

# FLASH



p/A EPFL – SERVICE INFORMATIQUE CENTRAL – CP 121 – CH 1015 LAUSANNE – TÉL. +41 21 693 22 11 – WEB: [HTTP://SIC.EPFL.CH](http://sic.epfl.ch)

## e(NTERPRISE)-pfl

FRANCIS.LAPIQUE@EPFL.CH, SHUBER@XO3.COM

«Les réseaux constituent la nouvelle morphologie sociale de nos sociétés, et la logique de la mise en réseau détermine largement les processus de production, d'expérience de pouvoir et de culture... Certes l'organisation en réseau a existé à d'autres époques et en d'autres lieux, ce qui est nouveau aujourd'hui c'est le fait que les technologies de l'information fournissent la base de son extension à la société tout entière.»

Manuel Castells, *L'Ere de l'Information, tome I: La Société en Réseaux*

### RÉSUMÉ

Le début 2002 représentera pour le projet e-pfl, une étape importante, marquée par une *approche applicative* des technologies Internet-Intranet. La stratégie poursuivie par ce projet sera résolument orientée vers une approche *globale* ou *entreprise* ouverte nous permettant d'accélérer et de simplifier le déploiement d'applications avec, comme perspective, l'idée d'offrir un accès à l'ensemble du système d'information. La présentation du projet e(ntreprise)-pfl se fera en deux articles, celui-ci décrit le socle technologique choisi, le suivant détaillera les différents composants.

Pour mener cette réflexion nous avons fait appel à deux consultants: Stéphane Croisier et Serge Huber de la société Hexocube.

### INTRODUCTION

A l'aube de cette ère de l'information, l'EPFL, comme toute entreprise, doit se doter d'une architecture pour son système d'information. Les pièges et défis sont nombreux et variés: fiabilité, sé-

curité et intégration des données, évolutivité des applications. Pour répondre à ces enjeux, il est nécessaire de disposer d'une infrastructure ouverte sur un ensemble de standards qui favorise le plus possible l'approche composants Web. A l'image de ce que l'on trouve dans le développement logiciel, on entend par composants Web des unités de développement et de déploiement présentant des interfaces et des services bien définis. Le développeur d'applications en s'appuyant sur cet éventail de fonctionnalités *normalisées* se concentre sur le développement de l'application elle-même. Cette approche ne laisse pas l'industrie indifférente! C'est la seule, dans la limite des technologies dont nous disposons aujourd'hui, qui soit en mesure de répondre aux problématiques de déploiement, de sécurité, d'administration et de changement de facteur d'échelle. Dans ce cadre il est nécessaire de disposer d'un environnement devant assurer la cohérence de cet ensemble.

SUITE EN PAGE 5

## SOMMAIRE FI10

- 1 e(nterprise)-pfl  
Francis Lapique & Serge Huber
- 2 Flash – PHP – MySQL  
le trio de choc  
Sylvain Demierre &  
Eric Tonicello
- 10 FileMaker Pro 5  
Isabelle Fernandez
- 12 Programme des cours
- 16 Calendrier



### PROCHAINES PARUTIONS

	déLAI RÉDACTION	PARUTION FI
1	03.01.02	22.01.02
2	07.02.02	26.02.02
3	07.03.02	26.03.02
4	11.04.02	30.04.02
5	16.05.02	04.06.02
6	20.06.02	09.07.02

## EVOLUTION VERS LE TRANSACTIONNEL

Actuellement la plate-forme J2EE (pour Java 2 Enterprise Edition), initiée par Sun, reprise par de grands éditeurs comme IBM, Bea ou Oracle et tout récemment par SAP, remporte un franc succès. Succès contré par l'initiative .NET de Microsoft qui est en train de se mettre en place. Ce projet qui trouve aussi ses racines dans J2EE se différencie de son aîné comme nous le verrons plus loin.

Avant d'aller plus en avant, il faut noter que la réalité d'un projet comme celui d'e-pfl est complexe car nous arrivons à un tournant d'Internet. Ce média qui classiquement s'inscrivait dans une perspective non-transactionnelle avec la mise en ligne de documents hypertextes HTML s'oriente de plus en plus vers un mode de fonctionnement transactionnel pour lequel il n'est pas fait. Le projet e-pfl comme tant d'autres qui lui ressemblent doit négocier ce tournant.

## ARCHITECTURE N-TIERS

Jusqu'alors une majorité d'applications Web pouvait se ramener à une architecture à deux niveaux, avec d'un côté un client (dans la plupart des cas un navigateur) et de l'autre un serveur Web. Les deux communiquant au travers du protocole HTTP (fig.1).

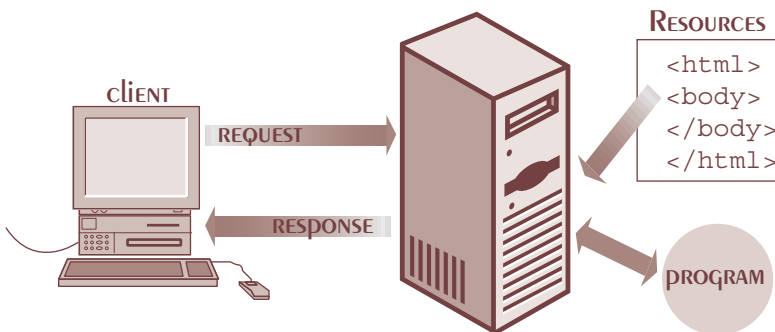


fig. 1

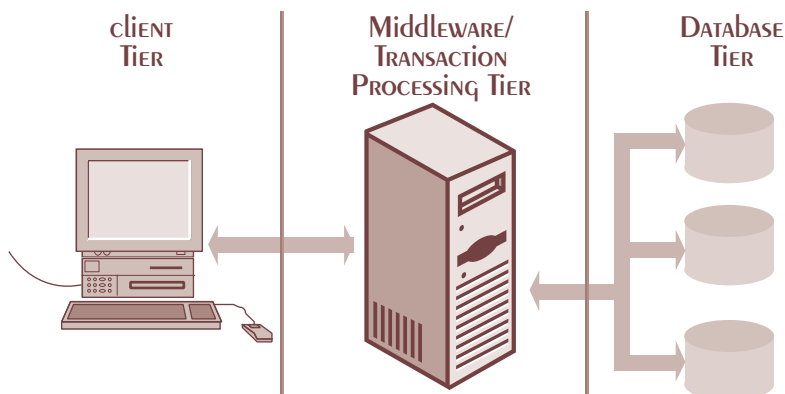


fig. 2

Pour faire face au tournant évoqué plus haut et pour répondre à des exigences de sécurité, par exemple celles demandées par des applications prenant en charge des achats ou transactions bancaires depuis le domicile, on assiste de plus en plus au déploiement d'architectures n-tiers où n est généralement égal à trois. Ces architectures 3-tiers distinguent, comme leur nom le laisse entendre, trois niveaux, celui dit de présentation, celui dit de métier ou de logique applicative et celui des données (fig. 2):

- le premier niveau est chargé de gérer la logique de navigation à l'aide de composants de présentation;
- le deuxième regroupe un ensemble de composants gérant la logique métier dans un serveur dit d'applications;
- le troisième abrite les *données* des systèmes existants.

L'approche entreprise J2EE fournit le cadre, ou *framework*, pour harmoniser le développement des trois niveaux.

## J2EE

J2EE reprend l'architecture 3-tiers avec des composants ou objets permettant de communiquer avec un navigateur et avec des bases de données, et un ensemble d'APIs Java, construites sur la plate-forme Java 2 pour communiquer avec d'autres types de serveurs. Pour les amateurs d'acronymes,

J2EE comprend notamment: EJB (Enterprise JavaBeans), JSP (JavaServer Pages), Servlet, JNDI (Java Naming and Directory Interface), RMI (Remote Method Invocation), JDBC (Java Database Connectivity), JavaMail, JTS (Java Transaction Service), JMS (Java Messaging Service). Quelques pages ne permettent pas de passer en revue tous ces composants qui sont par ailleurs largement couverts par une littérature abondante.

En allant à l'essentiel, les composants Web, Servlets et JSP sont destinés à mettre en forme l'application pour le navigateur, les composants EJB, conçus pour être réutilisables, contiennent la logique métier. Sans trop rentrer dans le détail, certains de ces EJB offrent des services tandis que d'autres contiennent des données persistantes.

A remarquer: une application J2EE est emballée dans un fichier EAR (Enterprise Archive). Ce fichier contient tous les composants de l'application: les composants Web (Servlets, JSP), les Enterprise JavaBeans (EJB) et un ensemble de fichiers *descriptifs* au format XML. Placé dans votre environnement J2EE, ce fichier EAR se déploiera comme par magie pour rendre un nouveau service. Nous verrons plus loin un exemple de déploiement.

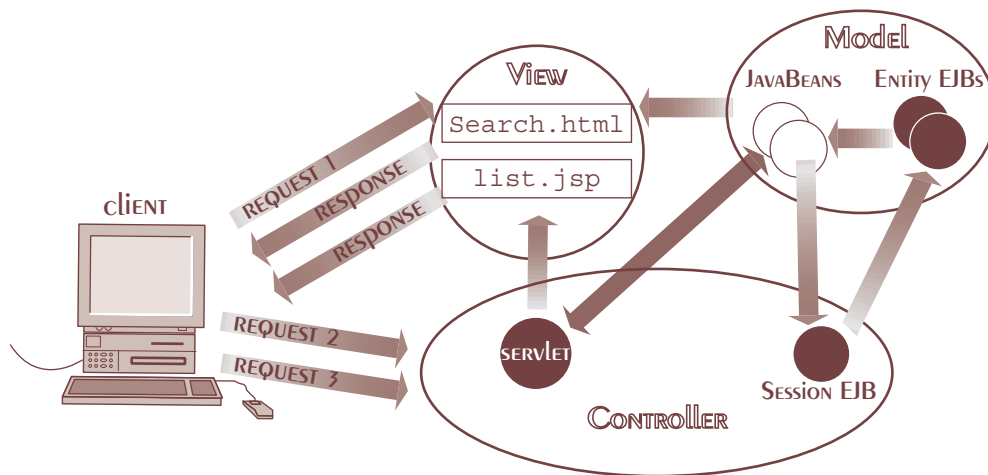


fig. 3

J2EE est une plate-forme *100% Java*, elle hérite des principales qualités de ce langage, portabilité et sécurité. Mais c'est un environnement contraignant, obligeant à une grande rigueur et faisant appel à de nombreuses technologies. Son concurrent .NET a comme parti pris celui de laisser au programmeur plus de libertés au niveau du choix des langages, ainsi qu'une approche composants un peu différente.

L'architecture J2EE trouve ses fondements dans le concept Modèle/Vue/Contrôleur qui a fait son apparition dans les années 80. Dans ce modèle, les JSP fournissent la vue, les Servlets contrôlent et dispatchent, les composants EJB servent à accéder aux données du modèle (fig. 3).

Dans le cadre de cet article nous ne pouvons pas aborder le détail des modules qui composent J2EE. Nous allons nous limiter à quelques exemples de Servlets JSP et EJB pour illustrer quelques éléments clefs de la plate-forme J2EE.

## LES SERVLETS

Les Servlets sont des composants Web qui communiquent avec le navigateur à travers le protocole HTTP. Ils s'exécutent dans un conteneur Web, conteneur que vous allez trouver tout intégré dans des environnements J2EE du commerce (iPlanet, WebSphere, etc.), intégré à des serveurs Web (Java WebServer) ou dans un environnement d'exécution séparé (TomCat, <http://jakarta.apache.org>). Le conteneur prend à sa charge la gestion des connexions réseaux et des concurrences d'accès en gérant un

*pool de threads*. Les classes et les interfaces pour les Servlets sont définies dans deux packages: `javax.servlet` et `javax.servlet.http`. Le premier est un package générique indépendant des protocoles, le second est spécifique à HTTP.

Du point de vue du programmeur, une Servlet est une sous-classe de `HttpServlet` dans laquelle on surcharge la ou les méthodes `doXXX(HttpServletRequest request, HttpServletResponse response)` où XXX désigne le type de requêtes HTML (pour GET, la méthode est `doGet`, etc.). Le code 1, `RequestParamExample`, montre pour les programmeurs un exemple d'application des méthodes GET et POST; notez l'utilisation d'un objet `PrintWriter` obtenu à l'aide de la méthode `getWriter()` sur l'objet `HttpServletResponse` pour envoyer un message au navigateur.

Les méthodes `getParameter` et `getParameterNames` appliquées sur un objet `HttpServletRequest` pour récupérer les paires clés/valeurs sont également utiles.

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*; // les deux packages nécessaires aux servlets

public class RequestParamExample extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html"); // type de la réponse de la servlet
        PrintWriter out = response.getWriter();
        out.println("GET Request. No Form Data Posted");
    }

    public void doPost(HttpServletRequest request, HttpServletResponse res)
    throws IOException, ServletException
    {
        Enumeration e = request.getParameterNames(); // récupère le noms des paramètres
        PrintWriter out = res.getWriter ();
        while (e.hasMoreElements()) {
            String name = (String)e.nextElement();
            String value = request.getParameter(name);
            out.println(name + " = " + value);
        }
    }
}
```

code 1

HTTP est un protocole sans état. Le client ouvre une connexion sur le serveur, émet sa requête, le serveur expédie la réponse et coupe la connexion. Tout recommence à zéro à chaque nouvelle requête. Quand il est nécessaire d'avoir un suivi des requêtes-réponses la solution consiste à échanger une forme quelconque de jeton entre client et serveur (champs masqués, cookies, session SSL...). Les Servlets proposent un mécanisme très simple de connexion géré par le conteneur: l'interface `HttpServletRequest` dispose de la méthode `getSession()` pour obtenir un objet `HttpSession`, qui permet de manipuler des paires clés/valeurs.

```
HttpSession session = req.getSession();
session.putValue(name, value);
(String) session.getValue(name);
```

En résumé, dans un modèle standard d'utilisation, les Servlets reçoivent des requêtes HTTP, exécutent une logique applicative, renvoient une réponse et les sessions sont automatiquement gérées par le conteneur d'applications en utilisant des cookies ou des identifiants encodés dans les URLs. On peut mettre sur pied des chaînages de Servlets ou des mécanismes de délégation de requête.

La grande majorité des lecteurs de cet article n'auront jamais à développer la moindre Servlet, par contre c'est quelque chose qu'ils manipuleront tous les jours en intégrant tel ou tel service Web.

## LES JSP ( JAVASERVER PAGES )

Pour simplifier l'écriture de la couche présentation et aller vers une séparation plus nette du contenu statique du code applicatif, on a avancé l'idée des pages JSP. Une page JSP est un fichier texte contenant du HTML et des fragments de code Java. Ce fichier JSP (code 2) est traduit en Servlet, compilé puis instancié lors de son évocation via une URL. Si le fichier JSP a subi des modifications, alors il sera recompilé (fig. 4).

Le code Java est inséré dans les JSP entre des balises `<% %>`. Ce bloc de code Java porte le doux nom de scriptlet. Il existe 3 autres types de directives:

- `<%@ .. %>` pour inclure une ressource dans la page JSP
- `<%! .. %>` pour définir des variables et des méthodes. Elles possèdent une portée de classe et sont initialisées en même temps que la page JSP
- `<%=...%>` pour renvoyer la valeur d'une expression vers le client.

Voici un exemple de page JSP:

```
<HTML>
<HEAD>
<TITLE>Hello</TITLE></HEAD>
<BODY>
<H1>
<% out.println("Hello World");
String[] noms= {"e-pfl","j2ee"};
for ( int i=0; i < noms.length; i++ ) {
%>
Le <%= i %> ème nom est <%= noms[i] %>
<%}%>

</H1>
</BODY>
</HTML>
```

code 2

Les pages JSP (code 3) offrent un mécanisme très intéressant, celui d'incorporer des composants réutilisables au moyen d'extensions de balises ou *tags*. L'exemple *place.jsp* n'est autre que la version jsp de la place centrale du site de l'EPFL où nous avons choisi l'approche *template*.

Le template contient tout le code HTML dans lequel on a défini des sections, une section pour le menu, une autre pour le sous-menu, etc. Le travail de l'éditeur de la page se résume à écrire des petits bouts de code HTML qu'il va associer aux différentes sections.

Si nous regardons dans le détail, la première ligne donne la correspondance entre nom et chemin d'accès du tag. La balise `<region:render template="e-pfl.jsp">` marque le début d'affichage et `</region:render>` la fin. Entre les deux on associe chacune des sections aux morceaux de code HTML.

L'avant-dernière ligne fait intervenir la notion de rôle. Si le profil client est de type **admin**, alors on affichera le fichier *goodies.html*, sinon pas.

Voici le morceau de code de *menu.html*:

```

```

Nous ne pouvons pas dans le cadre de cet article aborder l'implémentation de la balise elle-même. Sachez seulement que c'est une sous-classe du package `javax.servlet.jsp.tagext`.

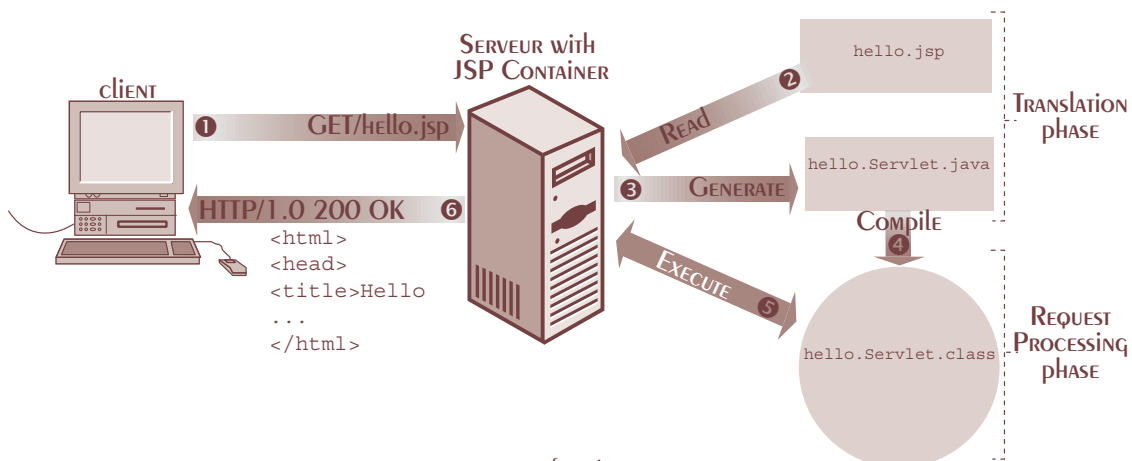


fig. 4

```
<%@ taglib uri='regions' prefix='region' %>
<region:render template='e-pfl.jsp'>
<region:put section='menu' content='/menu.html' />
<region:put section='sous_menu' content='/sous_menu.html' />
<region:put section='langue' content='/langue.html' />
<region:put section='splash' content='/splash.html' />
<region:put section='bloc1' content='/bloc1.html' />
<region:put section='bloc2' content='/bloc2.html' />
<region:put section='tool' content='/tool.html' />
<region:put section='goodies' role='admin' content='/
goodies.html' />
</region:render>
```

code 3

## CONTEXTE ET DÉPLOIEMENT

Une application Web ne peut s'exécuter que dans un contexte. A chaque contexte correspond une arborescence dans le système de fichiers qui contient les ressources accédées lors des requêtes vers le conteneur. Cette arborescence est identique pour chaque contexte. Voici comment se décompose la structure des répertoires :

- la racine: elle fait office de répertoire racine pour les ressources qui font partie du contexte. Par exemple l'URL *http://monserveur/epfl/index.html* fait référence au fichier index.html du répertoire racine «epfl» qui est donc également le nom du contexte;
- le répertoire WEB-INF, situé à la racine, contient un fichier web.xml qui est le descripteur de déploiement du contexte. Il contient tous les paramètres de configuration utilisés par le contexte;
- le répertoire WEB-INF/classes/, contient la logique applicative sous forme de classes.

Toute cette arborescence peut être regroupée dans une archive de la même manière qu'une application Java classique, en utilisant l'utilitaire (pour *Java Archive tool*). De cette façon il n'y a plus qu'un seul fichier à manipuler et votre application peut être signée, afin de la rendre digne de confiance auprès des gens qui utiliseront votre application Web.

Le fichier **web.xml** est écrit en XML et sa structure obéit aux règles d'une DTD définie spécifiquement par Sun.

Les informations données par le fichier de configuration sont:

- les paramètres d'initialisation du contexte;
- la configuration de la session;
- les définitions des Servlets et des JSPs;

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/Web-app_2.2.dtd">
<Web-app>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <taglib>
    <taglib-uri>regions</taglib-uri>
    <taglib-location>/Web-INF/tlds/regions.tld</taglib-location>
  </taglib>
</Web-app>
```

code 4

- les correspondances entre Servlets et entre JSPs;
- les correspondances entre types MIME;
- la liste des fichiers de bienvenue;
- les pages d'erreur;
- la sécurité.

## LES EJB

Nous avons vu le conteneur Web qui fournit le cadre de *vie* des Servlets et JSP où est implémentée la logique de présentation.

Mais il existe d'autres conteneurs comme celui pour les EJB qui prend en charge la gestion des composants, on y retrouve la logique métier. Il existe trois types d'EJB avec différentes responsabilités:

### LES ENTITY BEANS

Ils représentent les données persistantes, à savoir des données qui sont par exemple stockées dans des bases de données, ou extraites d'autres systèmes de stockages, comme des bases LDAP ou des systèmes de fichiers plats. Néanmoins la principale utilisation des Entity beans concerne des données stockées dans des bases de données. Au sein des Entity beans on trouve deux types de JavaBeans. Le premier type appelé *Bean-Managed Persistence* (BMP) est très simple. Il s'agit tout simplement de JavaBeans qui accèdent à une base de données en implémentant des appels JDBC (*Java Database Connectivity*) et qui manipulent donc des requêtes SQL. Le défaut de ce type de modules est la programmation de requêtes SQL qui n'est pas forcément très portable mais qui a au moins le mérite d'être une problématique connue. Le second type d'Entity beans s'appelle *Container-Managed Persistence* (CMP) et se veut nettement plus portable, et également très intéressant aussi au niveau de la simplification de la gestion des bases de données. Dans ce modèle, le JavaBean ne fait que déclarer les types de données qu'il souhaite rendre persistant (par exemple un entier, une chaîne de caractères, etc.) et le conteneur d'EJB se charge automatiquement de créer la table nécessaire dans la base de données, ainsi que de charger/stocker les données. La description des types de données ainsi que les relations entre Entity Beans (proche d'un modèle Entités-Relations dans les modélisations de bases de données relationnelles classiques) sont faites à l'aide des fichiers XML de déploiement, dont nous donnons un exemple dans le code 5.

### LES SESSION BEANS

Ils représentent des données de sessions. Ceux-ci sont donc *rattachés* à un utilisateur, et peuvent contenir toutes les données transitoires ainsi que les opérations offertes à l'utilisateur. A nouveau ici on peut trouver deux types de Session Beans, ceux appelés les *Stateless sessions beans* et les *Stateful*. La différence est par contre nettement plus importante ici que pour les Entity beans: les stateful ses-

```

<ejb-jar>
<display-name>MusicCDs</display-name>
<enterprise-beans>
<entity>
<description>Models a music CD</description>
<ejb-name>CDBean</ejb-name>
<home>org.jboss.docs.cmp.cd.interfaces.CDHome</home>
<remote>org.jboss.docs.cmp.cd.interfaces.CD</remote>
<ejb-class>org.jboss.docs.cmp.cd.bean.CDBean</ejb-class>
<persistence-type>Container</persistence-type>
<prim-key-class>java.lang.Integer</prim-key-class>
<reentrant>False</reentrant>
<cmp-field><field-name>id</field-name></cmp-field>
<cmp-field><field-name>title</field-name></cmp-field>
<cmp-field><field-name>artist</field-name></cmp-field>
<cmp-field><field-name>type</field-name></cmp-field>
<cmp-field><field-name>notes</field-name></cmp-field>
<primkey-field>id</primkey-field>
</entity>
<!-- ... autres beans ... -->
</enterprise-beans>
<!-- ... autres déclarations... -->
</ejb-jar>

```

code 5

sion beans stockent des valeurs entre deux appels de méthodes, alors que les stateless n'ont aucune notion d'état entre deux appels de méthode.

### LES MESSAGE BEANS

Ils sont le nouveau type d'EJB introduit dans la norme EJB 2.0 publiée en septembre dernier, et offrent la possibilité d'implémenter des modules qui répondent à des messages, à savoir des composants asynchrones qui échangent des messages plutôt que de faire appel à des fonctions synchrones. Ces beans offrent donc la possibilité de gérer des processus qui peuvent être longs, et dont l'ordre n'a pas forcément une importance primordiale.

### L'ALTERNATIVE MICROSOFT

Face au succès de Sun et de sa plate-forme J2EE, Microsoft riposte en proposant la plateforme .NET. La principale caractéristique de cette plate-forme est l'unification autour d'un interpréteur appelé CLI (Common Language Interpreter), ainsi que l'intégration de composants COM/DCOM. Pour beaucoup Passport.NET fait partie intégrante de la nouvelle plate-forme, mais Microsoft risque fort de découpler les deux si les réticences se font grandissantes. .NET s'est récemment doté d'une espèce de langage Java appelé J# mais qui ne semble convaincre personne. Il est néanmoins clair que Microsoft aura sa place avec sa plate-forme, mais que les intégrations avec les autres systèmes tel que J2EE resteront les enjeux du succès.

### INTÉGRATION GLOBALE

Il existe déjà plusieurs possibilités d'intégration de plates-formes Java et Microsoft, notamment les interfaces JNI (Java Native Interface) qui permettent d'appeler des fonctions système directement, ou encore des ponts COM-Java qui ont l'avantage de découpler à travers le réseau le code Java du code COM/DCOM. Mais il existe aussi des appro-

ches plus novatrices, comme SOAP (Simple Object Access Protocole) qui est une norme proposée par le W3C. Le principe est relativement simple, permettre des appels de méthodes distants (RPC) à travers l'utilisation de protocole Web, tel que HTTP et XML. L'appel est donc codé en XML et transmis via une connexion HTTP à l'objet qui répond à l'interface utilisée. La réponse est également encodée en XML et renvoyée comme une réponse HTTP. On peut donc par exemple imaginer développer une Servlet qui implémente un appel SOAP. Un autre aspect intéressant de SOAP c'est qu'il a l'aval de Microsoft, Sun, IBM et de la fondation Apache.

### CONCLUSION

Le projet e-pfl se doit d'être un projet ouvert. Le choix de la technologie J2EE est à la fois le choix d'une technologie bien éprouvée dans l'industrie ainsi qu'une plate-forme offrant des possibilités d'intégration intéressantes. La politique de la plate-forme J2EE est une approche basée sur des interactions entre des composants de taille raisonnable, plutôt qu'un système monolithique qui cherche à résoudre toutes les problématiques de façon centralisée.

### DEUX LIVRES DE RÉFÉRENCE

- Core J2EE Patterns: Best Practices and Design Strategies
- J2EE(tm) Technology in Practice: Building Business Applications with the Java(tm) 2 Platform, Enterprise Edition ■

