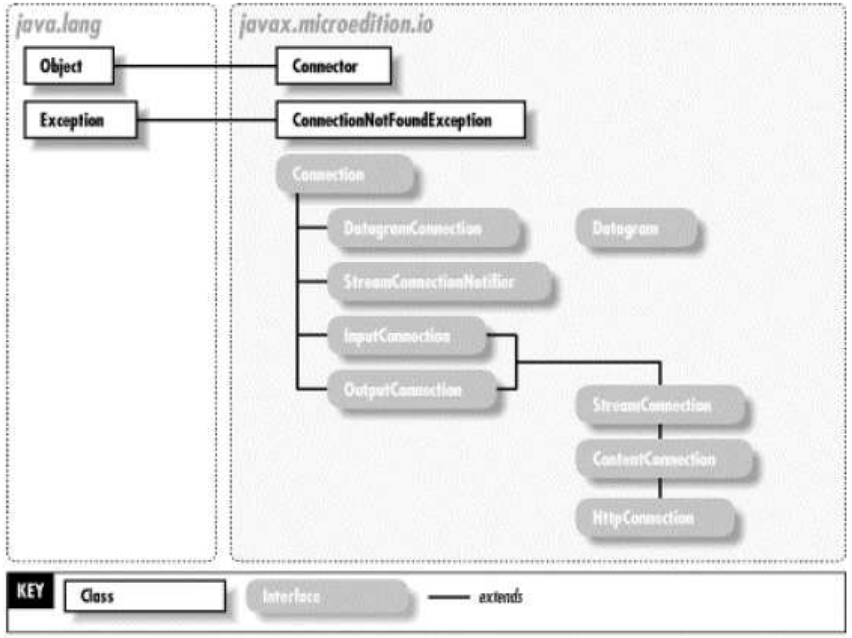
Programmation réseau pour Java ME (CLDC, MIDP)

Jean-Marc Farinone

Présentation

- Java SE contient des classes pour les protocoles TCP, UDP, IP (`java.net`) et aussi RMI, CORBA, JINI, etc.
- En Java ME, CLDC/MIDP 1.0, on ne peut avoir tout cela.
- La programmation réseau de CLDC est gérée par le Generic Connection Framework (GCF)
- GCF utilise le package `javax.microedition.io`

Classes, interfaces, exceptions de `javax.microedition.io`



- source J2ME in a nutshell (Kim Topley, ed O'Reilly)
- Donc finalement essentiellement des interfaces !

Les protocoles de communication pour mobiles

- Les mobiles (en fait les fournisseurs de réseau pour mobiles) ne fournissent pas toujours de connexion socket ou TCP (les émulateurs si ;-))
- Mais cela peut exister
  - sur les PDA
  - en payant (très ?) cher
- Ce qui est supporté est l'accès HTTP.

Le protocole HTTP

- Le seul protocole réellement supporté dans MIDP
- Car en général, les mobiles n'ont pas de communication directe à "l'internet (TCP)"
- rappel : HTTP est un protocole requête réponse
- Il suffit d'indiquer l'URL et chaque `getXXX()` récupère immédiatement la réponse.

Plan pour une communication socket

- 1°) Obtenir une connexion
- 2°) Récupérer les canaux d'écriture et de lecture sur cette connexion
- 3°) Envoyer une requête, récupérer et traiter la réponse
- 4°) Fermer la connexion
- Remarque : ce plan fonctionne si c'est le mobile qui est initiateur de la connexion

1°) Obtenir une connexion

```
StreamConnection socket;
try {
    String server = ...
    String port = ...
    String name = "socket://" + server + ":" + port;
    socket = (StreamConnection)Connector.open(name,
Connector.READ_WRITE);
} catch (Exception ex) { ... }
```

2°) Récupérer les canneaux d'écriture et de lecture sur cette connexion

```
OutputStream os = null;
InputStream is = null;

os = socket.openOutputStream();
is = socket.openInputStream();
```

3°) Envoyer une requête, récupérer et traiter la réponse

```
String request = "GET /index.html HTTP/1.0\n\n";
os.write(request.getBytes());

...

// On lit au plus 128 octets
final int MAX_LENGTH = 128;
byte[] buf = new byte[MAX_LENGTH];
int total = 0;
while (total < MAX_LENGTH) {
    int count = is.read(buf, total, MAX_LENGTH - total);
    if (count < 0) { break; }
    total += count;
}

String reply = new String(buf, 0, total);
```

4°) Fermer la connexion et demo

```
os.close();
is.close();
socket.close();
```

- source J2ME in a Nutshell, Kim Topley
- demo Socket dans le WTK, projet Chapter 6, MIDlet socket. Le vérifier :
  - en lançant tomcat localement
  - en se connectant dans la MIDlet socket sur le serveur `localhost` port `8080`,
  - en le vérifiant par `telnet localhost 8080` et lancée de la requête **`GET /index.html HTTP/1.0\n\n`**

Programmation pour le protocole HTTP

```
HttpConnection hc = null;
InputStream in = null;
String url = "http://localhost:8080/midp/hits";

try {
  hc = (HttpConnection)Connector.open(url);
  in = hc.openInputStream();

  int contentLength = (int)hc.getLength();
  byte[] raw = new byte[contentLength];
  int length = in.read(raw);

  in.close();
  hc.close();

  // traite la réponse
  String s = new String(raw, 0, length);
  ...
}
catch (IOException ioe) { ... }
```

Programmation réseau et multithreading

- De manière classique, le code de la partie réseau est (doit être !) mis dans une thread à part (sinon toute cette partie peut bloquer la MIDlet ou son IHM). On a plutôt un code comme :

```
public void commandAction(Command c, Displayable d) {
    ...
    if (c == cmdTransaction) {
        Thread t = new Thread(this);
         t.start();
     Display.getDisplay(this).setCurrent(connectForm);
    }
    public void run() {
        envoiRequete();
    }
    public void envoiRequete() { ... }
```

Bibliographie

- Java development on PDAs. Daryl Wilding-McBride ; éditions Addison-Wesley
- J2ME in a nutshell. Kim Topley ; éditions O'Reilly
- J2ME Wireless Toolkit 2.1 Download à

  `http://java.sun.com/products/j2mewtoolkit/download-2_1.html`
- J2ME, applications pour terminaux mobiles. Bruno Delb ; éditions Eyrolles
- Excellent tutorial des divers APIs de MIDP 2.0 à `http://developers.sun.com/mobility/learn/midp/midp20/`

Et le coté serveur ?

- Une application web (servlets, JSP, HTML, etc.)
- Donc supposé connu pour cette UE SMB111 du CNAM !

Déploiement d'application web

Jean-Marc Farinone

application web = ?

- Les servlets sont une des techniques utilisées pour construire des applications web.
- "A web application is a dynamic extension of a web or application server. There are two types of web applications:
  - Presentation-oriented: A presentation-oriented web application generates interactive web pages containing various types of markup language (HTML, XML, and so on) and dynamic content in response to requests.
  - Service-oriented: A service-oriented web application implements the endpoint of a web service."
- [`http://java.sun.com/javaee/5/docs/tutorial/doc/bnadr.html`]

application web = ?

- Euh en français :
- "Une application web est une extension dynamique d'un serveur web ou applicatif. Il y a deux types d'applications web :
  - les application web orientées présentation qui génèrent des pages web (HTML, XML) dynamiquement
  - les applications web orientées service : ce sont les web services
- [`http://java.sun.com/javaee/5/docs/tutorial/doc/bnadr.html`]

servlet = ?

- Une servlet est un programme (plug-in) à ajouter à un serveur (quel qu'il soit).
- Ce cours a trait à la programmation Java coté serveur (Java EE)
- Pour l'instant les serveurs acceptant des servlets sont plutôt des serveurs Web.
- Contre-exemple : une servlet pour un serveur de mail qui détruit les mails contenant des virus.

Rappel : une servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MaPremiereServlet extends HttpServlet {

     public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {

          // Etape 1.  Spécifier le type MIME du contenu de la réponse
          response.setContentType("text/html");

          // Etape 2.  Récupère le PrintWriter pour envoyer des données au client
          PrintWriter out = response.getWriter();

          // Step 3.  Envoyer l'information au client
          out.println("<html>");
          out.println("<head><title>Bonjour Servlet</title></head>");
          out.println("<body>");
          out.println("<h1> Bonjour à tous </h1>");
          out.println("Il est : " + new java.util.Date());
          out.println("</body></html>");
     }
     public void doPost(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
              doGet(request, response);
     }
}
```

Moteurs de servlets (et de JSP)

- Pour exécuter des servlets (resp. des JSP), il faut un moteur de servlets (resp. de JSP) dans le serveur Web.
- Ces moteurs sont des plug-in pour des serveurs Web existants
- Souvent des serveurs Web eux mêmes
- Un bon candidat plug-in : tomcat (`http://tomcat.apache.org/`)

Serveurs Web et servlets

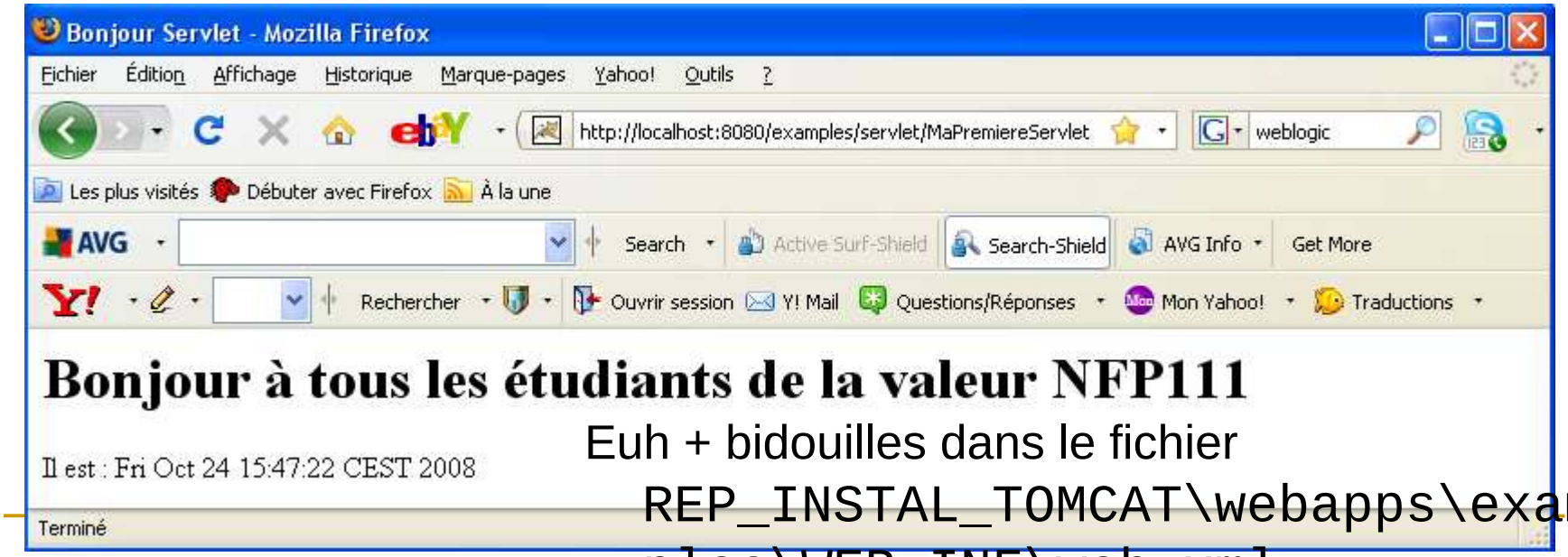
- Il existe des serveurs Web qui traitent les servlets (et JSP) :
  - IBM WebSphere
  - BEA WebLogic Server 10
  - JBoss
- Voir à `java.sun.com/products/servlet`

Tomcat

- Développé par la communauté qui implémente les spécifications des servlets et JSP.
- Téléchargeable (en version d'utilisation élémentaire) gratuitement à `http://tomcat.apache.org/download-60.cgi`
- Plug-in de Apache, Microsoft IIS, …
- Est aussi un mini-serveur Web.
- "Apache Tomcat 6.x is the current focus of development. It implements the Servlet 2.5 and JSP 2.1 specifications."

Démonstration

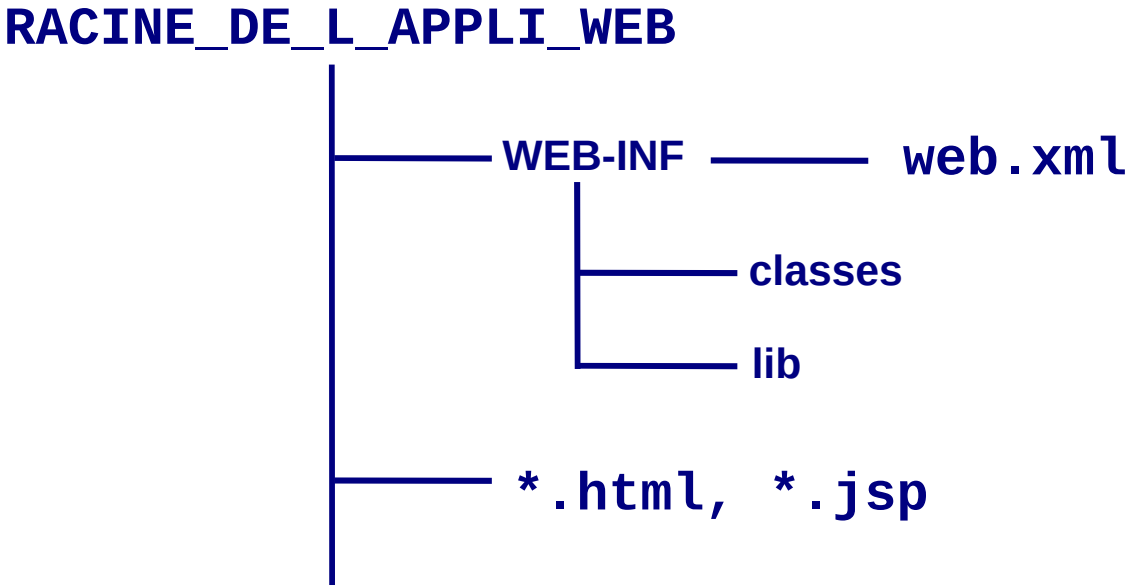
- On lance le serveur Web
- La servlet compilée est rangée sous `REP_INSTAL_TOMCAT\webapps\examples\WEB-INF\classes`
- correspond à l'URL : `http://localhost:8080/examples/servlet/MaPremiereServlet`



Euh + bidouilles dans le fichier `REP_INSTAL_TOMCAT\webapps\examples\WEB-INF\web.xml`

Rappel : architecture d'une application web

- Une application web doit avoir la structure suivante :



- On peut mettre tout cela dans un fichier compressé : un `.war`

Un exemple de `web.xml`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
<display-name>Servlet d'accueil</display-name>
<description>La toute première servlet d'accueil du site</description>

<servlet>
     <servlet-name>unNomQuelconque</servlet-name>
     <servlet-class>ExoServlet</servlet-class>
</servlet>

<servlet-mapping>
     <servlet-name>unNomQuelconque</servlet-name>
     <url-pattern>/ExoServlet</url-pattern>
</servlet-mapping>

<welcome-file-list>
     <welcome-file>debut.html</welcome-file>
</welcome-file-list>

</web-app>
```

Compilation des servlets

- On compile les servlets par :
  `javac -d ../classes ExoServlet.java`
- Mais il faut avoir les classes servlets (`HttpServlet`, ...) donc positionner avant :
  `set CLASSPATH=%TOMCAT_HOME%\lib\servlet-api.jar;%CLASSPATH%`
- Se faire un script !

Déploiement des servlets (1/3)

- C'est un peu plus complexe que les pages JSP et HTML car il faut les placer sous `WEB-INF/classes`
- D'abord construire le `web.xml` (qui sera dans `WEB-INF`) :
- Voir celui distribué

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
<display-name>Appli Web Certificat CNAM</display-name>
<description>Description de votre application web</description>

<servlet>
     <servlet-name>unNomQuelconque</servlet-name>
     <servlet-class>UnNomDeClasseServlet</servlet-class>
</servlet>

<servlet-mapping>
     <servlet-name>unNomQuelconque</servlet-name>
     <url-pattern>/UneArborescencePourlURL</url-pattern>
</servlet-mapping>

<welcome-file-list>
     <welcome-file>debut.html</welcome-file>
</welcome-file-list>

</web-app>
```

Déploiement des servlets (2/3)

- Ce qui est important pour la servlet sont les lignes :

```
<servlet>
     <servlet-name>unNomQuelconque</servlet-name>
     <servlet-class>UnNomDeClasseServlet</servlet-class>
</servlet>

<servlet-mapping>
     <servlet-name>unNomQuelconque</servlet-name>
     <url-pattern>/UneArborescencePourlURL</url-pattern>
</servlet-mapping>
```

Déploiement des servlets (3/3)

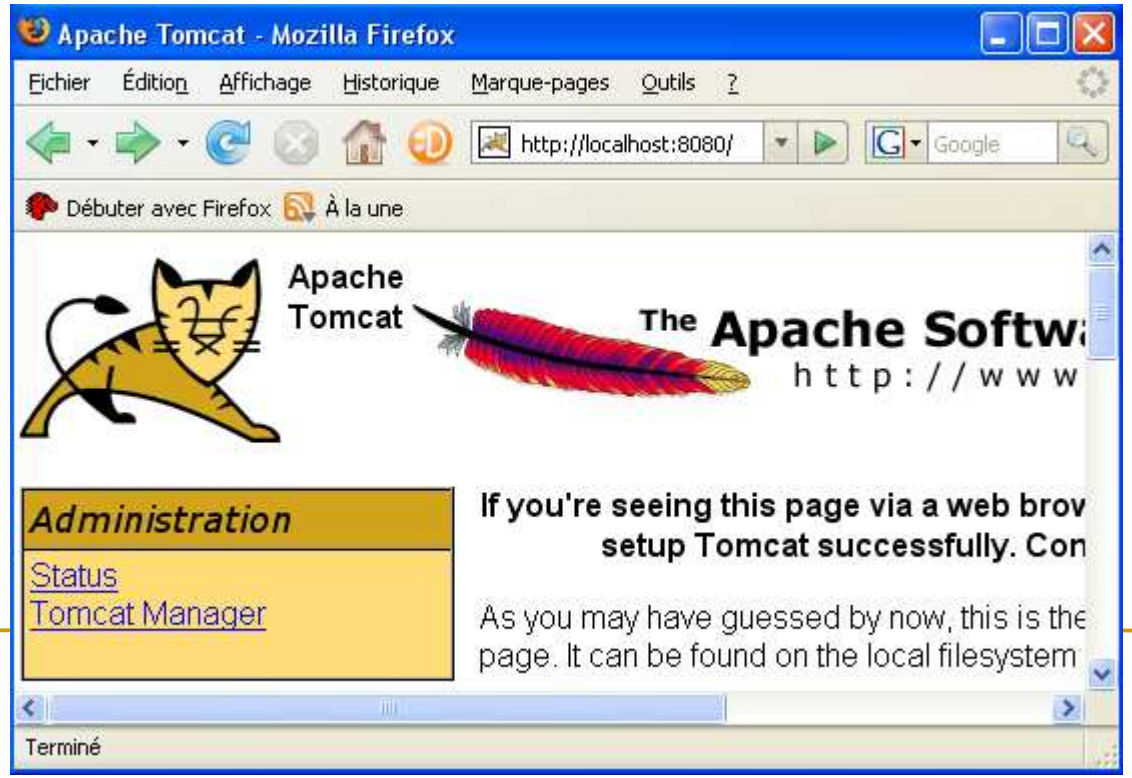
- Il ne reste plus qu'à construire le `.war` par exemple par

```
jar cvf exo1.war *.html
 WEB-INF\classes\*.class
 WEB-INF\lib
 WEB-INF\web.xml
```

- Puis à déployer ce `.war` par "Tomcat Manager"

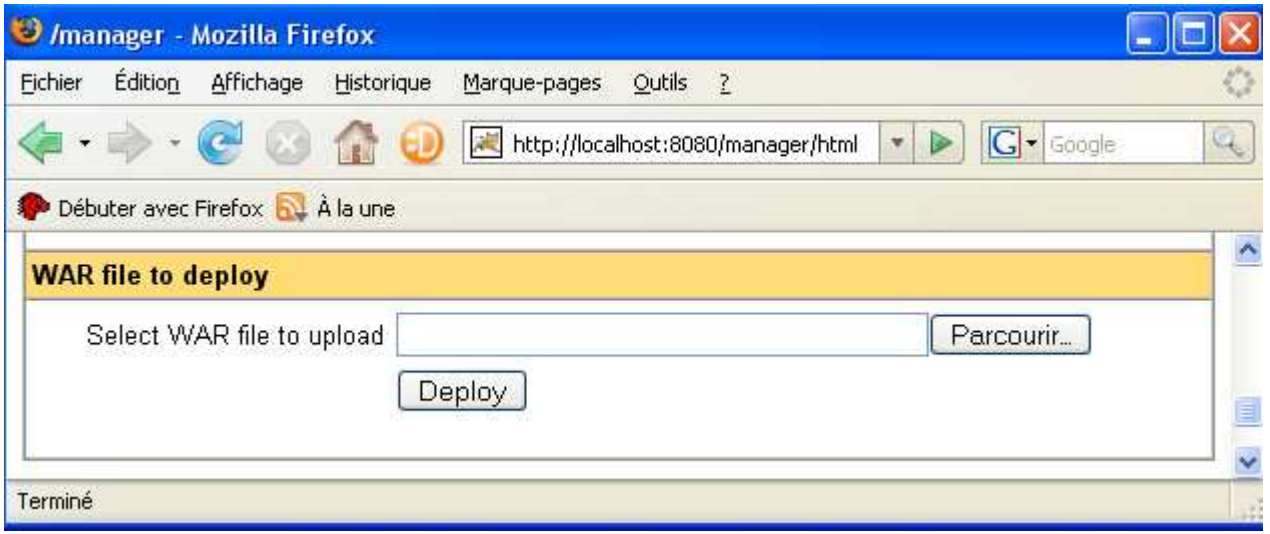
Déployer l'application web avec Tomcat Manager (1/2)

- Il faut construire un `.war` par `jar cvf NOTRE_SITE.war repEtRessources`
- Se connecter sur `http://localhost:8080/` et cliquer sur le lien "Tomcat Manager".



Déployer l'application web avec Tomcat Manager (2/2)

- Dans la page "Gestionnaire d'applications WEB Tomcat" obtenue, "descendre" jusqu'à "WAR file to deploy", puis suivre les indications (Parcourir puis Deploy) pour déployer le `.war`.



- Si le fichier déployé a pour nom `NOTRE_SITE.war`, on accède à ce site par des URL commençant par `http://localhost:8080/NOTRE_SITE`

JavaServer Pages (JSP)

Notre première JSP

- fichier `MaDate.jsp`

```
<html><head><title>Obtenu par une JSP</title></head>
<body>

<h3>Bonjour de ma part </h3> <hr>
La date courante est :  <%= new java.util.Date() %>
</body>
</html>
```

- Traité quand le client demande l'URL de la JSP : `http://serveurWeb:<port>/.../MaDate.jsp`

Comment ça marche ?

- Concrêtement :
  - toute la page JSP est convertie en une servlet
  - cette servlet est traitée par le moteur Java intégré au serveur Web (technologie des servlets) et retourne la page HTML construite

JSP vs. Servlets

- Servlet = du code Java contenant de l'HTML
- JSP = une page HTML contenant du code Java
- Concrètement avec les JSP :
  - les parties statiques de la page HTML sont écrites en HTML
  - les parties dynamiques de la page HTML sont écrites en Java

Exécution de JSP

- Il faut mettre les pages JSP dans un endroit particulier du serveur Web
- Cet endroit dépend du serveur Web et de sa configuration
- Pour tomcat en configuration standard,
  `http://localhost:8080/examples/jsp/MaDate.jsp`
  ~
  `REP_INSTAL_TOMCAT\webapps\examples\jsp\MaDate.jsp`
  pour tomcat 6.0

- Et sans bidouille !!

Exécution de JSP (suite)

- Une démo:
- Le résultat de `MaDate.jsp` est :

Bonjour de ma part

La date courante est : Sun Dec 23 18:04:36 CET 2001

- Une autre exécution donne une autre date => dynamicité

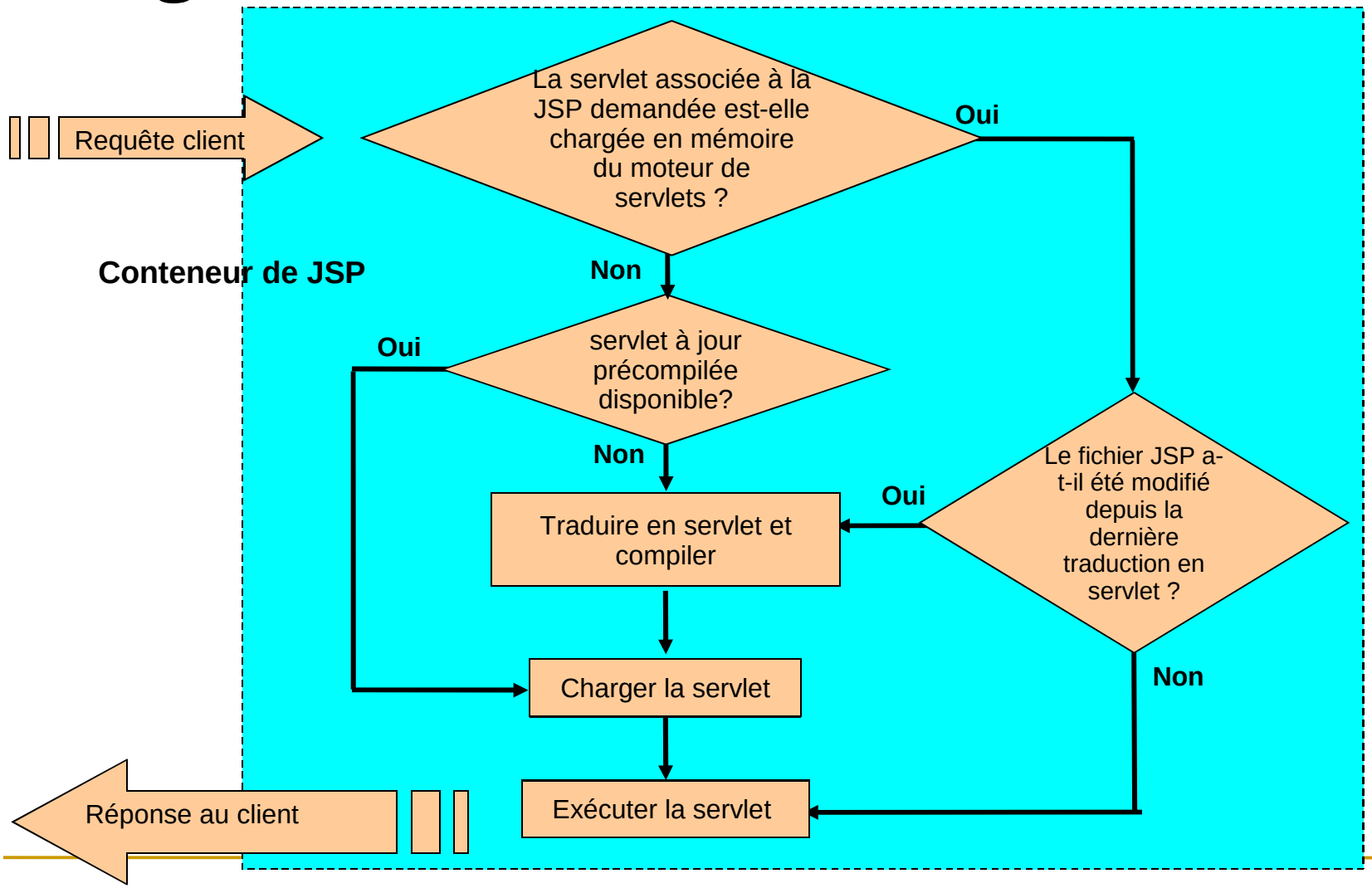
Que s'est il passé ?

- Le moteur de JSP a construit une servlet (`MaDate_jsp.java` sous l'arborescence `work` pour Tomcat 6.0)
- Cette phase est parfois appelée la traduction de la JSP (en servlet)
- Puis a compilé et exécuté la servlet

La servlet construite

```
package org.apache.jsp;
...
public class MaData_jsp extends HttpJspBase {
     ...
    public void _jspService(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException {
       ...
            pageContext = _jspxFactory.getPageContext(...);
            session = pageContext.getSession();
            out = pageContext.getOut();
            // HTML
   // begin [file="C:\\...\\examples\\jsp\\MaDate.jsp";from=(0,0);to=(4,24)]
         out.write("<html><head><title>Obtenu par une JSP</title></head>\r\n
<body>\r\n\r\n<h3>Bonjour de ma part</h3> <hr>\r\n
     La date courante est :  ");
     // end
//begin [file="C:\\...\\examples\\jsp\\MaDate.jsp";from=(4,27)to=(4,49)]
                out.print( new java.util.Date() );
            // end
            // HTML
// begin [file="C:\\...\\examples\\jsp\\date.jsp";from=(4,51);to=(6,7)]
                out.write("\r\n</body>\r\n</html>"); // end
            ...
    }
}
```

Algorithme d'exécution de la JSP



3 parties d'une JSP

- scriptlets <% %>
- déclarations <%! %>
- expressions <%= %>

En fait, servlet vs. JSP ?

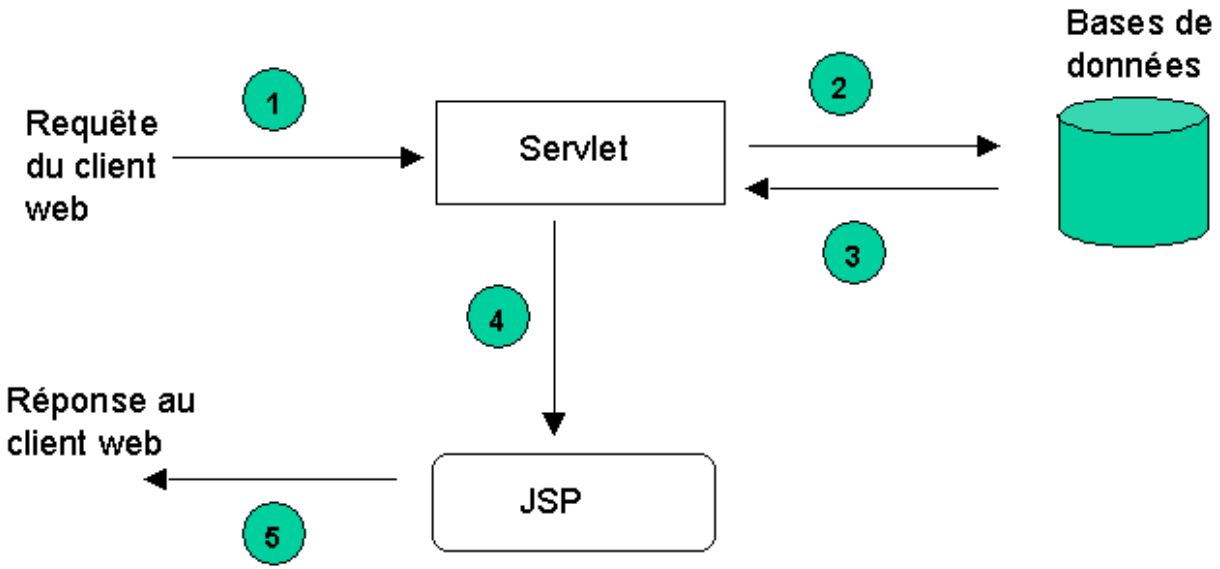
- A priori deux outils pour répondre aux mêmes besoins
- D'ailleurs on pourrait tout faire avec, pour chaque interaction une seule JSP à consulter : le modèle 1
- Ou, pour chaque interaction, consulter une page JSP construite à l'aide d'autres pages JSP (ou HTML) : le modèle 3
- En fait complémentaires : le modèle 2 (MVC)

Architecture MVC

- modèle = les données accédées par un code Java (JDBC, RMI, EJB, etc.)
- vues = JSP
- contrôleur = servlets

L'architecture MVC coté serveur
Le modèle 2

- modèle = les données accédées par un code Java (JDBC, RMI, EJB, etc.)
- vues = JSP
- contrôleur = servlets



Communication téléphone - application web

= Dialogue MIDlet-servlet : une démo

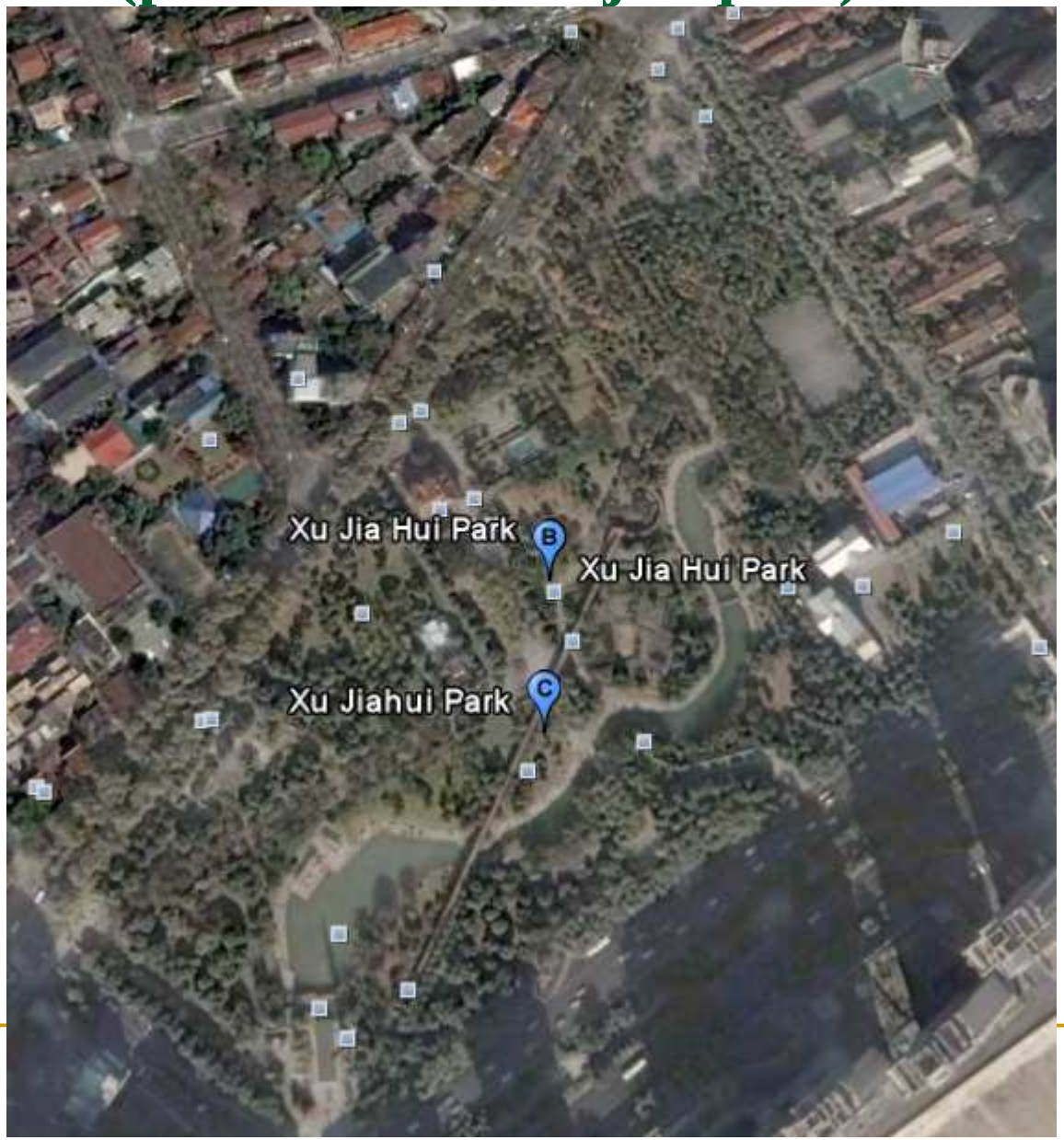
- Voir à `http://developers.sun.com/mobility/midp/articles/tutorial2/` d'une communication mobile sur une servlet par Jonathan Knudsen
- Retranscrit avec la MIDlet `HitMIDlet` de la suite HelloSuite se connectant à l'application web (la servlet) d'URL `http://localhost:8080/midp/hits`
- Le vérifier avec un navigateur
- Code de la MIDlet dans le répertoire `...\j2mewtk\2.5.2\apps\HelloSuite\src`
- Code de la servlet dans le fichier `HitServlet.java` dans le répertoire `...\SMB111...\annee1011\Conf2JavaMeEtServeurs\servletPourExempleHTTP`

Enhanced Go

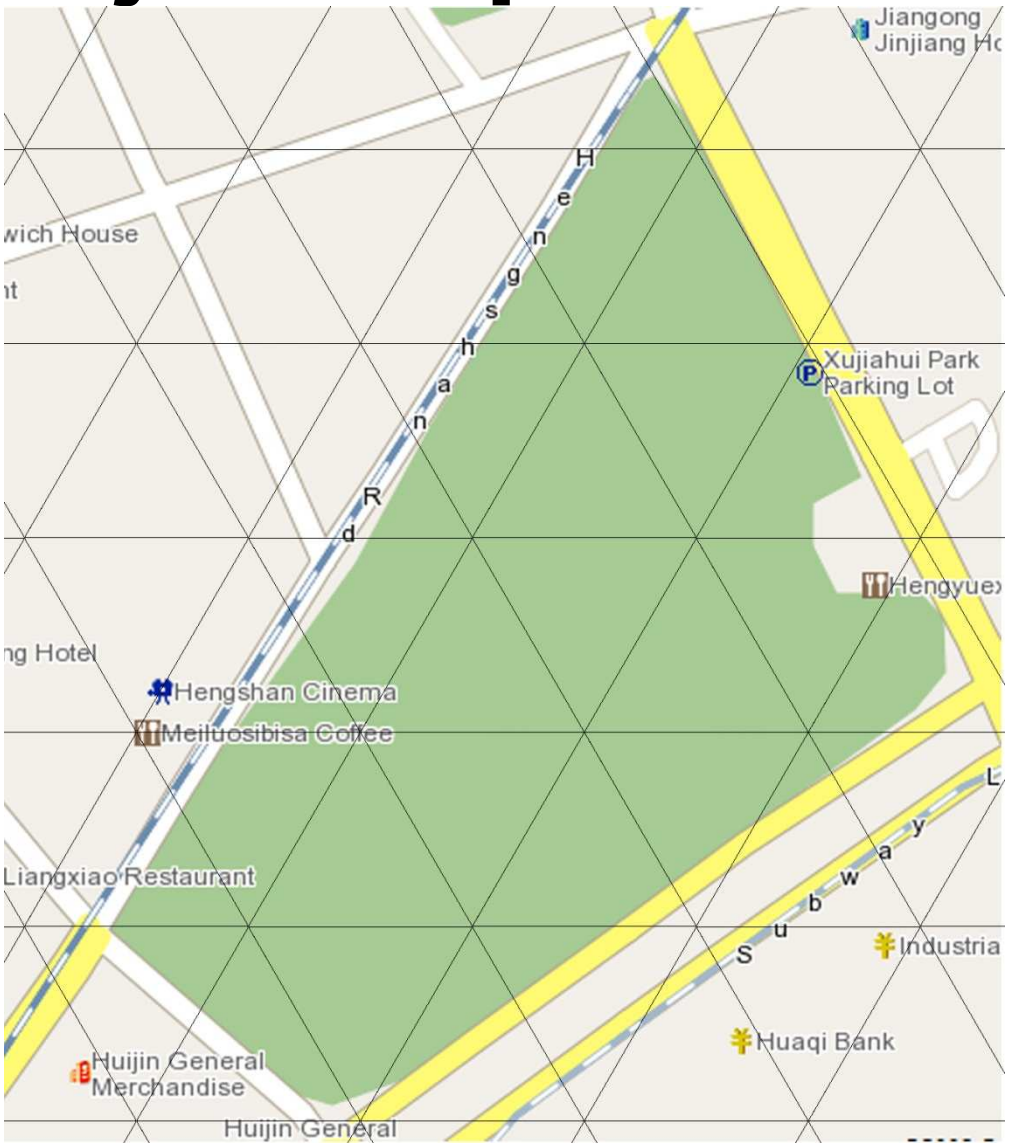
Un jeu urbain
Congrès eArts Festival
Shanghai 2008

Chen Yan
Nicolas Rempulski
Jean-Marc Farinone

Le parc Xu Jai Hui
(prononcer Chu jia qouè)

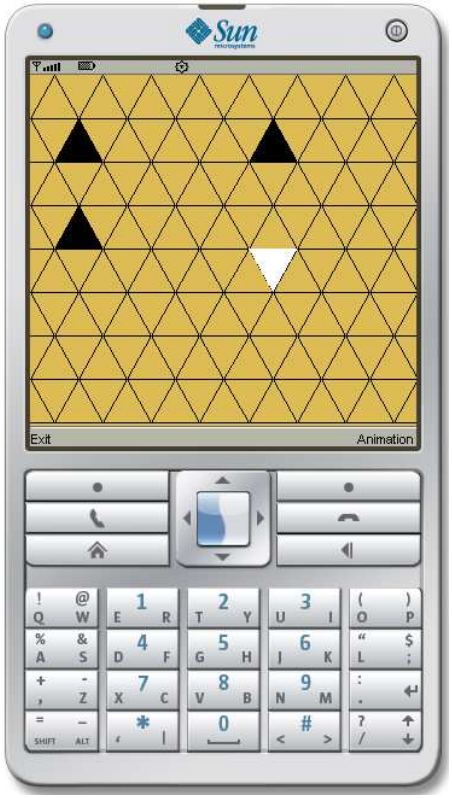
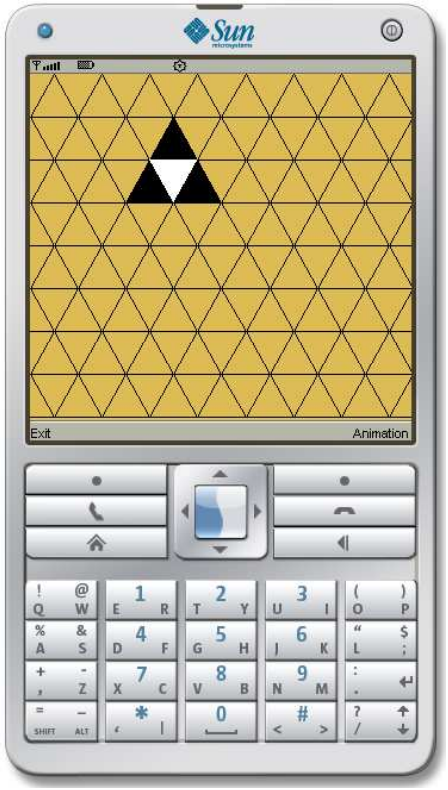


Le plateau de jeu = le parc !



- Bon en mieux :
- `1decouverteParc.mov`

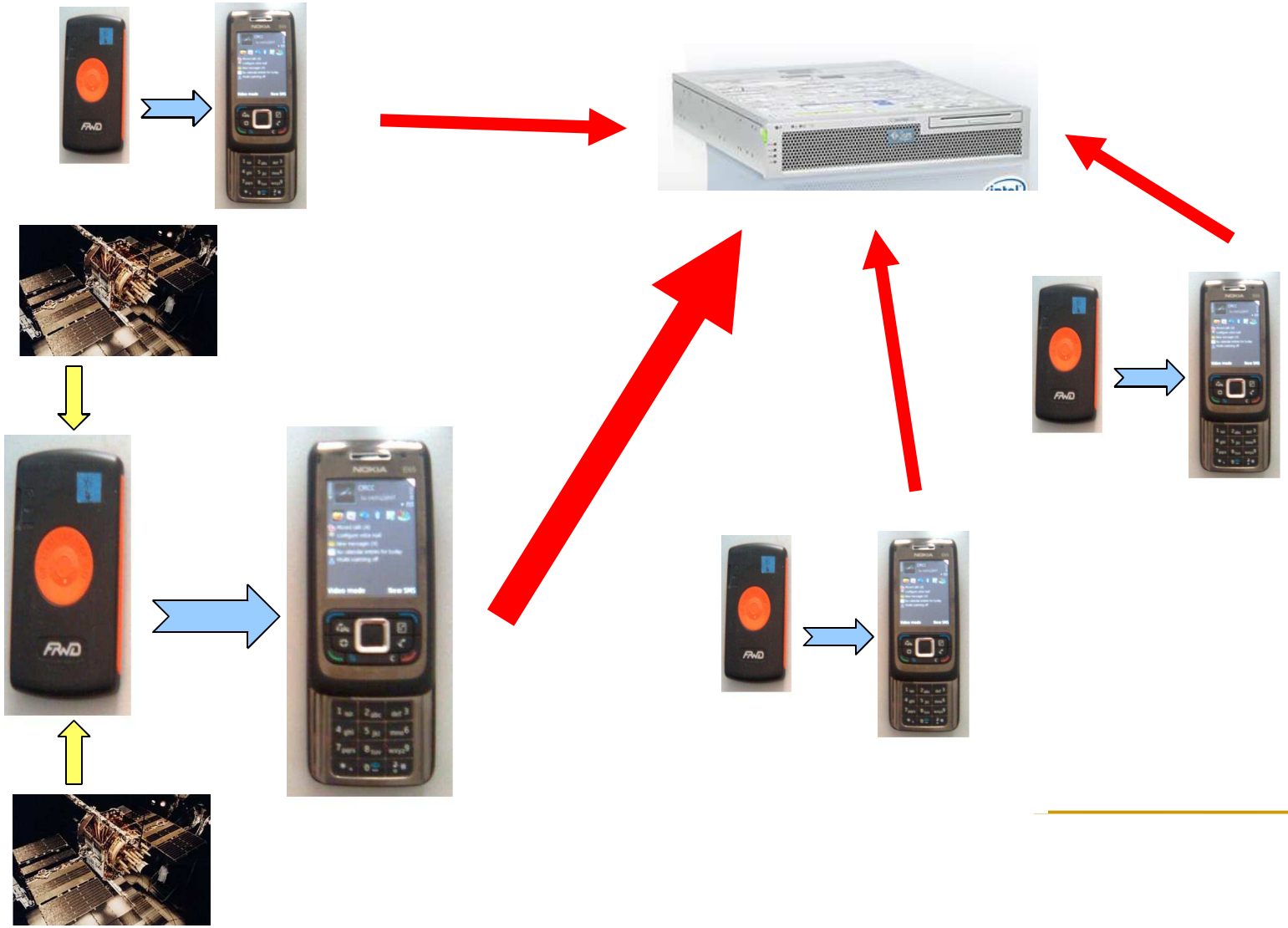
La règle du jeu

De  à 

une demo : OK

Dans eclipse `projet GoBanDemo`

L'architecture matérielle et réseau



L'architecture matérielle et réseau

Une peu de technique

- Sur les téléphones portables Java 1.4, CLDC 1.1, MIDP 2.0
- Sur le serveur : Java 1.5
- + réseau (Bluetooth, HTTP)

Démarrage du jeu

Le jeu en cours

- Contour de l'aire de jeu
- Numéro de joueur
- Score personnel et des équipes

- Une vidéo ? bon d'accord !
- `2PartieDuFestival.mov`

Le mot de la fin

- A Adrien Rappoport
- Et aux bénévoles de E Arts
- `3essaigoaugmenteok.mov`

Bibliographie

- GASP, Romain Pellerin
  `http://gasp.objectweb.org/`
- L'aventure au jour le jour le jour (site disparu ?)
  `http://www.adigitalexperience.com` et par exemple
  `http://www.adigitalexperience.com/exposition/paroles_penseurs.php?id=45`

Fin