

Java avancé



Tuyêt Trâm DANG NGOC

Laboratoire PRiSM
Université de Versailles-Saint-Quentin

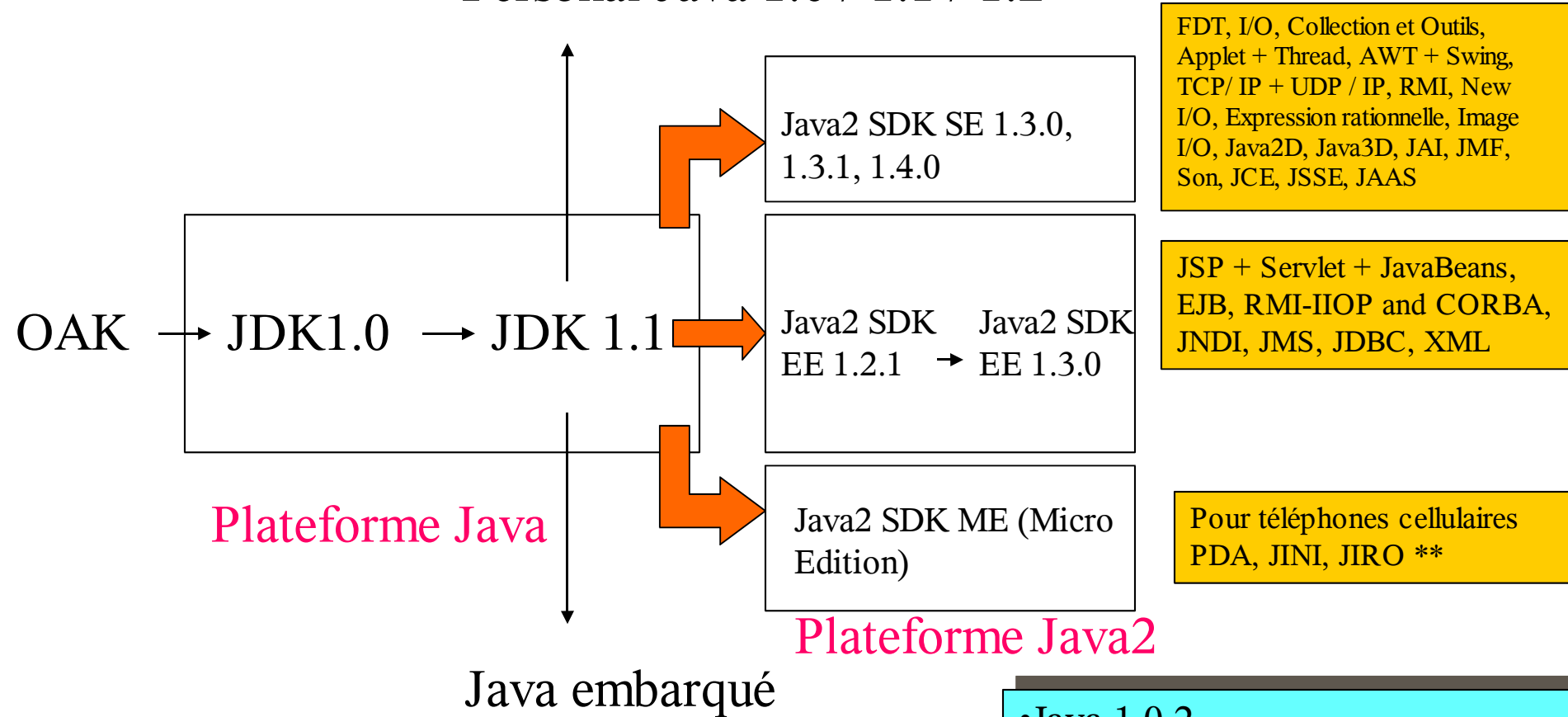
`<dntt@prism.uvsq.fr>`

Cours CNAM, le 26 novembre 2003

Evolution de Java



Personal Java 1.0 / 1.1 / 1.2



- Java 1.0.2
- Java 1.1.1 , 1.1.2, 1.1.3, 1.1.8
- Java 1.2.2

Architecture J2EE

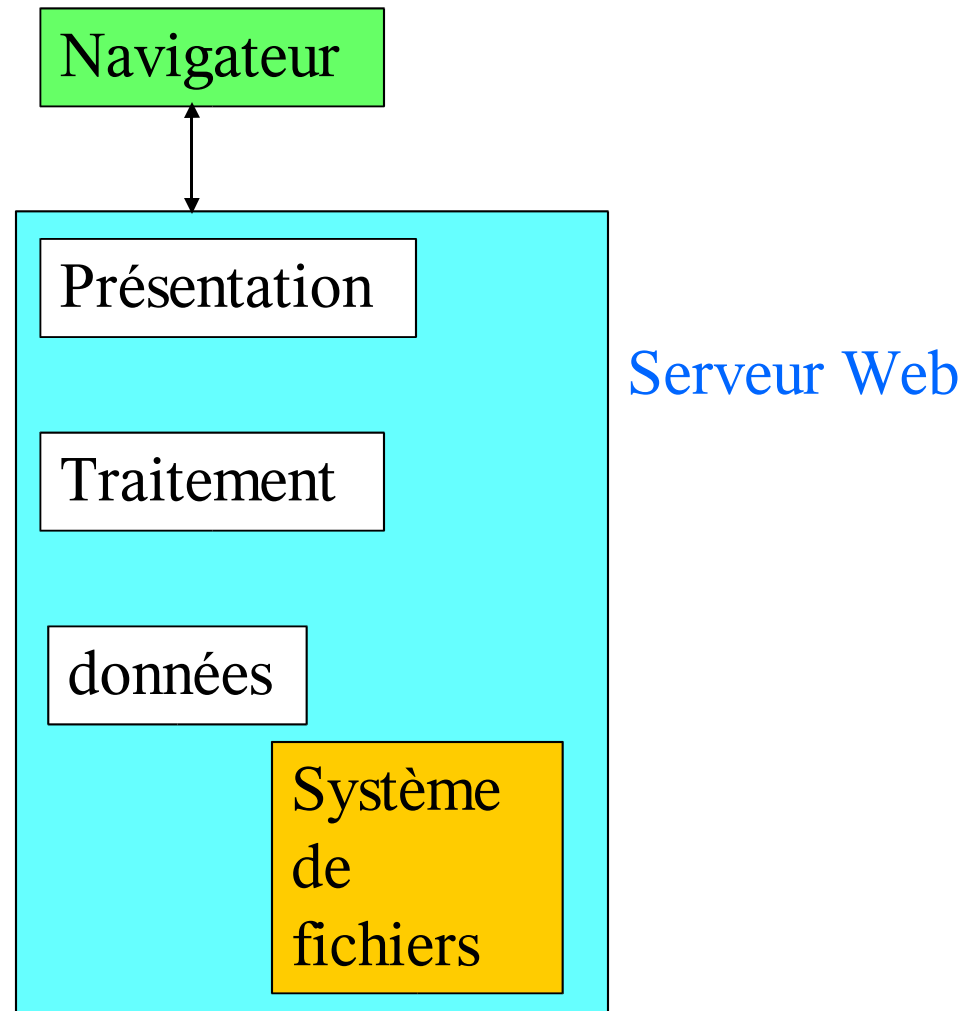


Plan

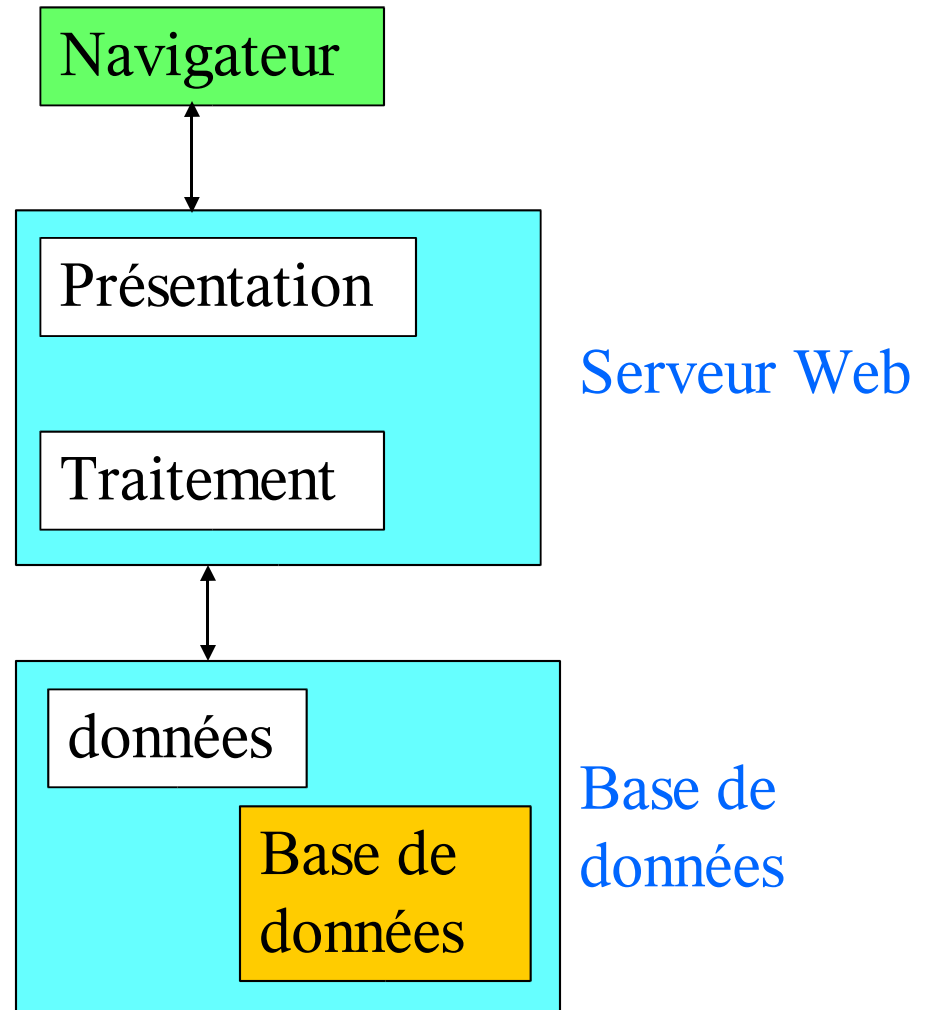


- ■ **Architectures multi-tiers : architecture 1, 2, 3-tiers**
 - **J2EE** : architecture générale
 - **Composants J2EE** : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc.
 - **Outils de développement et d'intégration**
 - **Produits existants**
 - **Conclusion**

Architecture 1-tiers

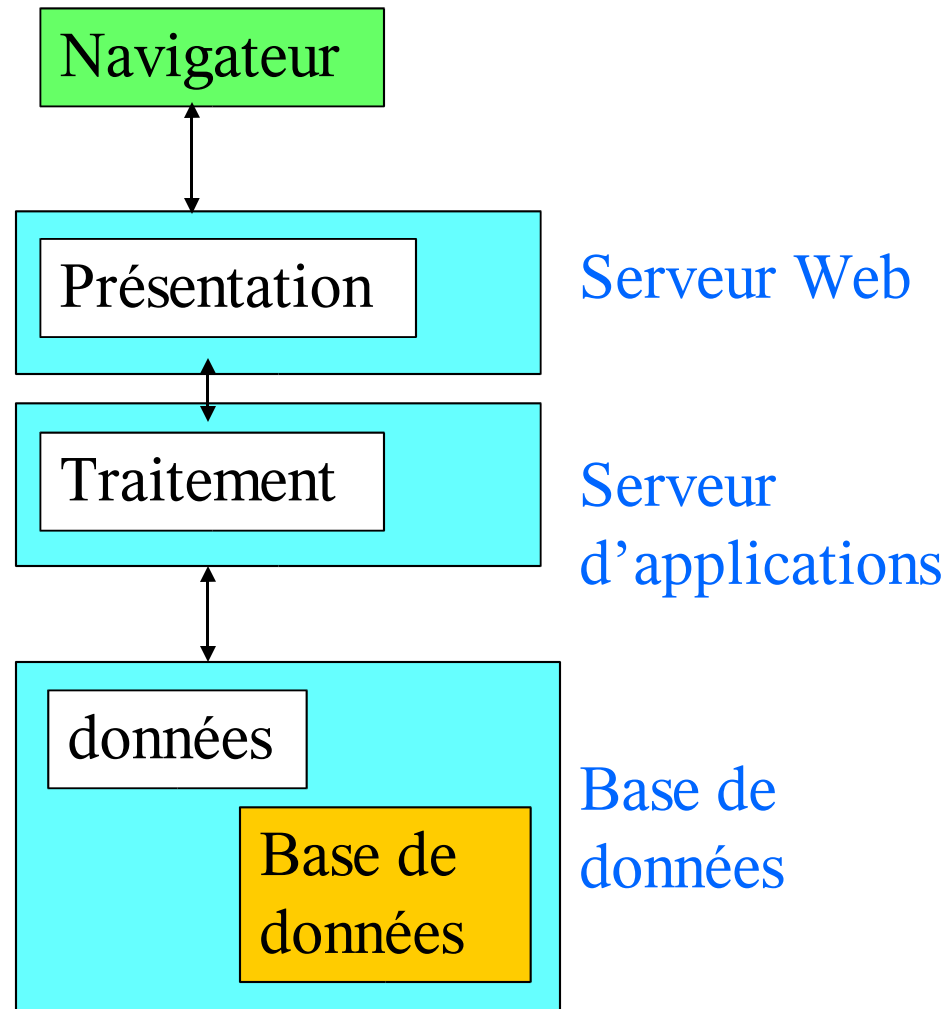


Architecture 2-tiers



Architecture 3-tiers

- + Dimensionnement
- + Maintenance
- + Fiabilité
- + Disponibilité
- + Extensibilité
- + Gestion du développement
- complexité



Plan



- Architectures multi-tiers : architecture 1, 2, 3-tiers
- ■ **J2EE : architecture générale**
- Composants J2EE : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc.
- Outils de développement et d'intégration
- Produits existants
- Conclusion

Qu'est ce que la plateforme J2EE ?



■ Environnement Java

- langage objet
- simple
- portable
- robuste
- indépendant de l'architecture (code virtuel)

■ Pour serveurs d'applications réparties

- ensemble de services
- ensemble de protocoles de communication

Container J2EE



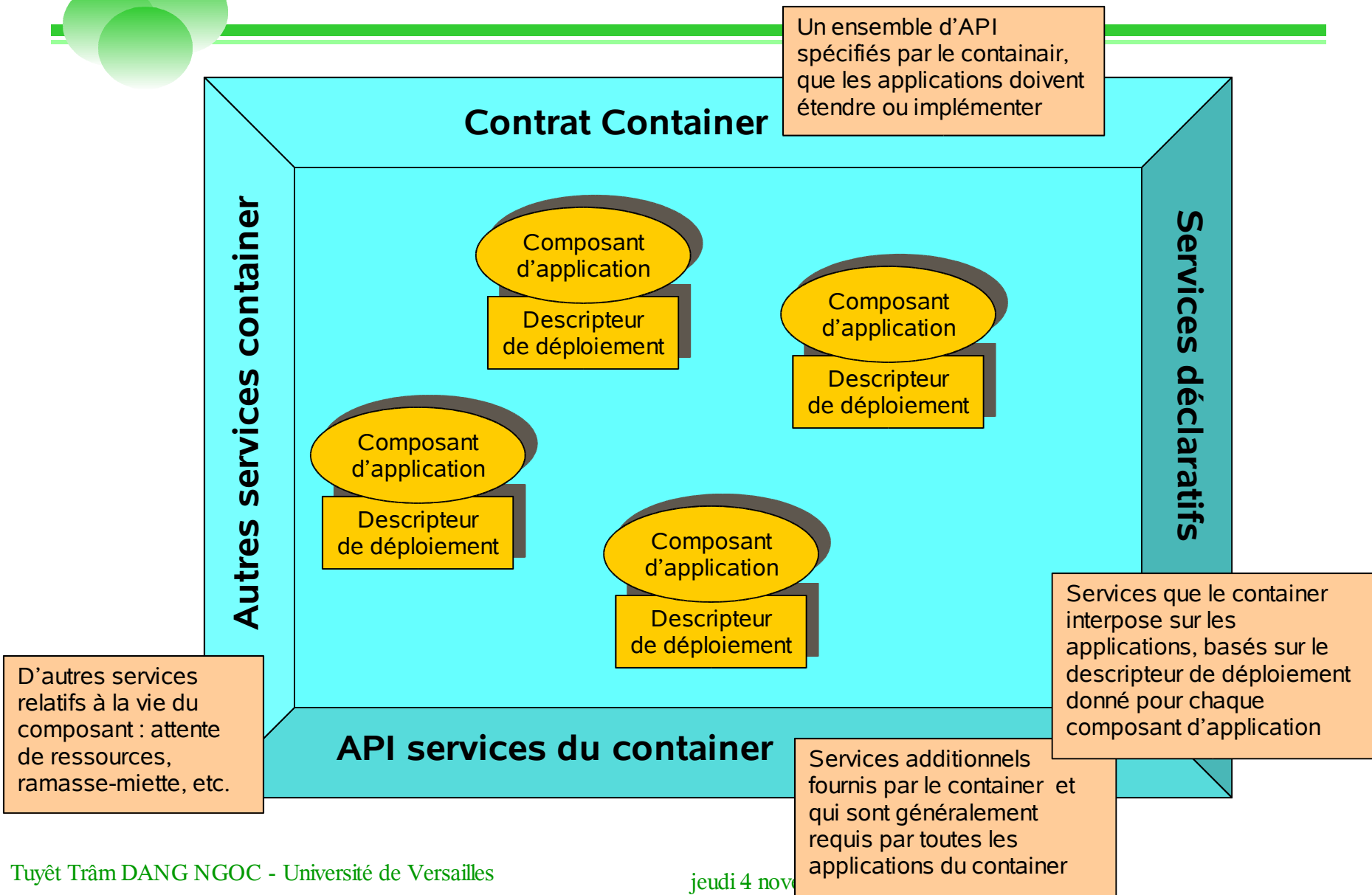
- **Container J2EE** : Environnement d'exécution Java 2 permettant d'héberger des composants applicatifs et de contrôler leur exécution. Il existe deux types de container :
 - **Container J2EE Web** : utilisés pour héberger des servlets ou des pages JSP
 - **Container J2EE EJB** : supportant l'exécution des composants EJB

Interfaces de container

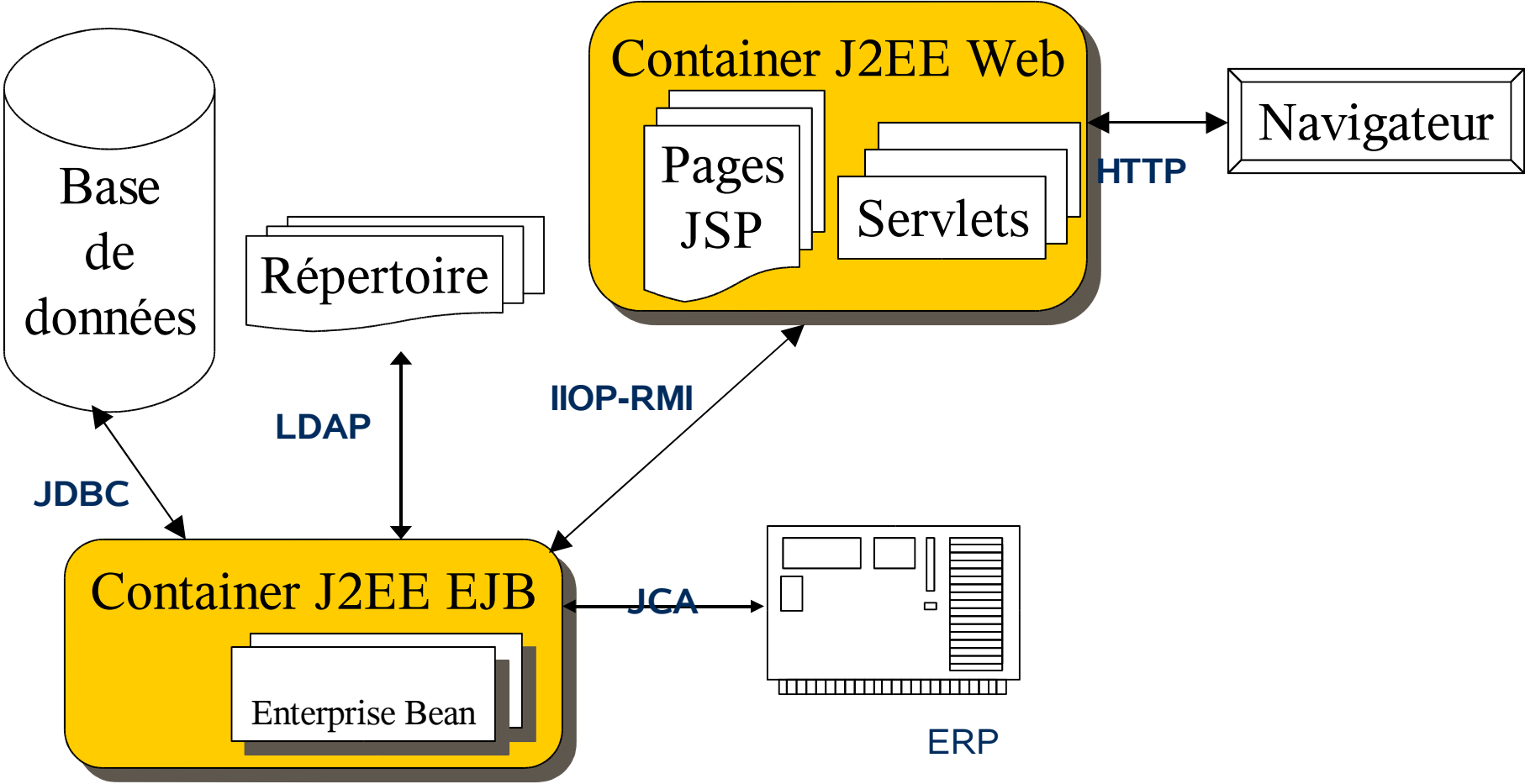


- Un container offre des interfaces constituant le **contrat de composant**. Il gère :
 - **des API de services** : accès SGBD, annuaires, gestionnaire de transactions...
 - **des API de communication** : protocole Internet, envois de messages ou de mail, accès à des objets distants...
- Composants d'application :
 - Servlets, JSP, EJB.
- Descripteurs de déploiement :
 - Fichier XML décrivant le composant d'application
 - Inclut des informations additionnelles requises par le container pour gérer les composants d'application


Container



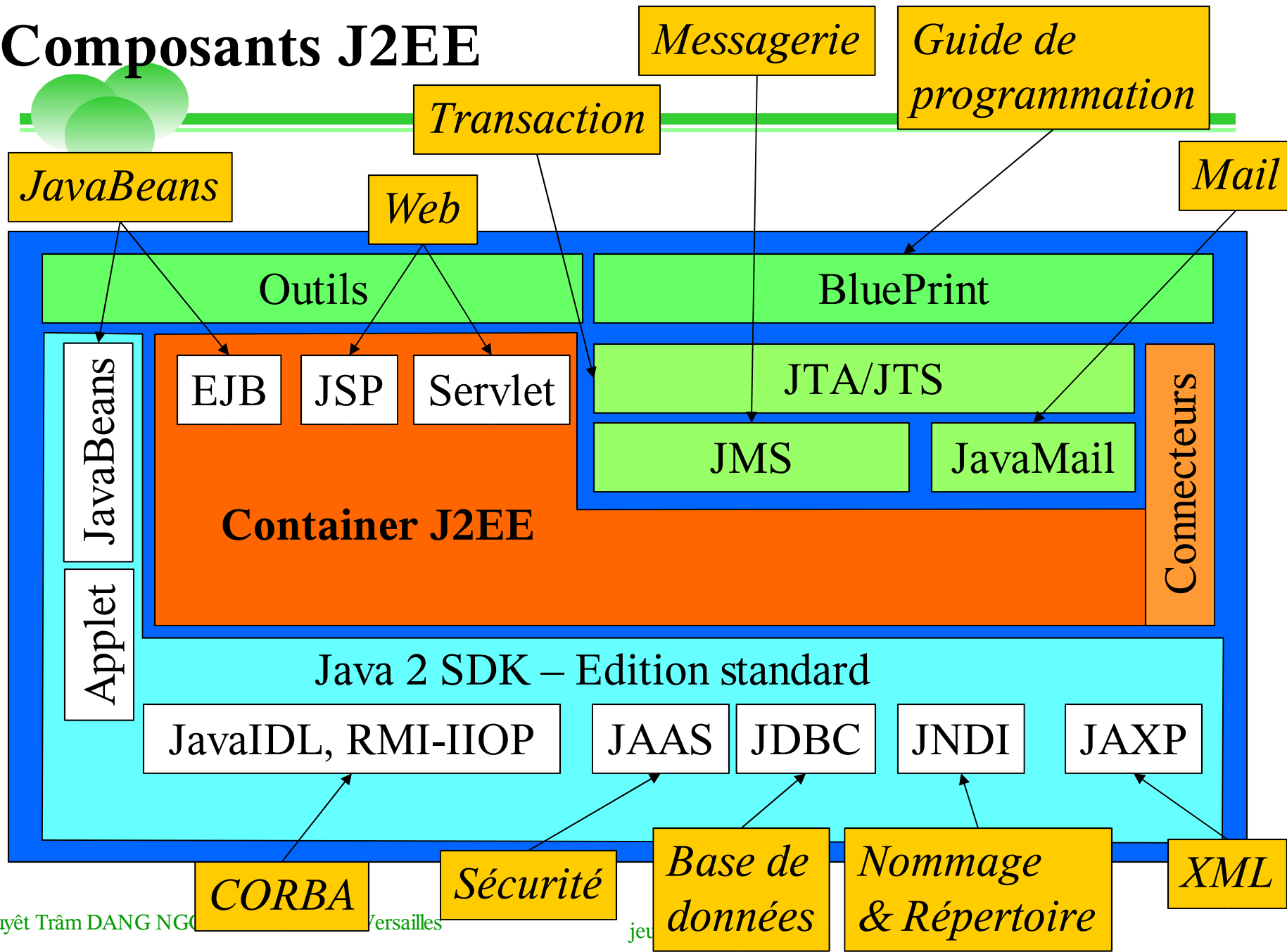
Intégration J2EE



Plan

- 
- Architectures multi-tiers : architecture 1, 2, 3-tiers
 - J2EE : architecture générale
 - ■ Composants J2EE : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc
 - Outils de développement et d'intégration
 - Produits existants
 - Conclusion

Composants J2EE



API pour J2EE



- **Java Servlet 2.3** : services web
- **JSP 1.2** : présentation des pages web
- **EJB 2.0** : les beans
- **JAF 1.0** : intégration des JavaBeans
- **JDBC 2.0** : accès aux bases de données
- **RMI-IIOP, RMI-JRMP, CORBA** : accès et exécution distants
- **JNDI 1.2** : gestion de noms et d'annuaire
- **JMS 1.0** : gestion de messages
- **JTA/JTS 1.0** : gestion de transactions
- **JavaMail 1.2** : gestion du courrier électronique
- **JAAS 1.0** : gestion de la sécurité, authentification et droits d'accès
- **JAXP 1.1** : gestion de documents XML

Java Servlet



- programmation côté serveur
- permet d'étendre les fonctionnalités des serveurs web
- gère le protocole HTTP
- construire des applications Web plus performantes que les CGI
- accède à toutes les API des classes Java
- axé sur la génération du contenu
 - les programmeurs ne se soucient pas de la présentation

Cycle de vie d'une servlet

- **init()** : initialisation de la servlet
chargement du code.

Souvent effectué lors de la première
requête cliente (doGet, doPost)

Allocation d'un pool de threads

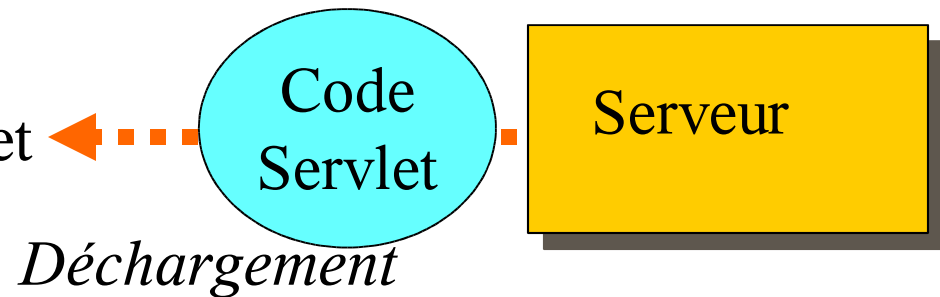
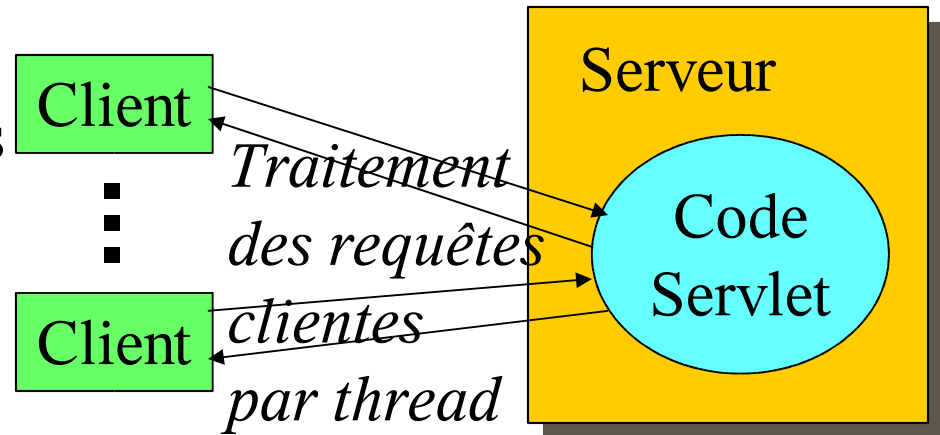
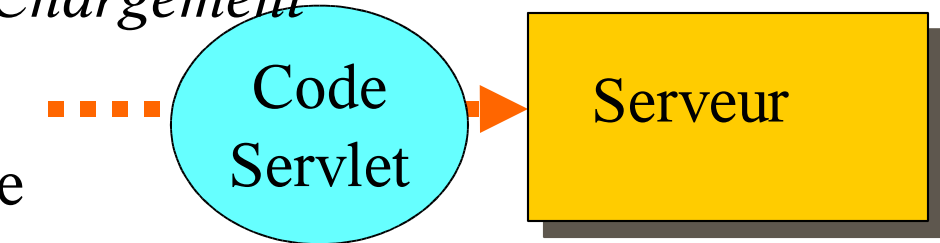
- **doGet ()** : Traitement des requêtes
HTTP GET

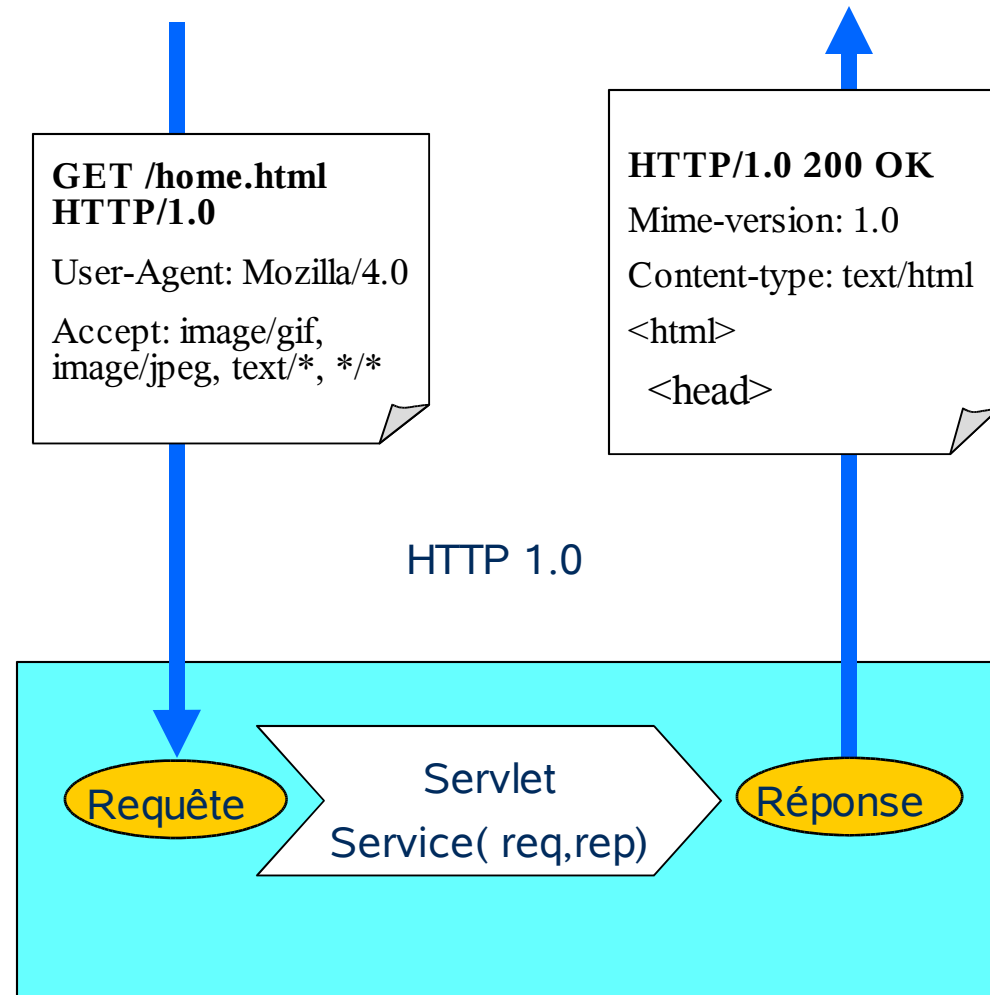
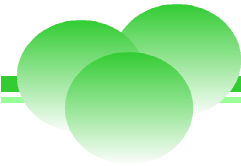
- **doPut ()** : Traitement des requêtes
HTTP PUT

- **doPost ()** : Traitement des requêtes
HTTP POST

- **destroy ()** : destruction de la servlet
par le serveur

Chargement





API Servlet

- `import javax.servlet.*;`
- `import javax.servlet.http.*;`

Programmation des Servlets

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

public class Hello extends HttpServlet
{
    public void doGet (HttpServletRequest requete, HttpServletResponse reponse)
    throws ServletException, IOException
    {
        // Récupère le flux d'écriture pour la réponse
        PrintWriter out = reponse.getWriter () ;

        // Prépare la réponse
        reponse.setContentType ("text/html");

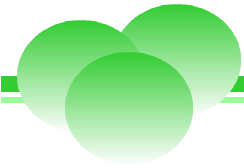
        // Récupère l'adresse IP du client
        String adresseDistante = requete.getRemoteAddr () ;

        // Imprime la page HTML
        out.println("<html>") ;
        out.println("<title>Bienvenue !</title> <body>")
        out.println("<h1>Bonjour !</h1>") ;
        out.println("Votre adresse IP est identifi&eacute; comme :<b>" + adresseDistante + "</b>." ) ;
        out.println("</body></html>") ;
    }
}
```

Java Server Page (JSP)



- Séparation entre la **présentation** et le **contenu** des pages web
- Encapsulé dans des pages HTML
- Création de **contenu dynamique**
- Situé **côté serveur**
- Axé sur la **présentation des données**
 - Les webmasters ne se soucient pas de la programmation
- Semblable à HTML ou XML
 - basé sur des balises



■ Directives

■ Balise de déclaration

<%! ... %>

■ Balise d'affichage

<%= ... %>

■ Balise de scriptlet

<% ... %>

■ Balise de directive

<%@ ... %>

■ Balise d'action

<jsp : ... />

API JSP



- `import javax.servlet.jsp.*;`
- `import javax.servlet.jsp.tagext.*;`

Programmation JSP

```
<html>
<!-- Copyright (c) 1999 The Apache Software Foundation. All rights reserved.-->
<body bgcolor="white">
  <!-- fixer les paramètres de la page →
  <%@ page language = "JAVA" session = "false" %>
  <!-- déclarer une variable caractère →
  <% char c = 0 ; %>
  <!-- scriptlet code Java →
  <%
    for (int i =0 ; i < 26 ; i++) {
      c = (char) ('a' + i ;
    }
  %>
  <!-- afficher c →
  <%= c %>
  <% } %>
  <jsp:useBean id='clock' scope='page' class='dates.JspCalendar' type='dates.JspCalendar' />
  <font size=4>
    <ul>
      <li> Jour du mois <jsp:getProperty name="clock" property="dayOfMonth"/> </li>
      <li> année <jsp:getProperty name="clock" property="year"/> </li>
      <li> mois <jsp:getProperty name="clock" property="month"/> </li>
      <li> temps <jsp:getProperty name="clock" property="time"/> </li>
    </ul>
  </font>
</body>
</html>
```

Java Beans Coding Convention (1/2)



- Convention pour les commentaires
- Convention de nommage
 - class et interface (*pas d'underscore, majuscule au début de chaque mot*)
 - méthode (*minuscule au début, majuscule au début de chaque mot*)
 - variable (*d'instance préfixé par 'i', de classe par 'c', de paramètre par 'a', temporaire sans, constante tout en capitale*)
 - paquetage (*tout en minuscule, dans le répertoire associé*)
- Convention d'accès aux variables
 - récupérer la valeur d'une variable d'instance (*get suivi du nom de la variable*)
 - positionner la valeur d'une variable d'instance (*set suivi du nom de la variable*)

Java Beans Coding Convention (2/2)

- Convention de constructeurs (*avoir un constructeur par défaut*)
- Convention d'initialisation (*utiliser les conventions de variables et les set*)
- Convention d'importation (*ne pas utiliser '*'*)
- Sériàlisation (*doit implémenter l'interface Serializable*)
- Convention d'évènement
 - Event handling methods:
void <eventOcurranceMethodName>(<EventStateObjectType> evt);
 - Event handling methods with arbitrary argument list:
void <eventOcurranceMethodName>(<ArbitraryParameterList>);
 - Multicast event delivery:
public void add<ListenerType>(<ListenerType> listener);
public void remove<ListenerType>(<ListenerType> listener);
 - Unicast event delivery:
public void add<ListenerType>(<ListenerType> listener) throws
java.util.TooManyListenersException;
public void remove<ListenerType>(<ListenerType> listener);

Enterprise Java Beans (EJB)



- **Modèle client/serveur distribué**
- **Code exécuté sur le serveur**
 - proche des données
 - serveur souvent plus puissant que le client
- **Code localisé sur le serveur**
 - changer le code du serveur ne change pas le code du client
- Un **EJB** est juste une collection de classes Java et d'un fichier XML. Les classes Java suivent un certains nombre de règles et fournissent des callbacks comme définis dans l'environnement J2EE et les spécifications EJB.

Container EJB



- Un container EJB est un **environnement d'exécution** pour un **composant EJB**.
- Un EJB s'exécute sur un container EJB.
- Un container EJB s'exécute par un serveur d'applications et prend la responsabilité des problèmes au niveau système
- Le container EJB gère le **cycle de vie** de l'EJB.
- Le container EJB fournit aussi un nombre de **services additionnels**
 - Gestion de la persistance
 - Transactions
 - Sécurité
 - Cadre de travail pour Business Logic
 - Dimensionnement
 - Portabilité
 - Gestion des erreurs

Beans entité



- Un **bean entité** est un objet persistant
- Un **bean entité** peut avoir plusieurs clients
 - **Bean Managed Persistence (BMP)** : le container EJB est responsable de la persistance du bean.
 - **Container Manager Persistence (CMP)** : le bean est responsable de sa propre persistance.

Beans session



- Un **bean session** pour chaque client
- Il y a deux types de **beans session**
 - **sans état**
 - pas d'état entre les invocations
 - rapide et efficace
 - **avec état**
 - maintient les états entre les invocations
 - persistance limitée

Structure EJB



■ Interface home

- fournit un moyen pour le client EJB de récupérer une instance d'un EJB

■ Interface distante

- expose les méthodes que le client EJB peut utiliser

■ Code EJB

- implémente l'interface distante et l'interface EJB approprié (SessionBean ou EntityBean par exemple)

API EJB



- `import javax.ejb.* ;`
- `import javax.ejb.spi.* ;`

Programmation EJB

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface HomeDe extends EJBHome
{
    De create() throws RemoteException, CreateException;
}
```

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface De extends EJBObject
{
    public int lancer() throws RemoteException;
}
```



```
import java.rmi.RemoteException ;
import javax.ejb.SessionBean ;
import javax.ejb.SessionContext ;

public class DeEJB implements SessionBean
{
    public DeEJB() {...}
    public void ejbCreate() {...}
    public void ejbRemove() {...}
    public void ejbActivate(){...}
    public void ejbPassivate() {...}
    public void setSessionContext (SessionContext sc) {...}
    public int lancer(){...}
}
```

JavaBeans Activation Framework (JAF)



- Les services d'activation sont des composants fondamentaux utilisés pour **activer automatiquement** des objets en mémoire à l'état dormant suivant une requête cliente
- JAF est un **cadre de travail** pour construire des applications qui peuvent activer automatiquement des objets pouvant manipuler des données reçues dans un flux d'entrée

API JAF



```
■ import javax.activation.* ;
```

Java Data Base Connectivity (JDBC)



- **JDBC** permet aux applications d'accéder à des SGBD relationnels et de manipuler des données provenant de ces bases
- API **Java**
- Calqué sur **ODBC** de Microsoft
- Implémente le standard **CLI de l'X/OPEN**

Pilotes JDBC

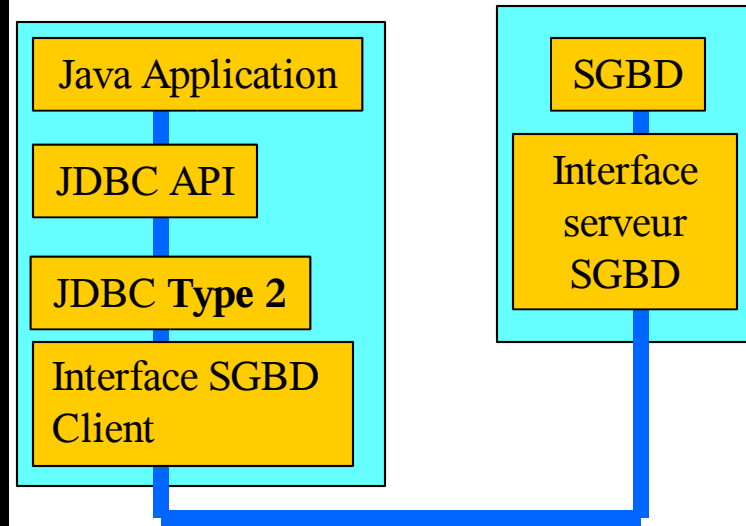
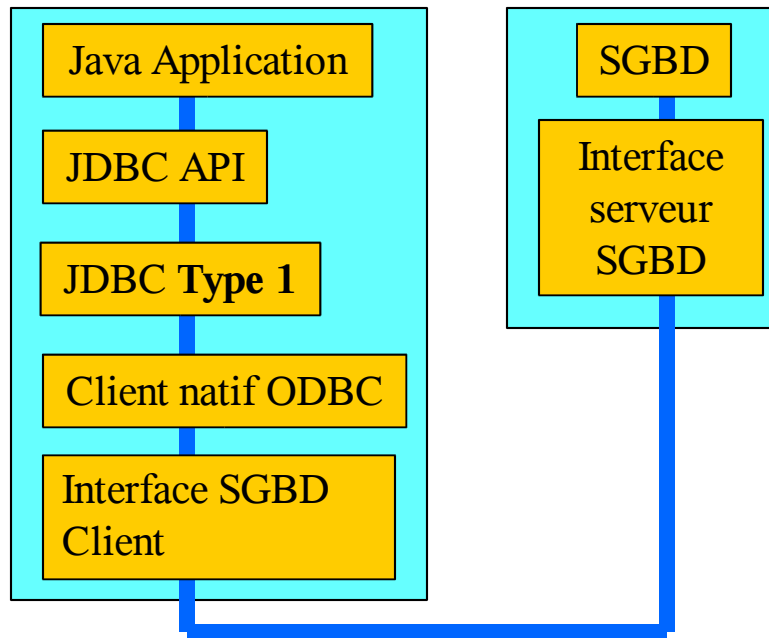


- **Type 1** : Pont *JDBC-ODBC* fournit un accès JDBC API par l'intermédiaire d'un ou plusieurs pilotes ODBC.
- **Type 2** : A *native-API partly Java technology-enabled driver* convertit les appels JDBC en un applet sur l'API client (Oracle, Sybase, Informix, DB2, et autres SGBD)
- **Type 3** : A *net-protocol fully Java technology-enabled driver* convertit les appels JDBC en un protocole indépendant traduit ensuite en protocole du SGBD par un serveur
- **Type 4** : A *native-protocol fully Java technology-enabled driver* convertit les appels JDBC directement dans le protocoles de communication utilisé par le SGBD

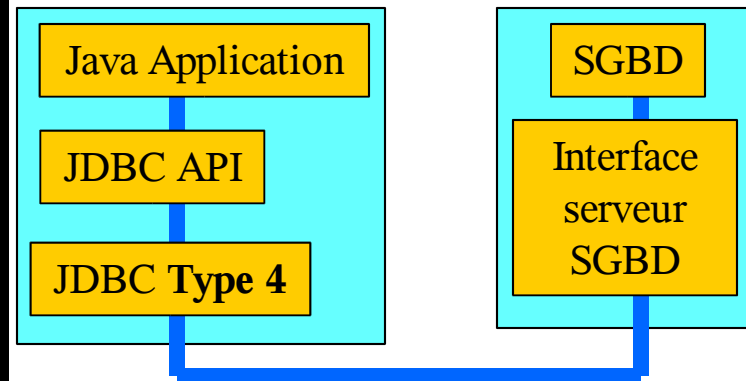
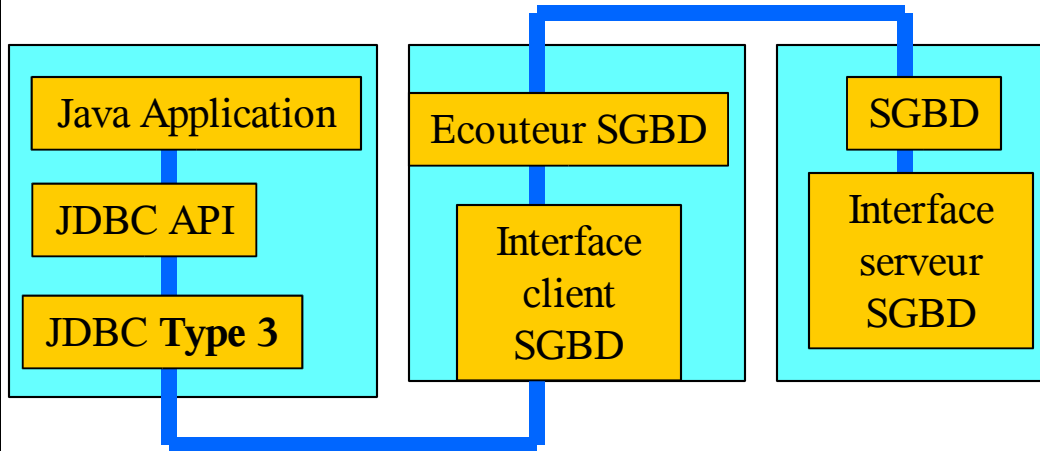
indépendant du constructeur

dépendant du constructeur


natif côté client



Ecoute distante



API JDBC



```
import java.sql.*;
```

Programmation JDBC

```
private void executer (String driver, String chaine_connexion, String login, String password)
{
    // On charge le driver
    Class.forName (driver) ;

    // on se connecte
    conn = DriverManager.getConnection (chaine_connexion, login, password) ;

    // on initialise le curseur
    stmt = conn.createStatement () ;

    // execution d'une chaine SQL sur le SGBD
    String SQL = "SELECT * from essai" ;
    ResultSet rs = stmt.executeQuery (SQL) ;
    ResultSetMetaData rsetdata = rs.getMetaData() ;
    System.out.println (rsetdata.getColumnNames(1) + "\t" + rsetdata.getColumnNames(2)) ;
    while (rs.next ())
    {
        // On recupere le premier champ qui est une chaine puis le deuxieme
        System.out.println (rs.getString (1) + "\t" + rs.getInt (2)) ;
    }
    stmt.close () ;
    conn.close () ;
}
```

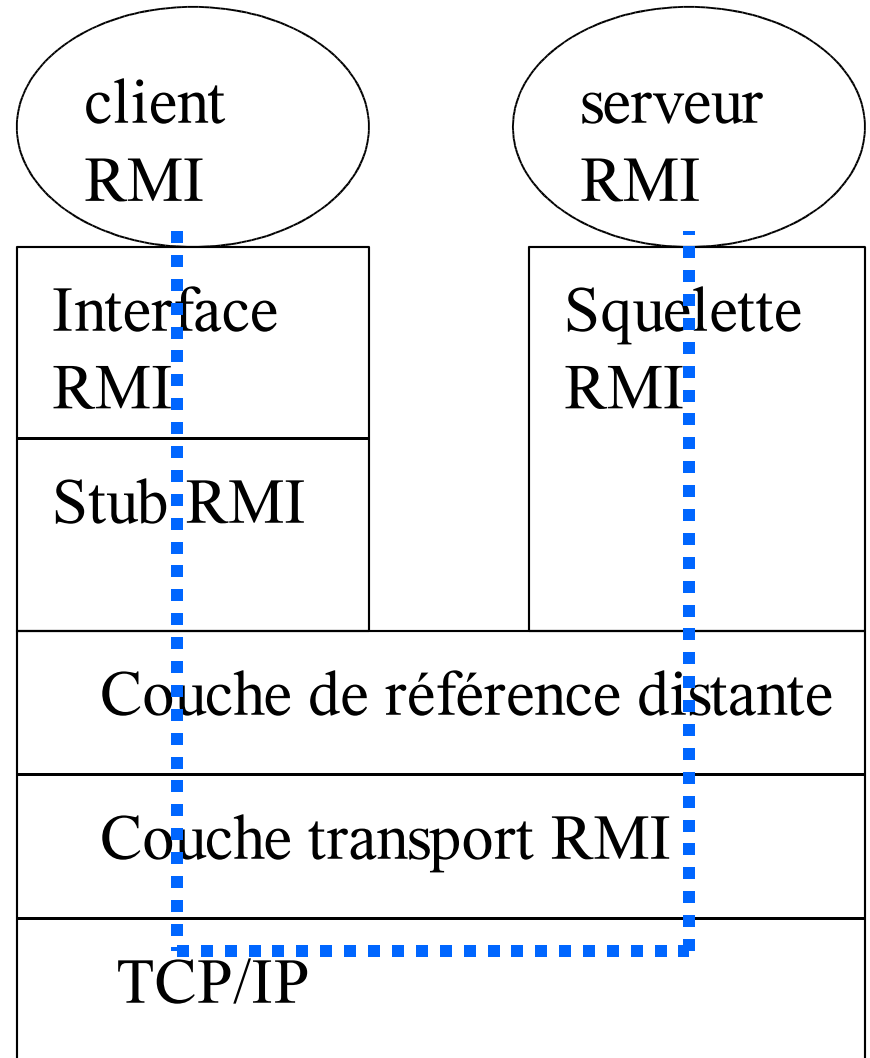
RMI-IIOP, RMI-JRMP, CORBA

- RMI et CORBA sont des architectures complexes permettant d'invoquer un objet distant géré par une autre machine virtuelle comme s'il était local.
 - CORBA : générique langage objets
 - RMI : spécifique à Java

	GIOP
RMI	IIOP
	TCP/IP

Remote method Invocation (RMI)

- **Objet distant** localisé par l'API JNDI
- Registre associe au nom objet une référence
- **stub** représente l'interface sur le client. Agit comme proxy de l'objet distant
- **squelette (skeleton)** sur le serveur pour recevoir les messages et invoquer l'objet serveur
- Génération de stub et skeleton à l'aide de rmic



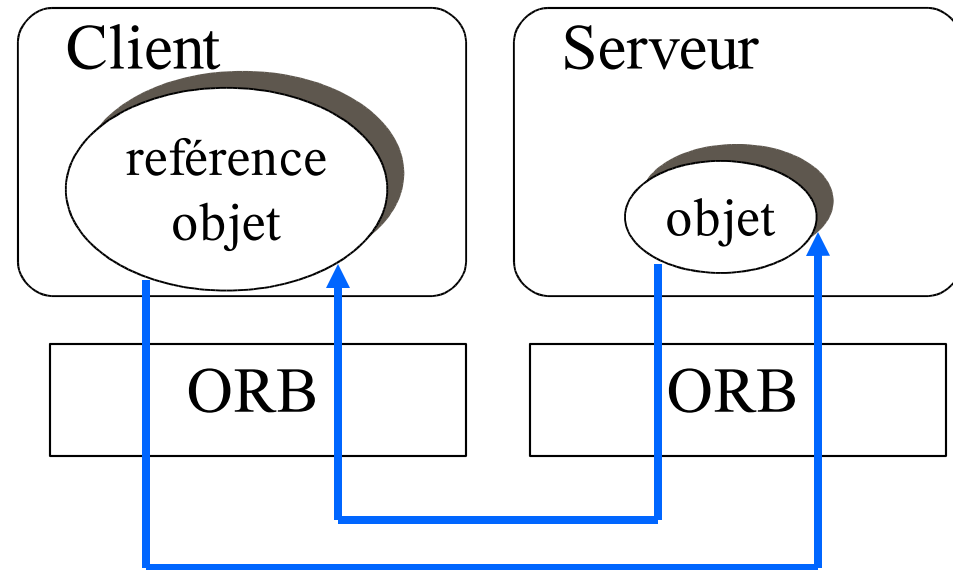
Common Object Request Broker Architecture (CORBA)



- **GIOP (General Inter-ORB Protocol)** : Le protocole de plus haut-niveau de communication CORBA qui convertit des messages créés par les stubs en format transport-message capable de communiquer entre les implémentations ORB (Object Request Broker).
- **IIOP (Internet Inter-ORB Protocol)** : Le protocole de plus bas niveau utilisé par CORBA ;c'est une couche transport décrivant comment les messages GIOP sont transmis au dessus de TCP/IP.

Architecture CORBA

- Portable Object Adapter (POA)
- Object Request Broker (ORB)



API CORBA, RMI



- `import javax.rmi.* ;`
- `import javax.rmi.CORBA ;`
- `import org.omg.CORBA.* ; // RMI-IIOP, IDL`
- `import org.omg.CosNaming.* ; // RMI-IIOP, IDL`

Programmation RMI

// Fichier Bonjour.java : Interface du service 'Bonjour'

```
import java.rmi.Remote ;
import java.rmi.RemoteException ;

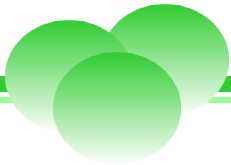
public interface Bonjour extends Remote
{
    String disBonjour (String nom) throws RemoteException ;
}
```

// Fichier 'BonjourImpl.java': implementation de l'interface 'Bonjour' definie dans 'Bonjour.java'.

```
import java.rmi.RemoteException ;
import java.rmi.server.UnicastRemoteObject ;

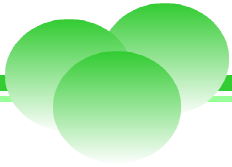
public class BonjourImpl extends UnicastRemoteObject implements Bonjour
{
    public BonjourImpl () throws RemoteException
    { super () ;}

    public String disBonjour (String nom)
    { return "Bonjour " + nom + " !" ;}
}
```

```
// Enregistre l'objet Bonjour dans le registry du serveur local
import java.rmi.Naming ;
import java.rmi.RMISecurityManager ;

public class ServiceBonjour
{
    public static void main (String args[])
        throws Exception
    {
        BonjourImpl obj = new BonjourImpl () ;
        Naming.rebind ("rmi://localhost/ObjetBonjour", obj) ;
        System.out.println ("Enregistre sous le nom 'ObjetBonjour'") ;
    }
}
```



// Fichier 'BonjourClient.java' : contacte le registry a l'URL donne et recupere la reference de l'objet.

```
import java.rmi.Naming ;
```

```
import java.rmi.RemoteException ;
```

```
public class BonjourClient
```

```
{  
    public static void main (String argv [])  
        throws Exception  
    {  
        String message = null ;  
        Bonjour obj = null ;  
  
        obj = (Bonjour) Naming.lookup ("rmi://gibet.prism.uvsq.fr/ObjetBonjour") ;  
        message = obj.disBonjour (argv [0]) ;  
        System.out.println ("Le serveur a dit :" + message + ".") ;  
    }  
}
```

Java Naming and Directory Interface (JNDI)



- JNDI permet d'accéder à de nombreux **services de noms et d'annuaire**.
- En utilisant JNDI, un objet Java peut stocker et récupérer des **objets de n'importe quel type**
- JNDI fournit des méthodes pour effectuer n'importe quelle opération standard sur un annuaire : **associer des attributs à un objet ou chercher un objet par ses attributs**.
- JNDI permet aux applications Java de bénéficier de toutes les informations d'annuaires comme LDAP, NDS, DNS et NIS (YP) et permet à Java de **cohabiter avec des applications existantes**.
- **Enregistrement dynamique** de services et de clients
- Traitement **point à point**

Architecture JNDI

Application Java

API JNDI

Gestionnaire de nommages et d'annuaire

JNDI SPI

SPI Système
de fichiers

Système de
fichiers

SPI nommage
CORBA

CosNaming
CORBA

SPI LDAP

Services
LDAP

SPI RMI

Services
RMI

Autres SPI

Autres services
(DNS, NDS,
NIS...)

API JNDI



- `import javax.naming.* ;`
- `import javax.naming.directory.* ;`
- `import javax.naming.event.* ;`
- `import javax.naming.ldap.* ;`
- `import javax.naming.spi.* ;`

Programmation

```
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

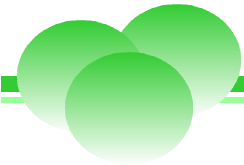
class ClientDe
{
    // Crée une instance EJB "dé" sur le serveur EJB et appelle ensuite la méthode lancer ()
    // et enfin affiche le résultat du lancé de dé
    public static void main (String[] args)
    {
        try
        {
            InitialContext jndiContext = new InitialContext ();
            ref = jndiContext.lookup ("jeux/De");
            DeHome home= (EnsDe) PortableRemoteObject.narrow (ref, DeHome.class);

            De de = home.create () ; // Crée un objet De depuis l'interface EnsCalculateur
            System.out.println (de.lancer (0.2, 1000, "vert"));
        }
        catch (Exception e)
        {
            System.out.println(e.toString());
        }
    }
}
```

Java Message Service (JMS)



- **JMS** définit un mécanisme standard permettant aux composants d'envoyer et de recevoir des messages de façon asynchrone
- Fournit un service **flexible** et **sûr** pour l'échange asynchrone de données et d'évènements commerciaux critiques à travers une entreprise
- L'API JMS y ajoute une API commune et un cadre de travail permettant le développement de messages portables basés en Java

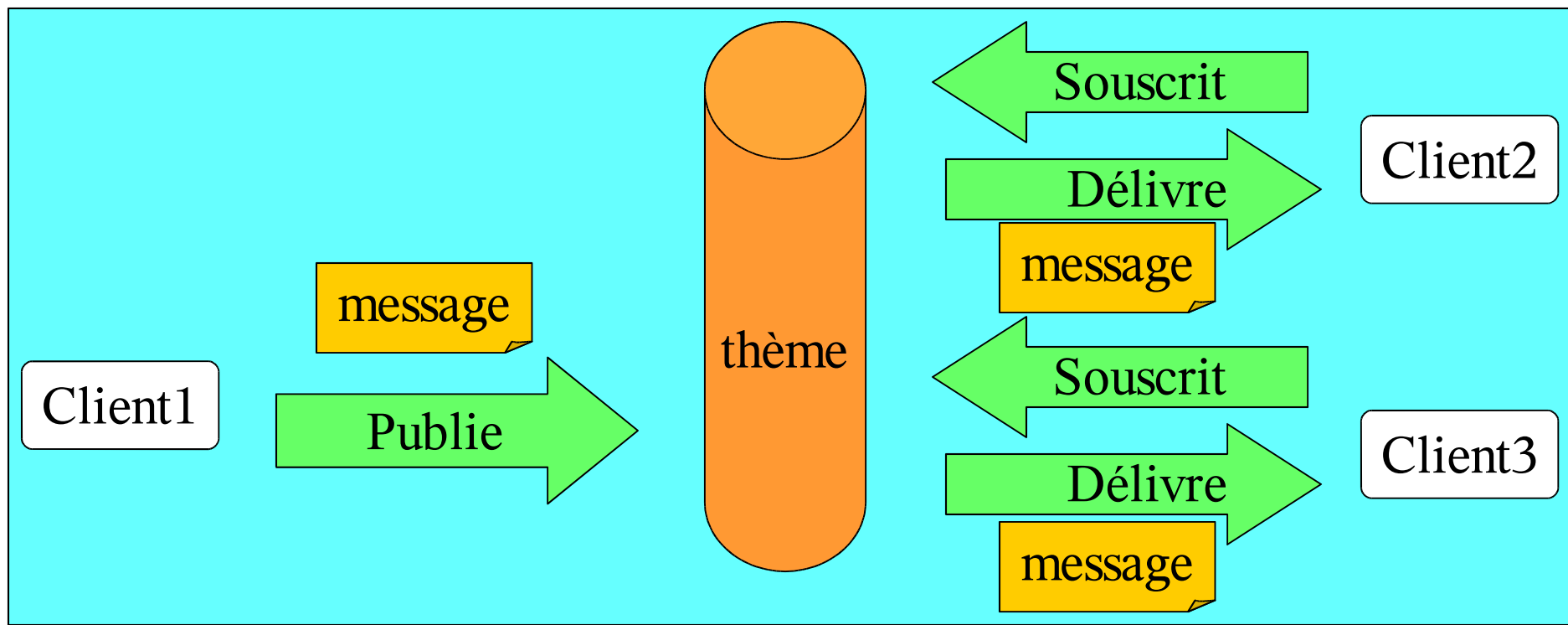
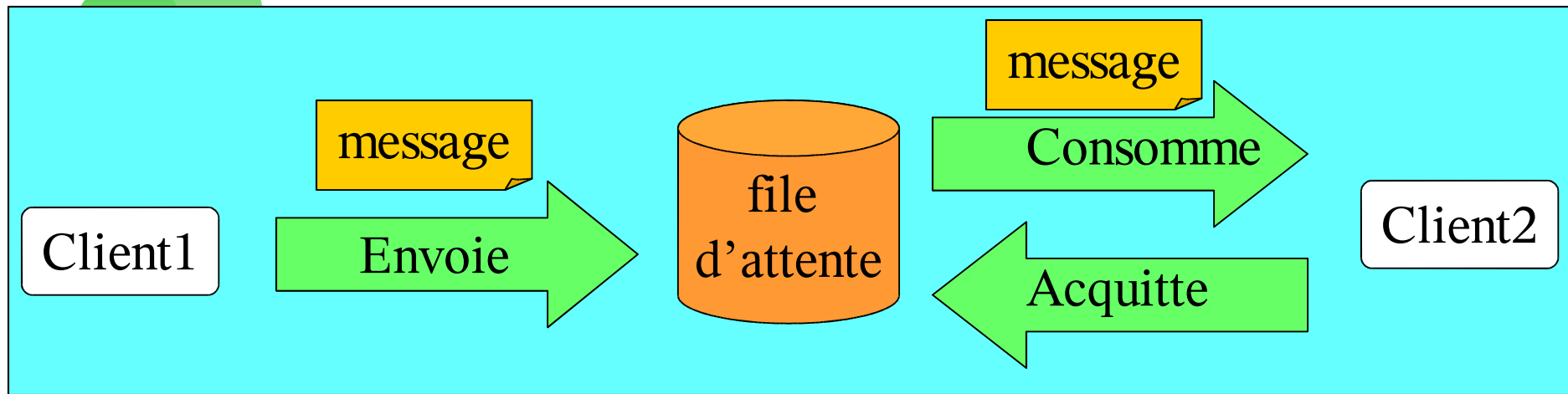


■ Synchrones

- envoyé une fois, non garanti d'atteindre le serveur
- perdu si non reçu

■ Asynchrones

- Envoyé jusqu'à acquittement
- Mise en file d'attente jusqu'à ce que le message soit reçu



Cas d'utilisation de JMS



■ Système faiblement couplé

- Sans connexion
- Supprime les dépendances entre le client et la plateforme serveur (langage de programmation, version)

■ Publication souscription

- Envoie, reçoit des information avec un ou plusieurs clients non identifiés

■ Intégration avec d'autres systèmes de messageries

- IBM MQ-Series
- Microsoft Message Queue

API JMS



```
■ import javax.jms.* ;
```

Java Transaction Support/API (JTS/JTA)



- JTA et JTS permettent aux applications J2EE d'affranchir le développeur de composant de la gestion des transactions
- Les développeurs peuvent définir les propriétés transactionnelles des composants JavaBeans pendant la conception et le déploiement à l'aide d'états déclaratifs dans le descripteur de déploiement.
- Le serveur d'applications prend la responsabilité de la gestion des transactions

Exemple de déroulement de transaction



```
begin transaction
...
update table-a
...
if (condition-x)
    commit
    transaction
else if (condition-y)
    update table-b
    commit
    transaction
else
    rollback transaction
begin transaction
update table-c
commit transaction
```

API JTA

- import `javax.transaction` ;
- import `javax.transaction.xa` ;

Programmation JTA

```
import javax.transaction.UserTransaction ;
[...]  
public void withdrawCash(double amount) {  
    UserTransaction ut = context.getUserTransaction();  
    try  
    {  
        ut.begin(); updateChecking(amount);  
        machineBalance -= amount; insertMachine(machineBalance);  
        ut.commit();  
    }  
    catch (Exception ex)  
    {  
        try  
        {  
            ut.rollback();  
        }  
        catch (SystemException syex)  
        {  
            throw new EJBException ("Rollback failed: " + syex.getMessage());  
        }  
        throw new EJBException ("Transaction failed: " + ex.getMessage());  
    }  
}  
[...]
```

JavaMail



- **JavaMail** 1.2 définit un ensemble d'interface de classe permettant de construire des applications sur les technologies de courrier électronique
- Gestion des standards de courrier
 - **SMTP** : protocole de transport de courrier électronique
 - **POP** : récupération du courrier sur serveur distant
 - **IMAP** : gestion de boîtes aux lettres distantes
 - **MIME** : standard de codage du contenu de courrier permettant notamment l'attachement de documents de tout format et l'utilisation de langues différentes

API JavaMail



- `import javax.mail.* ;`
- `import javax.mail.event.* ;`
- `import javax.mail.internet.* ;`
- `import javax.mail.search.* ;`

Programmation JavaMail

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

public class SendEmail
{
    public String SendMessage(String emailto, String emailfrom,
        String smtp host, String emailmultipart, String msgSubject,
        String msgText)
    {
        String msgText2 = "multipart message";
        boolean sendmultipart = Boolean.valueOf(emailmultipart).booleanValue();
        // set the host
        Properties props = new Properties();
        props.put("mail.smtp.host", smtp host);
        // create some properties and get the default Session
        Session session = Session.getDefaultInstance(props, null);
        try
        {
            // create a message
            Message msg = new MimeMessage(session);
            // set the from
            InternetAddress from = new InternetAddress(emailfrom);
            msg.setFrom(from);
            InternetAddress[] address =
            {
                new InternetAddress(emailto)
            };
            msg.setRecipients(Message.RecipientType.TO, address);
            msg.setSubject(msgSubject);
```

```
if(!sendmultipart)
{
    // send a plain text message
    msg.setContent(msgText, "text/plain");
}
else
{
    // send a multipart message// create and fill the first message
    part
    MimeBodyPart mbp1 = new MimeBodyPart();
    mbp1.setContent(msgText, "text/plain");
    // create and fill the second message part
    MimeBodyPart mbp2 = new MimeBodyPart();
    mbp2.setContent(msgText2, "text/plain");
    // create the Multipart and its parts to it
    Multipart mp = new MimeMultipart();
    mp.addBodyPart(mbp1);
    mp.addBodyPart(mbp2);
    // add the Multipart to the message
    msg.setContent(mp);
}
Transport.send(msg);
}
catch(MessagingException mex)
{
    mex.printStackTrace();
}
return "Email envoyé à " + emailto;
}
}
```

Java Authentication and Authorization Service (JAAS)



- **Authentication** : déterminer qui exécute le code Java (application, applet, bean, servlet)
- **Autorisation** : vérifier si celui qui exécute le programme a les permissions nécessaires pour le faire
- Modules d'authentification disponibles
 - JNDI
 - UNIX
 - Windows NT
 - Kerberos
 - KeyStore

API JAAS

- import `javax.security.auth` ;
- import `javax.security.auth.callback` ;
- import `javax.security.auth.login` ;
- import `javax.security.auth.spi` ;

```
grant <signer(s) field>, <codeBase URL> <Principal field(s)>  
> {  
  permission perm_class_name "target_name", "action";  
  ...  
  permission perm_class_name "target_name", "action";  
};
```

Java.policy

```
<Principal field> := Principal Principal_class "principal_name"
```

- import `javax.security.*` ;

Programmation JAAS (authentication)

```
import java.io.IOException;
import javax.security.auth.*;
import javax.security.auth.callback.*;
import javax.security.auth.login.*;
import javax.security.auth.spi.*;

public class SampleLoginModule implements LoginModule {
    public boolean abort () {
        // Method to abort the authentication process (phase 2).
    }
    public boolean commit() {
        // Method to commit the authentication process (phase 2).
    }
    public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options) {
        // Initialize this LoginModule.
    }
    public boolean login() {
        // Method to authenticate a Subject (phase 1).
    }
    public boolean logout() {
        // Method which logs out a Subject.
    }
}
```

```
grant codebase "file:SampleLoginModule.jar" {
    permission javax.security.auth.AuthPermission "modifyPrincipals";
};
```

Programmation JAAS (autorisation)

```
import java.io.File;
import java.security.PrivilegedAction;
public class SampleAction implements PrivilegedAction {
    public Object run() {
        System.out.println("\nYour java.home property value is: " + System.getProperty(
            "java.home"));
        System.out.println("\nYour user.home property value is: " + System.getProperty(
            "user.home"));
        File f = new File("foo.txt");
        System.out.print("\nfoo.txt does ");
        if (!f.exists())
            System.out.print("not ");
        System.out.println("exist in the current working directory.");
        return null; }
}
```

```
}
grant codebase "file:./SampleAction.jar", Principal sample.principal.SamplePrincipal "testUser" {
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "user.home", "read";
    permission java.io.FilePermission "foo.txt", "read";
};
```

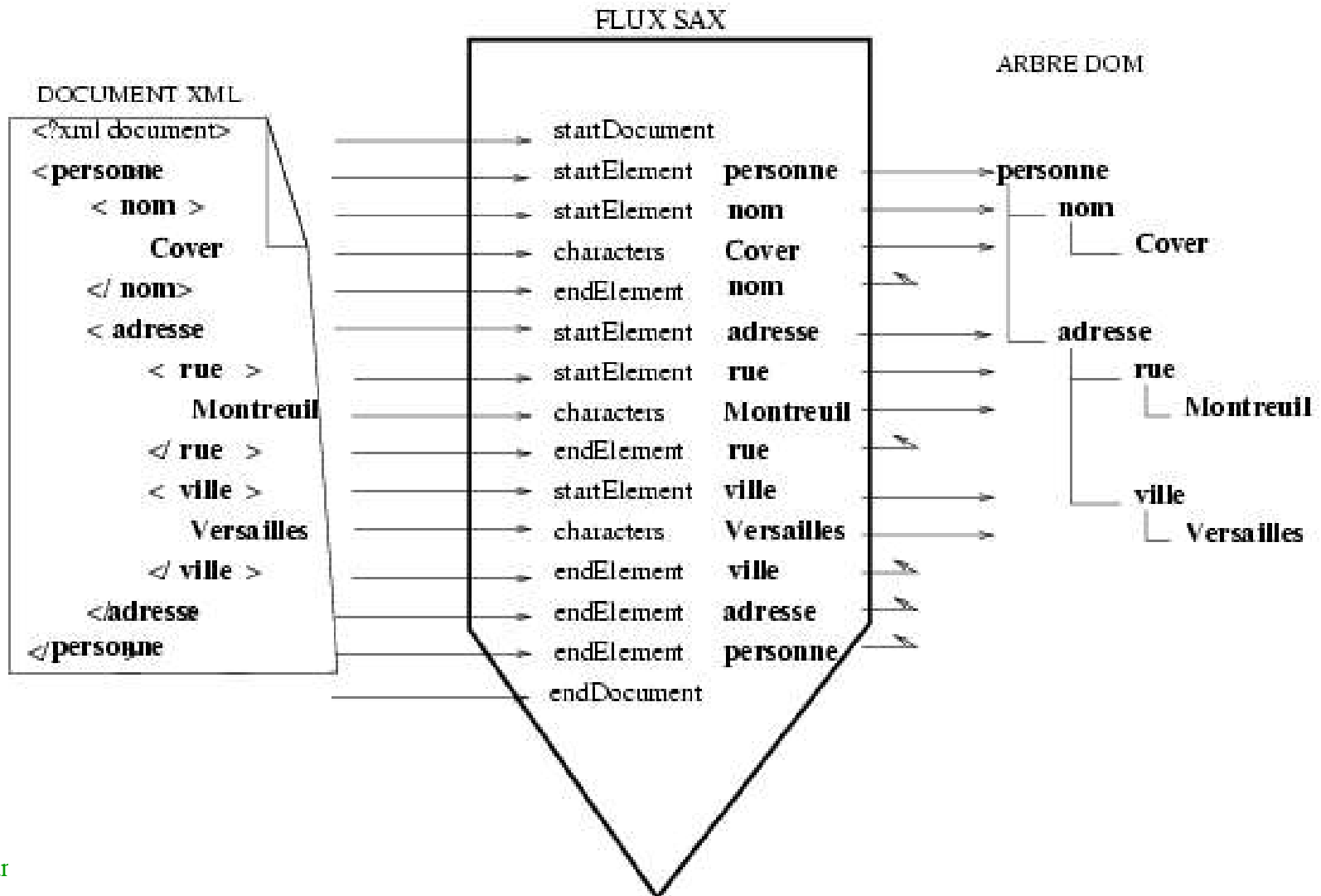
Java API for XML Processing (JAXP)



- **SAX** : gestion de flux d'événements XML
- **DOM** : structure de document XML en mémoire sous forme d'arbre
- **XSLT** : présentation de documents XML d'après des feuilles de style XSL

XML – SAX -DOM

DOCUMENT XML GENERE



API XML


- `import javax.xml.parsers ;`
- `import javax.xml.transform ;`
- `import javax.xml.transform.dom ;`
- `import javax.xml.transform.sax ;`
- `import javax.xml.transform.stream ;`
- `import org.xml.sax ;`
- `import org.xml.sax.ext ;`
- `import org.xml.sax.helpers ;`
- `import org.w3c.dom ;`
- `import org.w3c.dom.html ;`
- `import org.w3c.dom.range ;`
- `import org.w3c.dom.traversal ;`

Java pour XML



- **JAXP** : Java API for XML Parsing
 - assure interface avec analyseur de documents produisant des formats DOM ou SAX
- **JAXB** : Java Architecture for XML Binding
 - permet de construire des documents XML à partir d'objets Java et vice versa
- **JAXM** : Java API for XML Messaging
 - permet l'échange de messages XML avec des plateformes distantes en synchrone ou asynchrone
- **JAXR** : Java API for XML Registries
 - interface d'accès aux annuaires de services applicatifs
- **JAX-RPC** : JAVA API for XML-based RPC
 - appels à des procédures à distance avec XML

Plan

- 
- Architectures multi-tiers : architecture 1, 2, 3-tiers
 - J2EE : architecture générale
 - Composants J2EE : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc.
 - ■ **Outils de développement et d'intégration**
 - Produits existants
 - Conclusion

J2EE SDK



- J2EE SDK est une définition opérationnelle de la plateforme J2EE
- Fait par Sun Microsystem pour les démonstrations, les prototypes et les applications non-commerciales
- Contient :
 - J2EE Application Server
 - Serveur Web
 - Base de données relationnelles
 - API J2EE
 - un kit de développement et de déploiement
- But : permettre aux développeurs d'exécuter leurs applications J2EE et vérifier leur compatibilité

Déploiement J2EE



■ JAR – Java ARchive

- Fichier classe Java
- EJB

■ WAR - Web ARchive

- Servlets
- JSP


■ EAR - Enterprise ARchive

- Contient des JARs et WARs pour former une application complète

■ Descripteurs de déploiement

- XML

Plan

- 
- Architectures multi-tiers : architecture 1, 2, 3-tiers
 - J2EE : architecture générale
 - Composants J2EE : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc.
 - Outils de développement et d'intégration
 - ■ Produits existants
 - Conclusion

Serveurs J2EE



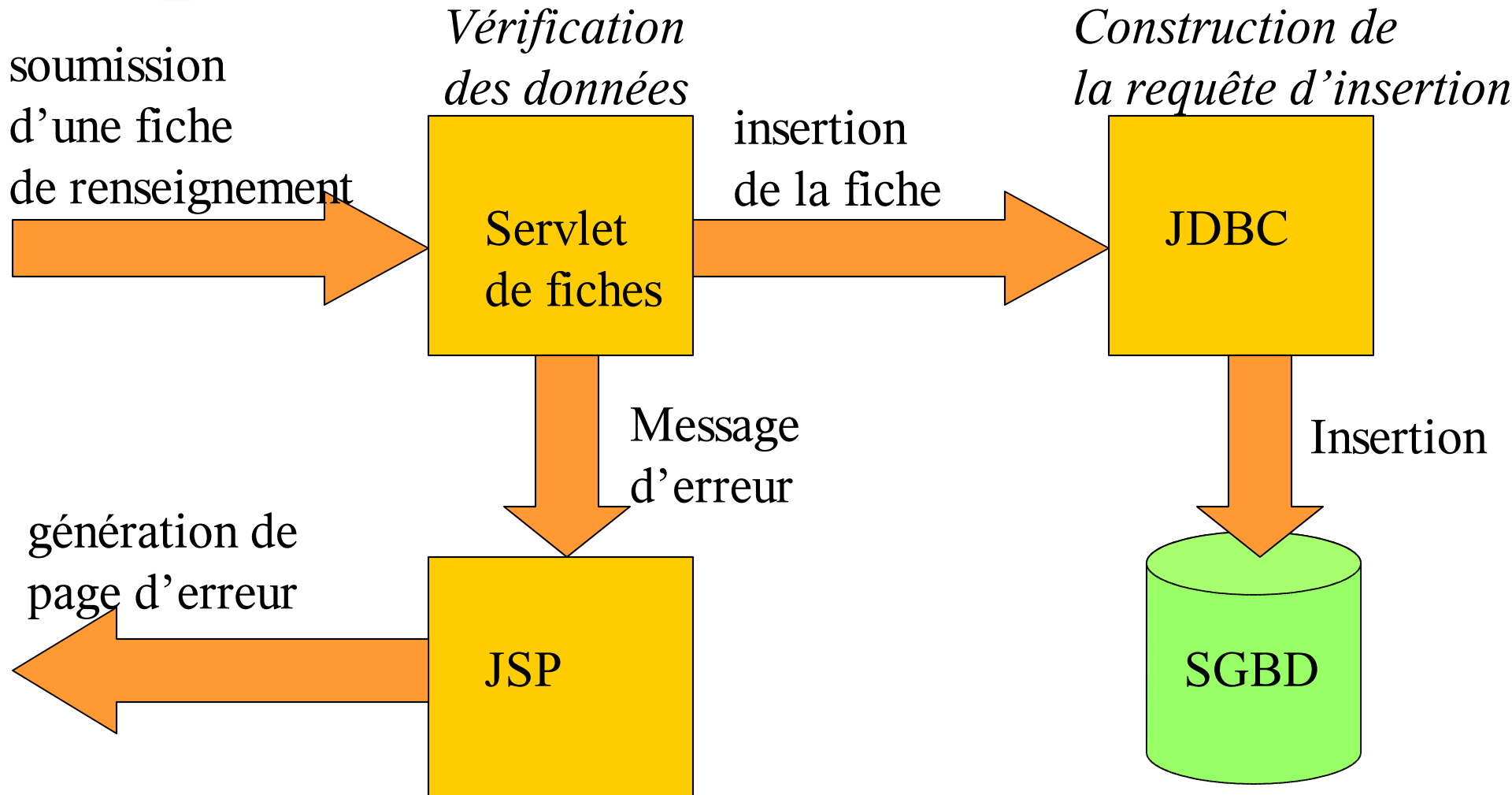
- Apache Tomcat, <http://jakarta.apache.org/>
- BEA WebLogic, <http://www.bea.com/>
- IBM WebSphere
- Sun iPlanet Application Server <http://java.sun.com/>
- Oracle Application Server <http://www.oracle.com>
- Caucho Resin, <http://www.caucho.com/>
- Allaire JRun, <http://www.allaire.com/>
- Orion, <http://www.orionserver.com/>
- SilverStream Application Server

Plan

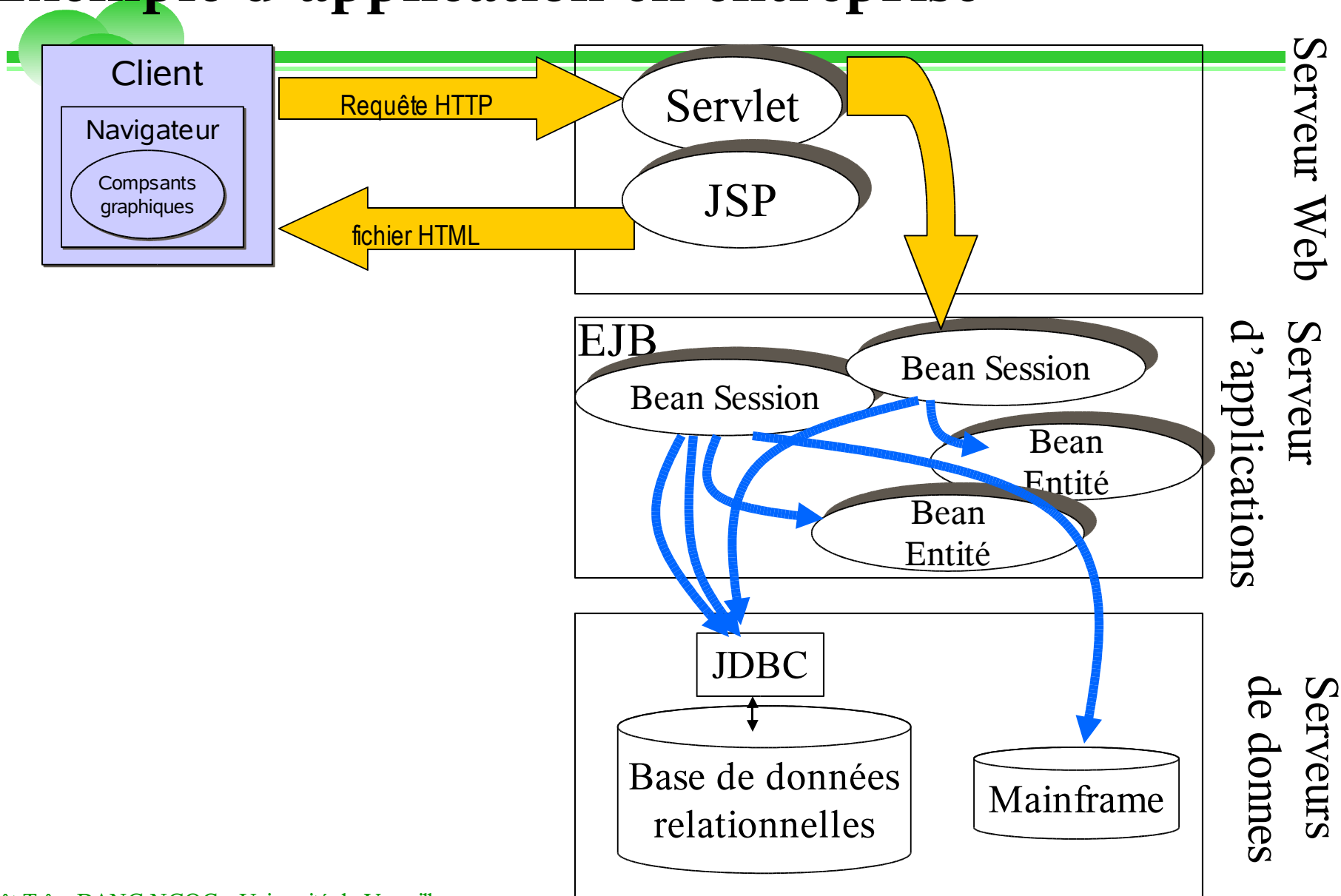


- Architectures multi-tiers : architecture 1, 2, 3-tiers
- J2EE : architecture générale
- Composants J2EE : Servlets, JSP, EJB, JMS, JAAS, JAXP, JavaMail, JNDI, etc.
- Outils de développement et d'intégration
- Produits existants
- ■ Conclusion

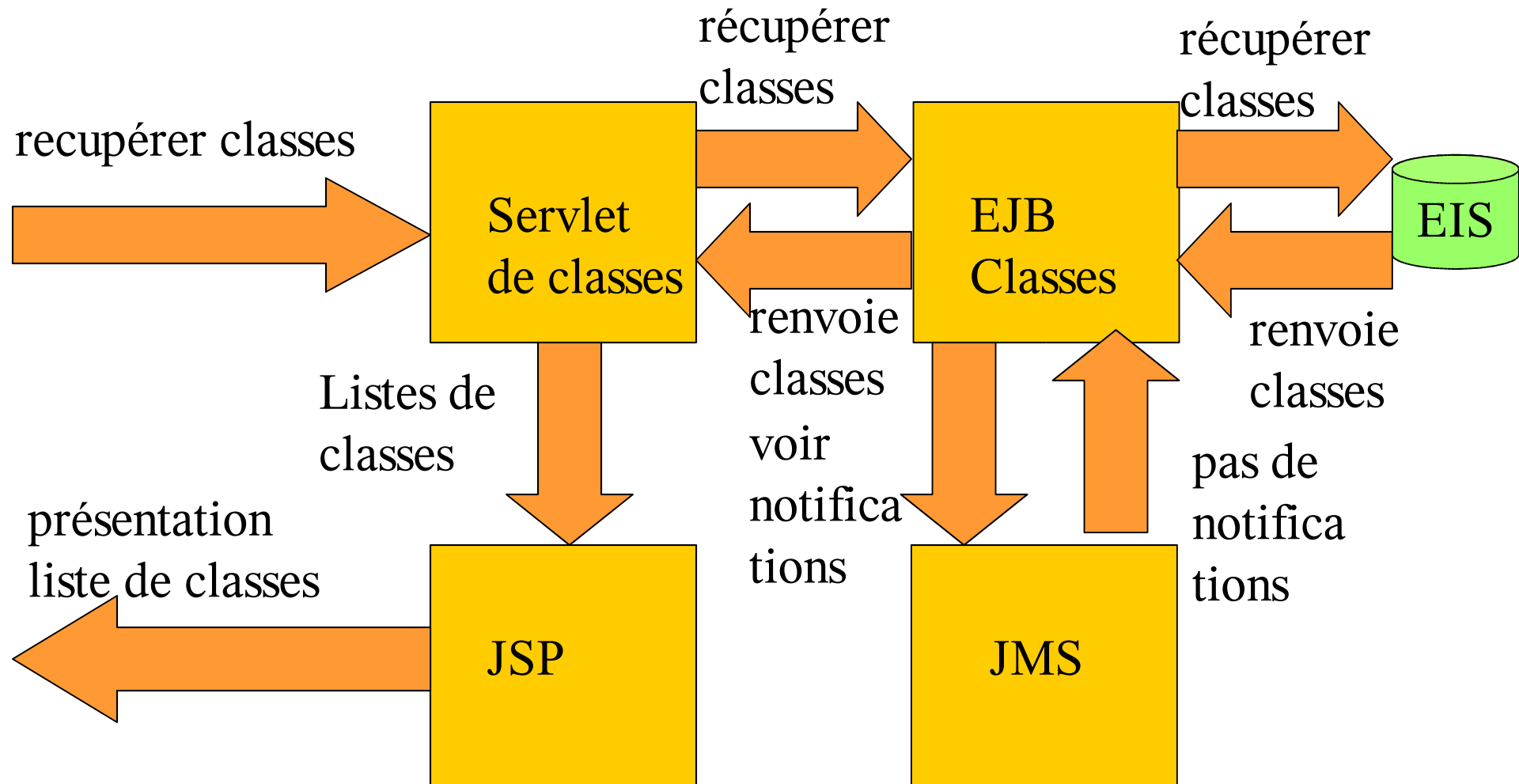
Exemple d'application en entreprise



Exemple d'application en entreprise

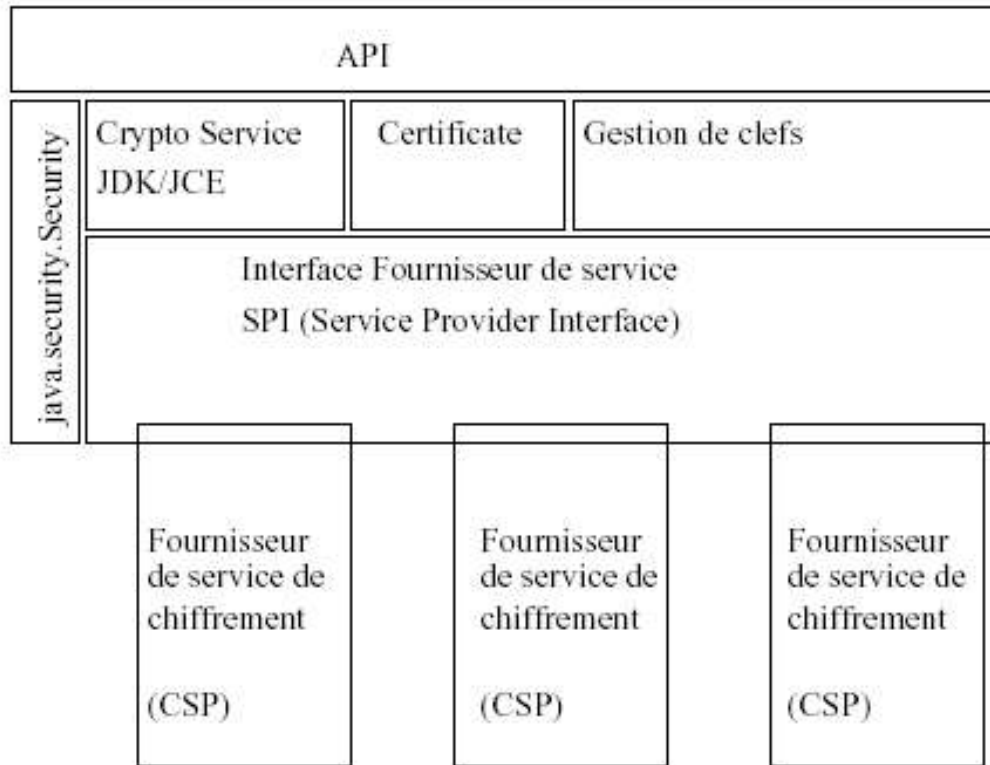


Exemple d'application en entreprise



Architecture sécurisée (JCA)

Applications



- Service de cryptographie
- Interface et classe de certificat
- Classe et interface de gestion de clef