Java et le Web.

Thème :

code mobile et les nouvelles architectures applicatives.

quoi?

pour quoi faire?

comment faire?

mots-clés:

- applet, servlet, JDBC, RMI, Corba,
- programmation distribuée

code mobile.

définition :

programme compilé (code), portable et transportable

transportable: lieux de stockage et d›exécution différents
portable: indépendant de la plate-forme.

Pour quoi faire?

- ne plus gérer les configurations des postes clients,
- distribuer facilement les applications et leurs versions successives.

Java = code mobile + sécurité.

architectures applicatives.

Type	Activité sur le serveur	Activité sur le client	Description
0	Données Traitements Présentation		serveur : application Java avec JDBC client : via telnet par exemple
1	Données Traitements Présentation	Présentation	serveur : application Java avec JDBC client : navigateur (documents sans applet)
2	Données Traitements	Présentation	serveur : servlet avec JDBC client : navigateur (documents avec applet) exemple : network computer

Table 1 Typologie de la répartition des activités

commentaires:

architectures classiques (Java non nécessaire)

nouvelles architectures applicatives.

Type	Activité sur le serveur	Activité sur le client	Description
3	Données Traitements	Traitements Présentation	serveur : servlet avec JDBC client : navigateur (documents avec applet) exemple : network computer
4	Données	Traitements Présentation	serveur : pur serveur de données client : navigateur (applet + JDBC en local) exemple : network computer
5	Données	Données Traitements Présentation	serveur : pur serveur de données client : navigateur (application + JDBC)
6		Données Traitements Présentation	Application ou applet Java sur le client JDBC en local
7	Données Traitements Présentation	Données Traitements Présentation	tout : servlet, applet, applications, JDBC, RMI

Table 2 Typologie de la répartition des activités

Piliers de la programmation distribuée.

Base de données :

droits d›accès, indexations, interrogations, transactions.

Hypertextes (Web) :

interface de présentation et de navigation entre les documents.

Code mobile Java :

comportement associé à des documents.

pour quoi faire? ... deux exemples

Exemple 1 : tribunal fédéral.

contexte :

- 5500 cas traités chaque année,
- stockage dans une BD textuelle BASISplus,
- 30 juges and clercs ont accès à cette base,
- 250 employés (15 informaticiens),
- développement d›applications en Ada83/Motif1.2,
- **futur proche : changer de plate-forme serveur.**

décision :

écrire les applications le + possible indépendemment du matériel.

source : http://java.sun.com/products/jdk/rmi/examples.html

Exemple 1 : tribunal fédéral.

démarrage du projet : début 1996,

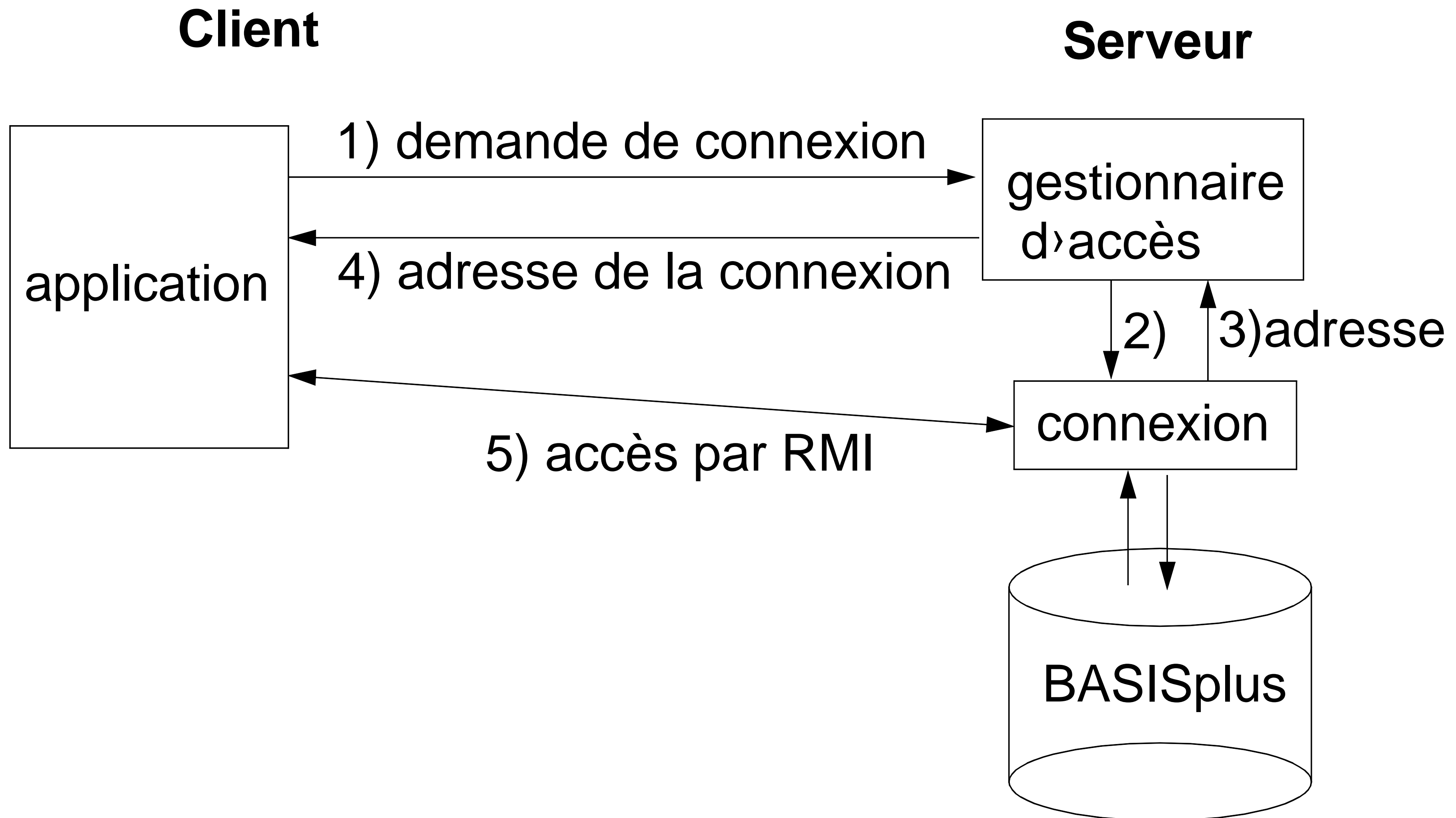
- client WWW pour accéder à la base,
- RMI pas encore là,
- donc CGI/Forms.

début 1997.

- API RMI arrive,
- autres types de clients envisagés,
- RMI : «middeware» entre la BD et les clients,
- Java Native Interface est utilisé pour faire un pilote d›accès à la BD.

source : http://java.sun.com/products/jdk/rmi/examples.html

Exemple 1 : tribunal fédéral.



source : schéma élaboré d›après http://java.sun.com/products/jdk/rmi/examples.html

Exemple 1 : commentaires.

- déploiement par internet des applications,

- facilité de mise en oeuvre (proto en deux jours),

- RMI et JNI sont utilisés,

- distribution du code :
    en partie sur le client,
    en partie sur le serveur (par RMI accéder aux données).

Exemple 2 : gestion de notes de frais.

source : http://java.sun.com/marketing/collateral/javarmi.html

contexte (fictif) :

gestion des notes de frais de l›entreprise en fonction d›une politique qui peut évoluer.

application distribuée:

client :

interface graphique de saisie des notes de frais,

communication avec le serveur via RMI.

serveur :

stocke les notes de frais dans une BD via JDBC.

--> conception classique multi-rangs.

Exemple 2 : gestion de notes de frais.

Comment anticiper des nouvelles politiques de notes de frais au niveau du design de l›application?

Architectures possibles :

a) installer la politique dans le client,

-->problème de mise-à-jour des clients

b) le serveur vérifie chaque entrée ajoutée à une note de frais, et ce relativement à la politique actuelle.

-->problème de charge du serveur et de performance.

c) le serveur vérifie chaque note de frais, au moment où elle est soumise, et ce relativement à la politique actuelle.

-->problème du traitement par lots.

Exemple 2 : gestion de notes de frais.

Comportement mobile:

- Possibilité via RMI, de déplacer des objets d›une application distribuée du client vers le serveur et réciproquement.

Utilisation dans l›exemple:

- La politique (actuelle ou future) de gestion des notes de frais est implantée sur le serveur par un objet.
- Cet objet est envoyé au client lorsqu›il saisit une note de frais.

Exemple 2 : gestion de notes de frais.

Cas d›usage :

1) l›utilisateur est prêt à saisir une note de frais,

2) le client demande un objet qui implante la politique courante,

3) le serveur lui envoie une copie de l›implantation de l›objet.

Commentaires :

- La politique de gestion des notes de frais est toujours dynamique.
- Pour la changer, il suffit de :

1) implanter une nouvelle politique,

2) l›installer sur le serveur,

3) configurer le serveur pour retourner des objets de ce nouveau type.

Exemple 2 : gestion de notes de frais.

Intérêts de cette approche:

- La vérification de la politique s›effectue sur le client. Ils sont avertis au plus tôt de leurs erreurs.

- Il ne faut ni arrêter les clients, ni les mettre à jour. Les objets sont mis-à-jour lorsque le client en a besoin.

- Pas de surcharge du serveur pour des vérifications qui peuvent être locales.

- Facilité de gestion des évolutions
(si elles ont été anticipées)

Exemple 2 : implantation.

Interface de l›objet serveur.

```
import java.rmi.*;
public interface ExpenseServer extends Remote {
      Policy getPolicy() throws RemoteException;
      void submitReport(ExpenseReport report)
      throws RemoteException, InvalidReportException;
}
```

Méthodes que le client peut invoquer sur le serveur.

Exemple 2 : implantation.

```
import java.rmi.*;
import java.rmi.server.*;
class ExpenseServerImpl extends UnicastRemoteObject
                        implements ExpenseServer
{
ExpenseServerImpl() throws RemoteException {
                              // ...set up server state...
}
public Policy getPolicy() { return new TodaysPolicy();}
public void submitReport(ExpenseReport report) {
// ...write the report into the db...
}
} // class
```

Exemple 2 : implantation.

interface de l›objet politique :

```
public interface Policy {
      void checkValid(ExpenseEntry entry)
      throws PolicyViolationException;
}
```

Interface connue du serveur et des clients.

Interface est implantée par un objet sur le serveur.

Exemple 2 : implantation.

objet qui implante la politique.

objet défini sur le serveur.

```
public class TodaysPolicy implements Policy {
public void checkValid(ExpenseEntry entry)
throws PolicyViolationException {
if (entry.dollars() < 20) {return; // no receipt required
} else if (entry.haveReceipt() == false) {
    throw new PolicyViolationException;
} // if
} // method
} // class
```

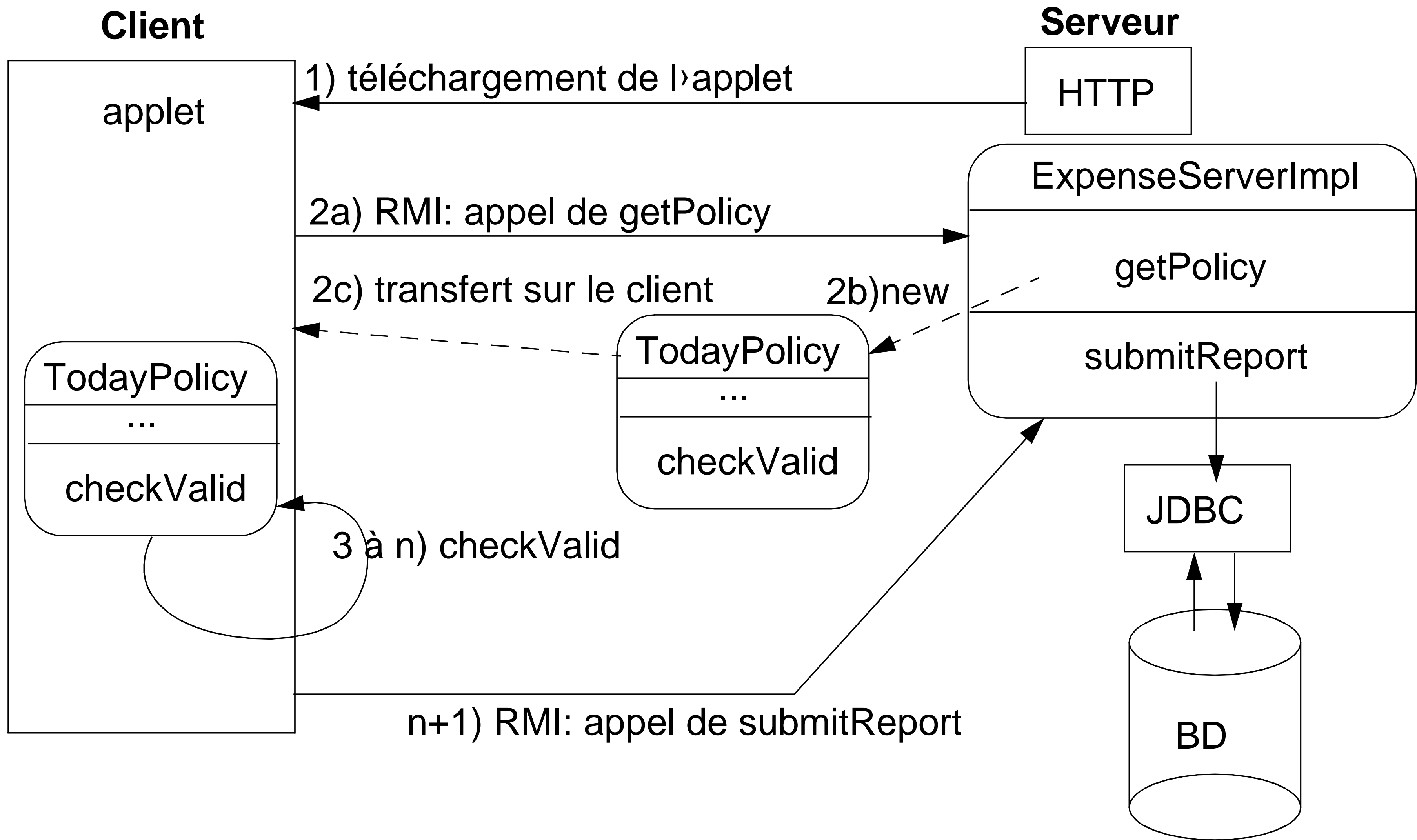
Exemple 2 : implantation.

applet client

```
Policy curPolicy = server.getPolicy();
// start a new expense report
// show the GUI to the user

while (user keeps adding entries) {
try {
     curPolicy.checkValid(entry);
// throws exception if not OK add the entry to the expense report
}
catch (PolicyViolationException e) {
     // show the error to the user
  }
}
server.submitReport(report);
```

Exemple 2: architecture et utilisation.



Exemple 2 : nouvelle politique.

Write this class, install it on the server, and tell the server to start handing out TomorrowsPolicy objects instead of TodaysPolicy objects, and your entire system will start using the new policy.

When the client invokes the server's getPolicy method, RMI on the client checks to see if the returned object is of a known type.

The first time each client encounters a TomorrowsPolicy object, RMI will download the implementation for the policy before getPolicy returns. The client will, without effort, start enforcing the new policy.

source : http://java.sun.com/marketing/collateral/javarmi.html

Exemple 2 : commentaires.

- architecture facile à mettre en oeuvre,

-maintenance de l›application facilitée, si les évolutions sont anticipées.

- voir page 14.

Pour quoi faire des applications distribuées?

Intégration de l›existant.

-nécessité liée à un parc machine et logiciels hétérogènes à faire coopérer :

----> JAVA - RMI - JNI - JDBC

Développement à partir de 0.

- mieux gérer le déploiement et la maintenance des applications,
- mieux tenir compte des performances de certains serveurs (serveur de gros calculs).

Quels sont les critères d›une bonne distribution?