Antonio **Goncalves**

les Cahiers du Programmeur

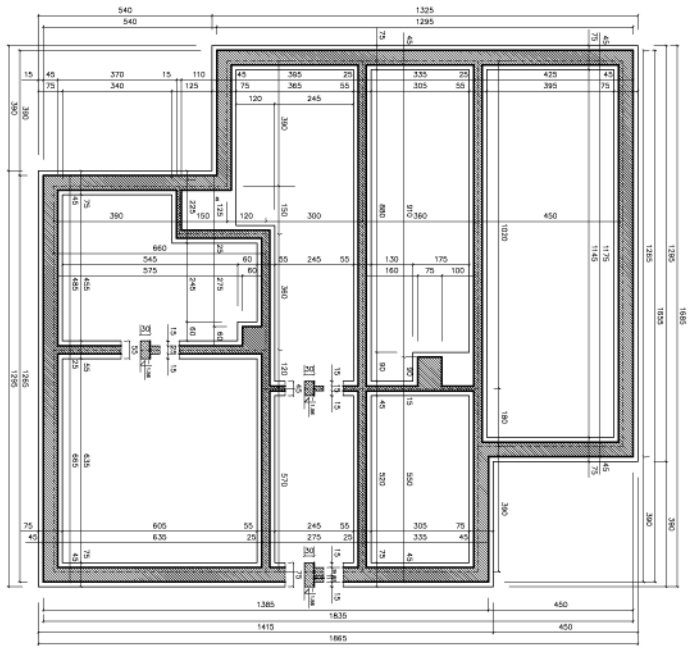
Java EE 5

2e édition

ISBN : 978-2-212-12363-0

EYROLLES

chapitre 2



Architecture de l'application

Dans le précédent chapitre, nous avons décrit le comportement souhaité de la future application de commerce électronique de la société YAPS. Vous allez maintenant découvrir les technologies utilisées pour développer cette application, c'est-à-dire le langage Java et la nouvelle plate-forme Java EE 5. Grâce à l'utilisation de diagrammes UML de composants et de déploiement, nous allons modéliser et décrire l'architecture logicielle de l'application YAPS Pet Store. Celle-ci s'inspire du Blueprint Java Pet Store de Sun et de ses design patterns. Elle est architecturée en couches (présentation, traitements et accès aux données) et utilise la plate-forme Java EE 5 qui s'appuie sur les nouveautés du langage Java 5 (annotations, génériques...).

SOMMAIRE

- Technologies utilisées
- Nouveautés du langage Java 5
- La plate-forme Java EE 5
- Les Blueprints et le Pet Store de Sun
- Architecture en couches

MOTS-CLÉS

- Java SE
- Java EE
- EJB
- JPA
- JMS et MDB
- JSP, JSTL
- JSF
- XML
- Web Service
- Design pattern
- UML

Présentation des langages utilisés

Java SE 5

Avant de parler de Java Enterprise Edition 5 (Java EE 5), il est nécessaire de présenter brièvement le langage sur lequel s'appuie cette plate-forme : Java 5.0. La version 5 de Java SE, ou *Java Standard Edition*, est une révision majeure du langage Java créé en 1995 par l'équipe de James Gosling. Cette version apporte de nombreuses nouveautés telles que l'autoboxing, les annotations, les génériques, une nouvelle boucle d'itération, les types énumérés, les méthodes à arguments variables, les imports statiques et bien d'autres encore. De nouveaux outils ainsi que de nouvelles API ont vu le jour ou ont été considérablement enrichis, comme l'API de concurrence, l'API de thread, la supervision de la JVM, etc.

API

Une API, ou *Application Programming Interface*, définit la manière dont un composant informatique peut être invoqué par un autre. Il s'agit généralement d'une liste de méthodes que l'on peut appeler.

À LIRE Java 5

Il existe beaucoup de livres sur le langage Java ainsi que de nombreuses références et articles en ligne, notamment :

- Emmanuel Puybaret, *Java 1.4 et 5.0*, Eyrolles, 2006
- Claude Delannoy, *Programmer en Java*, Eyrolles, 2007
- Anne Tasso, *Le livre de Java premier langage*, Eyrolles, 2006
- http://java.sun.com/docs/books/jls/

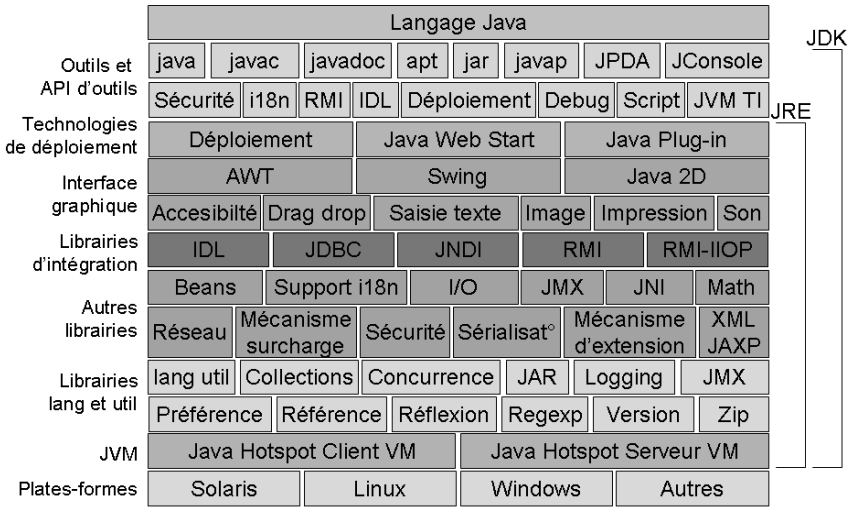


Figure 2–1
L'architecture de Java SE 5

La figure 2-1 montre les composants qui constituent Java SE 5. Cet ouvrage n'a pas la prétention de vous expliquer toutes les nouveautés du langage. Toutefois, il est important de s'attarder sur celles qui seront utilisées tout au long de notre étude de cas. La suite de cette partie tend à présenter succinctement les éléments caractéristiques de Java SE 5 auxquels nous allons nous confronter.

JAVA 5 Les technologies

Vous retrouverez sur le site de Sun la liste des outils, API et librairies que contient Java 5 :

- http://java.sun.com/javase/technologies/index.jsp

Autoboxing

Le langage Java s'appuie sur des types primitifs pour décrire les types de base (`byte`, `short`, `int`, `long`, `double`, `float`, `boolean` et `char`). Toutefois, comme tout n'est qu'objet en Java, il est nécessaire de constamment

encapsuler ces types primitifs dans des classes de référence (`Integer` pour `int`, `Double` pour `double`, etc.).

Ce type de transformation (appelé *boxing*) peut rapidement s'avérer pénible. D'autant que le processus inverse (appelé *unboxing*) est nécessaire pour retrouver son type primitif initial. Avec Java 5, cette conversion explicite devient obsolète puisque l'autoboxing convertit de manière transparente les types de base en références et réciproquement. Bien sûr, vous pouvez continuer à utiliser uniquement les types primitifs si vous le souhaitez.

Exemple d'autoboxing

```
// Boxing explicite entre un Integer et un int
Integer objet = new Integer(5);
int primitif = objet.intValue();

// Autoboxing
Integer objet = 5;
int primitif = objet;
```

Annotations

Une annotation permet de marquer (on dit alors *annoter*) certains éléments du langage Java afin de leur ajouter une propriété particulière. Ces annotations peuvent ensuite être utilisées à la compilation ou à l'exécution pour automatiser certaines tâches. Une annotation peut être utilisée sur plusieurs types d'éléments (paquetage, classe, interface, énumération, annotation, constructeur, méthode, paramètre, attribut de classe ou variable locale).

Exemple d'utilisation d'annotations

```
@CeciEstUneAnnotationSurUneClasse
public class MaClasse {

  @UneAnnotationSurUnAttribut
  private Date unAttribut;

  @SurUneMethode
  private void maMethode() {
    return;
  }
}
```

Comme vous le verrez tout au long de cet ouvrage, Java Enterprise Edition 5 utilise très fréquemment les annotations. Nous aurons l'occasion de nous y attarder plus longuement par la suite.

JAVA 5 JConsole

La JConsole est un utilitaire de surveillance fourni avec Java SE 5. Liée aux technologies JMX et MBean, la JConsole permet de surveiller et superviser les applications Java (occupation mémoire, threads en cours, classes chargées...) tout comme de prendre en charge certaines opérations (appeller le garbage collector, changer le niveau des logs, etc.).

INFORMATION Acronymes

La plate-forme Java est extrêmement riche. Elle a donc tendance à utiliser abondamment et à abuser d'acronymes en tout genre (souvent commençant par la lettre « J »). Vous trouverez en annexe un lexique d'acronymes et de sigles.

Génériques

Pour les personnes familières des templates C++, les génériques sont simples à comprendre, même si leur fonctionnement n'est pas du tout similaire. Ils permettent de ne pas spécifier le type à la compilation (paramètre ou retour de méthode, par exemple), tout en assurant que le type reste cohérent dans ses différentes utilisations. Il est par exemple possible de spécifier qu'une collection (un objet `Vector`, `Hashtable` ou `Array`, par exemple) ne peut être remplie que par un type de classe donné (`Vector<Integer>` pour déclarer un vecteur d'entiers). Il n'est donc plus nécessaire d'effectuer le contrôle du type au moment de l'exécution.

Exemple d'un vecteur générique

```
// Sans générique
Vector nombres = new Vector();
for (int i = 0; i < nombres.size(); i++) {
  Integer nombre = (Integer) nombres.elementAt(i);
}

// Avec générique
Vector <Integer> nombres = new Vector<Integer>();
for (int i = 0; i < nombres.size(); i++) {
  Integer nombre = nombres.elementAt(i);
}
```

Comme on peut le constater dans le fragment de code ci-dessus, le parcours des éléments du vecteur avec générique présente une meilleure lisibilité ainsi qu'une plus grande robustesse.

APPROFONDIR Annotations et génériques

Si vous voulez en savoir plus sur les annotations et les génériques, consultez le tutoriel de Sun. Vous y trouverez une information mise à jour ainsi que des exemples de code.

- http://java.sun.com/docs/books/tutorial

Les types énumérés

Java 5.0 introduit le nouveau mot-clé `enum`, utilisable comme le mot-clé `class`. Sa particularité est qu'il représente un type énuméré, c'est-à-dire un type qui n'accepte qu'un ensemble fini d'éléments. Il peut donc être utilisé pour définir un ensemble de constantes.

Exemple d'une énumération

```
public enum Saisons {
  PRINTEMPS,
  ETE,
  AUTOMNE,
  HIVER
};
```

APPROFONDIR Les types énumérés

Les types énumérés offrent d'autres possibilités : itération, utilisation dans un `switch`, `EnumSet`, `EnumMap`, etc. Pour en savoir d'avantage, consultez le site de Sun à l'adresse :

- http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html

RAPPEL Périmètre de ce livre

Ce livre n'a pas la prétention de vous enseigner les nouveautés du langage Java mais uniquement de vous présenter celles qui vont être utilisées lors de la réalisation de l'application YAPS Pet Store. Si vous n'êtes pas encore à l'aise avec les annotations, les génériques ou les types énumérés, reportez-vous aux références données dans cet ouvrage pour approfondir vos connaissances.

Swing

La plupart des applications destinées aux utilisateurs comportent des éléments graphiques de telle sorte qu'elles soient plus conviviales et

ergonomiques. La plate-forme Java dispose des API AWT et Swing permettant de construire des interfaces homme-machine (IHM) sophistiquées en client lourd.

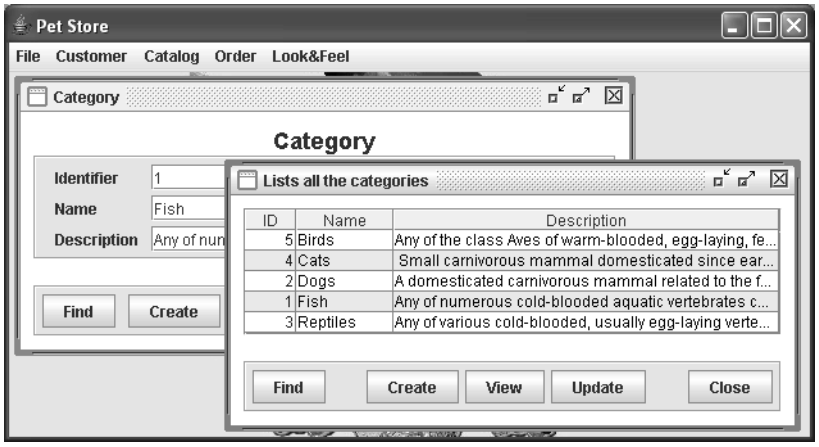


Figure 2–2
Application cliente développée en Swing

Swing sera utilisé dans cet ouvrage pour développer une partie de l'interface utilisateur. Toutefois, cette bibliothèque très riche, et parfois complexe, ne sera pas enseignée dans ce livre.

À LIRE **Swing**

Pour de plus amples informations, reportez-vous aux références suivantes :
- 🕮 Emmanuel Puybaret, *Swing*, Eyrolles, 2006
- 🕮 Kathy Walrath, *The JFC Swing Tutorial*, Addison-Wesley, 2004

JNDI 1.5

Ce composant, appelé communément *service de nommage*, est un service fondamental dans n'importe quel système informatique. Il permet d'associer des noms à des objets et de retrouver ces objets grâce à leurs noms. *Java Naming and Directory Interface* (JNDI) fournit les fonctionnalités de nommage et d'annuaire aux applications écrites en Java.

Omniprésent dans la version J2EE 1.4, JNDI se fait plus discret et il est intégré directement dans Java SE 5. Il continue à être une pièce maîtresse mais de manière plus transparente.

Dans le chapitre 6, nous utiliserons JNDI pour accéder aux composants distants à partir de l'interface Swing.

APPROFONDIR **JNDI**

Java Naming and Directory Interface est rarement utilisé seul. Il est généralement employé avec les EJB. Il n'y a donc que peu de livres s'attardant uniquement sur cette API.
- 🕮 Rosanna Lee, *The Jndi Api: Tutorial and Reference: Building Directory-Enabled Java Applications*, Addison-Wesley, 2000
- ▸ http://java.sun.com/products/jndi/

JDBC 3.0

JDBC (*Java Data Base Connectivity*) est une API permettant l'accès à des bases de données relationnelles à partir du langage Java. Elle se charge de trois étapes indispensables à l'accès aux données :

- la création d'une connexion à la base ;
- l'envoi d'instructions SQL ;
- l'exploitation des résultats provenant de la base.

Pour accéder à la base de données, JDBC s'appuie sur des drivers (pilotes) spécifiques à un fournisseur de SGBDR ou sur des pilotes génériques.

PERSISTANCE **Les pilotes JDBC**

Les pilotes JDBC sont classés en quatre catégories :
- Les pilotes de type 1 (pont JDBC/ODBC) permettent de convertir les appels JDBC en appel ODBC (*Open Database Connectivity*),
- Les pilotes de type 2 sont écrits en code natif et dépendent de la plate-forme.
- Les pilotes de type 3 utilisent un autre pilote JDBC intermédiaire pour accéder à la base de données.
- Les pilotes de type 4 sont écrits entièrement en Java. Ils sont donc portables.

XML et XSD

SGML (*Standard Generalized Markup Language*, ou langage normalisé de balisage généralisé), adopté comme standard en 1986, a été la première tentative pour créer des documents électroniques. L'idée principale était de séparer le contenu d'un document de sa forme. SGML a été une innovation, mais d'une complexité telle que sa manipulation s'est trouvée restreinte aux spécialistes.

XML (eX*tensible Markup Language*, ou langage extensible de balisage), issu de SGML, a été mis au point par le World Wide Web Consortium (W3C) en 1996. Contrairement à HTML, qui présente un jeu limité de balises orientées présentation (titre, paragraphe, image, lien hypertexte…), XML est un métalangage, qui va permettre d'inventer à volonté de nouvelles balises pour décrire des données et non leur représentation.

XML permet donc de définir des fichiers dont la structure est personnalisée par la création de balises. De fait, ce langage s'impose comme un standard dans les échanges inter-systèmes d'information. XML devient un format pivot, qualifié encore de format d'échanges. De plus, un certain nombre d'API offre des mécanismes pour créer, extraire et vérifier la validité d'un document XML. Cette validation n'est possible que si l'on connaît la structure du document. Cette structure est définie par un *XML Schema Definition* (XSD), technologie dérivée d'XML. Un schéma XML (XSD) est lui-même un fichier XML.

Exemple de document XML

```
<racine>
  <titre nom='exemple de message XML'/>
  <message>
    données envoyées entre émetteur et récepteur
  </message>
</racine>
```

Les balises

Une balise est une portion de texte encadré par les caractères < et >. Elle sert à délimiter des ensembles de données contenues dans un document afin de le structurer. XML est un langage de balisage.

APPROFONDIR XML et XSD

- Alexandre Brillant, *XML - Cours et exercices*, Eyrolles, 2007
- Antoine Lonjon, Jean-Jacques Thomasson, Libero Maesano, *Modélisation XML*, Eyrolles, 2006
- Mitch Amiano, Conrad D'Cruz, Michael D. Thomas , Kay Ethier, *XML*, Wrox, 2006
- http://www.w3.org/XML/
- http://www.w3.org/XML/Schema

HTML et XHTML

À partir de 1992, Internet popularise le langage HTML (*Hypertext Markup Language*, ou langage de balisage hypertexte, conçu vers 1990) pour la présentation de documents électroniques hypertextes. Issu de SGML, HTML définit un certain nombre de balises liées uniquement à la présentation. Depuis quelques années, le HTML tend à être remplacé par le XHTML qui lui apporte la rigueur de la notation XML.

APPROFONDIR HTML/XHTML

- Mathieu Nebra, *Réussir son site web avec XHTML et CSS*, Eyrolles, 2008
- Jean Engels, *XHTML et CSS*, Eyrolles, 2006
- Raphaël Goetter, *Mémento XHTML*, Eyrolles, 2006
- http://www.w3.org/MarkUp/
- http://www.w3.org/TR/xhtml1/

Exemple de page HTML

```
<html>
  <head>
      <title>Page HTML affichant Hello World</title>
  </head>
  <body>
      <center>Hello World</center>
  </body>
</html>
```

La plate-forme Java EE 5

Java EE, ou Java Enterprise Edition, est un ensemble de spécifications destinées aux applications d'entreprise. Java EE peut être vu comme une extension du langage Java afin de faciliter la création d'applications réparties, robustes, performantes et à haute disponibilité.

Comme beaucoup, je pense que Java EE est aujourd'hui la meilleure plate-forme de développement pour les entreprises. Elle combine les avantages du langage Java avec l'expérience acquise dans le développement au cours des dix dernières années. Elle bénéficie en outre du dynamisme des communautés Open Source ainsi que du JCP de Sun.

RAPPEL Java EE 5 dans cet ouvrage

La nouvelle plate-forme Java EE 5 comporte plus de vingt spécifications (voir annexe A). Il est impossible en un seul ouvrage de couvrir toute les particularités de ces spécifications. Le développement de notre étude de cas nous permettra d'utiliser les plus importantes et surtout de voir comment elles s'utilisent ou interagissent les unes par rapport aux autres. Pour approfondir vos connaissances, n'hésitez pas à consulter les nombreuses références contenues dans ce livre.

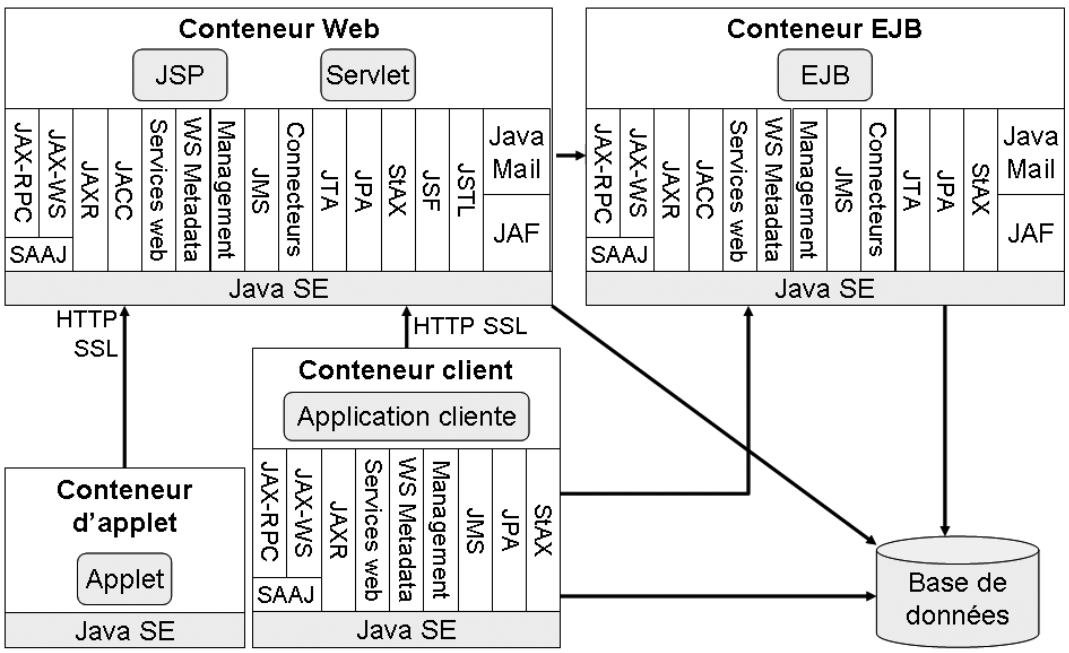


Figure 2–3 L'architecture de Java EE 5

JCP

Java Community Process est le processus utilisé par Sun et de nombreux partenaires pour coordonner l'évolution du langage Java et des technologies associées.

PRÉCISION Conteneur client

Le conteneur client, ou *Application Client Container* (ACC), apporte aux applications Java SE (par exemple, Swing) des services de sécurité, de nommage, d'injection...

PRÉCISION J2EE, Java EE, J2SE, Java SE

Pendant longtemps, Java et sa plate-forme entreprise ont été appelés respectivement J2SE et J2EE. Depuis la version 5, le chiffre « 2 » a disparu pour faciliter la compréhension de la version utilisée. Ce livre utilisera donc les nouvelles appellations Java SE et Java EE. Le terme J2EE sera utilisé pour désigner l'ancienne spécification.

JPA, Java Persistence API, est présenté au chapitre 4. Cette API est utilisée pour développer les objets métier de l'application YAPS Pet Store.

JSR

JSR, ou *Java Specification Requests*, est un système normalisé ayant pour but de faire évoluer la plate-forme Java en donnant la possibilité à la communauté de créer de nouvelles spécifications. Vous pouvez retrouver toutes les spécifications sur le site du JCP.

- http://www.jcp.org

Bien que cette plate-forme soit prédite à un bel avenir, ses promesses ne sont pas toujours honorées. Les systèmes délivrés sont souvent trop lents et compliqués, et le temps de développement est, quant à lui, fréquemment disproportionné par rapport à la complexité des demandes des utilisateurs.

Heureusement, au deuxième trimestre 2006, Java EE 5 est venu simplifier la précédente version (J2EE 1.4). S'appuyant sur la nouvelle mouture du langage Java et s'inspirant de frameworks Open Source, certains composants de la version 5 de Java EE ont été totalement réécrits dans le but de simplifier la plate-forme.

La figure 2-3 décrit les différents conteneurs spécifiés dans Java EE 5 ainsi que les spécifications qui peuvent y être employées. Les paragraphes suivants nous donnent un bref descriptif des spécifications utilisées pour le développement de l'application YAPS Pet Store. Certaines ont vu le jour avec la version 5 de Java EE, d'autres ont été complètement remaniées pour simplifier le travail des développeurs.

JPA 1.0

Depuis les débuts de J2EE, le modèle de persistance ne cesse d'évoluer et de s'engluer de version en version. Les entity beans 2.0 ont été complètement réarchitecturés pour laisser place aux entity beans 2.1. Bien que cette évolution ait apporté beaucoup d'améliorations, ce modèle de composants persistants continuait à faire des détracteurs parmi la communauté. Ce mécontentement a donné naissance à une nouvelle spécification (JDO, *Java Data Object*) ainsi qu'à différents outils de mapping objet/relationnel payants ou libres (TopLink, Hibernate...).

Java EE 5, avec son lot de nouveautés, nous apporte un nouveau modèle de persistance : JPA (*Java Persistence API*). Fortement inspirés par des outils Open Source tels qu'Hibernate ou par JDO, le mapping objet/relationnel et le langage de requête sont totalement différents de l'ancêtre entity bean 2.1. JPA, ou JSR-220, réconcilie ainsi les utilisateurs de la plate-forme Java EE avec les composants persistants.

Java Persistent API s'appuie sur JDBC (*Java Data Base Connectivity*) pour communiquer avec la base de données. En revanche, grâce à l'abstraction apportée par JPA, nous n'aurons nul besoin d'utiliser directement JDBC dans le code Java.

JMS 1.1

Une des manières d'avoir des traitements asynchrones en Java EE, consiste en l'utilisation d'un MOM (*Message-Oriented Middleware*), c'est-à-dire un système basé sur l'échange de messages. En utilisant JMS (*Java*

Message Service), un client produit un message et le publie dans une file d'attente. Ainsi, la communication des événements ou des données se fait de façon asynchrone : ni le client ni l'EJB ne dépendent de la réponse directe de l'autre et leurs traitements ne sont donc pas figés durant l'attente d'une réponse.

À LIRE JMS

- Richard Monson-Haefel, *Java Message Service*, O'Reilly, 2002
- Eric Bruno, *Java Messaging*, Charles River Media, 2005

EJB 3.0

Les *Entreprise Java Beans*, ou EJB, sont des composants serveurs qui respectent les spécifications d'un modèle édité par Sun. Ces spécifications définissent une architecture, un environnement d'exécution (un conteneur) et un ensemble d'API. Le respect de ces spécifications permet d'utiliser les EJB indépendamment du conteneur dans lequel ils s'exécutent. Ce dernier fournit un ensemble de fonctionnalités comme la gestion du cycle de vie de l'EJB, la sécurité, l'accès concurrent et les transactions. Le but des EJB est de faciliter la création d'applications distribuées pour les entreprises.

Serveur d'applications

Un serveur d'applications héberge les applications destinées à être utilisées dans un réseau distribué. Il est doté de services transactionnels entre composants, d'équilibrage de charge ou encore de tolérance aux pannes.

Pièce maîtresse de l'architecture Java EE, les EJB 3 apportent des modifications notables dans le mode de développement et intègrent de nombreuses nouveautés, notamment en ce qui concerne la persistance. Le passage des EJB 2.1 en 3.0 apporte une simplification radicale en supprimant les descripteurs de déploiement, les appels JNDI, etc., et laisse place aux annotations.

À LIRE EJB

- Laboratoire Supinfo, *EJB 3 : des concepts à l'écriture du code*, Dunod, 2006
- Rima Patel Sriganesh, Gerald Brose, Micah Silverman *Mastering Enterprise JavaBeans*, Wiley, 2006

Il existe deux grandes familles d'EJB : entité et session. Les EJB sessions sont différenciés entre EJB sans état, avec état ou s'exécutant en mode asynchrone.

EJB stateless

Les EJB sans état, ou *stateless session beans*, ne fonctionnent que de façon éphémère. Une fois qu'un client a demandé et reçu une fonctionnalité du composant, l'interaction avec ce composant prend fin, ne laissant aucune trace de ce qui s'est passé. Ils n'ont aucune connaissance du client ou d'un semblant de contexte concernant l'enchaînement des requêtes : ils sont donc parfaits pour une utilisation unique. Ils n'ont pas d'état, c'est-à-dire qu'on ne peut pas manipuler leurs attributs en étant sûr de leur valeur.

Le chapitre 5 présente les stateless session beans. Nous les utiliserons pour développer les composants métiers.

L'utilisation standard d'un EJB sans état réside dans le fait qu'une application cliente le contacte en lui transmettant des paramètres. L'EJB accède alors à une base de données, effectue plusieurs traitements, appelle d'autres EJB, puis retransmet un résultat au client. Lorsque la communication s'achève, le bean ne conserve aucun souvenir de l'interaction. Avec ce comportement, plusieurs clients distincts peuvent accéder simultanément à un même stateless session bean.

Exemple d'EJB stateless

```
@Stateless
public class MonBean {
  // Le code métier
  public void maMethode() {
    return;
  }
}
```

EJB stateful

EJB Avec ou sans état ?

Un EJB stateless est utile pour calculer cos(x), convertir des euros en dollars, supprimer tous les bons de commandes passés il y a cinq ans ou obtenir le cours d'une action.
Un EJB stateful sert à stocker des articles achetés en ligne dans un panier électronique ou à commander plusieurs billets de train.

Par opposition au composant sans état, les *stateful session beans* associent les requêtes à un client spécifique, unissant client et EJB dans une relation un-un. Ce type de composant fournit ainsi un ensemble de traitements via des méthodes, mais il a aussi la possibilité de conserver des données entre les différents appels de méthodes d'un même client. Une instance particulière est donc dédiée à chaque client, qui sollicite ses services et ce, tout au long du dialogue.

Les données conservées par le bean sont stockées dans les variables d'instances. Les données sont donc conservées en mémoire. Généralement, les méthodes proposées par le composant permettent de consulter et de mettre à jour ces données.

Le chapitre 8 présente les stateful session beans qui seront utilisés pour développer le panier électronique de l'application YAPS Pet Store.

Exemple d'EJB stateful

```
@Stateful
public class MonBean {
  // Attribut conservant sa valeur
  private String monAttribut;
  // Le code métier
  public void maMethode() {
    return;
  }
}
```

Message-driven bean

Les précédents types d'Enterprise Java Beans offrent des services de manière synchrone. Le client émet une requête, puis attend que l'EJB lui envoie un résultat.

L'API JMS et les message-driven beans sont décrits au chapitre 10. Les traitements asynchrones de l'application utilisent ces deux technologies.

Pour les *message-driven beans* (MDB), le comportement est complètement différent. Les clients n'appellent pas directement des méthodes mais utilisent JMS pour produire un message et le publier dans une file d'attente. À l'autre bout, le MDB est à l'écoute de cette file d'attente et se « réveille » à l'arrivée d'un message. Il extrait ce dernier de la file d'attente, en récupère le contenu puis exécute un traitement. Le client

n'a donc pas besoin de figer son exécution durant le traitement du MDB. Le traitement est asynchrone.

Exemple de MDB

```
@MessageDriven
public class MonMDB implements MessageListener {

  public void onMessage (Message msg) {
    // Traiter le message
  }
}
```

Entity

Les stateful session beans sont détruits lorsque la session du client se termine. Ils ne peuvent donc pas être utilisés pour stocker de façon permanente les informations de l'application. Les EJB entités peuvent répondre à ce besoin puisqu'ils sont persistants. En effet, leur état est sauvegardé sur un support de stockage externe, comme une base de données. Les entities représentent des données, ou plus exactement des objets métier, qui perdurent après la fin d'une session. Ils existent souvent sous la forme d'enregistrements uniques dans une base de données.

Depuis Java EE 5 et l'arrivée de JPA, on a plutôt tendance à parler d'entité (*entity*) que de bean entité (*entity bean*). En effet, si les entity beans 2.1 ont un modèle de développement assez lourd et compliqué, les entities sont, eux, de simples classes Java (Pojo) et peuvent même être utilisés en dehors de Java Enterprise Edition, c'est-à-dire dans une application Java standard. Il faut juste se faire à l'idée qu'un entity bean est devenu une simple classe Java (*lightweight*) et non un composant lourd (*heavyweight*) et complexe à développer.

Pojo

Pojo est un acronyme qui signifie *Plain Old Java Object*. Ce terme est principalement utilisé pour faire référence à la simplicité d'utilisation d'un objet Java en comparaison avec la lourdeur d'utilisation d'un composant EJB.

Exemple d'entité d'adresse

```
@Entity
@Table(name = "t_adresse")
public class Adresse {

  @Id @GeneratedValue
  private Long identifiant;
  private String rue;
}
```

ANNOTATIONS Java EE

Vous avez peut-être remarqué, dans les extraits de code précédents, l'utilisation des annotations Java dans le monde Java EE : `@Entity`, `@MessageDriven`, `@Stateless`, `@Stateful`. Comme nous le verrons dans les chapitres suivants, il en existe bien plus encore.

Le conteneur d'EJB

Souvent aussi appelé à tort s*erveur d'applications*, le conteneur d'EJB a la responsabilité de créer de nouvelles instances d'EJB et de gérer leur cycle de vie. Il est l'intermédiaire entre l'EJB et le serveur d'applications. Il

fournit des services tels que le transactionnel, la sécurité, la concurrence, la distribution, le service de nommage des EJB (JNDI) et l'exécution.

Les EJB interagissent avec le conteneur de plusieurs manières et peuvent exécuter des traitements déclenchés automatiquement par ce dernier. De même que si un EJB lève une exception, celle-ci est tout d'abord interceptée par le conteneur qui décidera d'effectuer telle ou telle action.

Servlet 2.5 et JSP 2.1

HTTP

HTTP, ou *Hypertext Transfer Protocol*, est un protocole de transfert de pages HTML sur le Web. Sa fonction première est d'établir la connexion avec un serveur, qui contient la page que l'on veut voir afficher, et de rapatrier cette page sur le poste de l'internaute.

Les servlets sont des programmes Java fonctionnant côté serveur au même titre que les CGI et les langages de script tels que ASP ou PHP. Les servlets permettent donc de recevoir des requêtes HTTP, de les traiter et de fournir une réponse dynamique au client. Elles s'exécutent dans un conteneur utilisé pour établir le lien entre la servlet et le serveur web. Les servlets étant des programmes Java, elles peuvent utiliser toutes les API Java afin de communiquer avec des applications externes, se connecter à des bases de données, accéder aux entrées-sorties (fichiers, par exemple)…

À LIRE Servlet et JSP

- Jean-Luc Déléage, *JSP et Servlets efficaces*, Dunod, 2005
- François-Xavier Sennesal, *JSP avec Eclipse et Tomcat*, ENI, 2007
- Anne Tasso, Sébastien Ermacore, *Initiation à JSP*, Eyrolles, 2004

Java Server Page, ou JSP, est une technologie basée sur Java qui permet aux développeurs de générer dynamiquement du code HTML, XML ou tout autre type de page web. Une page JSP (repérable par l'extension `.jsp`) aura un contenu pouvant être différent selon certains paramètres (des informations stockées dans une base de données, les préférences de l'utilisateur...) tandis qu'une page web « statique » (dont l'extension est `.htm` ou `.html`) affichera continuellement la même information.

Exemple de page JSP affichant la date du jour

```
<%@ page import="java.util.Date"%>
<html>
  <head>
    <title>JSP Affichant la date</title>
  </head>
  <body>
    <%! Date today = new Date();%>
    <br/>
    <center>La date est <%= today %></center>
  </body>
</html>
```

Une JSP est un autre moyen d'écrire une servlet. Lorsqu'un utilisateur appelle une page JSP, le serveur web crée un code source Java à partir du script JSP (c'est-à-dire qu'il constitue une servlet à partir du script JSP), le compile, puis l'exécute.

Langage d'expression

Le langage d'expression, ou *Expression langage* (EL), permet aux JSP d'accéder aux objets Java, de manipuler des collections ou d'exécuter des actions JSF. Une expression est de la forme suivante :

```
${expression}
```

Exemple de page JSP utilisant le langage d'expression

```
<html>
  <body>
    <c:if test="${bean.attr < 3}" >
      <center>La date est ${bean.today}</center>
    </c:if>
  </body>
</html>
```

JSTL 1.2

JSTL est le sigle de *JSP Standard Tag Library*. C'est un ensemble de balises personnalisées (*Custom Tag*), développées sous la JSR 052 facilitant la séparation des rôles entre le développeur Java et le concepteur de pages web. L'avantage de ces balises est de déporter le code Java contenu dans la JSP dans des classes dédiées. Ensuite, il suffit de les utiliser dans le code source de la JSP en utilisant des balises particulières, tout comme vous le feriez avec des balises HTML classiques.

Les bibliothèques de balises (*Taglibs*) ou balises personnalisés (*Custom Tag*) permettent de définir ses propres balises basées sur XML, de les regrouper dans une bibliothèque et de les réutiliser dans des JSP. C'est une extension de la technologie JSP apparue à partir de la version 1.1 des spécifications.

Exemple d'utilisation d'une balise choose dans une page JSP

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <c:choose>
    <c:when test="${empty sessionScope.cart}">
      Le panier est vide
    </c:when>
    <c:otherwise>
      Le panier contient des articles
    </c:otherwise>
  </c:choose>
</html>
```

À LIRE **JSF et Struts**

- Bill Dudney, Jonathan Lehr, Bill Willis, LeRoy Mattingly, *Mastering JavaServer Faces*, 2004, Wiley
- Jean-Charles Felicité, *Développement Java sous Struts*, ENI, 2006
- Jean-Charles Félicité, *Struts 1.2 : les bases pour un développement Open Source Java*, ENI, 2006
- Vic Cekvenich, Wolfgang Gehner, *Struts : les bonnes pratiques pour des développements web réussis*, Dunod, 2005
- http://struts.apache.org/

JSF 1.2

Entre les servlets et les JSP, il manquait un framework pour aiguiller, de manière simple, un événement utilisateur vers une action serveur. Des outils libres comme Struts sont venus aider le développeur en décorrélant la couche présentation de la couche métier. Mais aucune spécification n'existait jusqu'à l'apparition de JSF. *Java Server Faces* est venu combler ce vide en facilitant la conception d'interfaces graphiques web, en gérant automatiquement l'état HTTP ainsi que les événements entre client et serveur.

JSF établit une norme dont le rôle est de fournir aux développeurs une large palette d'outils leur permettant d'implémenter des applications web en respectant un modèle bien précis.

Le chapitre 7 se concentre sur le développement web de l'application YAPS Pet Store. Chaque API y est expliquée ainsi que la manière de les assembler pour obtenir un site web.

Le conteneur de servlet

Le cycle de vie d'une servlet, donc d'une JSP, est assuré par le moteur de servlet (aussi appelé *conteneur de servlet* ou *conteneur web*). Celui-ci est responsable de fournir la requête HTTP à la servlet, de l'exécuter et de renvoyer la réponse. C'est le « moteur » de toute application web simple, c'est-à-dire ne mettant pas en jeu d'EJB.

JavaMail 1.4

JAVAMAIL **Les protocoles de messagerie**

Le protocole SMTP (*Simple Mail Transfer Protocol*) est le protocole standard de messagerie. Le protocole POP (*Post Office Protocol*) permet d'aller récupérer son courrier sur un serveur distant et Imap (*Internet Message Access Protocol*) est une alternative à POP offrant plus de possibilités (comme la gestion d'accès simultanés, de plusieurs boîtes aux lettres...).

JavaMail est l'API standard de gestion de courriers électroniques de Java EE. Elle permet d'envoyer et de recevoir du courrier électronique et de manipuler les messages (en-tête, sujet, corps, pièces jointes...). JavaMail n'est pas un serveur de courrier en tant que tel, mais plutôt un outil pour interagir avec ce type de serveur. Il peut être vu comme un type de client de messagerie au même titre que Outlook, Lotus, Eudora, etc. Pour envoyer ou recevoir des messages, JavaMail utilise différents protocoles comme SMTP, Imap ou POP.

JAXB 2.0

XSD

XSD, ou *XML Schema Description*, est un langage de description de format de document XML permettant de définir la structure d'un document XML. XSD est communément appelé *grammaire*.

JAXB est l'acronyme de *Java Architecture for XML Binding*. Cette API permet de générer des classes Java à partir de schémas XML (XSD) et inversement. Autrement dit, il permet de convertir les fichiers XSD en classes Java. Il est ensuite possible de manipuler le document XML au travers de ces classes.

Une fois de plus, les annotations de Java 5 sont venues simplifier l'utilisation de l'API JAXB. En annotant un Pojo (*Plain Old Java Object*), on peut ensuite obtenir ses attributs au format XML.

Exemple d'annotations JAXB

```
@XmlRootElement
public class Adresse {

  @XmlID
  private Long identifiant;
  private String rue;
}
```

Grâce aux annotations situées dans la classe, JAXB pourra générer automatiquement la structure XML suivante :

XML généré par ces annotations

```
<adresse>
  <identifiant> </identifiant>
  <rue> </rue>
</adresse>
```

Services web

Comment faire dialoguer des logiciels écrits dans des langages de programmation différents et fonctionnant sur des systèmes d'exploitation divers et variés ? La réponse est simple : en utilisant des services web. Les services web permettent cette interopérabilité en s'appuyant sur un ensemble de protocoles répandus comme HTTP. Cette communication est basée sur le principe de demandes et réponses, effectuées via des messages XML.

Les services web sont décrits par des documents WSDL (*Web Service Description Language*), qui précisent les méthodes pouvant être invoquées, leurs signatures et les points d'accès de service (URL, port). Les services web sont accessibles via Soap, la requête et les réponses sont des messages XML transportés sur HTTP.

Exemple de service web

```
@WebService
public class MonWebService {
  // Le code métier
  public void maMethode() {
    return;
  }
}
```

À LIRE Services web

- Annick Fron, *Architectures réparties en Java : RMI, CORBA, JMS, sockets, SOAP, services web*, Dunod, 2007
- Hubert Kadima, Valérie Monfort, *Les Web Services - Techniques, démarches et outils*, Dunod, 2003
- Libero Maesano, Christian Bernard, Xavier Le Galles, *Services Web avec J2EE et .NET*, Eyrolles, 2003
- Steve Graham, Doug Davis, Simeon Simeonov, Glen Daniels, *Building Web Services with Java: Making Sense of XML, Soap, WSDL, and UDDI*, Sams, 2004

Le chapitre 9 présente les services web ainsi que les technologies qui y sont rattachées. Les services web sont utilisés par l'application YAPS Pet Store pour communiquer avec les partenaires externes.

Soap

Simple Object Access Protocol est un protocole standard destiné aux services web. Lancé par IBM et Microsoft, il permet d'utiliser des applications invoquées à distance par Internet.

Blueprints

Parallèlement à la plate-forme Java EE, Sun propose gratuitement des documents pour faciliter les développements Java : les *Blueprints*. Ces

derniers contiennent des tutoriaux, des design patterns, des exemples de code, des conseils et des FAQs.

Il existe plusieurs types de Blueprints. Sous le succès des services web, Sun a développé en 2004 un Blueprint baptisé Adventure Builder. Cette application vous permet de personnaliser un séjour pour vos vacances, en utilisant principalement des services web.

Concernant Java EE (ou J2EE à l'époque), Sun créa le Java Pet Store.

Java Pet Store

Java Pet Store est une application Java EE que Sun a créé pour son programme de Blueprints. C'est un site web marchand où l'on peut choisir des animaux domestiques, les ajouter dans un panier, puis payer électroniquement. Ce Blueprint a permis de documenter les meilleures pratiques (code Java, design pattern, architecture) pour développer une application Java EE.

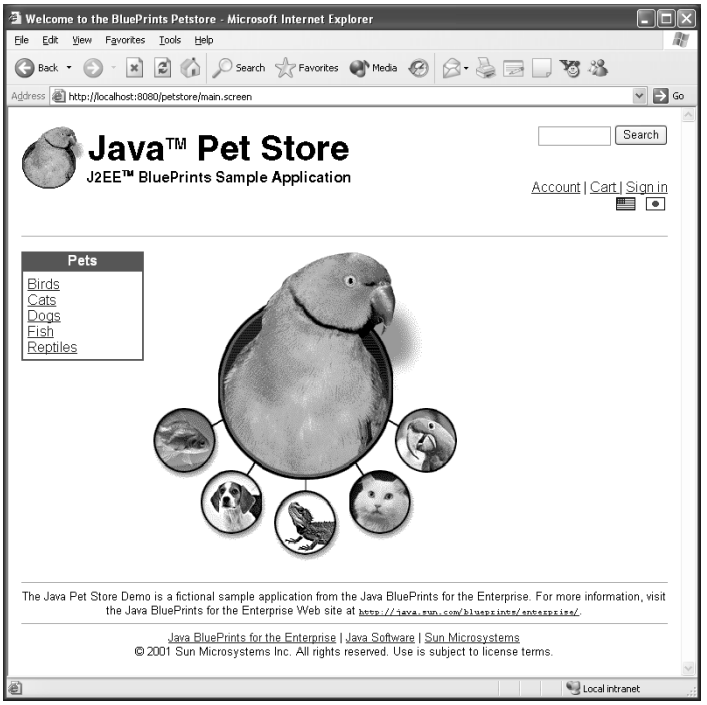


Figure 2–4
Page d'accueil du Java Pet Store de Sun

Le Java Pet Store est aussitôt devenu un standard de facto, puisque les constructeurs de serveur d'applications l'ont utilisé pour démontrer la compatibilité de leur produit avec les spécifications Java EE. En effet, Oracle fut le premier à l'utiliser pour ses tests de montée en charge. Bien que Java Pet Store ait été développé à des fins éducatives, Oracle déclara que cette application fonctionnait deux fois plus rapidement sur son ser-

ARCHITECTURE Pet Store

Les Blueprints de Sun se trouvent à l'adresse suivante :

- http://java.sun.com/reference/blueprints/

En ce qui concerne le Pet Store, vous pouvez consulter les adresses suivantes :

- http://blueprints.dev.java.net/petstore/
- http://java.sun.com/developer/releases/petstore/

REMARQUE Les autres Pet Store

Le Pet Store de Sun a été source d'inspiration pour d'autres technologies ou frameworks. Ci-dessous une liste non exhaustive de ces Pet Store :

- PetShop : utilisation du framework .NET de Microsoft ;
- xPetStore : utilisation des tags xDoclet ;
- Flash PetStore : version de Macromedia utilisant la technologie Flash ;
- Spring PetStore : utilisation du framework Spring ;
- openMDX PetStore : plate-forme Open Source MDA.

veur d'applications que sur ceux de BEA ou IBM. La communauté s'enflamma et tous les vendeurs commencèrent à utiliser le Java Pet Store pour démontrer leurs meilleures performances.

Cette anecdote contribua à augmenter la popularité de ce Blueprint, qui rentra très vite dans le langage commun. Tout le monde commença à l'utiliser pour illustrer une nouvelle technologie, une nouvelle idée ou implémentation.

L'étude de cas de cet ouvrage s'inspire de ce site de commerce électronique.

Les design patterns

Dans son livre *A Pattern Language* édité en 1977, l'architecte en bâtiment Christopher Alexander introduit le terme de *pattern* (patron) : « chaque patron décrit un problème qui se produit de manière récurrente dans notre environnement ». Si ce livre est dédié à une autre profession que celle de l'architecture informatique, il faudra attendre le livre du Gang of Four (GoF) en 1994 pour adapter ces idées au monde de l'orienté objet.

Il ne faut pas confondre ces patrons avec des briques logicielles (un pattern dépend de son environnement), des règles (un pattern ne s'applique pas mécaniquement) ou des méthodes (ne guide pas la prise de décision). Mais plutôt les voir comme une solution de conception à un problème récurrent.

Viendront alors, bien plus tard, deux livres s'inspirant du GoF mais dédié à la plate-forme J2EE : *EJB Design Pattern* et *Core J2EE Patterns*. Ces trois ouvrages ont créé un vocabulaire commun entre les développeurs, concepteurs et architectes.

Ce livre utilisera plusieurs design patterns pour concevoir l'application YAPS Pet Store.

UML 2

UML (*Unified Modeling Language* ou langage de modélisation unifié), est né de la fusion des trois méthodes qui ont le plus influencé la modélisation objet au milieu des années 1990 : OMT, Booch et OOSE. Issu d'un travail d'experts reconnus (James Rumbaugh, Grady Booch et Ivar Jacobson), UML est le résultat d'un large consensus qui est vite devenu un standard incontournable. Fin 1997, ce langage est devenu une norme OMG (*Object Management Group*).

UML est un langage de modélisation objet et non une démarche d'analyse. Il représente des concepts abstraits de manière graphique. UML est donc un langage universel et visuel qui permet d'exprimer et d'élaborer des modèles

GoF

Le *Gang of Four* désigne les quatre auteurs du livre *Design Pattern*, c'est-à-dire Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides

Les anti-patterns

Les anti-patterns sont des erreurs courantes de conception de logiciels. Leur nom vient du fait que ces erreurs sont apparues dès les phases de conception du logiciel, notamment par l'absence ou la mauvaise utilisation de design pattern.

À LIRE Design pattern

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, *Design Pattern*, Addison-Wesley, 1995
- Floyd Marinescu, *EJB Design Patterns*, Wiley, 2002
- Deepak Alur, Dan Malks, John Crupi, *Core J2EE Patterns*, Prentice Hall, 2003
- Laurent Debrauwer, *Design Patterns – Les 23 modèles de conception : descriptions et solutions illustrées en UML 2 et Java*, ENI, 2007
- Steven-John Metsker, William C. Wake, *Les Design Patterns en Java*, Campus Press, 2006

OMG

L'objectif de l'*Object Management Group* est de standardiser et de promouvoir le modèle objet sous toutes ses formes. L'OMG est notamment à la base des spécifications UML, MOF, CORBA, IDL et MDA.

objet, indépendamment de tout langage de programmation. Comme UML n'impose pas de méthodes de travail particulières, il peut être intégré à n'importe quel processus de développement logiciel de manière transparente.

UML 2 introduit quatre nouveaux diagrammes (paquetages, structures composites, global d'interaction et de temps) qui viennent enrichir les neufs initiaux (classes, objets, composants, déploiement, cas d'utilisation, états-transitions, activités, séquence et communication). La plupart de ces diagrammes seront utilisés tout au long des chapitres.

Architecture de l'application

L'application YAPS Pet Store va donc utiliser toutes les technologies énoncées ci-dessus. Comme nous l'avons vu au chapitre précédent, *Présentation de l'étude de cas*, on accède à l'application par des navigateurs (client léger utilisé par les internautes) et par les clients riches (développés en Swing) déployés sur les postes des employés. Ces interfaces graphiques accèdent elles-mêmes à un serveur qui va effectuer les traitements métier puis stocker les données dans une base.

Pour ce faire, l'application YAPS Pet Store est découpée en plusieurs couches.

L'architecture en trois couches

On peut donc dire que l'architecture logique de l'application YAPS Pet Store est découpée en trois couches (ou trois niveaux). L'architecture en trois couches est le modèle le plus général des architectures multicouches. Ces couches sont :

- **présentation des données** : affichage sur le poste de travail des données du système et interaction avec l'utilisateur ;
- **traitements métier** : ensemble des règles métiers de l'application ;
- **accès aux données** : manipulation et conservation des données.

Ci-après un diagramme de paquetages UML représentant ces trois couches. Chaque sous-système contient les technologies Java EE 5 utilisées dans l'application.

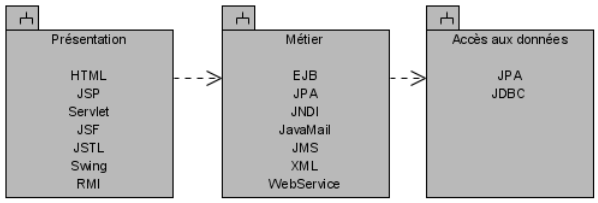


Figure 2–6 Architecture Java EE en trois couches

APPROFONDIR UML

- Pascal Roques, *UML 2 – Modéliser une application web*, Eyrolles, 2007
- Pascal Roques, Franck Vallée, *UML 2 en action : de l'analyse des besoins à la conception*, Eyrolles, 2007
- Xavier Blanc, Isabelle Mounier, UML *2 pour les développeurs – Cours avec exercices corrigés*, Eyrolles, 2006
- Jim Arlow, Ila Neustadt, *UML2 and the Unified Process*, Addison-Wesley, 2005
- http://www.uml.org/

Architecture

L'architecture spécifie la structure d'un système. On parle d'architecture fonctionnelle pour définir les services du système, d'architecture technique pour les composants techniques utilisés et d'architecture applicative pour décrire le découpage en sous-systèmes.

ARCHITECTURE Couches ou tiers

Lorsqu'on parle d'architecture en couches, on utilise souvent le terme anglais *tiers*. Ce terme signifie couche et non le tiers mathématique (1/3). On entend par conséquent les architectes parler d'architecture quatre tiers ou cinq tiers. Il ne faut pas comprendre par là que l'architecture est en 4/3 ou 5/3 mais bien qu'elle est découpée en 4 ou 5 couches.

UML Paquetages et sous-systèmes

Un paquetage (*package* en anglais) est un mécanisme destiné à regrouper des éléments comme des classes, des cas d'utilisation, voire d'autres paquetages. Le terme sous-système (*subsystem*) indique que le paquetage représente une partie indépendante du système. Ci-après la représentation graphique UML d'un paquetage et d'un sous-système.

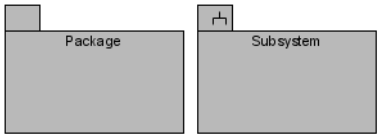


Figure 2–5 Paquetage et sous-système

Architecture applicative

Le précédent modèle en trois couches peut être affiné pour être plus fidèle à l'architecture finale de l'application YAPS Pet Store. Ci-après un diagramme de paquetage décrivant les couches de l'application :

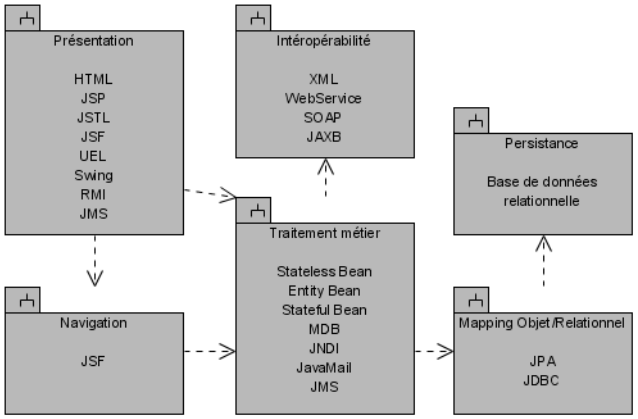


Figure 2–7
Couches de l'application YAPS Pet Store

Couche de présentation

La couche de présentation est la partie visible de l'application qui permet à un utilisateur d'interagir avec le système. Elle relaie les requêtes de l'utilisateur à destination de la couche métier, et en retour lui présente les résultats renvoyés par les traitements. On parle alors d'interface homme-machine (IHM) et aucun traitement n'est implémenté dans cette couche. L'application YAPS Pet Store possède deux types d'interfaces homme-machine : un client léger et un client riche.

Le client léger, utilisé par l'internaute, est décrit en langage HTML puis interprété par le navigateur. Dans le cas du YAPS Pet Store, nous ne développerons pas directement l'interface en HTML, mais plutôt à l'aide de JSP et de tags JSF et JSTL ainsi que du langage d'expression. Les appels à la couche métier sont délégués à la couche navigation.

Le client riche, lui, est développé en Swing et utilise le protocole RMI (*Remote Method Invocation*) pour communiquer avec la couche métier. Pour afficher les événements asynchrones reçus de l'application, cette couche utilise également JMS.

JMS L'application Swing

L'expression des besoins nous signale que les employés veulent être avertis lorsqu'un bon de commande contient des reptiles. Pour ce faire, l'application écoutera sur une file d'attente JMS et sera alertée à chaque fois qu'un reptile est vendu.

Couche de navigation

Cette couche, uniquement utilisée par le client léger, prend en charge la logique de navigation. De ce fait, elle gère l'enchaînement des JSP ainsi que les appels aux traitements métier. Cette couche est mise en œuvre par la technologie JSF.

Couche de traitement métier

La couche de traitement métier correspond à la partie fonctionnelle ou métier de l'application. Elle implémente la logique et les règles de gestion permettant de répondre aux requêtes de la couche présentation.

Pour fournir ces services, elle s'appuie, le cas échéant, sur les données du système, accessibles au travers des services de la couche inférieure, c'est-à-dire la couche de données. En retour, elle renvoie à la couche présentation les résultats qu'elle a calculés.

En pratique, on trouve au niveau de la couche métier :

- des **entities** dont la persistance est assurée par la couche de mapping ;
- des **stateless beans** qui proposent des méthodes pour manipuler les entities (CRUD) ;
- des **message-driven beans** qui assurent les traitements asynchrones ;
- les **API JNDI**, pour accéder au service de nommage, et **JavaMail**, pour envoyer des e-mails aux clients.

Les appels vers les systèmes externes (BarkBank et PetEx) sont orchestrés par la couche métier mais délégués à la couche d'intéropérabilité.

CRUD

CRUD est un terme communément utilisé pour l'accès aux bases de données. Il signifie en anglais *Create, Retrieve, Update and Delete*, c'est-à-dire création, lecture, mise à jour et suppression de données.

Couche de mapping objet/relationnel

La couche de mapping objet/relationnel transforme la représentation physique des données en une représentation objet et inversement. Ce mécanisme est assuré par JPA qui utilise le protocole JDBC pour exécuter les appels SQL. Cette couche n'est utilisée que parce que notre base de données est relationnelle. En effet, si la persistance était faite de manière native en objet, ce mapping n'aurait pas lieu d'être.

Les bases de données objets

Les bases de données objets, comme leur nom l'indique, organisent les données sous forme d'objets et non sous forme de tables (lignes et colonnes). Bien que certains grands acteurs du monde relationnel aient des implémentations objets, les bases objets n'ont jamais vraiment percé sur le marché.

Couche de persistance

Cette couche contient les données sauvegardées physiquement sur disque, c'est-à-dire la base de données. En Java, le protocole d'accès aux données est JDBC.

Couche d'interopérabilité

Pour dialoguer avec ses partenaires BarkBank et PetEx, l'application utilise une couche d'interopérabilité. Basée sur les technologies JAX-WS, elle accède à des services web distants via le protocole HTTP.

JAX-WS

JAX-WS est la nouvelle appellation de JAX-RPC (*Java API for XML Based RPC*) qui permet de développer très simplement des services web.

Architecture technique

Les couches que nous venons de décrire sont avant tout logiques et servent à décrire la conception de l'application. Nous allons maintenant

projeter cette architecture sur un ou plusieurs emplacements physiques. Le diagramme de déploiement ci-après nous montre les machines physiques utilisées. Pour l'application à proprement parler, il n'y a dans notre cas qu'un seul serveur physique. Mais nous aurions pu subdiviser. Les autres machines correspondent aux postes des internautes, des employés et des partenaires externes.

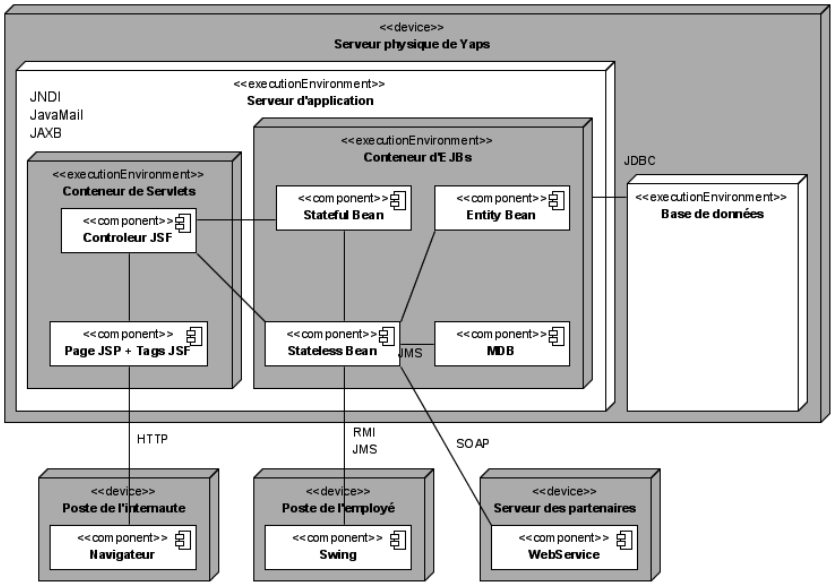


Figure 2–8
Diagramme de déploiement de l'application

Le serveur physique comporte une base de données et un serveur d'applications. Ce dernier est composé d'un conteneur de servlets et d'un conteneur d'EJB. À l'intérieur de chaque nœud (représenté sous forme de cube), on peut voir les composants qui sont déployés. Pour la partie web, on retrouve les JSP, JSF et JSTL, alors que le conteneur d'EJB héberge les stateless, stateful, MDB et entities. Les services web sont déployés sur les serveurs des partenaires.

UML Le diagramme de déploiement

Le diagramme de déploiement montre la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels. Les ressources matérielles sont représentées sous forme de nœuds (les cubes) qui peuvent être liés entre eux à l'aide d'un support de communication. Le stéréotype `<<device>>` renforce le fait que le nœud est un serveur physique, alors que `<<executionEnvironment>>` est utilisé pour les logiciels tels qu'un serveur d'applications ou de base de données.

PRÉCISION Machine physique

La répartition des composants sur différentes machines est une vision idéale de l'architecture. Pour simplifier le déploiement de la totalité de l'application, nous n'utiliserons qu'une seule machine qui hébergera le serveur d'applications, les services web ainsi que les interfaces homme-machine.

En résumé

Ce chapitre nous a présenté les différents langages ainsi que la plateforme Java EE 5 avec lesquels sera développée l'application YAPS Pet Store. L'architecture en couches a été détaillée à l'aide de diagrammes de paquetages et de déploiement. Nous avons désormais défini et recensé les technologies et spécifications qui seront utilisées dans les différentes couches applicatives.