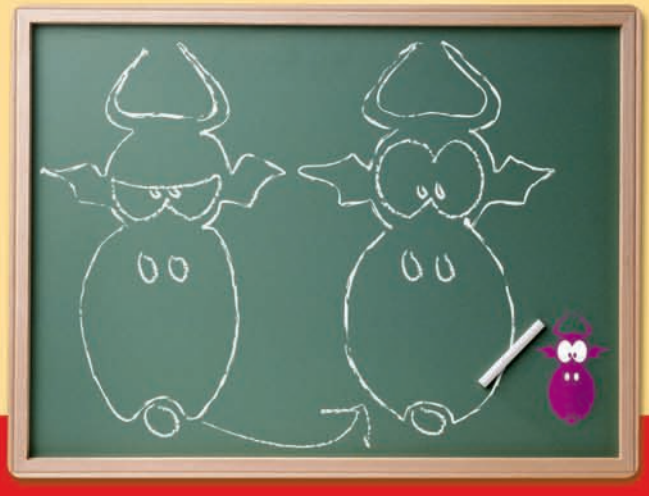
les Cahiers du Programmeur

UML 2

Modéliser une application web

Pascal **Roques**



4e édition

EYROLLES

les Cahiers du Programmeur

UML 2

Du même auteur

P. Roques. – **UML 2 par la pratique**. N°12322, 6e édition, 2008, 368 p.
P. Roques. – **Mémento UML.** N°11725, 2006, 14 pages.
P. Roques, F. Vallée. – **UML 2 en action**. *De l'analyse des besoins à la conception.* N°12104, 4e édition, 2007, 382 p.

Collection « Les cahiers du programmeur »

A. Goncalves. – **Java EE 5.** N°12363, 2e édition 2008, 370 pages
E. Puybaret. – **Swing.** N°12019, 2007, 500 pages
E. Puybaret. – **Java 1.4 et 5.0.** N°11916, 3e édition 2006, 400 pages
J. Molière. – **J2EE**. N°11574, 2e édition 2005.
R. Fleury – **Java/XML.** N°11316, 2004.
J. Protzenko, B. Picaud. – **XUL**. N°11675, 2005, 320 pages
S. Mariel. – **PHP 5**. N°11234, 2004, 290 pages.

Chez le même éditeur

V. Messager-Rota. – **Gestion de projet. Vers les méthodes agiles**. N°12165, 2007, 252 p.
H. Bersini, I. Wellesz. – **L'orienté objet**. N°12084, 3e édition, 2007, 600 p.
S. Bordage. – **Conduite de projet Web.** N°12325, 5e édition, 2008, 394 p.
O. Andrieu. – **Réussir son référencement Web.** N°12264, 2008, 302 p.
G. Ponçon. – **Best practices PHP 5. Les meilleures pratiques de développement en PHP**. N°11676, 2005, 480 p.
A. Patricio. – **Java Persistence et Hibernate**. N°12259, 2008, 364 p.
K. Djaafar. – **Développement JEE 5 avec Eclipse Europa**. N°12061, 2008, 380 p.
J.-M. Defrance. **– Premières applications Web 2.0 avec Ajax et PHP.** N°12090, 2008, 450 p.
J. Dubois, J.-P. Retaillé, T. Templier. **– Spring par la pratique**. *Java/J2EE, Spring, Hibernate, Struts, Ajax*. – N°11710, 2006, 518 p.
T. Ziadé. – **Programmation Python**. – N°11677, 2006, 530 p.

Collection « Accès libre »

Pour que l'informatique soit un outil, pas un ennemi !

Open ERP. *Pour une gestion d'entreprise efficace et intégrée*.
F. Pinckaers, G. Gardiner.
N°12261, 2008, 276 p.

Réussir son site web avec XHTML et CSS.
M. Nebra.
N°12307, 2e édition, 2008, 316 pages.

Ergonomie web. *Pour des sites web efficaces*.
A. Boucher.
N°12158, 2007, 426 p.

Gimp 2 efficace. *Dessin et retouche photo*.
C. Gémy.
N°12152, 2e édition, 2008, 402 p.

La 3D libre avec Blender.
O. Saraja.
N°12385, 3e édition, 2008, 400 pages avec CD et cahier couleur (À paraître).

Scenari – La chaîne éditoriale libre.
S. Crozat.
N°12150, 2007, 200 p.

Créer son site e-commerce avec osCommerce.
D. Mercer, adapté par S. Burriel.
N°11932, 2007, 460 p.

Réussir un site web d'association… avec des outils libres.
A.-L. et D. Quatravaux.
N°12000, 2e édition, 2007, 372 p.

Ubuntu efficace.
L. Dricot *et al*.
N°12003, 2e édition, 2007, 360 p. avec CD-Rom.

Réussir un projet de site Web.
N. Chu.
N°11974, 4e édition, 2006, 230 pages

Premiers pas en CSS et HTML.
F. Draillard
N°12390, 2e édition 2008, 250 p.

Gimp 2.4.
D. Robert.
N°12295, 3e édition, 2008, 316 p.

Firefox. *Un navigateur web sûr et rapide.*
T. Trubacz, préface de T. Nitot.
N°11604, 2005, 250 p.

SPIP 1.9. *Créer son site avec des outils libres.* Perline, A.-L. Quatravaux *et al.*. –
N°12002, 2e édition 2007, 376 pages.

Mozilla Thunderbird. *Le mail sûr et sans spam.*
D. Garance, A.-L. et D. Quatravaux.
N°11609, 2005, 320 p. avec CD-Rom.

Pascal **Roques**

les Cahiers du Programmeur

UML 2

Modéliser une application web

4e édition

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

À Margaux, Loriane, Maxime et Noémie,

qui m'aident tous les jours à donner un sens à ma vie …

À Sylvie, qui me donne l'énergie d'avancer dans la bonne direction…

Préface

Le développement de sites web est souvent le royaume où règne la loi du « vite fait, mal fait ». Il est vrai que tous les ingrédients sont là (langages simples, outils intuitifs) pour aider à la production de pages tant statiques que dynamiques. Cela autorise la création de sites pour des particuliers et de petites entreprises qui ne peuvent pas se permettre de trop gros investissements informatiques.

Néanmoins, si cette approche convient tout à fait aux sites simples, elle pose de gros problèmes de cohérence, de maintenance, de gestion de projet et de performances pour les applications de plus grande ampleur. Dès lors, la « bidouille » ou le tâtonnement n'ont plus leur place : il faut se résoudre à adopter une démarche plus carrée, méthodique, reproductible, bref, un tant soit peu scientifique.

En même temps, si vous êtes, comme moi, assez réticent à adopter des processus de développement de projet qui semblent contraignants ou des outils de modélisation basés sur UML, le pas est délicat à franchir…

Vous êtes un développeur passionné, un « code warrior », et vous souhaitez découvrir en quoi la modélisation UML peut vous aider à structurer votre travail et à communiquer avec le reste de votre équipe de développement ? Vous êtes un chef de projet, un analyste/concepteur, et vous souhaitez comprendre comment UML permet de modéliser non plus des classes Java ou C++ mais des sites web complets ? Ce livre est fait pour vous !

Pascal Roques réalise ici un véritable tour de maître : il est parvenu à lier modélisation UML et architecture technique d'applications web, le tout orchestré par une démarche simple, claire et légère. Ce livre propose de mettre en œuvre la syntaxe UML adaptée à la modélisation d'applications en ligne, et décline l'analyse réalisée en UML sur trois architectures techniques : .NET, J2EE, et les langages de scripts (type PHP).

Contrairement aux ouvrages dédiés à une technologie particulière qui entrent dans les entrailles du code et des problématiques techniques, le lecteur découvrira les concepts nécessaires à la compréhension de chaque étape du processus « juste à temps », c'est-à-dire progressivement, au fil d'une étude de cas concrète et issue d'expériences et de projets réels. Tout en adoptant cette démarche très novatrice, Pascal a su doser les ingrédients de ce livre avec finesse.

En tant que formateur Java et .NET pour la société Valtech Training, je côtoie de nombreux élèves qui se forment aux technologies JSP/Servlets ou ASP.NET : tous maîtrisent rapidement les langages et outils. La véritable valeur ajoutée des consultants, des formateurs et des auteurs comme Pascal avec ce livre est de proposer une démarche et un cadre de travail qui facilitent le développement d'applications web ambitieuses.

Thomas Gil

Consultant-formateur indépendant et
gérant de la société DotNetGuru SARL

Table des matières

Introduction

Objectifs

La conception d'applications web est un sujet à la mode ! En feuilletant les catalogues des éditeurs informatiques, on est un peu submergé par le nombre d'ouvrages qui y sont consacrés et la liste n'a pas l'air de vouloir s'arrêter…

Cependant, quand on prend la peine de parcourir la table des matières de la grande majorité de ces livres, on est frappé de retrouver toujours les mêmes mots-clés : ergonomie, HTML, page, lien, graphisme, cadre, navigation, typographie, couleur, etc.

Bref, tout pour améliorer la forme, mais où est passé le fond ? Que vient faire l'internaute sur le site ? Quelles informations s'attend-il à trouver ? Comment ces informations sont-elles structurées, reliées entre elles, mises à jour ? Bref, comment garantir que les choix de réalisation de l'application web sont bien adaptés aux objectifs de l'utilisateur ?

La réponse tient en un seul mot : modéliser !

Depuis quelques années, la modélisation objet avec le langage UML est devenue incontournable sur la plupart des projets informatiques. Alors pourquoi ne pas appliquer aux projets web ce qui marche pour les projets « classiques »[1] ? Contrairement à une idée répandue, les applications web sont justement, de par leur complexité croissante, des candidates idéales à la modélisation graphique et à l'application d'un processus de développement formalisé.

---

1. Voir par exemple : *UML2 en action : de l'analyse des besoins à la conception*, P. Roques, F. Vallée, Eyrolles, 2007.

Le pionnier sur le sujet a été l'américain Jim Conallen qui a écrit en 1999 un livre intitulé : *Designing Web Applications with UML*[2]. Mais depuis sa parution, les technologies web ont bien sûr continué à évoluer, avec en particulier l'arrivée de la plateforme .NET de Microsoft (avec son langage phare C#), l'émergence des WebServices et des clients « riches ». Les processus de développement également, avec le mouvement prometteur des méthodologies dites « agiles », popularisées en particulier par Alistair Cockburn dans son ouvrage : *Agile Software Development*[3]. Enfin, le langage de modélisation UML a franchi un palier important en passant de la version 1.5 (utilisée dans la première édition de ce livre) à la version 2.0, puis 2.1.

Dans cet esprit, mon objectif est donc de vous fournir un guide de modélisation UML 2 précis, à jour, mais néanmoins léger pour mieux spécifier et réaliser vos applications web. Il ne s'agit pas d'un long exposé théorique mais bien plutôt de conseils concrets et pragmatiques, illustrés pas à pas grâce à une étude de cas réaliste d'un site marchand de vente en ligne.

2. La traduction française de cet ouvrage est paru chez Eyrolles en 2000 : *Concevoir des applications Web avec UML*, J. Conallen.

3. *Agile Software Development: Software through people*, A. Cockburn, Addison-Wesley 2002.

Remerciements

Comme pour mes autres livres, je remercie tout d'abord la société Valtech Training (www.valtech-training.fr) pour son soutien et son appui (avec un clin d'œil affectueux à Corinne Martinez et Suzi Lavail). J'ai profité de nombreuses discussions avec mes collègues consultants et formateurs (Sami Jaber, Denis Peyrusaubes, Daniel Rosenblatt, Gwenaëlle Tisserand, et bien d'autres) pour affiner le processus et les techniques de modélisation que je vous propose dans cet ouvrage.

Une mention spéciale à Thomas Gil, pour ses remarques constructives et sa participation notable à l'écriture initiale du chapitre 8. Pour les éditions suivantes, mes collègues Jean-Louis Vidal et Xavier Paradon qui m'ont fourni des mises à jour sur .NET et JSF, et Christophe Porteneuve[4] qui a eu la gentillesse de contribuer notablement à améliorer la précision du dernier chapitre.

Merci également à Jean-Bernard Crampes de L'IUT de Blagnac ainsi qu'à son équipe pour l'échange d'idées constructif sur la modélisation de la navigation dont vous trouverez l'écho dans le chapitre 6.

Enfin, je ne veux pas oublier les éditions Eyrolles qui m'ont fait confiance une fois de plus. Un merci tout particulier à Muriel et toute l'équipe, Sophie et Éliza pour leur enthousiasme, leur professionnalisme et leur bonne humeur !

Quant à Sylvie, elle sait que mon énergie ne serait pas la même sans elle…

Pascal Roques, juin 2008

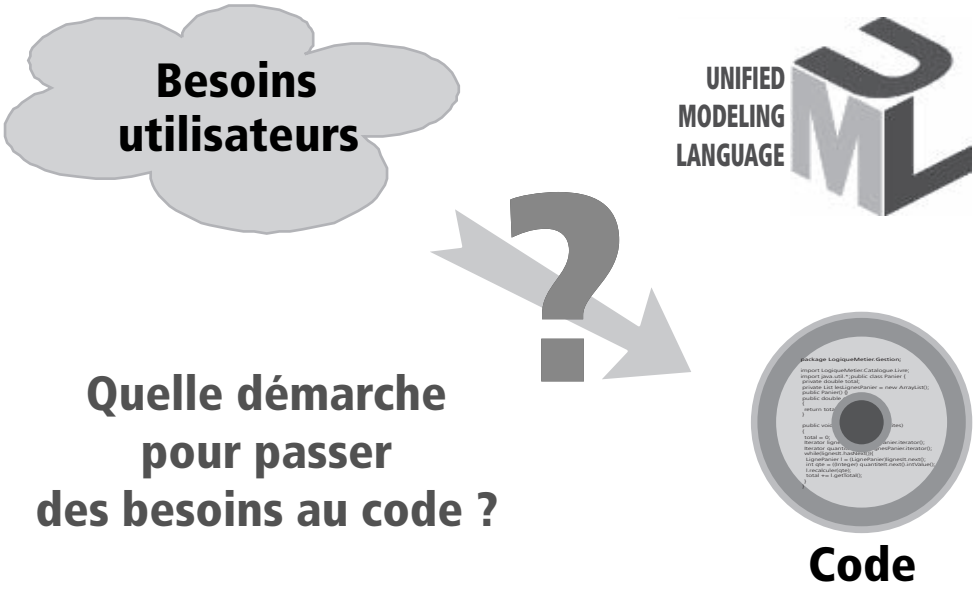
pascal.roques@gmail.com

blog : http://www.dotnetguru2.org/proques/

site : http://pascal.roques.googlepages.com/home

4. *Bien développer pour le Web 2.0 – Bonnes pratiques Ajax*, C. Porteneuve, Eyrolles 2006.

chapitre 1


Besoins utilisateurs
UNIFIED MODELING LANGUAGE
Quelle démarche pour passer des besoins au code ?
Code

Quelle démarche pour passer des besoins utilisateur au code de l'application ?

Dans ce chapitre introductif, nous dévoilons le processus simplifié que nous préconisons pour la modélisation des applications web. Après un premier tour rapide des différents types de diagrammes proposés par le langage de modélisation UML, nous introduirons ceux qui nous seront utiles. Nous présenterons également les principes fondamentaux du Processus Unifié (UP), du développement agile (avec eXtreme Programming et Scrum) et d'Agile Modeling (AM), afin d'éclairer les idées fortes auxquelles se rattache la démarche pratique adoptée dans la suite du livre.

SOMMAIRE

MOTS-CLÉS

- Modélisation
- UML
- Diagrammes
- Processus
- UP
- XP
- Scrum
- Agilité
- Web

Pourquoi modéliser ?

Le recours à la modélisation est depuis longtemps une pratique indispensable au développement logiciel, car un modèle est prévu pour arriver à anticiper les résultats du codage. Un modèle est en effet une représentation abstraite d'un système destiné à en faciliter l'étude et à le documenter. C'est un outil majeur de communication entre les différents intervenants au sein d'un projet. Chaque membre de l'équipe, depuis l'utilisateur jusqu'au développeur, utilise et enrichit le modèle différemment. En outre, les systèmes devenant de plus en plus complexes, leur compréhension et leur maîtrise globale dépassent les capacités d'un seul individu. La construction d'un modèle abstrait aide à y remédier. Le modèle présente notamment l'atout de faciliter la traçabilité du système, à savoir la possibilité de partir d'un de ses éléments et de suivre ses interactions et liens avec d'autres parties du modèle.

Associé au processus de développement, un modèle représente l'ensemble des vues sur une expression de besoins ou sur une solution technique. Pris à un niveau de détail pertinent, il décrit ou conçoit la cible de l'étape en cours. Le modèle sert donc des objectifs différents suivant l'activité de développement et sera construit avec des points de vue de plus en plus détaillés :

- Dans les activités de **spécification des exigences**, il convient premièrement de considérer le système comme une boîte noire à part entière afin d'étudier sa place dans le système métier plus global qu'est l'entreprise. On développe pour cela un modèle de niveau contexte, afin de tracer précisément les frontières fonctionnelles du système.

À RETENIR **Analogie**

Pour illustrer au mieux ce qu'est un modèle, Grady Booch a établi un parallèle entre le développement logiciel et la construction BTP. Cette analogie est judicieuse, car les plans tracés pour construire un immeuble reflètent parfaitement bien l'idée d'anticipation, de conception et de documentation du modèle. Chaque plan développe par ailleurs un point de vue différent suivant les corps de métier. Par exemple, le plan des circuits d'eau et le plan des passages électriques concernent le même immeuble mais sont nécessairement séparés. Enfin, chaque plan se situe à un niveau d'abstraction et de détail distinct suivant l'usage que l'on désire en faire. Ainsi, le plan de masse aide à anticiper les conséquences de l'implantation de l'immeuble sur son environnement, exactement comme le modèle de contexte. Viennent ensuite des plans de construction d'un étage, analogues aux modèles de conception.
Notons cependant que l'anticipation ne permet pas de prendre en compte les besoins changeants des utilisateurs, l'hypothèse de départ étant justement que ces besoins sont définis une bonne fois pour toutes. Or, dans bien des cas, ces besoins évoluent au fil du projet ; c'est pourquoi il est important de gérer le changement et d'admettre la nécessité de continuer à faire vivre nos modèles. Le processus de modélisation du logiciel doit être adaptatif et non pas prédictif, contrairement à ce qui se fait dans le BTP !

- Dans les **activités d'analyse**, le modèle commence à représenter le système vu de l'intérieur. Il se compose d'objets représentant une abstraction des concepts manipulés par les utilisateurs. Le modèle comprend par ailleurs deux points de vue, la structure statique et le comportement dynamique. Il s'agit de deux perspectives différentes qui aident à compléter la compréhension du système à développer.
- Dans les **activités de conception**, le modèle correspond aux concepts informatiques qui sont utilisés par les outils, les langages ou les plates-formes de développement. Le modèle sert ici à étudier, documenter, communiquer et anticiper une solution. Il est en effet toujours plus rentable de découvrir une erreur de conception sur un modèle, que de la découvrir au bout de milliers de lignes codées sans méthode. Pour la conception du déploiement enfin, le modèle représente également les matériels et les logiciels à interconnecter.

Le modèle en tant qu'abstraction d'un système s'accorde parfaitement bien avec les concepts orientés objet. Un objet peut en effet représenter l'abstraction d'une entité métier utilisée en analyse, puis d'un composant de solution logicielle en conception. La correspondance est encore plus flagrante lorsque les langages de développement sont eux-mêmes orientés objet. Cela explique le succès de la modélisation objet ces dernières années pour les projets de plus en plus nombreux utilisant C++, Java ou C#.

À RETENIR Qu'est-ce qu'un « bon » modèle ?

A est un bon modèle de B si A permet de répondre de façon satisfaisante à des questions prédéfinies sur B (d'après D.T. Ross).
Un bon modèle doit donc être construit :

- au bon niveau de détail,
- selon le bon point de vue.

Pensez à l'analogie de la carte routière. Pour circuler dans Toulouse, la carte de France serait de peu d'utilité. En revanche, pour aller de Toulouse à Paris, la carte de la Haute-Garonne ne suffit pas… À chaque voyage correspond la « bonne » carte !

Aujourd'hui, le standard industriel de modélisation objet est UML. Il est sous l'entière responsabilité de l'OMG.

B.A.-BA OMG

L'OMG (*Object Management Group*) est un groupement d'industriels dont l'objectif est de standardiser autour des technologies objet, afin de garantir l'interopérabilité des développements. L'OMG comprend actuellement plus de 800 membres, dont les principaux acteurs de l'industrie informatique (Sun, IBM, etc.), mais aussi les plus grandes entreprises utilisatrices dans tous les secteurs d'activité.

▸ www.omg.org

B.A.-BA Unified Modeling Language

Tous les documents sur UML élaborés dans le cadre de l'OMG sont publics et disponibles sur le site :

▸ www.uml.org.

Les bases d'UML

UML se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier et documenter des systèmes, esquisser des architectures logicielles, concevoir des solutions et communiquer des points de vue.

UML unifie à la fois les notations et les concepts orientés objet (voir l'historique d'UML sur la figure 1-1). Il ne s'agit pas d'une simple notation graphique, car les concepts transmis par un diagramme ont une sémantique précise et sont porteurs de sens au même titre que les mots d'un langage.

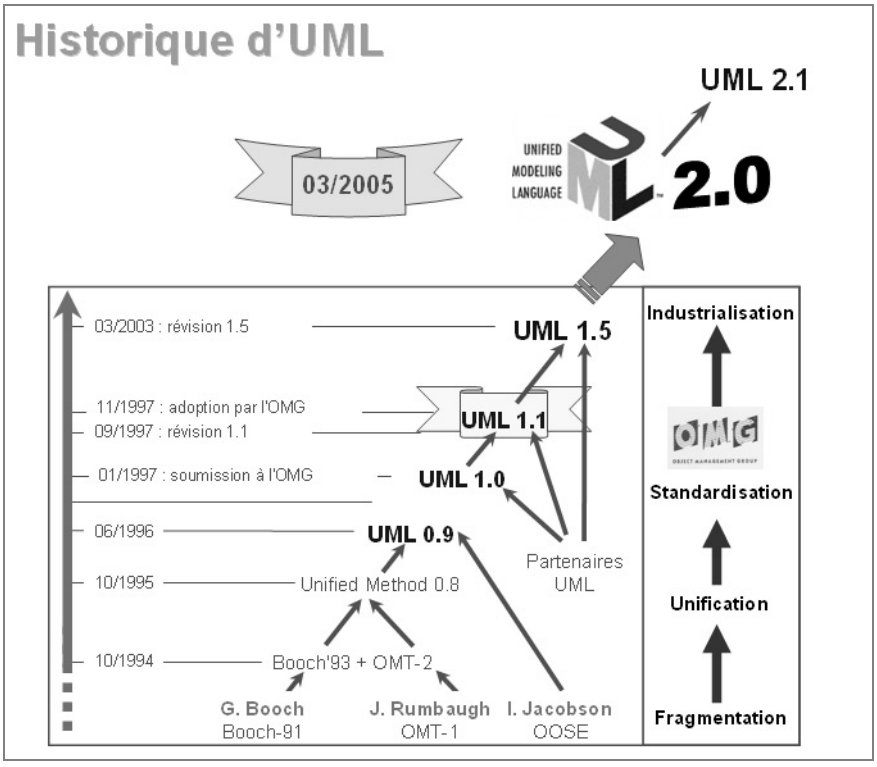


Figure 1–1
Historique d'UML

UML unifie également les notations nécessaires aux différentes activités d'un processus de développement et offre, par ce biais, le moyen d'établir le suivi des décisions prises, depuis l'expression de besoin jusqu'au codage. Dans ce cadre, un concept appartenant aux exigences des utilisateurs projette sa réalité dans le modèle de conception et dans le codage. Le fil tendu entre les différentes étapes de construction permet alors de remonter du code aux besoins et d'en comprendre les tenants et les aboutissants. En d'autres termes, on peut retrouver la nécessité d'un bloc de code en se référant à son origine dans le modèle des besoins.

UML 2 s'articule autour de treize types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. Ces types de diagrammes sont répartis en deux grands groupes :

- **Six diagrammes structurels** :
  - Diagramme de classes – Il montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.
  - Diagramme d'objets - Il montre les instances des éléments structurels et leurs liens à l'exécution.
  - Diagramme de packages - Il montre l'organisation logique du modèle et les relations entre packages.
  - Diagramme de structure composite – Il montre l'organisation interne d'un élément statique complexe.
  - Diagramme de composants – Il montre des structures complexes, avec leurs interfaces fournies et requises.
  - Diagramme de déploiement – Il montre le déploiement physique des « artefacts » sur les ressources matérielles.
- **Sept diagrammes comportementaux** :
  - Diagramme de cas d'utilisation - Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude.
  - Diagramme de vue d'ensemble des interactions - Il fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.
  - Diagramme de séquence - Il montre la séquence verticale des messages passés entre objets au sein d'une interaction.
  - Diagramme de communication - Il montre la communication entre objets dans le plan au sein d'une interaction.
  - Diagramme de temps – Il fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps.
  - Diagramme d'activité - Il montre l'enchaînement des actions et décisions au sein d'une activité.
  - Diagramme d'états – Il montre les différents états et transitions possibles des objets d'une classe.

Le diagramme de cas d'utilisation (figure 1-2) est utilisé dans l'activité de spécification des besoins. Il montre les interactions fonctionnelles entre les acteurs et le système à l'étude. Vous trouverez une description détaillée de son usage au chapitre 3 de cet ouvrage.

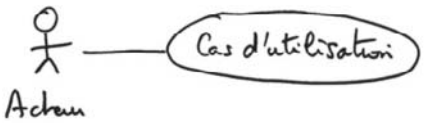


Figure 1–2
Diagramme de cas d'utilisation

Le diagramme de classes (figure 1-3) est le point central dans un développement orienté objet. En analyse, il a pour objet de décrire la struc-

ture des entités manipulées par les utilisateurs. Vous trouverez les explications relatives à cette utilisation au chapitre 5. En conception, le diagramme de classes représente la structure d'un code orienté objet. Vous retrouverez l'utilisation du diagramme de classes en conception aux chapitres 7 et 8.

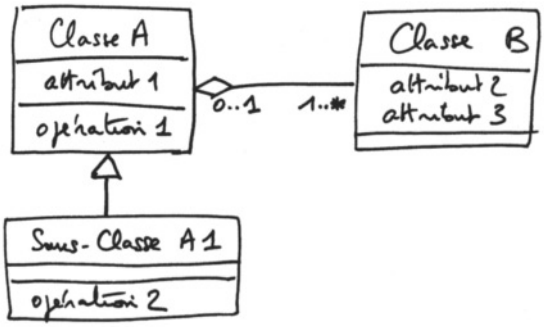


Figure 1–3
Diagramme de classes

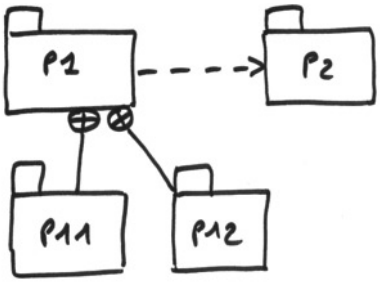


Figure 1–4
Diagramme de packages

Le diagramme de packages (figure 1-4) montre l'organisation logique du modèle et les relations entre packages. Il permet de structurer les classes d'analyse et de conception, mais aussi les cas d'utilisation. Vous verrez ces deux utilisations du diagramme de packages aux chapitres 3 et 8.

Les diagrammes de séquence (figure 1-5) et les diagrammes de communication (figure 1-6) sont tous deux des diagrammes d'interactions UML. Ils représentent des échanges de messages entre éléments, dans le cadre d'un fonctionnement particulier du système. Les diagrammes de séquence servent d'abord à développer en analyse les scénarios d'utilisation du système. Vous en trouverez des exemples au chapitre 4. Plus tard, les diagrammes de séquence et de communication permettent de concevoir les méthodes des classes comme indiqué aux chapitres 7 et 8. Nous privilégierons cependant nettement les diagrammes de séquence pour restreindre le nombre de diagrammes utilisés.

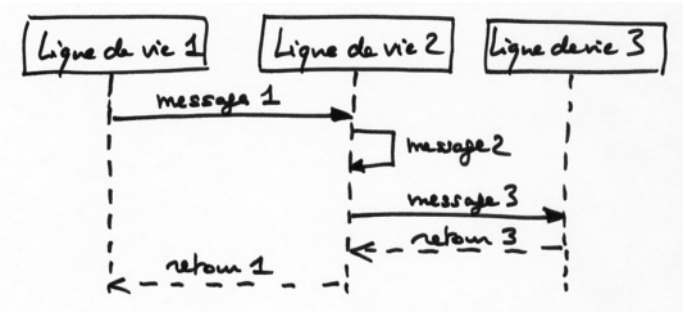


Figure 1–5
Diagramme de séquence

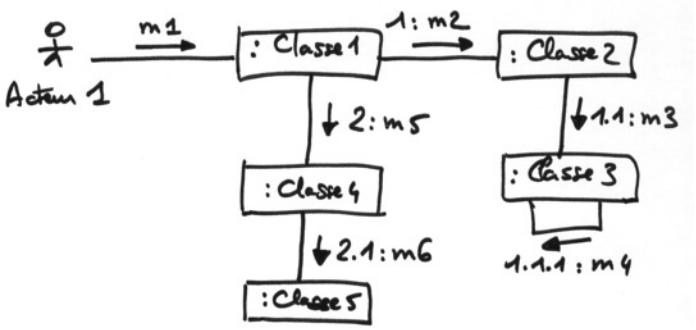


Figure 1–6
Diagramme de communication

Le diagramme d'états (figure 1-7) représente le cycle de vie commun aux objets d'une même classe. Ce diagramme complète la connaissance des classes en analyse et en conception en montrant les différents états et transitions possibles des objets d'une classe à l'exécution. Le chapitre 5 vous indiquera comment utiliser ce diagramme à des fins d'analyse.

Vous en verrez une utilisation particulière au chapitre 6 pour modéliser la navigation dans le site web.

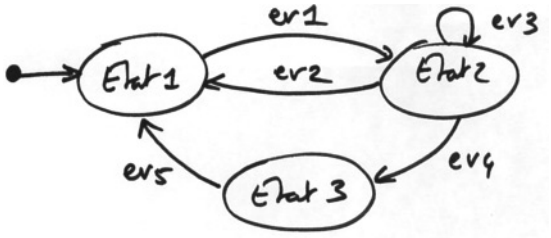


Figure 1–7
Diagramme d'états

Le diagramme d'activité (figure 1-8) représente les règles d'enchaînement des actions et décisions au sein d'une activité. Il peut également être utilisé comme alternative au diagramme d'états pour décrire la navigation dans un site web, comme illustré au chapitre 6.

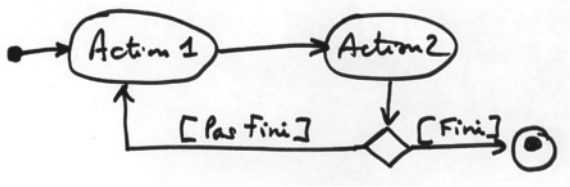


Figure 1–8
Diagramme d'activité

Le diagramme d'objets (figure 1-9) est un instantané, une photo d'un sous-ensemble des objets d'un système à un certain moment du temps. C'est probablement le diagramme le moins utilisé d'UML et nous n'en verrons pas d'illustration.

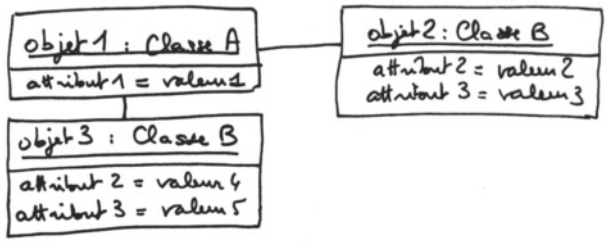


Figure 1–9
Diagramme d'objets

Le diagramme de composants (figure 1-10) montre les unités logicielles à partir desquelles on a construit le système informatique, ainsi que leurs dépendances.

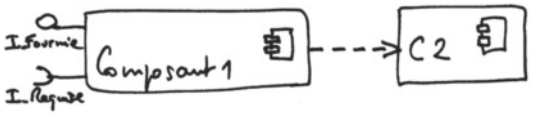


Figure 1–10
Diagramme de composants

Le diagramme de déploiement (figure 1-11) montre le déploiement physique des artefacts (éléments concrets tels que fichiers, exécutables, etc.) sur les ressources matérielles.

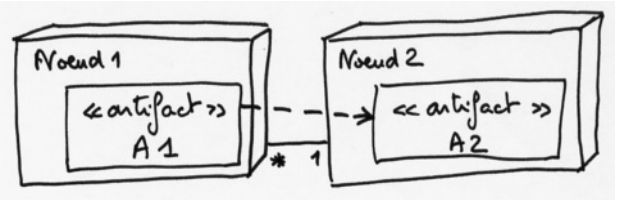


Figure 1–11
Diagramme de déploiement

Le diagramme de vue d'ensemble des interactions fusionne les diagrammes d'activité et de séquence pour combiner des fragments d'interaction avec des décisions et des flots.

Le diagramme de temps fusionne les diagrammes d'états et de séquence pour montrer l'évolution de l'état d'un objet au cours du temps et les messages qui modifient cet état.

Le diagramme de structure composite montre l'organisation interne d'un élément statique complexe sous forme d'un assemblage de parties, de connecteurs et de ports.

Dans un souci de simplicité, nous n'utiliserons pas ces trois nouveaux types de diagrammes proposés par UML 2.

L'ensemble des treize types de diagrammes UML peut ainsi être résumé sur la figure 1-12, en mettant en évidence les cinq diagrammes que nous utiliserons prioritairement.

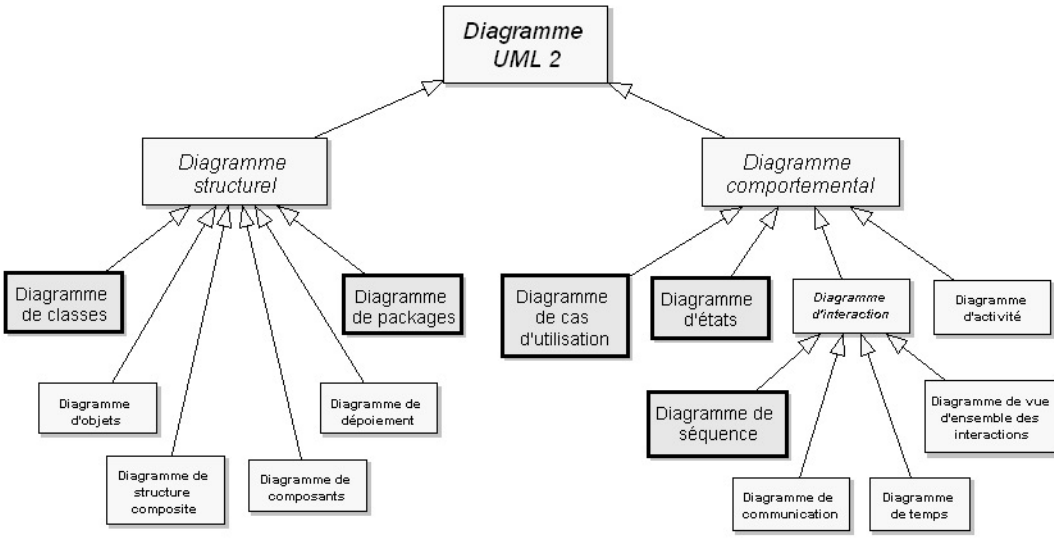


Figure 1–12 Les diagrammes UML utilisés dans notre démarche agile

Un processus simplifié pour les applications web

Le processus que nous vous proposons de suivre pour le développement d'applications web se situe à mi-chemin entre UP (Unified Process), un cadre général très complet de processus de développement, et les méthodes agiles en vogue actuellement, telles que XP (eXtreme Programming), et Scrum. Il s'inspire également des bonnes pratiques prônées par les tenants de la modélisation agile (Agile Modeling).

Les principes fondamentaux du Processus Unifié (UP)

Le Processus Unifié (UP, pour Unified Process) est un processus de développement logiciel « itératif et incrémental, centré sur l'architecture, conduit par les cas d'utilisation et piloté par les risques » :

- **Itératif et incrémental** : le projet est découpé en itérations de courte durée (environ 1 mois) qui aident à mieux suivre l'avancement global. À la fin de chaque itération, une partie exécutable du système final est produite, de façon incrémentale.

B.A.-BA **Processus de développement**

Un processus définit une séquence d'étapes, partiellement ordonnées, qui concourent à l'obtention d'un système logiciel ou à l'évolution d'un système existant. L'objet d'un processus de développement est de produire des logiciels de qualité qui répondent aux besoins de leurs utilisateurs dans des temps et des coûts prévisibles.
Plus simplement, un processus doit permettre de répondre à la question fondamentale : « Qui fait quoi et quand ? ».

- **Centré sur l'architecture** : tout système complexe doit être décomposé en parties modulaires afin de garantir une maintenance et une évolution facilitées. Cette architecture (fonctionnelle, logique, matérielle, etc.) doit être modélisée en UML et pas seulement documentée en texte.
- **Piloté par les risques** : les risques majeurs du projet doivent être identifiés au plus tôt, mais surtout levés le plus rapidement possible. Les mesures à prendre dans ce cadre déterminent l'ordre des itérations.
- **Conduit par les cas d'utilisation** : le projet est mené en tenant compte des besoins et des exigences des utilisateurs. Les cas d'utilisation du futur système sont identifiés, décrits avec précision et priorisés.

Les phases et les disciplines de UP

La gestion d'un tel processus est organisée suivant les quatre phases suivantes : initialisation, élaboration, construction et transition.

La phase d'initialisation conduit à définir la « vision » du projet, sa portée, sa faisabilité, son business case, afin de pouvoir décider au mieux de sa poursuite ou de son arrêt.

La phase d'élaboration poursuit trois objectifs principaux en parallèle :

- identifier et décrire la majeure partie des besoins des utilisateurs,
- construire (et pas seulement décrire dans un document !) l'architecture de base du système,
- lever les risques majeurs du projet.

La phase de construction consiste surtout à concevoir et implémenter l'ensemble des éléments opérationnels (autres que ceux de l'architecture de base). C'est la phase la plus consommatrice en ressources et en effort.

Enfin, la phase de transition permet de faire passer le système informatique des mains des développeurs à celles des utilisateurs finaux. Les mots-clés sont : conversion des données, formation des utilisateurs, déploiement, béta-tests.

Chaque phase est elle-même décomposée séquentiellement en itérations limitées dans le temps (entre 2 et 4 semaines). Le résultat de chacune d'elles est un système testé, intégré et exécutable. L'approche itérative est fondée sur la croissance et l'affinement successifs d'un système par le biais d'itérations multiples, feedback et adaptation cycliques étant les moteurs principaux permettant de converger vers un système satisfaisant. Le système croît avec le temps de façon incrémentale, itération par itération, et c'est pourquoi cette méthode porte également le nom de développement itératif et incrémental. Il s'agit là du principe le plus important du Processus Unifié.

Les activités de développement sont définies par cinq disciplines fondamentales qui décrivent la capture des exigences, l'analyse et la conception, l'implémentation, le test et le déploiement. La modélisation métier est une discipline amont optionnelle et transverse aux projets. Enfin, trois disciplines appelées de support complètent le tableau : gestion de projet, gestion du changement et de la configuration, ainsi que la mise à disposition d'un environnement complet de développement incluant aussi bien des outils informatiques que des documents et des guides méthodologiques.

UP doit donc être compris comme une trame commune des meilleures pratiques de développement, et non comme l'ultime tentative d'élaborer un processus universel.

Le schéma synthétique du RUP™ (Rational Unified Process)

Contrairement au processus en cascade (souvent appelé cycle en V, en France), le Processus Unifié ne considère pas que les disciplines sont purement séquentielles. En fait, une itération comporte une certaine quantité de travail dans la plupart des disciplines. Cependant, la répartition de l'effort relatif entre celles-ci change avec le temps. Les premières itérations ont tendance à mettre plus l'accent sur les exigences et la conception, les autres moins, à mesure que les besoins et l'architecture se stabilisent grâce au processus de feedback et d'adaptation.

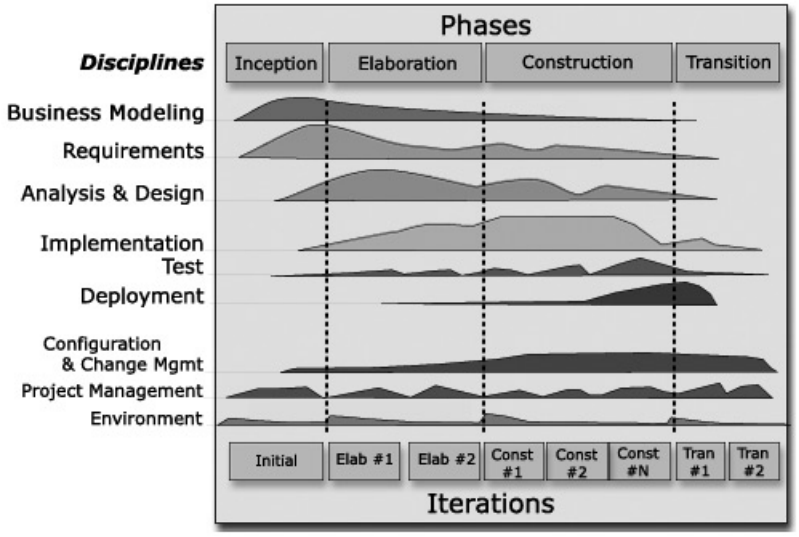


Figure 1–13
Les deux dimensions du Processus Unifié d'après RUP™

B.A.-BA OpenUP

OpenUP est une initiative intéressante pour simplifier le RUP et en proposer une version libre en tant que partie du framework EPF (*Eclipse Process Framework*) :

- http://epf.eclipse.org/wikis/openup/

Les principes du Manifeste Agile

▸ www.agilemanifesto.org

La notion de méthode agile est née à travers un manifeste signé en 2001 par 17 personnalités du développement logiciel (parmi lesquelles Ward Cunningham, Alistair Cockburn, Kent Beck, Martin Fowler, Ron Jeffries, Steve Mellor, Robert C. Martin, Ken Schwaber, Jeff Sutherland, etc.).

Ce manifeste prône quatre valeurs fondamentales :

- « Personnes et interactions plutôt que processus et outils » : dans l'optique agile, l'équipe est bien plus importante que les moyens matériels ou les procédures. Il est préférable d'avoir une équipe soudée et qui communique, composée de développeurs moyens, plutôt qu'une équipe composée d'individualistes, même brillants. La communication est une notion fondamentale.
- « Logiciel fonctionnel plutôt que documentation complète » : il est vital que l'application fonctionne. Le reste, et notamment la documentation technique, est secondaire, même si une documentation succincte et précise est utile comme moyen de communication. La documentation représente une charge de travail importante et peut être néfaste si elle n'est pas à jour. Il est préférable de commenter abondamment le code lui-même, et surtout de transférer les compétences au sein de l'équipe (on en revient à l'importance de la communication).
- « Collaboration avec le client plutôt que négociation de contrat » : le client doit être impliqué dans le développement. On ne peut se contenter de négocier un contrat au début du projet, puis de négliger les demandes du client. Le client doit collaborer avec l'équipe et fournir un feedback continu sur l'adaptation du logiciel à ses attentes.
- « Réagir au changement plutôt que suivre un plan » : la planification initiale et la structure du logiciel doivent être flexibles afin de permettre l'évolution de la demande du client tout au long du projet. Les premières *releases* du logiciel vont souvent provoquer des demandes d'évolution.

Les pratiques d'eXtreme Programming (XP)

L'eXtreme Programming (XP) est un ensemble de pratiques qui couvre une grande partie des activités de la réalisation d'un logiciel, de la programmation proprement dite à la planification du projet, en passant par l'organisation de l'équipe de développement et les échanges avec le client. Ces pratiques ne sont pas révolutionnaires : il s'agit simplement de pratiques de bon sens mises en œuvre par des développeurs ou des chefs de projet expérimentés, telles que :

- Un utilisateur à plein-temps dans la salle projet. Ceci permet une communication intensive et permanente entre les clients et les développeurs, aussi bien pour l'expression des besoins que pour la validation des livraisons.

- Écrire le test unitaire avant le code qu'il doit tester, afin d'être certain que le test sera systématiquement écrit et non pas négligé.
- Programmer en binôme, afin d'homogénéiser la connaissance du système au sein des développeurs, et de permettre aux débutants d'apprendre auprès des experts. Le code devient ainsi une propriété collective et non individuelle, que tous les développeurs ont le droit de modifier.
- Intégrer de façon continue, pour ne pas repousser à la fin du projet le risque majeur de l'intégration des modules logiciels écrits par des équipes ou des personnes différentes. Etc.

Pour résumer, on peut dire que XP est une méthodologie légère qui met l'accent sur l'activité de programmation et qui s'appuie sur les valeurs suivantes : communication, simplicité et feedback. Elle est bien adaptée pour des projets de taille moyenne où le contexte (besoins des utilisateurs, technologies informatiques) évolue en permanence.

- 🕮 *Gestion de projet Extreme Programming*, J.L. Bénard *et al.*, Eyrolles, 2002.
- 🕮 *Maîtriser les projets avec l'Extreme Programming – Pilotage par les tests-client*, T. Cros, Cépaduès, 2004
- ▸ http://fr.wikipedia.org/wiki/Extreme_programming
- ▸ http://etreagile.thierrycros.net/

Les bases de Scrum

Scrum est issu des travaux de deux des signataires du Manifeste Agile, Ken Schwaber et Jeff Sutherland, au début des années 1990. Le terme *Scrum* est emprunté au rugby et signifie mêlée. Ce processus agile s'articule en effet autour d'une équipe soudée, qui cherche à atteindre un but, comme c'est le cas en rugby pour avancer avec le ballon pendant une mêlée.

Le principe de base de Scrum est de focaliser l'équipe de façon itérative sur un ensemble de fonctionnalités à réaliser, dans des itérations de 30 jours, appelées *Sprints*. Chaque Sprint possède un but à atteindre, défini par le directeur de produit (*Product owner*), à partir duquel sont choisies les fonctionnalités à implémenter dans ce Sprint. Un Sprint aboutit toujours sur la livraison d'un produit partiel fonctionnel. Pendant ce temps, le *scrummaster* a la charge de réduire au maximum les perturbations extérieures et de résoudre les problèmes non techniques de l'équipe.

Un principe fort en Scrum est la participation active du client pour définir les priorités dans les fonctionnalités du logiciel, et choisir lesquelles seront réalisées dans chaque Sprint. Il peut à tout moment ajouter ou modifier la liste des fonctionnalités à réaliser, mais jamais ce qui est en cours de réalisation pendant un Sprint.

- ▸ http://fr.wikipedia.org/wiki/Scrum
- ▸ http://scrum.aubryconseil.com/

La modélisation agile (AM)

La « modélisation agile » prônée par Scott Ambler s'appuie sur des principes simples et de bon sens, parmi lesquels :

- Vous devriez avoir une grande palette de techniques à votre disposition et connaître les forces et les faiblesses de chacune de manière à pouvoir appliquer la meilleure au problème courant.

- N'hésitez pas à changer de diagramme quand vous sentez que vous n'avancez plus avec le modèle en cours. Le changement de perspective va vous permettre de voir le problème sous un autre angle et de mieux comprendre ce qui bloquait précédemment.
- Vous trouverez souvent que vous êtes plus productif si vous créez plusieurs modèles simultanément plutôt qu'en vous focalisant sur un seul type de diagramme.

▸ http://www.agile-modeling.com/

Le processus proposé dans cet ouvrage

Le processus que nous allons présenter et appliquer tout au long de ce livre est :

- conduit par les cas d'utilisation, comme UP, mais beaucoup plus simple ;
- relativement léger et restreint, comme les méthodes agiles, mais sans négliger les activités de modélisation en analyse et conception ;
- fondé sur l'utilisation d'un sous-ensemble nécessaire et suffisant du langage UML, conformément à AM.

Nous allons donc essayer de trouver le meilleur rapport « qualité/prix » possible afin de ne pas surcharger le lecteur de concepts et d'activités de modélisation qui ne sont pas indispensables au développement d'applications web efficaces. En revanche, nous nous efforcerons bien de montrer qu'il est important et utile de modéliser précisément certains aspects critiques de son système.

Le problème fondamental auquel ce livre va s'efforcer de répondre est finalement assez simple : comment passer des besoins des utilisateurs au code de l'application ? Autrement dit : « J'ai une bonne idée de ce que mon application doit faire, des fonctionnalités attendues par les futurs utilisateurs. Comment obtenir le plus efficacement possible un code informatique opérationnel, complet, testé, et qui réponde parfaitement au besoin ? ».

À RETENIR Règle des 80/20

La célèbre règle des 80/20 peut aussi s'appliquer dans notre cas : vous pouvez modéliser 80 % de vos problèmes en utilisant 20 % d'UML ! Encore faut-il savoir quels sont ces 20 % indispensables… Nous espérons que vous aurez une réponse claire et précise à cette question cruciale à l'issue de la lecture de cet ouvrage.

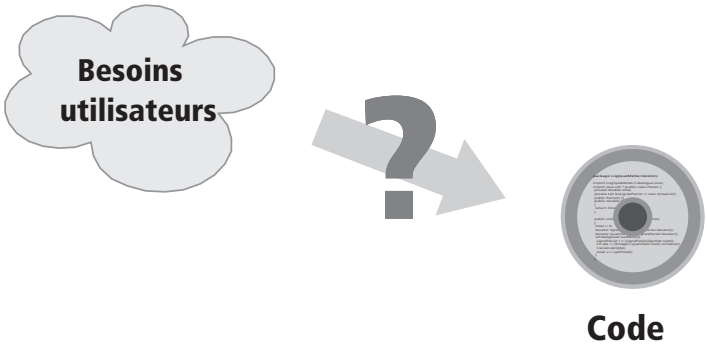


Figure 1–14
Comment passer des besoins au code ?

Il ne s'agit pas de se jeter sur l'écriture de code en omettant de formaliser les besoins des utilisateurs et d'élaborer une architecture robuste et évolutive. D'un autre côté, le but n'est pas de faire de la modélisation pour le plaisir, mais bien de produire le plus rapidement possible une application qui satisfasse au mieux ses utilisateurs !

Nous allons donc vous proposer une démarche de modélisation nécessaire et suffisante afin de construire efficacement une application web. Pour cela, nous utiliserons un sous-ensemble du langage de modélisation UML qui sera également nécessaire et suffisant pour la plupart des projets de même nature. Cette approche est le résultat de plusieurs années d'expérience dans des domaines variés. Elle a donc montré son efficacité dans la pratique.

Dans un premier temps, les besoins vont être modélisés au moyen des cas d'utilisation UML. Ils seront représentés de façon plus concrète par une maquette d'IHM (Interface Homme-Machine) destinée à faire réagir les futurs utilisateurs. La figure 1-15 montre bien de quoi nous partons et là où nous voulons arriver.

▸ La technique des cas d'utilisation sera expliquée au chapitre 3.

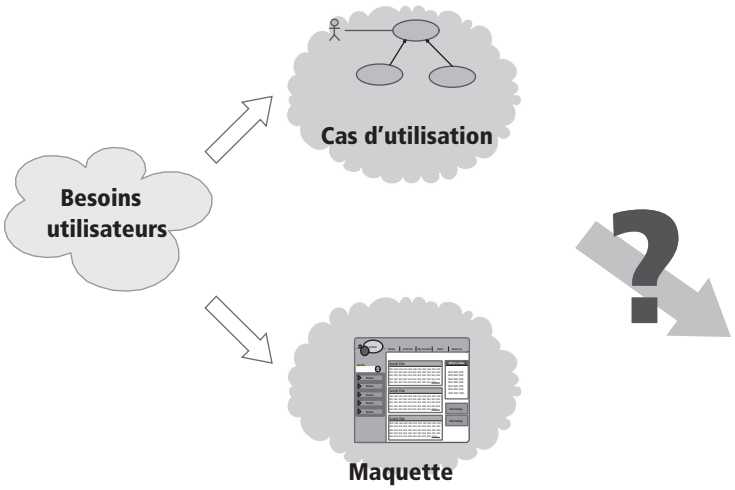


Figure 1–15
Les besoins donnent lieu à des cas d'utilisation et à une maquette

Repartons maintenant du but, c'est-à-dire du code que nous voulons obtenir, et remontons en arrière, pour mieux expliquer le chemin minimal qui va nous permettre de joindre les deux « bouts ». Dans le cadre de systèmes orientés objet, la structure du code est définie par les classes logicielles et leurs regroupements en ensembles appelés *packages* (paquetages en français). Nous avons donc besoin de diagrammes représentant les classes logicielles et montrant les données qu'elles contiennent (appelées attributs), les services qu'elles rendent (appelés opérations) ainsi que leurs relations. UML propose les diagrammes de classes pour véhiculer toutes ces informations.

Nous appellerons ces diagrammes « diagrammes de classes de conception » pour indiquer qu'ils sont à un niveau de détail suffisant pour en dériver automatiquement ou manuellement le code de l'application. Ils seront présentés aux chapitres 7 et 8.

B.A.-BA **Maquette**

Une maquette est un produit jetable donnant aux utilisateurs une vue concrète mais non définitive de la future interface de l'application. Cela peut consister en un ensemble de dessins réalisés avec des outils spécialisés tels que Dreamweaver, Adobe Illustrator ou plus simplement avec Powerpoint ou même Word. Par la suite, la maquette intégrera des fonctionnalités de navigation pour que l'utilisateur puisse tester l'enchaînement des écrans, même si les fonctionnalités restent fictives.
La maquette est développée rapidement afin de provoquer des retours de la part des utilisateurs. Elle permet ainsi d'améliorer la relation développeur-client. La plupart du temps, la maquette est considérée comme jetable, c'est-à-dire que la technologie informatique employée pour la réaliser n'est pas forcément assez robuste et évolutive pour être intégrée telle quelle. Pensez à l'analogie de la maquette d'avion qui est très utile en soufflerie, mais qui ne peut pas voler !

La figure 1-16 montre ainsi cette étape préliminaire au codage, mais il reste encore beaucoup de chemin à parcourir…

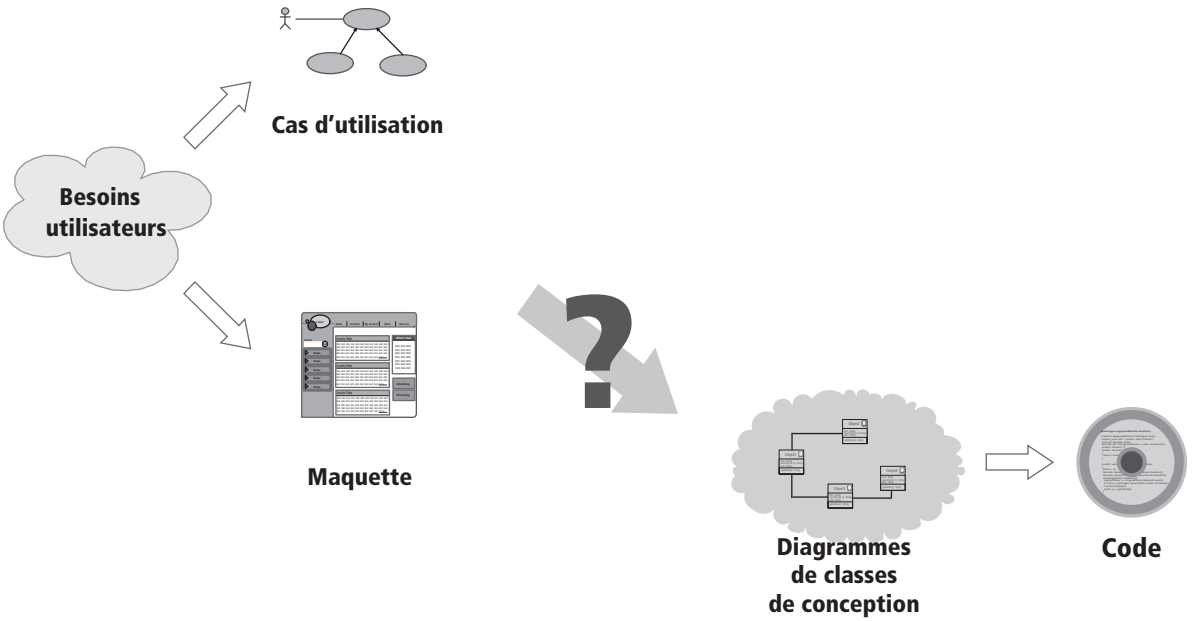


Figure 1–16 Les diagrammes de classes de conception donnent la structure du code.

Les diagrammes de classes de conception représentent bien la structure statique du code, par le biais des attributs et des relations entre classes, mais ils contiennent également les opérations (aussi appelées méthodes) qui décrivent les responsabilités dynamiques des classes logicielles. L'attribution des bonnes responsabilités aux bonnes classes est l'un des problèmes les plus délicats de la conception orientée objet. Pour chaque service ou fonction, il faut décider quelle est la classe qui va le/la con-

tenir. Nous devons ainsi répartir tout le comportement du système entre les classes de conception, et décrire les interactions induites.

Les diagrammes d'interactions UML (séquence ou communication) sont particulièrement utiles au concepteur pour représenter graphiquement ses décisions d'allocation de responsabilités. Chaque diagramme d'interaction va ainsi représenter un ensemble d'objets de classes différentes collaborant dans le cadre d'un scénario d'exécution du système. Dans ce genre de diagramme, les objets communiquent en s'envoyant des messages qui invoquent des opérations sur les objets récepteurs.

On peut donc suivre visuellement les interactions dynamiques entre objets, et les traitements réalisés par chacun. Les diagrammes d'interaction aident également à écrire le code à l'intérieur des opérations, en particulier les appels d'opérations imbriqués.

La figure 1-17 ajoute une étape du côté du code, mais ne nous dit pas encore comment relier tout cela aux cas d'utilisation.

MÉTHODE Allocation des responsabilités

Avec un outil de modélisation UML, à chaque fois que vous déclarez un message entre deux objets, vous pouvez créer effectivement une opération publique sur la classe de l'objet récepteur. Ce type d'outil permet vraiment de mettre en œuvre l'allocation des responsabilités à partir des diagrammes d'interaction.

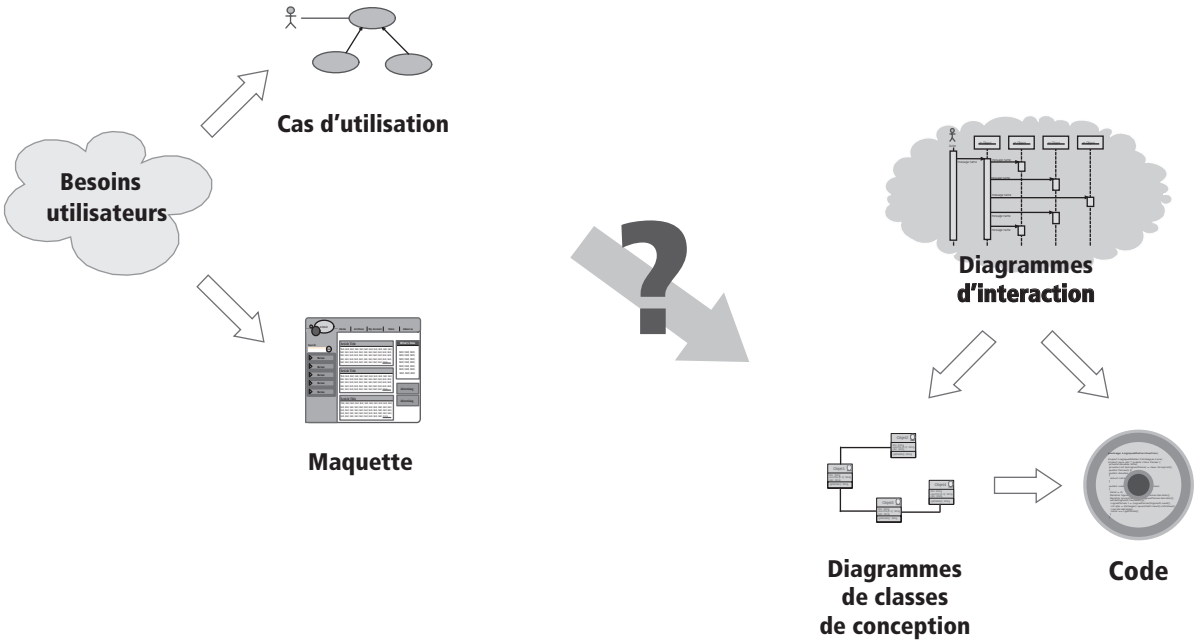


Figure 1–17 Les diagrammes d'interaction nous aident à attribuer les responsabilités aux classes.

Comment passer des cas d'utilisation aux diagrammes d'interaction ? Ce n'est ni simple ni direct, car les cas d'utilisation sont au niveau d'abstraction des besoins des utilisateurs alors que les diagrammes d'interaction se placent au niveau de la conception objet. Il faut donc au moins une étape intermédiaire.

▸ Les diagrammes d'interaction seront utilisés intensivement aux chapitres 7 et 8.

▸ La description textuelle détaillée des cas d'utilisation ainsi que le diagramme de séquence système seront présentés au chapitre 4.

Chaque cas d'utilisation est décrit textuellement de façon détaillée, mais donne également lieu à un diagramme de séquence simple représentant graphiquement la chronologie des interactions entre les acteurs et le système vu comme une boîte noire, dans le cadre du scénario nominal. Nous appellerons ce diagramme : « diagramme de séquence système ».

Par la suite, en remplaçant le système vu comme une boîte noire par un ensemble choisi d'objets de conception, nous décrirons l'attribution des responsabilités dynamiques, tout en conservant une traçabilité forte avec les cas d'utilisation. La figure 1-18 montre ainsi les diagrammes de séquence système en tant que liens importants entre les cas d'utilisation et les diagrammes d'interaction.

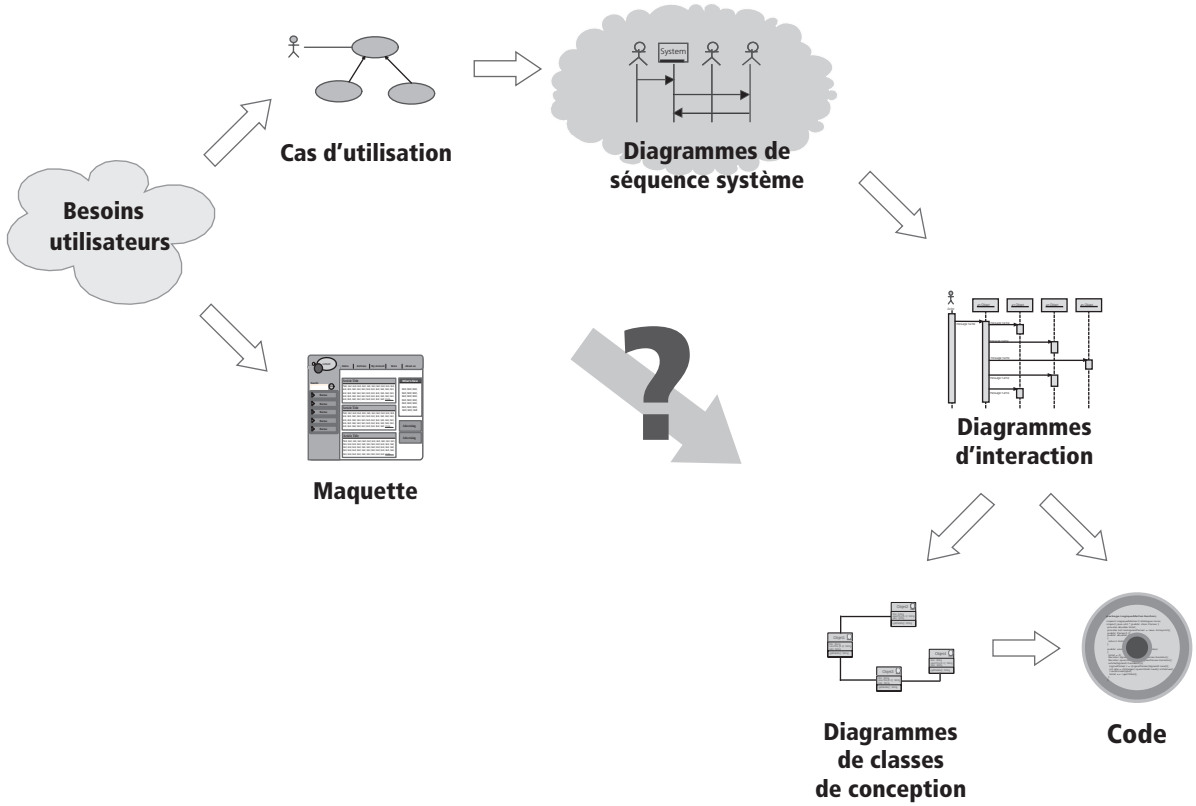


Figure 1–18 Les diagrammes de séquence système fournissent le squelette des diagrammes d'interaction.

Maintenant, comment trouver ces fameuses classes de conception qui interviennent dans les diagrammes d'interaction ?

Le chaînon manquant de notre démarche s'appelle les diagrammes de classes participantes. Il s'agit là de diagrammes de classes UML qui décrivent, cas d'utilisation par cas d'utilisation, les trois principales

classes d'analyse et leurs relations : les dialogues, les contrôles et les entités. Ces classes d'analyse, leurs attributs et leurs relations vont être décrits en UML par un diagramme de classes simplifié utilisant des conventions particulières.

Un avantage important de cette technique pour le chef de projet consiste en la possibilité de découper le travail de son équipe d'analystes suivant les différents cas d'utilisation, plutôt que de vouloir tout traiter d'un bloc. Comme l'illustre la figure 1-19, les diagrammes de classes participantes sont particulièrement importants car ils font la jonction entre les cas d'utilisation, la maquette et les diagrammes de conception logicielle (diagrammes d'interaction et diagrammes de classes).

▸ Les diagrammes de classes participantes seront détaillés au chapitre 5.

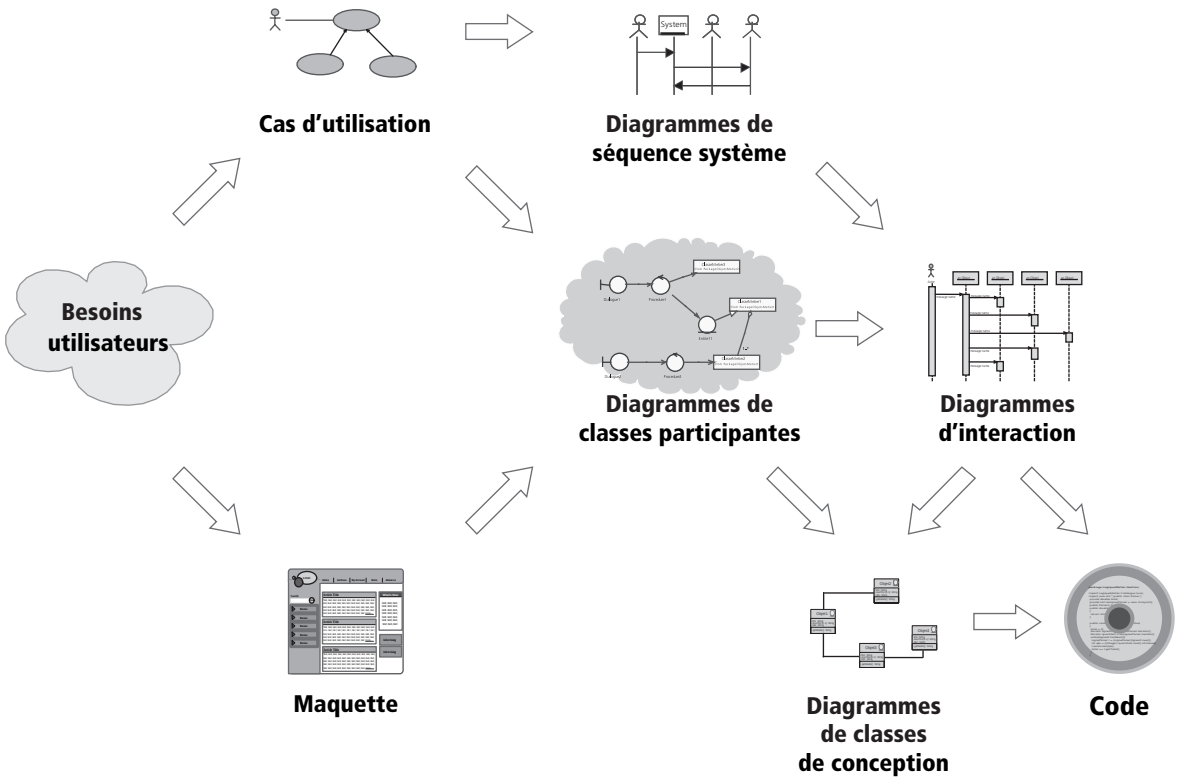


Figure 1–19 Les diagrammes de classes participantes font la jonction entre les cas d'utilisation, la maquette et les diagrammes de conception logicielle.

Pour que le tableau soit complet, il nous reste à détailler une exploitation supplémentaire de la maquette. Elle va nous permettre de réaliser des diagrammes dynamiques représentant de manière formelle l'ensemble des chemins possibles entre les principaux écrans proposés à l'utilisateur. Ces diagrammes, qui mettent en jeu les classes participantes de type « dialogues » et « contrôles », s'appellent des diagrammes de navigation.

▸ Les diagrammes de navigation seront présentés au chapitre 6.

B.A.-BA **Dialogues, contrôles, entités**

Les classes qui permettent les interactions entre le site web et ses utilisateurs sont qualifiées de « dialogues ». C'est typiquement les écrans proposés à l'utilisateur : les formulaires de saisie, les résultats de recherche, etc. Elles proviennent directement de l'analyse de la maquette.
Celles qui contiennent la cinématique de l'application seront appelées « contrôles ». Elles font la transition entre les dialogues et les classes métier, en permettant aux écrans de manipuler des informations détenues par un ou plusieurs objet(s) métier.
Celles qui représentent les règles métier sont qualifiées d'« entités ». Elles proviennent directement du modèle du domaine, mais sont confirmées et complétées cas d'utilisation par cas d'utilisation.

La trame globale de la démarche est ainsi finalisée, comme indiqué sur la figure 1-20.

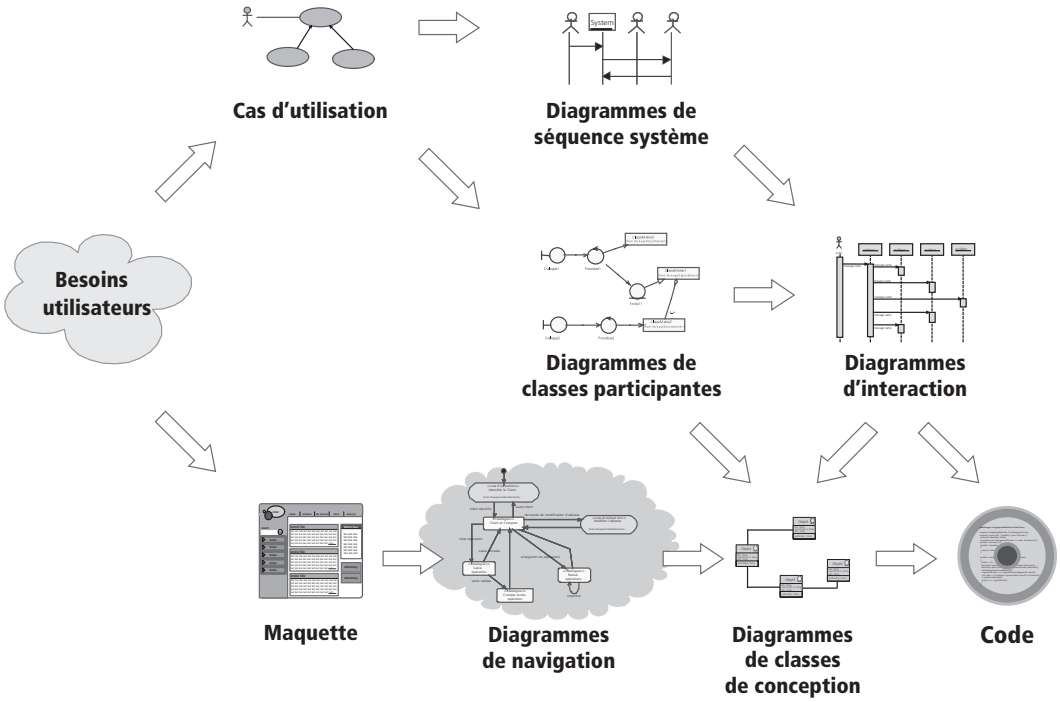


Figure 1–20 Schéma complet du processus de modélisation d'une application web

MÉTHODE **Quelques types de diagrammes seulement**

Nous n'utilisons pas les treize types de diagrammes proposés par UML 2, mais seulement un tiers, en insistant particulièrement sur les diagrammes de classes et les diagrammes de séquence. Cette limitation volontaire permet une réduction significative du temps d'apprentissage de la modélisation avec UML, tout en restant largement suffisante pour la plupart des projets.
C'est cela aussi la « modélisation agile » !

Nous avons obtenu un schéma complet montrant comment, en partant des besoins des utilisateurs formalisés par des cas d'utilisation et une maquette, et avec l'apport des diagrammes de classes participantes, on peut aboutir à des diagrammes de conception à partir desquels on dérivera du code assez directement.

Organisation du livre

Après ce premier chapitre introductif, qui nous a permis de poser les bases de la démarche, nous allons détailler une par une, dans la suite de ce livre, les étapes permettant d'aboutir à l'ensemble des livrables de la figure précédente.

La figure 1-21 replace ainsi chacun des chapitres suivants dans le cadre de la démarche de modélisation.

Ce schéma général nous servira d'introduction pour chaque chapitre du livre, afin de bien replacer l'étape courante dans la démarche globale de modélisation d'une application web.

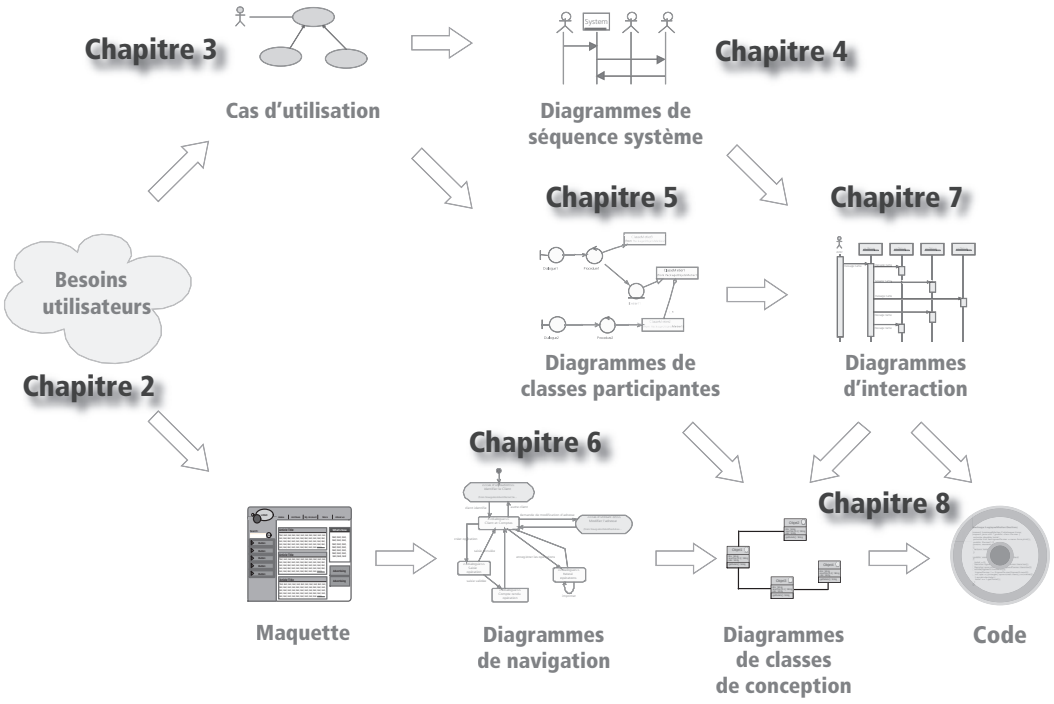
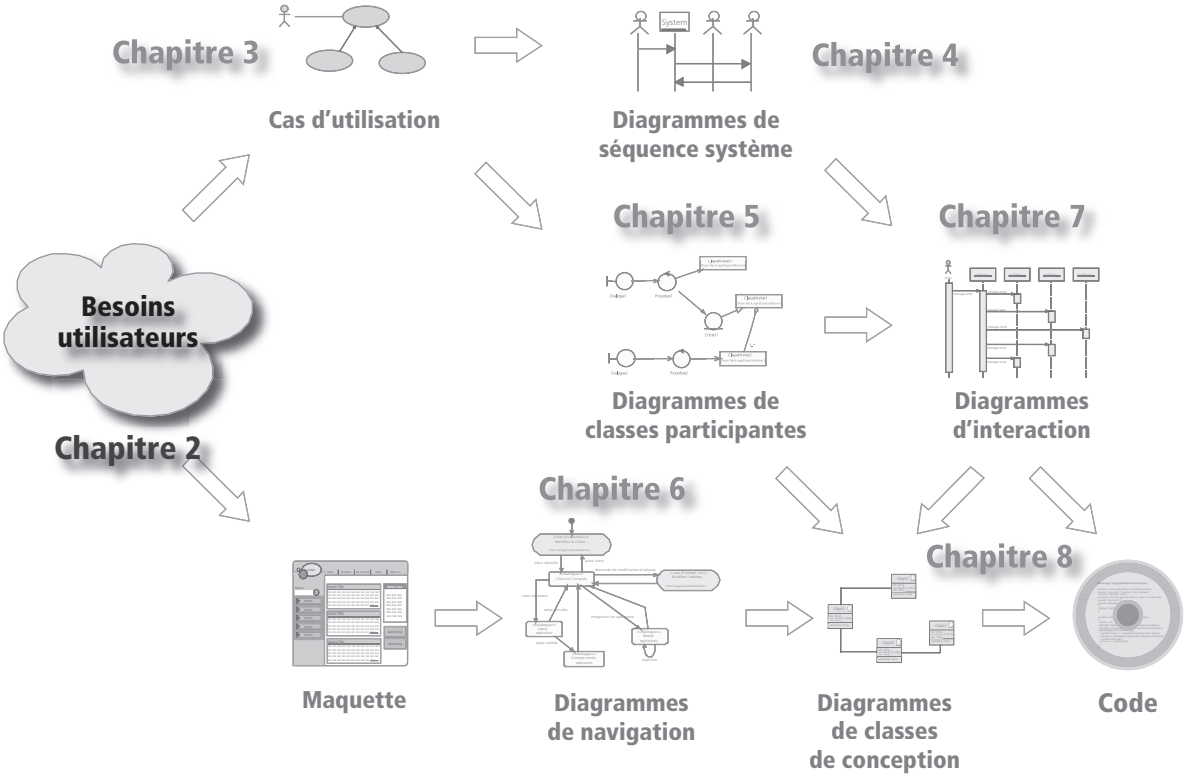


Figure 1–21 Les chapitres du livre replacés dans la démarche globale de modélisation

chapitre 2


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Diagrammes de classes participantes
Chapitre 7
Diagrammes d'interaction
Besoins utilisateurs
Chapitre 2
Chapitre 6
Maquette
Diagrammes de navigation
Chapitre 8
Diagrammes de classes de conception
Code

Fonctionnalités d'une librairie en ligne : l'application côté utilisateur

Présentons d'abord l'étude de cas traitée dans la suite du livre : une librairie en ligne. Pour cela, nous détaillerons dans un premier temps les exigences fonctionnelles du site marchand, à savoir les fonctionnalités requises par l'utilisateur : recherche, découverte détaillée, sélection et commande d'ouvrages sur un site web. Nous ajouterons ensuite des exigences non fonctionnelles (performances, ergonomie) et des contraintes de conception (sécurisation SSL) pour nous placer dans l'optique du démarrage d'un projet réel. Nous évoquerons enfin un processus de gestion des exigences et les outils associés.

SOMMAIRE

MOTS-CLÉS

- Expression des besoins
- Exigences
- Projet
- Vision
- Web

Choix du sujet

Le sujet de ce livre est la modélisation d'applications web. La librairie en ligne constitue un exemple concret, facile à comprendre et suffisamment représentatif de tels projets. Nous nous sommes inspirés des fonctionnalités de sites existants, comme www.amazon.fr, www.fnac.com, et bien sûr notre librairie préférée : www.eyrolles.com ! C'est donc le parti pris du livre, fournissant l'avantage de pouvoir nous raccrocher à des écrans et des règles de gestion concrètes, directement issues d'applications web opérationnelles (comme illustré sur les figures 2-1 et 2-2.)

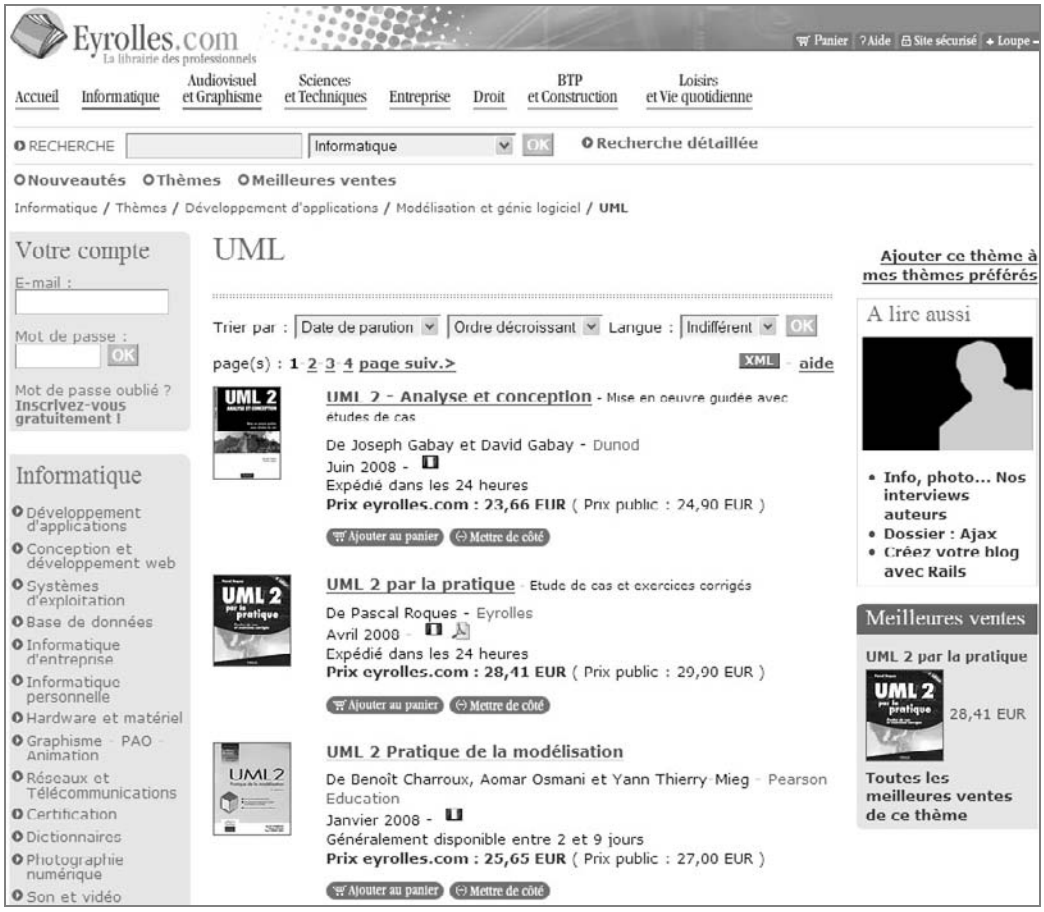


Figure 2–1
Exemple de page de librairie en ligne (eyrolles.com)

Autre point positif non négligeable : nous pourrons facilement montrer le lien entre la modélisation objet avec UML (sujet de ce livre) et l'implémentation, en nous appuyant sur des exemples réels. En outre, les solutions techniques réellement implantées utilisent aussi bien des langages objet purs et durs comme Java ou C# que des langages de scripts

Figure 2–2 Autre exemple de page de librairie en ligne (amazon.fr)

plus simples comme PHP. Elles permettent ainsi d'illustrer le fait que la modélisation UML n'implique pas forcément en aval la maîtrise d'un langage de programmation objet complexe.

LANGAGE **PHP**

Le site de la librairie Eyrolles a ainsi été développé en PHP. C'est un langage de script OpenSource disponible pour diverses plates-formes (Unix, Linux, Windows) comparable à ASP de Microsoft. Il prend en charge l'ensemble des protocoles du Web (HTTP, SMTP, LDAP, etc.) et offre un accès natif aux principales bases de données du marché. PHP offre toutes les fonctionnalités utiles pour construire des sites web dynamiques sophistiqués.

La venue de PHP 5 a amené de grandes nouveautés pour un outil qui se veut à double emploi : facile et utilisable pour des applications simples à destination d'un large public, performant et puissant pour des applications métier à destination d'un public professionnel. On ne parle plus alors uniquement de langage de programmation, mais de plate-forme à part entière.

Expression initiale des besoins

La société (fictive !) jeBouquine a décidé récemment de rejoindre les rangs des grands libraires francophones en ligne. Les rayons déjà ouverts sur le site web sont très divers : *Informatique*, *Sciences et techniques*, *Psychologie*, *Décoration* et *Jardinage*. La librairie jeBouquine assure également la distribution en langue anglaise d'une large sélection d'ouvrages des plus grands éditeurs anglais et américains. Par exemple, on trouve dans le rayon informatique des titres venant de chez Addison-Wesley, McGraw-Hill, O'Reilly, Wiley, Wrox Press, etc.

Il se trouve que depuis la parution de la première édition de ce livre, il existe un site http://www.je-bouquine.com/, du nom d'un magazine pour la jeunesse…

L'objectif fondamental du futur site www.jeBouquine.com est de permettre aux internautes de rechercher des ouvrages par thème, auteur, mot-clé, etc., de se constituer un panier virtuel, puis de pouvoir les commander et les payer directement sur le Web.

Vision du projet

L'objectif du premier document est de collecter, analyser et définir les besoins de haut niveau et les caractéristiques du futur site web marchand www.jeBouquine.com. Il se focalise sur les fonctionnalités requises par les utilisateurs, et sur la raison d'être de ces exigences. Le détail de la description des besoins se trouve dans les spécifications des cas d'utilisation (voir les chapitres 3 et 4).

Méthode Correspondance avec UP

Par rapport à la brève présentation du Processus Unifié que nous avons faite au chapitre précédent, la suite de ce chapitre correspond à une partie du travail effectué lors de la phase d'initialisation (inception) de UP.

Positionnement

www.jeBouquine.com se veut être le site web de la société jeBouquine, nouvelle venue dans le cercle des librairies en ligne d'origine française.

Le but du projet consiste à :

- Prendre place sur le marché de la librairie en ligne en face des concurrents généralistes tels que www.amazon.fr, www.fnac.com ou www.eyrolles.com ainsi que d'autres plus spécialisés comme www.infotheque.fr ou www.lmet.fr en informatique.
- Inventer rapidement des éléments différentiateurs pour devenir à moyen terme (moins de deux ans) le numéro un français de la vente de livres en ligne. Le site web devra donc être facilement évolutif pour pouvoir implémenter très rapidement de nouvelles fonctionnalités importantes.

Exigences fonctionnelles

Le site web de la société jeBouquine devra regrouper toutes les fonctionnalités nécessaires de recherche, de découverte détaillée, de sélection et de commande d'ouvrages.

Recherche

La première étape pour l'internaute consiste à trouver le plus rapidement possible un ouvrage recherché dans l'ensemble du catalogue. Les références de cet ouvrage pouvant être plus ou moins précises, il faut lui fournir plusieurs méthodes de recherche différentes. L'internaute pourra ainsi saisir un critère (titre, auteur, ISBN, etc.) ou même plusieurs critères à la fois (comme illustré sur la figure 2-3). Les résultats de la recherche seront disponibles sur une page particulière, et devront pouvoir être facilement parcourus et reclassés.

Figure 2–3
Exemple de formulaire et de résultat de recherche rapide

Toutefois, s'il n'a pas d'idée bien arrêtée, il faut également lui fournir le moyen de flâner comme il le ferait dans les rayons d'une vraie bibliothèque : pour cela, il pourra accéder directement à une classification thématique, aux nouveautés, aux meilleures ventes, etc.

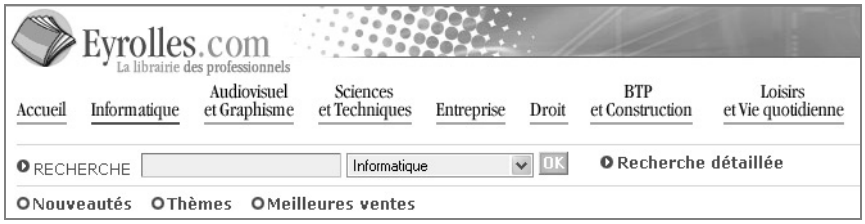


Figure 2–4
Exemple d'écran général de recherche

Découverte

Chaque livre vendu sur le site www.jeBouquine.com sera présenté en détail sur sa propre page (comme illustré sur la figure 2-5).

On y trouvera en particulier :

- une image (pour la majorité des ouvrages) que l'internaute pourra agrandir,
- son prix et sa disponibilité,
- des commentaires de lecteurs déjà clients,
- la table des matières détaillée, des extraits de chapitres, etc.

Figure 2–5
Exemple de fiche détaillée d'ouvrage

Sélection

Dans un véritable magasin, le client choisit ses articles les uns à la suite des autres, les dépose dans son panier, puis se rend à la caisse pour régler le tout. Les sites web marchands tentent de reproduire ces habitudes d'achat le plus fidèlement possible. Ainsi, lorsque l'internaute est intéressé par un ouvrage, il peut l'enregistrer dans un panier virtuel, comme indiqué sur l'exemple de la figure 2-3 (bouton *Ajouter au panier*).

Il doit pouvoir ensuite à tout moment en ajouter, en supprimer ou encore en modifier les quantités avant de passer commande (voir figure 2-6).

Figure 2–6
Exemple de panier virtuel

Commande

À tout moment, le client doit pouvoir accéder au formulaire du bon de commande, dans lequel il saisit ses coordonnées et les informations nécessaires au paiement et à la livraison (voir figure 2-7). Pour garantir la sécurisation et la confidentialité des échanges, il est impératif que l'envoi des données se fasse de manière cryptée. Dans le cas où le client le souhaiterait, le système doit être capable de lui imprimer un devis pour commander par fax ou par courrier.

Le client devra pouvoir ensuite suivre ses commandes récentes, et même les modifier avant expédition, de façon sécurisée, comme illustré sur la figure 2-8.

D'une manière générale, le client devra pouvoir gérer son compte, c'est-à-dire modifier ses coordonnées, ses préférences, ajouter des adresses, etc. (voir figure 2-9).

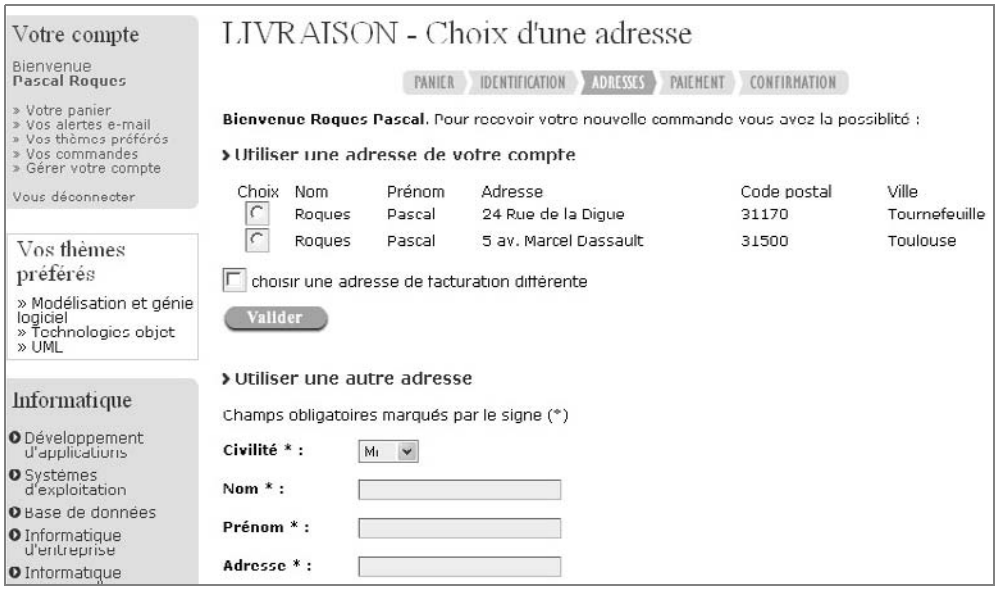


Figure 2–7 Exemple de bon de commande

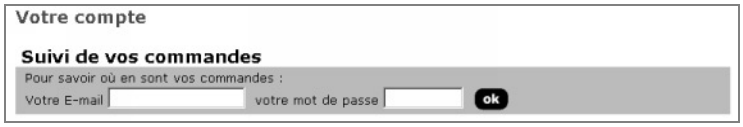


Figure 2–8 Exemple de formulaire de suivi de commandes

Figure 2–9 Exemple de formulaire de gestion de compte

Exigences non fonctionnelles

Exigences de qualité

Pour attirer un client sur un site marchand et ensuite le fidéliser, il est important de répondre aux exigences de qualité suivantes :

- **Ergonomie sobre et efficace**

  Acheter un livre sur le Web ne doit pas prendre beaucoup de temps ou demander une maîtrise de mathématiques ! La mise en page du site facilitera au maximum la démarche à l'aide d'une présentation claire et intuitive. Les sites trop riches et trop complexes n'incitent pas à l'achat, car ils demandent un effort intellectuel important et non souhaité.

- **Formulaire de commande simple**

  Très souvent, l'internaute cale au moment d'acheter, car l'effort le plus important à fournir est le renseignement du bon de commande ! La conception et la présentation de celui-ci seront donc particulièrement soignées pour ne pas rebuter le client.

- **Aide en ligne puissante**

  À tout moment, l'internaute peut consulter des pages d'aide contextuelle, ainsi que lancer une recherche dans l'ensemble des pages d'aide (voir la figure 2-10). Une visite guidée sera également proposée aux nouveaux visiteurs.

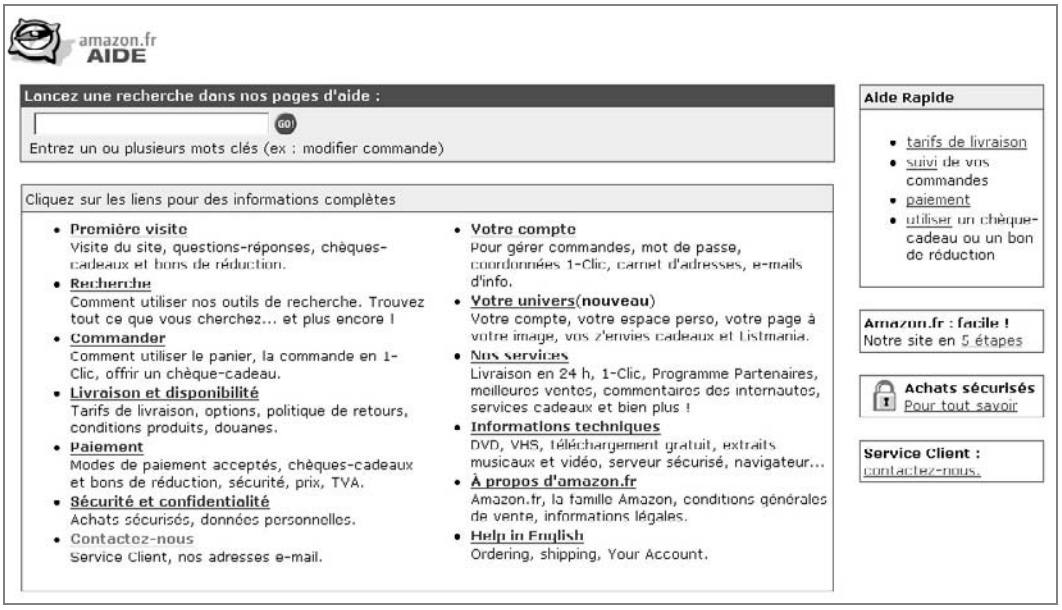


Figure 2–10
Exemple de page d'aide sophistiquée

Exigences de performance

N'oublions pas non plus les exigences quantitatives suivantes, très importantes également pour les utilisateurs :

- La librairie jeBouquine doit pouvoir gérer les comptes de plus de 10 000 clients.
- Le site web doit supporter plus de 1 000 connexions simultanées.
- Le catalogue d'ouvrages doit pouvoir comprendre plus de 1 000 000 titres.
- Aucune recherche ne doit prendre plus de 2 secondes.

Contraintes de conception

Mise à jour des données de référence

Les informations relatives aux ouvrages présentés sur le site proviendront essentiellement de deux sources complémentaires. La première servira à alimenter la base avec tous les nouveaux ouvrages, la seconde à mettre à jour les données qui concernent le prix et l'état du stock des livres du catalogue.

Ces deux sources externes seront automatiquement chargées dans la base de données de façon périodique. Toutes les autres informations seront saisies manuellement à l'aide d'une petite application intranet dédiée à l'enrichissement des données relatives aux ouvrages.

> **SGBDR Oracle**
>
> Le grand volume d'informations sur les livres, les commandes, etc., doit être sauvegardé de façon fiable et durable. Un SGBDR est incontournable et Oracle en est le leader incontestable sur le marché. Oracle supporte un très gros volume de transactions tout en étant extrêmement robuste. Des statistiques récentes montrent que près d'un cinquième des sites d'e-commerce passe à Oracle au bout de 2 à 3 ans, en particulier pour des problèmes de montée en charge mal évaluée initialement.

Mise à jour depuis les formulaires du site

Les données saisies depuis le site web et enregistrées dans la base décriront les coordonnées des clients, ainsi que les caractéristiques de leurs commandes.

Les coordonnées des clients seront mémorisées. Dans un premier temps, elles permettront l'envoi du colis correspondant à la commande. Dans un second temps, cela épargnera de les saisir de nouveau lors des prochaines commandes.

Toutes les données personnelles seront bien sûr protégées et leur confidentialité sera garantie.

Les commandes seront enregistrées, puis traitées ultérieurement par le service clientèle. Le client pourra consulter l'historique de toutes ses commandes.

Panier

Le panier de l'internaute ne sera pas sauvegardé dans la base de données. Sa durée de vie n'excèdera pas celle de la visite de l'utilisateur.

Paiement sécurisé

La saisie du numéro de carte de crédit par le client devra s'effectuer de manière sécurisée, en cryptant le transfert HTTP, via le protocole SSL.

La commande et le numéro de carte seront stockés dans la base, jusqu'au traitement de la commande. La banque concernée validera la transaction. À cette étape, le numéro de la carte de crédit sera supprimé de la base de données.

B.A.-BA SSL

Le système de chiffrement SSL (*Secure Socket Layer*) permet de sécuriser le protocole HTTP (HTTPS), avec un chiffrement sur 128 bits. Il s'agit de la norme de sécurité la plus répandue et la plus fiable actuellement sur Internet. Le serveur HTTP le plus utilisé dans le monde est Apache. Il permet d'utiliser SSL et offre l'avantage d'être Open-Source.

Gestion des exigences

La gestion des exigences est l'ensemble des activités permettant de définir et de suivre les exigences d'un système au cours d'un projet. Elle permet de :

- s'assurer de la cohérence entre ce que fait réellement le projet et ce qu'il doit faire ;
- faciliter l'analyse d'impact en cas de changement.

Les principaux outils de gestion des exigences sont : DOORS (IBM - Telelogic), RequisitePro (IBM - Rational), et CaliberRM (Borland).

Figure 2–11
Création des exigences dans Enterprise Architect

L'outil UML 2 que nous avons utilisé dans cet ouvrage s'appelle Enterprise Architect, de la société Sparx Systems. Un petit clin d'œil à cette firme australienne qui nous a aimablement fourni une licence gratuite complète.

▸ www.sparxsystems.com

Nous allons profiter d'une capacité particulière de l'outil EA, à savoir la possibilité de décrire les exigences comme des éléments de modélisation à part entière. Dans un premier temps, nous ne dessinerons pas de diagramme, mais créerons simplement des exigences dans un dossier particulier, lui-même structuré en sous-dossiers. Les exigences ont au moins un type, un identifiant et un texte descriptif, comme indiqué sur la figure suivante.

Sur les gros projets informatiques, des outils dédiés sont de plus en plus considérés comme indispensables et utilisés par les chefs de projet modernes. Ces outils fournissent les capacités de :

- capturer les exigences, souvent via des fonctions d'import de documents textuels ;
- les administrer, les gérer et les classer en définissant des attributs adaptés au projet et des valeurs possibles pertinentes pour ces attributs ;
- construire des liens entre les exigences et les différents livrables du projet : c'est le concept majeur de *traçabilité* ;
- générer automatiquement des documents (par exemple, des matrices de traçabilité).

Notre processus léger de modélisation ne nous empêche pas de montrer ce qu'un outil de ce type pourrait apporter concrètement au projet de création d'un site marchand.

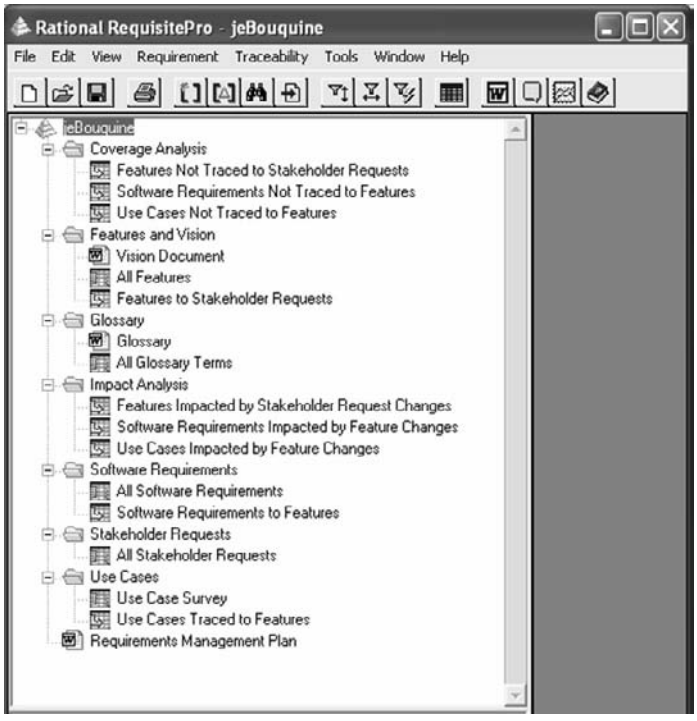


Figure 2–12
Création de projet sous RequisitePro

Pour cela, nous allons utiliser un des outils majeurs du marché : RequisitePro d'IBM – Rational. Commençons par créer un projet en utilisant le template fourni *Composite*, qui va nous permettre de mêler cas d'utilisation et exigences textuelles (figure 2–12).

Nous allons ensuite créer les exigences une par une, en commençant par `Recherche1` avec le texte suivant : « L'internaute pourra trouver le plus rapidement possible un ouvrage recherché dans l'ensemble du catalogue ». Positionnons la valeur de l'attribut `Priority` à `High` et celle de `Status` à `Approved`. Il est en effet très important de classer les exigences suivant plusieurs critères tels que la priorité fonctionnelle, la difficulté technique, la stabilité prévisible, etc.

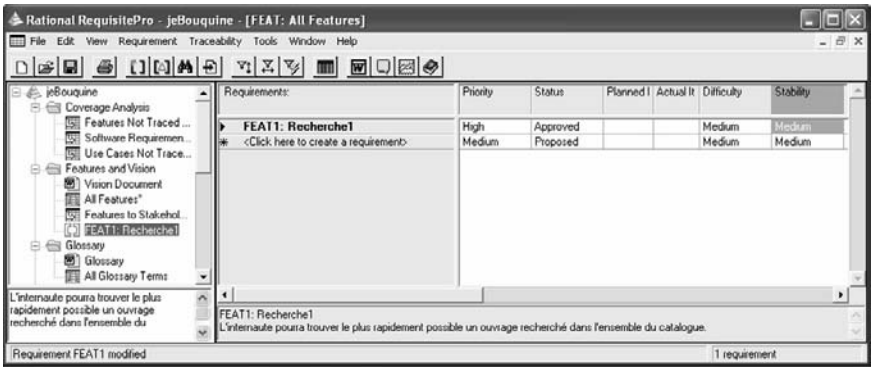


Figure 2–13
Création d'exigences sous RequisitePro

Continuons ainsi à saisir plusieurs exigences :

- `Recherche2` : l'internaute pourra saisir un critère (titre, auteur, ISBN, etc.) ou même plusieurs critères à la fois.
- `Panier1` : lorsque l'internaute est intéressé par un ouvrage, il peut l'enregistrer dans un panier virtuel.
- `Panier2` : il doit pouvoir ensuite à tout moment en ajouter, en supprimer ou encore en modifier les quantités.
- `Commande1` : à tout moment, le client doit pouvoir accéder au formulaire du bon de commande.
- `Commande2` : pour garantir la sécurisation et la confidentialité des échanges, il est impératif que l'envoi des données se fasse de manière cryptée.

Ajoutons également plusieurs `Stakeholder Requests` : ergonomie sobre et efficace, aide en ligne puissante, transactions sécurisées.

Nous allons maintenant sélectionner l'exigence non fonctionnelle `Performances4` et la relier à l'exigence fonctionnelle `Recherche1`. Faisons de même pour relier d'autres exigences à la `Stakeholder Request STRQ1`.

Stakeholder Requests

Ce sont les demandes des parties prenantes, à savoir les besoins de haut niveau à partir desquels on peut décliner de nombreuses exigences détaillées.

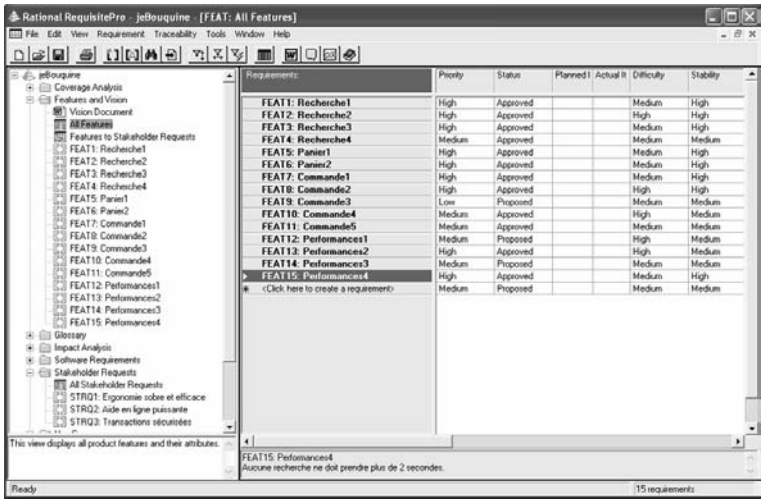
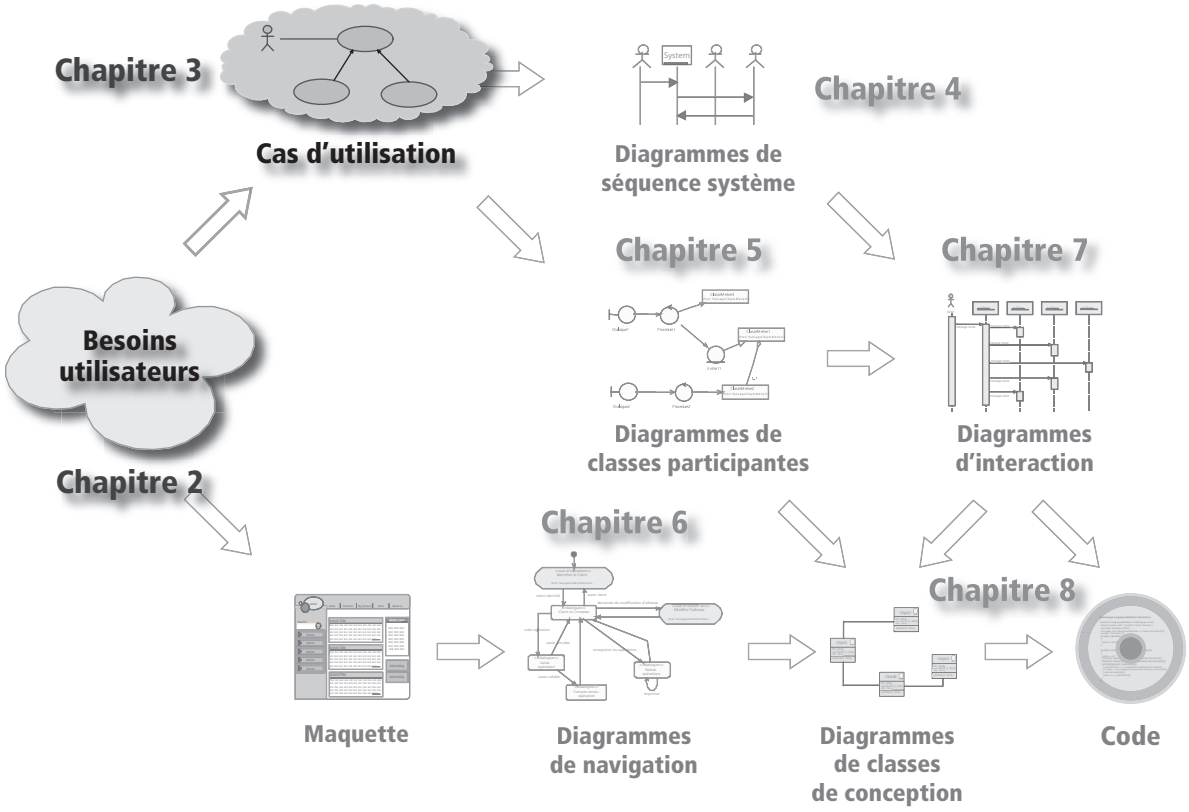


Figure 2–14
Exemple d'exigences pour le projet jeBouquine

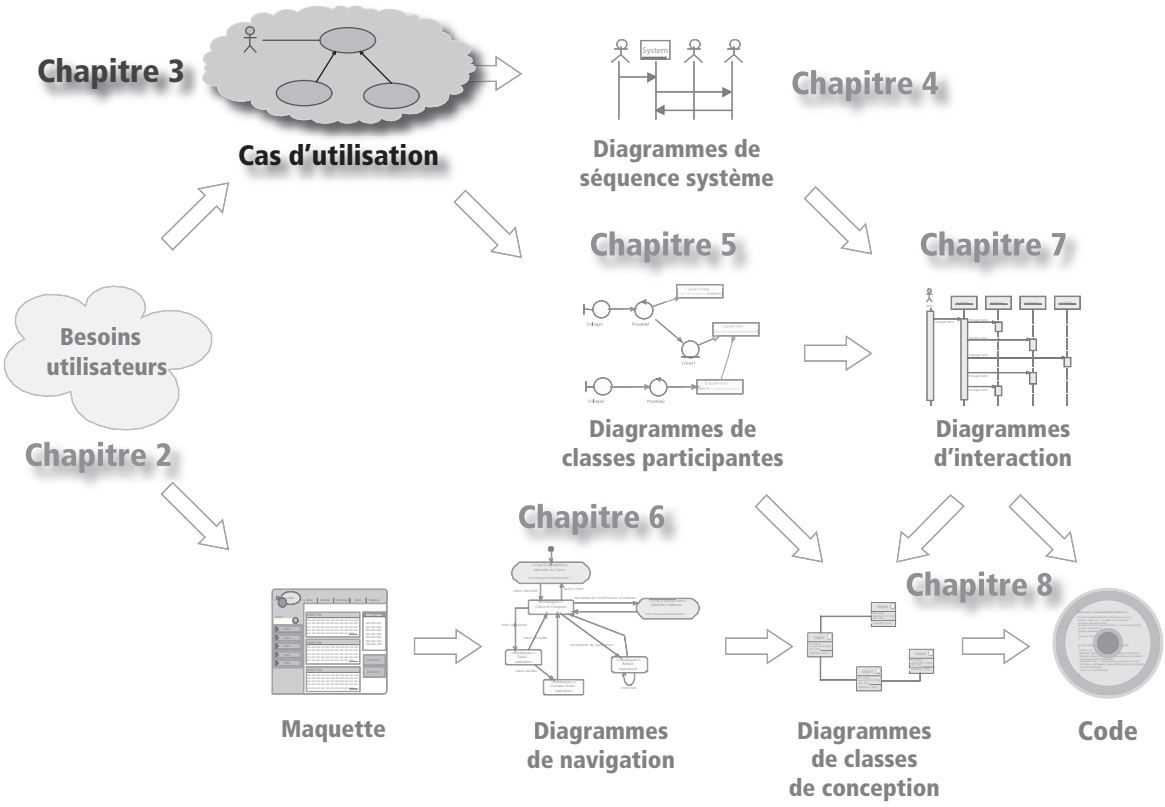
L'outil nous permet alors de consulter et d'éditer une matrice de traçabilité dans Coverage Analysis. Sur la figure suivante, nous voyons par exemple clairement qu'il manque des relations entre les exigences et la STRQ2.

Figure 2–15
Matrice de traçabilité
entre Features et Stakeholder Requests

Nous poursuivrons cette étude de la gestion des exigences outillée à la fin du chapitre 3.


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Besoins utilisateurs
Chapitre 2
Chapitre 5
Diagrammes de classes participantes
Chapitre 7
Diagrammes d'interaction
Chapitre 6
Maquette
Diagrammes de navigation
Chapitre 8
Diagrammes de classes de conception
Code

chapitre 3


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Chapitre 7
Besoins utilisateurs
Chapitre 2
Diagrammes de classes participantes
Diagrammes d'interaction
Chapitre 6
Chapitre 8
Maquette
Diagrammes de navigation
Diagrammes de classes de conception
Code

Spécification des exigences d'après les cas d'utilisation

Acteurs et cas d'utilisation sont les concepts UML fondamentaux pour la spécification des exigences. Nous apprendrons à les identifier à partir de l'expression initiale des besoins de notre étude de cas. Nous verrons ensuite comment structurer, relier et classer ces cas d'utilisation ainsi que les représentations graphiques UML associées. Nous aborderons enfin l'impact de cette étude sur la planification du projet et les bénéfices qu'en tire le chef de projet, ainsi que la problématique de la traçabilité avec les exigences textuelles.

SOMMAIRE

MOTS-CLÉS

- Acteur
- Cas d'utilisation
- Package
- Planification
- Itération
- Traçabilité

Démarche

Rappelons comment cette activité de spécification des exigences se situe par rapport à l'ensemble du processus décrit au chapitre 1. L'expression préliminaire des besoins donne lieu à une modélisation par les cas d'utilisation et à une maquette d'interface homme-machine (IHM), comme indiqué sur la figure 3-1.

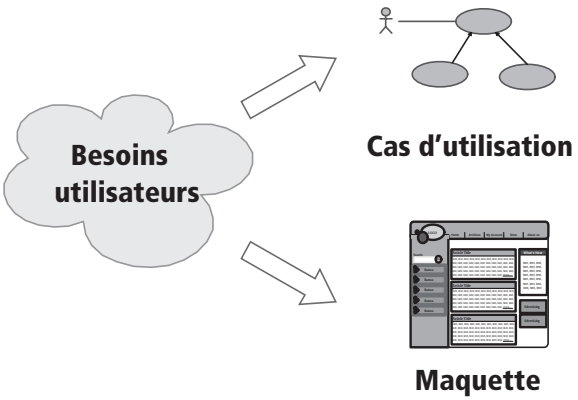


Figure 3–1
Les besoins donnent lieu à des cas d'utilisation et à une maquette.

Dans ce chapitre, nous allons détailler la branche supérieure concernant la modélisation des cas d'utilisation. La réalisation d'une maquette graphique est une activité courante mettant en jeu des outils de dessin. Elle montre rapidement l'aspect visuel (le « look ») du site web. De nombreux ouvrages traitent déjà de ce sujet particulier qui est en dehors de la portée de ce livre.

Nous allons au contraire nous concentrer sur ce qu'il y a « derrière » la maquette, c'est-à-dire : quelles sont les informations à montrer, à qui et pour quoi faire ? Quelles sont les informations à montrer ? C'est ce que nous décrirons dans le chapitre 5 traitant des classes d'analyse. À qui et pour quoi faire ? C'est précisément ici que les concepts UML d'acteurs et de cas d'utilisation interviennent.

Détaillons un peu plus les différentes étapes de la démarche que nous allons adopter afin d'aboutir au modèle des cas d'utilisation (voir la figure 3-2) :

- identifier les acteurs,
- identifier les cas d'utilisation,
- structurer les cas d'utilisation en packages,
- ajouter les relations entre cas d'utilisation,
- finaliser un ou plusieurs diagramme(s) de cas d'utilisation par package.

RÉFÉRENCES Design graphique de site web

- *Réussir un projet de site web*, N. Chu, Eyrolles, 2008
- *Réussir son site web avec XHTML et CSS*, M. Nebra, Eyrolles, 2008
- *Sites web – Les bonnes pratiques*, E. Sloïm, Eyrolles, 2007
- *CSS 2 – Pratique du design web*, R. Goetter, Eyrolles, 2007
- *Ergonomie web*, A. Boucher, Eyrolles, 2007
- *Je crée mon site Internet avec Dreamweaver 8 et Flash 8*, C. Bergé, Eyrolles, 2006
- *Design web : utiliser les standards – CSS et XHTML,* J. Zeldman, Eyrolles, 2006
- *Création et optimisation de sites web – CSS2, RSS, XHTML et XML,* Collectif Campus Press, 2006
- *Ergonomie du logiciel et design web,* J.F. Nogier, Dunod, 2005

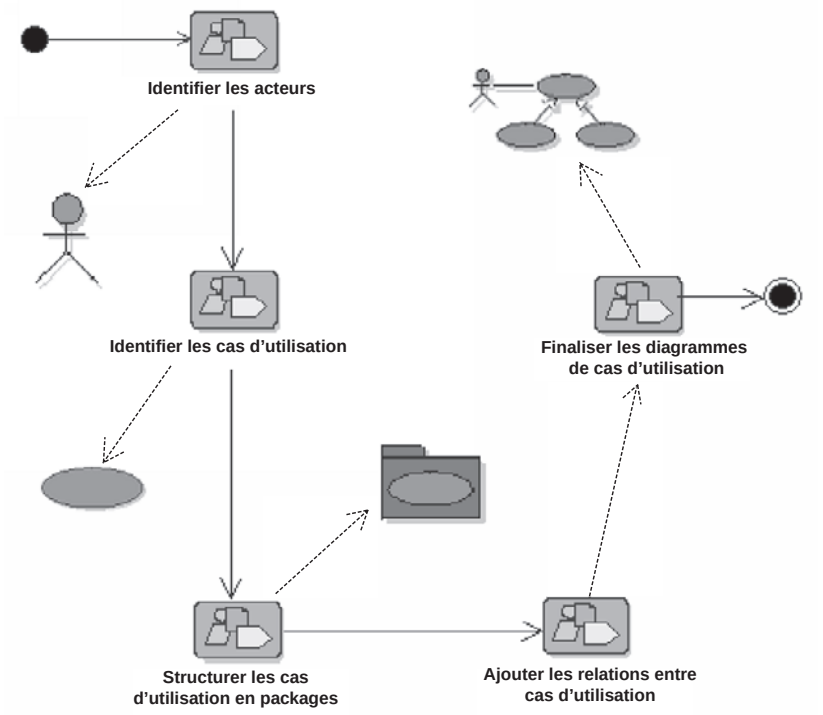


Figure 3–2
Synoptique de la démarche de construction du modèle des cas d'utilisation

Le modèle ainsi obtenu permet d'établir les priorités entre les cas d'utilisation afin d'aider le chef de projet à planifier ses itérations en connaissance de cause.

Identification des acteurs

Les acteurs humains pour le site web www.jeBouquine.com sont les suivants :

- L'`Internaute` : la personne qui visite le site pour rechercher des ouvrages et éventuellement passer une commande. Il s'agit bien sûr de l'acteur le plus important, celui pour lequel le site existe !
- Le `Webmaster` : rôle des employés qui sont en charge du bon fonctionnement et de la maintenance du site web.
- Le `Service clients` : rôle des employés qui s'occupent du suivi des commandes des clients.
- Le `Libraire` : rôle des employés qui sont responsables du contenu rédactionnel du site.

Nous allons également prendre en compte les systèmes informatiques connectés au site web, à savoir le système `Nouveautés` qui alimente la base avec tous les nouveaux ouvrages, et le système `Gestion des stocks`

B.A.-BA **Acteur**

Un acteur représente un rôle joué par une entité externe (utilisateur humain, dispositif matériel ou autre système) qui interagit directement avec le système étudié.
Un acteur peut consulter et/ou modifier directement l'état du système, en émettant et/ou en recevant des messages susceptibles d'être porteurs de données.

Attention
Acteurs logiques vs acteurs physiques

Éliminez autant que faire se peut les acteurs « physiques » au profit des acteurs « logiques » : l'acteur est celui qui bénéficie de l'utilisation du système. Cette règle permet en particulier de s'affranchir des technologies d'interface qui risquent d'évoluer au fil du projet.

qui met à jour les données concernant le prix et l'état du stock des livres du catalogue. Ces deux sources externes seront automatiquement chargées dans la base de données de façon périodique.

L'ensemble des acteurs est représenté graphiquement sur la figure 3-3 autour d'un rectangle figurant le système à l'étude. La représentation graphique standard de l'acteur en UML est l'icône appelée *stick man*, avec le nom de l'acteur sous le dessin. On peut également figurer un acteur sous la forme rectangulaire d'une classe, avec le mot-clé `<<actor>>`. Une bonne recommandation consiste à faire prévaloir l'utilisation de la forme graphique du stick man pour les acteurs humains et une représentation rectangulaire pour les systèmes connectés.

ATTENTION Rôle vs entité concrète

Ne confondez pas rôle et entité concrète. Une même entité concrète peut jouer successivement différents rôles par rapport au système étudié, et être modélisée par plusieurs acteurs. Réciproquement, le même rôle peut être tenu simultanément par plusieurs entités concrètes, qui seront alors modélisées par le même acteur. Par conséquent, même si une seule personne physique peut jouer successivement les rôles de `Libraire` et `Webmaster` vis à vis du site web, il s'agit bien de deux acteurs distincts, de deux profils différents.

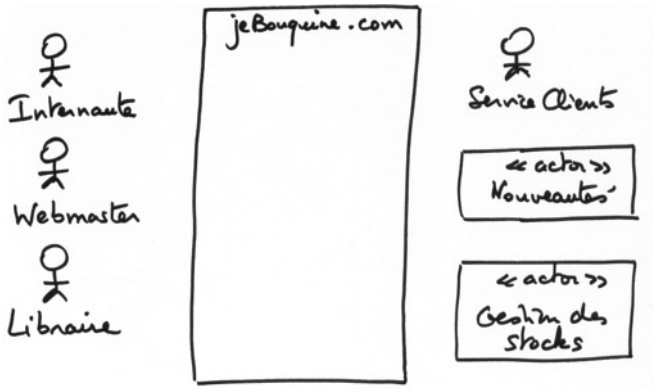


Figure 3–3
Acteurs du site jeBouquine.com

Identification des cas d'utilisation

Pour chaque acteur identifié précédemment, il convient de rechercher les différentes intentions « métier » selon lesquelles il utilise le système.

Commençons par l'acteur le plus important pour un site d'e-commerce : l'`Internaute`.

B.A.-BA Cas d'utilisation

Un cas d'utilisation (*use case*) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur particulier.
Un cas d'utilisation modélise un service rendu par le système. Il exprime les interactions acteurs/système et apporte une valeur ajoutée « notable » à l'acteur concerné.

ATTENTION Cas d'utilisation = ensemble de séquences d'actions

Une erreur fréquente concernant les cas d'utilisation consiste à vouloir descendre trop bas en termes de granularité. Un cas d'utilisation représente un ensemble de séquences d'actions réalisées par le système, et le lien entre ces séquences d'actions est précisément l'objectif métier de l'acteur. Le cas d'utilisation ne doit donc pas se réduire systématiquement à une seule séquence, et encore moins à une simple action.

Ces cas d'utilisation principaux ont été bien mis en évidence par l'expression de besoins préliminaire du chapitre précédent, à savoir :

- rechercher des ouvrages,
- gérer son panier,
- effectuer une commande.

On obtient un diagramme préliminaire (voir figure 3-4) en représentant sur un schéma les cas d'utilisation (ovales) reliés par des associations (lignes) à leurs acteurs. Un cas d'utilisation doit être relié à au moins un acteur.

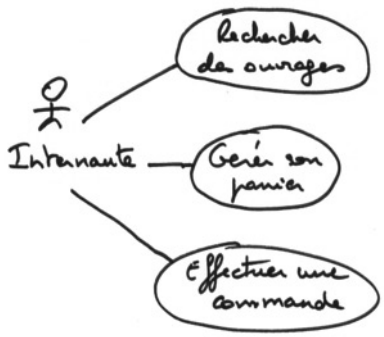


Figure 3–4
Cas d'utilisation principaux de l'internaute

L'internaute peut également consulter l'historique de ses commandes en cours.

Il faut bien comprendre que chaque cas d'utilisation doit avoir un objectif en soi et pouvoir être réalisé indépendamment des autres. Un internaute visite quelquefois le site dans le seul but de chercher des ouvrages, sans intention d'acheter. Dans d'autres cas, il gère un panier virtuel pour faire une simulation, ou obtenir un devis.

Il peut également se connecter pour surveiller l'état de sa dernière commande. Tous ces objectifs sont bien indépendants et auto-suffisants : il s'agit de vrais cas d'utilisation ! L'ensemble des cas d'utilisation de l'internaute est représenté sur la figure 3-5.

CONCEPT AVANCÉ Flèche sur l'association

L'utilisation de la flèche sur l'association entre le cas d'utilisation `Effectuer une commande` et l'acteur `Service clients` signale un sens unique de transmission d'information. Cet acteur ne fait que recevoir des messages du système, sans lui en envoyer, dans le cadre de ce cas d'utilisation. En revanche, dans le cadre du cas d'utilisation `Consulter ses commandes`, l'acteur `Service clients` va interagir dans les deux sens avec le système. Remarquez également qu'un cas d'utilisation peut faire participer plusieurs acteurs et qu'un acteur peut participer à plusieurs cas d'utilisation.

MÉTHODE Nom des cas d'utilisation

Nommez les cas d'utilisation par un verbe à l'infinitif suivi d'un complément, du point de vue de l'acteur (et non pas du point de vue du système).

B.A.-BA Association

Une association est une relation entre éléments UML (classes, cas d'utilisation, etc.) qui décrit un ensemble de liens. Elle est utilisée dans le cadre du diagramme de cas d'utilisation pour relier les acteurs et les cas d'utilisation par une relation qui signifie simplement : « participe à ».

ATTENTION Taille des cas d'utilisation

Obtenir un devis est trop « petit » pour être un cas d'utilisation ! Rappelez-vous qu'un cas d'utilisation est un ensemble de séquences d'actions : sûrement pas une seule action.

B.A.-BA Acteur principal

Contrairement à ce que l'on pourrait croire, tous les acteurs n'utilisent pas forcément le système ! Nous appelons acteur principal celui pour qui le cas d'utilisation produit un résultat observable. Par opposition, nous qualifions d'acteurs secondaires les autres participants du cas d'utilisation. Les acteurs secondaires sont souvent sollicités pour des informations complémentaires ; ils peuvent uniquement consulter ou informer le système lors de l'exécution du cas d'utilisation. Une bonne pratique consiste à faire figurer les acteurs principaux à gauche des cas d'utilisation et les acteurs secondaires à droite.

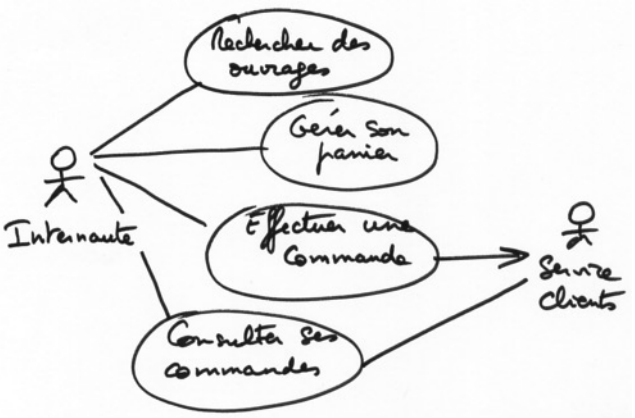


Figure 3–5
Cas d'utilisation de l'internaute

Quels sont les cas d'utilisation des employés de jeBouquine ? D'après le chapitre 2, on identifie :

- maintenir le catalogue, qui fait intervenir les deux systèmes `Nouveautés` et `Gestion des stocks`,
- maintenir les informations éditoriales,
- maintenir le site.

Représentons ces cas sur un diagramme avec leurs acteurs associés (voir figure 3-6).

CONCEPT AVANCÉ Flèche sur l'association

Notez cette fois-ci les flèches de navigabilité vers le cas d'utilisation `Maintenir le catalogue` : les acteurs non humains ne font qu'envoyer des messages au système, sans jamais en recevoir. Ce sont de parfaits exemples d'acteurs secondaires.

MÉTHODE modélisation agile

Notez que nos premiers diagrammes ont été faits « à la main », afin de montrer qu'il n'est pas indispensable d'acheter un progiciel coûteux pour utiliser UML ! C'est aussi cela la modélisation « agile ».

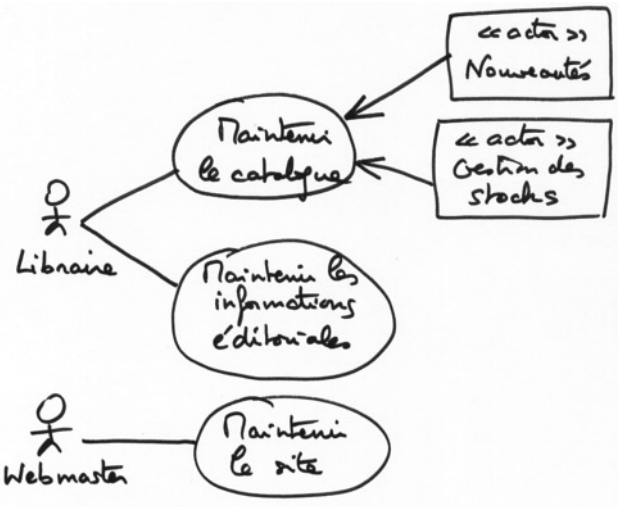


Figure 3–6
Cas d'utilisation des employés

Structuration en packages

Pour améliorer notre modèle, nous allons organiser les cas d'utilisation et les regrouper en ensembles fonctionnels cohérents. Pour ce faire, nous utilisons le concept général d'UML, le *package*.

Les acteurs ont également été regroupés dans un package séparé sur lequel s'appuient les deux packages de cas d'utilisation. L'organisation générale du modèle dans un outil de modélisation est représentée sur la figure 3-7.

Le sigle `UC` pour *use case* est souvent utilisé pour raccourcir les noms de packages.

B.A.-BA **Package**

Le package est un mécanisme général de regroupement d'éléments en UML, qui peut être utilisé par exemple pour regrouper des cas d'utilisation, mais aussi des acteurs, des classes, etc.

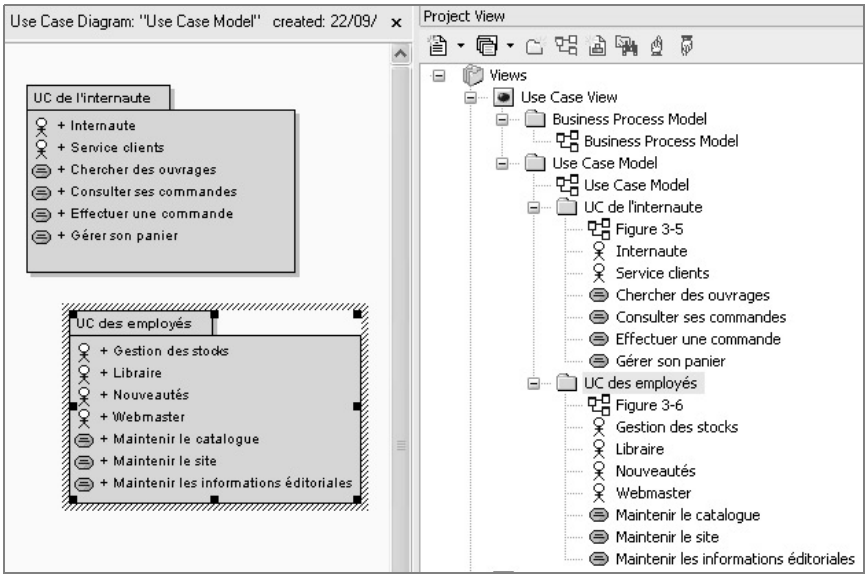


Figure 3–7
Organisation des cas d'utilisation et des acteurs en packages (avec l'outil Enterprise Architect)

Affinement du modèle de cas d'utilisation

Après une nouvelle réunion d'expression des besoins avec le responsable du projet, nous sommes arrivés à la conclusion qu'il était nécessaire de distinguer deux profils d'internautes :

- le `Visiteur`, inconnu du site web, mais qui peut néanmoins rechercher des ouvrages et gérer un panier ;
- le `Client`, déjà connu par le site web, qui peut seul effectuer une commande et en suivre l'état.

À tout moment, le visiteur peut choisir de créer un compte, afin de devenir client. Le client a également la possibilité de modifier les informations le concernant (adresse de facturation, adresses de livraison, adresse électronique, etc.) stockées par le site web.

Nous avions également oublié de mentionner l'important système externe de `Paiement sécurisé`, nécessaire au moment du paiement en ligne.

Le diagramme de cas d'utilisation des internautes devient donc tel que représenté sur la figure 3-8.

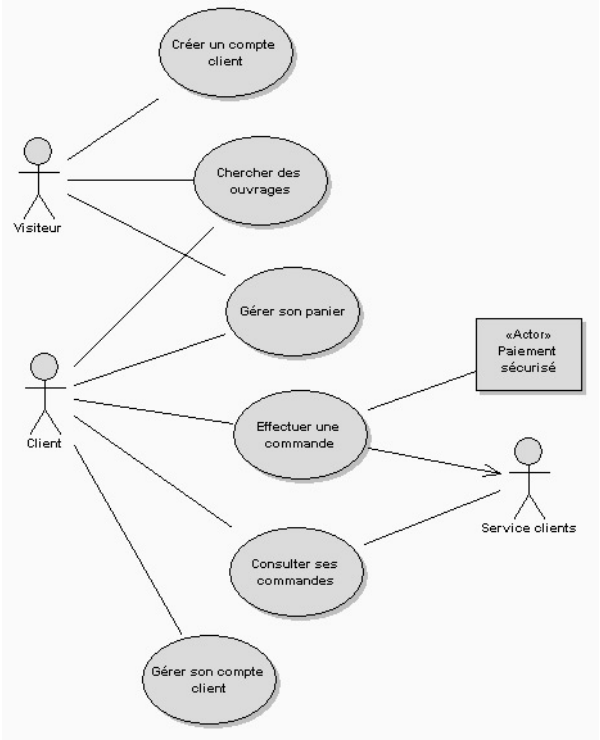


Figure 3–8
Cas d'utilisation du Visiteur et du Client

B.A.-BA Acteur généralisé

Il arrive que deux acteurs, ou plus, présentent des similitudes dans leurs relations aux cas d'utilisation. On peut l'exprimer en créant un acteur généralisé, éventuellement abstrait, qui modélise les aspects communs aux différents acteurs concrets. Dans notre exemple, l'acteur `Internaute` est la généralisation abstraite des rôles `Visiteur` et `Client`. Notez que le nom d'un acteur abstrait s'écrit en italique.

Nous remarquons maintenant que les deux acteurs `Client` et `Visiteur` partagent deux cas d'utilisation : ils sont également acteurs principaux de `Chercher des ouvrages` et `Gérer son panier`. Or, une bonne pratique consiste à identifier un seul acteur principal par cas d'utilisation. UML nous fournit la solution en permettant de créer un acteur généralisé `Internaute`, dont `Client` et `Visiteur` seront des spécialisations.

Le diagramme devient alors celui de la figure 3-9, avec un seul acteur principal par cas d'utilisation.

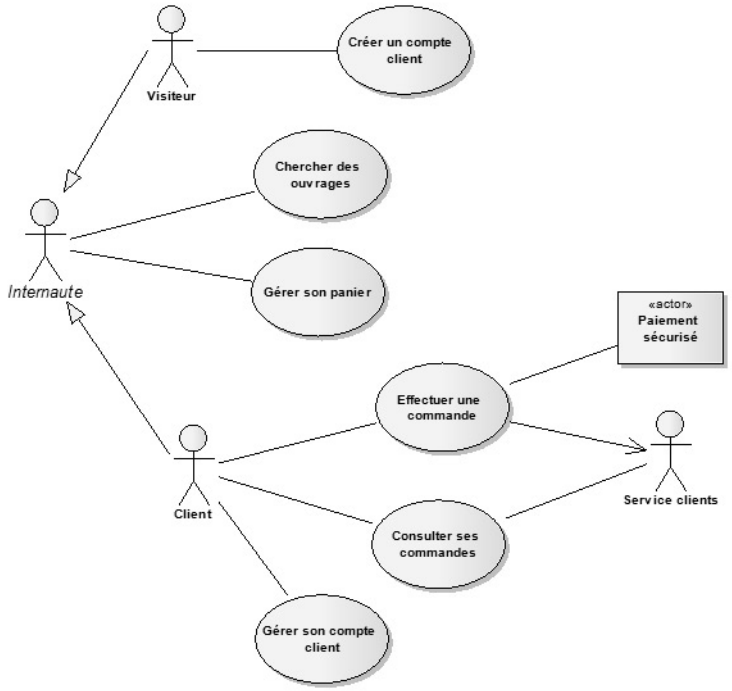


Figure 3–9
Cas d'utilisation des internautes

On pourrait relier les cas d'utilisation des internautes par des relations d'extension :

- La recherche d'ouvrages peut donner lieu à leur mise dans le panier (et réciproquement !).
- La gestion du panier peut donner lieu au passage d'une commande.
- Le passage d'une commande peut donner lieu à la gestion du compte client (ajout d'une adresse, etc.).

De même, les différentes possibilités de recherche d'ouvrages seront modélisées plus finement par une relation de généralisation/spécialisation.

Enfin, l'authentification du client est nécessaire au début du passage d'une commande, du suivi des commandes, ou de la modification des informations du compte.

Toutes ces relations entre cas d'utilisation, légales en UML, mais souvent mal utilisées et sources de confusion pour les experts métier, sont illustrées sur la figure 3-10.

Pensez-vous vraiment que le diagramme ainsi obtenu soit satisfaisant ? Non, même s'il est légal en UML, ce schéma est devenu maintenant trop complexe par rapport à l'intérêt de l'information ajoutée. Nous vous mettons formellement en garde contre l'usage abusif des relations entre cas d'utilisation !

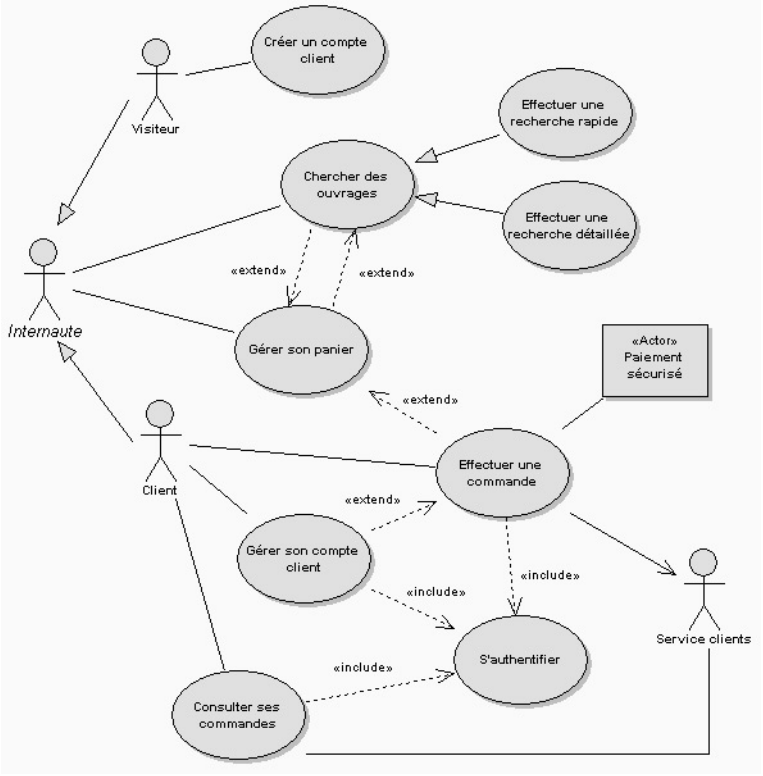


Figure 3–10
Relations entre cas d'utilisation des internautes

ATTENTION **N'abusez pas des relations entre cas d'utilisation !**

N'abusez pas des relations entre cas d'utilisation (inclusion, extension, généralisation) : elles peuvent rendre les diagrammes de cas d'utilisation trop difficiles à décrypter pour les experts métier qui sont censés les valider.

CONCEPT AVANCÉ **Relations entre cas d'utilisation**

Pour affiner le diagramme de cas d'utilisation, UML définit trois types de relations standardisées entre cas d'utilisation :

- Une **relation d'inclusion**, formalisée par le mot-clé `<<include>>` : le cas d'utilisation de base en incorpore explicitement un autre, de façon obligatoire.
- Une **relation d'extension**, formalisée par le mot-clé `<<extend>>` : le cas d'utilisation de base en incorpore implicitement un autre, de façon optionnelle.
- Une **relation de généralisation/spécialisation** : les cas d'utilisation descendants héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des interactions spécifiques supplémentaires.

CONCEPT AVANCÉ **Types de relations**

Notez les trois différents types de relations présentes sur le diagramme 3-10 :

- l'**association** (trait plein avec ou sans flèche, comme sur la figure 3-8) entre acteurs et cas d'utilisation ;
- la **dépendance** (flèche pointillée) entre cas d'utilisation, avec les mots-clés «`extend`» ou «`include`» ;
- la **relation de généralisation** (flèche fermée vide) entre cas d'utilisation.

Trop de projets en ont souffert, avec une incompréhension légitime de la part des experts métier à qui l'on demande de valider ces diagrammes. La figure 3-9 nous paraît donc largement suffisante pour exprimer les besoins fonctionnels majeurs de l'application web.

Néanmoins, le cas d'utilisation `S'authentifier` est à conserver, puisqu'il devra être réalisé afin de permettre au `Client` d'exécuter ses propres cas d'utilisation majeurs. Nous qualifierons donc ce petit cas d'utilisation de « fragment » : il ne représente pas un objectif à part entière du client, mais plutôt un objectif de niveau intermédiaire.

Avons-nous maintenant identifié tous les cas d'utilisation ? En fait, non. Nous avons omis un cas d'utilisation un peu particulier de l'internaute : `Consulter l'aide en ligne`. Certes, il ne s'agit pas non plus d'un cas d'utilisation majeur, mais il donnera lieu à des développements importants au niveau du projet informatique ! Il ne faut donc pas l'oublier, même si nous le qualifions de « secondaire » par rapport aux autres cas identifiés précédemment.

Pour bien structurer notre expression de besoin, nous préférons isoler les cas d'utilisation de second rang (`S'authentifier` et `Consulter l'aide en ligne`) dans un package à part, intitulé `UC de second rang`. L'organisation du modèle devient alors celle indiquée sur la figure 3-11.

B.A.-BA Stéréotype

Les mots-clés UML comme «`include`» et «`extend`» sont notés entre guillemets typographiques. Toutefois, nous pouvons également créer nos propres mots-clés, tels que «`fragment`» (pour indiquer qu'un cas d'utilisation n'est qu'un fragment factorisé d'autres cas d'utilisation) ou «`secondaire`» (pour indiquer qu'un cas d'utilisation est moins important que les autres). Ces mots-clés inventés par les utilisateurs s'appellent alors des stéréotypes.

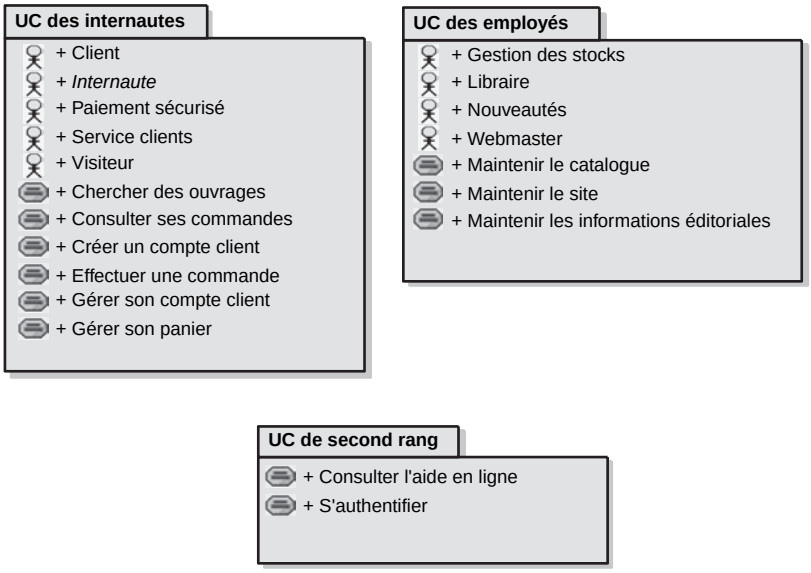


Figure 3–11
Organisation complétée des cas d'utilisation et des acteurs

Le diagramme de cas d'utilisation du nouveau package `UC de second rang` est donné sur la figure 3-12.

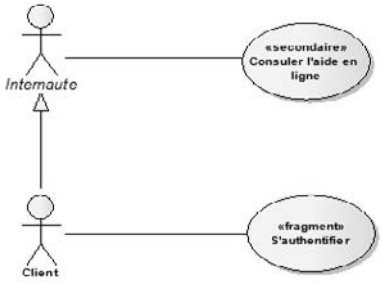


Figure 3–12
Cas d'utilisation de second rang

Classement des cas d'utilisation

Après tout ce travail d'identification des cas d'utilisation, nous pouvons maintenant les classifier en tenant compte des deux facteurs suivants :

1. la priorité fonctionnelle, déterminée par le service Marketing de jeBouquine ;
2. le risque technique, estimé par le chef de projet.

Le tableau suivant illustre cette démarche sur notre étude de cas.

Tableau 3–1 Classement des cas d'utilisation

Cas d'utilisation	Priorité	Risque
Chercher des ouvrages	Haute	Moyen
Gérer son panier	Haute	Bas
Effectuer une commande	Moyenne	Haut
Créer un compte client	Haute	Bas
Consulter ses commandes	Basse	Moyen
Consulter l'aide en ligne	Basse	Bas
Gérer son compte client	Moyenne	Bas
Maintenir le catalogue	Haute	Haut
Maintenir les informations éditoriales	Moyenne	Bas
Maintenir le site	Moyenne	Bas

Nous avions demandé expressément au service Marketing de ne pas classer tous les cas d'utilisation en priorité haute, mais de vraiment faire un effort pour distinguer trois niveaux d'importance. Effectuer une commande est ainsi en priorité moyenne, car l'internaute peut toujours imprimer son devis et commander ensuite par courrier ou fax en envoyant son paiement par la Poste. En revanche, l'accent est mis sur la gestion du catalogue et les capacités de recherche afin de fidéliser l'internaute.

Au niveau des risques techniques, le chef de projet a classé au plus haut la maintenance du catalogue, à cause des problèmes liés à l'intégrité des informations régulièrement mises à jour semi-automatiquement dans la base de données centrale, et à la nécessité absolue d'avoir un catalogue valide et à jour. De même, le passage de commande est noté avec un risque haut à cause des problèmes de confidentialité et de cryptage à résoudre de façon sûre.

Planification du projet en itérations

À partir du classement précédent, le chef de projet a proposé au comité de pilotage le découpage en itérations suivant :

Tableau 3–2 Planification des itérations grâce aux cas d'utilisation

Cas d'utilisation	Priorité	Risque	Itération #
Chercher des ouvrages	Haute	Moyen	2
Gérer son panier	Haute	Bas	4
Effectuer une commande	Moyenne	Haut	3
Créer un compte client	Haute	Bas	5
Consulter ses commandes en cours	Basse	Moyen	7
Consulter l'aide en ligne	Basse	Bas	10
Gérer son compte client	Moyenne	Bas	9
Maintenir le catalogue	Haute	Haut	1
Maintenir les informations éditoriales	Moyenne	Bas	8
Maintenir le site	Moyenne	Bas	6

Un des bons principes du Processus Unifié consiste à identifier et lever les risques majeurs au plus tôt. Le chef de projet doit donc prendre en compte de façon combinée la priorité fonctionnelle et l'estimation du risque :

- Si la priorité est haute et le risque également, il faut planifier le cas d'utilisation dans une des toutes premières itérations (exemple : `Maintenir le catalogue`).
- Si la priorité est basse et le risque également, on peut reporter le cas d'utilisation à une des toutes dernières itérations (exemple : `Consulter l'aide en ligne`).
- Les choses se compliquent lorsque les deux critères sont antagonistes ! Le chef de projet doit alors décider en pesant le pour et le contre. Il peut être amené à négocier avec le client pour le convaincre qu'il vaut mieux pour le projet traiter en premier un cas d'utilisation risqué mais peu prioritaire, au lieu d'un cas d'utilisation plus prioritaire mais ne comportant pas de risque.

Traçabilité avec les exigences textuelles

À la fin du chapitre précédent, nous avons saisi un certain nombre d'exigences textuelles dans l'outil RequisitePro (IBM – Rational). Nous avions également commencé à établir des liens de traçabilité entre exigences, par exemples entre exigences de performances et exigences fonctionnelles.

ATTENTION Processus adaptatif

Ce découpage préliminaire en itérations devra être remis en cause et réévalué à chaque fin d'itération. On qualifie le processus itératif et incrémental d'adaptatif, par opposition à prédictif.
De plus, les itérations ont habituellement une durée fixe, de l'ordre de quatre semaines. Or, il n'est pas certain du tout que chaque cas d'utilisation pourra être conçu, développé et testé dans ce laps de temps. Certains nécessiteront plusieurs itérations, d'autres au contraire pourront être regroupés dans la même.

MÉTHODE Check-list de questions pour le chef de projet

- Quel est le problème métier que vous essayez de résoudre ?
- Quelles fonctionnalités sont essentielles à la solution ?
- Comment la solution proposée peut-elle être décrite de façon à être comprise par des lecteurs techniques et d'autres qui ne le sont pas du tout ?
- Quelles sont les ressources disponibles (temps, personnes, argent) ?
- Comment les exigences doivent-elles être priorisées ?
- Comment peut-on vérifier que le système fonctionnera comme décrit, avec une efficacité et des performances acceptables ?
- Comment peut-on garder la trace des relations de dépendance entre les exigences ?
- Comment peut-on limiter et négocier les demandes de modification de telle sorte que le produit puisse être réalisé dans les temps sans pour autant mécontenter les parties prenantes ?
- Quelle est la procédure de revue et de décision pour les demandes de modification des exigences ?

Nous allons maintenant établir la correspondance entre les cas d'utilisation que nous venons d'identifier et les exigences textuelles. Cette étude permet à la fois de valider la complétude des cas d'utilisation, mais aussi d'améliorer celle des exigences textuelles (qui sont parfois contractuelles pour les projets industriels).

Pour réaliser concrètement cette traçabilité, nous allons mettre en œuvre le pont entre les outils RSM (Rational Software Modeler) et RequisitePro. Pour cela, nous allons d'abord ouvrir le référentiel d'exigences à l'intérieur de l'outil de modélisation UML. RSM permet ainsi d'ouvrir une perspective « exigences » en même temps que la perspective « modélisation », ainsi que le montre la figure suivante.

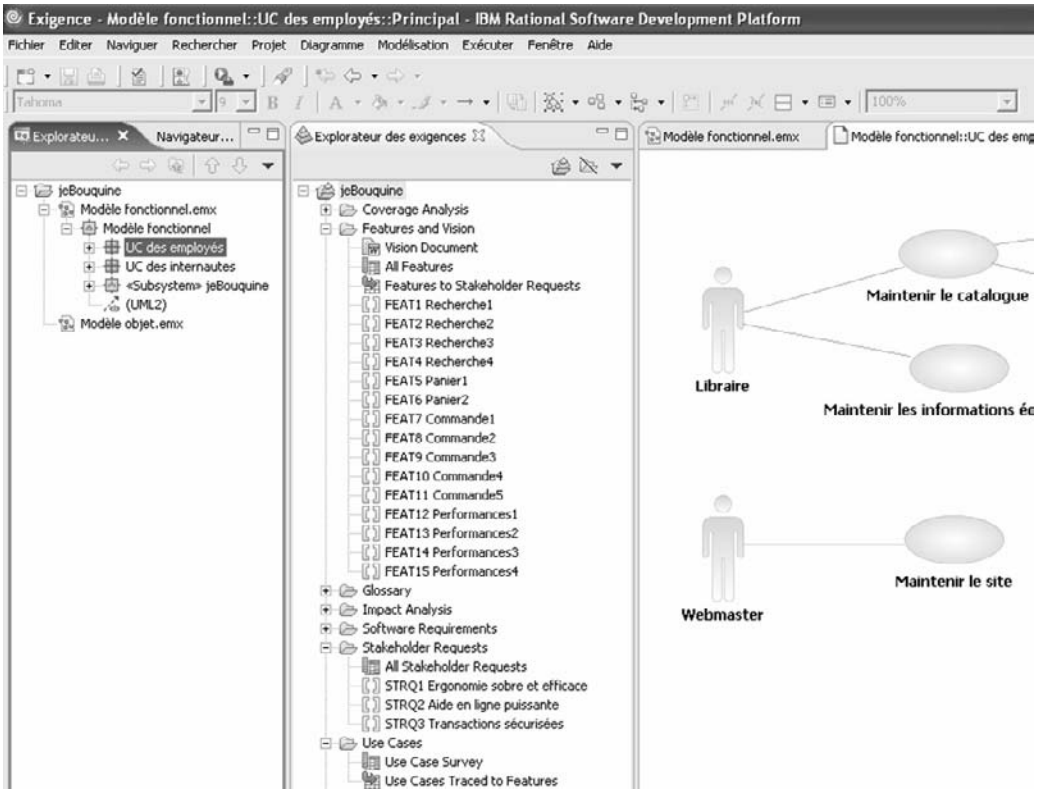


Figure 3–13 Les exigences visibles dans RSM

Nous allons pouvoir ajouter les cas d'utilisation dans RequisitePro par de simples glisser-déplacer de l'explorateur de modèle UML vers l'explorateur d'exigences à l'intérieur de RSM. Le résultat est illustré par la figure 3-14.

Il ne reste plus maintenant qu'à relier les cas d'utilisation aux exigences par des liens de traçabilité dans RequisitePro. Le cas d'utilisation `Chercher des livres` sera ainsi relié aux exigences appelées `RechercheN`, ainsi qu'à

certaines exigences de performances, etc. RequisitePro permet ensuite de visualiser une matrice de traçabilité telle que celle de la figure 3-15.

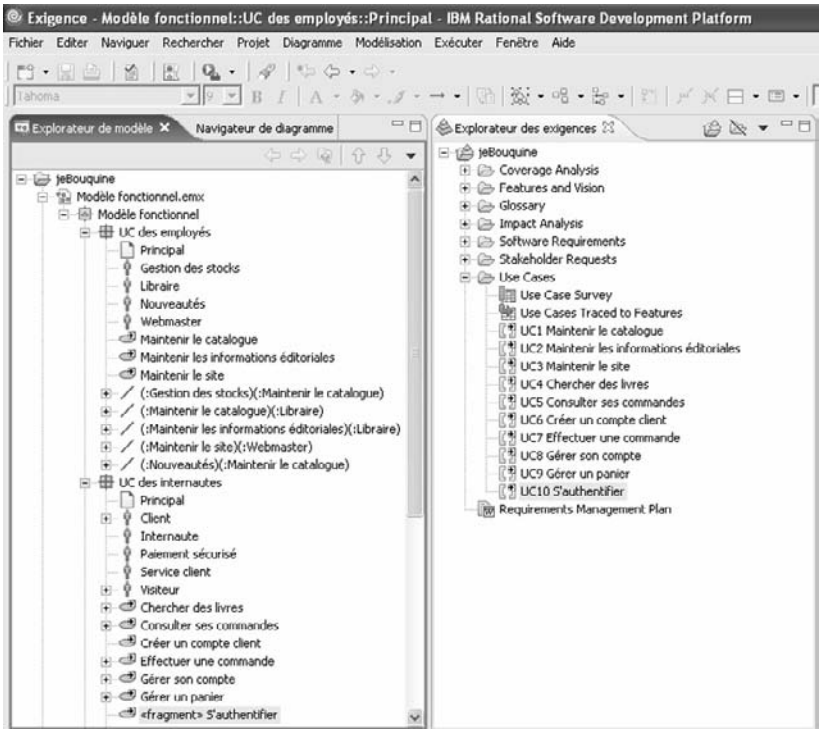


Figure 3–14
Les cas d'utilisation ajoutés dans l'explorateur d'exigences

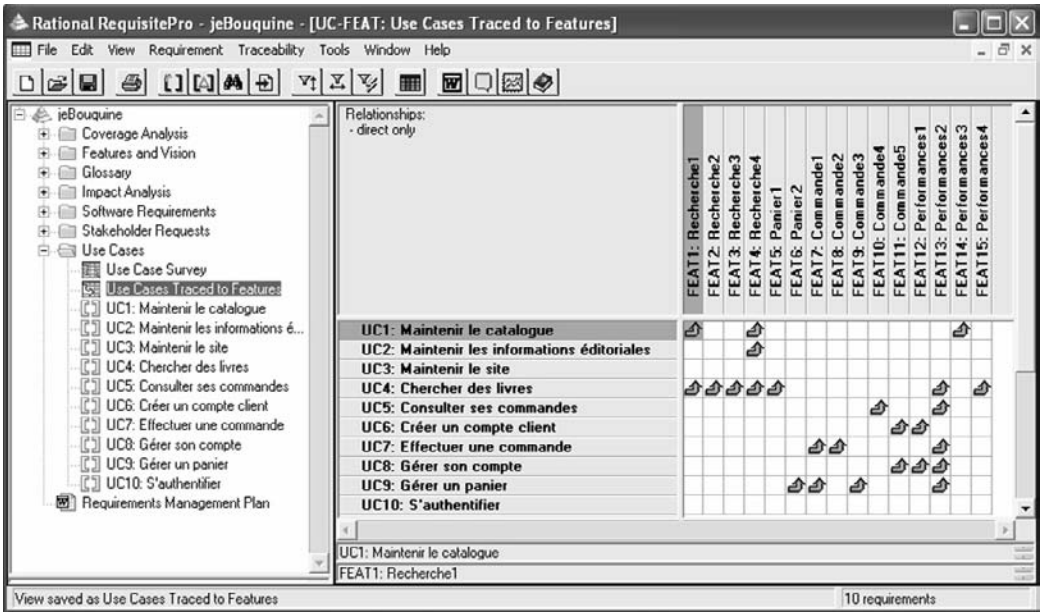


Figure 3–15 Matrice de traçabilité entre cas d'utilisation et exigences

Nous pouvons, par exemple, déduire de cette matrice que toutes les exigences textuelles ont bien été tracées par rapport à au moins un cas d'utilisation. Par contre, le cas d'utilisation `s'authentifier` doit induire de nouvelles exigences, puisqu'il n'est pas relié du tout. Il est tout à fait courant que l'identification des cas d'utilisation amène ainsi à une révision des exigences textuelles.

Il est à noter qu'Enterprise Architect permet également de réaliser cette matrice de traçabilité grâce à une capacité très pratique de l'outil. En effet, EA fournit une vue appelée *Relationship Matrix* permettant de montrer sous forme tabulaire les relations qui nous intéressent entre éléments à sélectionner. On peut ainsi facilement choisir les cas d'utilisation et les exigences, puis se limiter à la relation de dépendance, comme montré sur la figure suivante.

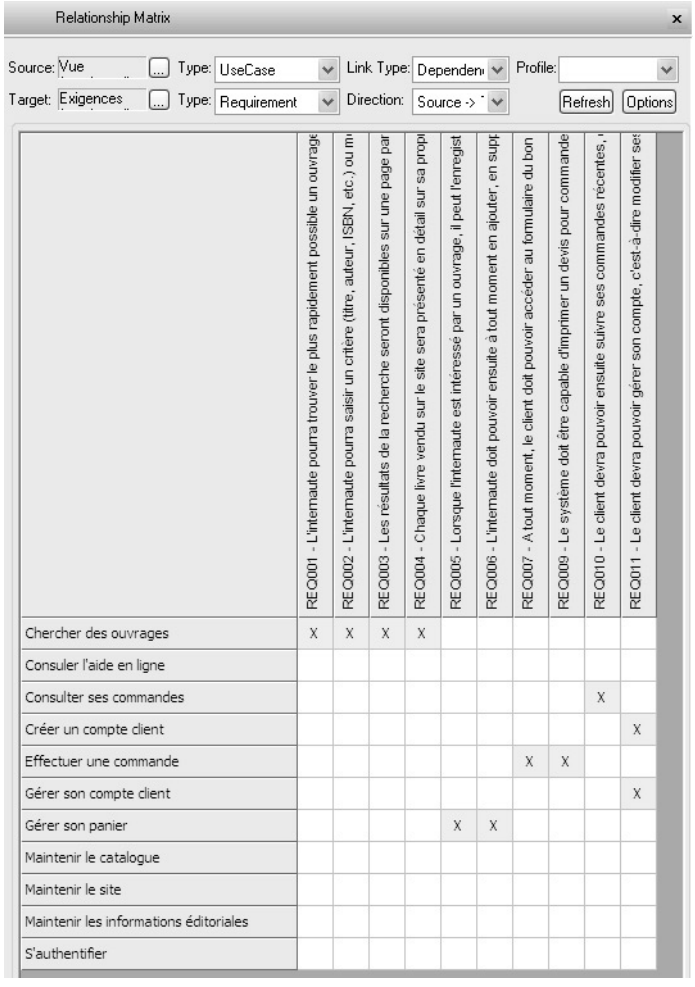
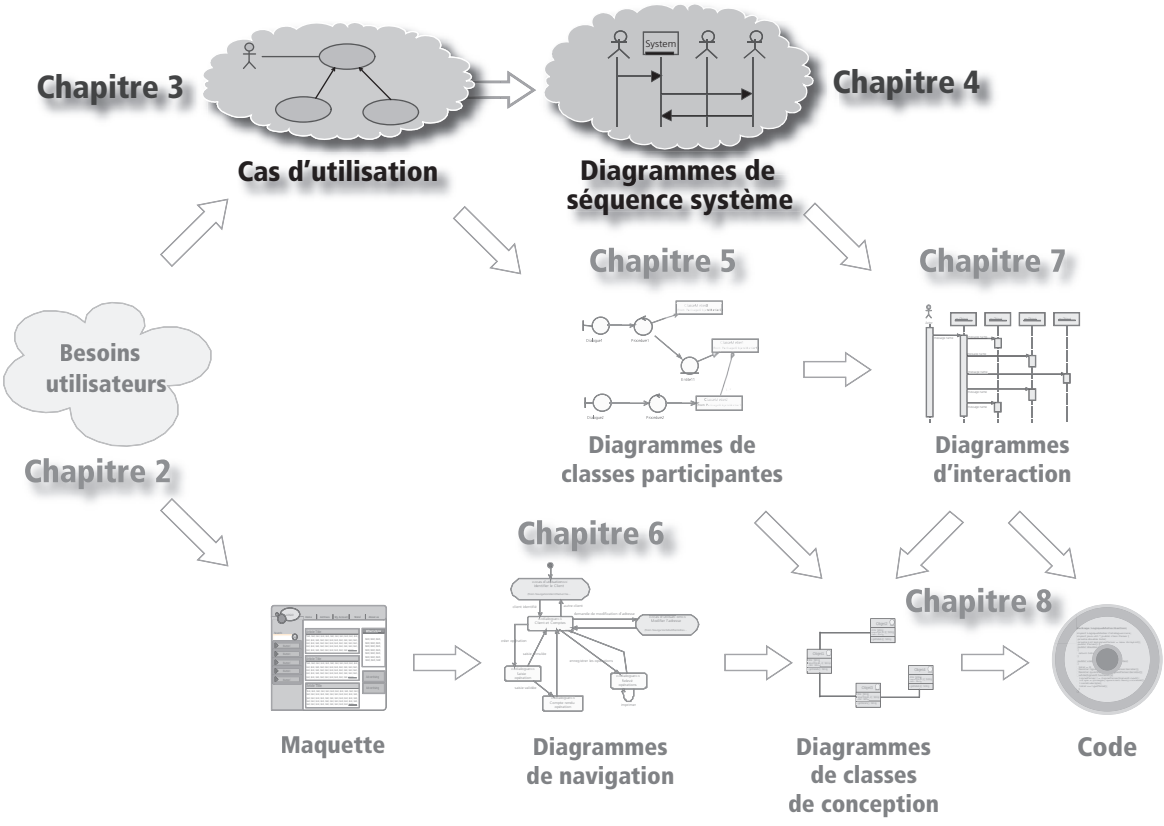
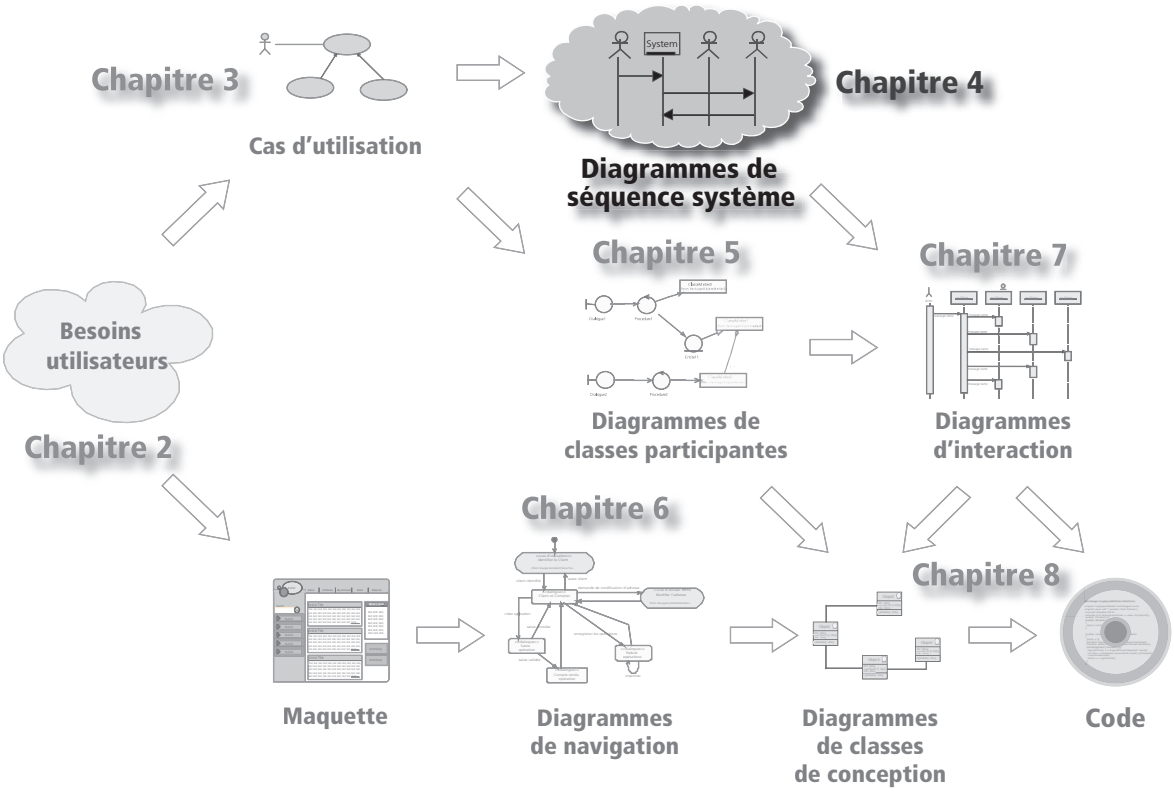


Figure 3–16
Matrice de relations entre cas d'utilisation et exigences sous EA


Chapitre 3
Cas d'utilisation
System
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Chapitre 7
Besoins utilisateurs
Chapitre 2
Diagrammes de classes participantes
Diagrammes d'interaction
Chapitre 6
Chapitre 8
Maquette
Diagrammes de navigation
Diagrammes de classes de conception
Code

chapitre 4


Chapitre 3
Cas d'utilisation
System
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Diagrammes de classes participantes
Chapitre 7
Diagrammes d'interaction
Besoins utilisateurs
Chapitre 2
Chapitre 6
Maquette
Diagrammes de navigation
Chapitre 8
Diagrammes de classes de conception
Code

Spécification détaillée des exigences

Nous allons maintenant décrire de façon détaillée les cas d'utilisation que nous avons identifiés au chapitre 3. Nous apprendrons ainsi à remplir une fiche-type pour chaque cas d'utilisation. Nous complèterons cette description textuelle par une représentation graphique UML très utile : le diagramme de séquence « système ».

SOMMAIRE

MOTS-CLÉS

- Acteur
- Cas d'utilisation
- Scénario
- Diagramme de séquence
- Opération système

Démarche

Rappelons la situation de cette activité de spécification des exigences par rapport à l'ensemble du processus décrit au chapitre 1. L'expression préliminaire des besoins donne lieu à une modélisation par les cas d'utilisation et à une maquette d'interface homme-machine (IHM). Nous avons identifié les acteurs et les cas d'utilisation au chapitre 3. Nous allons maintenant apprendre à les décrire de façon détaillée afin d'obtenir une expression précise des besoins avant d'attaquer l'analyse et la conception objet.

Méthode Requirements

Le Processus Unifié (UP) place le modèle de cas d'utilisation et la maquette dans la discipline Requirements (Spécifications). Les chapitres 3 et 4 de ce livre vont tout à fait dans ce sens.

Méthode Processus itératif

L'identification des cas d'utilisation, comme réalisée dans le chapitre 3, se fait lors de la phase d'Inception du RUP. Il s'agit d'un travail préliminaire visant à définir le périmètre fonctionnel du projet.
La description détaillée des cas d'utilisation, que nous allons réaliser dans ce chapitre 4, se fait principalement lors de la phase d'Élaboration, mais de façon itérative et incrémentale.

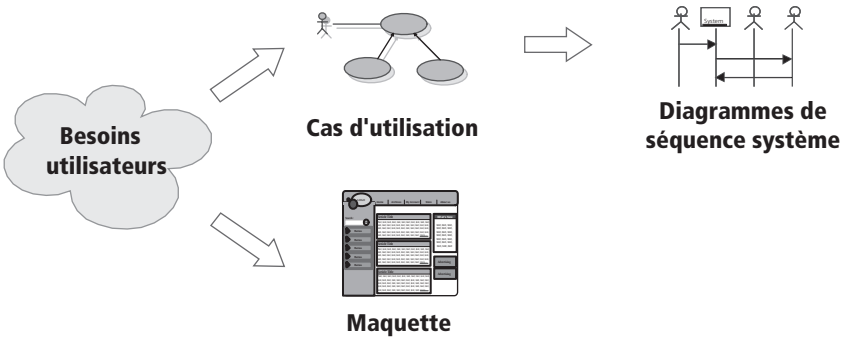


Figure 4–1 Les cas d'utilisation et leurs prolongements dans la démarche

Plan-type de description textuelle des cas d'utilisation

La fiche de description textuelle d'un cas d'utilisation n'est pas normalisée par UML. Nous allons donc en proposer une qui soit adaptée à notre problème.

Référence Rédiger des cas d'utilisation

Nous nous sommes particulièrement inspirés de l'excellent ouvrage d'Alistair Cockburn :

📖 *Rédiger des cas d'utilisation efficaces*, A. Cockburn, Eyrolles, 2001.

Scénarios

Pour donner une autre définition du cas d'utilisation, on peut dire que c'est une collection de scénarios de succès ou d'échec qui décrit la façon dont un acteur particulier utilise un système pour atteindre un objectif. Pour détailler la dynamique du cas d'utilisation, la procédure la plus évidente consiste à recenser de façon textuelle toutes les interactions entre les acteurs et le système. Le cas d'utilisation doit avoir un début et une fin clairement identifiés. Il faut aussi préciser les variantes possibles tout en essayant d'ordonner séquentiellement les descriptions afin d'améliorer leur lisibilité.

Nous allons ainsi distinguer :

- le scénario « nominal », celui qui satisfait les objectifs des acteurs par le chemin le plus direct de succès,
- des alternatives, qui comprennent tous les autres scénarios, de succès (fin normale) ou d'échec (erreur).

La figure 4-2 propose une vue graphique stylisée d'un cas d'utilisation avec ses scénarios.

B.A.-BA **Scénario**

Un scénario est une suite spécifique d'interactions entre les acteurs et le système à l'étude. On peut dire que c'est une « instance » du cas d'utilisation, un chemin particulier dans sa combinatoire.

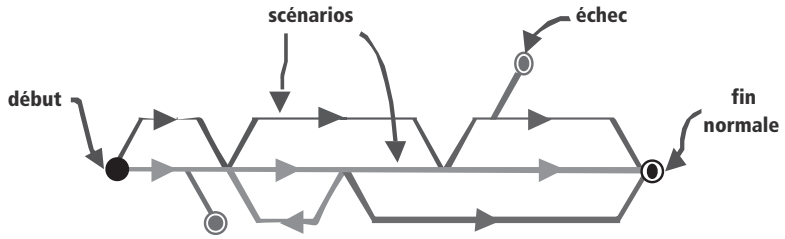


Figure 4–2
Représentation graphique des scénarios d'un cas d'utilisation

Chaque scénario est composé d'étapes qui peuvent être de trois sortes :

- un message d'un acteur vers le système,
- une validation ou un changement d'état du système,
- un message du système vers un acteur.

Les étapes sont numérotées séquentiellement afin de pouvoir facilement indiquer par la suite les alternatives possibles.

EXEMPLE **DAB : scénario nominal**

Cas d'utilisation : Retirer de l'argent au distributeur (DAB) avec une carte bancaire.

1. Le porteur de carte introduit sa carte dans le DAB.
2. Le DAB vérifie que la carte introduite est bien une carte bancaire.
3. Le DAB demande au porteur de carte de fournir son code d'identification.
4. Le porteur de carte entre son code d'identification.
5. Le DAB valide le code d'identification (par rapport à celui qui est codé sur la puce de la carte).
6. Le DAB demande une autorisation au système d'autorisation externe.
7. Le système d'autorisation externe donne son accord et indique le solde hebdomadaire.
8. Le DAB demande au porteur de carte de saisir le montant désiré du retrait.
9. ...

Les alternatives sont très importantes. Elles indiquent tous les autres scénarios ou branchements possibles, aussi bien de succès que d'échec. Comme elles doivent se brancher sur le scénario nominal, la convention de numérotation des étapes prend toute son importance. Ainsi, à une étape « X », une première alternative se notera « Xa ». Elle identifiera d'abord la condition qui provoque l'alternative, puis la réponse du système. Le principe consiste à décrire la condition comme quelque chose qui peut être détecté par le système. Ensuite la réponse du système peut être résumée en une seule étape ou comprendre également une séquence d'étapes comme dans l'exemple du DAB. Une seconde alternative à la même étape se notera « Xb » et ainsi de suite. S'il est nécessaire de signaler qu'une condition d'alternative peut survenir entre les étapes X et Y, elle sera notée « X-Ya ». Enfin, si elle peut arriver à tout moment du scénario nominal, elle sera notée « *a ».

EXEMPLE DAB : alternatives

2a. La carte introduite n'est pas reconnue par le DAB.
1. Le DAB éjecte la carte et le cas d'utilisation se termine en échec.

5a. Le DAB détecte que le code saisi est erroné, pour la première ou deuxième fois.
1. Le DAB indique au porteur de carte que le code est erroné.
2. Le DAB enregistre l'échec sur la carte et le cas d'utilisation reprend à l'étape 5 du scénario nominal.

5b. Le DAB détecte que le code saisi est erroné, pour la troisième fois.
1. Le DAB indique au porteur de carte que le code est erroné pour la troisième fois.
2. Le DAB confisque la carte.
3. Le DAB informe le système d'autorisation externe et le cas d'utilisation se termine en échec.

Préconditions et postconditions

Les préconditions définissent ce qui doit être vrai en amont du cas d'utilisation pour que celui-ci puisse démarrer. Elles ne sont pas testées à l'intérieur du cas d'utilisation, mais sont tenues pour acquises. Souvent, une précondition implique que le scénario nominal d'un autre cas d'utilisation s'est déroulé normalement. Certaines préconditions triviales n'ont pas besoin d'être mentionnées (« Le système doit être alimenté en courant électrique… ») : seules celles que le rédacteur juge importantes et dignes d'intérêt doivent être répertoriées.

Les postconditions définissent ce qui doit être vrai lorsque le cas d'utilisation se termine avec succès, qu'il s'agisse du scénario nominal ou d'un scénario alternatif.

EXEMPLE Quelques conditions

Préconditions :
- La caisse du DAB n'est pas vide.
- La connexion avec le système d'autorisation est opérationnelle.

Postconditions :
- La caisse du DAB contient moins de billets qu'au début du cas d'utilisation (le nombre de billets manquants est fonction du montant du retrait).
- Une opération de retrait a été archivée (en cas de succès comme en cas d'échec).

Exigences supplémentaires

Très souvent, les exigences non fonctionnelles et les contraintes de conception se rapportent spécifiquement à un cas d'utilisation plutôt qu'au système dans sa totalité. Dans ce cas, on les documente dans un paragraphe complémentaire. Typiquement, pour notre site e-commerce, il s'agira de performance, de sécurité ou d'ergonomie. On complètera par exemple la description des scénarios par des copies d'écran de la maquette.

EXEMPLE Exigences supplémentaires (pour le DAB)

- Un scénario nominal de retrait doit durer moins de 2 minutes dans 90 % des cas.
- La comparaison du code d'identification saisi avec celui de la carte doit être fiable à 10-6.

Spécification détaillée des cas d'utilisation du site web

Rappel des résultats des spécifications préliminaires

Nous avons structuré les cas d'utilisation en trois packages : ceux de l'internaute, ceux des employés de jeBouquine et les cas de second rang. Comme nous l'avons vu au chapitre 3, le chef de projet a proposé au comité de pilotage le découpage en itérations du tableau 4-1.

ATTENTION Travail itératif

Nous ne préconisons pas une démarche séquentielle comme la numérotation des chapitres pourrait le faire croire. Dans le cadre d'un processus itératif et incrémental, tous les cas d'utilisation ne sont pas décrits en détail d'un bloc, puis conçus également d'un bloc, etc. Le travail est réalisé itération après itération.
Le découpage en itérations effectué au chapitre 3 va nous guider pour planifier notre travail de rédaction détaillée des cas d'utilisations. Celui-ci sera effectué au moment voulu, lors de l'itération où il a été décidé de concevoir et développer le cas d'utilisation en question.

Tableau 4–1 Planification des itérations grâce aux cas d'utilisation

Cas d'utilisation	Priorité	Risque	Itération #
Chercher des ouvrages	Haute	Moyen	2
Gérer son panier	Haute	Bas	4
Effectuer une commande	Moyenne	Haut	3
Créer un compte client	Haute	Bas	5
Consulter ses commandes en cours	Basse	Moyen	7
Consulter l'aide en ligne	Basse	Bas	10
Gérer son compte client	Moyenne	Bas	9
Maintenir le catalogue	Haute	Haut	1
Maintenir les informations éditoriales	Moyenne	Bas	8
Maintenir le site	Moyenne	Bas	6

Pour illustrer notre démarche, nous allons détailler les cas d'utilisation des quatre premières itérations, à savoir, dans l'ordre :

- `Maintenir le catalogue`,
- `Chercher des ouvrages`,
- `Gérer son panier`,
- `Effectuer une commande`.

Maintenir le catalogue

Acteur principal

Le `Libraire`.

Acteurs secondaires

Les deux systèmes `Nouveautés` et `Gestion des stocks`.

Objectifs

Le `Libraire` veut pouvoir contrôler la mise à jour automatique du catalogue des ouvrages présentés sur le site web.

L'initiale du nom des acteurs est volontairement en capitale afin de faciliter leur identification dans le texte.

Préconditions

- Le `Libraire` s'est authentifié sur l'intranet (voir le cas d'utilisation `S'authentifier`).
- La version courante du catalogue est accessible.

Postconditions

Une nouvelle version du catalogue est disponible.

Scénario nominal

1 Le système `Nouveautés` alimente le site avec les nouveaux ouvrages.

2 Le système `Gestion des stocks` met à jour les données qui concernent le prix et l'état du stock.

3 Le `Libraire` valide la mise à jour du catalogue.

Alternatives

1-2a Le `Système` détecte un dysfonctionnement des systèmes externes de mise à jour.

1. Le `Système` signale le dysfonctionnement au `Libraire`.
2. Le `Libraire` invalide la mise à jour partielle ou erronée et revient à la version précédente du catalogue. Il prévient également le `Webmaster` pour qu'il engage des actions de maintenance. Le cas d'utilisation se termine en échec.

3a Le `Libraire` détecte des erreurs ou des incohérences parmi les nouvelles informations.

1. Le `Libraire` modifie toutes les informations erronées.
2. Le `Libraire` valide la mise à jour du catalogue.

3b Le `Libraire` veut ajouter d'autres informations.

1. Le `Libraire` exécute le cas d'utilisation `Maintenir les informations éditoriales`.
2. Le `Libraire` valide la mise à jour du catalogue.

Exigences supplémentaires

Le catalogue est mis à jour quotidiennement.

MÉTHODE **Liens entre descriptions de cas d'utilisation**

Comme indiqué en 3b.1, l'idéal consiste à établir un lien hypertexte avec le fichier de description textuelle du cas d'utilisation concerné (ici : `Maintenir les informations éditoriales`).
Le cas qui participe à l'alternative sera implémenté dans une itération ultérieure. Ce n'est pas vraiment un problème : on testera tous les scénarios sauf le 3b dans l'itération 1, et le travail de l'itération 8 se raccrochera de manière incrémentale à celui de l'itération 1.

L'IHM du libraire pourra ressembler au dessin suivant, réalisé avec l'outil EA, permettant ainsi de relier les dessins de la maquette aux cas d'utilisation !

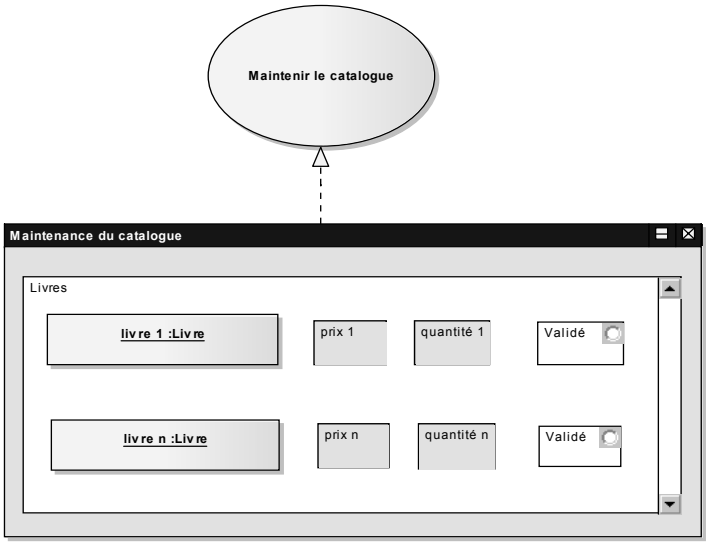


Figure 4–3
Dessin d'écran pour la maintenance du catalogue

Chercher des ouvrages

Acteur principal

L'`Internaute` (qu'il soit déjà client, ou simple visiteur)

Objectifs

L'`Internaute` veut trouver le plus rapidement possible un ouvrage précis dans l'ensemble du catalogue. Il veut également pouvoir flâner comme il le ferait dans une vraie bibliothèque et chercher des livres avec des critères variés.

Préconditions

Le catalogue est disponible (voir le cas d'utilisation `Maintenir le catalogue`).

Postconditions

L'`Internaute` a trouvé l'ouvrage précis qu'il cherchait, ou un ouvrage qui l'intéresse, voire plusieurs.

Scénario nominal

1 L'`Internaute` lance une recherche rapide à partir de mots-clés : un thème, un titre, le nom d'un auteur, etc.

2 Le `Système` affiche une page de résultat. Les ouvrages sont classés par défaut par date de parution, le plus récent en premier.

Figure 4–4
Exemple de recherche rapide et de page de résultats

3 L'`Internaute` sélectionne un ouvrage.

4 Le `Système` lui présente une fiche détaillée pour l'ouvrage sélectionné. On y trouvera en particulier (voir figure 4–5) :
- une image (pour la majorité des ouvrages),
- ses titre, sous-titre, auteur(s), éditeur, date de parution, nombre de pages, langue,
- son prix et sa disponibilité,
- des éventuels commentaires de lecteurs déjà clients,
- la table des matières détaillée, des extraits de chapitres, etc.

Alternatives

1a L'`Internaute` n'a pas d'idée préconçue et préfère flâner dans les rayons de la librairie virtuelle. Pour cela, le `Système` lui propose un ensemble de pages telles que : *nouveautés*, *meilleures ventes*, *sélection du libraire* (par thème).

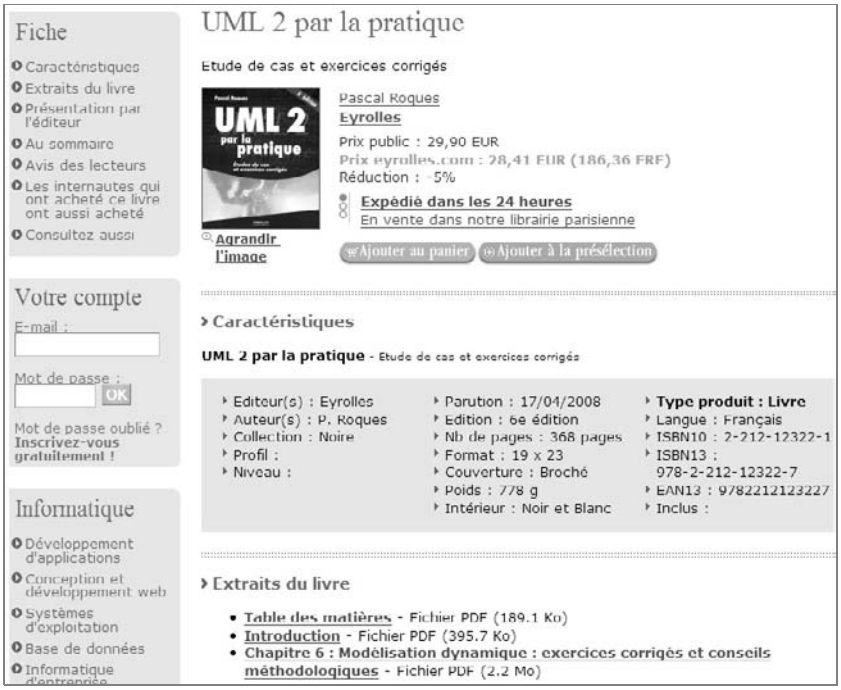


Figure 4–5
Exemple de fiche d'ouvrage

Figure 4–6
Exemple d'écran général de recherche

1. L'`Internaute` navigue dans ces pages et peut enchaîner sur l'étape 3 du scénario nominal.

1b L'`Internaute` choisit d'effectuer une recherche avancée.

1. L'`Internaute` accède à un formulaire spécialisé lui permettant de combiner plusieurs types de recherche : par sujet, titre, auteur, éditeur, langue, etc. Il peut également saisir directement un numéro ISBN, un code éditeur, etc.

   Il peut saisir seulement le début significatif d'un mot en terminant par « * ». Le moteur de recherche cherchera tous les mots commençant ainsi.

   Les suffixes de pluriel sont supprimés automatiquement pendant la recherche. Ainsi, une recherche sur « programme » permettra de trouver également « programmer » et « programmes ».

   L'`Internaute` peut utiliser des opérateurs logiques entre les mots de sa recherche. L'opérateur « ET » (« AND ») est utilisé par défaut.

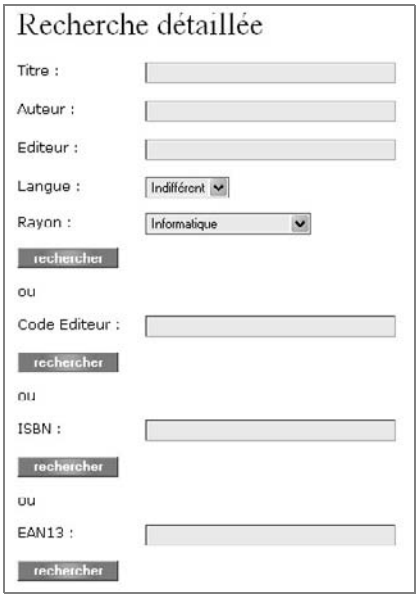


Figure 4–7 Exemple de recherche avancée

Par exemple, la recherche sur le titre « UML NON C++ », combinée à la recherche sur l'éditeur « CampusPress OU Eyrolles » retournera les ouvrages dont le titre contient le mot « UML » mais pas le mot « C++ » et qui sont édités par CampusPress ou Eyrolles.

2a Le `Système` n'a pas trouvé d'ouvrage correspondant à la recherche.

1. Le `Système` signale l'échec à l'`Internaute` et lui propose d'effectuer une nouvelle recherche. Le cas d'utilisation redémarre à l'étape 1 du scénario nominal.

2b Le `Système` a trouvé de très nombreux ouvrages.

1. Le `Système` signale le nombre d'ouvrages à l'`Internaute` et lui affiche une première page de résultats. Les autres pages sont accessibles directement ou par des symboles *Suivante* et *Précédente*.
2. L'`Internaute` navigue dans ces pages et enchaîne éventuellement sur l'étape 3 du scénario nominal. Il peut également reclasser les ouvrages obtenus par différents critères : titre, auteur, langue, disponibilité, etc.

3a L'`Internaute` n'est pas intéressé par les résultats.

1. L'`Internaute` revient à l'étape 1 du scénario nominal pour lancer une nouvelle recherche.
2. L'`Internaute` abandonne la recherche. Le cas d'utilisation se termine en échec.

1 L'`Internaute` est intéressé par le résultat et met un ouvrage dans le panier.

1. Le système affiche le panier de l'internaute (voir le cas d'utilisation `Gérer son panier`).

Méthode Choix exclusif

Soit l'internaute lance une nouvelle recherche, soit il abandonne. Les deux branches exclusives de l'alternative sont simplement indiquées ici en utilisant le même numéro « 1 ». Il ne s'agit donc pas d'une séquence comme dans l'extension 2b.

Attention Modélisation des enchaînements de cas d'utilisation

Le fait qu'un cas d'utilisation puisse optionnellement en lancer un autre sera décrit de façon plus formelle dans le chapitre 6 consacré à la modélisation de la navigation. Le diagramme de cas d'utilisation, même avec les relations de type « include » et « extend », n'est pas le bon endroit pour modéliser ces enchaînements, contrairement à ce que certains croient !

Exigences supplémentaires

- La recherche doit être la plus rapide possible : 95 % des requêtes doivent aboutir en moins de 3 s.
- Les résultats de la recherche doivent être pertinents, c'est-à-dire correspondre à la requête dans au moins 99 % des cas.
- Le formulaire de recherche rapide doit être toujours visible et donc se situer dans la partie supérieure de toutes les pages, quelle que soit la résolution d'écran de l'`Internaute`.

Gérer son panier

Acteur principal

L'`Internaute` (qu'il soit déjà client, ou simple visiteur).

Objectifs

Lorsque l'`Internaute` est intéressé par un ouvrage, il faut qu'il puisse l'enregistrer dans un panier virtuel. Ensuite, il doit pouvoir ajouter

d'autres livres, en supprimer ou encore en modifier les quantités avant de passer commande.

Préconditions

Néant.

Postconditions

Néant.

Scénario nominal

1 L'`Internaute` enregistre les ouvrages qui l'intéressent dans un panier virtuel (voir le cas d'utilisation `Chercher des ouvrages`).

2 Le `Système` lui affiche l'état de son panier. Chaque ouvrage qui a été préalablement sélectionné est présenté sur une ligne, avec son titre, son auteur et son éditeur. Son prix unitaire est affiché, la quantité est positionnée à « 1 » par défaut, et le prix total de la ligne est calculé. Le total général est calculé par le `Système` et affiché en bas du panier, avec le nombre d'articles.

Figure 4–8
Exemple de panier virtuel

3 L'`Internaute` continue ses achats (voir le cas d'utilisation `Chercher des ouvrages`).

Alternatives

2a Le panier est vide.

1. Le `Système` affiche un message d'erreur à l'`Internaute` (« Votre panier est vide ») et lui propose de revenir à une recherche d'ouvrage (voir le cas d'utilisation `Chercher des ouvrages`).

> À RETENIR **Préconditions et postconditions pertinentes**
>
> Il n'y pas toujours de préconditions ou de postconditions pertinentes !
> Rien n'interdit à l'`Internaute` de demander à accéder à son panier alors qu'il n'a rien sélectionné : le panier sera vide. De même, à la fin de la gestion du panier, on ne peut rien prédire sur son état.

4a L'`Internaute` modifie les quantités des lignes du panier, ou en supprime.

1. L'`Internaute` revalide en demandant la mise à jour du panier.
2. Le cas d'utilisation reprend à l'étape 2 du scénario nominal.

4b L'`Internaute` demande un devis pour commander par courrier.

1. Le `Système` fournit un devis imprimable à joindre au règlement récapitulant la commande et le total à payer.

1 L'`Internaute` souhaite commander en ligne.

1. Le `Système` l'amène sur la page d'identification.

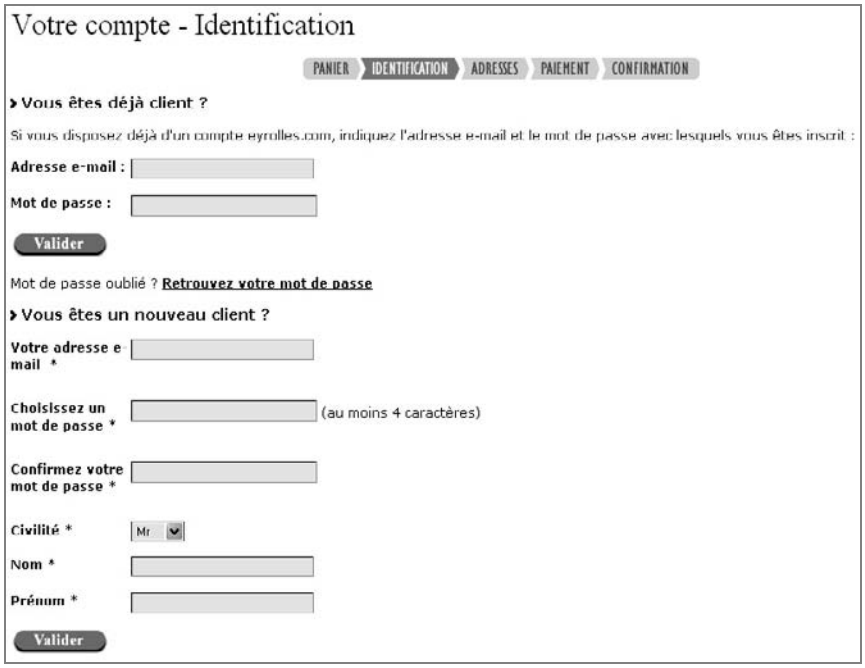


Figure 4–9
Exemple de page d'identification

2a L'`Internaute` s'identifie en tant que `Client` (voir le cas d'utilisation `S'authentifier`).

2b L'`Internaute` `Visiteur` demande à créer un compte client (voir le cas d'utilisation `Créer un compte client`).

Exigences supplémentaires

- Le calcul du total doit toujours être exact.
- Le panier de l'`Internaute` est sauvegardé pendant toute la durée de sa visite sur le site web.

Effectuer une commande

Acteur principal

Le `Client`.

Objectifs

À tout moment, le client doit pouvoir accéder au formulaire du bon de commande, dans lequel il peut saisir ses coordonnées et les informations nécessaires au paiement et à la livraison.

Préconditions

Le panier du client n'est pas vide et il s'est identifié.

Postconditions

- Une commande a été enregistrée et transmise au service `Commandes`.
- Une transaction cryptée a été réalisée avec le système externe de `Paiement sécurisé` et sauvegardée.

Scénario nominal

1 Le `Client` saisit l'ensemble des informations nécessaires à la livraison, à savoir :
   – les coordonnées de l'adresse de facturation (nom, prénom, adresse postale complète, téléphone),
   – les coordonnées de l'adresse de livraison si elle est différente de l'adresse de facturation (nom, prénom, adresse postale complète, téléphone).

2 Le `Système` affiche un récapitulatif des adresses indiquées et du panier à commander (voir figure 4–10).

3 Le `Client` sélectionne le paiement par carte bancaire et valide sa commande. Il doit pour cela fournir un numéro de carte de crédit avec son type, sa date de validité et son numéro de contrôle.

4 Le `Système` envoie les informations cryptées au système externe de `Paiement sécurisé`.

5 Le `Paiement sécurisé` autorise la transaction.

6 Le `Système` confirme la prise de commande à l'`Internaute`.

7 Le `Système` envoie la commande validée au `Service clients` de jeBouquine.

8 Le `Système` enregistre la commande.

Alternatives

1-3a Le `Client` annule sa commande.

1. Le `Système` revient sur l'affichage du panier et le cas d'utilisation se termine en échec.

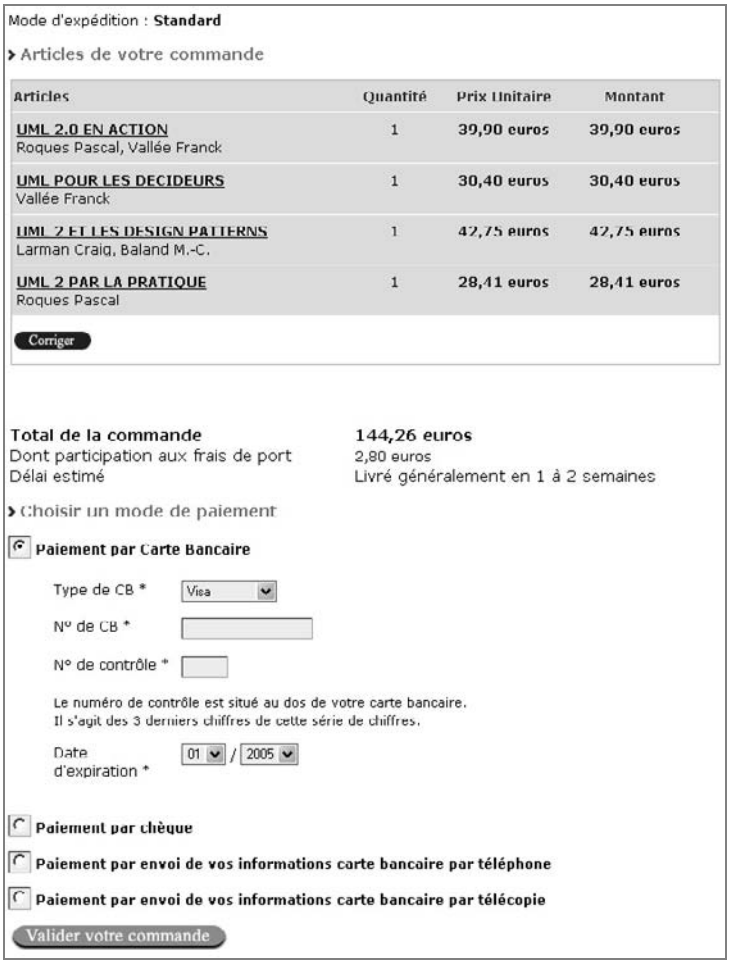


Figure 4–10
Exemple de récapitulatif de commande

1 Le `Client` choisit un paiement différé (chèque, etc.).

1. Le `Système` confirme la prise de commande à l'`Internaute` et lui indique la démarche à suivre pour la terminer.
2. Le `Système` enregistre la commande dans un état « non finalisée ».

4a Le `Système` détecte que les informations sur la carte sont incomplètes ou erronées (date de validité dépassée, etc.).

1. Le `Système` demande au client de modifier ou compléter les informations sur la carte.
2. Le cas d'utilisation reprend à l'étape 3 du scénario nominal.

5a Le `Paiement sécurisé` n'autorise pas la transaction ou ne répond pas.

1. Le `Système` indique au `Client` que le paiement par carte bancaire a échoué et propose de choisir un autre type de paiement.
2. Le cas d'utilisation reprend à l'étape 3 du scénario nominal.

MÉTHODE Unité de connexion

Le client devra pouvoir ensuite consulter ses commandes récentes, et même les modifier avant expédition, de façon sécurisée. Comme il s'agit d'actions que l'internaute peut entreprendre lors d'une autre visite au site web, il ne peut s'agir du même cas d'utilisation. En effet, une règle importante consiste à considérer qu'un cas d'utilisation a une unité de connexion, de session. C'est vrai pour tous les autres cas d'utilisation identifiés jusqu'à présent.
`Consulter ses commandes` est donc bien un cas d'utilisation à part entière.

Exigences supplémentaires :

- Pour garantir la sécurisation et la confidentialité des échanges, il est impératif que l'envoi des données se fasse de manière cryptée (protocole SSL).
- Les seules cartes bancaires acceptées sont les Visa, Eurocard-Mastercard et American Express.

Diagrammes de séquence système

Les cas d'utilisation décrivent les interactions des acteurs avec le site web que nous voulons spécifier et concevoir. Lors de ces interactions, les acteurs produisent des messages qui affectent le système informatique et appellent généralement une réponse de celui-ci. Nous allons isoler ces messages et les représenter graphiquement sur des diagrammes de séquence UML.

Pour les messages propres à un cas d'utilisation, les DSS (diagrammes de séquence système) montrent non seulement les acteurs externes qui interagissent directement avec le système, mais également ce système (en tant que boîte noire) et les événements système déclenchés par les acteurs. L'ordre chronologique se déroule vers le bas et l'ordre des messages doit suivre la séquence décrite dans le cas d'utilisation.

Nous allons représenter le DSS d'un scénario représentatif de chacun des cas d'utilisation décrits précédemment, en commençant par ceux des internautes.

B.A.-BA Diagramme de séquence « système » (DSS)

Nous utilisons le terme de diagramme de séquence « système » pour souligner le fait que nous considérons le système informatique comme une boîte noire. Le comportement du système est décrit vu de l'extérieur, sans préjuger de comment il le réalisera. Nous ouvrirons la boîte noire seulement en conception.

Chercher des ouvrages

Il faut repartir de la description textuelle détaillée du cas d'utilisation et transformer chaque étape en une flèche représentant un message. L'acteur principal (`Internaute`) est représenté à gauche du diagramme et le système (`JeBouquine`) à droite. Notez le mot-clé «`system`» que nous avons utilisé dans la boîte représentant le système boîte noire pour le différencier encore plus nettement des acteurs (surtout lorsqu'il y aura des acteurs non humains dans les scénarios).

B.A.-BA Message de retour

Sur la figure 4-11, la flèche pointillée partant du système à l'étude représente un retour au sens UML. Cela signifie que le message en question (par exemple : ouvrages trouvés) est le résultat direct du message précédent par une relation forte de cause à effet. En général, on ne fait figurer que les retours intéressants et non triviaux.

Attention Spécification d'activation

Sur la figure 4-11, la notation des bandes blanches le long des lignes verticales (appelées lignes de vie) représente l'activation de l'élément en question. Le système informatique n'est actif que lorsqu'il est sollicité par un acteur, alors que les acteurs sont a priori toujours actifs. Cette notation est optionnelle, mais aide à mieux comprendre la flèche pointillée du message de retour. Toutefois, dans un souci de simplicité, nous ne l'utiliserons pas pour les autres DSS.

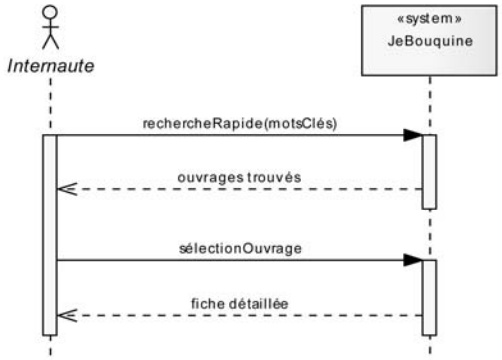


Figure 4–11
Premier diagramme de séquence système de Chercher des ouvrages

UML 2 fournit quelques notations complémentaires très utiles. Des rectangles, appelés fragments d'interaction, sont ainsi utilisables pour indiquer qu'un groupe de messages est optionnel (mot-clé opt), répété (mot-clé loop) ou alternatif (mot-clé alt). Nous avons utilisé ces nouvelles notations sur la figure 4-12 pour indiquer que :

- On peut effectuer soit une recherche rapide soit une recherche avancée.
- On peut optionnellement mettre l'ouvrage trouvé dans le panier.
- La recherche d'ouvrages est répétable à volonté…

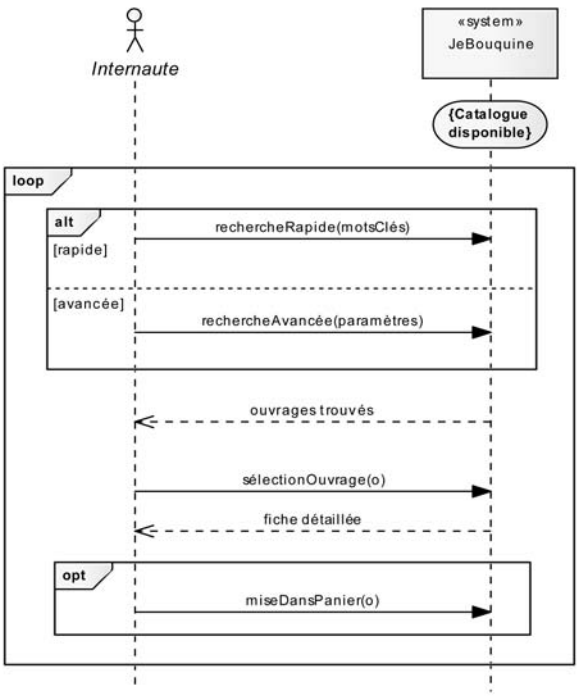


Figure 4–12
Deuxième DSS de Chercher des ouvrages

La précondition du cas d'utilisation peut également être matérialisée graphiquement grâce au symbole d'état sur la ligne de vie.

Il est même possible de représenter également le cas d'erreur où le système ne trouve pas d'ouvrage correspondant à la requête. Il suffit d'ajouter un cadre alt, contenant un cadre break (pour indiquer que les messages suivants, comme la sélection et la mise dans le panier, ne sont pas possibles). Le diagramme devient alors comme sur la figure 4-13.

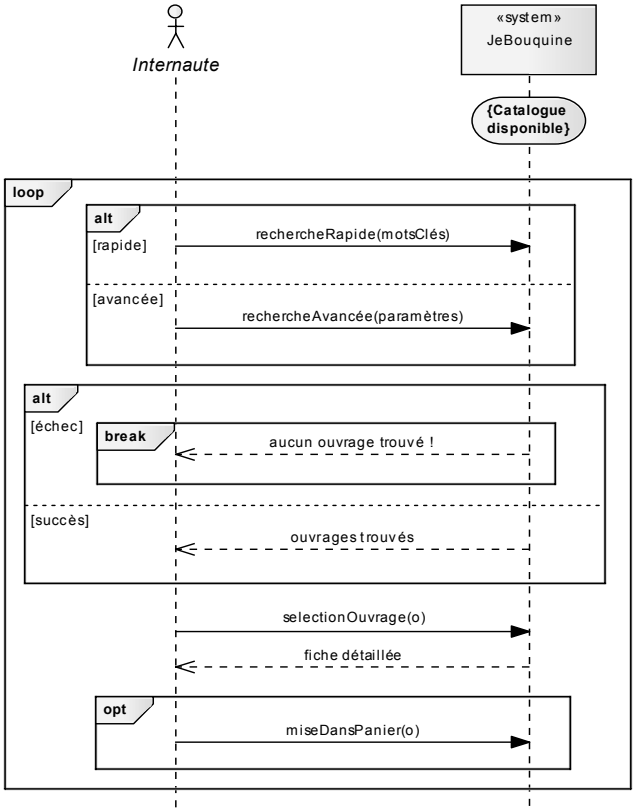


Figure 4–13
DSS plus complet de Chercher des ouvrages

Gérer son panier

Le DSS de la figure 4–14 représente ici plus que le scénario nominal. En effet, nous avons décrit un exemple plus complet de gestion du panier afin d'illustrer les actions répétables de modification de quantité et de suppression de lignes. Si nous voulons également représenter le fait qu'à la fin du cas d'utilisation, l'internaute peut optionnellement demander un devis ou passer une commande, il suffit de l'ajouter à la fin du diagramme comme indiqué sur la figure 4-15.

Nous n'avons pas continué dans le cas du passage de commande, car c'est l'objet du cas d'utilisation suivant, à savoir : `Effectuer une commande`.

UML 2 Renvoi vers d'autres cas d'utilisation

On notera le renvoi au cas d'utilisation `Chercher des ouvrages` en début de diagramme, pour montrer un exemple de remplissage du panier suite à une recherche réussie. Cette nouveauté UML 2, matérialisée par un cadre avec le mot-clé ref, est extrêmement utile. Une interaction peut ainsi en référencer une autre (et les outils gèrent des hyperliens très pratiques entre diagrammes).

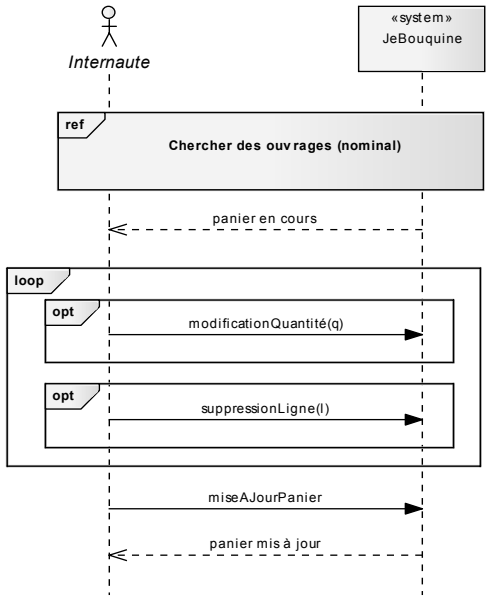


Figure 4–14
Premier DSS de Gérer son panier

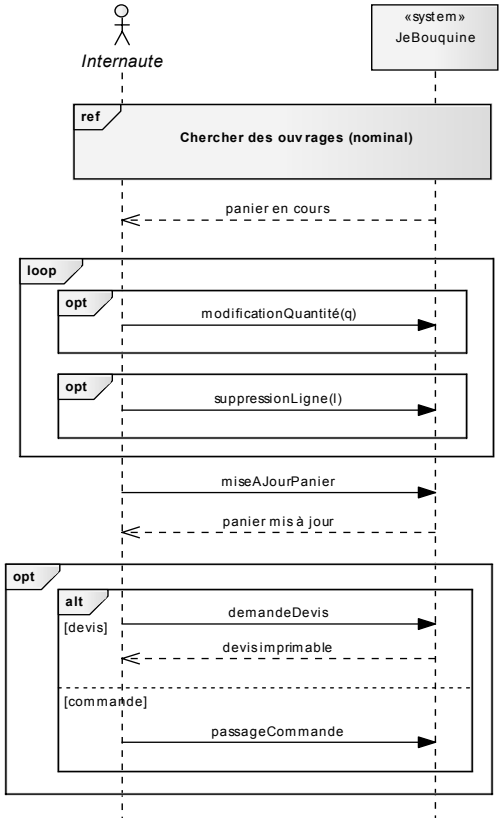


Figure 4–15
DSS complété de Gérer son panier

Effectuer une commande

Rappelons que pour passer une commande, l'internaute doit maintenant s'authentifier s'il est déjà client, ou bien créer un compte client s'il ne l'était pas. Nous pouvons également représenter la précondition concernant le panier qui ne doit pas être vide. Le début du DSS est donné sur la figure 4-16.

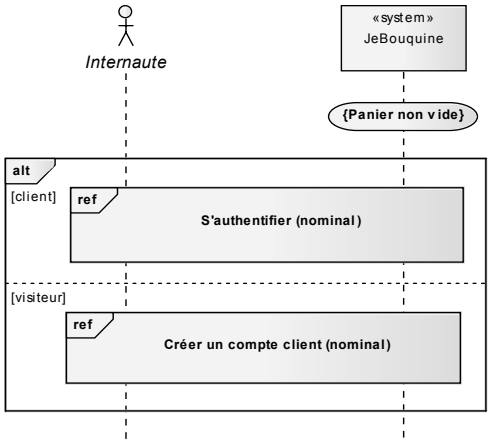


Figure 4–16
Début du DSS de Effectuer une commande

Nous allons ensuite décrire le passage de commande proprement dit, tel que nous l'avions explicité dans la description textuelle. Notez le symbole de l'état sur la ligne de vie de l'`Internaute` indiquant qu'il est devenu `Client` dans tous les cas pour pouvoir exécuter la suite du cas d'utilisation (voir figure 4-17).

Le diagramme atteint ses limites en terme de complexité et il ne serait pas réaliste de vouloir ajouter ici des alternatives au cas nominal en superposant des fragments d'interaction avec le mot-clé alt.

Notez également que nous avons modifié l'ordre des messages par rapport à la description textuelle du cas d'utilisation. En effet, le retour de confirmation de commande vers le client termine la transaction de validation de commande. Il est donc logique de le positionner après l'enregistrement de la commande et son envoi à destination du `Service clients`. La fin de la description du cas d'utilisation devient donc :

5 Le `Paiement sécurisé` autorise la transaction.

6 Le `Système` envoie la commande validée au `Service clients` de jeBouquine.

7 Le `Système` enregistre la commande.

8 Le `Système` confirme la prise de commande à l'`Internaute`.

MÉTHODE **Intérêt du DSS lorsqu'il y a des acteurs secondaires**

L'intérêt de la vue graphique du DSS se révèle pleinement quand il existe des acteurs secondaires. Ici, l'interaction entre le site web, le système externe de paiement sécurisé et le `Service clients` apparaît clairement avec son positionnement précis dans la séquence des messages.

B.A.-BA Message synchrone ou asynchrone

La notation avec la flèche pleine représente des messages synchrones, c'est-à-dire des appels où l'émetteur se bloque en attente de la réponse. Dans le cas contraire, on parle de message asynchrone, et on le représente par une flèche évidée. C'est le cas sur la figure 4-17 pour le message à destination du `Service clients` (typiquement un courrier électronique).

B.A.-BA Flèche qui reboucle

La flèche qui reboucle sur le système (`enregistrementCommande`) permet de représenter graphiquement un comportement interne majeur sur lequel on veut mettre l'accent. Il ne faut cependant pas en abuser car ce n'est pas l'objectif premier de ce type de diagramme dit d'interaction.

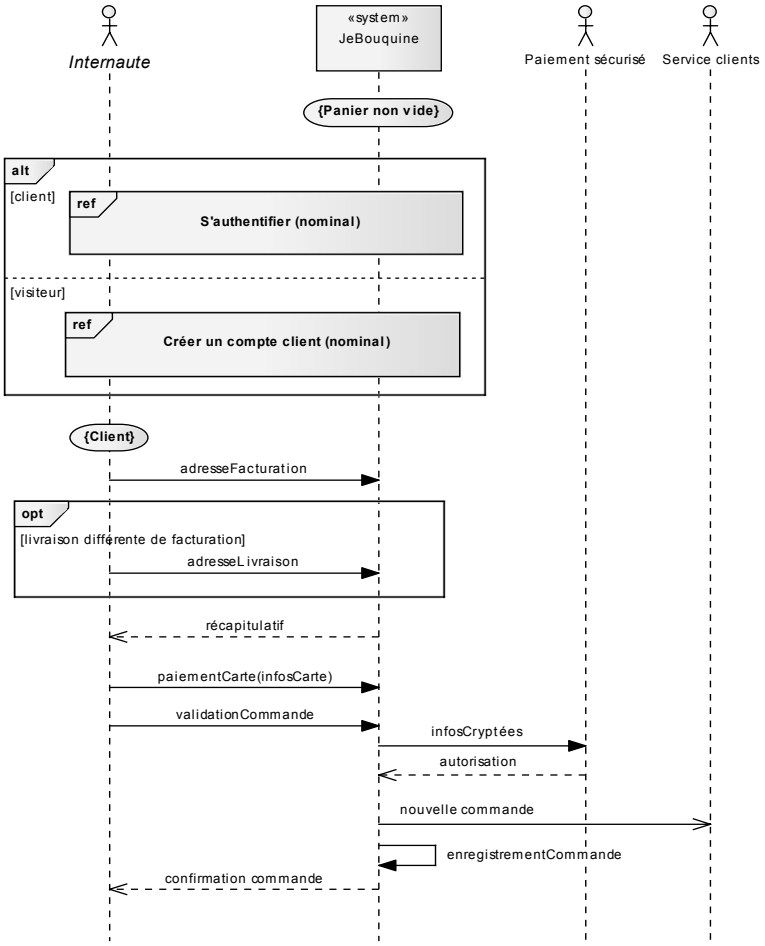


Figure 4–17
DSS nominal complété de Effectuer une commande

Maintenir le catalogue

Les systèmes externes alimentent de façon périodique et asynchrone le système `JeBouquine`, qui se met à jour en cache. Ensuite, le `Libraire` valide la mise à jour et le nouveau catalogue est disponible.

On pourrait facilement ajouter la modification optionnelle des informations éditoriales en fin de scénario pour compléter le diagramme, comme sur la figure 4-19. Notez également les contraintes temporelles précisant quand sont réalisées l'alimentation externe périodique et la mise à jour automatique.

UML 2
Cadre de diagramme : tag et nom

Notez que depuis UML 2.0, un diagramme peut être inclus dans un cadre accueillant tout le contenu graphique. Le cadre a pour intitulé le nom du diagramme et établit sa portée. C'est un rectangle avec un petit pentagone (appelé *tag* de nom), placé dans l'angle supérieur gauche, qui contient le type du diagramme et son nom. Le cadre n'est cependant pas obligatoire lorsque le contexte est clair. Dans le diagramme 4–19, « sd » signifie *sequence diagram*.

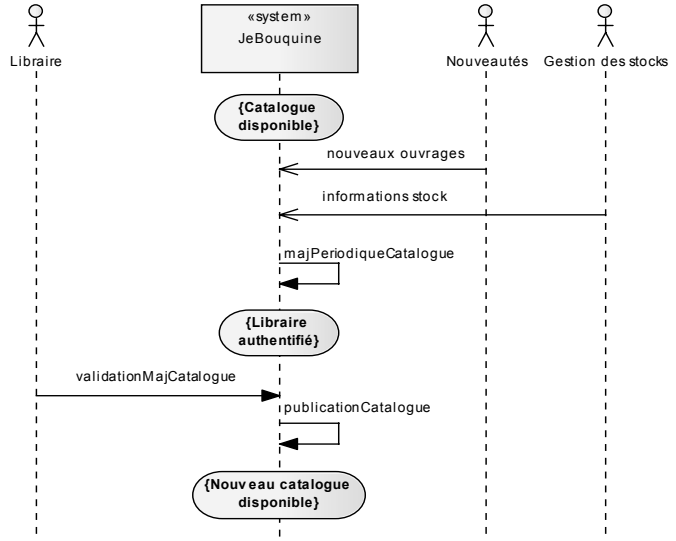


Figure 4–18
Premier DSS de Maintenir le catalogue

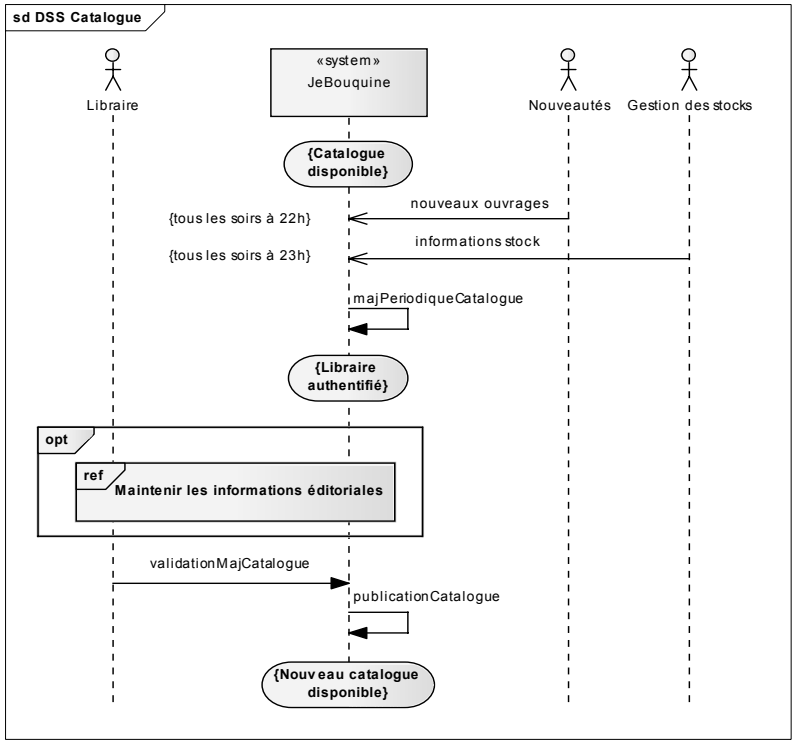


Figure 4–19
DSS complété de Maintenir le catalogue

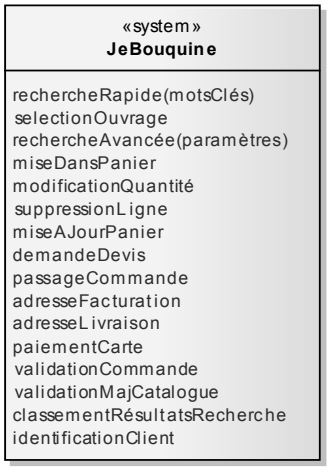
Opérations système

Les événements système envoyés par les acteurs au site web jeBouquine.com déclenchent des traitements internes que nous appellerons opérations système.

L'ensemble des opérations système de tous les cas d'utilisation définit l'interface publique du système, qui visualise le système comme une entité unique, boîte noire offrant des services. En UML, le système pris dans son ensemble peut être représenté par une classe, avec le mot-clé «`system`», comme nous l'avons déjà fait dans les diagrammes de séquence précédents.

À la suite de la création des différents DSS et en enrichissant cette description préliminaire avec les extensions des cas d'utilisation, nous obtenons la liste d'opérations système de la figure 4-20.

Figure 4–20
Opérations système du site web d'après les cas d'utilisation décrits



B.A.-BA Interface

Une interface est un ensemble d'opérations abstraites (sans algorithme) constituant une sorte de contrat qui devra être réalisé. EA dessine les interfaces et leurs opérations en italiques pour insister sur leur caractère abstrait (c'est-à-dire, non instanciable). Graphiquement, une interface est soit représentée comme un rectangle avec un mot-clé « `interface` », soit un cercle dans la notation condensée.

B.A.-BA Réalisation

La relation de réalisation entre une classe et une interface se dessine comme un héritage en pointillés. On parle aussi d'héritage d'interface, par opposition à l'héritage d'implémentation. Si l'interface est représentée par un simple cercle, la réalisation devient un simple trait (voir figure 4-22).

Cette liste n'est bien sûr pas exhaustive, car nous n'avons pas encore décrit dans le détail tous les cas d'utilisation. Elle sera donc progressivement complétée au fur et à mesure des itérations successives. On commence cependant à voir apparaître les fonctionnalités majeures du site web, à un niveau de détail proche de ce qui sera disponible dans l'IHM (Interface Homme-Machine).

Il serait également possible de commencer à définir des interfaces, pour découper l'ensemble des opérations en groupes cohérents, comme représenté sur la figure 4-21.

On peut alors ajouter un lien direct avec les cas d'utilisation, comme représenté par les dépendances stéréotypées « trace » sur la figure 4-22.

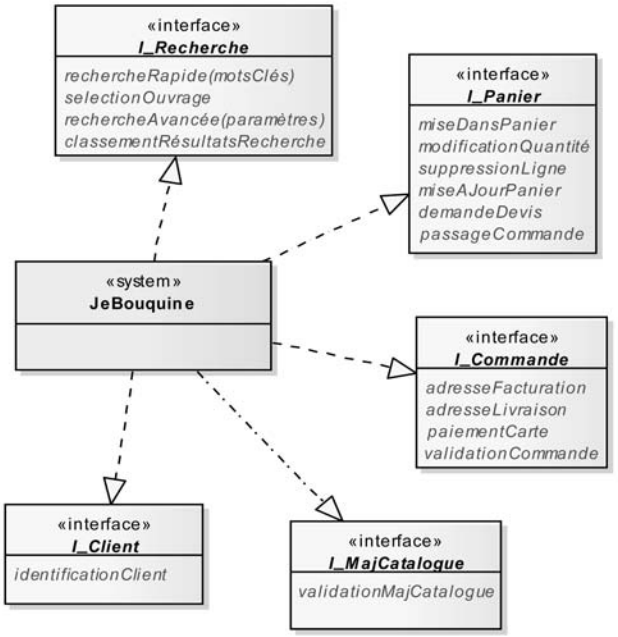


Figure 4–21
Opérations système du site web structurées en interfaces

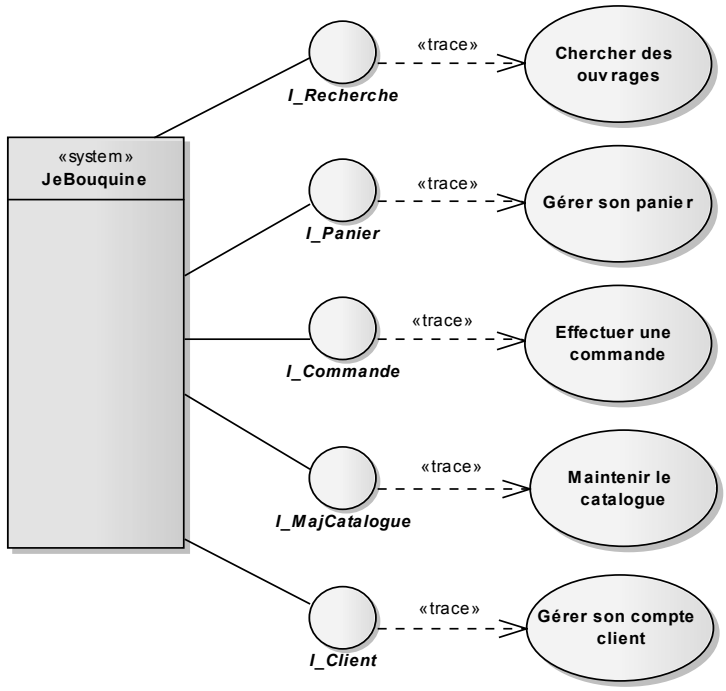
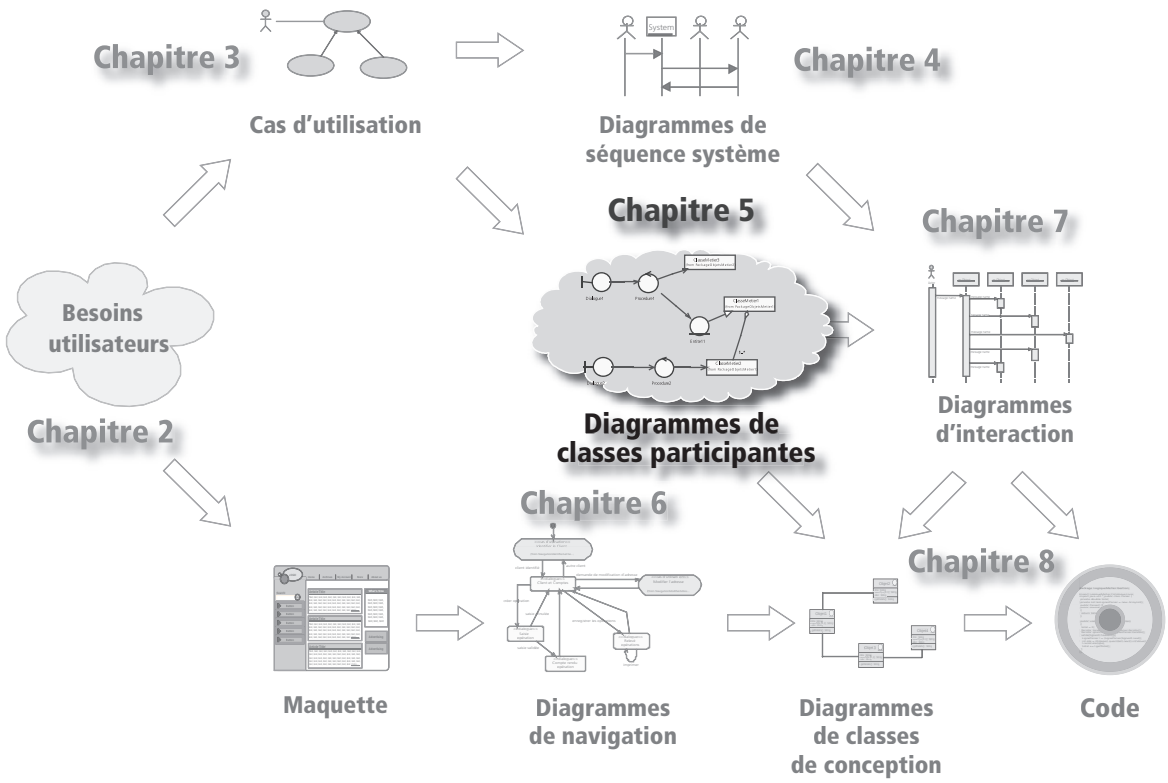


Figure 4–22
Opérations système structurées en interfaces et reliées aux UC

chapitre 5


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Chapitre 7
Besoins utilisateurs
Chapitre 2
Diagrammes de classes participantes
Diagrammes d'interaction
Chapitre 6
Chapitre 8
Maquette
Diagrammes de navigation
Diagrammes de classes de conception
Code

Réalisation des cas d'utilisation : classes d'analyse

Les classes, les associations et les attributs sont les concepts UML fondamentaux pour l'analyse orientée objet. Nous apprendrons à identifier les concepts du domaine à partir de l'expression initiale des besoins de notre étude de cas. Nous verrons comment ajouter des attributs et des associations à ces concepts ainsi que les représentations graphiques UML associées. Nous distinguerons ensuite trois types de classes d'analyse : les « dialogues » qui représentent les moyens d'interaction avec le système, les « contrôles » qui contiennent la logique applicative et les « entités » qui sont les objets métier manipulés. Nous représenterons le résultat de cette investigation dans un diagramme de classes UML que nous appellerons diagramme de classes participantes.
Nous aborderons enfin le diagramme d'états, qui pemet de représenter le cycle de vie d'un objet générique d'une classe particulière au fil de ses interactions, dans tous les cas possibles.

SOMMAIRE

MOTS-CLÉS

- Classe
- Objet
- Association
- Attribut
- Diagramme de classes
- Diagramme d'états

Démarche

Rappelons le positionnement de cette activité de modélisation du domaine par rapport à l'ensemble du processus décrit au chapitre 1.

L'expression préliminaire des besoins donne lieu assez directement à une modélisation par les cas d'utilisation (comme nous l'avons expliqué au chapitre 3) et à une maquette d'IHM. Il s'agit là de descriptions fonctionnelles qui vont nous servir en particulier pour les tests de recette à la fin du projet, mais aussi de point d'entrée pour la description dynamique des scénarios d'exécution du futur système.

En revanche, la conception objet demande principalement une description structurelle, statique, du système à réaliser sous forme d'un ensemble de classes logicielles, éventuellement regroupées en packages. Les meilleures classes candidates sont celles issues d'une analyse du domaine (souvent appelée aussi analyse métier), c'est-à-dire des concepts manipulés par les experts du domaine. Pour passer en douceur à la conception, il nous faut encore identifier les principales classes d'IHM ainsi que celles qui décrivent la cinématique de l'application.

Identification des concepts du domaine

L'étape typiquement orientée objet de l'analyse est la décomposition d'un domaine d'intérêt en classes conceptuelles représentant les entités

B.A.-BA Classe

Une classe représente la description abstraite d'un ensemble d'objets possédant les mêmes caractéristiques. On peut parler également de type. Exemples : la classe `Voiture`, la classe `Personne`.

B.A.-BA Objet

Un objet est une entité aux frontières bien définies, possédant une identité et encapsulant un état et un comportement. Un objet est une instance (ou occurrence) d'une classe.
Exemple : `Pascal Roques` est un objet instance de la classe `Personne`. `La voiture de Pascal Roques` est une instance de la classe `Voiture`.

B.A.-BA Attribut

Un attribut représente un type d'information contenu dans une classe. Exemples : `vitesse courante`, `cylindrée`, `numéro d'immatriculation`, etc. sont des attributs de la classe `Voiture`.

B.A.-BA Association

Une association représente une relation sémantique durable entre deux classes.
Exemple : Une personne peut posséder des voitures. La relation `possède` est une association entre les classes `Personne` et `Voiture`.
Attention : même si le verbe qui nomme une association semble privilégier un sens de lecture, une association entre concepts dans un modèle du domaine est par défaut bidirectionnelle. Donc implicitement, l'exemple précédent inclut également le fait qu'une voiture est possédée par une personne.

B.A.-BA Opération

Une opération représente un élément de comportement (un service) contenu dans une classe. Nous ajouterons plutôt les opérations en conception objet, car cela fait partie des choix d'attribution des responsabilités aux objets.

significatives de ce domaine. Il s'agit simplement de créer une représentation visuelle des objets du monde réel dans un domaine donné.

Si l'on emploie la notation UML, il s'agit d'un ensemble de diagrammes de classes dans lesquels on fait figurer les éléments suivants :

- les classes conceptuelles ou les objets du domaine ;
- les associations entre classes conceptuelles ;
- les attributs des classes conceptuelles.

Comment identifier les concepts du domaine ? Plutôt que de partir à l'aveugle et nous heurter à la taille du problème à résoudre, nous allons prendre les cas d'utilisation un par un et nous poser pour chacun la question suivante : quels sont les concepts métier qui participent à ce cas d'utilisation ?

Par exemple, pour le cas d'utilisation `Chercher des ouvrages`, nous identifions les concepts fondamentaux suivants :

- ouvrage,
- auteur,
- éditeur.

De même, pour le cas d'utilisation `Gérer son panier`, nous identifions :

- panier,
- livre.

Enfin, pour le cas d'utilisation `Effectuer une commande`, nous identifions :

- commande,
- panier,
- client,
- carte de crédit.

ATTENTION Vocabulaire métier

Un modèle du domaine est une sorte de carte des concepts d'un domaine. Il faut donc absolument utiliser le vocabulaire de ce dernier pour nommer les classes conceptuelles et leurs attributs. Un cartographe ne change pas le nom des villes !
Méfions-nous également des synonymes ! La modélisation ne peut pas tolérer l'utilisation de plusieurs noms pour le même concept. Il faut donc en choisir un et s'y tenir. Par exemple ici, `ouvrage`, `livre` et `bouquin` sont synonymes. Nous garderons `livre` dans le modèle.

ATTENTION Attribut ou concept ?

L'erreur la plus courante lors de la création d'un modèle d'analyse consiste à représenter quelque chose comme un attribut alors que ce devrait être un concept à part entière.
Un bon critère à appliquer peut s'énoncer de la façon suivante : si l'on ne peut demander à une entité que sa valeur, il s'agit d'un simple attribut, mais si l'on peut lui poser plusieurs questions, il s'agit plutôt d'un concept qui possède à son tour plusieurs attributs, ainsi que des liens avec d'autres objets.
Exemple : pour un livre, date de parution et langue sont des attributs, alors qu'auteur est un concept à part entière, car on peut lui demander son nom, son prénom, mais aussi quels sont les livres qu'il a écrits.

Ajout des associations et des attributs

Une fois que l'on a identifié les concepts fondamentaux, il est utile d'ajouter :

- les associations nécessaires pour prendre en compte les relations qu'il est fondamental de mémoriser ;
- les attributs nécessaires pour répondre aux besoins d'information.

Reprenons donc les quatre cas d'utilisation majeurs un par un.

Chercher des ouvrages

Nous avons vu dans l'expression préliminaire des besoins (voir chapitre 2) que l'internaute saisira un critère (`titre`, `auteur`, `ISBN`, etc.)

B.A.-BA Multiplicité

Aux deux extrémités d'une association, on doit faire figurer une indication de multiplicité. Elle spécifie sous la forme d'un intervalle le nombre d'objets qui peuvent participer à une relation avec un objet de l'autre classe dans le cadre d'une association.

Exemple : une personne peut posséder plusieurs voitures (entre zéro et un nombre quelconque) ; une voiture est possédée par une seule personne.

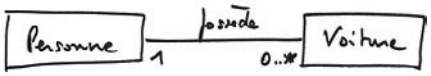


Figure 5–1
Exemples de multiplicités d'association

ou même plusieurs critères à la fois. Les copies d'écran des figures 2-3 et 2-5 nous font trouver également les attributs `prix`, `date de parution`, `éditeur`, `langue`, `sous-titre`, `nombre de pages`.

Un premier diagramme représentant toutes ces informations se trouve sur la figure 5-2.

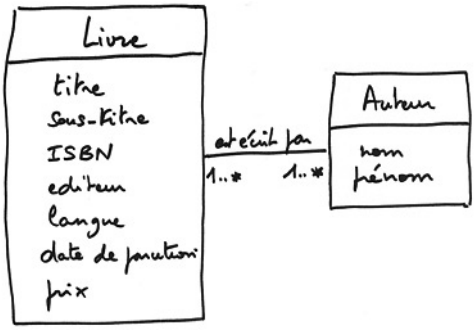


Figure 5–2
Concepts et attributs liés à la recherche d'ouvrages (premier jet)

Nous pouvons déjà faire plusieurs remarques sur la figure 5-2.

- L'attribut `editeur` devrait plutôt être modélisé comme un concept : un éditeur a un nom, mais aussi une nationalité, et il est relié à de nombreux livres. Il s'agit bien d'un concept à part entière dans le domaine, au même titre qu'un auteur.
- L'attribut `sous-titre` est optionnel : tous les livres n'ont pas de sous-titre, alors qu'ils ont un titre, une langue, etc. UML indique ce caractère optionnel en ajoutant une multiplicité `[0..1]` derrière l'attribut, comme indiqué sur la figure 5-3.

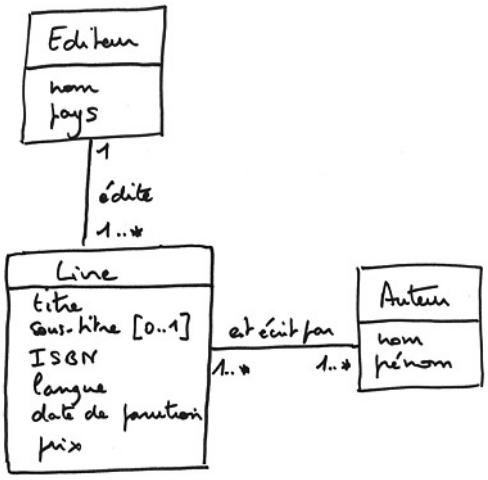


Figure 5–3
Concepts et attributs liés à la recherche d'ouvrages

Gérer son panier

Le `Panier` est bien un concept du domaine, car dans les librairies réelles, le client remplit également un panier avant de passer à la caisse. Le panier est simplement un conteneur des livres sélectionnés par le client. Une première représentation de ces concepts est donnée en figure 5-4.

CONCEPT AVANCÉ **Agrégation**

Notez l'utilisation du losange vide du côté de la classe `Panier` sur la figure 5-4, indiquant une relation d'agrégation. Une agrégation est un cas particulier d'association non symétrique exprimant une relation de contenance. Les agrégations n'ont pas besoin d'être nommées : implicitement elles signifient « contient », « est composé de ».
Au passage, notez que si un panier contient un nombre quelconque de livres (`0..*`), un livre peut également faire partie d'un nombre quelconque de paniers simultanément.

Figure 5–4
Concepts liés à la gestion du panier (premier jet)

Nous devons prendre en compte le fait que le client peut choisir plusieurs exemplaires du même livre et que nous voulons le total du panier. Pour cela, nous allons utiliser deux éléments UML supplémentaires du diagramme de classes : l'attribut dérivé et la classe d'association.

CONCEPT AVANCÉ **Attribut dérivé**

Un attribut dérivé est un attribut dont la valeur peut être déduite d'autres informations disponibles dans le modèle, par exemple d'autres attributs de la même classe, ou de classes en association. Cet attribut, qui pourrait donc être considéré comme redondant, est néanmoins gardé par l'analyste s'il correspond à un concept important aux yeux de l'expert métier.
Exemple : un attribut dérivé très classique est l'`âge` d'une personne, dérivé de l'attribut `date de naissance` appartenant à la même classe. Il est noté en UML avec un « `/` » avant son nom, comme illustré sur la figure 5-5.

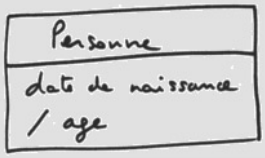


Figure 5–5 Exemple d'attribut dérivé

Dans notre cas, le prix du panier est calculable simplement à partir du prix des livres sélectionnés, ce qui donne un attribut dérivé `/total` représenté sur la figure 5-6.

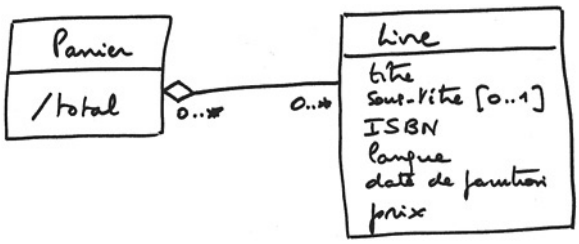


Figure 5–6
Concepts liés à la gestion du panier (deuxième jet)

Comment exprimer le fait que le panier peut contenir plusieurs exemplaires du même livre, comme illustré sur la figure 2-6 par la colonne `quantité` ?

Deux solutions sont possibles. La première consiste à ajouter un concept intermédiaire qui correspond à une ligne du panier et qui concerne donc un seul livre, mais avec un attribut `quantité`, comme sur la figure 5-7.

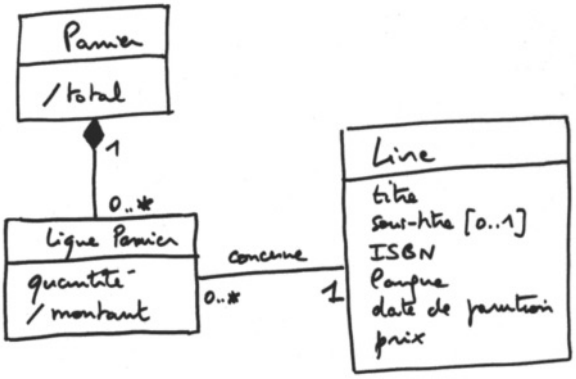


Figure 5–7
Concepts liés à la gestion du panier (troisième jet)

CONCEPT AVANCÉ Composition

Notez l'utilisation du losange plein du côté de la classe `Panier` sur la figure 5-7, indiquant une relation de composition. Une composition est une agrégation plus forte impliquant que :

- Une ligne panier ne peut appartenir qu'à un seul panier (agrégation non partagée).
- La destruction du panier entraîne la destruction de toutes ses lignes (le composite est responsable du cycle de vie des parties).

La deuxième solution, plus élégante mais plus sophistiquée, consiste à introduire une classe d'association portant l'attribut `quantité`, comme représenté sur la figure 5-8.

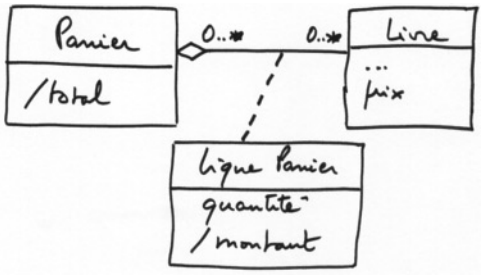


Figure 5–8
Solution alternative pour modéliser la ligne de panier

CONCEPT AVANCÉ Classe d'association

Il s'agit d'une association promue au rang de classe. Elle possède tout à la fois les caractéristiques d'une association et celles d'une classe et peut donc porter des attributs qui se valorisent pour chaque lien.
Exemple : chaque lien entre un panier et un livre peut porter une valeur d'attribut `quantité` représentant une donnée de la ligne.

Nous retiendrons dans la suite de l'étude la première solution afin de ne pas compliquer le modèle.

Une dernière remarque cependant : l'attribut `quantité` est positionné par défaut à « `1` ». Cette valeur initiale significative (car non nulle) peut être représentée en UML comme indiqué sur la figure 5-9.

Profitons-en pour ajouter dans le panier le nombre d'articles, somme des quantités des différentes lignes.

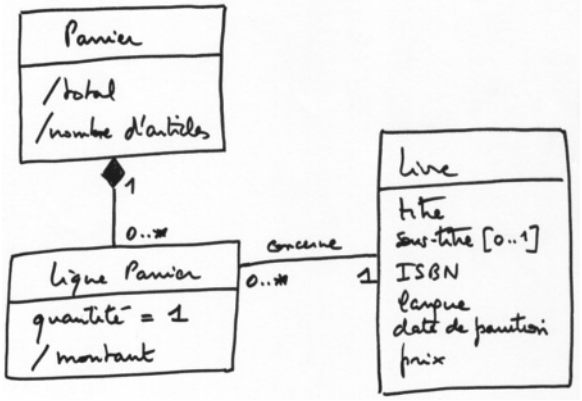


Figure 5–9
Concepts liés à la gestion du panier (solution retenue)

Effectuer une commande

Une fois que l'internaute a un panier non vide, il peut passer sa commande. Il lui faut pour cela s'identifier en tant que client (ou créer un compte la première fois) puis confirmer ou modifier les informations nécessaires au paiement et à la livraison (voir figure 2-7).

Le concept de `Client` est donc incontournable, ainsi que celui de `Commande`. Une première version du diagramme de classes correspondant est donnée à la figure 5-10.

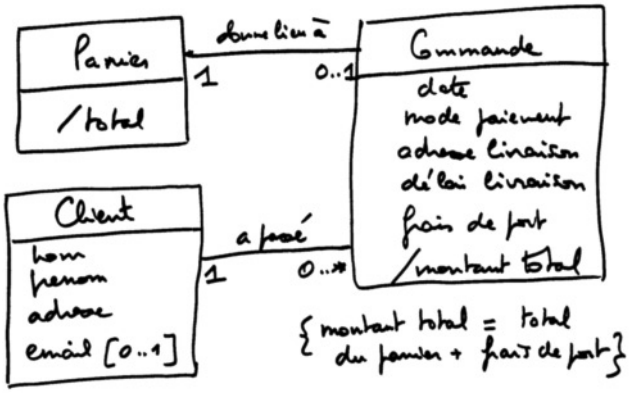


Figure 5–10
Concepts liés à la prise de commande (premier jet)

B.A.-BA Contrainte

Notez l'attribut dérivé `/montant total` dans la classe `Commande`, sur la figure 5-10. La relation permettant de calculer la valeur de cet attribut à partir d'autres éléments du modèle est décrite en UML au moyen d'une contrainte. Une contrainte est simplement une condition entre éléments du modèle qui doit être vérifiée par les éléments concernés. Elle est notée entre accolades `{ }` et peut être insérée au besoin dans une note graphique (le post-it).

Les associations de ce diagramme précisent qu'une commande est forcément liée à un client et à un panier, mais qu'un panier ne donne pas forcément lieu à une commande et qu'un même client peut avoir passé plusieurs commandes.

Nous allons considérer par la suite que le mode de règlement privilégié est la carte bancaire. Les informations concernant la carte sont-elles des attributs de la commande ou du client ? Ni l'un ni l'autre : il s'agit bien d'un concept à part entière, qui possède plusieurs attributs et qui est relié au client, mais aussi à la commande, comme représenté sur la figure 5-11. Notez la multiplicité `0..1` du côté de la carte bancaire : nous aurions plutôt mis `0..*` si nous avions souhaité stocker les informations de plusieurs cartes bancaires pour un même client, mais cela ne fait pas partie des exigences fonctionnelles.

B.A.-BA **Contrainte {ordered}**

La contrainte prédéfinie `{ordered}` ajoute une précision sur une multiplicité supérieure à 1. Les commandes passées par un client sont typiquement ordonnées par date dans le système. Cette précision ajoute une notion de séquence à la collection de commandes reliées à un client, typiquement implémentée en conception par le concept de liste.

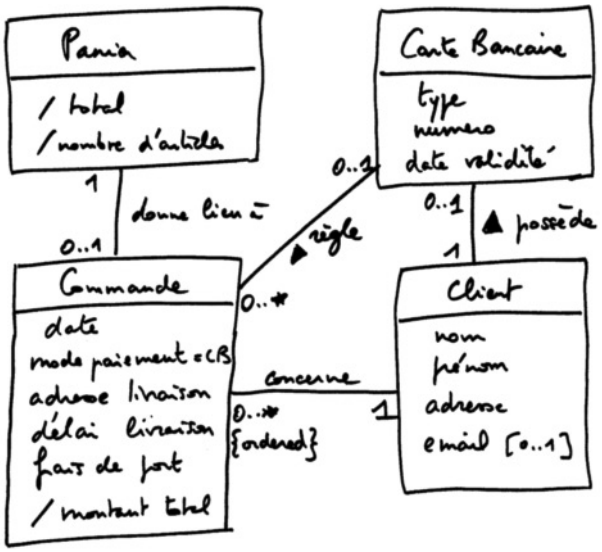


Figure 5–11
Concepts liés à la prise de commande (deuxième version)

B.A.-BA **Sens de lecture des associations (▶)**

Notez les deux bouts de flèches noires sur les associations `possède` et `règle` de la classe `Carte Bancaire`. Ces symboles ne servent qu'à faciliter la compréhension du diagramme en indiquant explicitement le sens de lecture du verbe sur l'association. C'est bien le client qui possède la carte et pas l'inverse.

Maintenir le catalogue

La librairie jeBouquine a déjà ouvert un certain nombre de rayons bien séparés. Les livres sont donc classés en rayons au sein du catalogue, comme illustré sur la figure 5-12.

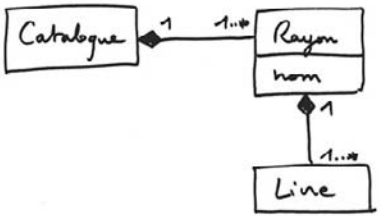


Figure 5–12
Concepts liés à la maintenance du catalogue (premier jet)

Notez que nous pouvons tout à fait omettre les attributs de la classe `Livre` sur un diagramme et les montrer dans un autre. N'oubliez pas qu'un diagramme UML n'est qu'une vue filtrée du modèle global et que le modélisateur a toute latitude pour ne montrer que ce qui lui paraît important.

Nous souhaitons également modéliser la notion de thème. Les livres peuvent appartenir à plusieurs thèmes car ceux-ci ne sont pas forcément disjoints. Par exemple, un livre comme UML pour les bases de données appartiendrait au moins aux thèmes *Technologies objet* et *Bases de données*. En revanche, tout livre doit appartenir à au moins un thème.

Notez qu'un thème peut lui-même se décomposer en sous-thèmes : *Technologies objet* pourrait ainsi se décomposer en *UML*, *Java*, *.Net*, etc. Le concept de thème est introduit à la figure 5-13.

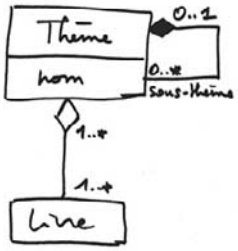


Figure 5–13
Introduction du concept de thème

ATTENTION **Agrégation ou composition ?**

Une des relations de la figure 5-13 est une agrégation, mais pas une composition. En effet, celle entre `Livre` et `Thème` est partagée (multiplicité `1..*` du côté `Thème`) et ne respecte donc pas le premier critère de la composition. De plus, cette relation n'implique pas de responsabilité au niveau du cycle de vie des objets contenus. La destruction d'un thème n'entraîne pas forcément la destruction des livres concernés, mais plutôt un reclassement dans d'autres thèmes.
Notez également la relation de composition qui « reboucle » sur la classe `Thème`. Elle relie ainsi des objets de la même classe, en nommant simplement un des rôles (`sous-thème`). Créer un nouveau concept appelé `SousTheme` serait une erreur de débutant : il s'agit bien du même concept ! Les thèmes de premier niveau ne sont pas inclus dans un thème de niveau supérieur, mais directement dans un rayon, d'où la multiplicité `0..1` à l'extrémité opposée de `sous-thème`.

Finalement, si nous ajoutons les informations liées aux auteurs et aux éditeurs, qui font également partie du catalogue, nous aboutissons à la figure complète 5-14.

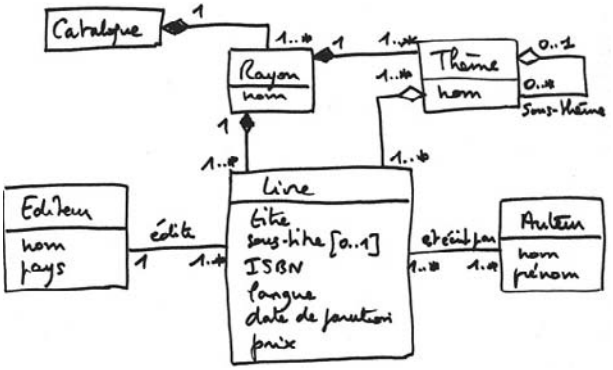


Figure 5–14
Concepts liés à la maintenance du catalogue (deuxième version)

Recherche d'améliorations

Pour améliorer notre modèle, nous allons chercher les possibilités de factorisation en identifiant puis en extrayant les similitudes entre classes (attributs et associations similaires).

Dans notre modèle d'analyse, il n'y a pas de généralisation qui saute aux yeux. Eh oui, un diagramme de classes ne comporte pas forcément de relation d'héritage, même s'il utilise les concepts objets !

B.A.-BA **Super-classe, sous-classe, héritage**

Une super-classe est une classe générale reliée à d'autres classes plus spécialisées (sous-classes) par une relation de généralisation. Les sous-classes « héritent » des propriétés de leur super-classe et peuvent comporter des propriétés spécifiques supplémentaires.
Exemple (voir figure 5-15) : les voitures, les bateaux et les avions sont des moyens de transport. Ils possèdent tous une marque, un modèle, une vitesse, etc. En revanche, seuls les bateaux ont un tirant d'eau et seuls les avions ont une altitude…

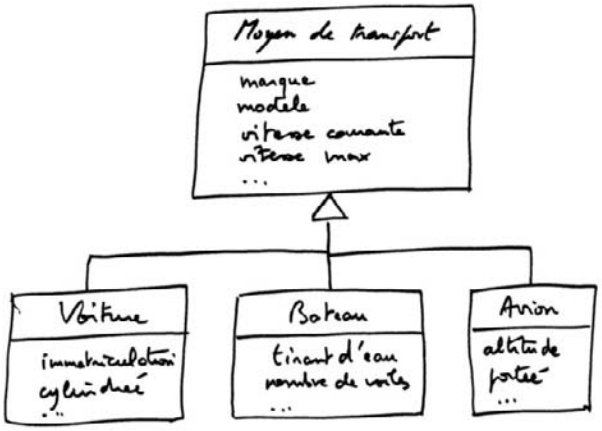


Figure 5–15 Exemple de super-classe et sous-classes

On pourrait bien sûr anticiper dès à présent une diversification du champ d'action de jeBouquine en proposant à la vente des disques et des vidéos ! Dans ce cas, une relation de généralisation permettrait d'identifier une super-classe `Article` et fournirait une solution souple et évolutive, comme illustré sur la figure 5-16.

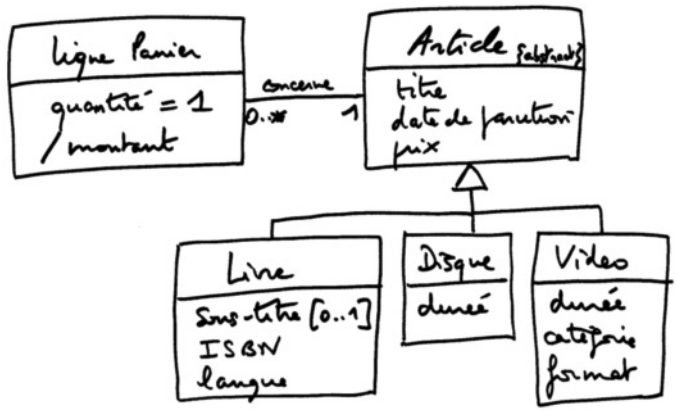


Figure 5–16
Diversification possible du site marchand…

CONCEPT AVANCÉ Classe abstraite

Une classe abstraite est simplement une classe qui ne s'instancie pas directement mais qui représente une pure abstraction afin de factoriser des propriétés. Elle se note en italique, ou suivie de la contrainte `{abstract}`. Le client ne va pas acheter des articles, mais bien des livres, des disques ou des vidéos !

Toutefois, cette diversification ne fait pas vraiment partie des objectifs du site www.jeBouquine.com.

Une factorisation moins évidente à trouver concerne les informations de facturation et de livraison d'une commande. Sur la figure 5-11, nous avions indiqué qu'une commande possède un attribut `adresse livraison` et que le client relié a une `adresse`. En fait, dans le cas d'un cadeau par exemple, l'adresse de livraison doit également comporter le nom et le prénom du destinataire. Cela signifie que les informations de facturation et de livraison sont vraiment similaires, comme on peut le constater d'ailleurs sur la capture d'écran 2-7. Enfin, dans le cas fréquent où l'adresse de facturation est identique à l'adresse de livraison, la seconde est optionnelle (d'où la multiplicité `0..1`).

Toutes ces considérations mous amènent à modifier le diagramme 5-11 comme suit (figure 5-17).

ATTENTION Factorisation par association

Remarquez bien que nous n'avons pas eu besoin d'introduire de super-classe. La classe `Client` n'hérite pas de la classe `Adresse` : un client n'est pas une sorte d'adresse, il est bien plus ! En revanche, il est légitime de dire qu'un client possède une adresse de facturation et qu'une commande possède éventuellement une adresse de livraison. Le nommage de l'extrémité de chaque association (rôle) permet de bien représenter ces deux notions. C'est un bon exemple de factorisation par association plutôt que par relation d'héritage.

Typologie des classes d'analyse

Pour élargir cette première identification des concepts du domaine, nous allons utiliser une catégorisation des classes d'analyse qui a été proposée par I. Jacobson et popularisée ensuite par le RUP. Les classes d'analyse qu'ils préconisent se répartissent en trois catégories :

- Celles qui permettent les interactions entre le site web et ses utilisateurs sont qualifiées de dialogues. Il s'agit typiquement des écrans proposés à l'utilisateur : les formulaires de saisie, les résultats de recherche, etc.

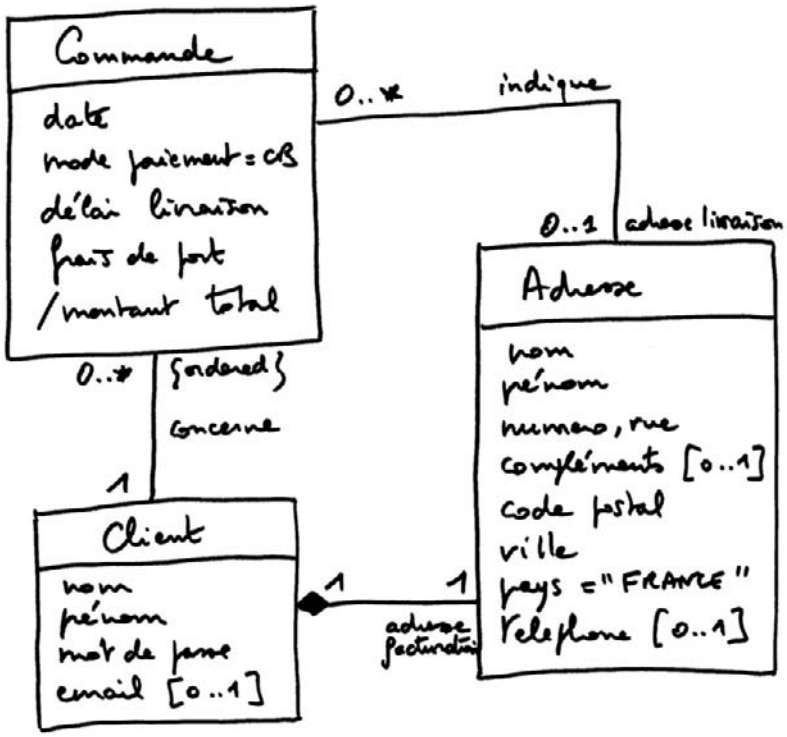


Figure 5–17
Factorisation du concept d'adresse

Ces classes proviennent directement de l'analyse de la maquette. Il y a au moins un dialogue pour chaque paire acteur-cas d'utilisation. Ce dialogue peut avoir des objets subalternes auxquels il délègue une partie de ses responsabilités. En général, les dialogues vivent seulement le temps durant lequel le cas d'utilisation est concerné.

- Les classes qui contiennent la cinématique de l'application sont appelées contrôles. Elles font la transition entre les dialogues et les concepts du domaine, en permettant aux écrans de manipuler des informations détenues par des objets métier. Elles contiennent les règles applicatives et les isolent à la fois des objets d'interface et des données persistantes. Les contrôles ne donnent pas forcément lieu à de vrais objets de conception, mais assurent que nous n'oublions pas de fonctionnalités ou de comportements requis par les cas d'utilisation.
- Celles qui représentent les concepts métier sont qualifiées d'entités. C'est elles que nous avons appris à identifier au début de ce chapitre, cas d'utilisation par cas d'utilisation. Elles sont très souvent persistantes, c'est-à-dire qu'elles vont survivre à l'exécution d'un cas d'utilisation particulier et qu'elles permettront à des données et des relations d'être stockées dans des fichiers ou des bases de données.

Diagramme de classes participantes (DCP)

Le point névralgique de notre démarche s'appelle le diagramme de classes participantes. Il s'agit de diagrammes de classes UML qui décrivent, cas d'utilisation par cas d'utilisation, les trois principales classes d'analyse et leurs relations.

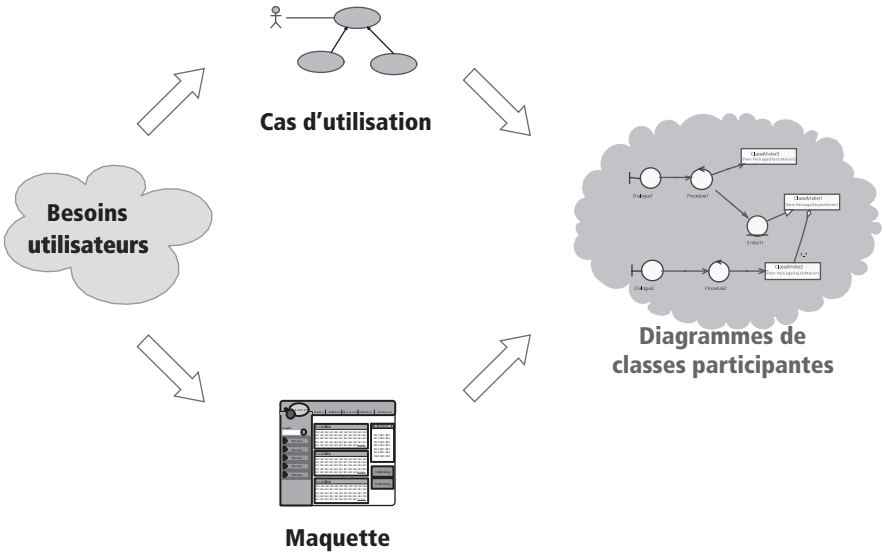


Figure 5–18 Les diagrammes de classes participantes dans la démarche

Un avantage important de cette technique pour le chef de projet consiste en la possibilité de découper le travail de son équipe d'analystes suivant les différents cas d'utilisation, plutôt que de vouloir tout traiter d'un bloc. Les diagrammes de classes participantes sont donc particulièrement importants car ils font la jonction entre les cas d'utilisation, la maquette et les diagrammes de conception logicielle (diagrammes d'interaction et diagrammes de classes).

Pour compléter ce travail d'identification, nous allons ajouter des attributs et des opérations dans les classes d'analyse, ainsi que des associations entre elles.

- Les entités vont seulement posséder des attributs. Ces attributs représentent en général des informations persistantes de l'application.
- Les contrôles vont seulement posséder des opérations. Ces opérations montrent la logique de l'application, les règles transverses à plusieurs entités, bref les comportements du système informatique. Il y a souvent un seul contrôle par cas d'utilisation, mais il peut également y en avoir plusieurs, en fonction du nombre et de la cohérence des comportements associés.

Méthode Pourquoi un modèle objet d'analyse

Les tenants du Processus Unifié (UP) considèrent le modèle objet d'analyse comme optionnel. Nous avons choisi de l'intégrer dans notre démarche car il fait à peu de frais le lien entre l'expression des besoins et le modèle de conception.
Cela permet également de contrôler la validité du texte de nos cas d'utilisation par rapport au modèle d'analyse. Avons-nous toutes les classes d'analyse qu'il nous faut pour chacun des cas d'utilisation ? Nous allons certainement trouver des concepts du domaine (entités) que nous avions oubliés ainsi que des classes dialogues et contrôles qui faciliteront la transition avec la conception.

Méthode Domain-Driven Design (DDD)

Il est intéressant de noter qu'une approche très à la mode ces derniers temps s'appelle *Domain-Driven Design* ! Il s'agit de (re)mettre le modèle du domaine au centre du projet de développement, comme expliqué en détail par Eric Evans dans son livre sur le sujet.

- 🕮 *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Eric Evans, Addison-Wesley, 2003.
- ▸ http://www.domaindrivendesign.org/

Figure 5–19 Entité

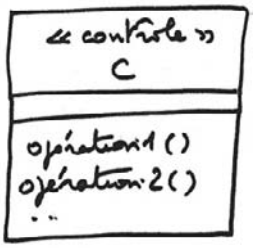


Figure 5–20 Contrôle

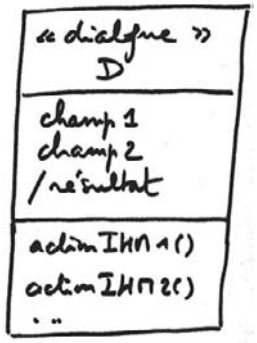


Figure 5–21 Dialogue

- Les dialogues vont posséder des attributs et des opérations. Les attributs représenteront des champs de saisie ou des résultats. Les résultats seront distingués en utilisant la notation de l'attribut dérivé. Les opérations représenteront des actions de l'utilisateur sur l'IHM.

Nous allons également ajouter des associations entre les classes d'analyse, mais en respectant des règles assez strictes :

- Les dialogues ne peuvent être reliés qu'aux contrôles ou à d'autres dialogues, mais pas directement aux entités.
- Les entités ne peuvent être reliées qu'aux contrôles ou à d'autres entités.
- Les contrôles ont accès à tous les types de classes, y compris d'autres contrôles.

Enfin, nous ajouterons les acteurs sur les diagrammes en respectant la règle suivante : un acteur ne peut être lié qu'à un dialogue. Un exemple simple de DCP (diagramme de classes participantes) est donné à la figure 5-22.

MÉTHODE Attribution des responsabilités

Le travail détaillé d'attribution des responsabilités aux objets sous forme de méthodes positionnées dans des classes d'implémentation se fera en conception, en tenant compte de l'architecture.

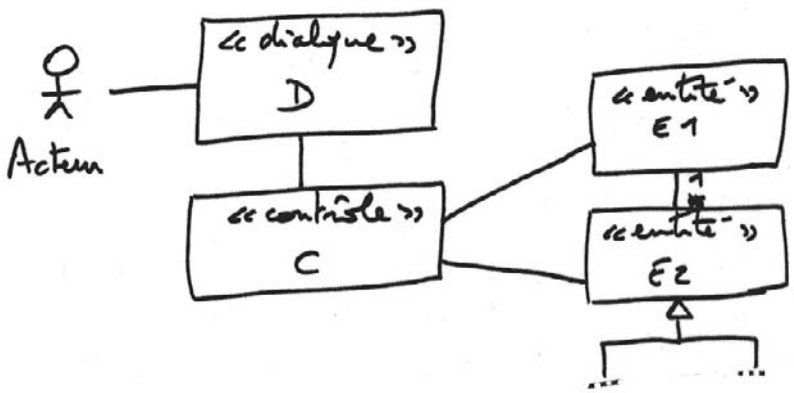


Figure 5–22
Exemple de diagramme de classes participantes

Classes d'analyse participantes des cas d'utilisation du site web

Pour illustrer notre démarche, nous allons identifier les classes d'analyse participantes des principaux cas d'utilisation du site web. Les entités ayant déjà été répertoriées dans la section traitant des concepts du domaine, nous allons donc ajouter les dialogues et les contrôles et réaliser les DCP correspondants.

Maintenir le catalogue

Le libraire veut contrôler la mise à jour automatique du catalogue des ouvrages présentés sur le site web. Pour cela, il va pouvoir valider le catalogue, ou modifier des informations erronées sur certains ouvrages. Il doit également être en mesure de modifier l'organisation générale du catalogue en créant de nouveaux thèmes, etc.

Nous supposons que la maquette nous a montré trois écrans principaux :

- l'écran d'organisation du catalogue (dialogue `OrganisationCatalogue`) à partir duquel on crée de nouveaux thèmes, de nouveaux rayons, etc. ;
- l'écran de gestion des mises à jour (dialogue `GestionMiseAJour`) qui parcourt les informations modifiées automatiquement et valide le catalogue ;
- l'écran de gestion des infos détaillées d'un livre (dialogue `GestionDétailLivre`) permettant de modifier le prix, la disponibilité, etc. d'un ouvrage particulier.

Les comportements correspondant à toutes ces fonctionnalités ont été découpés en deux classes contrôles afin de distinguer ce qui relève du niveau global du catalogue (contrôle `CtrCatalogue`) et ce qui concerne un ouvrage particulier (contrôle `CtrLivre`). Notez que nous ne sommes rentrés dans le détail ni des paramètres ni du type de retour (d'où le `void` ajouté par l'outil Enterprise Architect) au niveau des opérations.

Ensuite, nous avons fait figurer toutes les entités manipulées (sans les associations pour ne pas surcharger inutilement). Le diagramme global ainsi obtenu est montré sur la figure 5-23.

Dans ce DCP, les opérations des classes dialogues et contrôles sont très similaires. On peut noter cependant quelques actions purement d'IHM comme la navigation entre pages, ou l'affichage du détail d'un livre.

En général, les dialogues sont reliés aux contrôles qui manipulent les entités correspondantes.

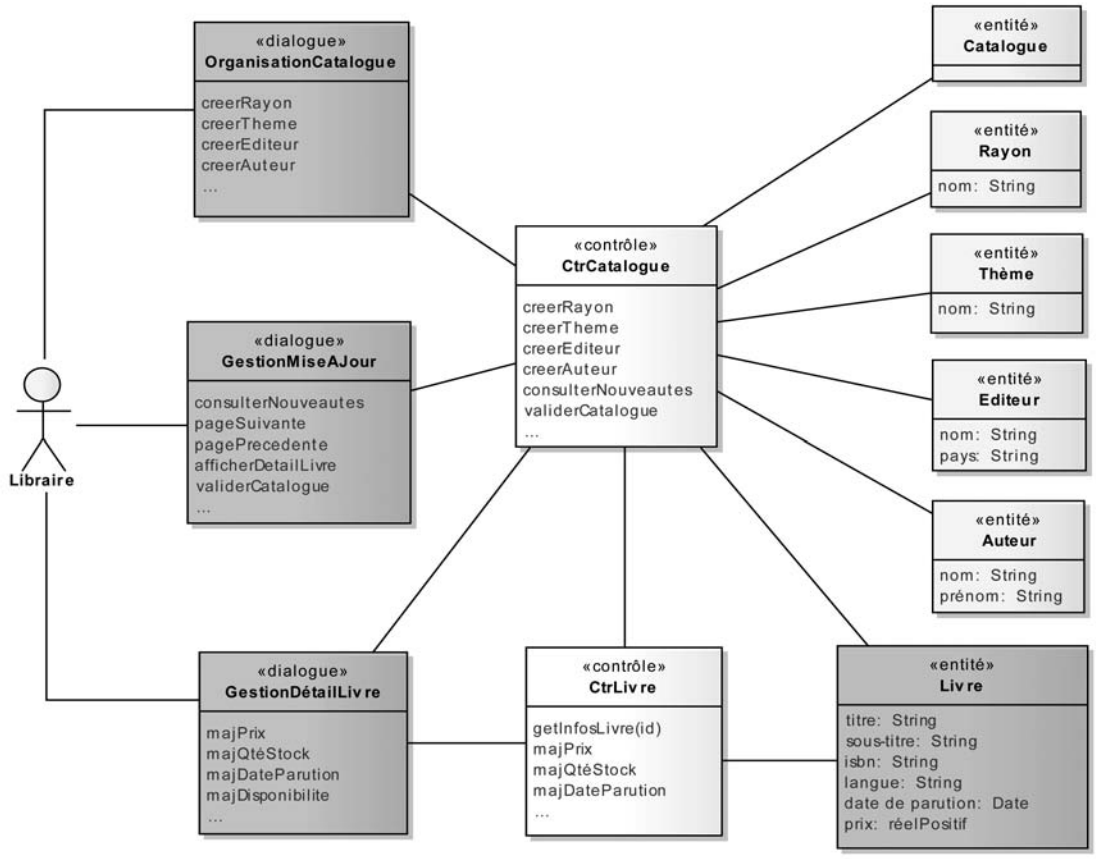


Figure 5–23
DCP de Maintenir le Catalogue

Remarquez ici que `CtrCatalogue` est relié au dialogue `GestionDétailLivre`, car c'est le contrôle global qui initialise l'écran de mise à jour. Celui-ci dialogue ensuite avec son contrôle particulier (`CtrLivre`).

Chercher des ouvrages

L'internaute veut trouver le plus rapidement possible un ouvrage recherché dans l'ensemble du catalogue. Il veut également pouvoir consulter la fiche détaillée d'un ouvrage particulier avant de le mettre dans son panier.

Notez le stéréotype «`optionnel`» devant l'attribut `sous-titre` de l'entité `Livre`. C'est une solution alternative à l'indication de multiplicité `[0..1]` du diagramme 5-24.

Nous supposons là encore que la maquette nous a montré trois écrans principaux :

- les écrans de recherche rapide et de recherche avancée ;
- le résultat de recherche, sur une ou plusieurs page(s), qui permet d'accéder à la fiche détaillée d'un ouvrage particulier.

Les comportements correspondant à ces fonctionnalités ont été séparés en deux classes contrôles afin de distinguer ce qui relève de la recherche au niveau global du catalogue et ce qui concerne l'obtention d'informations détaillées sur un ouvrage particulier. Le diagramme ainsi obtenu est montré sur la figure 5-24.

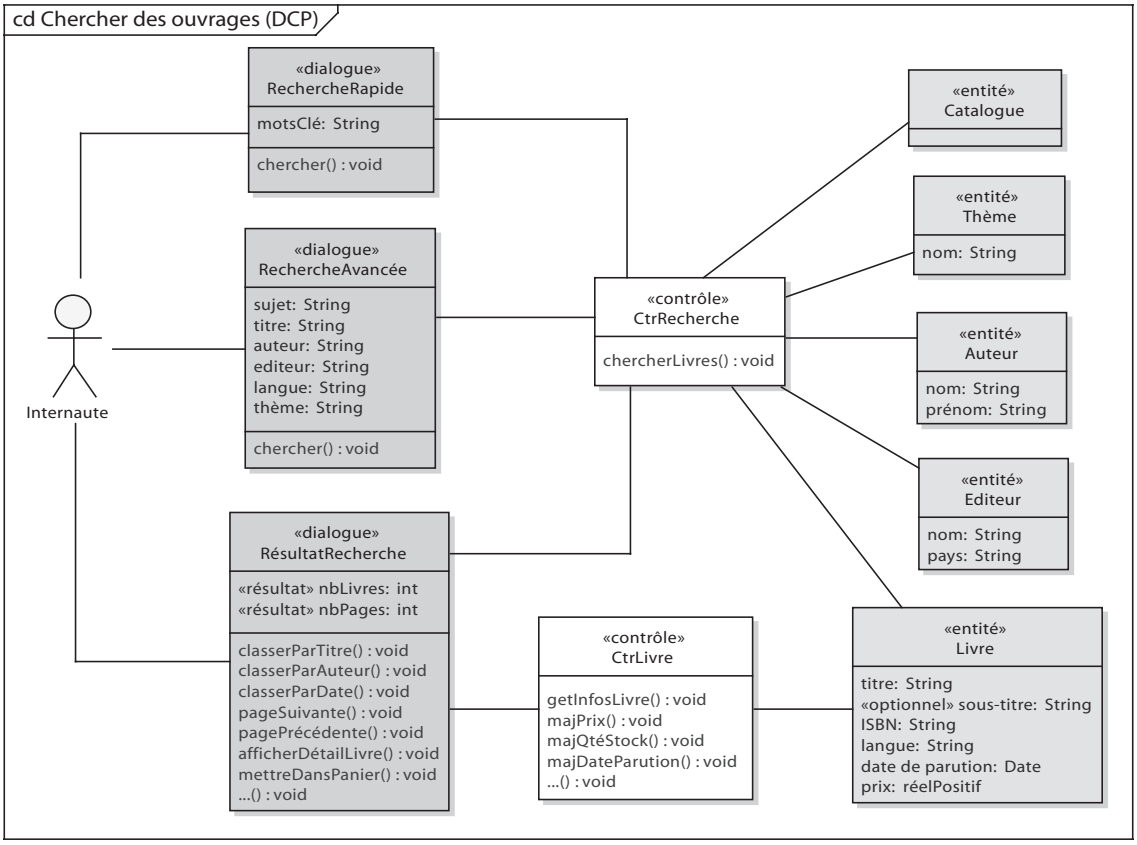


Figure 5–24
DCP de Chercher des ouvrages

Dans ce DCP, les opérations des classes dialogues sont beaucoup plus riches que celles des classes contrôles. On trouve en effet de nombreuses actions purement d'IHM comme le classement du résultat de recherche selon différents critères.

Le dialogue `RésultatRecherche` est relié à la classe contrôle `CtrLivre`, que nous avons identifiée dans le DCP précédent, car l'opération d'IHM `afficherDétailLivre` va invoquer l'opération `getInfosLivre` du contrôle.

Les attributs `nbLivres` et `nbPages` sont stéréotypés «`résultat`» pour indiquer que ces informations sont calculées par le système et non pas fournies par l'internaute.

L'action `mettreDansPanier` de la classe `ResultatRecherche` permet d'accéder au cas d'utilisation `Gérer son panier`, décrit ci-après.

Gérer son panier

Lorsque l'internaute est intéressé par un ouvrage, il a la possibilité de l'enregistrer dans un panier virtuel. Ensuite, il doit pouvoir ajouter d'autres livres, en supprimer ou encore en modifier les quantités avant de passer commande.

Ici, il n'y a qu'un seul écran, avec un seul contrôle, comme indiqué sur la figure 5-25. Toutefois, l'établissement d'un devis affiche un dialogue supplémentaire que l'on ne peut qu'imprimer.

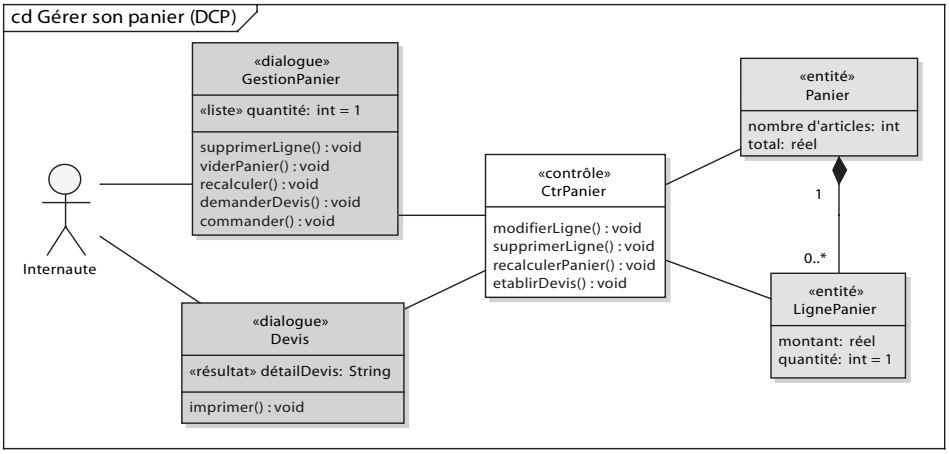


Figure 5–25 DCP de Gérer son panier

Les opérations des dialogues correspondent assez directement aux noms des boutons sur les écrans. Celles des contrôles peuvent être nommées différemment pour une meilleure correspondance avec les entités manipulées. L'attribut `quantité` dans la classe `GestionPanier` est précédé d'un stéréotype «`liste`» pour indiquer que le dialogue permet de gérer une liste de quantités (une par ligne) initialisées à `1`.

Nous avons choisi de montrer dans le diagramme la relation de composition entre `Panier` et `LignePanier`, à cause de son importance vis-à-vis du cas d'utilisation. En effet, l'internaute peut modifier des lignes précises ou le panier dans son ensemble. L'action `commander` de la classe `GestionPanier` donne accès au cas d'utilisation `Effectuer une commande`, décrit ci-après.

Effectuer une commande

À tout moment, le client doit pouvoir accéder au formulaire du bon de commande, dans lequel il peut saisir ses coordonnées et les informations nécessaires au paiement et à la livraison. L'authentification du client et la création d'un compte par un internaute visiteur seront traités dans les cas d'utilisation correspondants (non détaillés dans ce chapitre).

Il y a deux écrans principaux :

- l'écran de saisie des adresses ;
- l'écran de paiement.

Les comportements correspondant à ces fonctionnalités sont gérés par un contrôle unique. Le diagramme ainsi obtenu est montré sur la figure 5-26.

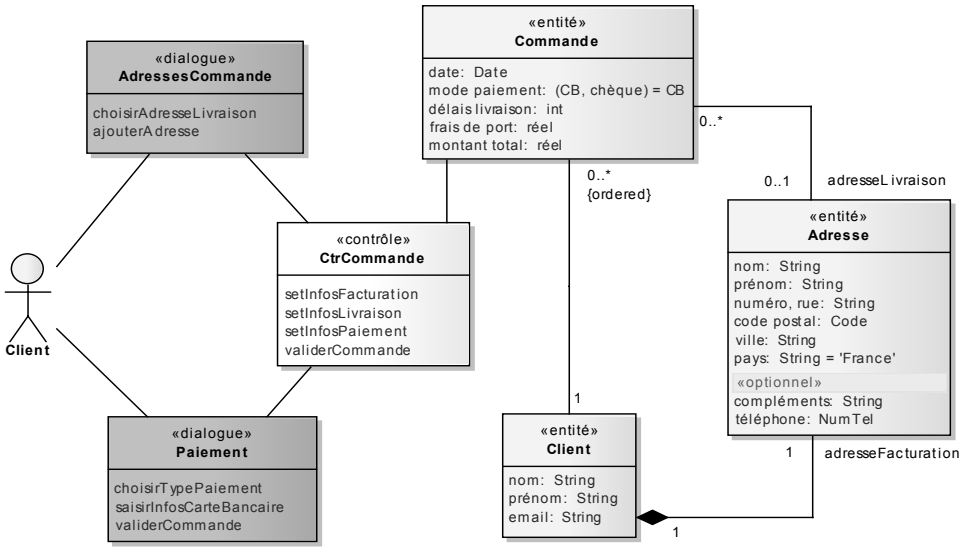


Figure 5–26
DCP de Effectuer une commande

Diagramme d'états

Définitions et notation graphique

UML a repris le concept bien connu de machine à états finis, qui consiste à s'intéresser au cycle de vie d'un objet générique d'une classe particulière au fil de ses interactions, dans tous les cas possibles. Cette vue locale d'un objet, qui décrit comment il réagit à des événements en fonction de son état courant et comment il passe dans un nouvel état, est représentée graphiquement sous la forme d'un diagramme d'états.

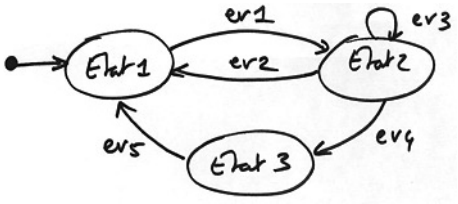


Figure 5–27
Premier exemple simple de diagramme d'états

Heureusement, toutes les classes du modèle ne requièrent pas nécessairement une machine à états. Il s'agit donc de trouver celles qui ont un comportement dynamique complexe nécessitant une description plus poussée. Cela correspond à l'un des deux cas suivants :

- Les objets de la classe réagissent différemment à l'occurrence du même événement : chaque type de réaction caractérise un état particulier.
- La classe doit organiser certaines opérations dans un ordre précis : dans ce cas, des états séquentiels permettent de préciser la chronologie forcée des événements d'activation.

Dans notre modèle, une des seules classes entités répondant à l'un des deux cas précédents (en l'occurrence le deuxième) est la classe `Commande`. En effet, les objets de cette classe sont manipulés au travers de plusieurs cas d'utilisation : `Effectuer une commande`, bien sûr, mais aussi `Consulter ses commandes` (qui permet de les modifier, sous certaines conditions que nous allons préciser).

B.A.-BA État

Un état représente une situation durant la vie d'un objet pendant laquelle :

- il satisfait une certaine condition,
- il exécute une certaine activité,
- ou bien il attend un certain événement.

Un objet passe par une succession d'états durant son existence. Un état a une durée finie, variable selon la vie de l'objet, en particulier en fonction des événements qui lui arrivent.

B.A.-BA Transition

Une transition décrit la réaction d'un objet lorsqu'un événement se produit (généralement l'objet change d'état, mais pas forcément). En règle générale, une transition possède un événement déclencheur, une condition de garde, un effet et un état cible.

B.A.-BA État initial et état final

En plus de la succession d'états « normaux » correspondant au cycle de vie d'un objet, le diagramme d'états comprend également deux pseudo-états :

- L'état initial du diagramme d'états correspond à la création de l'instance.
- L'état final du diagramme d'états correspond à la destruction de l'instance.

B.A.-BA Effet, action, activité

Une transition peut spécifier un comportement optionnel réalisé par l'objet lorsqu'elle est déclenchée. Ce comportement est appelé « effet » en UML 2 : cela peut être une simple action ou une séquence d'actions. Une action peut représenter la mise à jour d'un attribut, un appel d'opération, la création ou la destruction d'un autre objet, ainsi que l'envoi d'un signal à un autre objet. Les activités durables (*do-activity*) ont une certaine durée, sont interruptibles et sont associées aux états.

La notation de base du diagramme d'états est donnée par la figure 5-28.

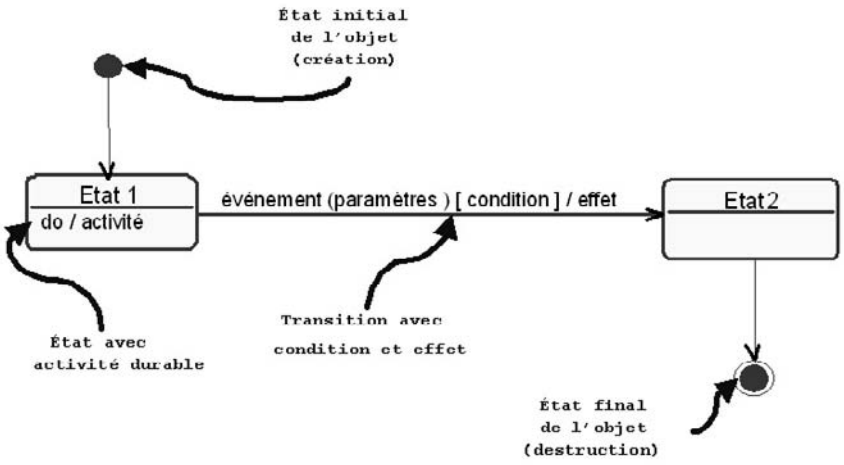


Figure 5–28
Notation de base du diagramme d'états

Diagramme d'états de la classe Commande

La démarche de construction d'un diagramme d'états peut s'énoncer comme suit :

- Représentez tout d'abord la séquence d'états qui décrit le comportement nominal d'un objet, avec les transitions qui lui sont associées.
- Ajoutez progressivement les transitions qui correspondent aux comportements alternatifs ou d'erreur.
- Complétez les effets sur les transitions et les activités dans les états.

Nous allons appliquer cette démarche à la construction du diagramme d'états de la classe `Commande`.

Commençons par représenter les états et transitions de la vie normale d'une commande (figure 5-29).

Ajoutons ensuite les alternatives et erreurs suivantes :

- La validation et le paiement de la commande peuvent se heurter à des problèmes.
- Des incidents peuvent intervenir au cours de la mission de transport ou au moment de la livraison.
- On peut annuler la commande jusqu'à ce qu'elle soit prise en compte par le service clients.

Le diagramme se complexifie comme représenté sur la figure 5-30.

B.A.-BA **Condition**

Une condition (ou condition de garde) est une expression booléenne qui doit être vraie lorsque l'événement arrive pour que la transition soit déclenchée. Elle se note entre crochets.
Elle peut concerner les attributs de l'objet concerné ainsi que les paramètres de l'événement déclencheur. Plusieurs transitions avec le même événement doivent avoir des conditions de garde différentes.

Concept avancé **Événement temporel**

Le passage du temps (*time event*) se modélise en utilisant le mot-clé `after` suivi d'une expression représentant une durée, décomptée à partir de l'entrée dans l'état courant.

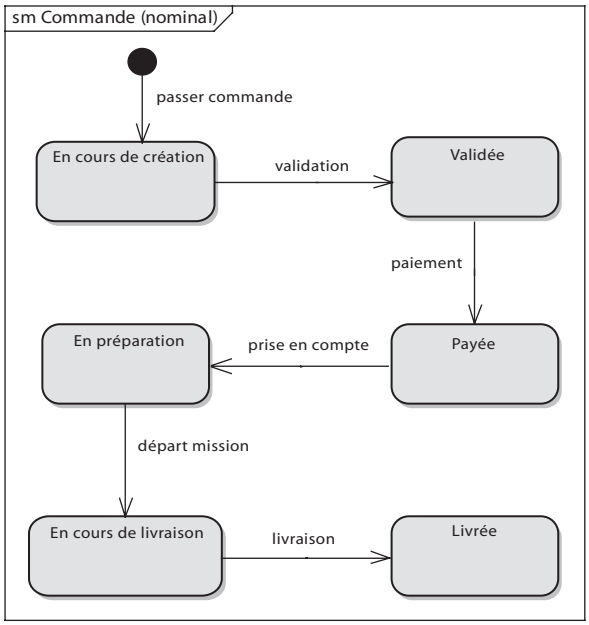


Figure 5–29
Première version du diagramme d'états de la commande

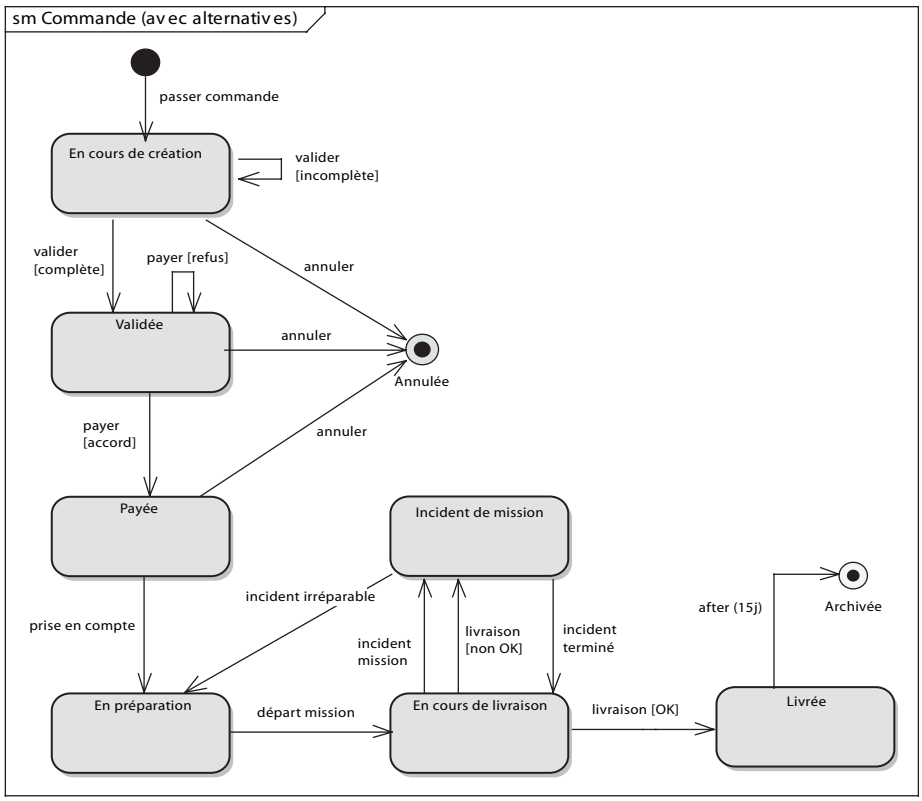


Figure 5–30
Deuxième version du diagramme d'états de la commande

Complétons maintenant avec les effets sur les transitions et les activités dans les états.

Il faut en effet avertir le client de l'état d'avancement de sa commande, des éventuels problèmes, etc.

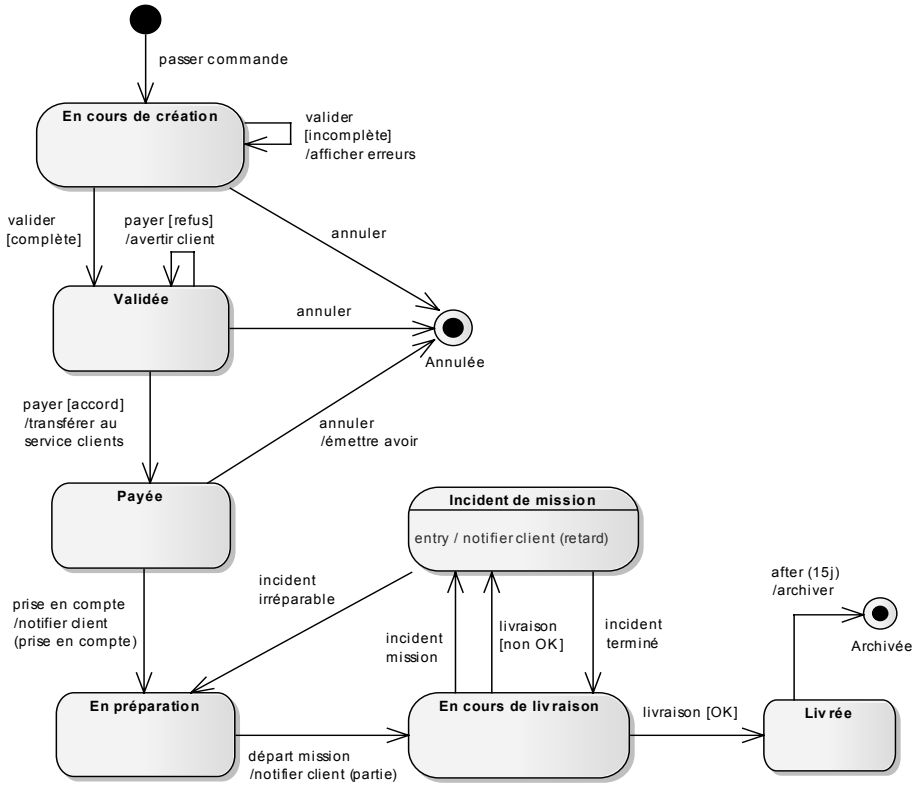
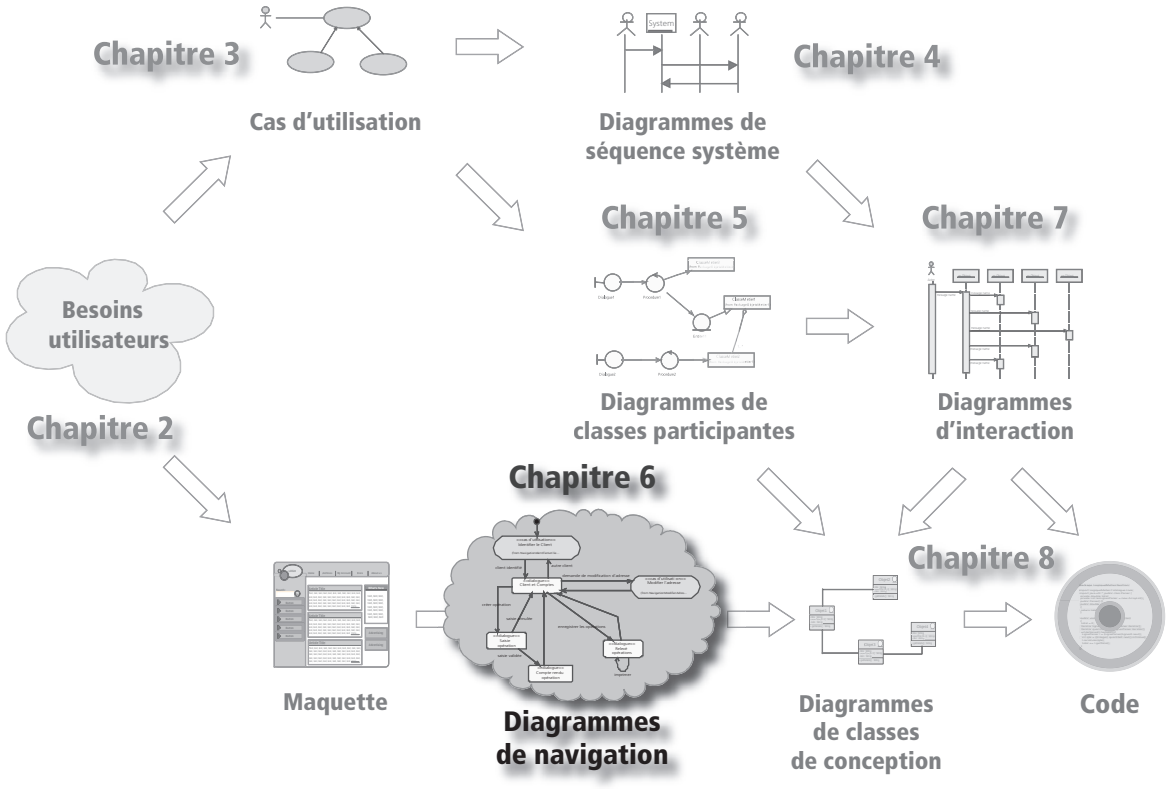


Figure 5–31
Version complétée du diagramme d'états de la commande

CONCEPT AVANCÉ **Effet d'entrée (ou de sortie) – entry (exit)**

Un effet d'entrée (introduit par le mot-clé «`entry`» à l'intérieur du symbole d'un état) représente une action ou une activité qui est exécutée chaque fois que l'on entre dans cet état. Ainsi, on factorise un même effet qui sera déclenché par toutes les transitions entrant dans l'état.
L'effet de sortie (introduit par le mot-clé «`exit`») est l'effet symétrique en sortie de l'état.

chapitre 6


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Chapitre 7
Besoins utilisateurs
Chapitre 2
Diagrammes de classes participantes
Diagrammes d'interaction
Chapitre 6
Chapitre 8
Maquette
Diagrammes de navigation
Diagrammes de classes de conception
Code

Modélisation de la navigation

Apprenons maintenant à utiliser le diagramme d'états UML pour modéliser précisément la navigation dans le site web. Nous verrons également une alternative intéressante d'utilisation du diagramme d'activité proposée par la méthode MACAO.

SOMMAIRE

MOTS-CLÉS

- Navigation
- Écran
- IHM
- Diagramme d'états
- Diagramme d'activité
- Dialogue

Démarche

Rappelons le positionnement de cette activité d'analyse par rapport à l'ensemble du processus décrit au chapitre 1. Nous avons identifié les cas d'utilisation au chapitre 3 et poursuivi leur description détaillée au chapitre 4. Pour passer en douceur à la conception, nous avons identifié au chapitre 5 les principales classes d'IHM (dialogues) ainsi que celles qui décrivent la cinématique de l'application (contrôles). Pour que le tableau soit complet, il nous reste à détailler une exploitation supplémentaire de la maquette ou une extension éventuelle de celle-ci. Il s'agit de réaliser des diagrammes dynamiques représentant de manière formelle l'ensemble des chemins possibles entre les principaux écrans proposés à l'utilisateur. Ces diagrammes s'appellent des diagrammes de navigation.

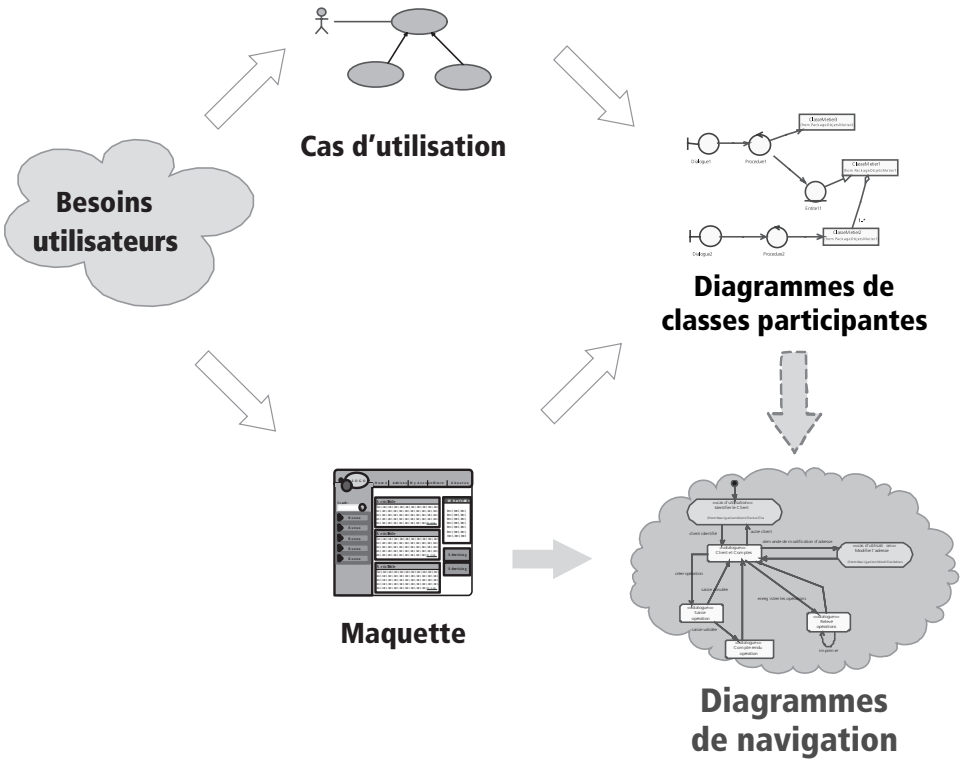


Figure 6–1
Les diagrammes de navigation dans la démarche

Les IHM modernes, en particulier celles destinées aux internautes via les sites web marchands, cherchent à faciliter la communication avec l'utilisateur. Elles sont en général basées sur des combinaisons de fenêtres, frames, boîtes de dialogue et autres boutons qui permettent :

- de présenter des éléments graphiques ou textuels reflétant l'état courant du système ;

- de modifier cet état courant en envoyant des messages au système,
- de naviguer à l'intérieur du site web afin d'en explorer toutes les possibilités en respectant certaines séquences logiques.

Cette combinaison possible d'options d'affichage, d'interaction et de navigation ouvre la voie à des interfaces de plus en plus riches et puissantes. Toutefois, cela oblige également le concepteur du site à réfléchir de façon très précise au comportement attendu de tous ses éléments graphiques et pas seulement d'un point de vue visuel.

Actuellement, les concepteurs utilisent pour la plupart des outils de modélisation limités, consistant principalement en des sortes d'organigrammes accompagnés de captures d'écrans. Néanmoins, ces techniques sont généralement faibles pour illustrer précisément la dynamique d'enchaînement des écrans, ainsi que l'effet de cette navigation dans le système, ou la prise en compte de règles métier dans le séquencement possible.

UML nous offre la possibilité de représenter formellement cette navigation, au moyen d'un diagramme d'états (ou éventuellement d'un diagramme d'activité). L'utilisation de la modélisation UML pour la navigation nous permet en outre de la relier efficacement aux classes dialogues que nous avons identifiées au chapitre 5.

Le diagramme de navigation représente ainsi un ajout important dans l'arsenal des outils de modélisation du concepteur de site web. Comme nous allons le montrer dans la suite de ce chapitre, il fournit la possibilité de décrire précisément et exhaustivement les aspects dynamiques de l'interface utilisateur.

Méthode Diagramme d'états ou diagramme d'activité ?

La discussion sur les mérites comparés du diagramme d'états et du diagramme d'activité, tous deux proposés par UML, est en dehors de la portée de ce livre. Le choix entre les deux est quasiment une question de goût ainsi que des possibilités comparées de votre outil de modélisation préféré.

En toute rigueur, il faudrait préférer le diagramme d'états, puisque nous allons modéliser un comportement événementiel. Cependant, en UML 1.x, le diagramme d'activité était une spécialisation du diagramme d'états, ce qui a conduit certains auteurs à le préférer pour des raisons pratiques. Citons en particulier la méthode MACAO, élaborée par J.-B. Crampes à l'IUT de Blagnac, dont nous allons expliquer les grands principes plus avant dans ce chapitre.

Conseil Soignez la navigation

La navigation est particulièrement importante pour les sites web. Elle peut prendre la forme de menus, liens, hyperliens, boutons directionnels, etc. Une navigation fastidieuse n'incitera pas l'internaute à revenir sur le site web incriminé… Pour plus de détails, le lecteur se référera avec profit au Mémento paru sur le sujet :

📖 *Sites web - Les bonnes pratiques*, E. Sloïm, Eyrolles, 2007

Méthode Quelques questions à se poser

Comment, par exemple, décrire formellement les quelques règles suivantes :

- Pour commander, il faut que mon panier ne soit pas vide.
- Si je sors de l'écran de création de commande pour continuer ma recherche, j'ai perdu ma commande non confirmée, mais pas mon panier courant.
- Je ne peux pas modifier ma fiche client si je ne suis pas déjà enregistré comme tel.

Diagramme d'états de navigation

Notations de base

Nous avons vu la notation de base du diagramme d'états dans le chapitre 5.

Pour modéliser la navigation dans un site web, nous allons utiliser un nombre restreint d'éléments standards, à savoir :

- des états pour représenter les classes dialogues,
- des transitions entre états déclenchées par des événements et pouvant porter des conditions, pour représenter les actions IHM.

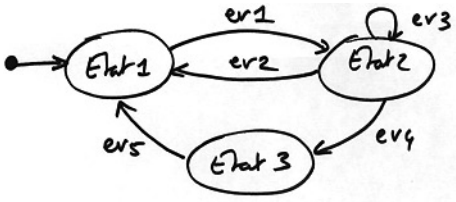


Figure 6–2
Notation graphique de base du diagramme d'états

Conventions spécifiques

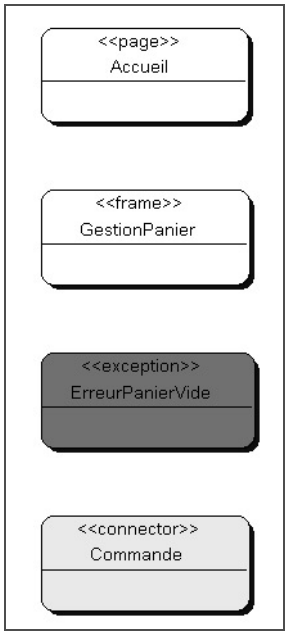


Figure 6–3
Conventions graphiques spécifiques

Nous allons pousser un peu plus loin et nous servir du concept d'état pour modéliser plusieurs concepts différents, grâce aux conventions graphiques suivantes :

- une page complète du site («`page`»),
- un frame particulier à l'intérieur d'une page («`frame`»),
- une erreur ou un comportement inattendu du système («`exception`» avec un niveau de gris intermédiaire),
- une liaison vers un autre diagramme d'activité, pour des raisons de structuration et de lisibilité («`connector`» avec un niveau de gris soutenu).

On pourrait facilement multiplier les concepts et les conventions correspondantes, mais nous pensons que ces quatre types d'états constituent un ensemble simple quoique suffisamment complet pour notre utilisation courante (voir figure 6-3).

Les concepts de page et de frame sont directement en rapport avec les classes dialogues du chapitre 5.

Structuration de la navigation

Pour commencer le travail de modélisation, nous devons d'abord le séparer en ensembles maîtrisables et les plus indépendants possibles. Il

est clair par exemple que la navigation du site par l'internaute sera complètement différente de celle du libraire qui n'accède pas du tout aux mêmes fonctionnalités, comme nous l'a confirmé l'analyse des acteurs et des cas d'utilisation effectuée aux chapitres 3 et 4.

La modélisation de la navigation peut donc se structurer tout d'abord par acteur. Le début du diagramme de navigation de chaque acteur est ainsi représenté sur la figure 6-4.

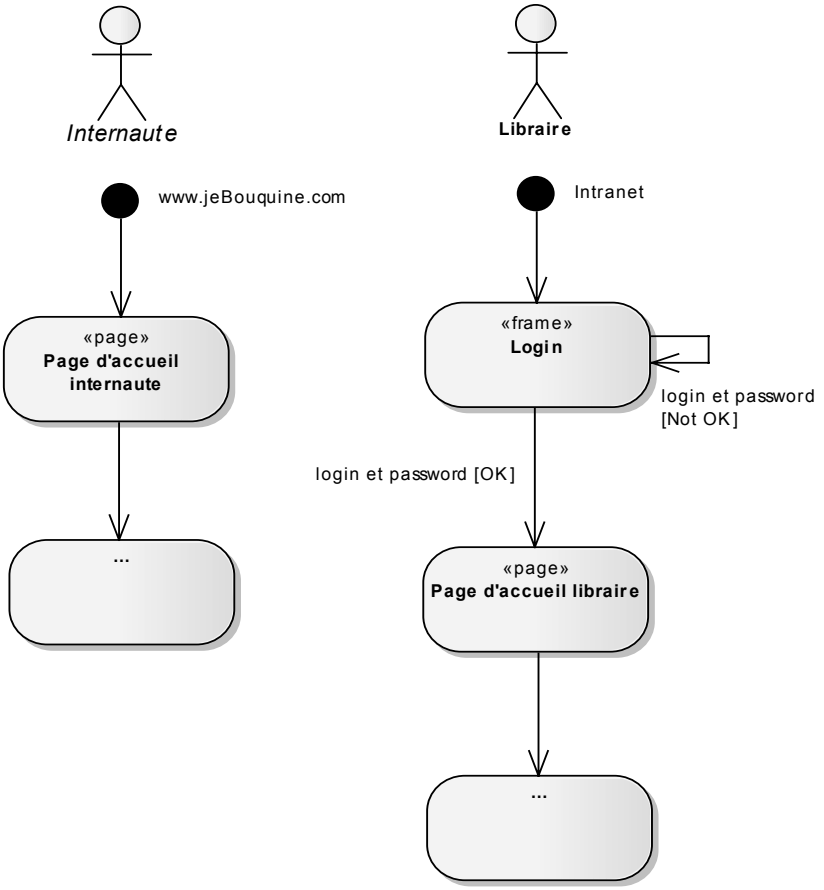


Figure 6–4
Début des diagrammes de navigation des acteurs Internaute et Libraire

L'internaute démarre par la page d'accueil du site web www.jeBouquine.com. Le libraire, lui, se connecte sur l'intranet de la société. Il doit saisir son identifiant et son mot de passe dans un frame particulier. Suivant le résultat du contrôle effectué par le système, il se retrouve soit sur la page d'accueil propre à son profil, soit de nouveau sur le frame d'identification avec un message d'erreur. Nous n'avons pas modélisé la limite de trois erreurs consécutives pour ne pas compliquer le diagramme, mais il faudrait bien sûr le faire dans la réalité.

Demandons-nous ensuite si la navigation se structure également par cas d'utilisation. Si les cas d'utilisation sont complètement indépendants, il n'y a aucune raison de ne pas réaliser un diagramme de navigation par cas, ce qui donne des diagrammes simples et lisibles. Toutefois, nous avons identifié aux chapitres 3 et 4 que les cas d'utilisation de l'internaute, en particulier, ont des relations les uns avec les autres : impossible de commander si le panier est vide, etc. Ces dépendances importantes vont induire très directement des contraintes de navigation entre les pages correspondantes. Le diagramme de navigation va nous permettre de les représenter de façon formelle.

Navigation de l'internaute

Nous nous intéressons dans un premier temps aux trois cas d'utilisation majeurs :

- `Chercher des ouvrages` ;
- `Gérer son panier` ;
- `Effectuer une commande`.

Chercher des ouvrages

L'internaute commence en général par sélectionner des ouvrages pour pouvoir ensuite les commander, après avoir éventuellement géré son panier. Partons donc de la description détaillée du cas d'utilisation `Chercher des ouvrages` (voir chapitre 5).

À partir de la page d'accueil, l'internaute doit ainsi avoir la possibilité de lancer une recherche rapide, une recherche avancée, ou de flâner dans un ensemble de pages telles que : *nouveautés*, *meilleures ventes*, etc.

MÉTHODE Événement curseur

Nous avons pris la convention d'utiliser un événement appelé `curseur` pour indiquer que l'internaute va saisir des informations dans un frame à l'intérieur d'une page, en déplaçant son curseur textuel dans la zone concernée, comme c'est le cas pour la recherche rapide.

La recherche rapide est un frame toujours présent dans les pages de recherche (voir figures 4-6 et 4-7), contrairement à la recherche avancée qui est une page à part entière.

La recherche elle-même est une action dont le résultat aboutit à un frame contenant la liste des livres ou bien à un frame d'erreur indiquant qu'aucun ouvrage n'a été trouvé. Nous avons utilisé un état composite (nommé `Recherche`) afin de factoriser les transitions activées par l'événement `recherche`, ainsi qu'un pseudo-état historique pour le retour après le message d'erreur.

CONCEPT AVANCÉ Pseudo-état Historique (H)

Un état historique permet à un état composite de se souvenir de son dernier sous-état actif avant sa dernière sortie. On le représente par un petit cercle contenant la lettre H.

Sur le résultat de recherche, l'internaute pourra soit sélectionner un ouvrage pour obtenir sa fiche détaillée, soit le mettre directement de côté dans son panier, soit reclasser les résultats. Si le nombre d'ouvrages trouvés est grand, l'internaute pourra naviguer entre les différentes pages de résultats.

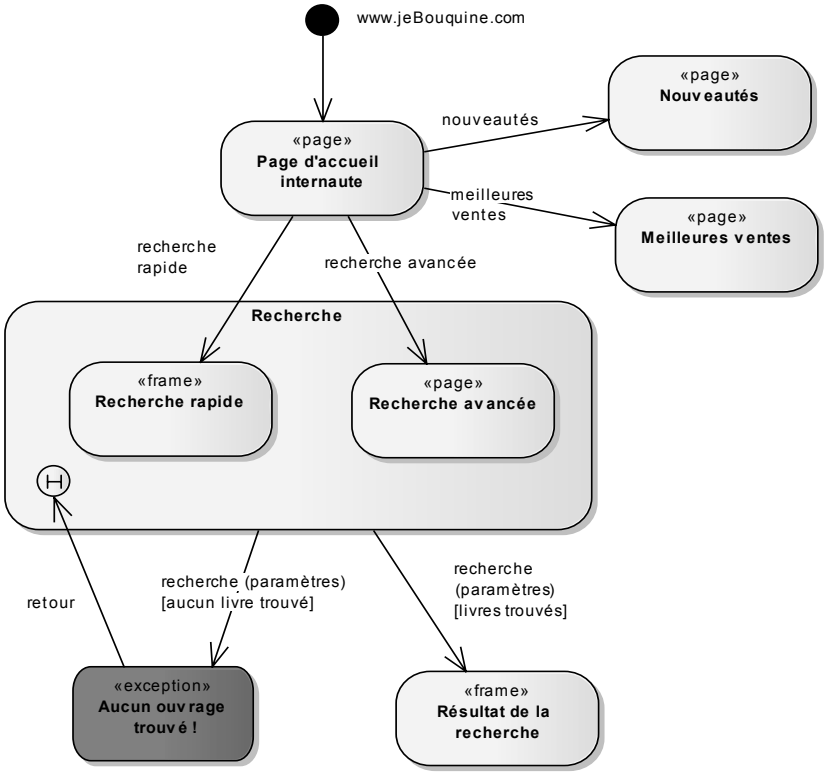


Figure 6–5
Navigation de la recherche (début)

La fiche d'ouvrage permet à son tour de mettre le livre de côté dans le panier, ou d'accéder à la fiche de l'auteur indiquant une biographie et l'ensemble des ouvrages qu'il a écrits. Elle donne également l'accès à des informations complémentaires telles que : image agrandissable, commentaires éventuels, table des matières détaillée, extraits de chapitre, etc. Le diagramme de navigation de la recherche devient ainsi comme indiqué sur la figure 6-6. Nous avons choisi de détailler la gestion du panier (elle-même complexe) dans un autre diagramme, en utilisant le concept de «connector».

La recherche d'ouvrage a pour objectif de mettre les livres sélectionnés dans le panier virtuel pour finalement passer une commande. C'est en tout cas le but avoué du site marchand ! Nous allons donc étudier maintenant la navigation concernant la gestion du panier.

MÉTHODE Transitions triviales

Il est à noter que nous n'indiquons pas les retours systématiques à la page précédente qui ajouteraient de nombreuses transitions triviales alourdissant notablement le diagramme. De même, l'internaute a accès en permanence à l'aide… Nous considérerons donc toutes ces transitions comme implicites, pour nous concentrer sur celles qui ont une véritable signification « métier ».

Gérer son panier

Dans la figure 6-6, on voit des actions `mettre dans le panier` et un «connector» `Panier`. C'est donc ce «connector» que nous allons détailler ici. Néanmoins, nous allons également envisager le cas du panier vide, ce qui nous oblige à repartir de la page d'accueil.

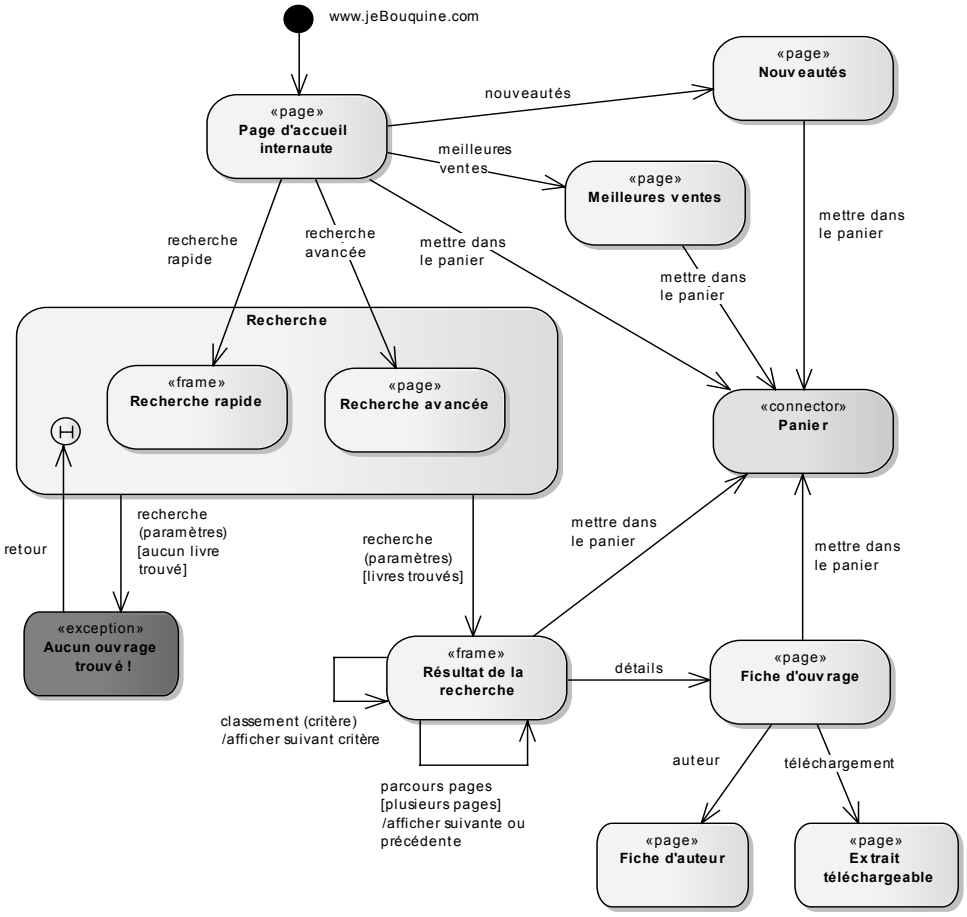


Figure 6–6
Navigation de la recherche (deuxième jet)

En effet, l'accès au panier est disponible de partout, y compris de la page d'accueil. Il faut donc traiter une «exception» `Panier vide`, qui empêche de passer à la page de commande. Le «connector» `Panier` de la figure 6-6 correspond donc cette fois-ci à la «page» `Panier` de la figure 6-7 et pas au point d'entrée du diagramme comme précédemment.

Nous avons représenté la règle métier importante qui consiste à interdire l'accès à la commande dans le cas où le panier est vide, puisque l'action `commander` n'est possible que dans l'état `Panier`. Or, on aboutit dans cet état uniquement en mettant un livre dans le panier, et on le quitte forcément dès que le panier est vide.

MÉTHODE Annuler la commande

Nous avons représenté la règle simple suivante : « je ne peux plus annuler ma commande une fois qu'elle a été validée. En fait, je pourrai le faire ultérieurement en passant par le cas d'utilisation Consulter ses commandes, mais c'est une autre histoire de navigation… »

Effectuer une commande

Le `connector Commande` de la figure 6-7 sert de point d'entrée au diagramme 6-8 qui décrit la navigation pour effectuer une commande.

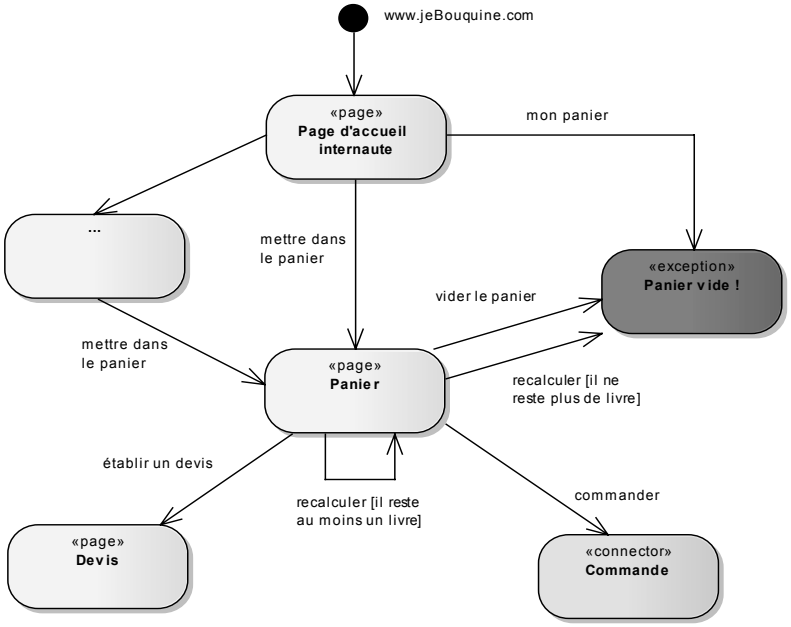


Figure 6–7
Diagramme de navigation de la gestion du panier

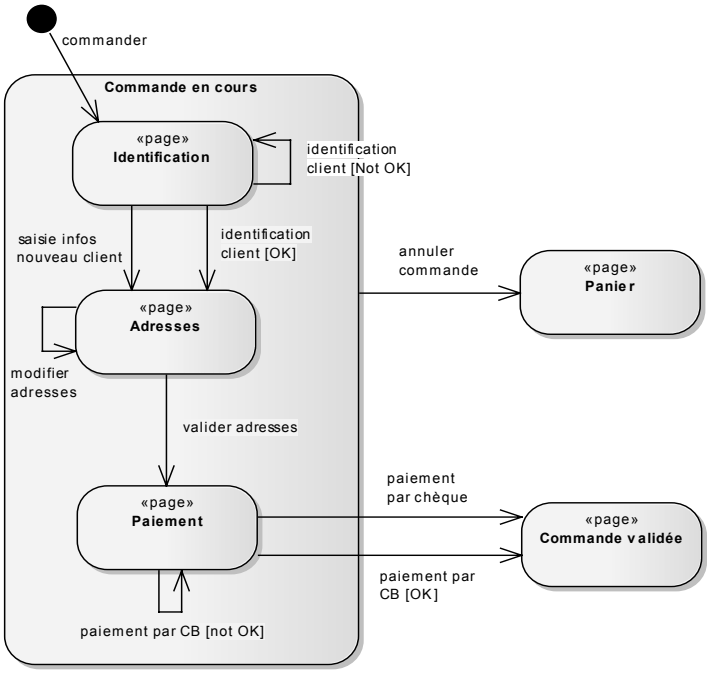


Figure 6–8
Diagramme de navigation pour effectuer une commande

Résumé de la navigation de l'internaute

Nous pouvons maintenant réaliser un diagramme global de navigation de l'internaute. Il va comprendre l'ensemble des pages, frames et actions principales des trois cas d'utilisation de l'internaute. Pour simplifier, nous n'avons pas repris les exceptions.

Le schéma ainsi obtenu (figure 6-9) est très intéressant, car il fournit sur une seule page les règles fondamentales de déplacement dans le site web.

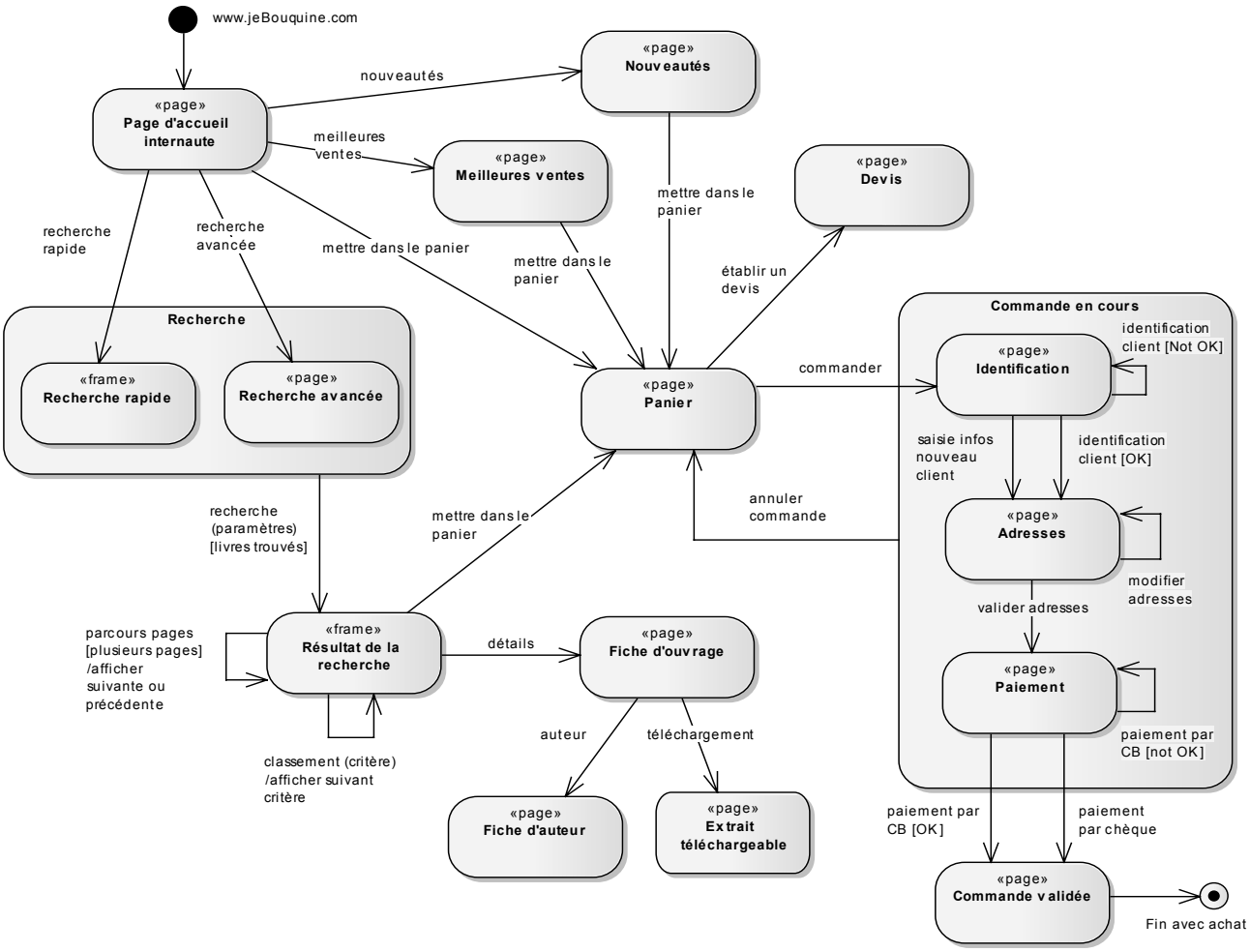


Figure 6–9 Diagramme global simplifié de la navigation de l'internaute

MÉTHODE **Ergonomie du site**

Il reste encore à l'ergonome la charge de rendre la navigation simple et pratique. Une navigation efficace comprend non seulement des liens vers d'autres pages du site web, mais aussi des éléments permettant au visiteur de se situer. Il doit pouvoir répondre aux questions suivantes :

- Où suis-je ?
- Où puis-je aller ?
- Comment puis-je y aller ?
- Comment puis-je revenir à mon point de départ ?

Quelques conseils complémentaires :

- Limitez la surcharge d'information (plutôt des pages courtes avec des liens internes que des pages sans fin…).
- Utilisez une navigation textuelle plutôt que purement graphique.
- Facilitez le retour vers vos options de navigation.

Alternative : diagramme d'activité de navigation

Notations de base

Le diagramme d'activité est l'un des diagrammes dynamiques d'UML. Il ressemble fondamentalement à un ordinogramme, montrant le flot de contrôle d'action en action. Les éléments de base du diagramme d'activité sont les suivants :

- des actions,
- des flots de contrôle entre actions,
- des décisions (aussi appelés branchements conditionnels),
- un début et une ou plusieurs terminaison(s) possible(s).

La notation graphique de base est présentée sur la figure 6-10.

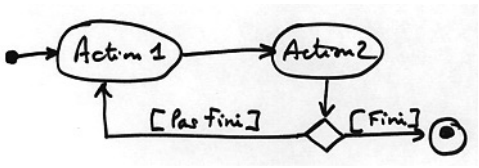


Figure 6–10
Notation de base du diagramme d'activité

B.A.-BA **Action**

L'action est l'unité fondamentale de spécification comportementale en UML 2. Elle représente un traitement ou une transformation. Les actions sont contenues dans les activités, qui fournissent leur contexte.

B.A.-BA **Flot**

Un flot est un contrôle de séquençage pendant l'exécution de nœuds d'activité. Les flots de contrôle sont de simples flèches reliant deux nœuds (actions, décisions, etc.). Le diagramme d'activité permet également d'utiliser des flots d'objets (reliant une action et un objet consommé ou produit).

B.A.-BA **Décision-Fusion**

Une décision est un nœud de contrôle structuré représentant un choix dynamique entre plusieurs conditions. Il est représenté par un losange qui possède un arc entrant et plusieurs arcs sortants. À l'inverse, une fusion représente un emplacement où plusieurs chemins de contrôle alternatifs s'assemblent.

Conventions spécifiques (méthode MACAO)

Pour modéliser la navigation dans un site web, certains auteurs préconisent d'utiliser les éléments standard mais en ajoutant des conventions particulières.

Voir par exemple l'article de Ben LieberMan dans le numéro d'octobre 2001 du magazine en ligne www.therationaledge.com intitulé :

📖 *UML activity diagrams : detailing user interface navigation*

Nous allons présenter dans cette section les conventions et ajouts proposés par la méthode MACAO. Cette démarche propose plusieurs modèles spécifiques pour gérer la conception des IHM, et cela à deux niveaux.

Le lecteur curieux d'en savoir plus se reportera à l'ouvrage de J.-B. Crampes, paru aux éditions Ellipses en 2003, intitulé :

📖 *Méthode orientée objet intégrale MACAO*

Au niveau conceptuel, le SNI (Schéma navigationnel d'IHM) représente l'enchaînement fonctionnel du point de vue de l'utilisateur. Il est indépendant de toute plate-forme d'implémentation. Au niveau logique, qui est une représentation plus détaillée, on ajoute des informations proches d'un type d'IHM particulier. À ce jour, MACAO gère deux types de modèles logiques : le schéma d'enchaînement des fenêtres (SEF) pour les applications de type fenêtré sur client lourd, et le schéma d'enchaînement des pages (SEP) pour les applications de type web sur client léger.

Nous allons surtout détailler le SNI, qui se dérive du diagramme d'activité UML par stéréotypage. Pour cela, MACAO identifie un certain nombre d'actions élémentaires typiques des IHM. Elle les généralise en introduisant la notion d'Unité de dialogue élémentaire (UDE), avec les cinq types suivants :

- T1 : Saisie de données ;
- T2 : Affichage d'un objet ;
- T3 : Affichage d'une liste d'objets ;
- T4 : Affichage d'un message ;
- T5 : Décision à choix multiple (menu).

MACAO introduit des symboles spécifiques pour ces différentes UDE, comme illustré sur la figure 6-11.

On peut ensuite combiner ces UDE par juxtaposition afin de représenter des UDC (Unités de dialogue composées). Quelques exemples en sont donnés sur la figure 6-12.

Le SNI est un modèle conceptuel qui fait volontairement abstraction du type d'IHM utilisé (texte, GUI, Web, Palm-OS, etc.). MACAO propose également de modéliser les IHM orientées fenêtre (GUI) et celles orientées page (PUI). Nous allons utiliser le deuxième type d'IHM (PUI) pour notre étude cas. Le schéma d'enchaînement des pages (SEP) permet de représenter la dynamique d'ouverture des diverses pages d'une application intranet ou Internet.

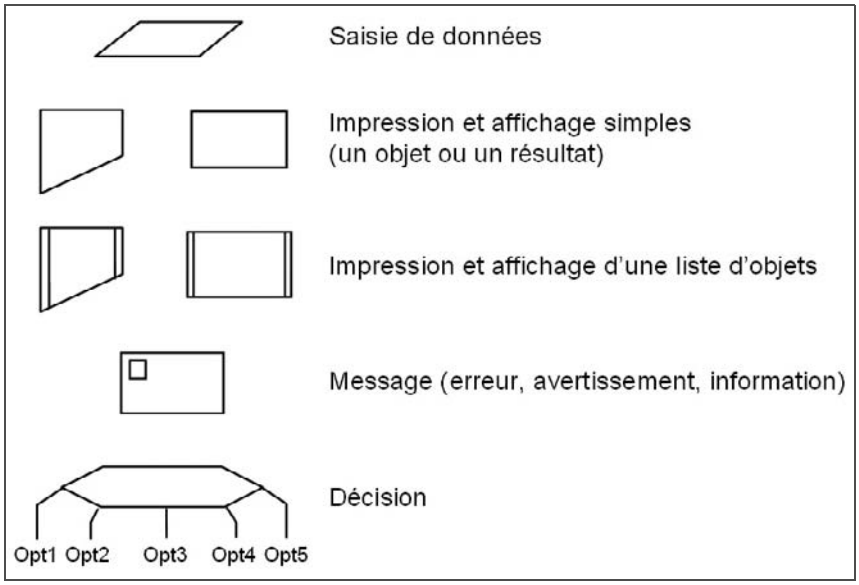


Figure 6–11
Symboles graphiques
des UDE du SNI MACAO

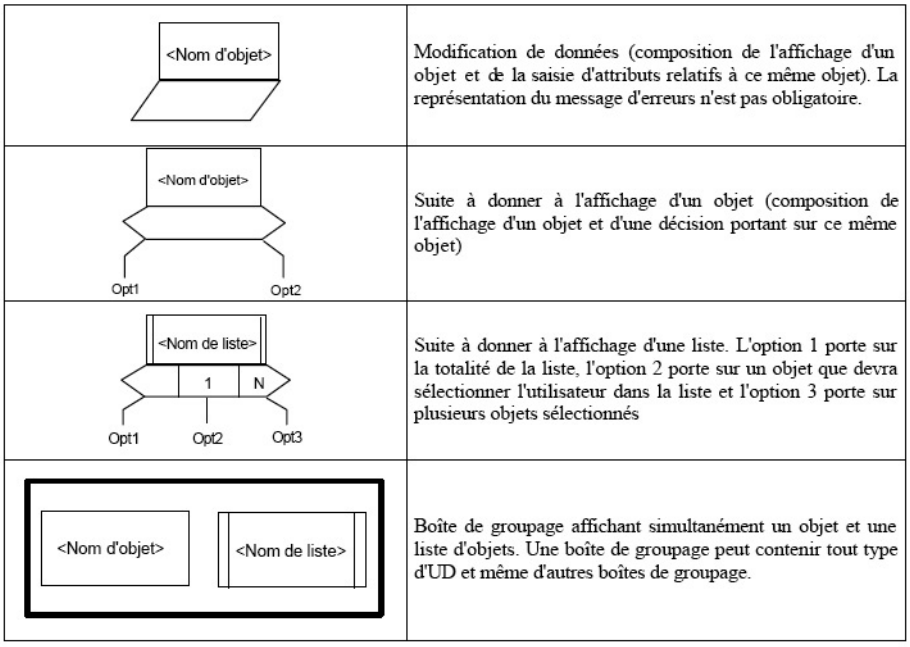


Figure 6–12
Exemples de représentations
d'UDC du SNI MACAO

Le SEP n'utilise qu'un seul type de symbole pour représenter les pages : un rectangle divisé en deux verticalement. La partie gauche contient les caractéristiques générales de la page. La partie droite contient une liste d'objets visuels initiateurs d'une nouvelle page.

Application à l'étude de cas

J.-B. Crampes nous a fait l'amitié de réaliser le SNI du site www.jeBouquine.com, que nous présentons sur les diagrammes qui suivent.

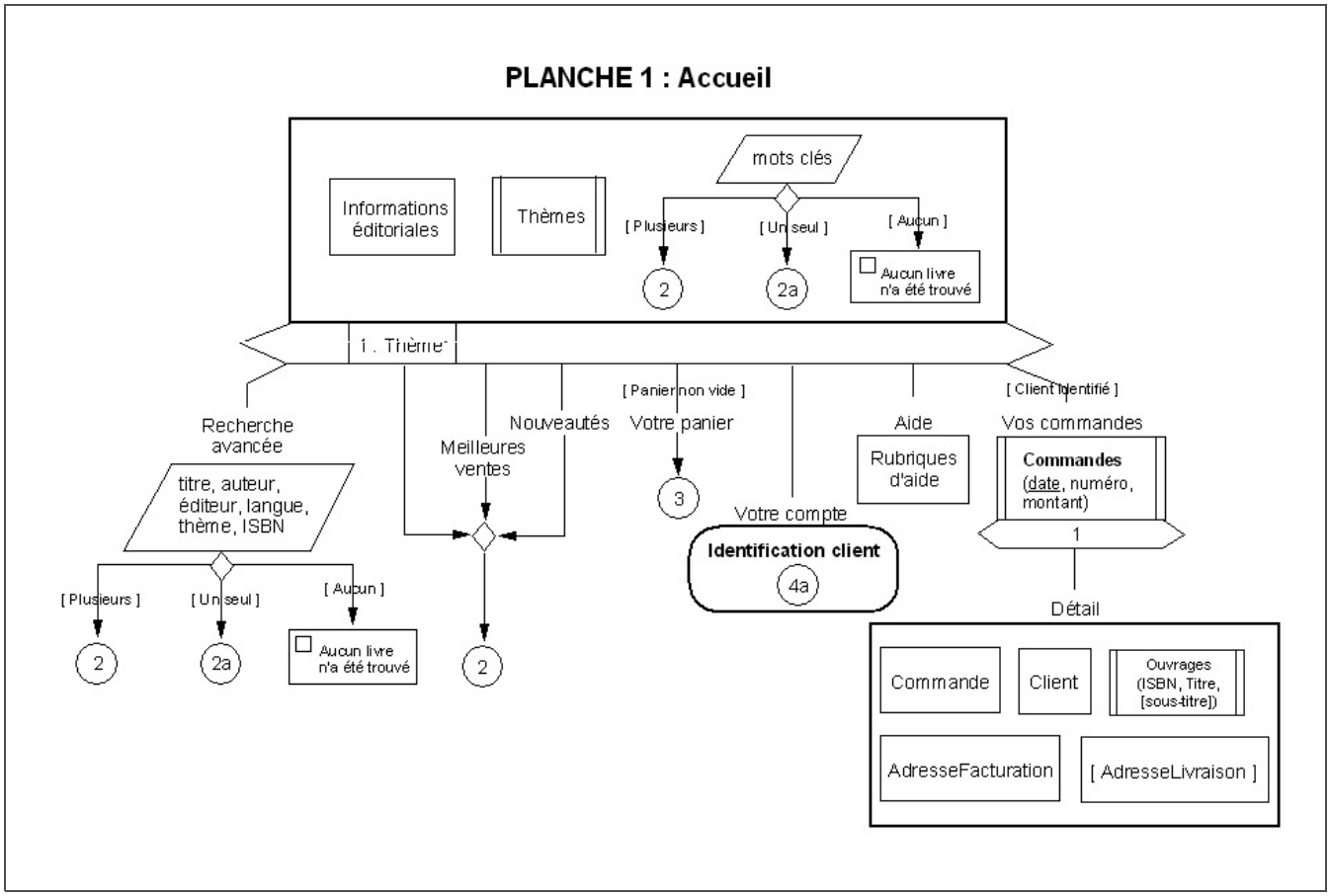


Figure 6–13 Début du SNI MACAO du site jeBouquine

Les différents renvois sont explicités par les planches 6-14 à 6-17.

Le début du SEP est également fourni à titre d'exemple de la précision apportée par la démarche MACAO.

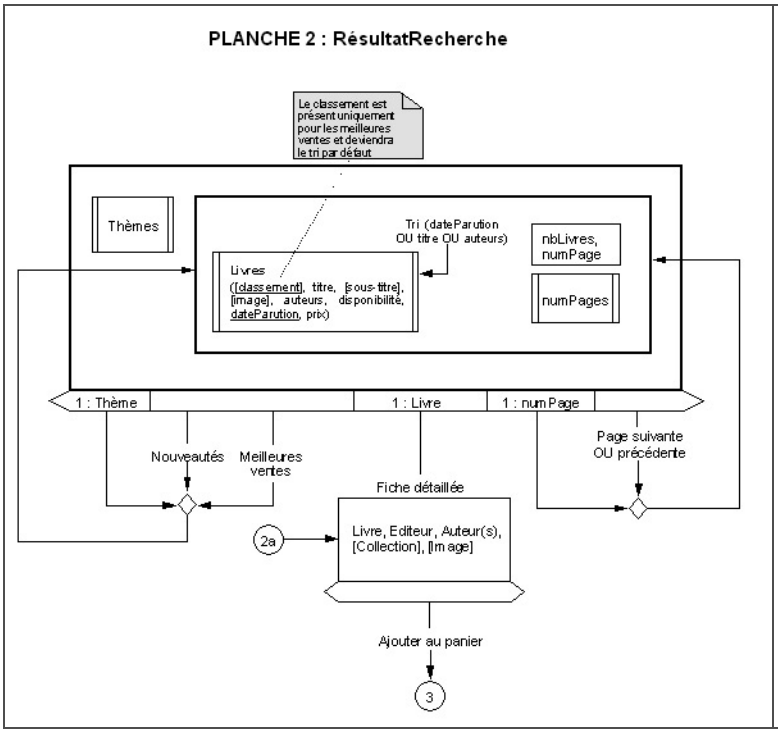


Figure 6–14
Suite du SNI MACAO : RésultatRecherche

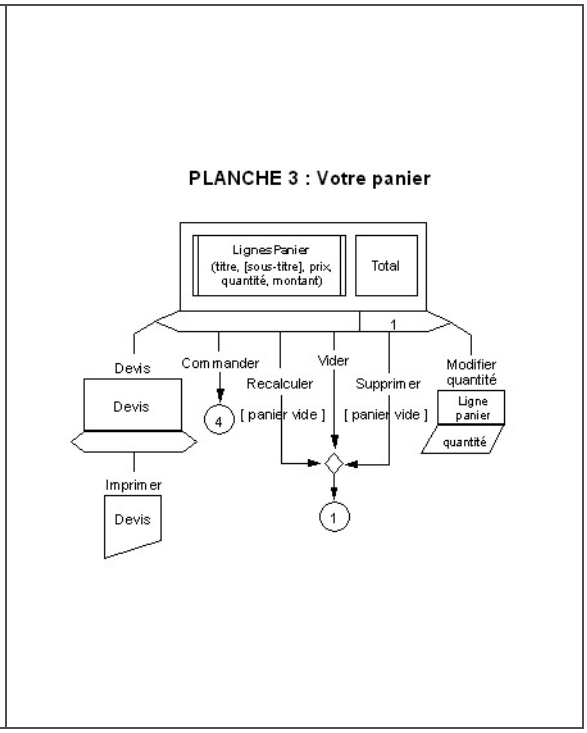


Figure 6–15
Suite du SNI MACAO : GestionPanier

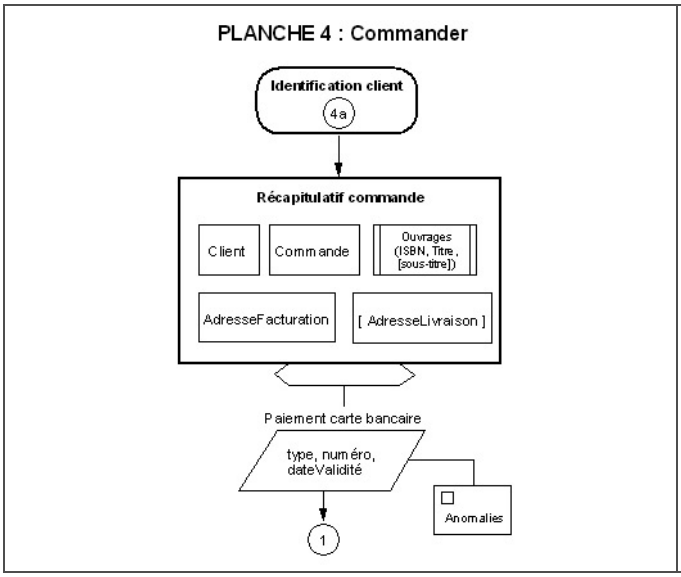


Figure 6–16
Suite du SNI MACAO : Commander

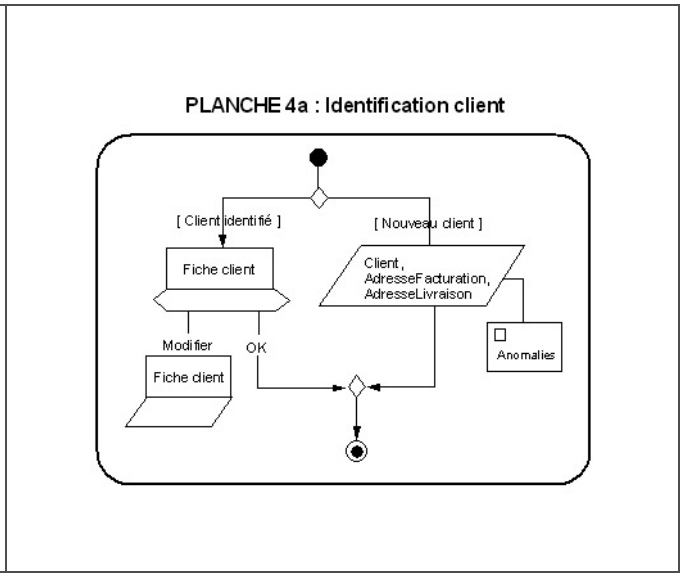


Figure 6–17
Suite du SNI MACAO : Identification client

Le SEP utilise un certain nombre de préfixes pour codifier les principaux types d'objets visuels des PUI. Les principaux utilisés dans les figures 6-18 à 6-20 sont les suivants :

- BP : bouton poussoir ;
- CA : cadre ;
- ES : entrée simple ;
- HTC : lien hypertexte constant ;
- HTV : lien hypertexte variable ;
- PCA : page de cadres ;
- PFO : page de formulaire ;
- TAB : tableau ;
- V : variables.

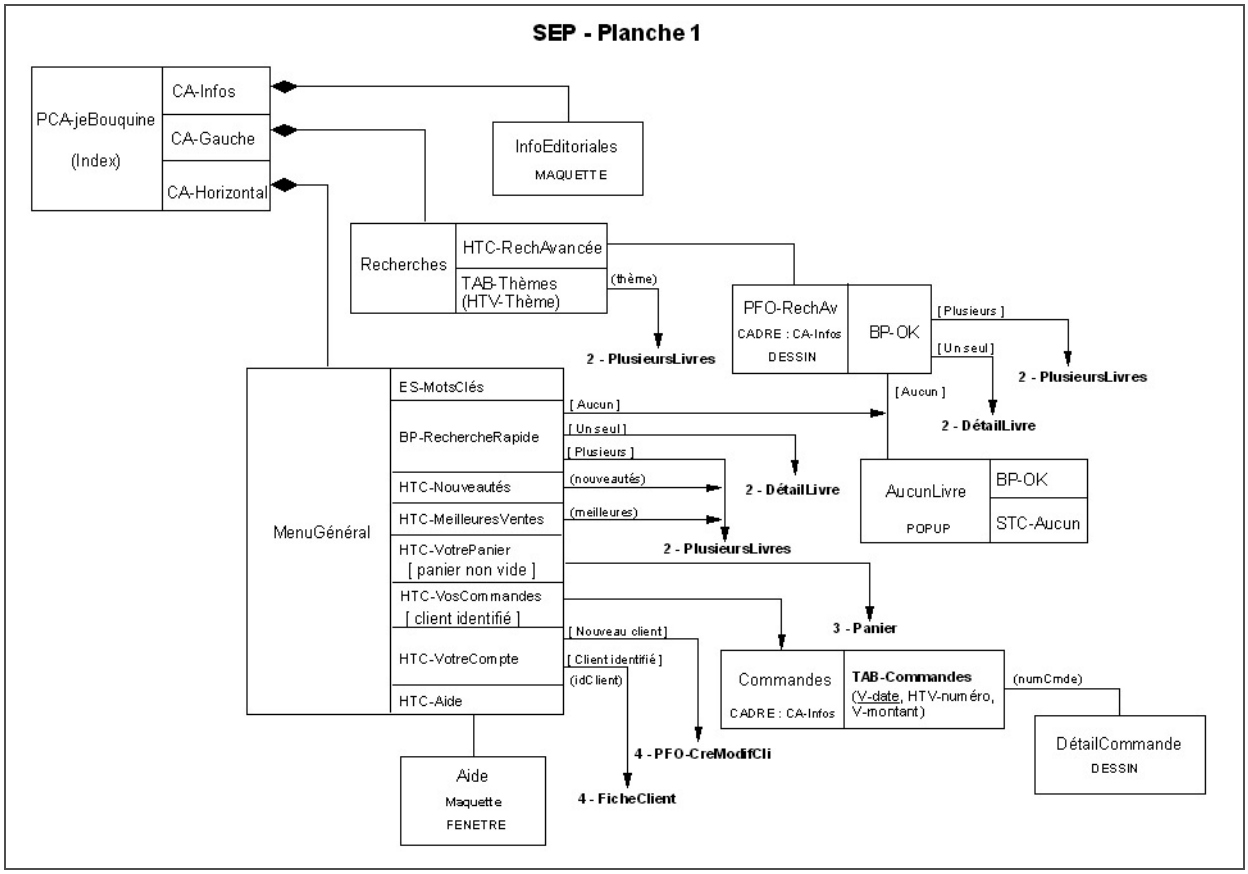


Figure 6–18 Début du SEP MACAO du site jeBouquine

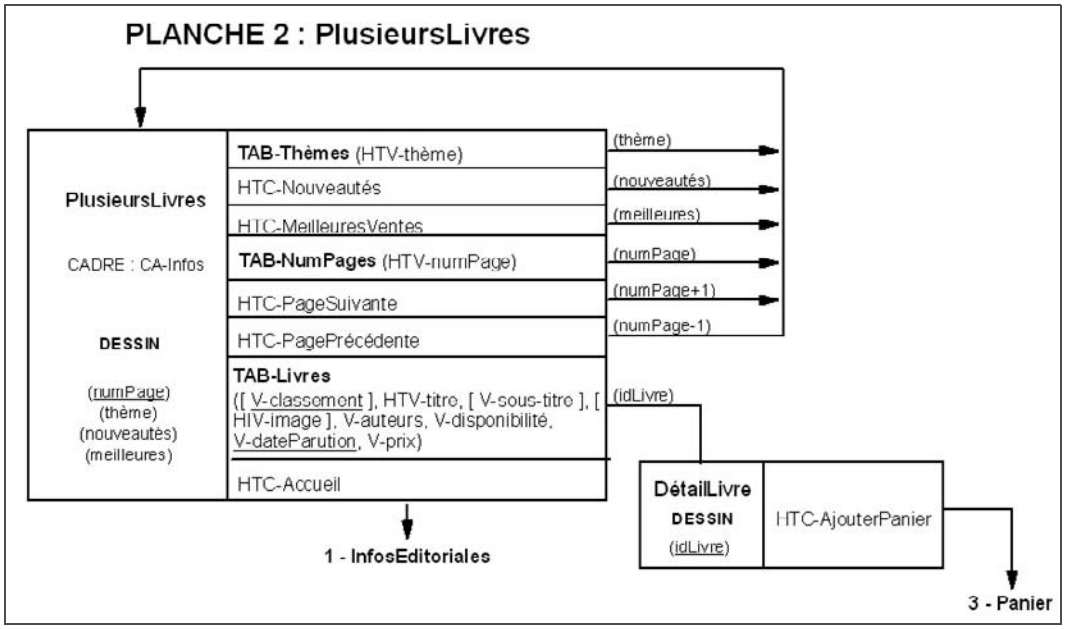


Figure 6–19
Suite du SEP MACAO : PlusieursLivres

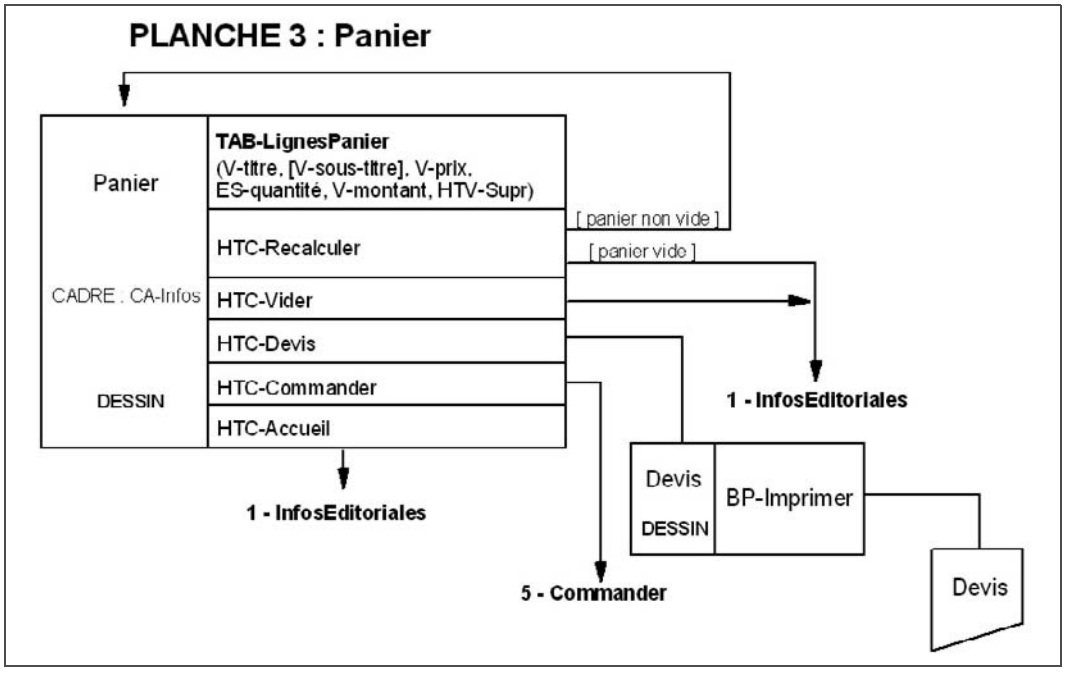
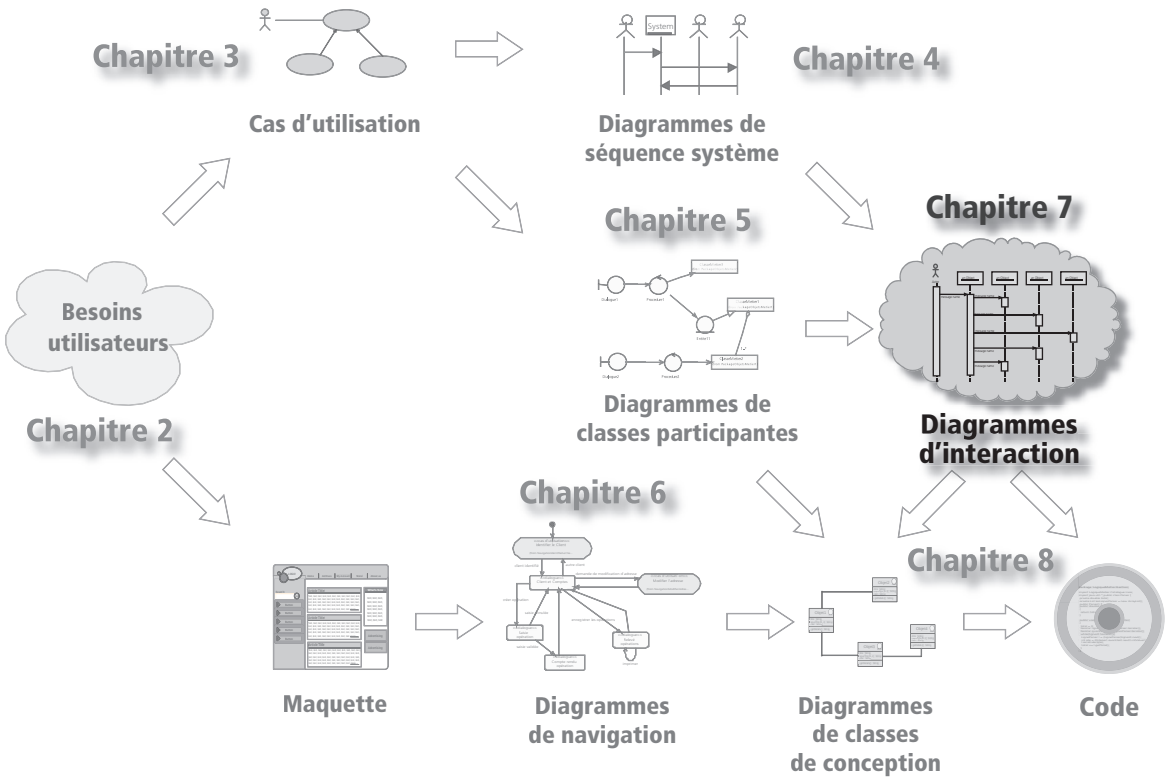


Figure 6–20
Suite du SEP MACAO : Panier

chapitre 7


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Diagrammes de classes participantes
Chapitre 7
Diagrammes d'interaction
Besoins utilisateurs
Chapitre 2
Chapitre 6
Maquette
Diagrammes de navigation
Chapitre 8
Diagrammes de classes de conception
Code

Conception objet préliminaire

À présent, nous allons attribuer des responsabilités précises de comportement aux classes d'analyse identifiées au chapitre 5. Nous représenterons le résultat de cette étude dans des diagrammes de séquence UML. Nous construirons également une vue statique complétée sous forme de diagrammes de classes de conception préliminaire, indépendamment des choix technologiques qui seront effectués au chapitre suivant.

SOMMAIRE

MOTS-CLÉS

- Diagramme de séquence
- Conception préliminaire
- Diagramme de classes
- Package

Démarche

Rappelons le positionnement de cette activité de conception par rapport à l'ensemble du processus décrit au chapitre 1. Nous avons identifié les cas d'utilisation au chapitre 3 et poursuivi leur description détaillée au chapitre 4, grâce en particulier aux diagrammes de séquence système. Nous avons également réalisé un modèle d'analyse au chapitre 5, concrétisé par des diagrammes de classes participantes. Pour passer maintenant vraiment à la conception, nous allons répartir tout le comportement du système entre ces classes d'analyse et nous décrirons les interactions correspondantes.

L'attribution des bonnes responsabilités aux bonnes classes est l'un des problèmes les plus délicats de la conception orientée objet. Pour chaque service ou fonction, il faut décider quelle est la classe qui va la contenir. Les diagrammes d'interactions (séquence ou communication) sont particulièrement utiles au concepteur pour représenter graphiquement ses décisions d'allocation de responsabilités. Chaque diagramme va ainsi représenter un ensemble d'objets de classes différentes collaborant dans le cadre d'un scénario d'exécution du système. Dans ce genre de diagramme, les objets communiquent en s'envoyant des messages qui invoquent des opérations (ou méthodes) sur les objets récepteurs. Il est ainsi possible de suivre visuellement les interactions dynamiques entre objets, et les traitements réalisés par chacun.

Méthode Étude détaillée de la dynamique des objets

Au chapitre 4, nous avions déjà identifié un certain nombre d'opérations potentielles dans les classes dialogues et entités. Cependant, ce n'était qu'un premier jet, une base de travail.
Dans ce chapitre et le suivant, nous allons vraiment nous atteler au problème crucial de concevoir un ensemble de classes faiblement couplées entre elles et fortement cohérentes. Cela ne peut passer que par une étude détaillée de la dynamique des objets, matérialisée en UML par des diagrammes d'interactions.

Message et opération

Avec les principaux outils de modélisation UML du marché, lorsque vous décrivez un message entre deux objets, vous pouvez également créer du même coup une opération publique sur la classe de l'objet récepteur.

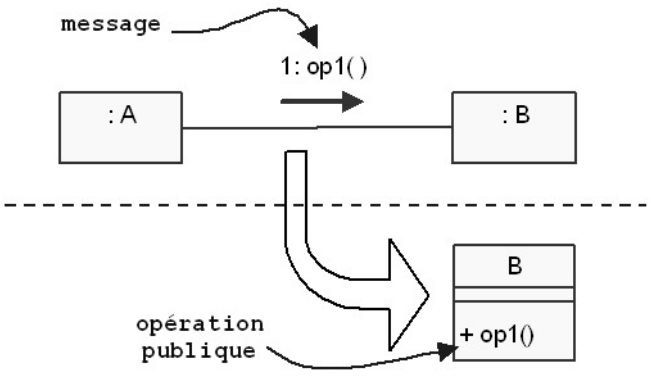


Figure 7–1

Par rapport aux diagrammes de séquence système du chapitre 4, nous allons remplacer le système vu comme une boîte noire par un ensemble d'objets en interaction. Pour cela, nous utiliserons encore dans ce chapitre les trois types de classes d'analyse, à savoir les dialogues, les contrôles et les entités. Nous respecterons également les règles que nous avions fixées sur les relations entre classes d'analyse, mais en nous intéressant cette fois-ci aux interactions dynamiques entre objets :

- Les acteurs ne peuvent interagir (envoyer des messages) qu'avec les dialogues.
- Les dialogues peuvent interagir avec les contrôles.
- Les contrôles peuvent interagir avec les dialogues, les entités, ou d'autres contrôles.
- Les entités ne peuvent interagir qu'entre elles.

Le changement de niveau d'abstraction par rapport au diagramme de séquence système peut ainsi se représenter comme sur la figure 7-2.

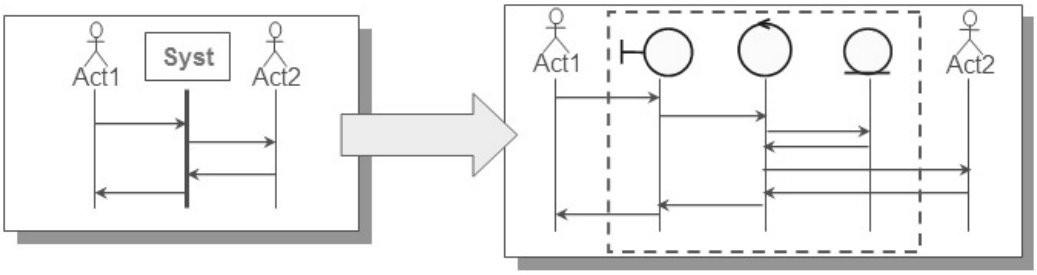


Figure 7–2 Passage de l'analyse à la conception préliminaire

Notation détaillée des diagrammes de séquence

L'expression diagramme d'interactions englobe principalement deux types de diagrammes UML spécialisés, qui peuvent servir tous les deux à exprimer des interactions de messages similaires :

- les diagrammes de communication ;
- les diagrammes de séquence.

UML 2 Autres diagrammes d'interactions

En fait, UML 2 a ajouté deux diagrammes d'interactions : l'interaction overview diagram et le timing diagram. Cependant, nous n'utiliserons pas ces nouveaux types de diagrammes pour rester conformes à notre volonté de ne pas complexifier inutilement.

Les diagrammes de séquence représentent les interactions dans un format où chaque nouvel objet est ajouté en haut à droite. On représente la ligne de vie de chaque objet par un trait pointillé vertical. Cette ligne de vie sert de point de départ ou d'arrivée à des messages représentés eux-mêmes par des flèches horizontales (voir figure 7-3). Par convention, le temps coule de haut en bas. Il indique ainsi visuellement la séquence relative des envois et réceptions de messages, d'où la dénomination : diagramme de séquence.

B.A.-BA Notations

Notez la représentation des spécifications d'activation (aussi appelées focus of control) – bandes blanches qui représentent les périodes d'activité sur les lignes de vie des objets. Remarquez également les flèches en pointillé de retour d'invocation d'une opération.

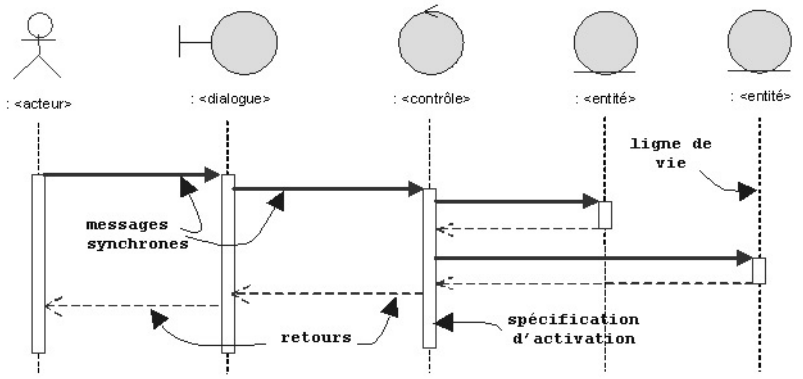


Figure 7–3
Notations du diagramme de séquence

Les diagrammes de communication illustrent les interactions entre objets sous forme de graphes ou de réseaux. Les objets peuvent être placés en tout point du diagramme. Ils sont connectés par des liens qui indiquent qu'une forme de navigation et de visibilité entre ces objets est possible. Tout message entre objets est représenté par une expression et une petite flèche indiquant son sens de circulation. Chaque lien permet le trafic de plusieurs messages et chaque message est assorti d'un numéro d'ordre.

B.A.-BA Numérotation décimale

Notez la numérotation décimale qui montre l'imbrication des messages, d'une façon comparable à la représentation des spécifications d'activation sur le diagramme de séquence précédent.

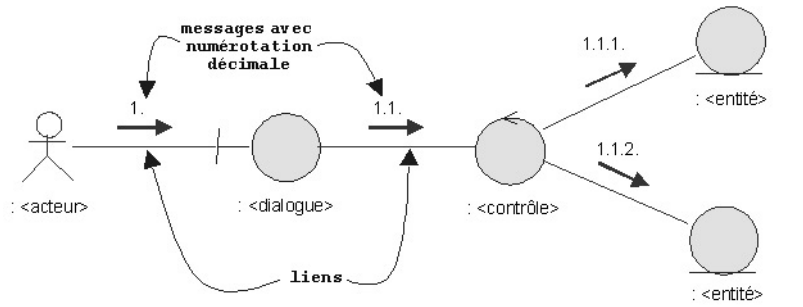


Figure 7–4
Notations du diagramme de communication

Nous avons fait le choix de n'utiliser que le diagramme de séquence dans la suite de ce livre, afin de réduire au maximum l'effort d'apprentissage du langage UML.

De plus, les nouveautés introduites par UML 2 au niveau du diagramme de séquence, et que nous présenterons au fur et à mesure de leur utilisation, n'ont pas d'équivalent dans le diagramme de communication. Nous pensons que le diagramme de communication sera du coup de moins en moins utilisé par les modélisateurs…

Les diagrammes d'interactions font participer des objets et non pas des classes. Il ne s'agit d'ailleurs pas forcément d'occurrences spécifiques, mais plus généralement de rôles possédant un type. Du coup, UML n'utilise pas directement la notation des instances dans les diagrammes d'objets (`nomObjet:NomClasse`), mais une notation légèrement différente, sans soulignement (`nomRôle:NomType`). En l'absence d'un nom de rôle, on fait simplement précéder le nom du type du symbole « : » (`:NomType`).

Tout message est bon pour créer une instance, mais la convention UML que nous appliquerons ici veut que l'on utilise à cet effet un message standardisé appelé `create`. Le message `create` peut comprendre des paramètres d'initialisation. Il peut s'agir, par exemple, d'un appel de constructeur avec paramètres en Java ou en C#.

De même, lorsque nous souhaiterons montrer explicitement la destruction des objets, nous utiliserons le message standardisé `destroy`.

Le message de création a une représentation particulière (flèche pointillée ouverte) sur le diagramme de séquence. L'événement de destruction est pour sa part représenté par une croix noire qui termine la ligne de vie détruite. Dans l'exemple de la figure 7-5, l'objet 2 est créé et détruit durant le scénario, contrairement aux objets 1 et 3 qui préexistent et survivent au scénario concerné. Le message `m11` est imbriqué dans le message `m1`.

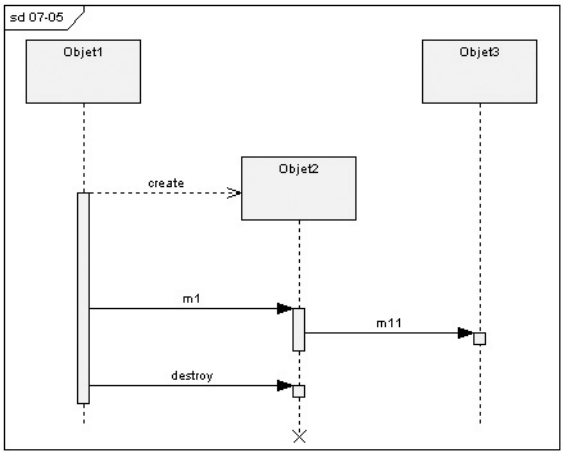


Figure 7–5
Création et destruction d'objet dans le diagramme de séquence

Méthode — Seulement le diagramme de séquence

Nous avons fait le choix de n'utiliser que le diagramme de séquence dans la suite de ce livre, afin de réduire au maximum l'effort d'apprentissage du langage UML.

Langages — Destruction des objets

Les langages objets récents comme Java et C# ont un garbage collector qui détruit automatiquement les objets qui ne sont plus utilisés. Ce n'est pas le cas en C++ où les objets doivent être détruits explicitement.

Notez la flèche pointillée ouverte pour le message de création et la croix noire indiquant la terminaison de la ligne de vie, suivant la spécification UML 2.

Diagrammes d'interactions des cas d'utilisation de l'internaute

Pour illustrer notre démarche, nous allons étudier les deux premiers cas d'utilisation majeurs de l'internaute, à savoir :

- `Chercher des ouvrages` ;
- `Gérer son panier`.

Pour chaque cas d'utilisation, nous réaliserons un diagramme de séquence ou plusieurs représentant notre choix d'allocation de responsabilités dynamiques.

Chercher des ouvrages

L'internaute veut trouver le plus rapidement possible un ouvrage recherché dans l'ensemble du catalogue. Il veut également pouvoir consulter la fiche détaillée d'un ouvrage particulier avant de la mettre dans son panier.

Étudions un scénario nominal de recherche avancée par nom d'auteur.

Sur la page d'accueil du site www.jeBouquine.com, l'internaute choisit le lien vers la page de recherche avancée. Il saisit une phrase de recherche (par exemple ici un nom d'auteur). Notez que la vérification syntaxique de la phrase de recherche est de la responsabilité de la classe dialogue elle-même (et pas du contrôle associé qui n'est invoqué que dans le cas favorable où il n'y a pas d'erreur de syntaxe). Le contrôle délègue ensuite à une entité la recherche proprement dite. Quelle est la classe la mieux placée pour effectuer une recherche parmi l'ensemble des ouvrages du catalogue ? C'est le catalogue lui-même puisqu'il contient tous les livres. Le `Catalogue` construit alors une collection dynamique de livres (`resultats`) correspondant à la recherche, qu'il renvoie au contrôle. Celui-ci initialise le dialogue chargé du résultat, en lui passant la collection en paramètre. L'internaute peut ensuite naviguer dans les différentes pages du résultat.

B.A.-BA Message à soi-même

Un objet peut s'envoyer via un lien un message à lui-même.

C'est le cas du dialogue `RechercheAvancée` qui s'envoie le message `verifierSyntaxe`. Il s'agit d'un traitement interne à l'objet, mais important car son résultat influe sur la suite du scénario. Nous avons donc envie de le représenter même s'il ne s'agit pas d'une interaction entre objets.

Notez que ce traitement interne se traduit généralement par une méthode privée (`private`) en Java, C++ ou C#, alors que la réception d'un message venant d'un autre objet correspond forcément à l'invocation d'une méthode publique (`public`).

B.A.-BA Multi-objet/collection

Pour indiquer que le catalogue contient une collection de livres, nous utilisons une notation particulière : `nomRôle:TypeCollection<Type>`.

Le `nomRôle` sera au pluriel pour bien exprimer l'idée de multi-objet. Le `TypeCollection` sera par convention un des mots-clés suivants : `Set` (ensemble, non ordonné, non trié), `List` (ensemble ordonné) ou `Map` (ensemble trié). Il s'agit là d'une convention très répandue, naturelle pour les concepteurs objet. La notation entre crochets est inspirée des generics de la dernière version Java 5. Nous utiliserons également des noms de messages génériques comme `find()` ou `add()` sur les multi-objets.

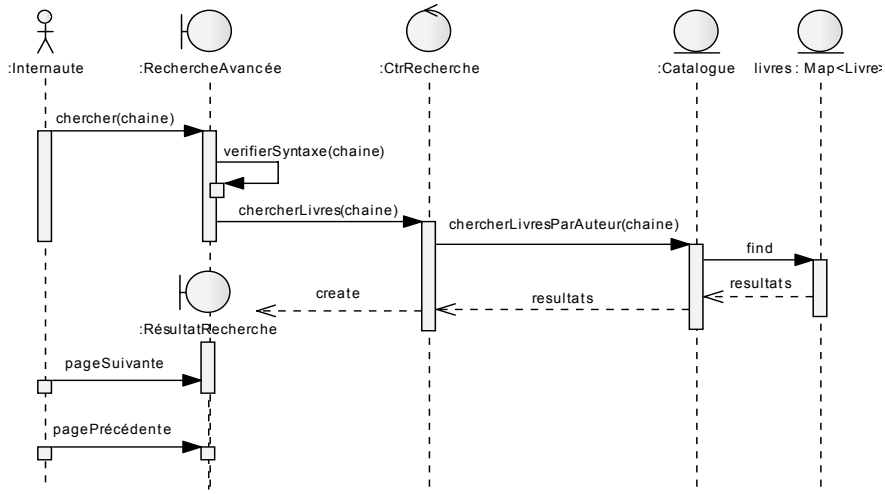


Figure 7–6
Diagramme de séquence du scénario nominal de recherche avancée

Nous avons représenté le scénario nominal sur le diagramme de la figure 7-6. Cela signifie que tout « se passe bien » : la syntaxe de la phrase de recherche est correcte et la recherche aboutit à un ensemble d'ouvrages. Les cas d'erreur ont été abordés au chapitre 6 sur la navigation. Nous pourrions donc tout à fait représenter l'échec de la recherche avec un autre diagramme comme illustré sur la figure 7-7.

ÉTUDE DE CAS **Deux cas d'erreur**

La distinction est bien claire sur le diagramme entre les deux cas d'erreur :

- erreur de syntaxe dans la recherche (chaîne 1 erronée) détectée directement par le dialogue sans intervention du contrôle ;
- aucun ouvrage trouvé suite à la recherche (chaîne 2 syntaxiquement correcte) dans le catalogue.

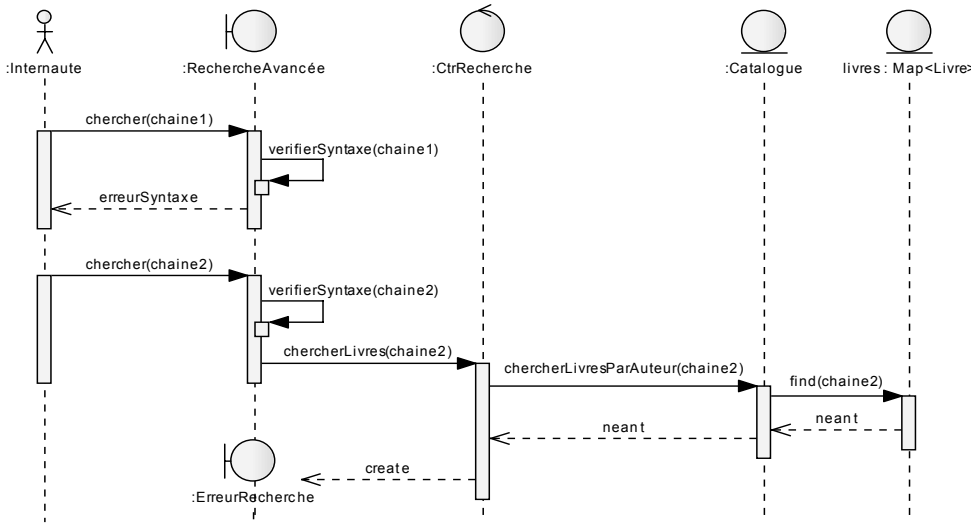


Figure 7–7
Diagramme de séquence des scénarios d'erreur de recherche avancée

Revenons au scénario nominal. Nous l'avons provisoirement terminé lorsque l'internaute navigue dans les pages de résultats. Il peut ensuite demander l'affichage du détail d'un livre sélectionné parmi ceux de la page de résultats. Que se passe-t-il alors ? Le dialogue passe la main à un contrôle spécialisé (`CtrLivre`) qui sait récupérer les informations détaillées d'un livre à partir de son identifiant. Pour cela, il fait encore appel à « l'expert » des livres, à savoir l'entité catalogue. Ensuite, le contrôle crée un nouveau dialogue de fiche détaillée avec ces informations. C'est par exemple à ce moment-là que l'internaute choisit de mettre le livre sélectionné dans son panier virtuel.

Voilà ce que montre le diagramme 7-8 qui nous permet de faire la jonction avec le cas d'utilisation `Gérer son panier`.

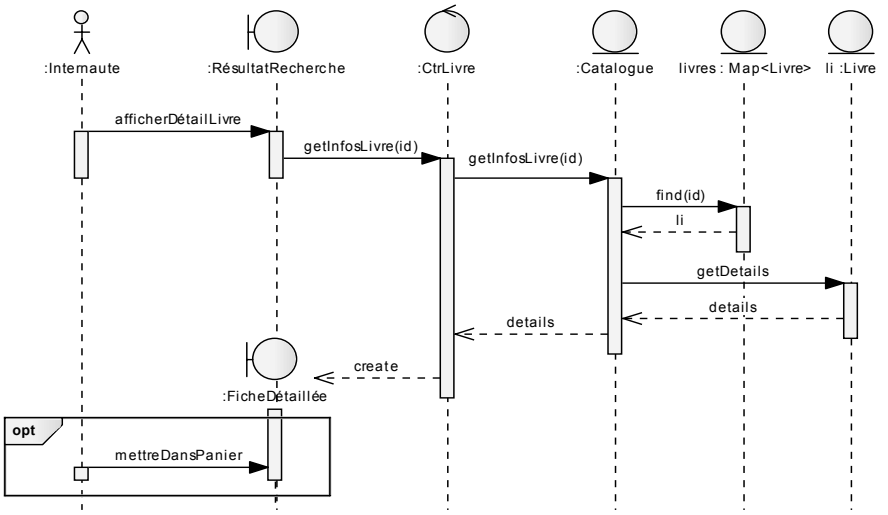


Figure 7–8
Diagramme de séquence de la suite du scénario nominal de recherche avancée

Gérer son panier

Lorsque l'internaute est intéressé par un ouvrage, il doit avoir la possibilité de l'enregistrer dans un panier virtuel. Ensuite, il doit pouvoir ajouter d'autres livres, en supprimer ou encore en modifier les quantités avant de passer commande.

Dans le diagramme 7-8, nous avons laissé l'internaute au moment où il met de côté un premier livre dans son panier virtuel. Que se passe-t-il derrière le dialogue concerné ? Celui-ci passe la main à un contrôle spécialisé dans la gestion du panier. Ce contrôle a la responsabilité de créer le panier lors de la première sélection, mais aussi toutes les lignes du panier au fur et à mesure. Il est également responsable d'afficher un dialogue particulier récapitulant le panier en cours et permettant à l'internaute de le modifier et de le recalculer.

ÉTUDE DE CAS Ordre de création des objets

Notez bien l'ordre précis de création de tous les objets liés au panier (entités et dialogue).
L'initialisation du panier provoque la création en cascade de la collection vide de lignes du panier. Cette collection n'est d'ailleurs pas persistante : elle ne survit pas à une session de l'internaute.
Lors de la création des lignes suivantes du panier, il suffira que le contrôle omette le message `create` vers le panier. Cela court-circuitera également le message imbriqué de création de la collection.

Tout cela est représenté sur la figure 7-9.

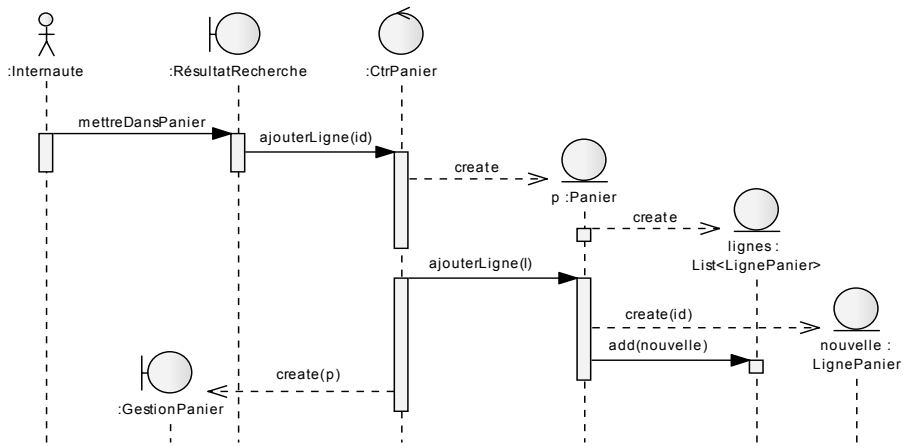


Figure 7–9
Diagramme de séquence du scénario nominal de création de la première ligne du panier

Continuons maintenant en considérant un scénario dans lequel l'internaute modifie la quantité d'un ouvrage sélectionné, puis demande un recalcul de son panier.

Le contrôle reçoit une collection de quantités et la passe à l'entité `panier`. Celle-ci est responsable de la gestion de ses lignes. Elle va donc demander à chaque ligne de se recalculer individuellement en lui passant en paramètre la quantité qui la concerne. Si cette quantité a été positionnée à zéro par l'internaute, la ligne est supprimée.

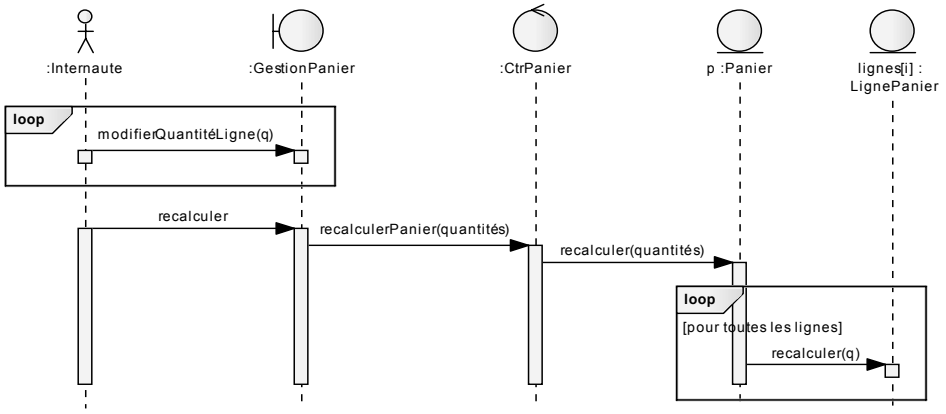


Figure 7–10
Diagramme de séquence du scénario de recalcul du panier

Pour terminer la gestion du panier, considérons enfin un scénario dans lequel l'internaute supprime explicitement une ligne du panier puis le vide totalement.

B.A.-BA **Itération sur une collection (loop)**

Un algorithme très courant consiste à opérer une itération (ou boucle) sur tous les éléments d'une collection en envoyant un message à chacun d'eux. Nous avons déjà vu qu'une collection est modélisée sur le diagramme de séquence par la notation `nomRôle:TypeCollection<Type>`. Pour indiquer que nous considérons maintenant un élément quelconque d'une collection, nous allons utiliser la notation particulière : `nomRôle[i]:Type`. L'opérateur de boucle (`loop`) à l'intérieur d'un fragment d'interaction représenté par un rectangle autour d'un message indique que celui-ci est adressé à chaque élément de la collection. C'est le cas dans le diagramme 7-10 pour le message `recalculer(q)` qui est envoyé par le panier à chacune de ses lignes.

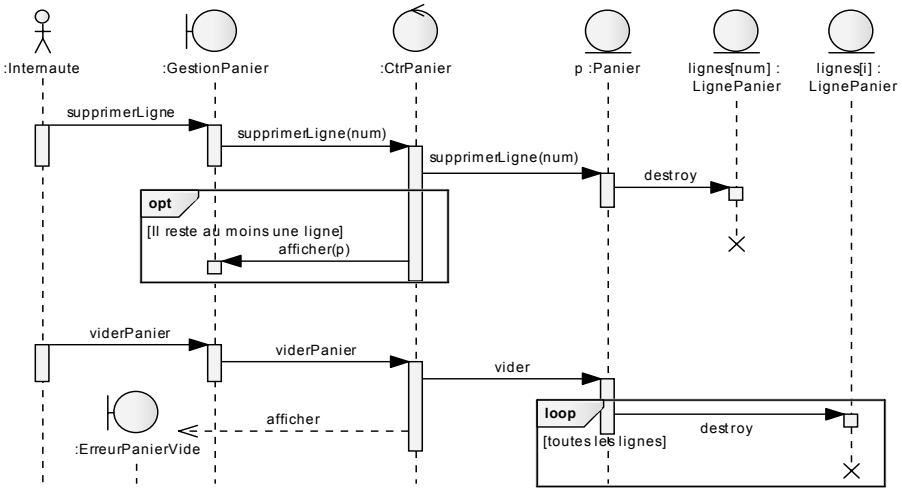


Figure 7–11
Diagramme de séquence d'un scénario de vidage du panier

ÉTUDE DE CAS **Destruction explicite**

Notez les premières utilisations de la notation de destruction explicite d'un objet sur le diagramme de séquence, lors de la suppression d'une ligne du panier, ou de toutes les lignes. (figure 7-11).

Dans le cas où l'internaute demande à vider son panier, celui-ci n'est pas supprimé : seules le sont les lignes qu'il contient (grâce à la boucle `destroy`). En effet, le panier n'est supprimé qu'avec la fin de la session de l'internaute.

ATTENTION **Type des attributs**

Ces types ne sont pas encore ceux d'un langage de programmation, puisque nous voulons être encore indépendants des choix technologiques à ce niveau. Le typage définitif sera effectué au chapitre 8. Nous utiliserons bien des noms comme `String` ou `Date` dans les diagrammes qui suivent ; il ne s'agit pas des types Java, mais bien de types « universels » !

Classes de conception préliminaire

Nous avons traité en détail les deux premiers cas d'utilisation majeurs de l'internaute, à savoir `Chercher des ouvrages` et `Gérer son panier`.

En partant du modèle d'analyse (voir le chapitre 5), nous allons affiner et compléter les diagrammes de classes participantes obtenus précédemment. Pour cela nous utiliserons les diagrammes de séquence que nous venons de réaliser pour :

- Ajouter ou préciser les opérations dans les classes (un message ne peut être reçu par un objet que si sa classe a déclaré l'opération publique correspondante).
- Ajouter des types aux attributs et aux paramètres et retours des opérations.
- Affiner les relations entre classes : associations (avec indication de navigabilité), généralisations ou dépendances.

MÉTHODE Liens durables ou temporaires

Un lien durable entre éléments va donner lieu à une association navigable entre les classes correspondantes ; un lien temporaire va donner lieu à une relation de dépendance.

Sur l'exemple du schéma 7-12, le lien entre les éléments `:A` et `:B` devient une association navigable entre les classes correspondantes. Le fait que l'élément `:A` reçoive en paramètre d'un message une référence sur un objet de la classe `C` induit une dépendance entre les classes concernées.

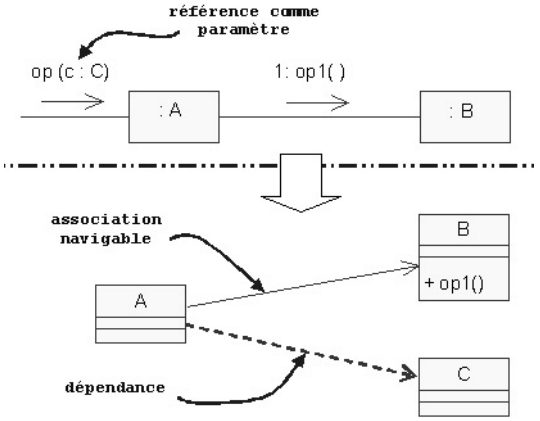


Figure 7–12
Liens temporaires et dépendances

Nous utiliserons également le travail réalisé au chapitre 6 sur la navigation pour ajouter les éventuels dialogues qui manquaient.

Chercher des ouvrages

Repartons du diagramme de classes participantes de `Chercher des ouvrages`, tel qu'il était au chapitre 5 (figure 5-24).

Qu'avons nous appris de nouveau sur ces classes aux chapitres 6 et 7 ?

Tout d'abord, le chapitre 6 nous a montré qu'il existe deux dialogues supplémentaires : l'un correspondant à l'erreur `aucun ouvrage trouvé`, l'autre permettant d'afficher la page détaillée d'un ouvrage.

Ensuite, les figures 7-6 à 7-8 nous amènent à ajouter un certain nombre d'opérations aux classes existantes, par exemple `verifierSyntaxe(chaine)` à la classe dialogue `RechercheAvancee`, ainsi que des opérations importantes dans les classes entités.

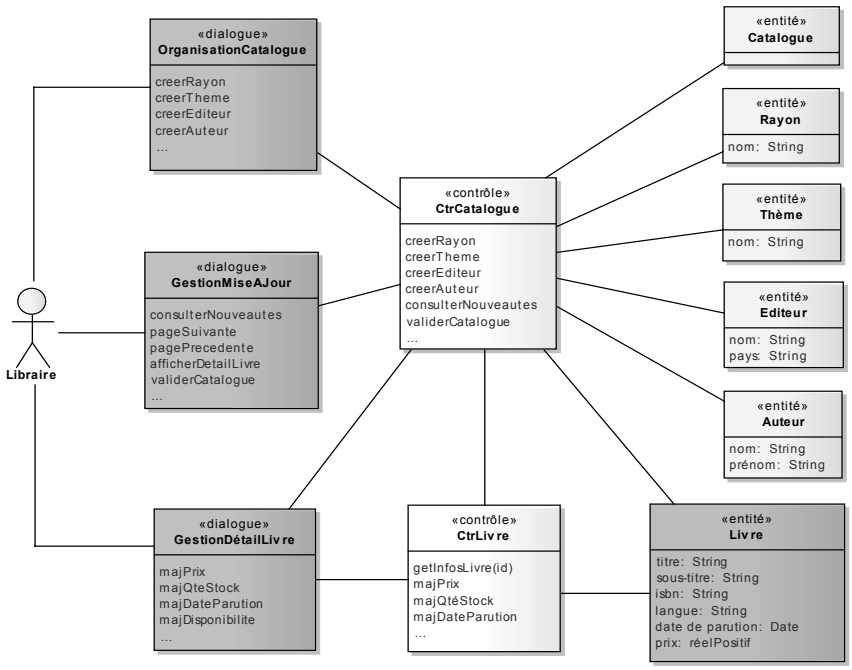


Figure 7–13
DCP de Chercher des ouvrages

ÉTUDE DE CAS **Quelques remarques sur le diagramme de classes de conception préliminaire**

- Les multi-objets ne sont pas représentés en tant que classes collections de façon à garder le plus longtemps possible notre indépendance vis-à-vis du langage de programmation cible. Du coup les opérations génériques sur les collections (comme `find` ou `add`) n'apparaissent pas.
- Nous ne faisons pas figurer les opérations systématiques de création ou destruction d'instances, ainsi que les accesseurs des attributs (`get` et `set`). Cela permet de garder des diagrammes lisibles et qui contiennent seulement les comportements les plus importants.
- Les navigabilités des associations sont limitées à une seule direction dans la mesure du possible. C'est le cas entre les contrôles et les entités, entre les contrôles et certains dialogues, mais aussi entre les entités elles-mêmes. Une seule association entre entités a été laissée bidirectionnelle : celle entre les classes `Livre` et `Auteur`. En effet, l'opération `getDetails()` sur un livre va chercher le nom et le prénom de l'auteur. La classe `Livre` doit donc pouvoir naviguer vers la classe `Auteur`. À l'inverse, pour l'opération `chercherLivresParAuteur()` du catalogue, nous avons gardé la possibilité de passer par l'objet auteur qui doit alors connaître ses livres. L'association est donc pour l'instant bidirectionnelle, mais un choix plus restrictif pourra être effectué en conception détaillée.

Le diagramme de séquence 7-8 précise que le contrôle `CtrLivre` doit passer par l'entité `Catalogue` pour accéder à un livre. L'association existante doit donc être modifiée. De plus, les sens de circulation des mes-

sages nous permettent de limiter la navigabilité de certaines associations entre entités.

Le résultat de toutes ces cogitations est donné sur le diagramme de classes de conception de la figure 7-14, à comparer avec le diagramme de classes d'analyse participantes.

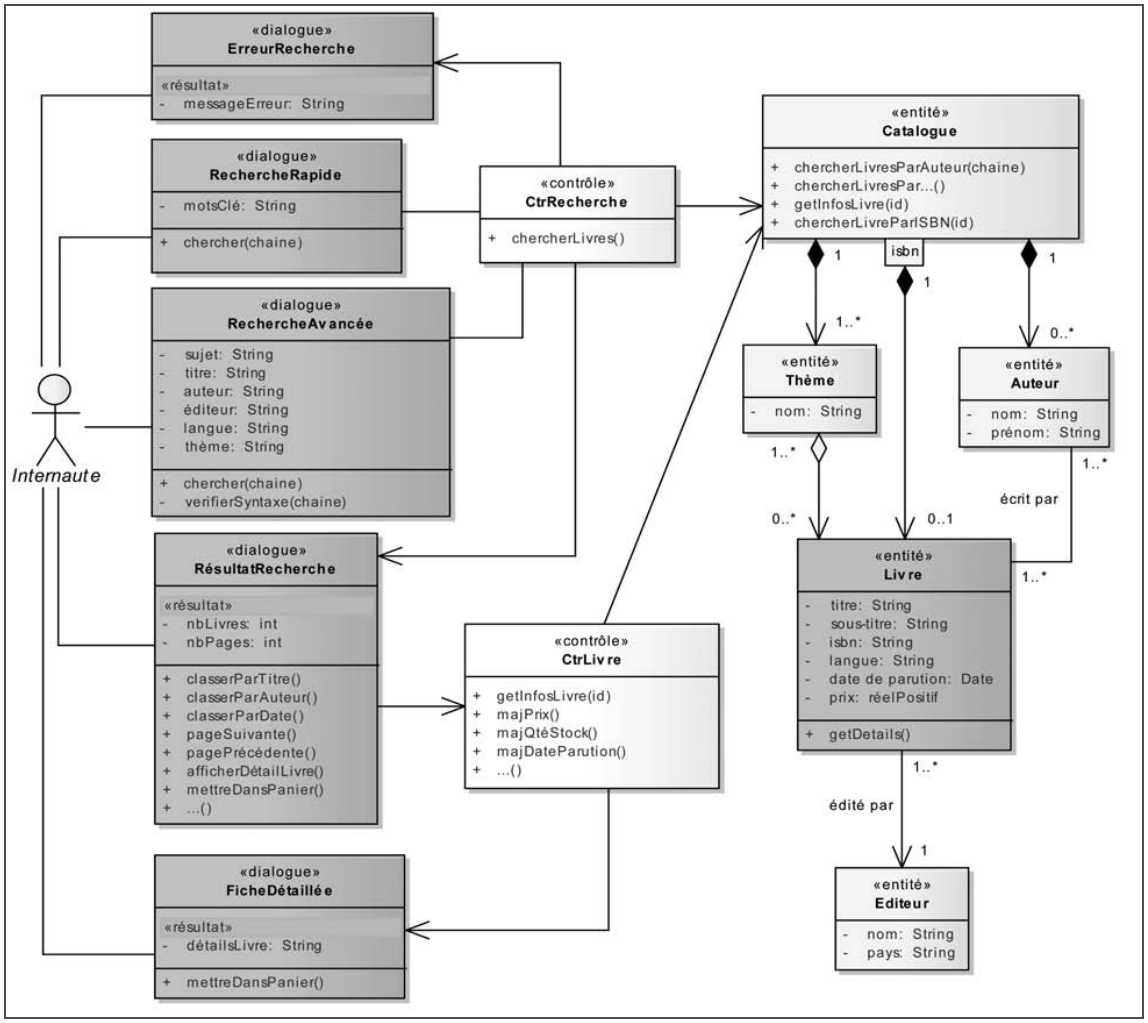


Figure 7–14 Diagramme de classes de conception de Chercher des ouvrages

Gérer son panier

Repartons là aussi du diagramme de classes participantes de `Gérer son panier`, tel qu'il était au chapitre 5 (figure 5-25).

Qu'avons nous appris de nouveau sur ces classes aux chapitres 6 et 7 ?

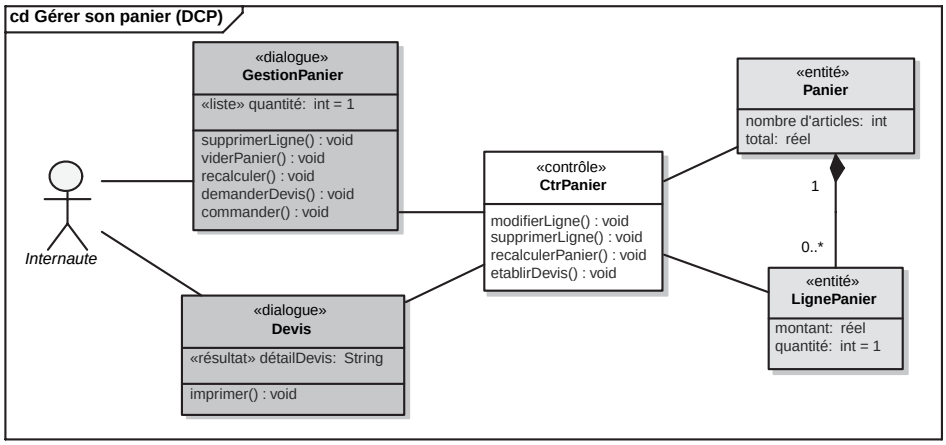


Figure 7–15
DCP de Gérer son panier

Tout d'abord, le chapitre 6 nous a montré qu'il existe deux dialogues supplémentaires : l'un correspondant à l'erreur `Panier vide`, l'autre permettant d'afficher le devis pour impression par l'internaute.

Ensuite les figures 7-9 à 7-11 nous amènent à ajouter un certain nombre d'opérations aux classes existantes, par exemple `ajouterLigne(id)` à `CtrPanier` et à l'entité `Panier`. Nous voyons sur la figure 7-9 que le dialogue `ResultatRecherche` peut appeler `CtrPanier` comme d'ailleurs tous les dialogues qui possèdent le bouton `Mettre dans le panier` associé à au moins une référence d'ouvrage. Le contrôle passe toujours par le panier et n'accède pas directement aux lignes.

Le diagramme de classes de conception résultant est fourni par la figure 7-16, à comparer avec le diagramme de classes d'analyse participantes.

Revenons maintenant sur la création d'une ligne du panier. Sur la figure 7-9, nous l'avons simplement représentée par un message `create` venant du `Panier`. Or, nous avons vu au chapitre 5 que chaque ligne doit être reliée à un livre pour pouvoir afficher correctement ses informations. Le plus simple consiste à passer en paramètre du message `create` une référence sur le livre concerné. Pour obtenir cette référence à partir d'un identifiant (comme l'ISBN) fourni par le dialogue au contrôle, celui-ci doit d'abord interroger le `Catalogue` puisque celui-ci est l'expert des livres.

Le diagramme de séquence détaillé de la création d'une ligne de panier (en supposant que le panier existe déjà) est présenté sur le diagramme 7-17.

Ce diagramme de séquence va nous donner l'occasion d'illustrer le concept de dépendance entre classes, expliqué sur le diagramme 7-12.

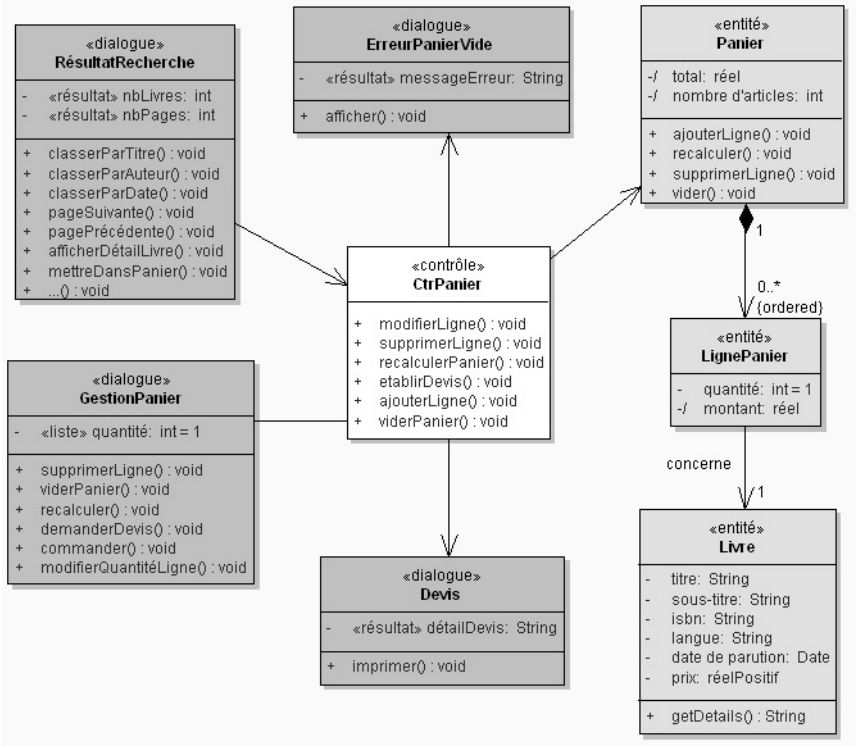


Figure 7–16
Diagramme de classes de conception de Gérer son panier

ÉTUDE DE CAS Association bidirectionnelle entre GestionPanier et CtrPanier

Remarquez l'association bidirectionnelle entre le dialogue `GestionPanier` et `CtrPanier`. Les messages pouvant circuler dans les deux sens d'après les diagrammes d'interactions, nous n'avons pas pu éviter de laisser cette double navigabilité.

La contrainte `{ordered}` sur la composition entre `Panier` et `LignePanier` indique que les instances de `LignePanier` sont ordonnées sans obliger à indiquer tout de suite une solution technologique (numéro, etc.).

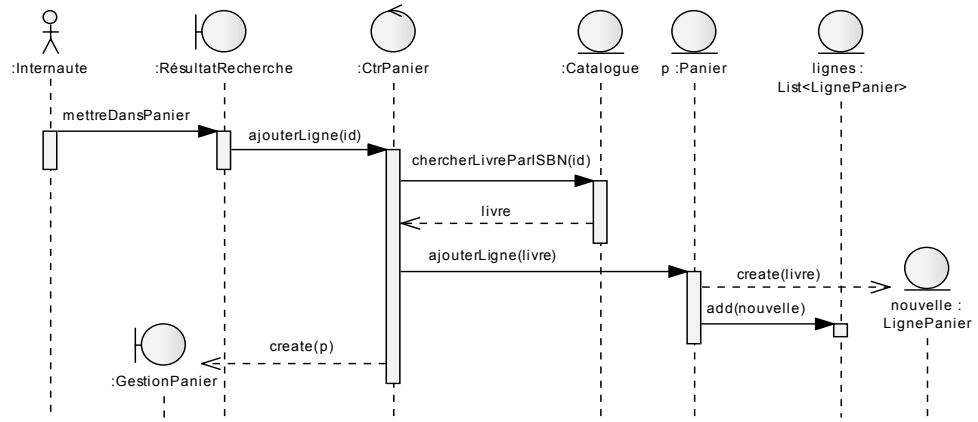


Figure 7–17
Diagramme de séquence détaillé de la création d'une ligne de panier

L'objet `:CtrPanier` récupère une référence temporaire sur un objet livre par l'intermédiaire du catalogue. Il passe ensuite ce livre en paramètre au panier `p` pour que ce dernier puisse créer sa ligne. La méthode `ajouterLigne()` de la classe `Panier` prend donc en paramètre une instance de `Livre` et non pas un simple identifiant.

Il existe ainsi une dépendance (de type local) entre les classes `CtrPanier` et `Livre`, ainsi qu'une (de type paramètre) entre `Panier` et `Livre`. Les modifications mènent au diagramme de classes de la figure 7-18.

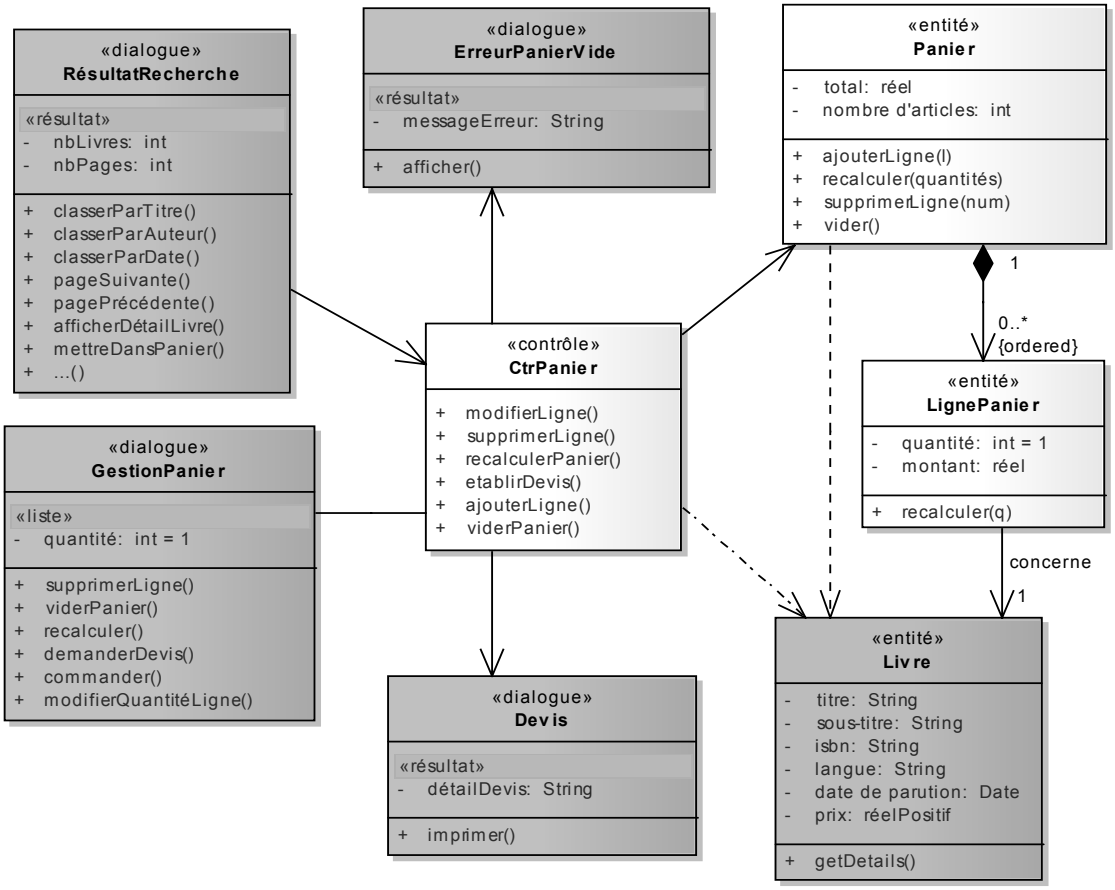


Figure 7–18
Diagramme de classes de conception complété de Gérer son panier

Structuration en packages de classes

Démarche

Pour structurer notre modèle, nous allons organiser les classes et les regrouper en ensembles cohérents. Pour ce faire, nous utilisons une fois de plus le concept général d'UML, le package.

Les systèmes informatiques modernes sont organisés en couches horizontales, elles-mêmes découpées en partitions verticales. Cette découpe est d'abord logique, puis éventuellement physique en termes de machines.

Nous allons donc structurer les classes identifiées jusqu'à présent en trois couches principales :

- une couche Présentation, rassemblant toutes les classes dialogues ;
- une couche Logique Applicative, rassemblant toutes les classes contrôles ;
- une couche Logique Métier, rassemblant toutes les classes entités ;

L'architecture logique de notre étude de cas est ainsi représentée par un premier diagramme de packages, comme illustré par la figure 7-19.

B.A.-BA Package

Package : mécanisme général de regroupement d'éléments en UML, qui est principalement utilisé en analyse et conception objet pour regrouper des classes et des associations.

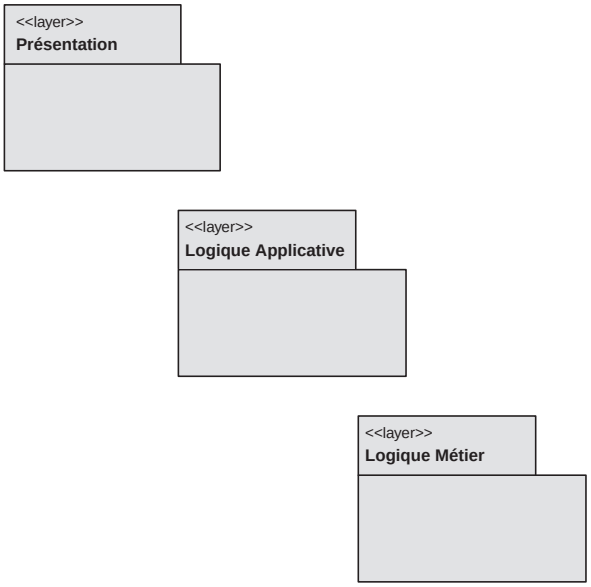


Figure 7–19
Diagramme de packages de l'architecture logique

B.A.-BA **Architecture en couches**

Le principal objectif de la séparation en couches est d'isoler la logique métier des classes de présentation (IHM), ainsi que d'interdire un accès direct aux données stockées par ces classes de présentation. Le souci premier est de répondre au critère d'évolutivité : pouvoir modifier l'interface de l'application sans devoir modifier les règles métier, et pouvoir changer de mécanisme de stockage sans avoir à retoucher l'interface, ni les règles métier.

Une architecture en couches se décrit en UML par un diagramme structurel qui ne montre que des packages et leurs dépendances. UML 2 a souligné l'importance de ce type de diagramme de haut niveau en officialisant le diagramme de packages comme un type de diagramme à part entière. Vous pouvez utiliser le mot-clé `layer` pour distinguer les packages qui représentent les couches.

▸ http://www.sparxsystems.com

Si nous répartissons maintenant les classes identifiées précédemment, nous obtenons une structuration comme celle illustrée sur la copie d'écran 7-20, issue de l'outil Enterprise Architect.

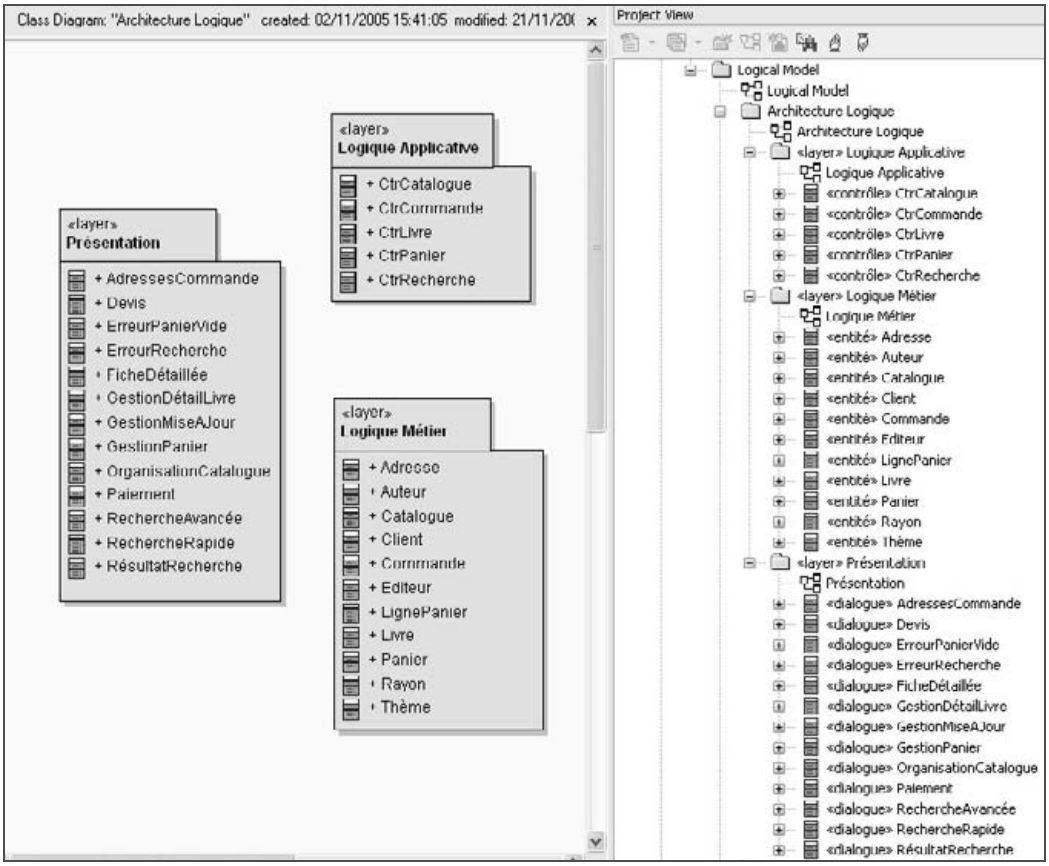


Figure 7–20 Détail de l'architecture logique

La structuration des couches `Présentation` et `Logique Applicative` peut s'effectuer en tenant compte des cas d'utilisation. La structuration de la couche `Logique Métier` est à la fois plus délicate et plus intéressante, car elle fait appel à deux principes fondamentaux : cohérence et indépendance.

Le premier principe consiste à regrouper les classes qui sont proches d'un point de vue sémantique. Un critère intéressant consiste à évaluer les durées de vie des instances et à rechercher l'homogénéité. Par exemple, les entités `Thème`, `Editeur`, `Auteur` et `Rayon` ont une durée de vie de plusieurs mois, voire de plusieurs années. En contrepartie, les concepts de `Panier` et `LignePanier` sont beaucoup plus volatiles. Il est donc naturel de les séparer dans des packages différents.

Le deuxième principe s'efforce de minimiser les relations entre packages, c'est-à-dire plus concrètement les relations entre classes de packages différents.

Les considérations précédentes amènent à une solution naturelle consistant à découper le modèle en trois packages comme indiqué sur la figure 7-21. Les packages ainsi constitués vérifient bien les principes de forte cohérence interne et de faible couplage externe.

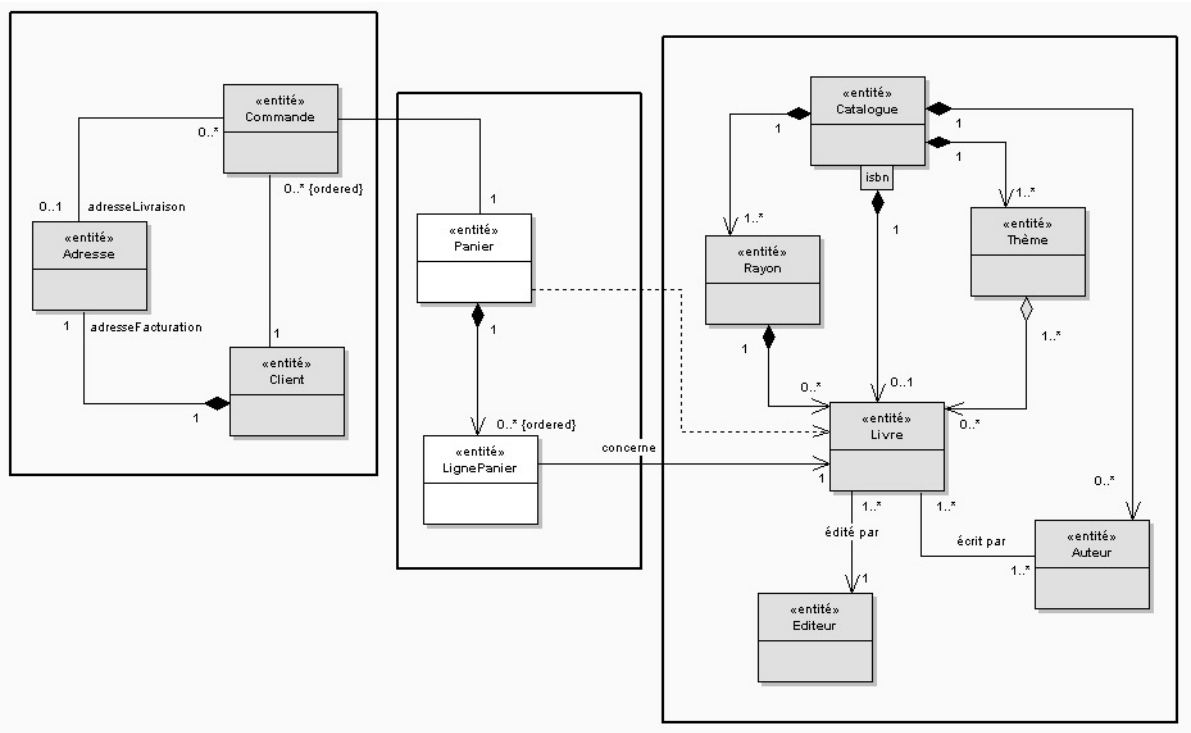


Figure 7–21 Proposition de découpage en packages

CONCEPT AVANCÉ **Espace de noms**

Les packages sont des espaces de noms : deux éléments ne peuvent pas porter le même nom au sein du même package. En revanche, deux éléments appartenant à des packages différents peuvent porter le même nom. Du coup, UML propose une notation complète pour un élément : `nomPackage::nomElément`.

Remarquons que les associations qui traversent les packages sont déjà unidirectionnelles, sauf celle entre les entités `Commande` et `Panier`, car nous n'y avions pas encore réfléchi. Or, une association bidirectionnelle entre des classes appartenant à des packages différents induirait une dépendance mutuelle entre ces derniers. Il est donc indispensable de réduire la navigabilité de l'association concernée à une seule direction. Dans notre exemple, c'est clairement la `Commande` qui a besoin du `Panier` et pas le contraire : nous pouvons donc affiner le diagramme précédent comme indiqué sur la figure 7-22. Nous en profiterons pour réaliser la structuration en packages, qui va se traduire par la nouvelle notation des noms de classes.

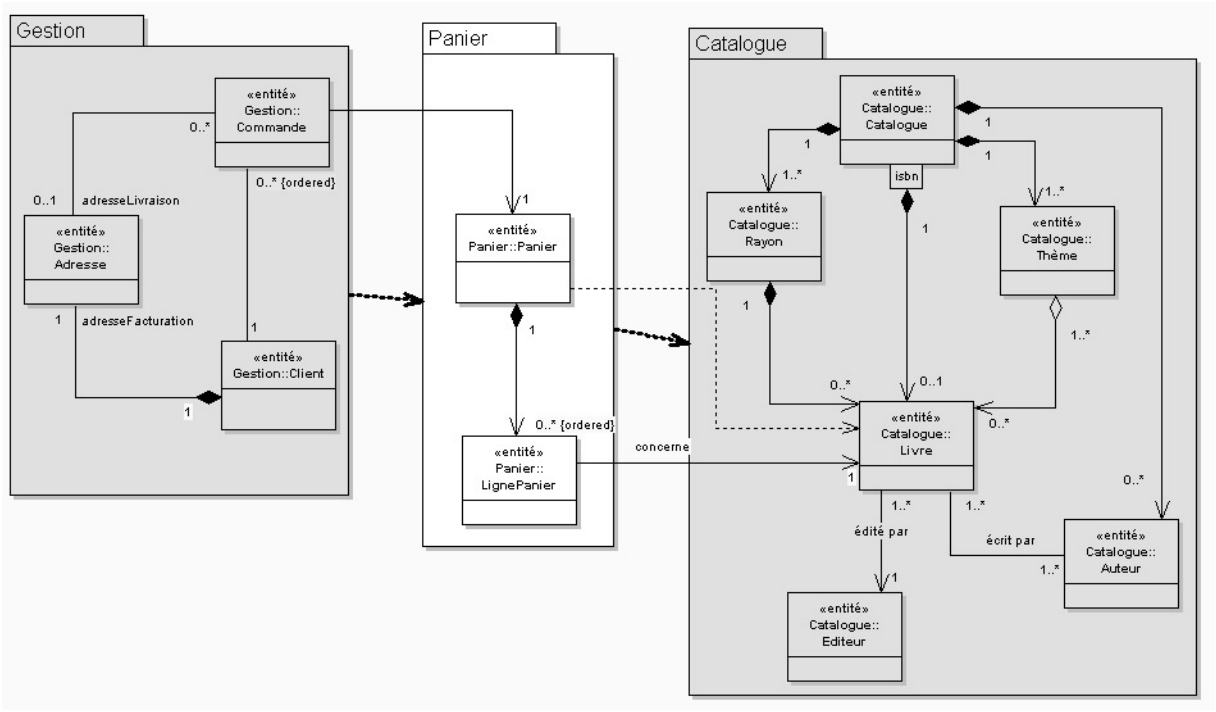


Figure 7–22 Découpage en packages montrant leur dépendances

Diagrammes de classes des packages de la couche métier

Terminons ce chapitre en peaufinant les diagrammes de classes de chacun des packages.

Dans le package `Catalogue`, nous pouvons ajouter le concept de `Collection`. Un éditeur peut proposer des collections qui regroupent des livres, mais tous les livres n'appartiennent pas forcément à une collection. Nous ne fixons pas la navigabilité des nouvelles associations pour l'instant.

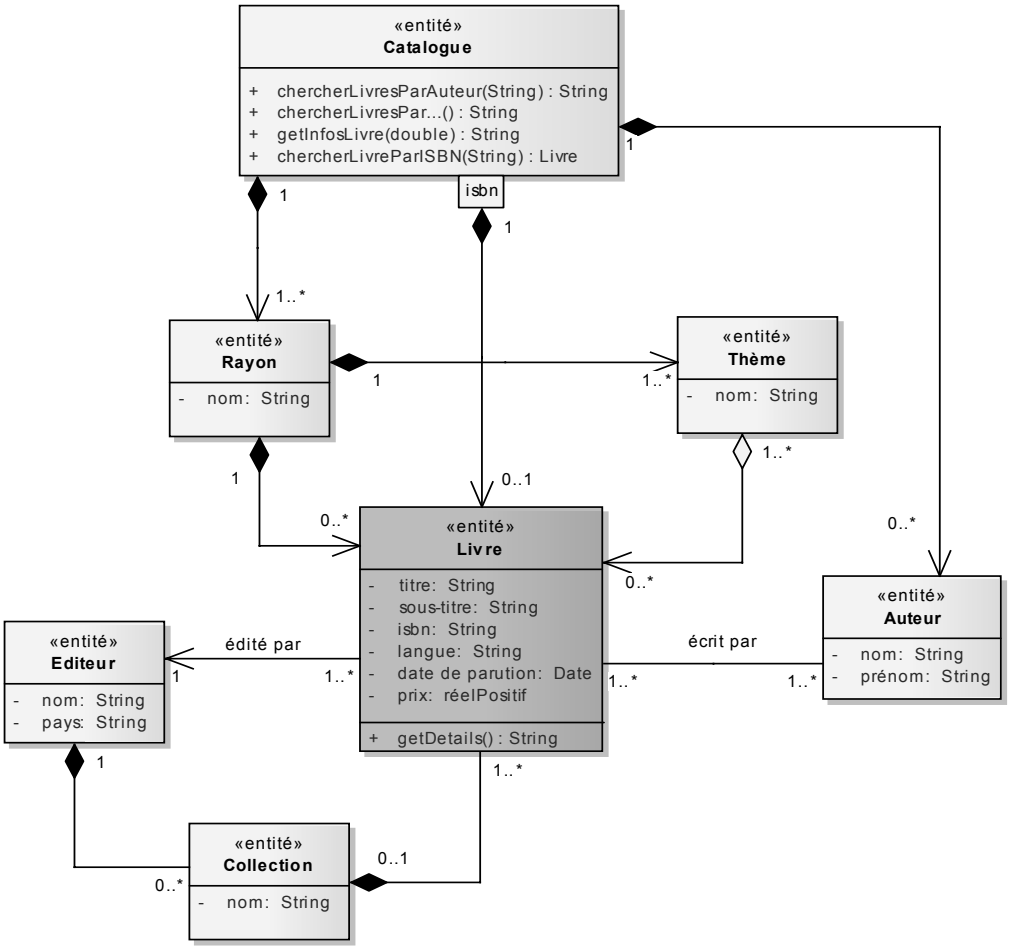


Figure 7–23
Diagramme de classes du package Catalogue

Pour le package intitulé `Gestion`, nous pouvons remarquer que nous avions omis le concept de carte bancaire associée à un client. Si nous voulons pouvoir mémoriser les informations d'une ou plusieurs carte(s) bancaire(s) pour chaque client, il faut faire apparaître la classe entité correspondante.

Il est également important de montrer les classes utilisées (comme `Panier` pour le package `Gestion`) et surtout l'association qui pointe dessus dans le diagramme de classes du package utilisateur. Le diagramme de classes résultat est représenté sur la figure 7-24.

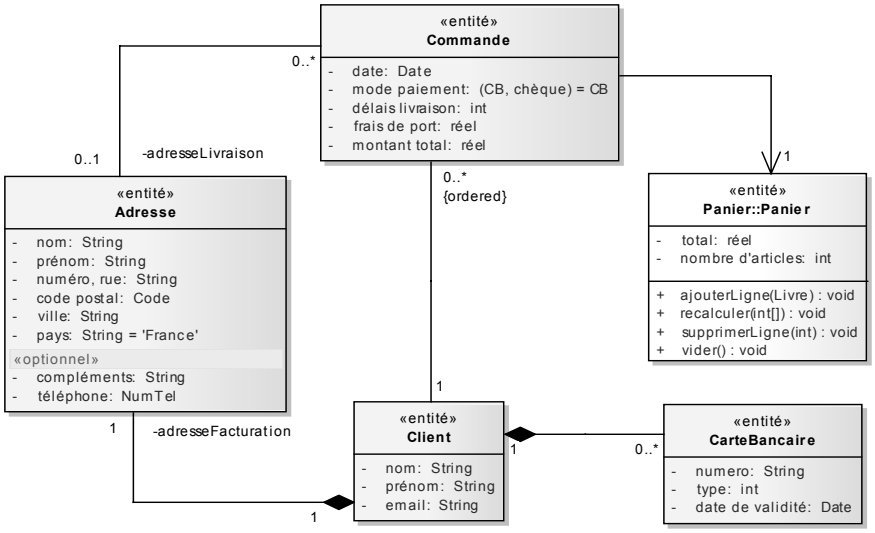


Figure 7–24
Diagramme de classes du package Gestion

Pour compléter, le diagramme de classes du package `Panier` est également montré sur la figure suivante.

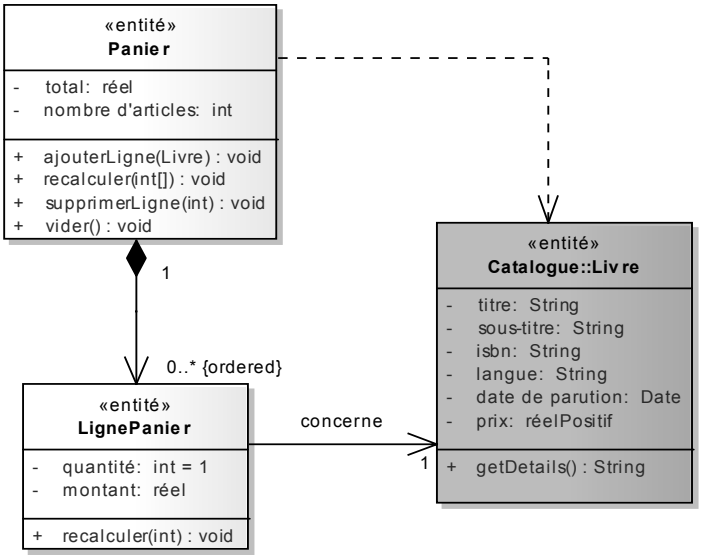


Figure 7–25
Diagramme de classes du package Panier

> CONCEPT AVANCÉ **Attribut dérivé**
>
> Remarquez que nous n'avons plus aucun attribut dérivé dans notre modèle de conception ! En effet, il s'agit d'un concept utile pour l'analyste (comme expliqué au chapitre 5), mais pas pour le concepteur. Celui-ci doit décider comment l'implémenter : attribut stocké avec méthode de mise à jour, ou juste une méthode de calcul à la demande sans stockage de valeur.

Les dépendances entre packages de la couche métier sont représentées sur le diagramme synthétique de la figure 7-26.

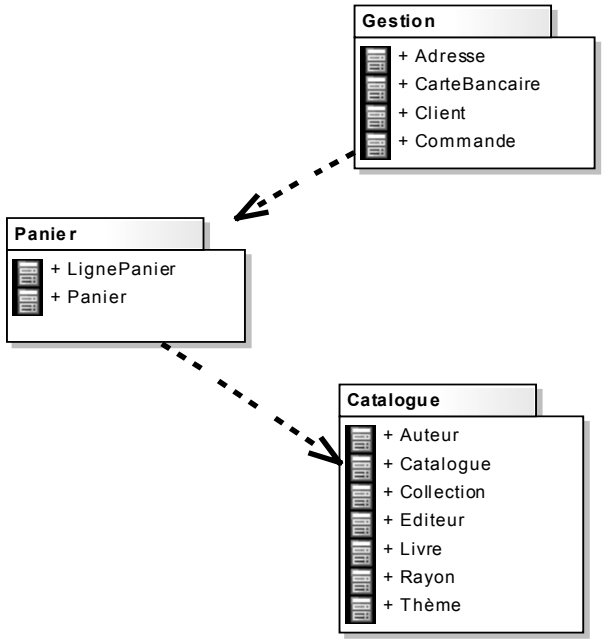
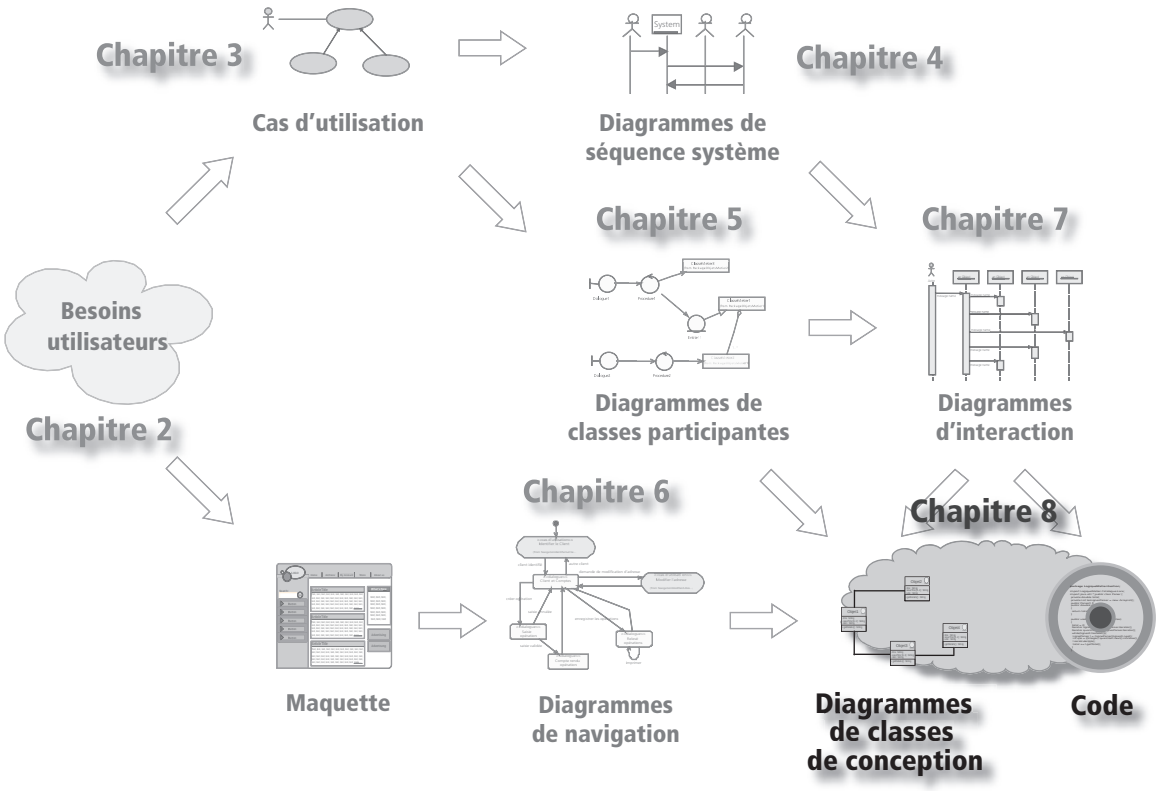


Figure 7–26
Diagramme de packages de la couche Logique Métier

chapitre 8


Chapitre 3
Cas d'utilisation
Chapitre 4
Diagrammes de séquence système
Chapitre 5
Diagrammes de classes participantes
Chapitre 7
Diagrammes d'interaction
Besoins utilisateurs
Chapitre 2
Chapitre 6
Diagrammes de navigation
Chapitre 8
Maquette
Diagrammes de classes de conception
Code

Conception objet détaillée

Dressons d'abord un panorama synthétique de l'éventail des technologies disponibles actuellement pour construire des applications web. Ensuite, nous enrichirons les différents diagrammes réalisés au chapitre 7 en fonction des choix technologiques effectués. Nous finirons en donnant des exemples de code pour montrer que nous avons atteint l'objectif initial du livre : aller des besoins utilisateurs au code de l'application !

SOMMAIRE

MOTS-CLÉS

- Architecture
- Couche
- Package
- Framework
- PHP
- Java
- J2EE
- JSP
- Servlet
- Struts
- JSF
- C#
- .NET
- ASP
- XUL
- AJAX

Démarche

Rappelons le positionnement de cette dernière activité de conception par rapport à l'ensemble du processus décrit au chapitre 1. Il s'agit d'un affinement de ce que nous avons réalisé de façon générique au chapitre 7. Cela revient à dire que nous allons maintenant incorporer dans nos diagrammes le choix d'architecture et les choix technologiques qui vont modifier les classes de conception préliminaire, les préciser, ajouter de nouvelles classes plus techniques, etc.

Nous allons détailler en particulier comment se traduisent les trois types de classes d'analyse avec les différentes solutions technologiques. Un contrôle, par exemple, peut devenir plus concrètement une page PHP, une servlet Java, une classe CodeBehind associée à un ASP.NET, etc.

N'oublions pas l'objectif principal du modèle de conception détaillée qui est de pouvoir être traduit directement en code !

Architecture des applications web

B.A.-BA **Pattern architectural**

Un pattern architectural est l'expression d'un schéma fondamental d'organisation pour des systèmes logiciels. Il inclut un ensemble de sous-systèmes prédéfinis, précise leurs responsabilités et prescrit des règles et des conseils pour organiser leurs relations.
Un site très intéressant sur les architectures, mais aussi sur Java, .NET, Ruby, etc. :
▸ http://www.infoq.com

Patterns architecturaux

Le nombre impressionnant de produits et de technologies liés à Internet disponibles actuellement a suscité des architectures d'application web multiples et variées. Cependant, une application d'e-commerce digne de ce nom implique l'existence d'au moins quatre composants d'architecture significatifs :

- le navigateur client,
- le serveur web,
- le serveur d'applications,
- le serveur de données.

À un haut niveau, on peut identifier dans les applications web d'aujourd'hui plusieurs *patterns* architecturaux.

Les plus courants actuellement sont les suivants :

- Le **client web très léger et universel** est employé pour les applications destinées à Internet, pour lesquelles la configuration du client n'est pas maîtrisable. Le client ne nécessite qu'un navigateur web standard et la logique métier ainsi que la logique de présentation sont intégralement exécutées sur le serveur :
  - Seul le langage HTML est utilisé côté client, ce qui maximise l'accessibilité au site (tous les navigateurs web sont aptes à y naviguer) et la sécurité du poste client.

  - Ce pattern implique toutefois d'innombrables allers-retours entre navigateur et serveur web : cela touche donc l'interactivité.
- Le **client web léger** (pattern le plus classique aujourd'hui, voir également l'encadré sur le client « riche ») correspond également aux applications Internet/intranet pour lesquelles la configuration du client n'est pas maîtrisable, à ceci près que l'on requiert côté client un navigateur web assez récent, comprenant le langage JavaScript. Le client navigue sur des pages dotées d'intelligence (programmée en JavaScript), donc :
  - L'interactivité est améliorée (la validation des formulaires est réalisée côté client, avant l'envoi de requêtes sur le serveur web).
  - La plupart des navigateurs modernes proposent des options fines sur ce qu'ils autorisent JavaScript à faire ou non (par exemple, lui interdire de désactiver le menu contextuel ou de changer le texte de la barre d'état, d'ouvrir des fenêtres trop fréquemment, etc.).
  - La portabilité des pages est assez facile à garantir (par exemple par l'emploi de bibliothèques telles que Prototype).

Cependant, Javscript doit être utilisé pour le confort : l'accessibilité implique que s'il est désactivé, l'application doit rester utilisable avec une ergonomie raisonnable ou au pire tolérable.

- Le **client web alourdi** embarque dans les pages web des composants plus complexes : ActiveX, applets Java, plug-ins, ce qui permet à une partie significative de la logique métier d'être exécutée sur le poste client. Du coup :
  - L'interface graphique peut être beaucoup plus évoluée (graphiques dynamiques...).
  - Le composant peut se rafraîchir spontanément (et télécharger des données brutes du serveur).
  - Le déploiement, bien qu'automatique, est beaucoup plus lourd. Les prérequis sur les postes sont plus importants.
  - Le risque est beaucoup plus important (typiquement avec les ActiveX et les plug-ins qui s'exécutent sans restriction sur le système d'exploitation).

B.A.-BA Applet

Une applet est un programme Java dont le code est téléchargé sur le poste client depuis le serveur web et s'exécute ensuite dans le navigateur en utilisant un interpréteur Java (la machine virtuelle, ou JVM, qui est intégrée dans la plupart des navigateurs). Le modèle de sécurité des applets est très strict mais évolutif depuis Java 2 (utilisation des certificats). Autant le HTML pur et les techniques de script peuvent être utilisés par des non-informaticiens (des graphistes par exemple), autant l'utilisation de Java (pour être efficace) nécessite des connaissances préalables en conception et programmation objet.

B.A.-BA HTML

HTML (*HyperText Markup Language*) est un langage de description de contenu fondé sur des balises. Elles permettent de spécifier l'apparence d'un document lors de son affichage ou de son impression. Les balises HTML autorisées sont normalisées par le W3C.

▸ http://www.w3c.org

Les pages HTML sont des documents écrits en langage HTML. Elles sont composées de balises, de texte et de références à d'autres ressources : autres pages HTML (liens), données multimédia (images, sons, etc.), contenus actifs (applet, ActiveX, plug-in).

B.A.-BA JavaScript

JavaScript est un langage léger, mais relativement complexe et puissant, qui apporte des fonctions dynamiques à HTML dans les navigateurs. Malgré son nom, il est très différent du langage Java et pas vraiment orienté-objet (pas d'héritage …).

B.A.-BA ActiveX

Ce sont des composants logiciels inclus dans une page HTML et téléchargés par le navigateur en même temps que celle-ci. ActiveX est une technologie propriétaire qui repose sur le COM (*Component Object Model*), l'infrastructure de Microsoft, dans laquelle les développeurs peuvent construire des composants, dans les langages de leur choix, et les partager pour élaborer des systèmes plus complexes et plus puissants.
La tendance actuelle consiste, de plus en plus, à ignorer purement et simplement les ActiveX, qui sont souvent refusés en raison de leur très grande vulnérabilité.

B.A.-BA Plug-in

Programme téléchargé et installé dans un navigateur. Lorsque le navigateur détecte un format de fichier (texte, multimédia, etc.) qu'il reconnaît, il fait appel au "plug-in" concerné pour le traiter.

À RETENIR **Le client web « riche »**

Depuis quelques années maintenant, de nouvelles technologies permettant de développer la couche présentation d'une application web fleurissent à grande vitesse. XUL, AJAX, Flash, JSF, Flex, Laszlo, Eclipse RCP, XAML, etc., les *buzzwords* se multiplient et le choix sur un projet devient de plus en plus difficile !

En ce qui concerne l'architecture, les solutions sont néanmoins assez similaires, avec l'apparition en particulier d'une notion de conteneur sur le poste client. Ce conteneur peut être de natures diverses : navigateur web (AJAX), interpréteur Flash (Flex, Laszlo), machine virtuelle Java, Eclipse.

Un précurseur est XUL : XML User interface Language. XUL est un langage fondé sur XML, qui est utilisé pour décrire la structure de l'application, mais pas les actions et les comportements (il faut pour cela lui adjoindre un langage de script comme JavaScript.

Voir les *Cahiers du programmeur XUL* par J. Protzenko, aux Éditions Eyrolles.

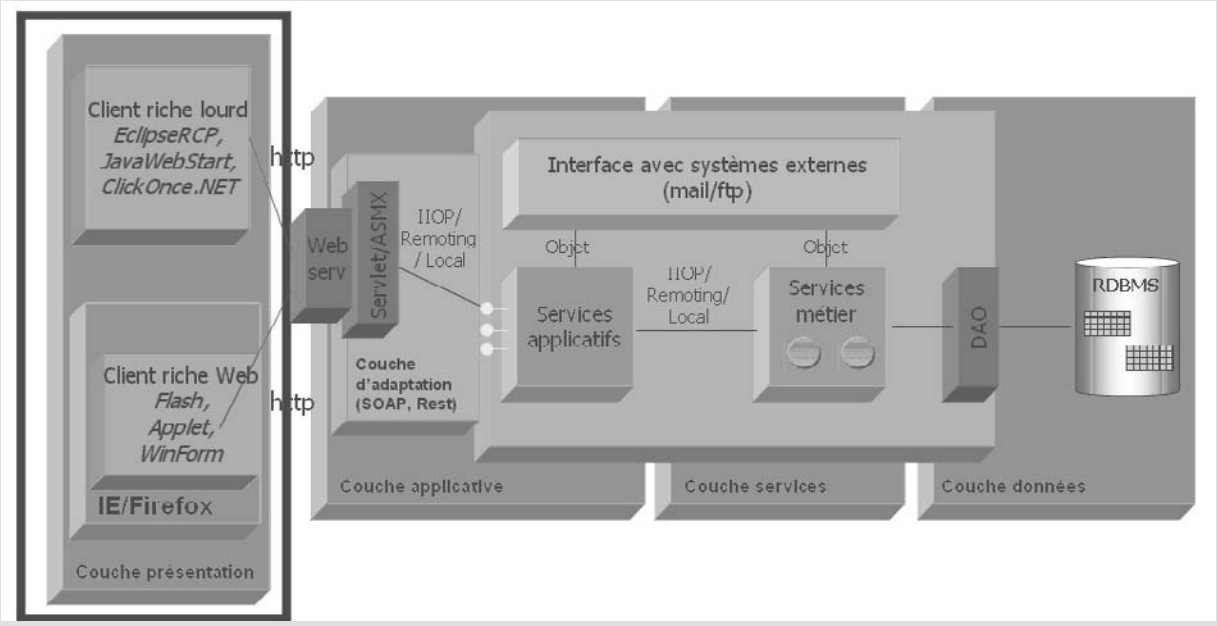


Figure 8–1

Une alternative est proposée par Microsoft avec XAML, basée également sur XML et qui décrit des interfaces graphiques, mais repose sur le framework .NET 3.0 (anciennement nommé WinFX), partie du système d'exploitation Vista.

Citons également AJAX, qui a le vent en poupe ces temps-ci … AJAX signifie Asynchronous JavaScript And XML. Comment cela fonctionne-t-il ? A une requête du client, le serveur répond sous forme de contenu XML, de fragments (X)HTML, du JavaScript, du JSON (JavaScript Object Notation), etc. Cette réponse est analysée par le client qui modifie dynamiquement l'interface sans recharger intégralement la page web, à l'aide de JavaScript, CSS et HTML.

C'est donc une technique qui s'exécute dans le navigateur et qui fait appel à du DHTML (Dynamic HTML).

Pour conclure (provisoirement) sur le sujet en pleine effervescence du client riche, on peut dire qu'il existe de nombreuses solutions différentes avec des techniques et des niveaux de maturité inégaux. Les enjeux dépendent finalement beaucoup du contexte : applications de gestion développées sur un intranet ou sites marchands sur Internet, etc.

- Le **client lourd** est une application à part entière, qui s'exécute sur le poste client. Ce pattern correspond typiquement aux applications Intranet pour lesquelles on maîtrise la configuration du client :
  - La contrainte du déploiement de ces logiciels est moins forte aujourd'hui que par le passé, grâce à l'apparition de techniques de livraison « via le Web » (*web delivery*).
  - La communication client/serveur est moins problématique ; en effet, les protocoles réseau sont pour la plupart persona non grata sur le Web à cause des contraintes imposées par les pare-feu depuis l'avènement des *services web*.

Il est bien sûr possible d'appliquer plusieurs patterns à une même application d'e-commerce. Notre site marchand www.jeBouquine.com pourrait ainsi mettre en œuvre d'une part le pattern du client web léger pour les cas d'utilisation de l'internaute, et d'autre part le client lourd pour les cas d'utilisation des employés. En effet, le système ne contrôle pas la configuration des postes clients des internautes (et nous ne voulons pas restreindre la clientèle possible !), mais maîtrise en revanche totalement les postes clients du `Libraire` ou du `Webmaster`.

Une vue d'ensemble des architectures web (privilégiant les technologies Java) incluant l'intégration de l'existant est présentée sur la figure 8-2.

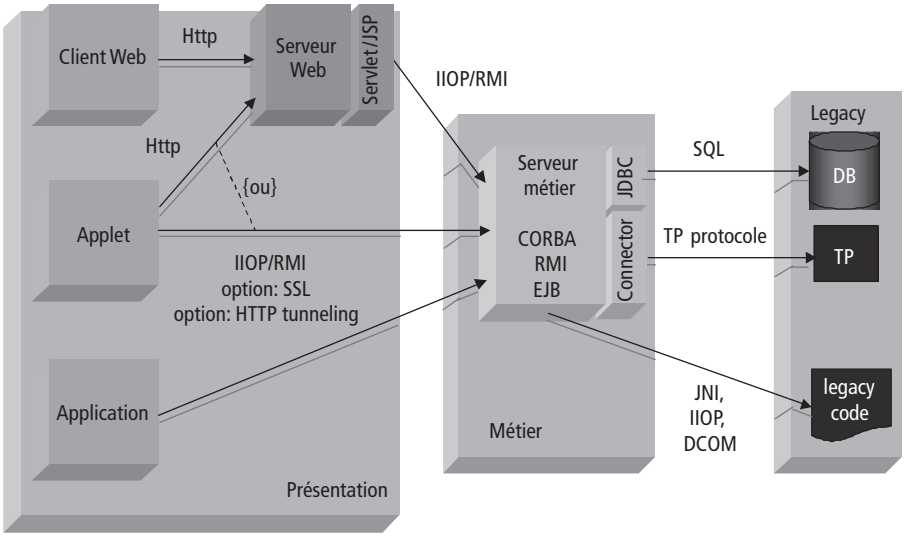


Figure 8–2
Vue d'ensemble des architectures web (avec Java)

Nous détaillons ci-après le pattern du client web léger que nous appliquerons aux cas d'utilisation de l'internaute.

B.A.-BA **Services web**

Un service web (*WebService*) est un service basé sur le Web, utilisant XML (eXtended Markup Language) et plus précisément le format SOAP pour coder à la fois l'emballage du message et le contenu du corps du message. Il en résulte une intégration totalement indépendante du système d'exploitation, du langage ou de tout autre produit middleware utilisé par chacun des composants prenant part au service.
Un service web n'utilise pas forcément du XML. Les technologies initiales, encore les plus répandues, restent XML-RPC et SOAP, mais REST a le vent en poupe, et le format des données est libre.

Le client web léger

Les composants majeurs du pattern architectural client web léger se trouvent sur le serveur. Dans bien des sens, cette architecture est effectivement celle d'une application web minimale.

- Le **navigateur client** est un navigateur HTML standard compatible avec les formulaires et DHTML. Il agit comme un dispositif universel d'interface utilisateur. Son unique fonction supplémentaire peut être d'accepter et de renvoyer des *cookies*. L'utilisateur de l'application requiert des pages HTML auprès du serveur à travers le navigateur. La page renvoyée contient une interface entièrement formatée qui est présentée par le navigateur dans la fenêtre client.
- Le **serveur web** est le point d'accès principal pour tous les navigateurs clients. En fonction de la requête (page HTML statique ou page serveur), des traitements côté serveur peuvent être déclanchés. Dans tous les cas, le résultat est une page HTML affichable par un navigateur HTML standard.
- La **page serveur** est une page qui subit une forme de traitement du côté du serveur. D'une façon typique, ces pages sont implémentées sur le serveur sous la forme de pages de script (ASP, JSP, etc.) qui sont traitées par un filtre sur le serveur d'applications ou par un module exécutable. Ces pages ont potentiellement accès à toutes les ressources du côté serveur ; cela inclut les composants de la logique métier, des bases de données, des systèmes traditionnels (*legacy*) ou des systèmes de paiement.
- Le **serveur d'applications** est le principal exécuteur de la logique métier du côté du serveur. L'exécution du code dans les pages serveur est de son ressort. Il peut se trouver sur la même machine que le serveur web et peut également s'exécuter dans le même espace de processus. Le serveur d'applications est un élément architectural logiquement distinct puisqu'il n'est concerné que par l'exécution de la logique métier et qu'il met donc potentiellement en œuvre des technologies autres que celles du serveur web (EJB, Serviced Components).
- Le **serveur de données** permet de gérer la persistance des objets métier, par exemple dans une base de données relationnelle. Pour la connecter au système, le moyen le plus simple est d'autoriser les scripts des pages serveur à accéder directement au composant de persistance. Cet accès direct passera néanmoins par l'utilisation de bibliothèques standards d'accès aux données, telles que RDO, ADO, ODBC, JDBC, etc. Pour des systèmes plus complexes et plus robustes, on préfère mettre en œuvre une couche objet métier complète. C'est l'optique que nous avons choisie pour notre étude de cas. Comme le système de persistance choisi est une base de données

B.A.-BA Cookie

Petit ensemble d'informations qu'un serveur peut demander à un client de garder, pour lui demander de les restituer par la suite. Une application web peut par exemple utiliser un cookie transitoire pour que le serveur suive la trace d'un navigateur client particulier, tout au long de son parcours parmi les pages du site web. Quant aux cookies permanents, ils servent souvent de tickets d'entrée virtuels évitant au client de ressaisir ses informations d'accès personnelles.

B.A.-BA Quelques définitions

ASP = Active Server Pages (Microsoft)

JSP = Java Server Pages (origine : Sun)
Les JSP utilisent Java. Les ASP sont fondées principalement sur VBScript. Ces pages sont interprétées sur le serveur. Elles peuvent ainsi avoir accès aux ressources de l'entreprise. C'est le résultat de cette exécution qui est ainsi renvoyée au client.

EJB = Enterprise Java Beans
Les Serviced Components sont des composants métier réutilisables hébergés par le serveur d'application de Microsoft : MTS. En ce sens, ils sont une évolution du modèle COM+, adaptée au framework .NET, le pendant des EJB de la plateforme Java.

MTS = Microsoft Transaction Server

RDO = Remote Data Object

ADO = ActiveX Data Object

ODBC = Open Database Connectivity

JDBC = Java Database Connectivity

relationnelle, nous ajoutons une couche (souvent appelée DAO) chargée d'effectuer le *mapping* objet/relationnel.

La figure 8-3 présente la vue logique des principaux composants d'architecture du pattern client web léger. Nous représentons ces composants logiques sous forme de packages UML, en incluant provisoirement le modèle d'analyse dans la couche logique métier.

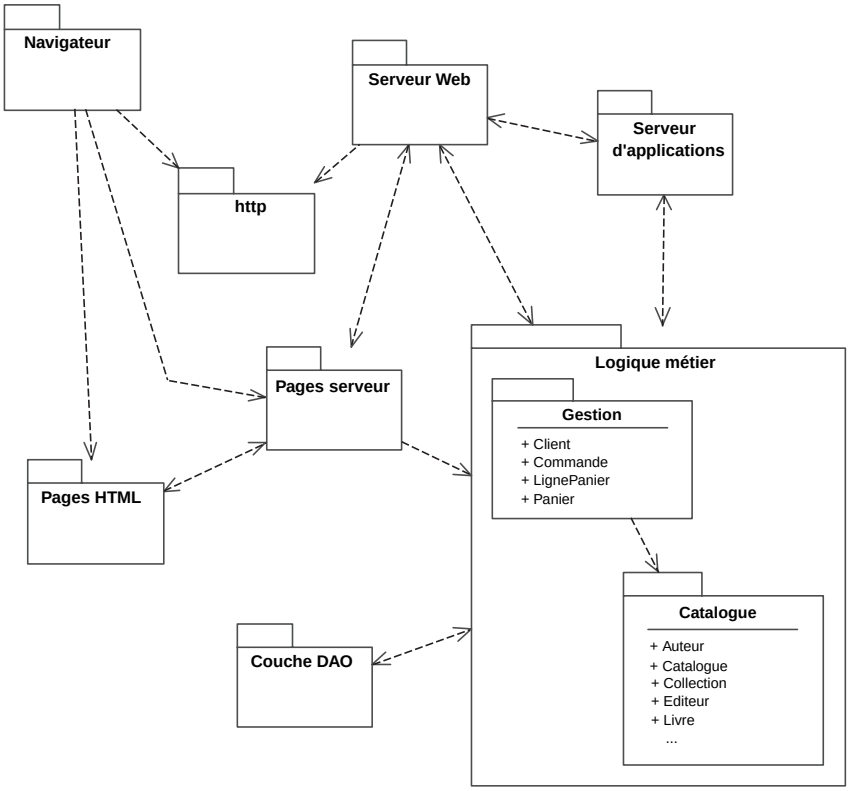


Figure 8–3
Vue logique complète du pattern client web léger

Solutions techniques proposées

Nous n'avons pas pour objectif de présenter dans le détail l'ensemble des technologies web existantes. Cela déborderait de beaucoup le cadre de cet ouvrage. Cependant, nous pensons qu'il est utile d'en brosser un tableau synthétique afin de bien comprendre les transformations que nous allons appliquer aux diagrammes d'interactions et aux diagrammes de classes du chapitre précédent.

Différentes technologies existent pour développer des applications web, par ordre d'apparition :

- les **CGI**, ou *Common Gateway Interface*, qui ont permis de développer les premiers sites web dynamiques ;

Cette technologie permet d'appeler un programme externe au serveur web lors d'une requête. Ce programme peut accéder à toutes les ressources nécessaires, comme une base de données, et construire la page en fonction de la requête. Néanmoins, les CGI ont certains inconvénients : mixage de code HTML et de code de programmation qui rend la maintenance difficile, surcharge mémoire et latence du service avec le lancement d'un nouveau processus pour chaque requête, etc. Les CGI ne sont donc pas adaptés à la création d'applications web importantes.

- les extensions propriétaires de serveurs web (**ISAPI** chez Microsoft, **NSAPI** chez Netscape, modules **PHP**, **Perl** chez Apache, etc.) ;
- les **ASP** (Microsoft) : pages HTML et VBScript interprété ;
- les **ASP.NET** (Microsoft) : pages HTML et code .NET (VB ou C#) compilé ;
- les **servlets** Java, qui sont un peu le pendant des applets, mais sur le serveur web. Elles n'ont pas d'IHM, sont rapides et performantes. Elles sont également portables sous réserve que les serveurs web concernés possèdent une machine virtuelle Java embarquée.
- les **JSP (Java)**.

Nous allons décrire dans la suite de ce chapitre deux grands types de solutions techniques :

1. des solutions « simples » à base de langages de script comme PHP (ou Python) ;
2. des solutions plus puissantes, mais plus complexes, fondées soit sur la plateforme Java (J2EE), avec en particulier le *framework* Struts, soit sur la plateforme .NET de Microsoft.

- www.php.net
- www.phpindex.com
- www.phpfrance.com
- www.afup.org

À RETENIR **PHP couramment proposé**

Pour information, les solutions d'hébergement de sites personnels offrent très souvent la possibilité d'utiliser le langage PHP (et parfois la base de données MySQL). De nombreux fournisseurs offrent également de l'hébergement JSP/Servlets tant mutualisé que dédié. En revanche, l'hébergement mutualisé ASP.NET est clairement très rare...

Solution à base de scripts : PHP

De nombreux langages de script sont utilisables pour développer des sites web dynamiques. Parmi les plus usités, citons Perl, Python et PHP. Ce dernier gagne progressivement des parts de marché : il a séduit de nombreux particuliers et sa simplicité de mise en œuvre convainc également de plus en plus d'entreprises.

Un site développé en PHP est très simple : il se compose de pages (dont l'extension est `.php`) contenant à la fois du code HTML et des encarts de code de programmation écrits en langage PHP. Ce dernier lit et manipule les informations provenant des clients du site via leurs navigateurs web, et réalise tout type de traitement. Des bibliothèques complémentaires permettent à PHP d'accéder aux bases de données relationnelles telles que Oracle, MySQL, etc.

Si l'apprentissage du langage est trivial, l'architecture basique que nous venons d'évoquer n'assure ni la possible maintenance du site, ni la sépa-

ration des rôles de programmeur et de graphiste car les langages HTML et PHP sont complètement entremêlés.

Les férus de PHP ont donc souvent eu recours aux techniques élémentaires de génie logiciel en externalisant un maximum de code applicatif dans des fichiers PHP séparés dont on invoque les fonctions depuis les pages de présentation frontales. PHP appliquant également les concepts orientés objet (classe, héritage, polymorphisme), les programmeurs initiés à l'objet y trouvent souvent leur bonheur.

La principale critique que l'on peut formuler à l'égard de ce langage et de sa plateforme concerne les performances : PHP est un langage interprété et s'exécute donc moins vite que les pages dynamiques du type servlets/JSP ou ASP.NET, qui sont pré-compilées. Toutefois, l'interpréteur PHP ne cesse de s'améliorer ; les performances globales de PHP sont donc aujourd'hui tout à fait acceptables pour la majorité des sites personnels et professionnels.

Solution à base de scripts et d'un serveur d'applications : Zope / Plone

En sus des langages de scripts que nous avons cités (PHP, Python, Perl), il existe des initiatives que l'on peut qualifier de frameworks globaux ou de serveurs d'application, implémentés pour ces mêmes langages et qui apportent non seulement des services techniques supplémentaires, mais aussi des guides de bonne conduite concernant l'architecture technique. Il en existe de plus en plus ; prenons simplement l'exemple de Zope, un serveur d'applications permettant de développer des applications d'entreprise en mode client léger, et qui se repose sur Python côté serveur.

Zope est un logiciel Open Source, écrit lui-même en grande partie en Python. Il offre des services techniques indispensables à une application d'entreprise, tels que la persistance des données métier (et la gestion de l'historique des données), l'authentification des usagers, l'administration à distance et à travers le web des applications installées… Bref, s'appuyer sur Zope pour développer un site de librairie en ligne raccourcit les délais, limite les risques liés à l'architecture technique, et améliore les performances de l'application (grâce à des techniques de cache et de *pooling* avancées et automatiques).

En deux mots, Zope inclut un serveur web, ce qui lui permet d'héberger et de servir des pages HTML statiques. Pour dynamiser le contenu des pages, Zope propose le langage DTML, qui consiste en un ensemble de balises installant de la logique simple dans les pages (itérations, affichage d'une variable, calcul très simple). Néanmoins, le gros de la logique métier réside ailleurs : elle est portée par des classes Python qui seront manipulées par les pages DTML, et dont les instances seront (si vous le désirez) automatiquement stockées en base de données orientée objet (elle aussi incluse dans Python).

LANGAGE PHP 4 ou PHP 5 ?

PHP est un jeune langage et la version 4 souffre encore de nombreuses lacunes. Par exemple, s'il permet en apparence de créer des objets, il n'en connaît en réalité pas vraiment la notion, et se contente de voir ces objets comme des tableaux associatifs banals. PHP 5 tente de combler les lacunes de PHP 4 en intégrant un nouvel interpréteur : la version 2 du fameux Zend Engine. Ce nouvel interpréteur est plus rapide et surtout, il rénove complètement la gestion des objets. Désormais, PHP va réellement disposer d'un type objet jusqu'au cœur du moteur.

- 🕮 *Les Cahiers du programmeur PHP 5*, S. Mariel, Eyrolles, 2004
- 🕮 *Premières applications Web 2.0 avec Ajax et PHP*, J.M. Defrance, Eyrolles, 2008

D'ailleurs, la librairie Eyrolles elle-même a fait le choix de PHP pour son site web…

- www.zope.org
- www.zopera.org
- www.python.org

L'avantage d'un langage comme Python est le compromis qu'il offre entre la simplicité d'apprentissage et de mise en œuvre et sa puissance d'expression (Python est concis, orienté objet, dynamique, supporte l'introspection, etc.). Bref, c'est un langage aussi puissant que Java ou C#, mais plus aisément accessible.

Zope complète Python en offrant toute l'infrastructure technique au développeur de sites web dynamiques.

📖 *Les Cahiers du programmeur Zope/Plone*, K. Ayeva, O. Deckmin, P-J. Grizel et M. Röder, Eyrolles, 2004

📖 *Programmation Python*, T. Zadié, Eyrolles, 2006.

Solution Java J2EE

Figure 8–4

Avec le langage Java est apparue une nouvelle technologie : les servlets. Ces petits « serveurs » ou « services » sont écrits en langage Java et utilisent une API spécifique. Ils corrigent certaines faiblesses des CGI. Les performances sont améliorées par les fonctionnalités multi-threads des serveurs J2EE, évitant la création de processus externes.

J2EE = Java 2 Enterprise Edition

Cependant, le développeur doit toujours mixer le code Java et HTML. De plus, la moindre modification oblige à recompiler la servlet et à la recharger. Les JSP, ou Java Server Pages, viennent résoudre ces problèmes de re-compilation. Ici, c'est le code Java que l'on incorpore dans la page HTML avec des techniques de *scripting*. Le serveur compile automatiquement la page en une servlet et l'exécute ensuite. Les approches à base de scripting nécessitent l'incorporation importante de code applicatif dans le HTML. Ces techniques limitent aussi la réutilisation de code. Pour ce qui est du monde Java, il a donc été proposé de faire collaborer les servlets et les JSP dans les applications. Les servlets sont utilisées par les développeurs pour gérer les aspects programmation d'une application web et les JSP sont utilisées par les infographistes pour effectuer l'affichage. On retrouve ainsi une servlet et une JSP par requête possible sur le site web. La servlet ne contient plus de HTML, et la JSP contient juste le code nécessaire à l'affichage. Ce style de programmation respecte le paradigme MVC.

📖 *Les Cahiers du programmeur J2EE*, J. Molière, Eyrolles, 2005

📖 *Les Cahiers du programmeur Java/XML*, R. Fleury, Eyrolles, 2005

CONCEPT AVANCÉ **Le paradigme MVC (Modèle–Vue–Contrôleur)**

Le paradigme MVC est un schéma de programmation qui propose de séparer une application en trois parties :

- le modèle, qui contient la logique et l'état de l'application ;
- la vue, qui représente l'interface utilisateur ;
- le contrôleur, qui gère la synchronisation entre la vue et le modèle.

Le point essentiel consiste à séparer les objets graphiques des objets métier, afin de pouvoir les faire évoluer indépendamment et les réutiliser.

On peut également gérer facilement plusieurs vues du même modèle.

Le paradigme MVC de base est représenté par le schéma 8-5.

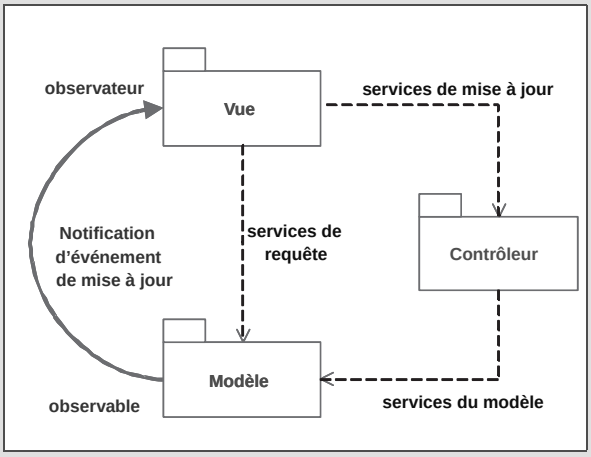


Figure 8–5

La vue utilise le contrôleur pour modifier le modèle. Les événements ne font que notifier d'un changement, pas du contenu de ce changement. Cependant, la vue a une certaine connaissance du modèle puisqu'elle utilise ses accesseurs.

Un MVC optimisé a été proposé pour réduire la communication entre les couches (figure 8-6).

Les événements notifient du changement et de leur contenu. Le contrôleur ne fournit que des mutateurs. Pour les systèmes distribués, la notification peut être effectuée par le biais du contrôleur. Ce paradigme est implémenté par Java/Swing.

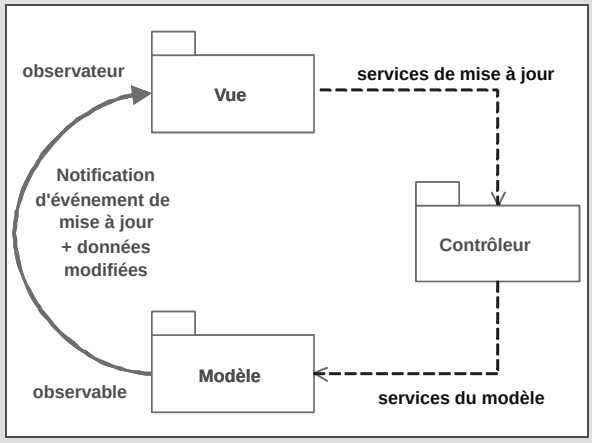


Figure 8–6

Une troisième version du paradigme MVC pour les applications web est donnée par le schéma 8-7.

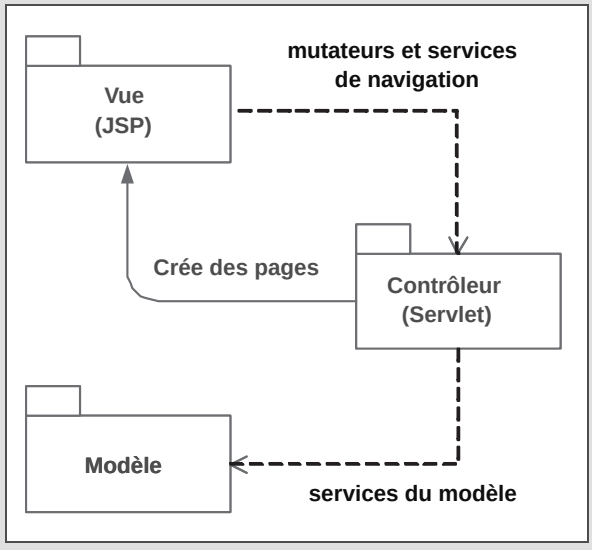


Figure 8–7

B.A.-BA JavaBean

Un JavaBean est une classe Java se conformant à quelques règles qui en permettent l'utilisation dynamique. C'est une classe concrète publique, possédant un constructeur public sans argument. Elle peut définir des propriétés similaires à des attributs possédant des accesseurs publics suivant une convention de nommage particulière :

```
<Property Type> get<PropertyName>()
void set<PropertyName>
     (<PropertyType>)
```

Au final, une telle séparation favorise le développement et la maintenance des applications :

- Le modèle étant séparé des autres composants, il est développé indépendamment. Le développeur du modèle se concentre sur le fonctionnel et le transactionnel de son application.
- Le modèle n'est pas lié à une interface, il peut donc être réutilisé (passage d'une application avec interface en Java à une application avec interface web). Dans les applications J2EE, le modèle est assuré par des EJB et/ou des JavaBeans, le contrôleur est assuré par des servlets et la vue par des JSP.

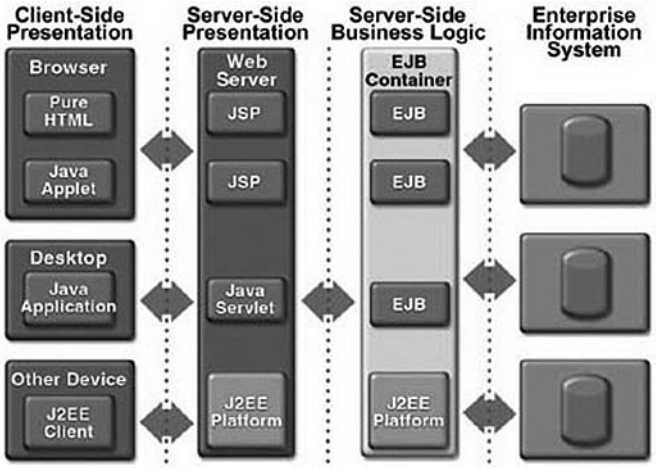


Figure 8–8
Architecture d'une application J2EE

Elle n'est cependant pas encore idéale : elle oblige à écrire une multitude de servlets, qui sont autant de points d'entrée dans l'application. Pour pallier cet inconvénient des frameworks ont été développés. Ces frameworks qui sont composés d'une seule servlet (i.e. un seul contrôleur) sont regroupés sous l'étiquette *MVC2*. Les frameworks les plus utilisés actuellement s'appellent **Struts**, **Spring** et **JSF**.

B.A.-BA Framework

Un framework est un ensemble cohérent de classes et d'interfaces collaborant pour fournir des services à la partie centrale d'un sous-système logique.
Il contient principalement des classes abstraites que l'utilisateur devra spécialiser pour ses besoins fonctionnels propres, ainsi que des interfaces auxquelles il lui faudra se conformer.

Struts est un projet Open Source développé par la communauté Jakarta d'Apache.

▸ jakarta.apache.org/struts

Il fournit un framework MVC2 comprenant les composants suivants :

- un contrôleur facilement configurable permettant d'associer des actions (objets Java) à des requêtes http ;
- des bibliothèques de *tags* spécifiques pour créer facilement une vue ;
- un *Digester*, permettant de *parser* un fichier XML et d'en récupérer seulement les informations voulues ;

- des bibliothèques pour remplir automatiquement les champs de formulaires et créer des applications proposant plusieurs langues (internationalisation).

Spring, une implémentation à la mode du pattern IoC (*Inversion of Control*) propose aussi son implémentation MVC2.

Java Server Faces (JSF) est un framework web récent de type MVC2 tentant de réconcilier le monde du développement Internet avec celui du développement RAD (Delphi, etc.). Son but est de permettre le développement d'applications web en se basant sur des composants graphiques de haut niveau et un modèle événementiel abstrayant le développeur des tracas liés à l'utilisation du protocole http. En clair, il est question d'être plus productif dans la création des pages web.

D'un point de vue stratégique et concurrentiel, JSF est la réponse de Sun à la technologie ASP.Net de Microsoft et à la notion de WebForms. De plus, Sun souhaite promouvoir JSF comme étant le framework standard de développement web, ce qui explique son inclusion dans JEE 5, là où les autres frameworks du monde Java (Struts en particulier) restent des initiatives Open Source indépendantes.

Techniquement, JSF est comparable à Struts sur de nombreux points comme la présence d'un contrôleur unique (FacesServlet), de bibliothèques de Tags (pour faciliter le développement des vues) et d'un fichier de configuration pour définir la cinématique de l'application.

JSF se démarque par le fait que les pages sont en autosoumission (PostBack), ce qui occasionne une gestion de l'état de la vue entre une réponse du serveur et la requête suivante de l'utilisateur. Le cycle de traitement d'une demande est donc beaucoup plus précis qu'avec Struts, la programmation de plus haut niveau, la présence d'événements côté serveur rend plus facile l'assimilation de cette technologie par des équipes peu familières avec le développement web et la présence de nombreuses bibliothèques de composants JSF (Tomahawk du projet Open Source MyFaces étant la plus connue) améliore considérablement le temps de développement.

La guerre des frameworks ?

Struts a une courbe d'apprentissage élevée, une documentation déficiente, et semble « embourbé » dans sa version 1.3, remplie de limitations, depuis un long moment. La version 2.0 devrait remédier à beaucoup de choses, mais alors qu'elle se fait toujours attendre, Spring dévore les parts de marché. Il existe une pléthore d'autres frameworks comme Tapestry ou, plus récemment, le très prometteur Wicket. Enfin, JSF pourrait bien mettre tout le monde d'accord …

Solution Microsoft .NET

Microsoft propose depuis quelques années un outil très simple pour le développement de sites web (architecture client léger) : les ASP. Très proches des techniques de scripting que nous avons déjà mentionnées, les ASP mêlent langage HTML et langage de programmation (VBScript, JScript…). Toutefois, cette technique présente un certain nombre de limitations (performances, maintenance) qui ont poussé Microsoft à améliorer ce framework et à le faire évoluer en ASP.NET.

▸ www.dotnetguru.org
Un excellent site sur les architectures .NET mais aussi J2EE !

On peut maintenant retirer le « ? » de Linux grâce à l'implémentation du projet Open Source Mono.

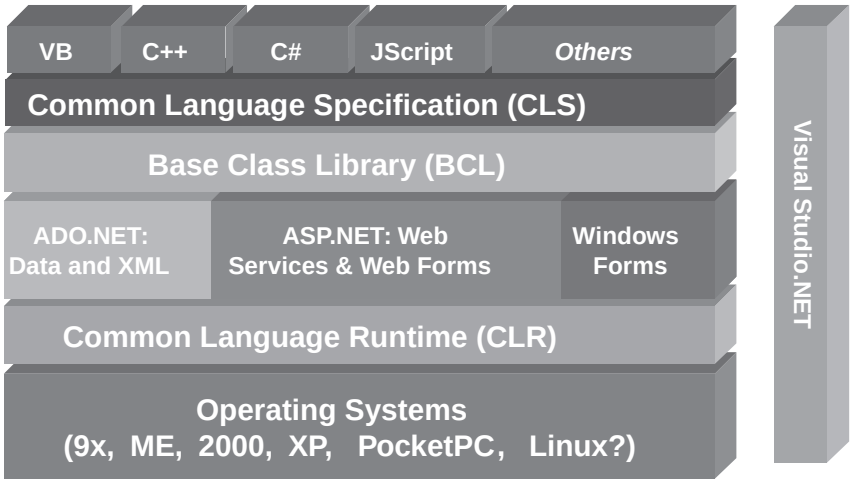
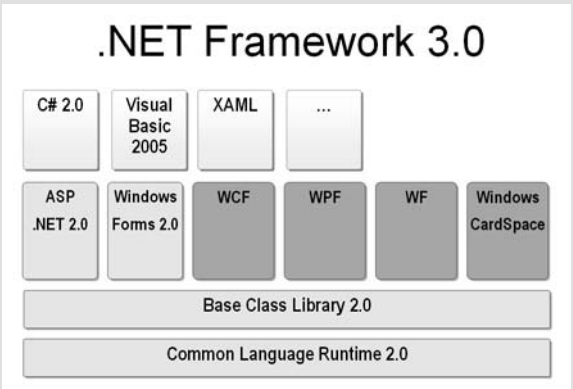


Figure 8–9
Les ASP.NET dans l'ensemble de la plate-forme .NET

À SAVOIR **Le framework .NET 3.0**

Le framework .NET 3.0 est un ajout au framework .NET 2.0. Il est livré d'office avec Windows Vista et peut être installé sur Windows XP SP2 ou Windows 2003 Server.
Les apports concernent :

- WCF (Windows Communication Foundation, anciennement Indigo) : un modèle de programmation pour des applications orientées service (SOA), en particulier des Services web.
- WPF (Windows Presentation Foundation, anciennement Avalon) : XAML.
- WF (Windows Workflow Foundation, car WWF était déjà pris !) : un modèle de programmation pour des applications sous forme de processus métier.
- WCS (Windows CardSpace) : facilite l'échange d'informations d'identification personnelle.



Voici l'application Hello Web en WPF (version XAML)
le fichier Application.XAML :

```
<Application x:Class="MonNS.App"
  xmlns="http://schemas.microsoft.com/winfx/
    ➥ 2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/
    ➥ 2006/xaml"
 StartupUri="MaPage.xaml"
  >
</Application>
```

le fichier MaPage.XAML:

```
<Page x:Class="MonNS.MaPage"
  xmlns="http://schemas.microsoft.com/winfx/
    ➥ 2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/
    ➥ 2006/xaml"
  Title="MaPage"
  >
  <Grid>
    <StackPanel >
      <TextBlock Margin="10,10,0,0" MaxWidth="200" >
        Hello, World!
      </TextBlock>
    </StackPanel>
  </Grid>
</Page>
```

Les ASP.NET offrent une manière originale de découpler le code applicatif du code de présentation : il est toujours possible d'écrire des pages HTML et de dynamiser leur contenu par l'inclusion de quelques lignes de code (cette fois-ci en langages C# ou VB.NET), mais l'essentiel du code applicatif est déporté dans une classe associée à la page ASP.NET : une classe dite *CodeBehind* (Framework 1.1) ou *CodeBeside* (Framework 2.0). Cette classe peut contenir par exemple toutes les méthodes de réaction aux événements de l'utilisateur (clic sur un hyperlien, modification ou saisie d'une zone de texte, soumission d'un formulaire…)

Ce n'est pas tout : ASP.NET propose d'aller encore plus loin avec le framework WebForms, qui aide à développer graphiquement des pages dynamiques (et réactives aux événements utilisateur), tout comme on développerait une application « client lourd » en VisualBasic, Delphi, PowerBuilder ou Java/Swing. La facilité de prise en main de ces outils est déconcertante et invite à s'appuyer sur une architecture propre favorisant maintenance et performance, ce qui accélère les temps de développement.

Enfin, dans ce nouveau framework ASP.NET, les performances sont nettement améliorées du fait que toutes les pages (ASP.NET) et les classes (CodeBehind) sont pré-compilées et non plus exécutées comme c'était le cas en ASP.

Conception détaillée du cas d'utilisation « Gérer son panier »

Pour finaliser notre démarche, nous allons maintenant définir les classes de conception détaillée de l'un des trois cas d'utilisation majeurs de l'internaute, à savoir : `Gérer son panier`. Pour cela, nous allons affiner les diagrammes d'interactions génériques réalisés au chapitre précédent, ainsi que les diagrammes de classes correspondants. Pour ce faire, nous allons envisager plusieurs solutions techniques, de la plus simple à la plus sophistiquée.

Solution technique à base de langage de scripts (PHP)

Implémentation des trois types d'analyse

Pour résumer, en PHP :

- Le dialogue est réalisé par les pages PHP (scripts embarqués dans du code HTML).
- Le contrôle est également implémenté par les pages PHP qui traitent les résultats d'un formulaire ou d'un clic sur un lien hypertexte. On

PHP 5 Concept d'interface

PHP 5 prend maintenant en charge le concept d'interface (à la manière de Java ou C#). On peut ainsi découper une application PHP selon le modèle MVC en définissant les interfaces nécessaires pour baliser ce découpage.

peut cependant externaliser une partie de ce rôle dans des classes écrites en PHP (stockées dans des fichiers séparés).
- Les entités sont des classes ou des bibliothèques de fonctions PHP, qui accèdent généralement aux bases de données.

Détaillons maintenant tout cela sur notre exemple.

Pages PHP

Chaque page du site est implémentée par une page PHP qui contiendra par exemple :
- un bandeau horizontal, qui permet l'accès à l'ensemble des principaux menus ;
- un bandeau gauche, pour un accès rapide à la recherche avancée, ainsi qu'un accès thématique ;
- le corps spécifique de la page.

Nous pouvons représenter tout cela précisément par un diagramme de classes UML (figure 8-10)

ÉTUDE DE CAS Représentation des relations d'agrégation et de généralisation

Nous avons représenté des relations d'agrégation et de généralisation UML sur le diagramme 8-10 pour rester fidèles à la modélisation objet. Cependant, en PHP, elles vont toutes se traduire par des inclusions, d'où la notation `include` sur chaque relation.

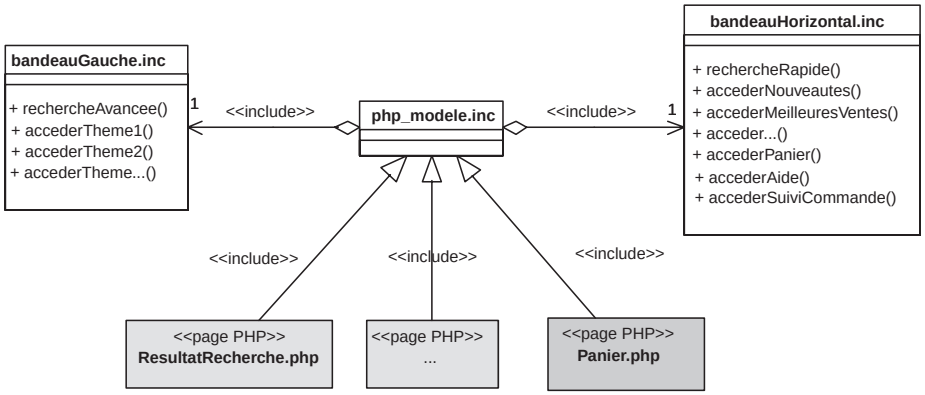


Figure 8–10
Diagramme de classes des pages PHP

Gestion du panier

Pour mettre en œuvre la mémorisation des ouvrages choisis par l'internaute dans son panier virtuel, PHP 4 propose une bibliothèque appelée la « Phplib ». Les fonctionnalités offertes par la Phplib ne se réduisent pas uniquement à une classe `Session`, qui conserve les mêmes variables sur plusieurs pages PHP. Cette bibliothèque permet également de gérer des sessions avec authentification des utilisateurs et protection des pages, selon un système de droits.

ÉTUDE DE CAS Nouveau stéréotype

Nous utilisons un nouveau stéréotype «`Classe Phplib`» pour bien distinguer la classe `Session` des pages PHP ou de futures classes créées par le développeur.

Afin de savoir quelles sont les données propres à un utilisateur, un identifiant de session unique lui est attribué, dès sa connexion à la page

d'accueil. Cet identifiant sert de clé à l'objet session. Il est conservé et transmis à toutes les autres pages de l'application (figure 8-11).

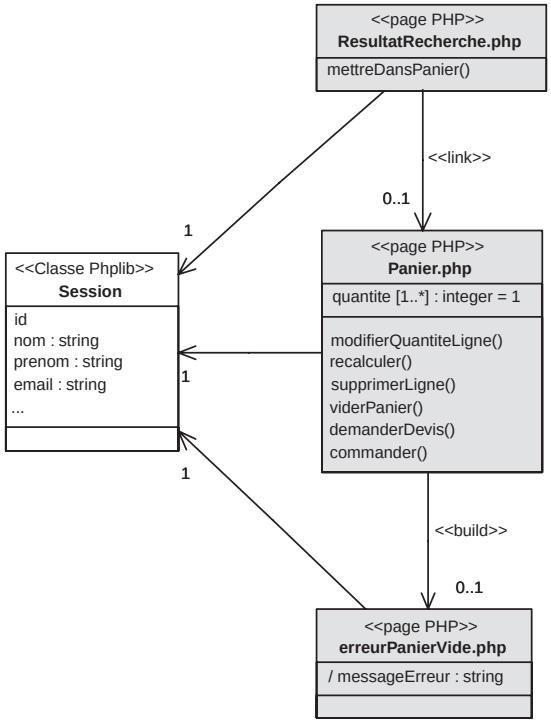


Figure 8–11
Intégration de la classe prédéfinie Session

Actuellement, on utilise plutôt la solution du tableau superglobal `_SESSION` décrite ci-après. L'instruction `session_start` active le suivi des visiteurs. Cependant, hormis l'identifiant, le profil d'un visiteur ne comporte par défaut aucune donnée. Pour résoudre ce manque, PHP permet d'associer certaines données à la session.

Celles-ci seront alors préservées pendant toute la durée de vie de la session. Pour ce faire, PHP met à notre disposition le tableau superglobal `_SESSION`. Accessible en permanence, ce dernier est automatiquement initialisé avec les données de session. De même, chaque élément ajouté à ce tableau sera automatiquement ajouté aux données de la session.

Classes PHP

PHP est un langage orienté objet, surtout avec la version 5 (comme nous l'avons évoqué précédemment). Nous allons donc continuer à représenter sous forme de classe notre panier virtuel. De plus, la gestion de contexte de la Phplib va nous permettre d'en mémoriser le contenu page après page, sous forme d'un tableau que nous manipulerons grâce à des

PHP Notation des variables

Les variables PHP sont toutes dotées du préfixe $. Pour la syntaxe précise et complète du langage PHP, nous vous renvoyons aux ouvrages récents qui ne manquent pas sur le sujet…

- *PHP 5 Avancé*, É. Daspet et C. Pierre de Geyer, Eyrolles, 2008
- *Best practices PHP 5*, G. Ponçon, Eyrolles, 2005
- *Les Cahiers du programmeur PHP 5*, S. Mariel, Eyrolles, 2004

fonctions standards PHP. Voici les principales fonctionnalités offertes par cette classe `Panier` :

- L'objectif est de gérer une liste d'objets de même nature : des livres. Seul, l'identifiant du livre est enregistré dans la liste, avec sa quantité. Par exemple, le livre de code `$code` est stocké dans le panier, avec une quantité `$nombre`.
- Pour chaque internaute, l'objet panier instancié est enregistré en session grâce à la gestion de contexte.

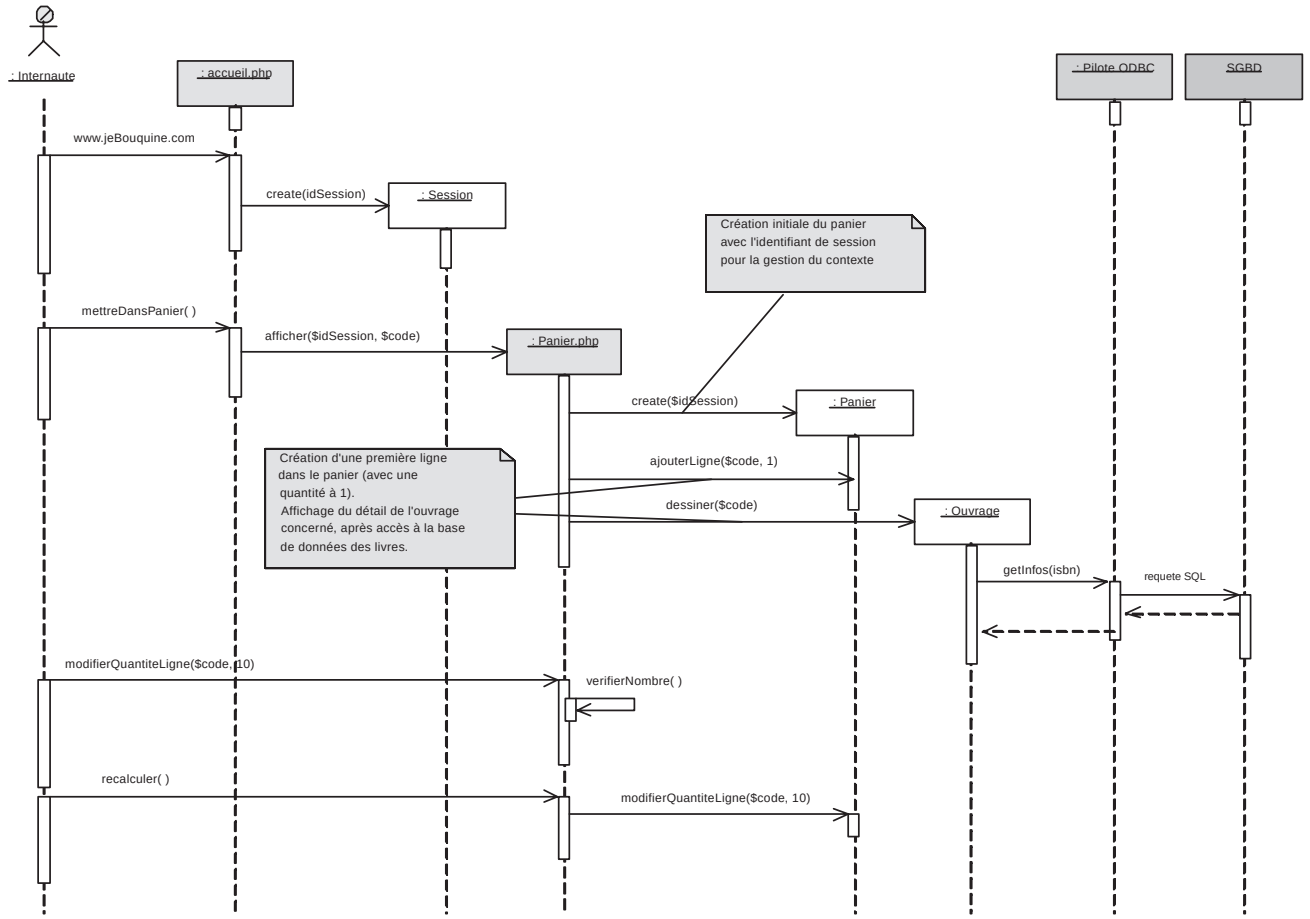


Figure 8–12 Exemple de diagramme de séquence de conception détaillée pour la gestion du panier

Imaginons un scénario nominal simple où l'internaute met directement de côté un livre présenté sur la page d'accueil du site. Cette page `accueil.php` va tout d'abord créer un objet session, comme nous l'avons expliqué précédemment. Elle passe ensuite la main à la page `panier.php` dès que l'internaute clique sur le bouton *Mettre dans le panier*. C'est `panier.php` qui va initialiser le panier de l'internaute en lui passant

l'identifiant de la session, puis demander au panier d'ajouter une ligne et récupérer les informations nécessaires à l'affichage des caractéristiques du livre. Cette récupération est déléguée au pilote ODBC, moyen offert par PHP pour accéder aux bases de données. Le diagramme de séquence correspondant est montré sur la figure 8-12.

En extrapolant, le diagramme de classes de conception détaillée mettant en valeur la page `panier.php` est celui de la figure 8-13.

ÉTUDE DE CAS Disparition d'une classe contrôle

Remarquez que la classe contrôle du chapitre 7 a disparu. En effet, la page `panier.php` joue à la fois le rôle de dialogue et de contrôle ; elle accède directement à l'entité `Panier`.

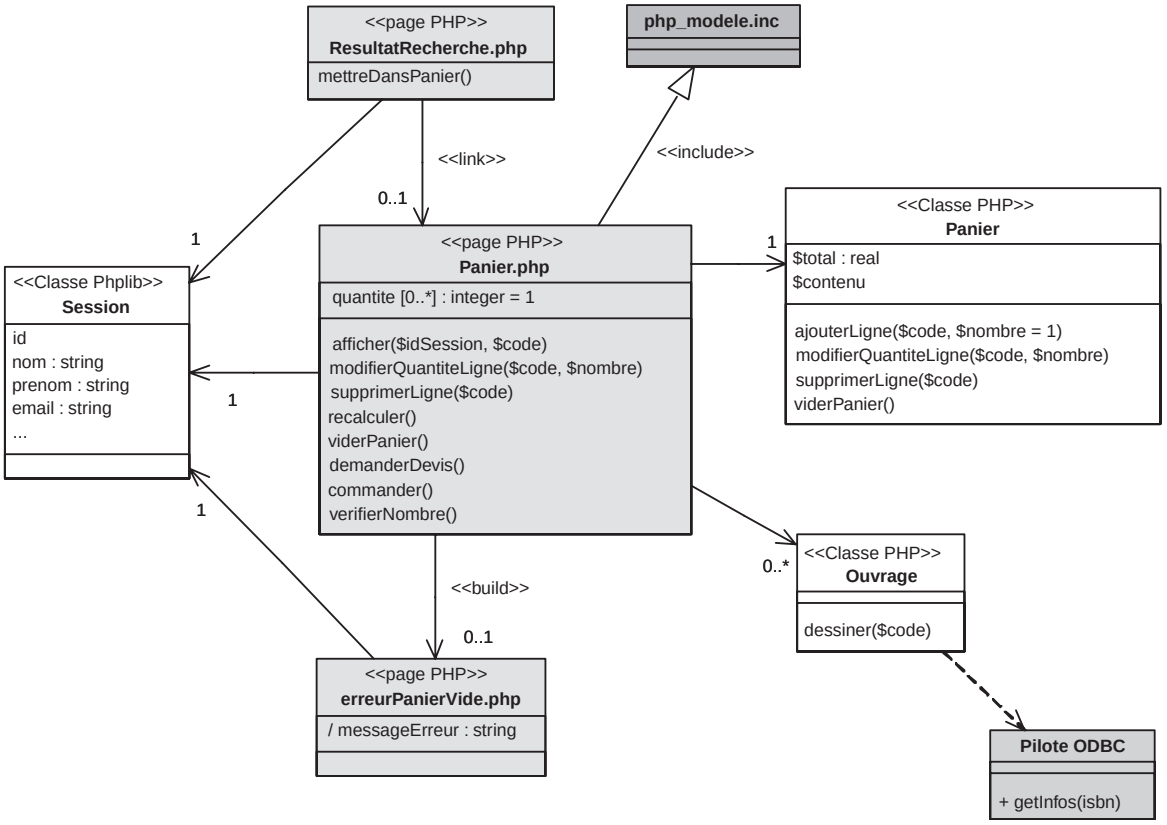


Figure 8–13 Diagramme de classes de conception détaillée PHP de la gestion du panier

ÉTUDE DE CAS Encore des stéréotypes

Nous utilisons encore d'autres stéréotypes pour ajouter le plus de précision possible :

- «`link`» indique qu'une page peut amener à une autre par navigation ;
- «`build`» indique qu'une page en construit une autre de toutes pièces ;
- «`Classe PHP`» est utilisé pour les classes créées par le développeur, par opposition à celles qui sont fournies par la Phplib.

Notez également que nous utilisons les types du langage PHP (`string`, `integer`, `real`, etc.) et non plus des types génériques comme au chapitre 7.

Exemple de code

Un extrait du code PHP de la classe `Panier` est donné ci-après pour illustration :

```
Class Panier {

// Le contenu est un tableau d'identifiants de livres avec leur quantité
var $contenu = array() ;

function ajouterLigne($code, $nombre = 1) {
   $nb = $this->nombreDe($code) ;
   if ($nb > 0) {
      $this->contenu[$code] += $nombre ;}
   else {
      $this->contenu[$code] = $nombre ;}
}

function modifierQuantiteLigne($code, $nombre) {
   if ($nombre < 0)
      return false ;
   $nb = $this->nombreDe($code) ;
   if ($nb > 0) {
      if ($nombre > 0) {
         $this->contenu[$code] = $nombre ;}
      elseif ($nombre == 0) {
         // si la nouvelle quantité est nulle, on désactive la variable
         unset($this->contenu[$code]) ;}
         return true ;}
   else {
      return false ;}
}

function supprimerLigne($code) {
   $nb = $this->nombreDe($code) ;
   if ($nb > 0) {
      unset($this->contenu[$code]) ;
   }
}

function viderPanier() {
   $this->contenu[$code] = array();
}

function nombreDe($code) {
   // renvoie le nombre de livres avec l'identifiant $code
   $resultat = $this->contenu[$code];
   if ($resultat == NULL){
      $resultat = 0;
   }
   return $resultat;
}
```

Solution technique J2EE

Architecture logique avec Struts

Le framework Struts réside dans la couche présentation et contrôle d'une application web. On peut même dire que la puissance de Struts réside principalement dans la couche contrôle : pour la présentation, c'est au développeur de choisir une bibliothèque (JSF, Struts-Layout, JSP classiques, Tiles...). La figure 8-14 illustre les principales couches et les packages les plus intéressants grâce à un diagramme de packages UML.

STRUTS Classe Action

Les frameworks de présentation web incluent habituellement les responsabilités de contrôle de l'application, ce qui est aussi vrai dans le cas de Struts. Cela demande aux développeurs de créer des sous-classes de la classe *`Action`* de Struts, qui seront responsables des décisions de contrôle.

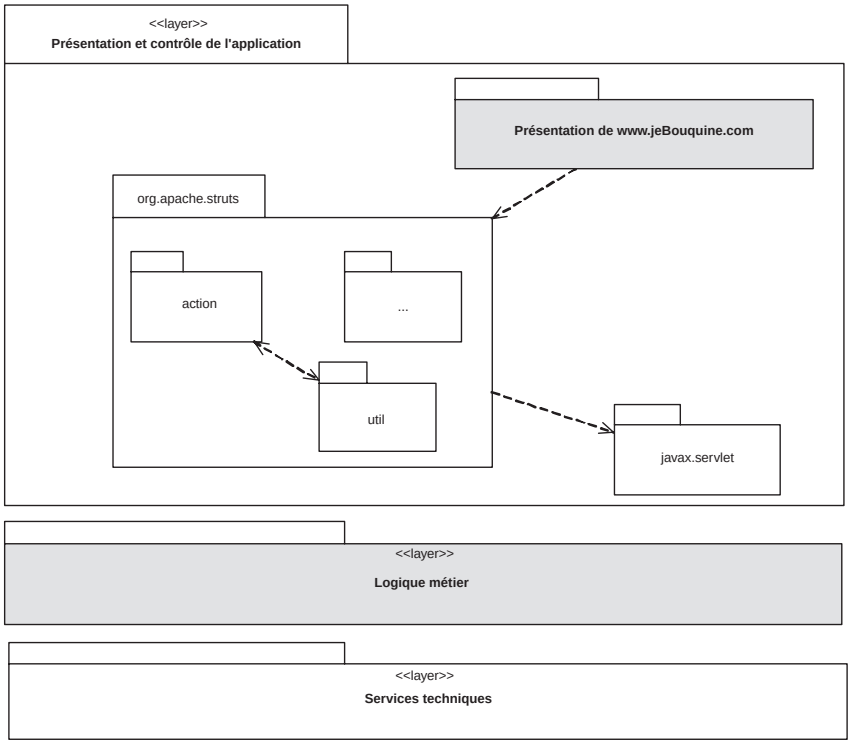


Figure 8–14
Packages et couches notables reliés à Struts

Si nous détaillons les classes fondamentales du package `action` et celles que nous devons écrire, nous obtenons le schéma 8-15. La classe `ActionServlet` joue le rôle de contrôle, alors que les `JSP` et `JSPServlets` sont des vues. Les objets métier sont de purs modèles. Les classes `Action` et `ActionForm`, ainsi que leurs sous-classes mettent en relation les vues, le contrôle et les modèles. En effet, la classe `MonAction2`, par exemple, agit sur `MonObjetMetier1`, mais prend en paramètre une référence sur une `ActionForm` qui est elle-même en liaison avec l'interface utilisateur.

STRUTS Fichier struts-config.xml

Le fichier `struts-config.xml` décrit quelles sont les actions possibles et à quelles classes *`Action`* il faut les associer.

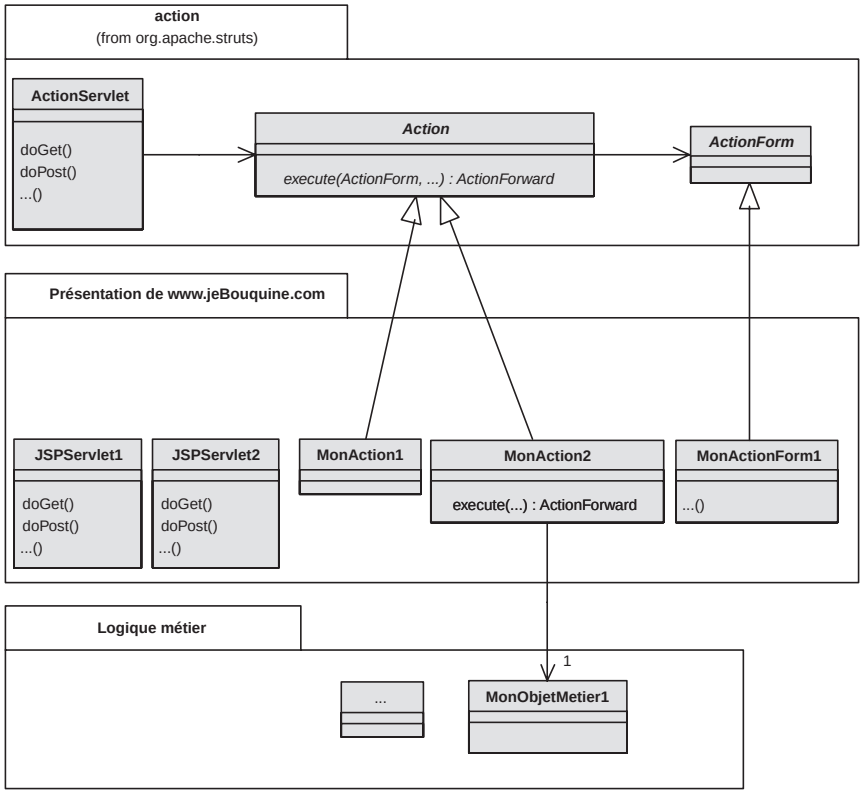


Figure 8–15
Paradigme MVC dans Struts

Pour résumer, dans Struts :

- Le dialogue est un ensemble de composants :
  - JSP (qui génère les pages et les formulaires) ;
  - + FormBean qui se charge de « vérifier la syntaxe de la requête (voir chapitre 7) » ;
  - + page HTML puisque le résultat téléchargé côté client est bien écrit en HTML.
- Le contrôle est implémenté par le contrôleur servlet et par les actions (commandes) qui exécutent des traitements directement ou se connectent à un annuaire/base de données, ou encore invoquent des méthodes à distance, etc.
- Les entités : nous pourrions utiliser des EJB (sessions et entités, car elles ne sont pas toutes persistantes). Toutefois, pour une application web de complexité moyenne, ces entités peuvent être des JavaBeans, nettement plus simples à mettre en œuvre.

Diagrammes de séquence

Nous avons représenté dans le schéma 8-16 le scénario de création d'une nouvelle ligne du panier suite à l'action de l'internaute `Mettre dans le panier`. Nous avons supposé que le panier avait été créé au préalable, comme dans la figure 7-17 du chapitre précédent.

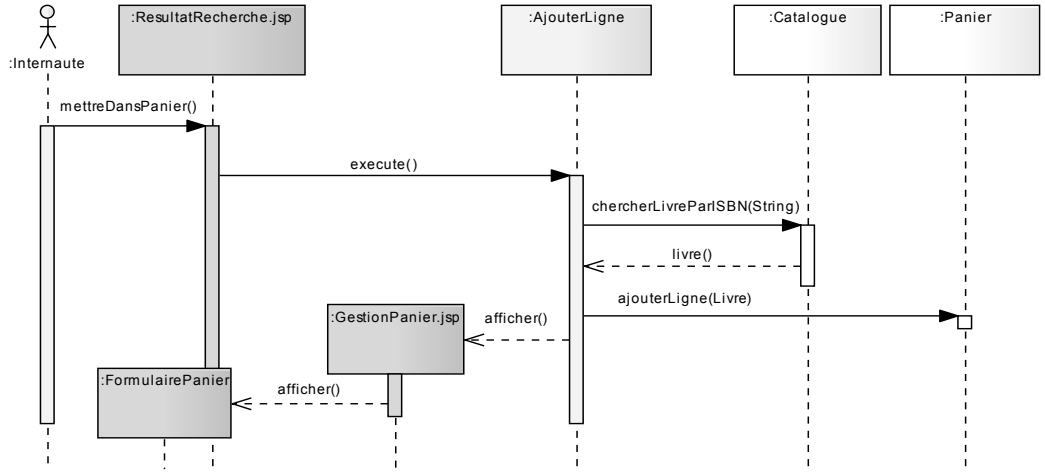


Figure 8–16
Diagramme de séquence de la création d'une ligne du panier avec Struts

En comparaison avec le diagramme 7-17 du chapitre précédent, la correspondance est finalement assez simple :

- Les dialogues sont devenus des `.jsp` (munies éventuellement d'un formulaire).
- Le contrôle est réparti entre la servlet `ActionServlet` et les actions associées (mais nous n'avons pas fait figurer l'`ActionServlet` sur le diagramme de séquence car il ne s'agit pas vraiment d'interaction par message).
- Les entités restent quasiment telles quelles, sous forme de JavaBeans.

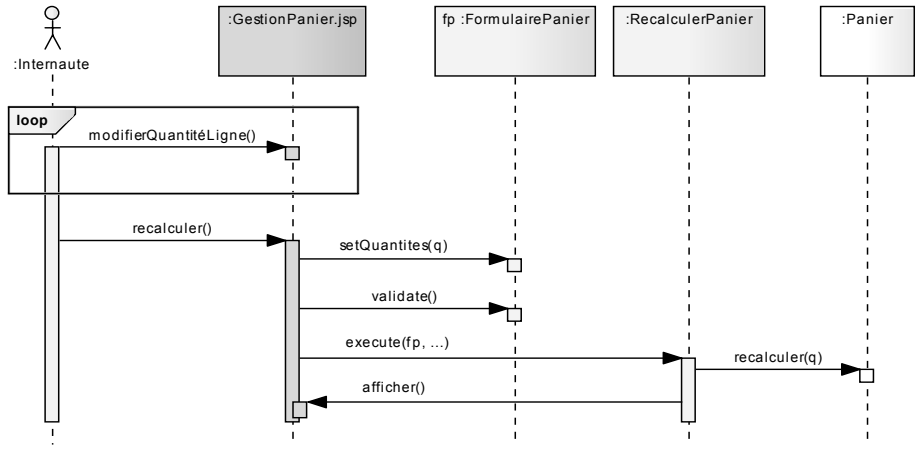


Figure 8–17
Diagramme de séquence de la gestion du panier avec Struts

ÉTUDE DE CAS Interactions avec LignePanier

Nous n'avons pas détaillé de nouveau comment le *Panier* interagit ensuite avec ses *LignePanier*. Les diagrammes élaborés au chapitre 7 sont tout à fait suffisants dans le cas de classes Java.

Le recalcul du panier suite à des modifications de quantité sur une ou plusieurs ligne(s) va faire intervenir le formulaire pour validation. Regardons comment sur le diagramme 8-17.

Diagrammes de classes de conception détaillée

En extrapolant le travail précédent sur les diagrammes de séquence, le diagramme de classes de conception détaillée autour de la gestion du panier est le suivant (figure 8-18).

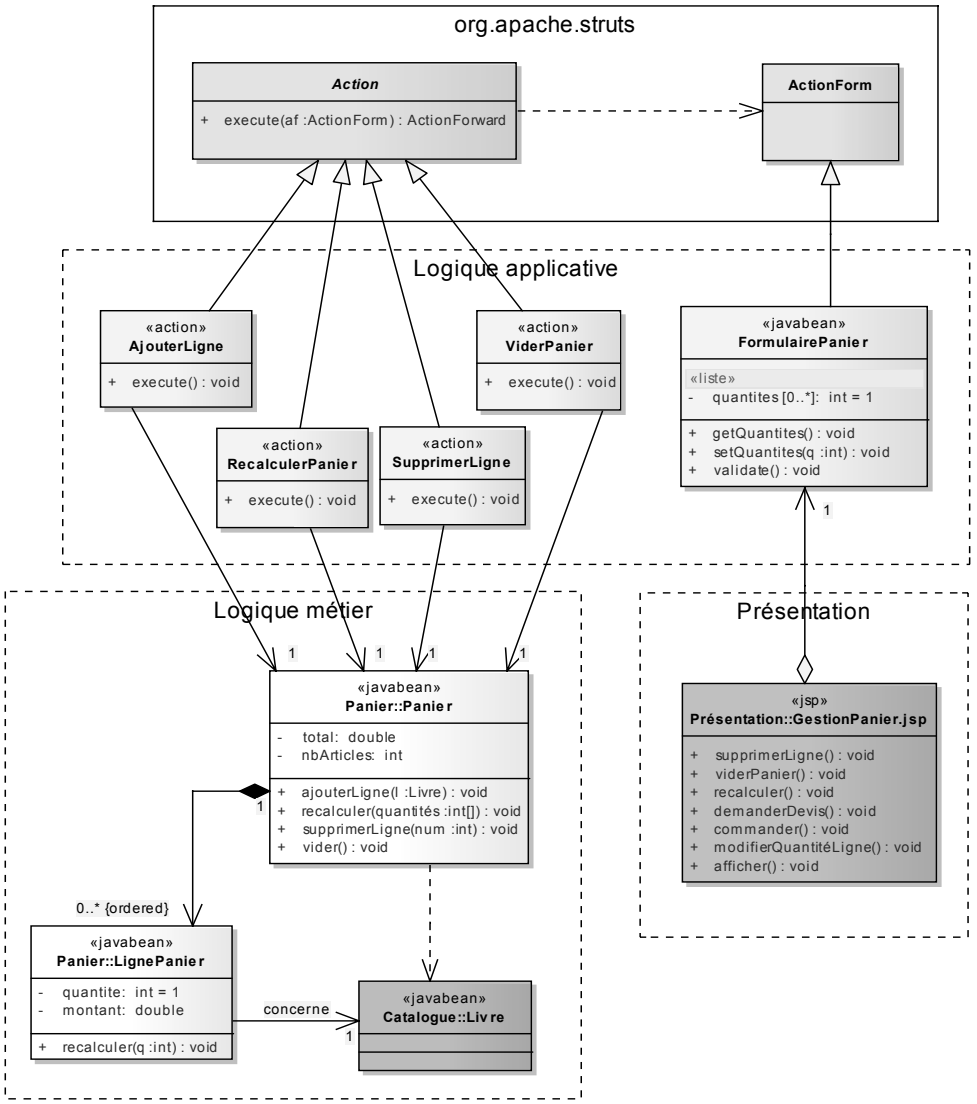


Figure 8–18 Diagramme de classes de conception détaillée Struts de la gestion du panier

Exemple de code

La classe `Panier`, codée en tant que JavaBean, est présentée sur le listing suivant.

```
package LogiqueMetier.Gestion;

import LogiqueMetier.Catalogue.Livre;
import java.util.*;

public class Panier implements java.io.Serializable {
   private double total;
   private List lesLignesPanier = new ArrayList();

   public Panier() {}

   public double getTotal()
   {
      return total;
   }
   public void recalculer(List quantites)
   {
     total = 0;
     Iterator lignesIt = lesLignesPanier.iterator();
     Iterator quantiteIt = lesLignesPanier.iterator();
     while(lignesIt.hasNext()){
        LignePanier l = (LignePanier)lignesIt.next();
        int qte = ((Integer) quantiteIt.next().intValue();
        l.recalculer(qte);
        total += l.getTotal();
     }
   }
   public void ajouterLigne(Livre l)
   {
   // Ne pas ajouter de ligne s'il en existe déjà une pour ce livre
     Iterator i = lesLignesPanier.iterator();
     while (i.hasNext()){
        LignePanier ligne = (LignePanier)i.next();
        if (ligne.getLivreConcerne().equals(l)){
           return;
        }
     }
     lesLignesPanier.add(new LignePanier(l));
   }
   public void supprimerLigne(Livre l)
   {
   Iterator i = lesLignesPanier.iterator();
      while (i.hasNext()){
      LignePanier ligne = (LignePanier) i.next();
         if (ligne.getLivreConcerne().equals(l)){
            i.remove();
            break;
         }
      }
   }
```

```
    public void viderPanier()
    {
      lesLignesPanier.clear();
    }
}
```

ÉTUDE DE CAS Quelques remarques sur le code Java précédent

- La relation de dépendance avec la classe `Livre` du package `Catalogue` se traduit par une directive `import`.
- La contrainte `{ordered}` sur la composition avec `LignePanier` se traduit de façon naturelle par l'utilisation d'une `ArrayList`.
- La classe étant un JavaBean, son attribut possède des accesseurs avec les conventions de nommage citées précédemment. Notez que l'attribut `total` est en lecture seule ; il n'y a donc pas de méthode `setTotal()`.

ÉTUDE DE CAS Recalcul des sommes

Dans notre exemple, la demande de recalcul est superflue car chaque modification génère un aller-retour client-serveur et par conséquent un réaffichage cohérent du panier.

Voici également un exemple de JSP qui gère le panier (il faudrait bien sûr peaufiner le graphisme) :

- affichage de la liste des lignes ;
- pour chaque ligne, possibilité de modifier la quantité ou de supprimer la ligne ;
- possibilité à la fin de vider le panier, de demander un devis ou passer une commande.

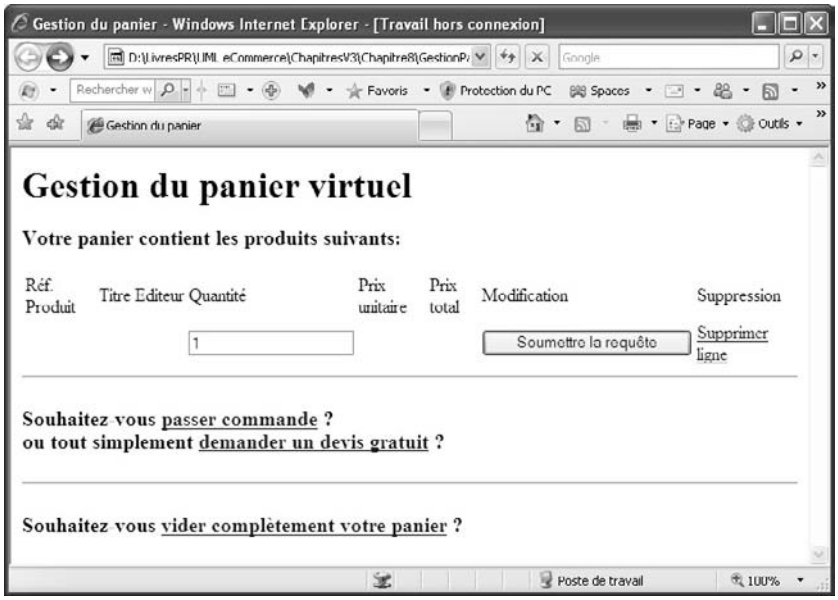


Figure 8–19
Exemple de page JSP simple de gestion du panier

Le code correspondant est donné par le listing ci-après.

ÉTUDE DE CAS Pas de TagLibs

La JSP présentée n'utilise pas les TagLibs de Struts pour une meilleure lisibilité du code.

```
<%@page contentType="text/html" import="java.util.*"%>
<jsp:useBean id="panier" scope="session"/>

<HTML>
    <HEAD><TITLE>Gestion du panier</TITLE></HEAD>
    <BODY>
        <H1>Gestion du panier virtuel</H1>
        <H3>Votre panier contient les produits suivants:</H3>
        <TABLE>
            <TR>
                <TD>Réf. Produit</TD>
                <TD>Titre</TD>
                <TD>Éditeur</TD>
                <TD>Quantité</TD>
                <TD>Prix unitaire</TD>
                <TD>Prix total</TD>
                <TD>Modification</TD>
                <TD>Suppression</TD>
            </TR>
        <%
            // En pratique, il faudrait gérer la pagination
            Iterator i = panier.iterator();
            while(i.hasNext()){
                LignePanier lp = (LignePanier) i.next();
                Livre l = lp.getLivre();
        %>
            <TR>
                <%-- Le client doit pouvoir modifier la
                  -- quantité aisément, dans le panier --%>
                <FORM action="/controleur?cmd=modifierQuantité&produit=<%=
                        ➥ l.getCode()%>" method="POST">
                    <TD><%= l.getISBN() %></TD>
                    <TD><%= l.getTitre() %></TD>
                    <TD><%= l.getEditeur() %></TD>
                    <TD>
                        <INPUT type="text" value="<%= lp.getQuantité() %>">
                    </TD>
                    <TD><%= lp.getPrixUnitaire() %></TD>
                    <TD><%= lp.getPrixTotal() %></TD>
                    <TD><INPUT type="submit" name="Modifier Quantité"></TD>

                    <%-- Le client doit pouvoir supprimer la ligne courante --%>
                    <TD><A href="/controleur?cmd=supprimerLigne&produit=<%=
                        ➥ l.getCode()%>">Supprimer ligne</A></TD>
                </FORM>
            </TR>
        <%
            }
        %>
        </TABLE>
        <HR>
        <%-- Le client doit pouvoir passer à la phase d'achat ou au devis--%>
        <H3>
            Souhaitez-vous
```

```
            <A href="/controleur?cmd=commander">passer commande</A> ?
            <BR>
            ou tout simplement
            <A href="/controleur?cmd=demandeDevis">demander un devis gratuit</A> ?
        </H3>
        <HR>
        <%-- Le client doit pouvoir vider son panier d'un coup --%>
        <H3>
            Souhaitez-vous
            <A href="/controleur?cmd=viderPanier">vider complètement votre panier</A> ?
        </H3>
    </BODY>
</HTML>
```

Solution technique .NET

Implémentation des trois types d'analyse

ASP.NET De meilleures performances

Différence majeure par rapport à la version précédente des ASP : les ASP.NET sont précompilées, donc les performances sont bien meilleures.

Pour résumer, en .NET :

- Le dialogue est réalisé par les pages ASP.NET (comme en PHP ou avec les JSP) : l'idée est également d'insérer des instructions de script dans des pages HTML (le langage peut être C# ou VB.NET). Les pages ASP.NET peuvent contenir des WebForms, espèces de *TagLibs* qui génèrent des vues en HTML et qui permettent également de placer des contraintes (*composant validators*) sur les champs de saisie utilisateur. Ces contraintes sont vérifiées côté serveur par défaut, mais un module JavaScript permet de pré-valider le tout côté client à condition d'utiliser InternetExplorer.
- Le contrôle est implémenté soit par des classes associées aux ASP.NET (classes CodeBehind), soit par des classes supplémentaires déployées dans l'application web (dans le « moteur d'ASP » : IIS).
- Les entités sont représentées par des classes de l'application web, ou par des *Serviced Components* (que l'on installe dans le serveur d'applications MTS).

ASP

Comme indiqué précédemment, le dialogue `GestionPanier` va être implémenté par une page ASP.NET. Celle-ci va contenir du code HTML, des WebForms (pour les lignes du panier), ainsi qu'une référence à une classe C# que l'on qualifie de « CodeBehind ». Cette classe renferme un ensemble de méthodes que la page ASP.NET va invoquer pour simplifier sa tâche (un traitement un peu complexe gagne à être implémenté dans le CodeBehind et appelé par une instruction de script dans la page ASP.NET).

Cette structure de la page ASP.NET est représentée sur la figure 8-20.

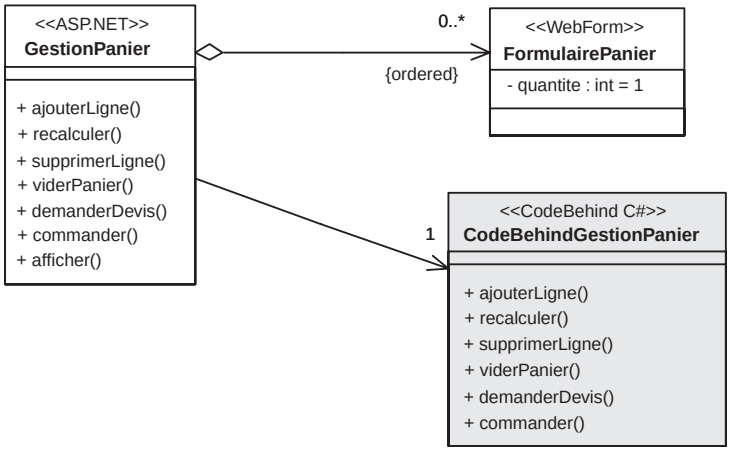


Figure 8–20
Diagramme de classes de la page ASP GestionPanier

Diagrammes de séquence

Si nous reprenons les mêmes hypothèses de scénario qu'à la figure 8-16, nous obtenons le diagramme 8-21.

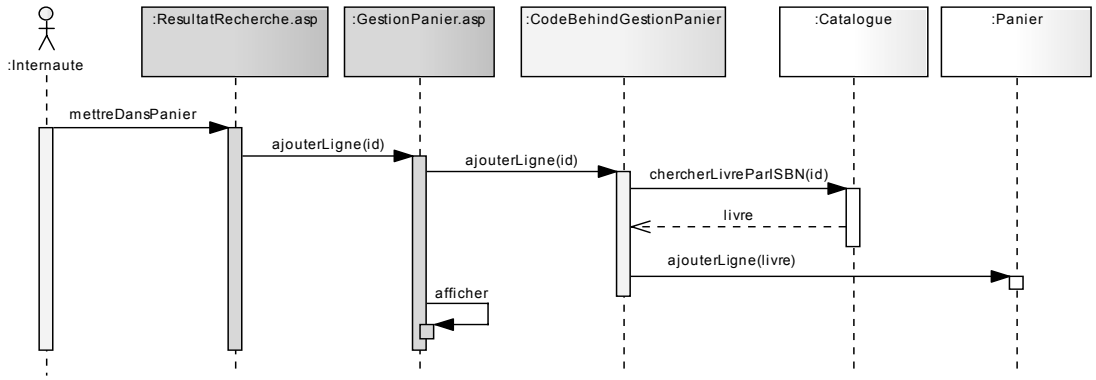


Figure 8–21 Diagramme de séquence de la création d'une ligne du panier avec .NET

> LANGAGES **Comparaison avec la solution Struts**
>
> Notez la similitude avec la conception Java réalisée en utilisant *Struts* (figure 8-16). Cependant l'architecture est à la fois plus simple (pas d'équivalent de la servlet contrôleur), et moins puissante puisque les pages sont couplées entre elles, contrairement à ce qui se passait avec *Struts*.

De même, si nous reprenons les hypothèses du scénario de la figure 8-17, nous obtenons le diagramme 8-22.

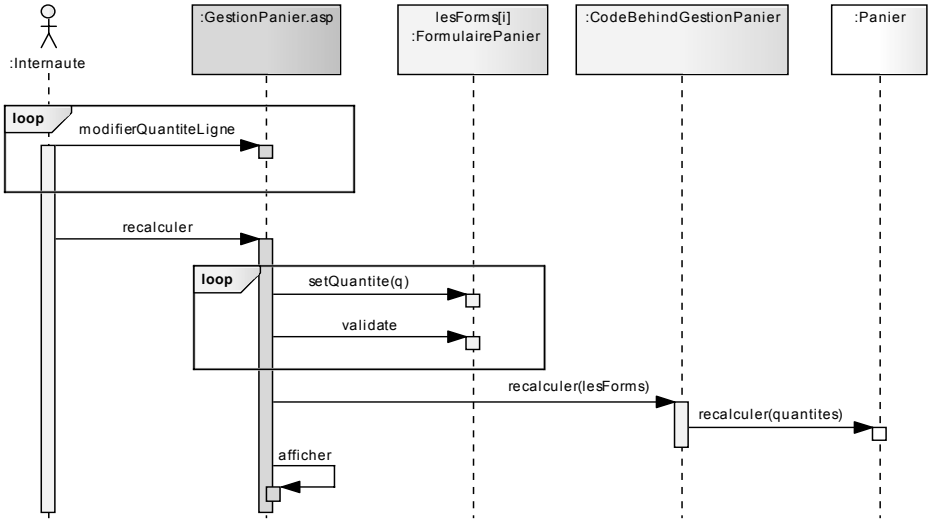


Figure 8–22
Diagramme de séquence
de la gestion du panier avec .NET

Diagrammes de classes de conception détaillée

En extrapolant le travail précédent sur les diagrammes d'interactions, le diagramme de classes de conception détaillée autour de la gestion du panier est celui de la figure 8-23.

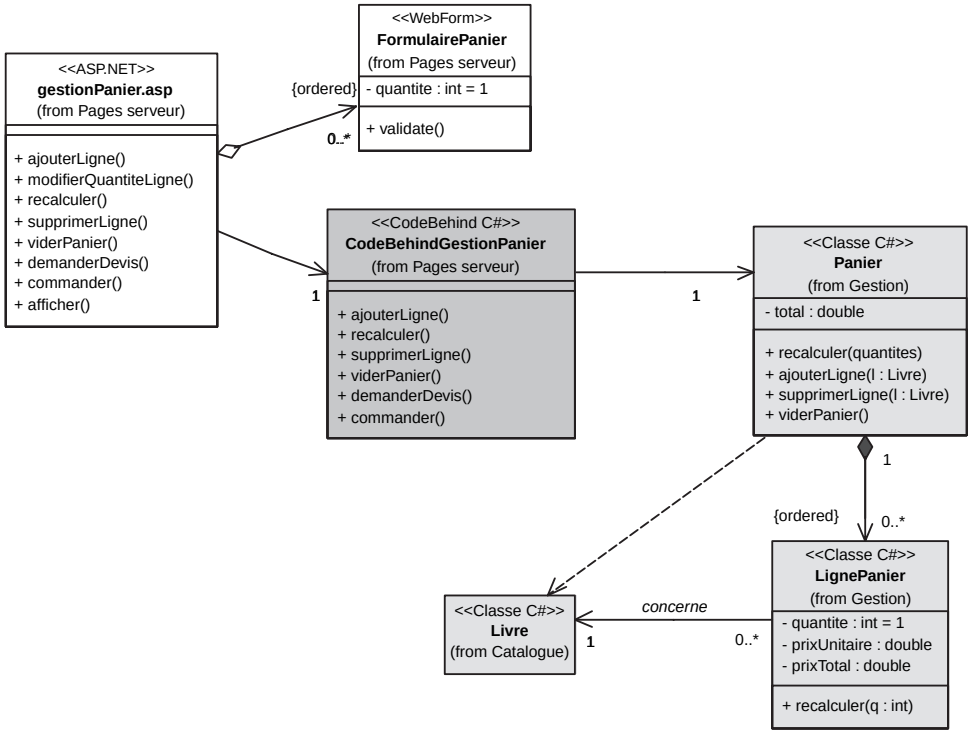


Figure 8–23
Diagramme de classes
de conception détaillée .NET
de la gestion du panier

Exemple de code

Le code C# de la classe `Panier` est donné sur le listing suivant :

```
namespace LogiqueMetier.Gestion;

using LogiqueMetier.Catalogue.Livre;
using System.Collections;

public class Panier
{
   private double tot ;
   public double total {
     get {
        return tot;
     }
   }

   private IList lesLignesPanier = new ArrayList();

   public Panier()
   {
   }

   public void recalculer(IList quantites)
   {
      total = 0;
      int i = 0;
      foreach (LignePanier ligne in lesLignesPanier){
         ligne.recalculer(quantites[i]);
         total += ligne.prixTotal;
      }
   }

   public void ajouterLigne(Livre l)
   {
      // Ne pas ajouter de ligne s'il en existe déjà une pour ce livre
      foreach (LignePanier ligne in lesLignesPanier){
         if (ligne.livreConcerne.Equals(l)){ return; }
      }
      lesLignesPanier.Add(new LignePanier(l));
   }

   public void supprimerLigne(Livre l)
   {
      foreach (LignePanier ligne in lesLignesPanier)
      {
         if (ligne.livreConcerne.Equals(l)){
            lesLignesPanier.Remove(ligne);
            break;
         }
      }
   }
```

```
 public void viderPanier()
   {
      lesLignesPanier.Clear();
   }

}
```

ÉTUDE DE CAS Quelques remarques sur le code C# présenté

Globalement, les remarques sur le code Java correspondant s'appliquent ici aussi !

Notez cependant l'utilisation du concept C# de `property` pour l'attribut `total`. Les propriétés sont un moyen d'accéder aux propriétés membres d'une classe en respectant les règles d'encapsulation. Dans le monde Java, les accesseurs (`get` et `set`) sont utilisés à cet effet. L'avantage des propriétés en C# est de donner à l'utilisateur un accès aux attributs d'un objet de la même manière que s'il effectuait directement l'opération `object.attribut`, alors qu'en réalité il appelle une méthode de manière totalement transparente (par exemple : `total+=ligne.prixTotal`).

Le code de la page ASP.NET qui gère le panier à l'identique de la page JSP précédente est donné par le listing ci-après :

```
<%@ Page language="c#" Codebehind="GestionPanier.aspx.cs"
AutoEventWireup="false" Inherits="Addition.GestionPanier" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>GestionPanier</title>
    </HEAD>
    <body>
        <form id="GestionPanier" method="post" runat="server">
            <H1>Gestion du panier virtuel</H1>
            <H3>Votre panier contient les produits suivants:</H3>
            <P>
                <asp:Repeater id="Repeater1" runat="server"
                     DataSource='<%# Session["panier"] %>'>
                    <HeaderTemplate>
                        <table border="1">
                            <TR>
                                <TD>Réf. Produit</TD>
                                <TD>Titre</TD>
                                <TD>Éditeur</TD>
                                <TD>Quantité</TD>
                                <TD>Prix unitaire</TD>
                                <TD>Prix total</TD>
                                <TD>Modification</TD>
                                <TD>Suppression</TD>
                            </TR>
                    </HeaderTemplate>
```

```
                    <FooterTemplate>
                        </table>
                    </FooterTemplate>
                    <ItemTemplate>
                        <TR>
                            <%-- Le client doit pouvoir modifier la
                            quantité aisément, dans le panier --%>
                            <FORM action="/controleur?cmd=modifierQuantité&produit=<%#
                                  ➥ DataBinder.Eval(Container, "DataItem.Livre.Code")%>"
                                  method="POST">
                                <TD><%# DataBinder.Eval(Container,
                                        "DataItem.Livre.ISBN")%></TD>
                                <TD><%# DataBinder.Eval(Container,
                                        "DataItem.Livre.Titre")%></TD>
                                <TD><%# DataBinder.Eval(Container,
                                        "DataItem.Livre.Editeur")%></TD>
                                <TD>
                                    <INPUT type="text" value="<%#
                                           DataBinder.Eval(Container, "DataItem.Quantite")%>">
                                </TD>
                                <TD><%# DataBinder.Eval(Container,
                                        "DataItem.PrixUnitaire")%></TD>
                                <TD><%# DataBinder.Eval(Container,
                                        "DataItem.PrixTotal")%></TD>
                                <TD><INPUT type="submit" name="Modifier Quantité"></TD>
                                <%-- Le client doit pouvoir supprimer la ligne courante --%>
                                <TD><A href="/controleur?cmd=supprimerLigne&produit=<%#
                                       DataBinder.Eval(Container, "DataItem.Livre.Code")%>">
                                       Supprimer ligne</A></TD>
                            </FORM>
                        </TR>
                    </ItemTemplate>
                </asp:Repeater></P>
        </form>
        <HR>
        <%-- Le client doit pouvoir passer à la phase d'achat ou au devis--%>
        <H3>
            Souhaitez-vous <A href="/controleur?cmd=commander">passer commande</A> ?
            <BR>
            ou tout simplement
            <A href="/controleur?cmd=demandeDevis">demander un devis gratuit</A>?
        </H3>
        <HR>
        <%-- Le client doit pouvoir vider son panier d'un coup --%>
        <H3>
            Souhaitez-vous
            <A href="/controleur?cmd=viderPanier">vider complètement votre panier</A> ?
        </H3>
    </body>
</HTML>
```

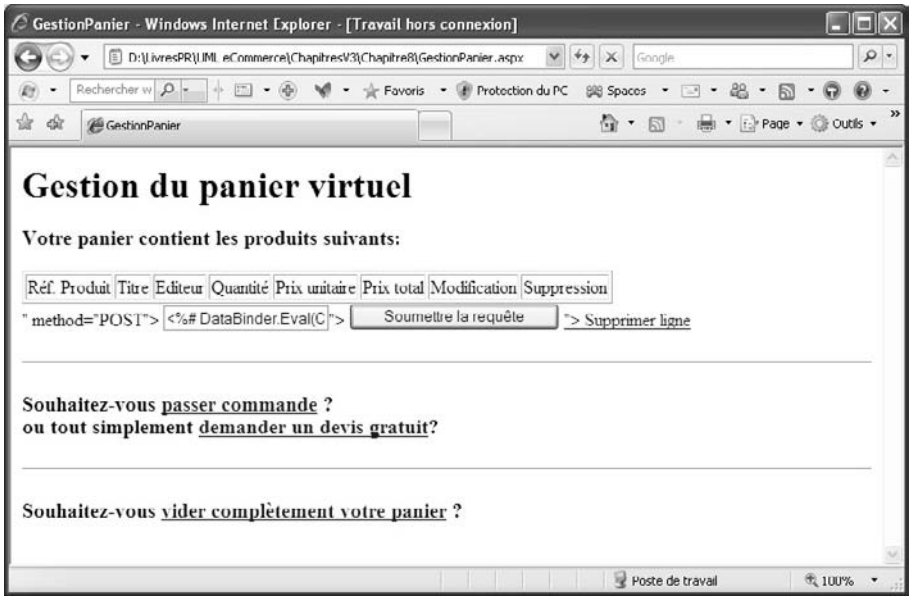


Figure 8–24
Exemple de page ASP.NET simple de gestion du panier

annexe

A

Résumé du sous-ensemble de la notation UML 2 utilisé dans ce livre

Diagramme de cas d'utilisation
Diagramme de séquence
Diagramme de classes
Diagramme de packages
Diagramme d'états

Diagramme de cas d'utilisation

Montre les interactions fonctionnelles entre les acteurs et le système à l'étude

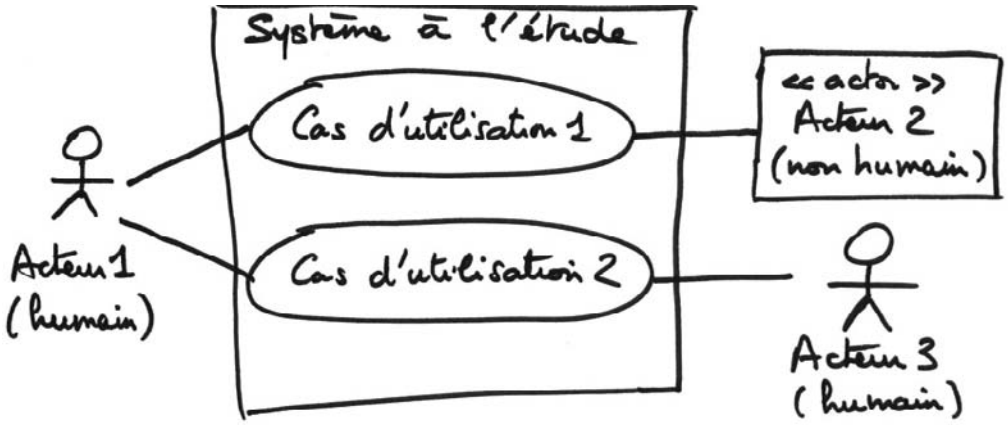


Figure A–1 Diagramme de cas d'utilisation (bases)

Acteur : rôle joué par un utilisateur humain ou un autre système qui interagit directement avec le système étudié. Un acteur participe à au moins un cas d'utilisation.

Cas d'utilisation (use case) : ensemble de séquences d'actions réalisées par le système produisant un résultat observable intéressant pour un acteur particulier. Collection de scénarios reliés par un objectif utilisateur commun.

Association : utilisée dans ce type de diagramme pour relier les acteurs et les cas d'utilisation par une relation qui signifie simplement « participe à ».

Inclusion : le cas d'utilisation de base en incorpore explicitement un autre, de façon obligatoire, à un endroit spécifié dans ses enchaînements.

Extension : le cas d'utilisation de base en incorpore implicitement un autre, de façon optionnelle, à un endroit spécifié indirectement dans celui qui procède à l'extension (déconseillé !)

Généralisation : les cas d'utilisation descendants héritent de la description de leur parent commun. Chacun d'entre eux peut néanmoins comprendre des relations spécifiques supplémentaires avec d'autres acteurs ou cas d'utilisation (déconseillé !). La généralisation d'acteurs est en revanche parfois utile.

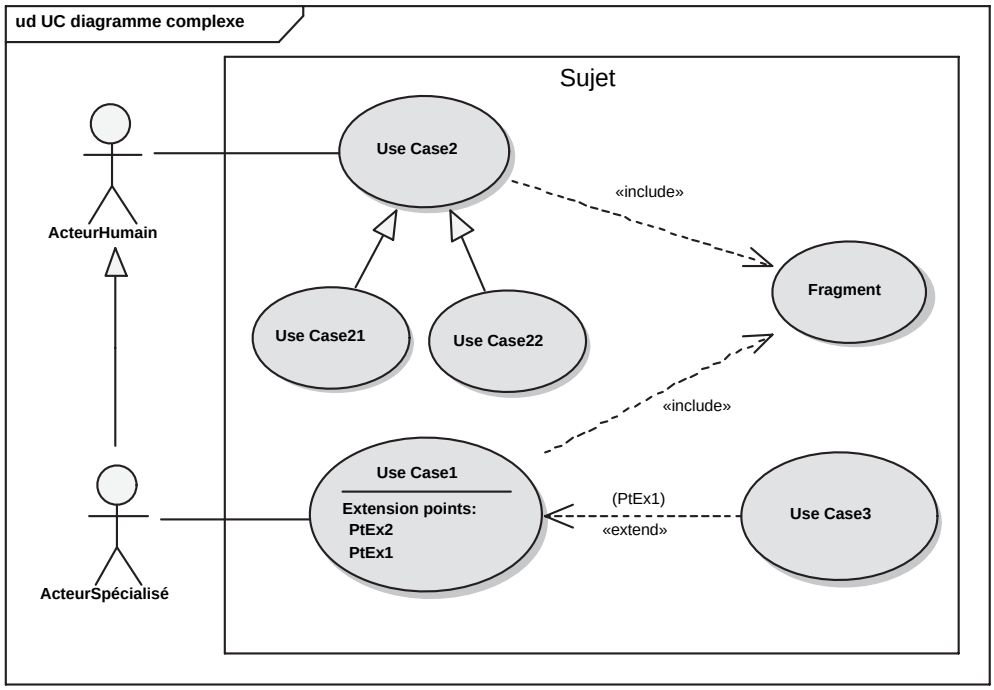


Figure A–2 Diagramme de cas d'utilisation (avancé)

Diagramme de séquence

Montre la séquence verticale des messages passés entre objets au sein d'une interaction

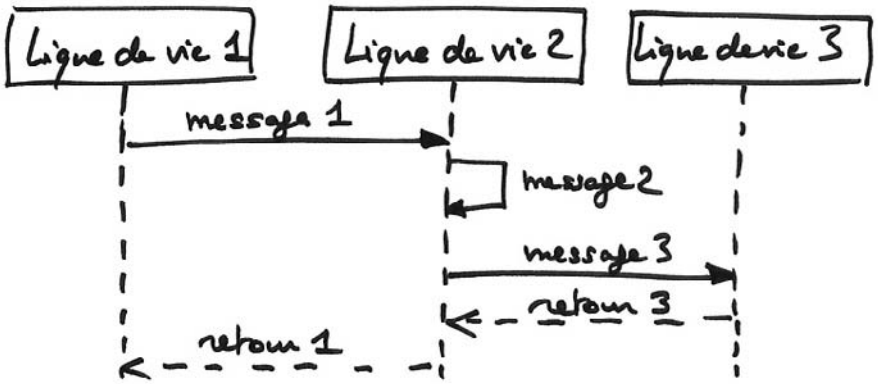


Figure A–3
Diagramme de séquence (bases)

Ligne de vie : représentation de l'existence d'un élément participant dans un diagramme de séquence. Cela peut être un acteur ou le système en modélisation d'exigences, des objets logiciels en conception préliminaire ou conception détaillée.

Message : élément de communication unidirectionnel entre objets qui déclenche une activité dans l'objet destinataire. La réception d'un message provoque un événement dans l'objet récepteur. La flèche pointillée représente un retour au sens UML. Cela signifie que le message en question est le résultat direct du message précédent.

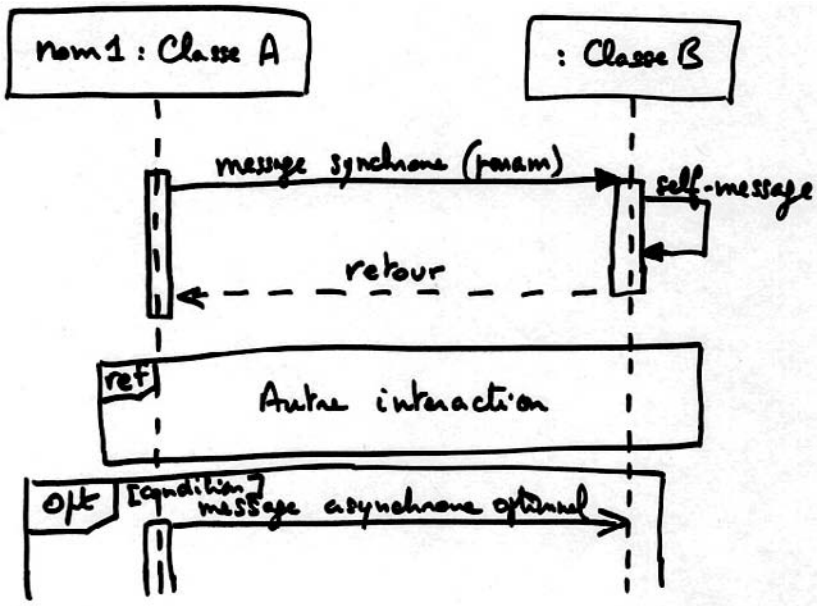


Figure A–4
Diagramme de séquence (avancé)

Spécification d'activation : bande blanche qui représente une période d'activité sur une ligne de vie.

Message synchrone : envoi de message pour lequel l'émetteur se bloque en attente du retour et qui est représenté par une flèche pleine. Un message asynchrone, au contraire, est représenté par une flèche ouverte.

Occurrence d'interaction : une interaction peut faire référence explicitement à une autre interaction grâce à un cadre avec le mot-clé `ref` et indiquant le nom de l'autre interaction.

UML 2 a ajouté une nouvelle notation très utile : les cadres d'interaction. Chaque cadre possède un opérateur et peut être divisé en fragments. Les principaux opérateurs sont :

- `loop` : boucle. Le fragment peut s'exécuter plusieurs fois, et la condition de garde explicite l'itération.

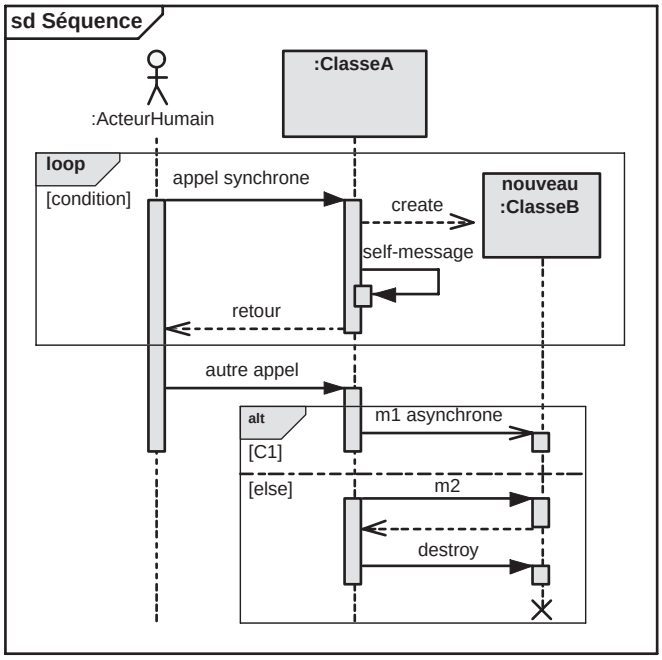


Figure A–5
Diagramme de séquence (avancé-suite)

- `opt` : optionnel. Le fragment ne s'exécute que si la condition fournie est vraie.
- `alt` : fragments alternatifs. Seul le fragment possédant la condition vraie s'exécutera.

Diagramme de classes

Montre les briques de base statiques : classes, associations, interfaces, attributs, opérations, généralisations, etc.

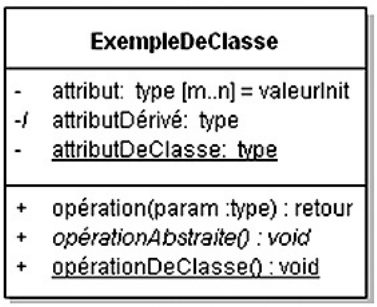


Figure A–6
Diagramme de classes (bases)

Classe : description abstraite d'un ensemble d'objets qui partagent les mêmes propriétés (attributs et associations) et comportements (opérations et états).

Attribut : donnée déclarée au niveau d'une classe, éventuellement typée, à laquelle chacun des objets de cette classe donne une valeur. Un attribut peut posséder une multiplicité et une valeur initiale. Un attribut dérivé (« / ») est un attribut dont la valeur peut être déduite d'autres informations disponibles dans le modèle.

Opération : élément de comportement des objets, défini de manière globale dans leur classe. Une opération peut déclarer des paramètres (eux-mêmes typés) ainsi qu'un type de retour.

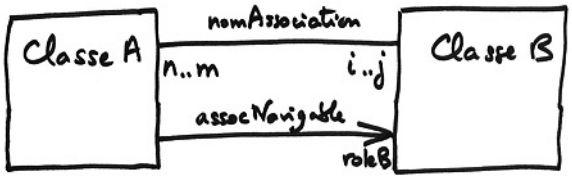


Figure A–7
Diagramme de classes (bases-suite1)

Association : relation sémantique durable entre deux classes, qui décrit un ensemble de liens entre instances. Une association est bidirectionnelle par défaut, sauf si l'on restreint sa navigabilité en ajoutant une flèche.

Rôle : nom donné à une extrémité d'une association ; par extension, manière dont les instances d'une classe voient les instances d'une autre classe au travers d'une association.

Multiplicité : le nombre d'objets (`min..max`) qui peuvent participer à une relation avec un autre objet dans le cadre d'une association. Multiplicités fréquentes :

- `0..1` = optionnel (mais pas multiple)
- `1` = exactement 1
- `0..*` = * = quelconque
- `1..*` = au moins 1

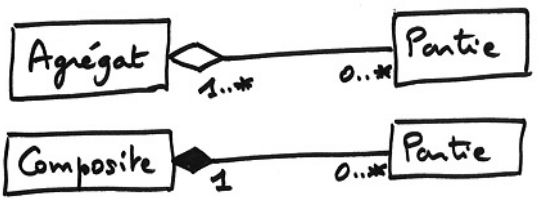


Figure A–8
Diagramme de classes (bases-suite2)

Agrégation : cas particulier d'association non symétrique exprimant une relation de contenance.

Composition : forme forte d'agrégation, dans laquelle les parties ne peuvent appartenir à plusieurs agrégats et où le cycle de vie des parties est subordonné à celui de l'agrégat.

Figure A–9
Diagramme de classes (bases-suite3)

Super-classe : classe générale reliée à d'autres classes plus spécialisées (sous-classes) par une relation de généralisation.

Généralisation : relation entre « classifieurs » où les descendants héritent des propriétés de leur parent commun. Ils peuvent néanmoins comprendre chacun des propriétés spécifiques supplémentaires, mais aussi modifier les comportements hérités.

Figure A–10
Diagramme de classes (bases-suite4)

Note : commentaire visible dans le diagramme. Peut être attaché à un élément du modèle (ou plusieurs) par un trait pointillé. Utilisable dans tout type de diagramme UML !

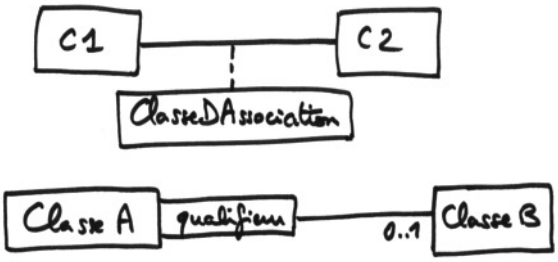


Figure A–11
Diagramme de classes (avancé)

Classe d'association : association promue au rang de classe. Elle possède tout à la fois les caractéristiques d'une association et celles d'une classe et peut donc porter des attributs qui prennent des valeurs pour chaque lien entre objets.

Qualifieur (ou qualificatif) : attribut qui permet de « partitionner » l'ensemble des objets en relation avec un objet donné dans le cadre d'une association multiple.

Figure A–12
Diagramme de classes (avancé-suite1)

Contrainte : condition portant sur un ou plusieurs élément(s) du modèle qui doit être vérifiée par les éléments concernés. Elle est notée entre accolades `{ }`, et souvent insérée dans une note graphique (le post-it).

Figure A–13
Diagramme de classes (avancé-suite2)

Dépendance : relation sémantique entre deux éléments, dans laquelle la modification d'un des éléments (l'élément indépendant) peut affecter la sémantique de l'autre élément (l'élément dépendant).

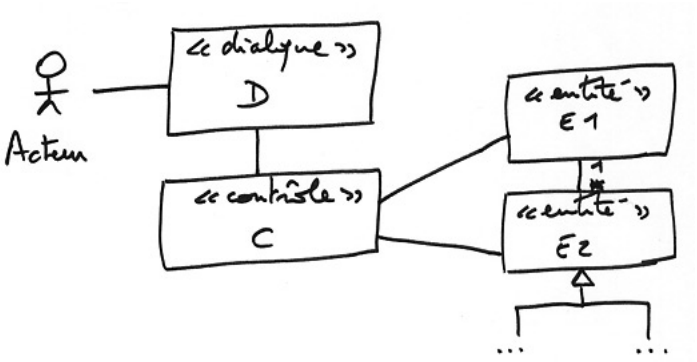


Figure A–14
Diagramme de classes (avancé-suite3)

Catégorisation des **classes d'analyse** en trois catégories qui a été proposée par I. Jacobson et popularisée ensuite par le RUP (Rational Unified Process) :

- Celles qui permettent les interactions entre le site web et ses utilisateurs sont qualifiées de « dialogues ». Il s'agit typiquement des écrans proposés à l'utilisateur.
- Les classes qui contiennent la cinématique de l'application sont appelées « contrôles ». Elles font la transition entre les dialogues et les concepts du domaine, et contiennent les règles applicatives.
- Celles qui représentent les concepts métier sont qualifiées d'« entités ». Elles sont très souvent persistantes.

Diagramme de packages

Montre l'organisation logique du modèle et les relations entre packages

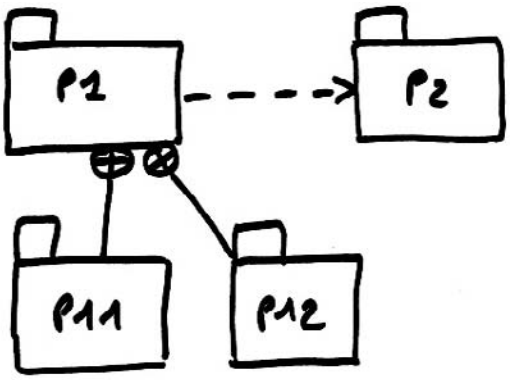


Figure A–15
Diagramme de packages

Package (ou paquetage) : mécanisme général de regroupement d'éléments tels que classes, interfaces, mais aussi acteurs, cas d'utilisation, etc. Les packages peuvent être imbriqués dans d'autres packages.

Importation : relation de dépendance entre packages qui rend visibles les éléments publics de l'un des packages au sein d'un autre.

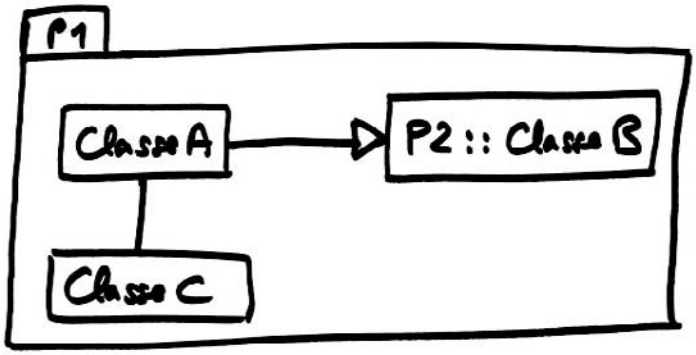


Figure A–16
Diagramme de packages (suite)

Les packages sont des espaces de noms (*namespace*) : deux éléments ne peuvent pas porter le même nom au sein du même package. En revanche, deux éléments appartenant à des packages différents peuvent porter le même nom. Du coup, UML propose une notation complète pour un élément : `nomPackage::nomÉlément`.

Diagramme d'états

Montre les différents états et transitions possibles des objets d'une classe à l'exécution

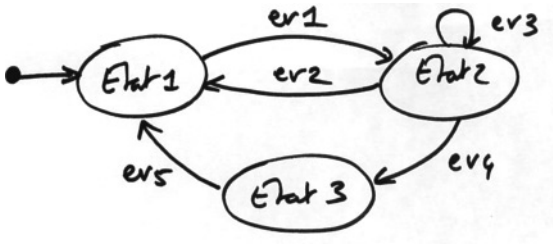


Figure A–17
Diagramme d'états (bases)

État : condition ou situation qui se produit dans la vie d'un objet pendant laquelle il satisfait une certaine condition, exécute une activité particulière ou attend certains événements.

Événement : spécification d'une occurrence remarquable qui a une localisation dans le temps et l'espace. Un événement peut porter des paramètres qui matérialisent le flot d'information ou de données entre objets.

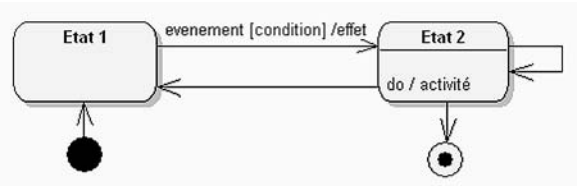


Figure A–18
Diagramme d'états (bases-suite)

Transition : connexion entre deux états d'un automate, qui est déclenchée par l'occurrence d'un événement, et conditionnée par une condition de garde, induisant certains effets.

Effet : action ou activité qui s'exécute lorsqu'une transition se déclenche. L'exécution de l'effet est unitaire et ne permet de traiter aucun événement supplémentaire pendant son déroulement.

Un état peut contenir une **activité durable** (do-activity), qui est interruptible par un événement, mais peut également se terminer d'elle-même.

annexe

B

Récapitulatif du modèle UML 2 illustrant la démarche de modélisation d'un site e-commerce

Modèle des cas d'utilisation
Modèle d'analyse
Modèle de navigation
Modèle de conception préliminaire
Modèle de conception détaillée

Modèle des cas d'utilisation

Structuration en packages

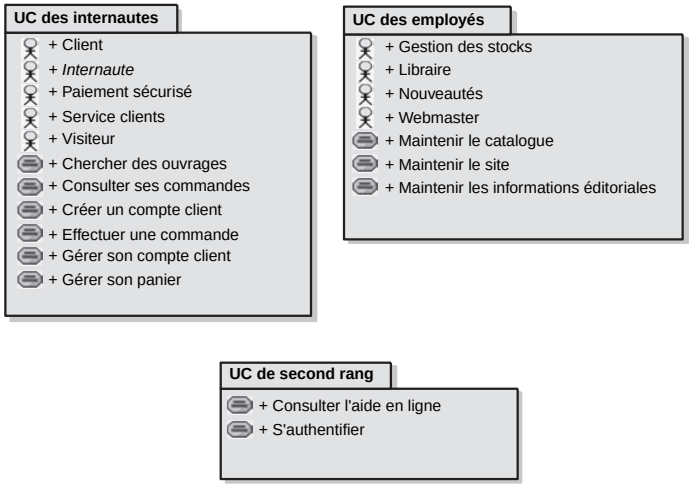


Figure B–1
Packages de la vue des cas d'utilisation

Package des cas d'utilisation des internautes

Diagramme de cas d'utilisation

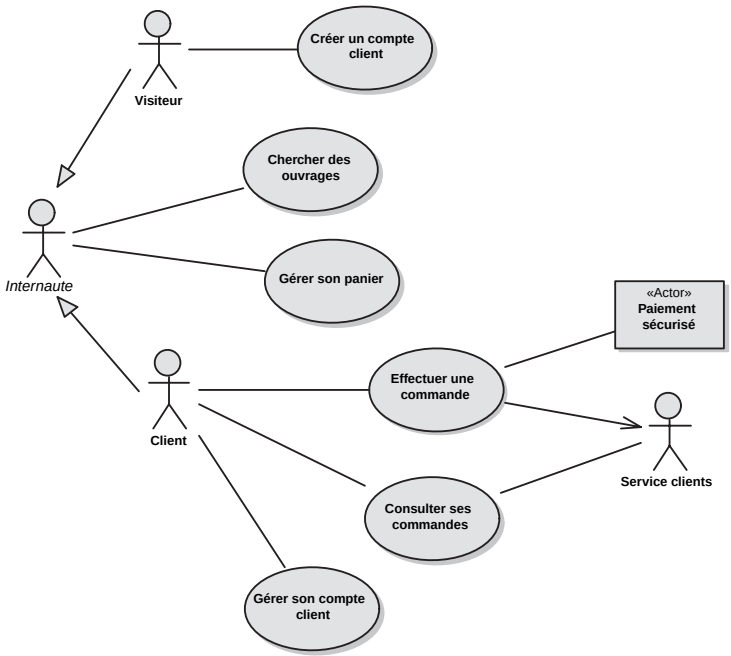


Figure B–2
Cas d'utilisation des internautes

Chercher des ouvrages

Diagramme de Séquence Système (DSS)

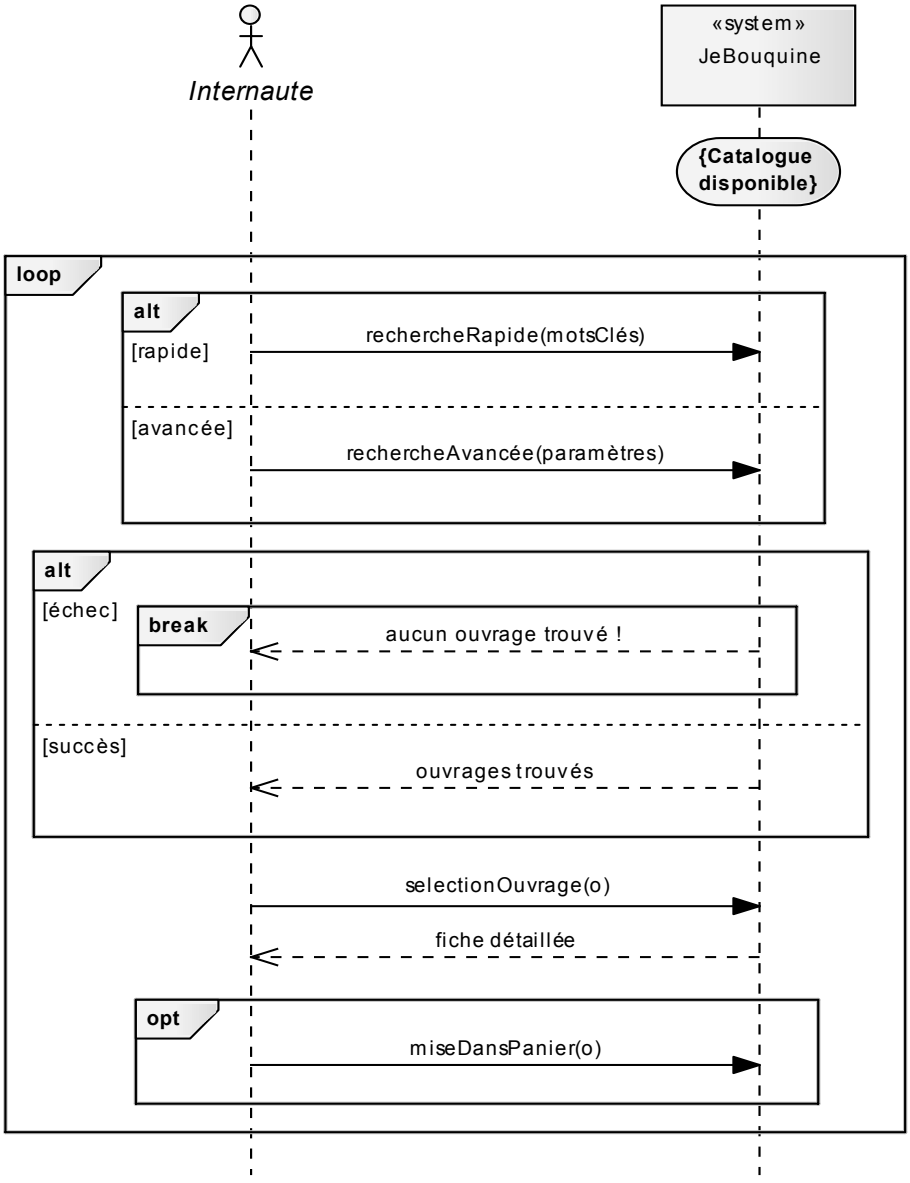


Figure B–3
Diagramme de séquence système de Chercher des ouvrages

Gérer son panier

Diagramme de Séquence Système (DSS)

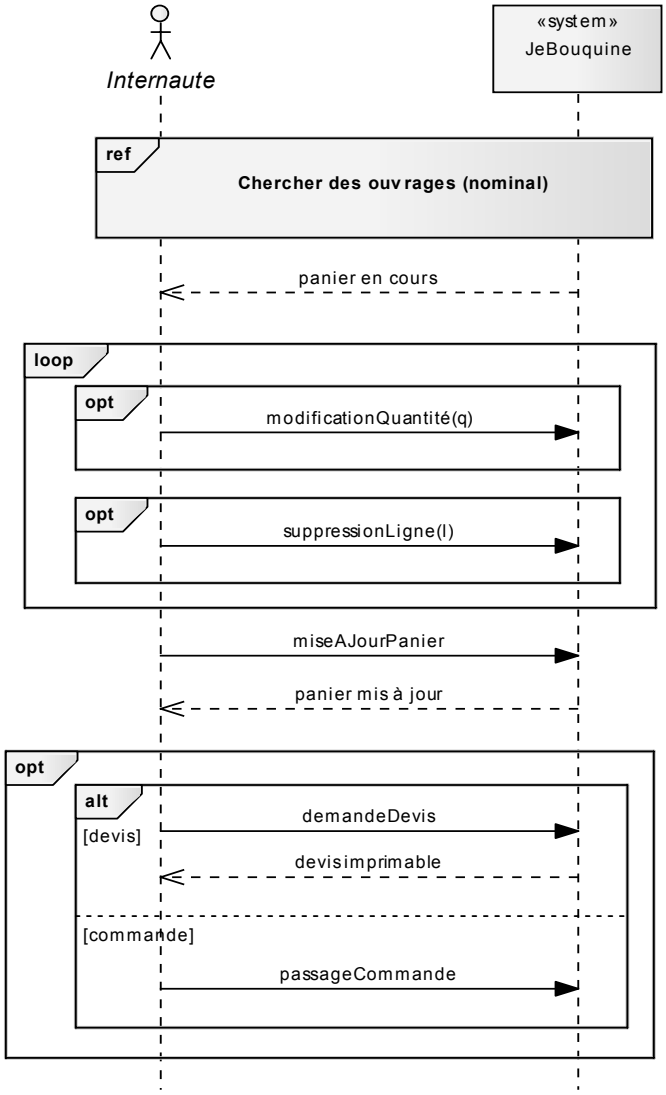


Figure B–4
Diagramme de séquence système de Gérer son panier

Effectuer une commande

Diagramme de Séquence Système (DSS)

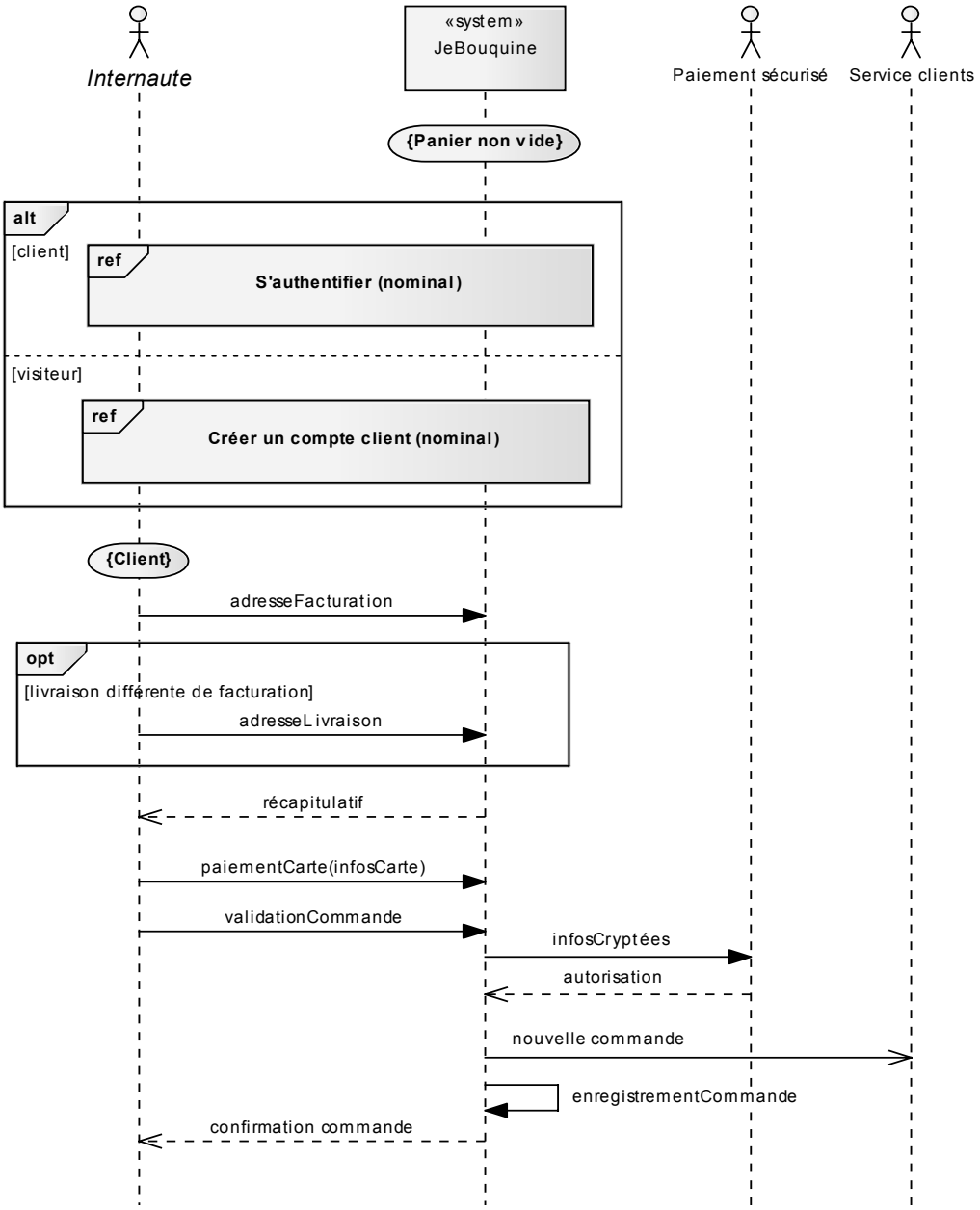


Figure B–5
Diagramme de séquence système de Effectuer une commande

Package des cas d'utilisation des employés

Diagramme de cas d'utilisation

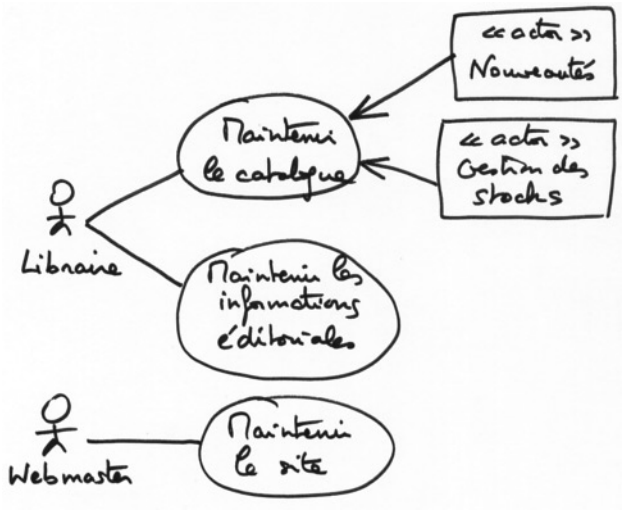


Figure B–6
Cas d'utilisation des employés

Maintenir le Catalogue

Diagramme de Séquence Système (DSS)

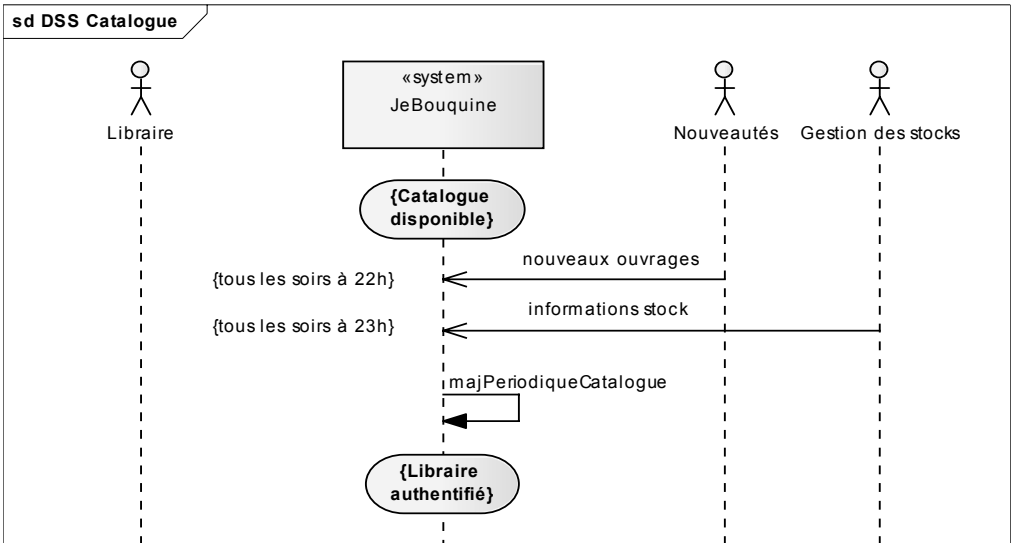


Figure B–7 Diagramme de séquence système de Maintenir le catalogue (1re partie)

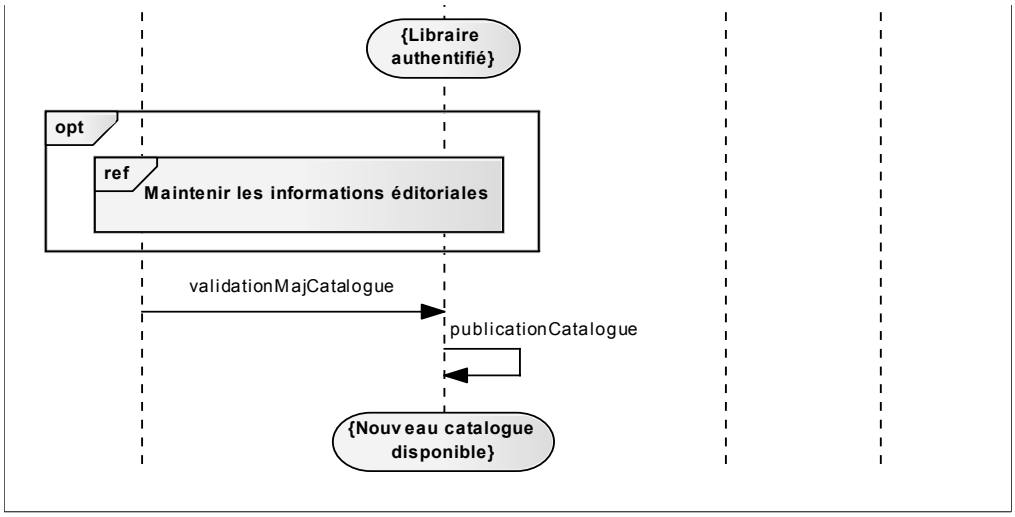


Figure B–7 Diagramme de séquence système de Maintenir le catalogue (2e partie)

Package des cas d'utilisation de second rang

Diagramme de cas d'utilisation

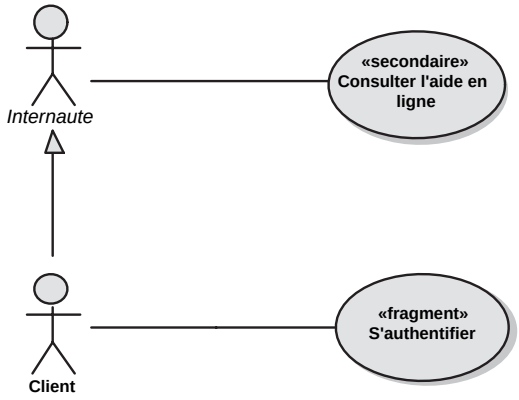


Figure B–8
Cas d'utilisation de second rang

Modèle d'analyse

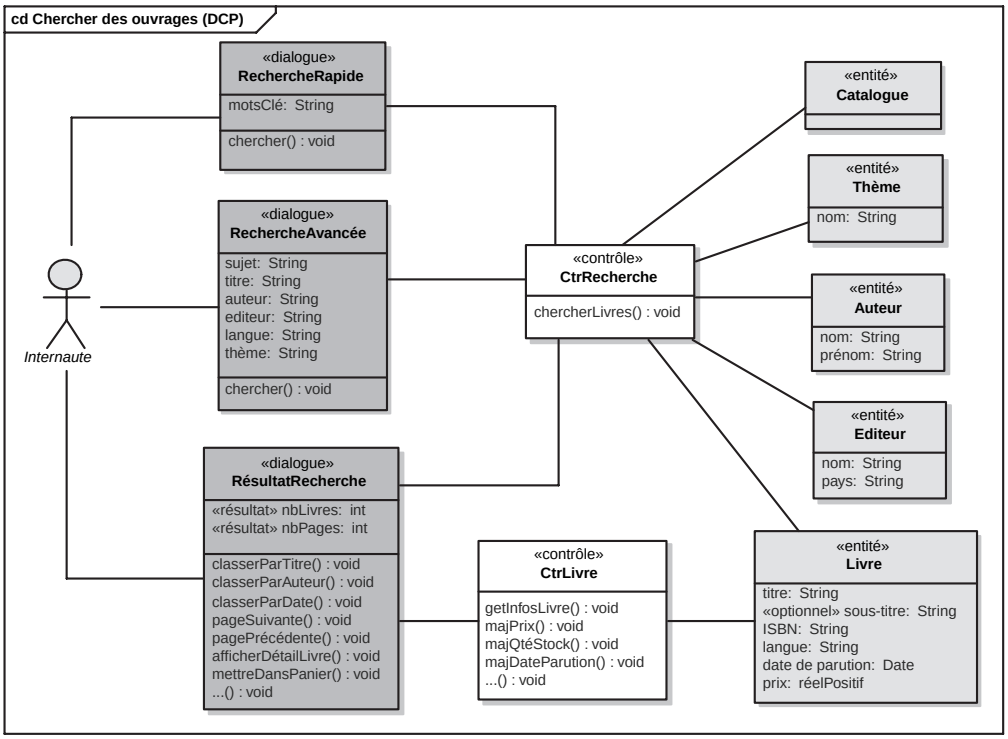


Figure B–9
DCP de Chercher des ouvrages

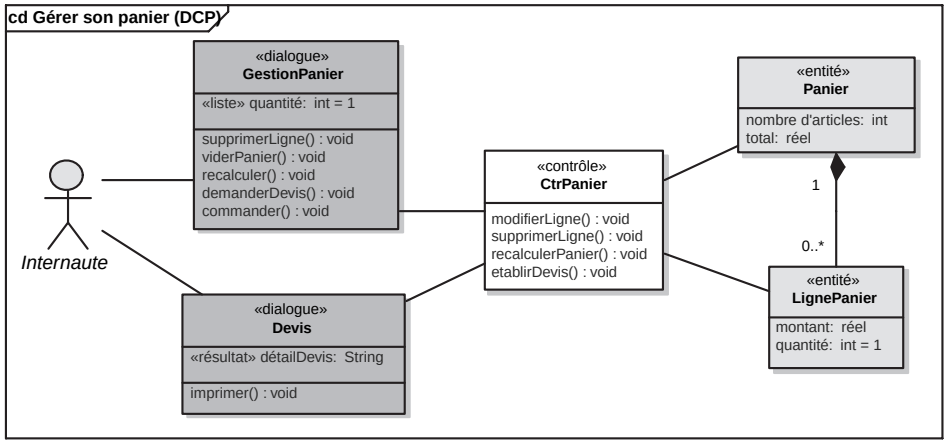


Figure B–10
DCP de Gérer son panier

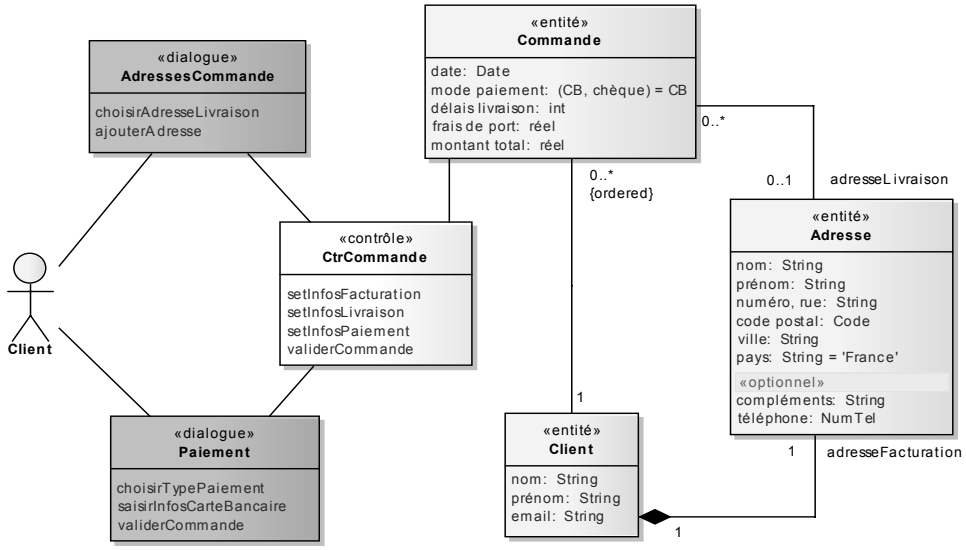


Figure B–11 DCP de Effectuer une commande

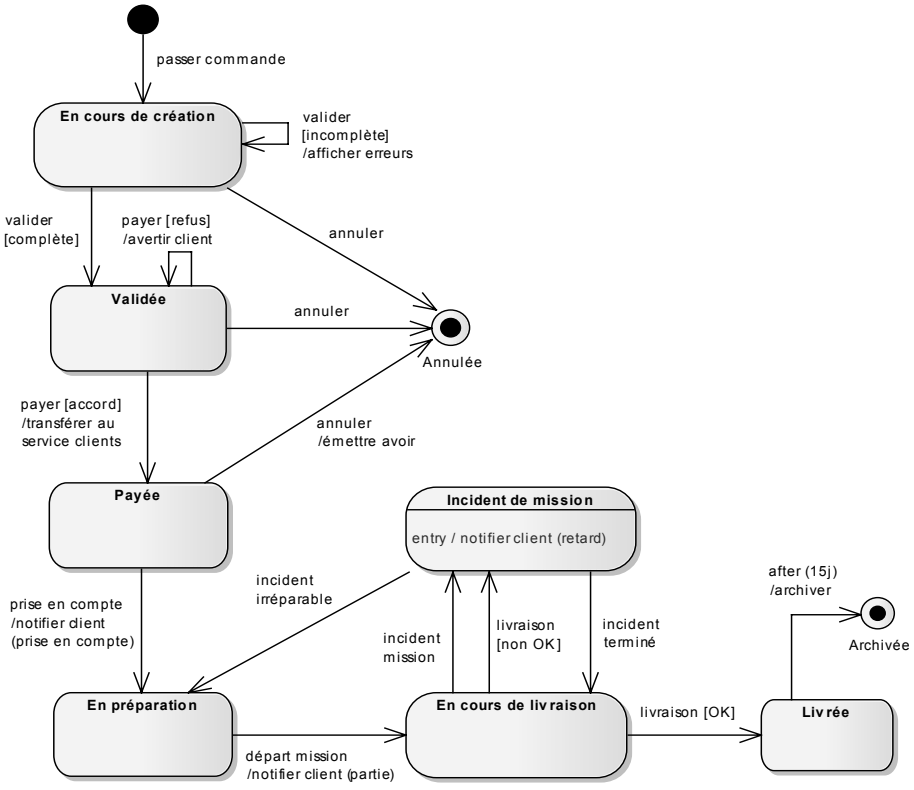


Figure B–12 Diagramme d'états de la commande

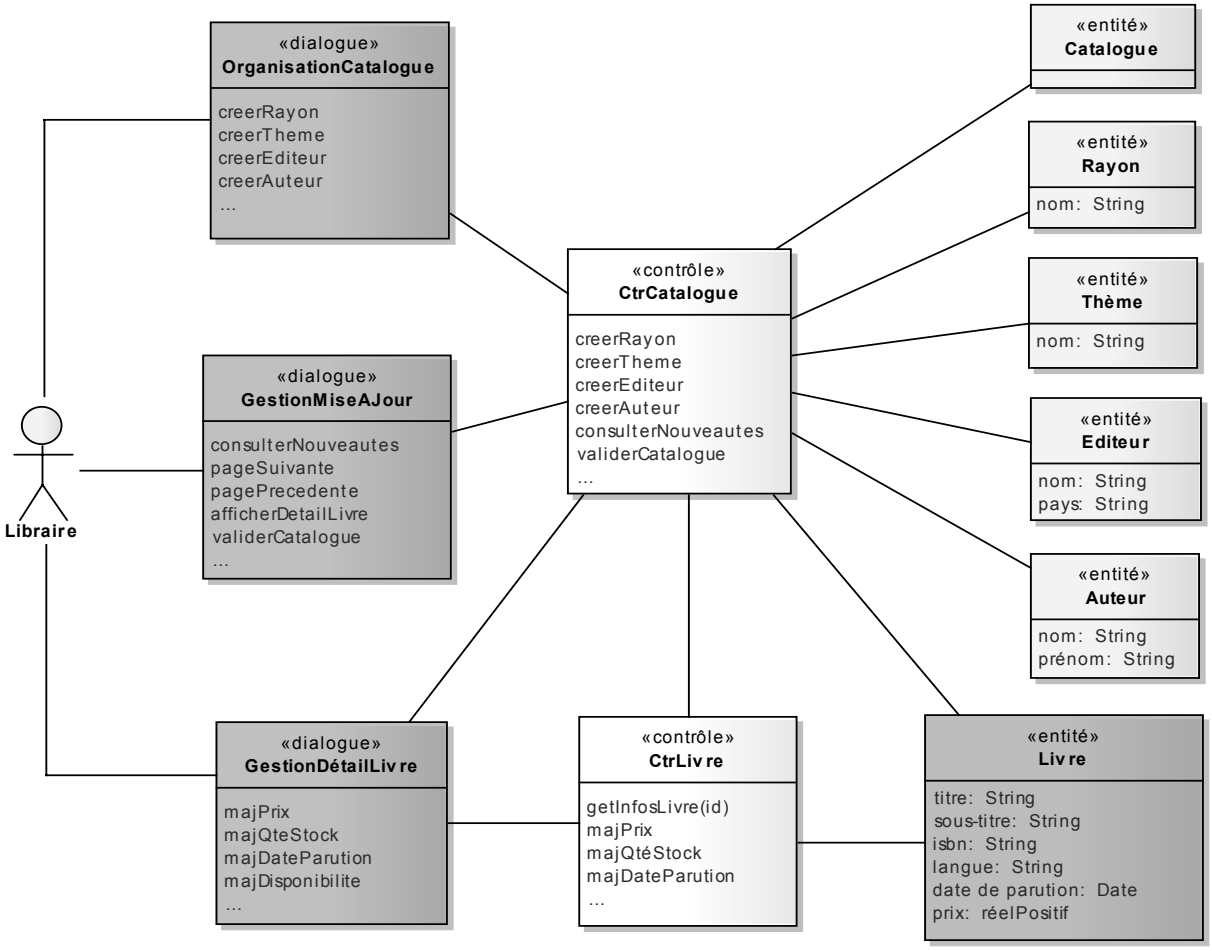


Figure B–13
DCP de Maintenir le Catalogue

Modèle de navigation

Navigation de la recherche

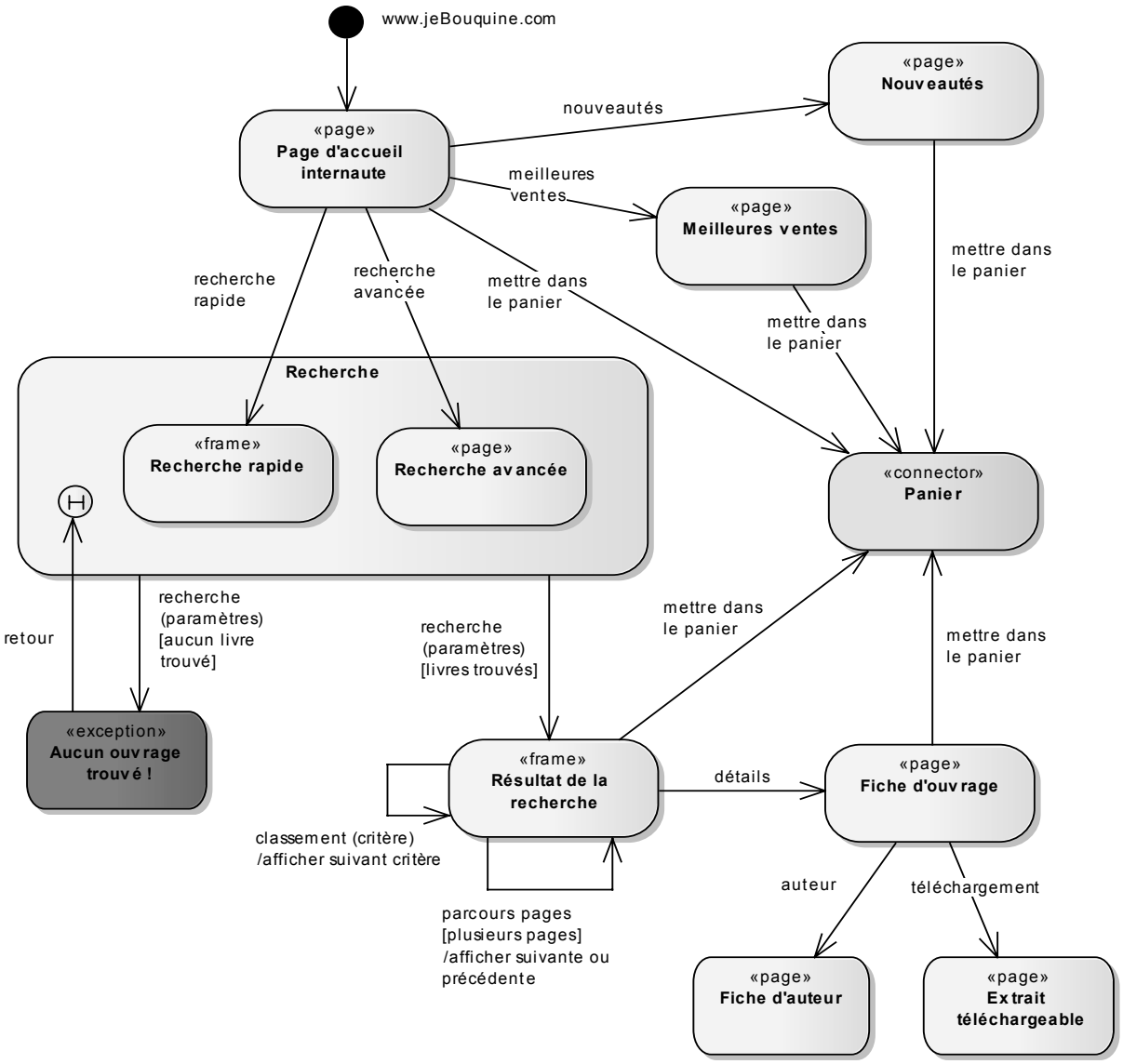


Figure B–14 Diagramme de navigation de la recherche d'ouvrages

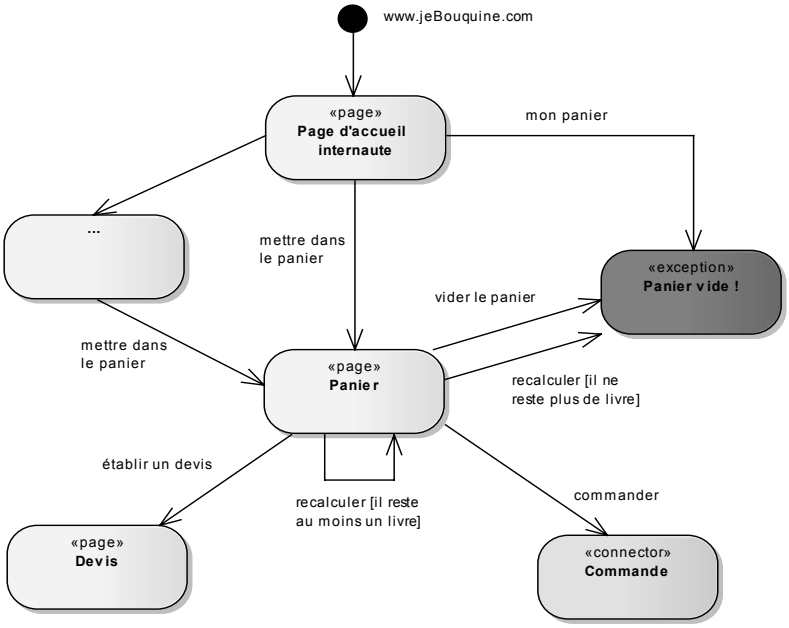


Figure B–15
Diagramme de navigation de la gestion du panier

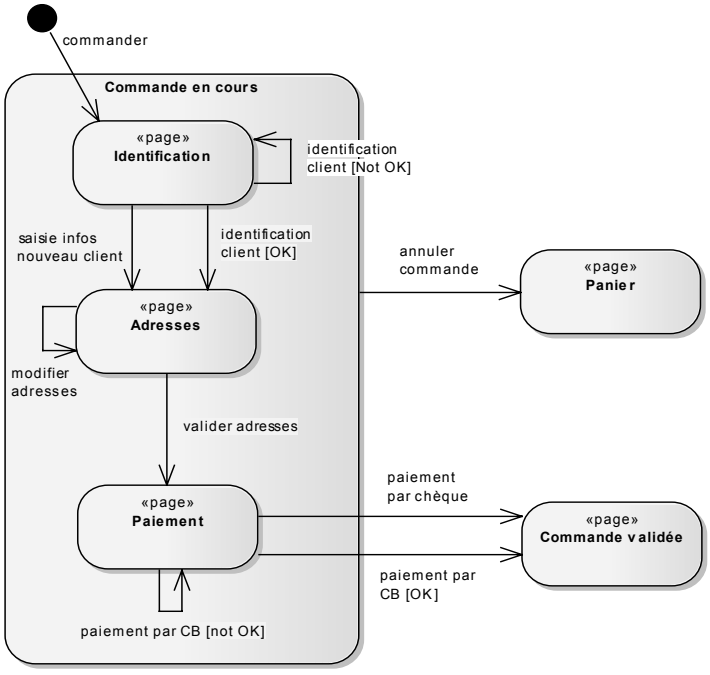


Figure B–16
Diagramme de navigation pour la prise de commande

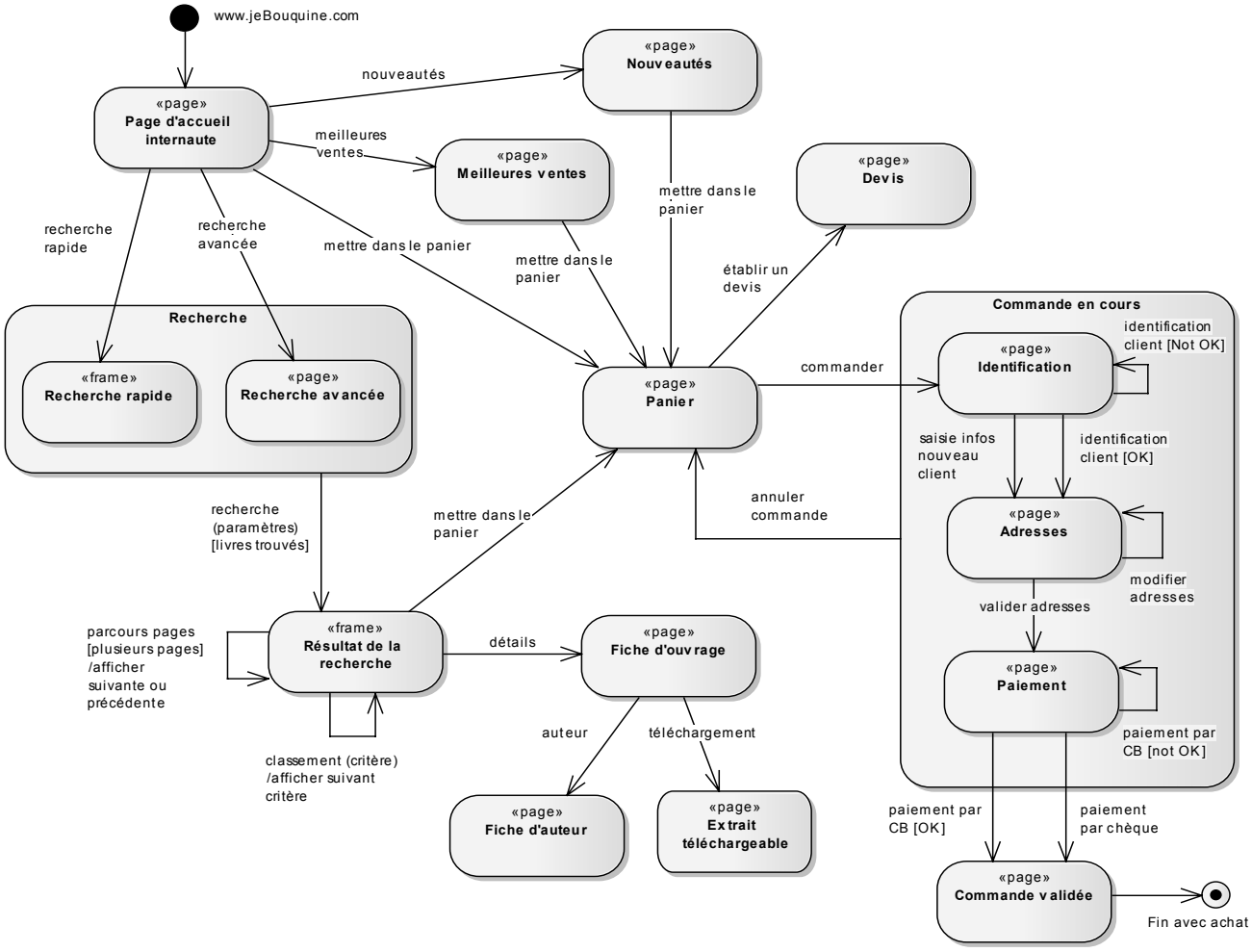


Figure B–17
Diagramme global de navigation pour l'internaute

Modèle de conception préliminaire

Diagrammes de séquence

Chercher des ouvrages

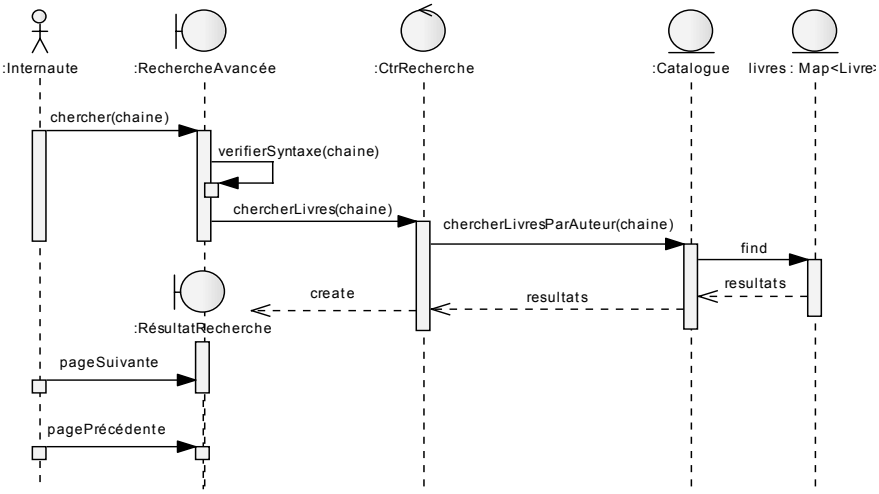


Figure B–18
Diagramme de séquence du scénario nominal de recherche avancée

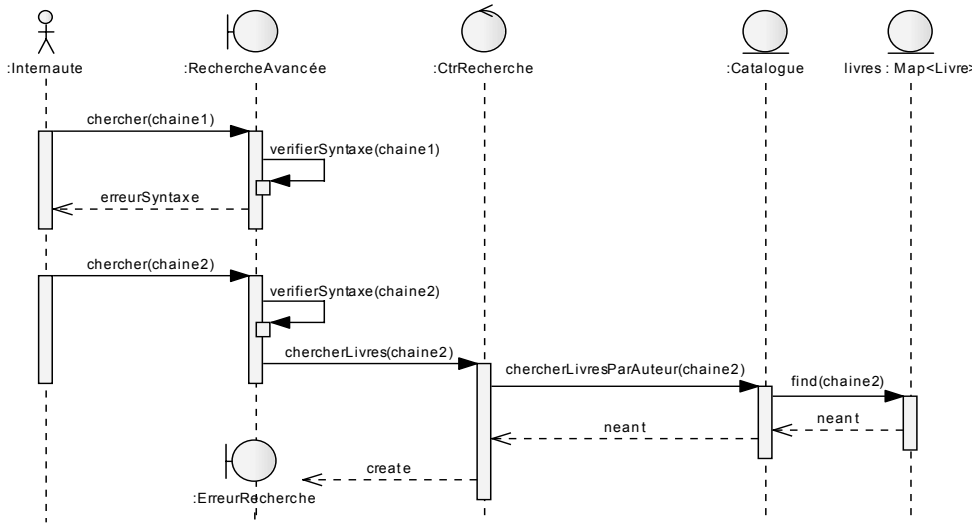


Figure B–19
Diagramme de séquence des scénarios d'erreur de recherche avancée

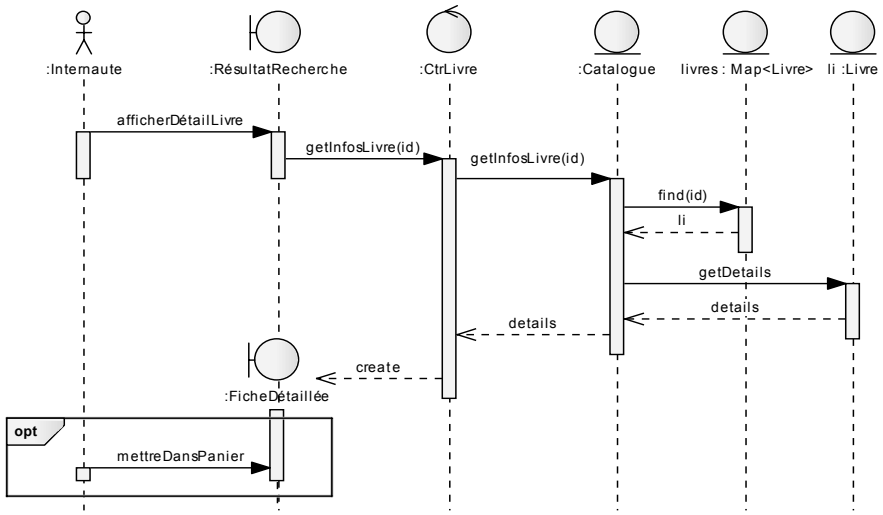


Figure B–20 Diagramme de séquence
de la suite du scénario nominal de recherche avancée

Gérer son panier

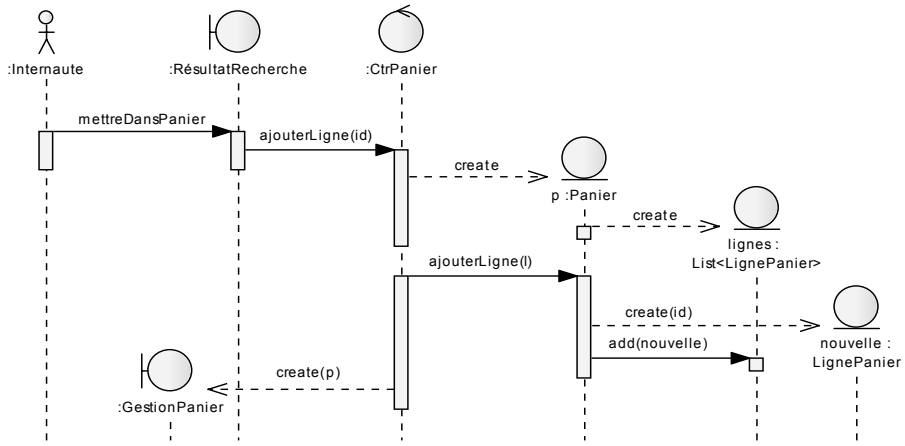


Figure B–21 Diagramme de séquence
du scénario nominal de création de la première ligne du panier

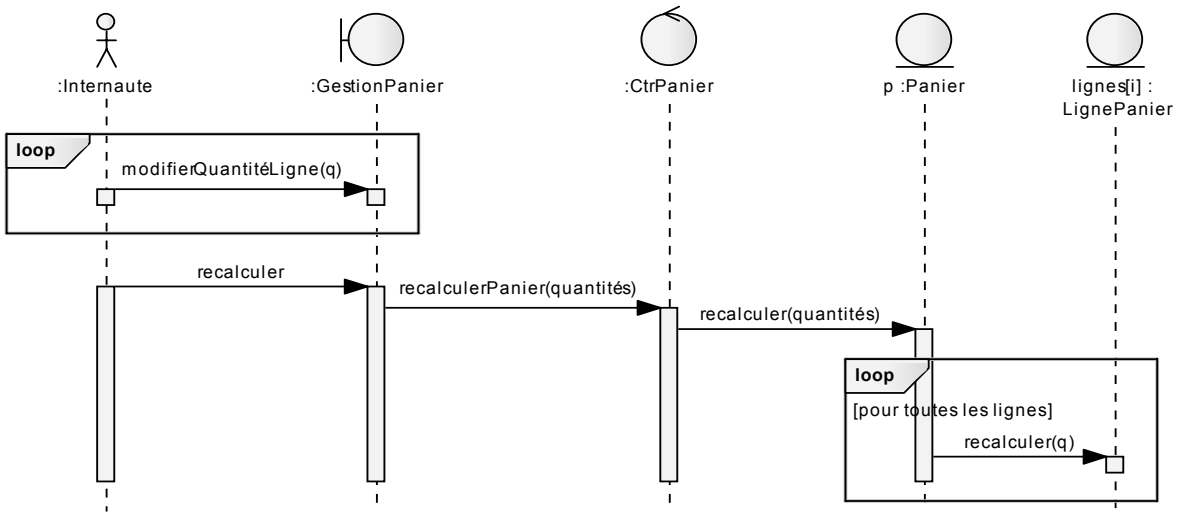


Figure B–22
Diagramme de séquence du scénario de recalcul du panier

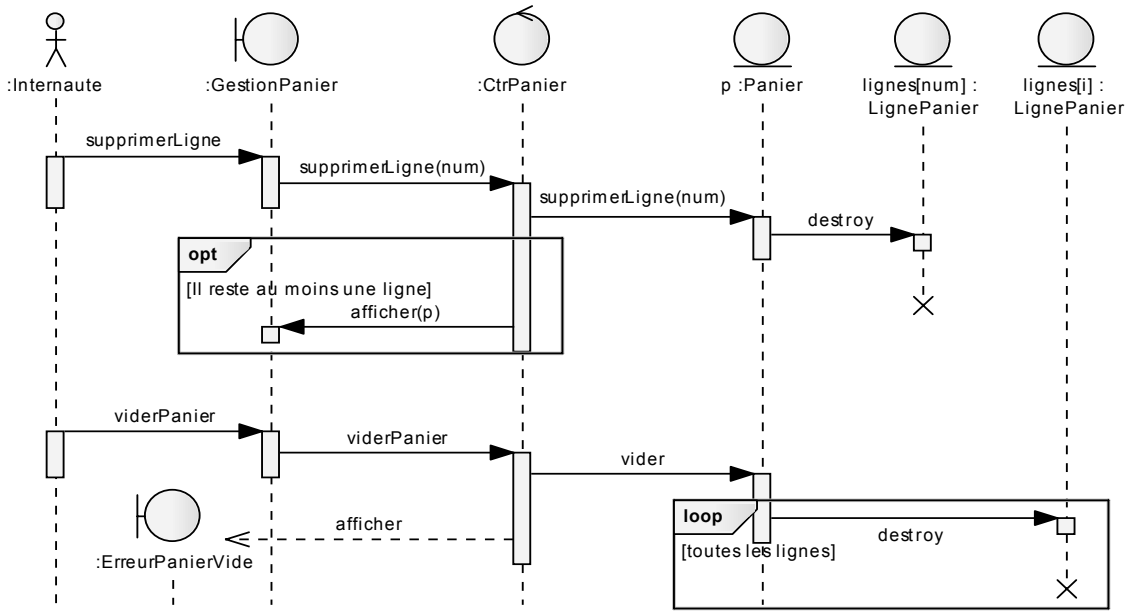


Figure B–23
Diagramme de séquence d'un scénario de vidage du panier

Diagrammes de classes de conception préliminaire

Chercher des ouvrages

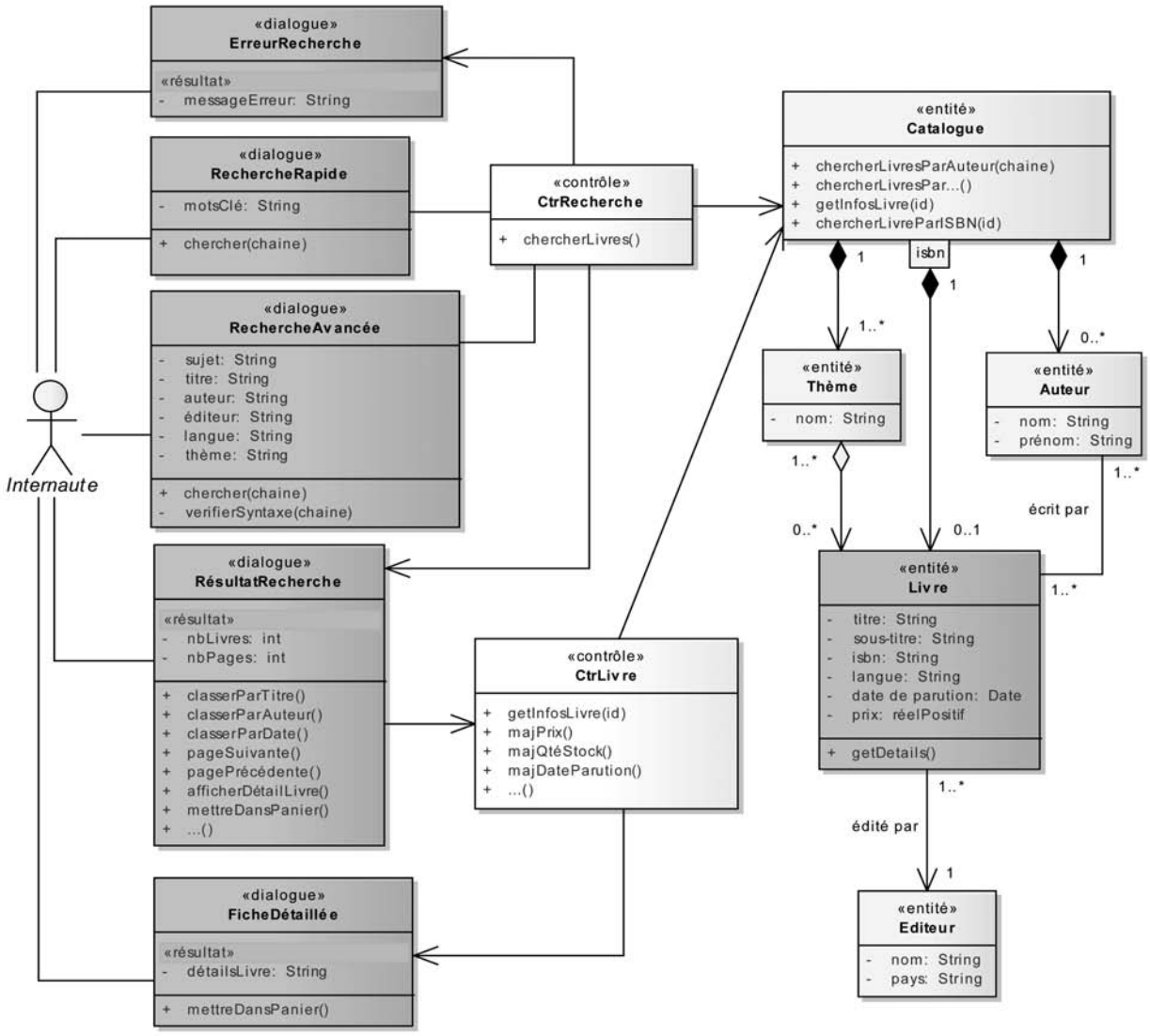


Figure B–24
Diagramme de classes de conception préliminaire de Chercher des ouvrages

Gérer son panier

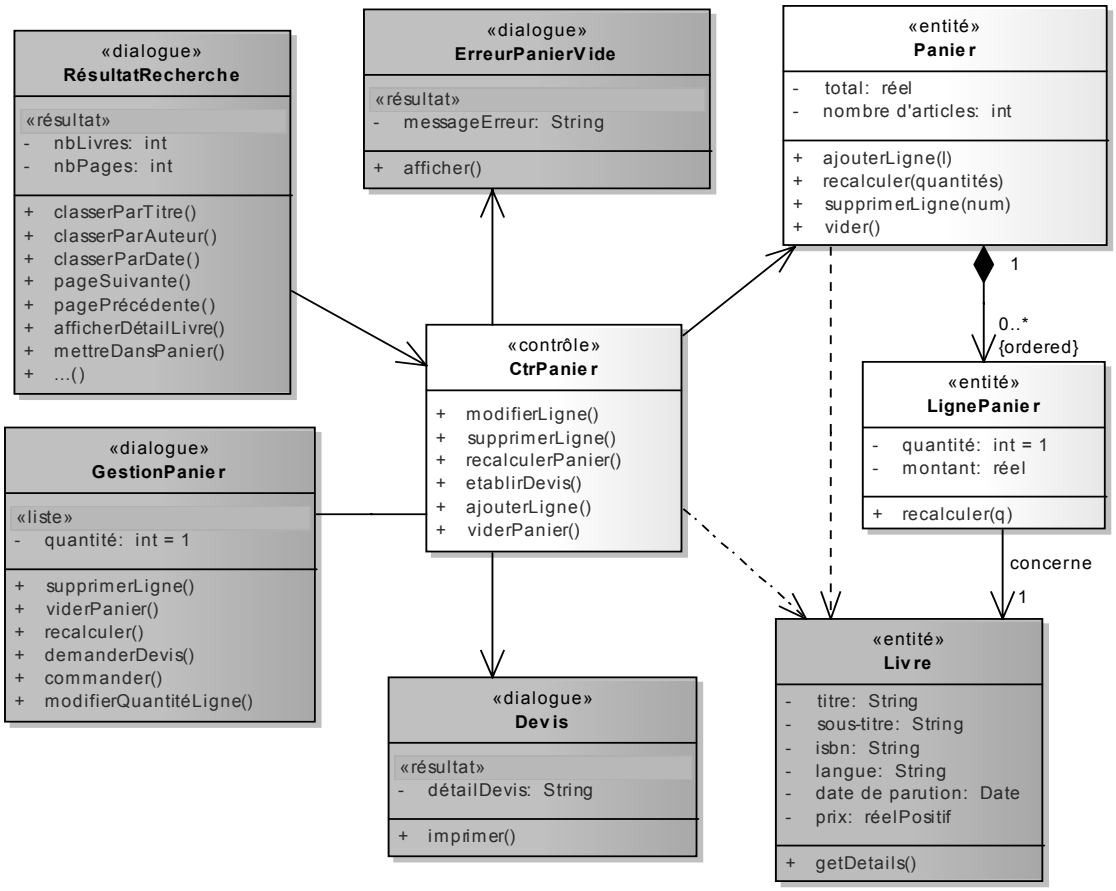


Figure B–25
Diagramme de classes de conception préliminaire de Gérer son panier

Structuration en packages

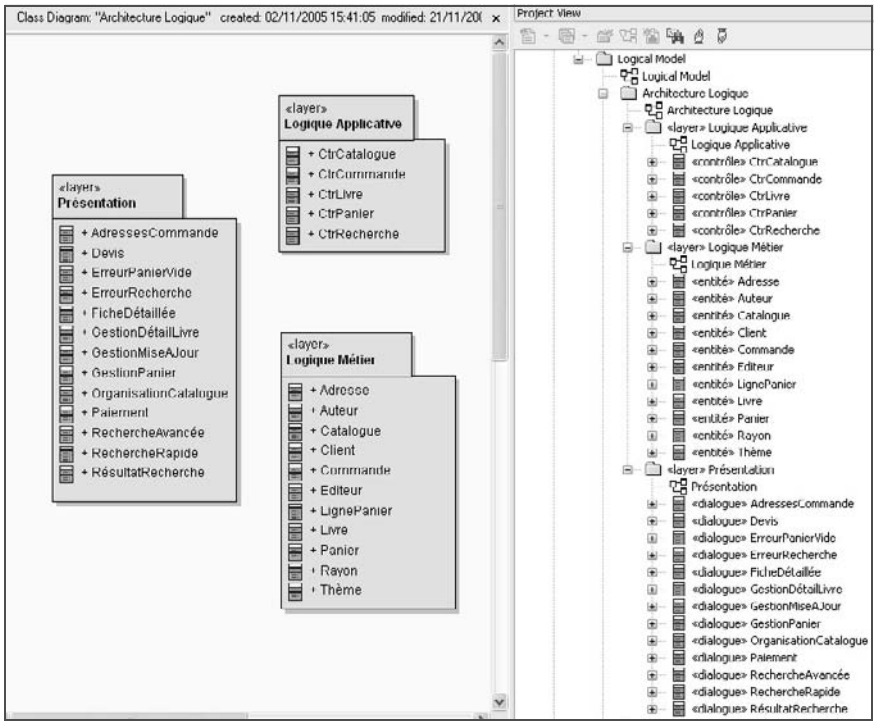


Figure B–26
Détail de l'architecture logique

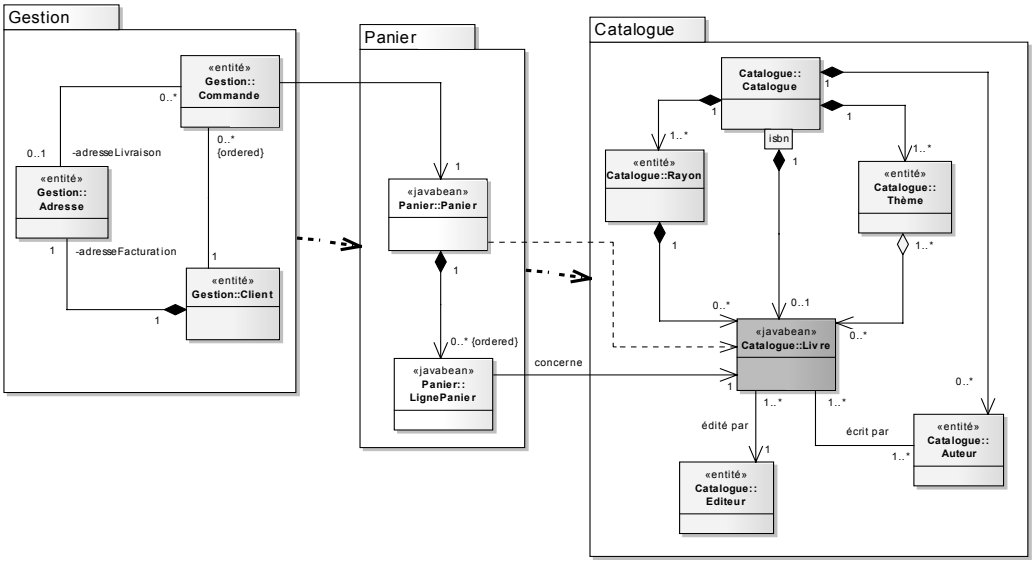


Figure B–27 Détail de la couche Logique Métier

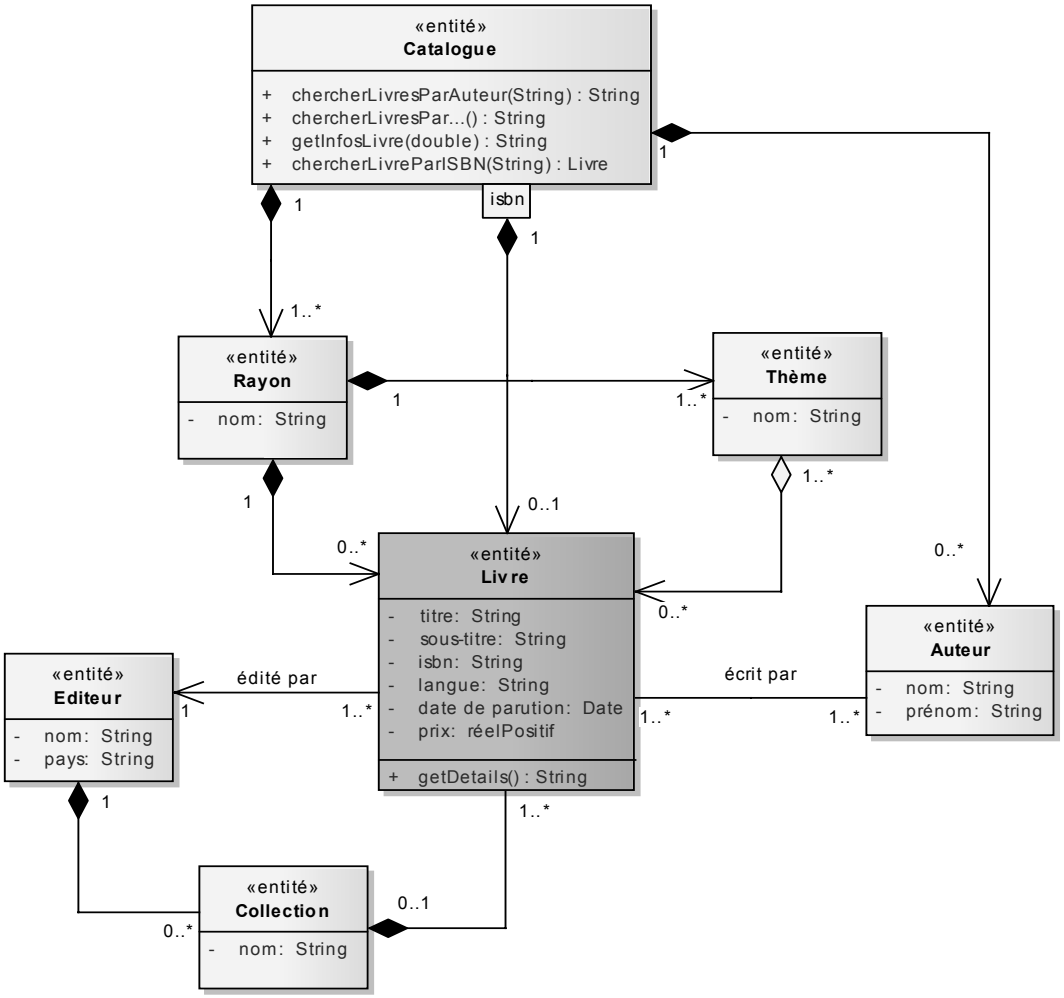


Figure B–28
Diagramme de classes du package Catalogue

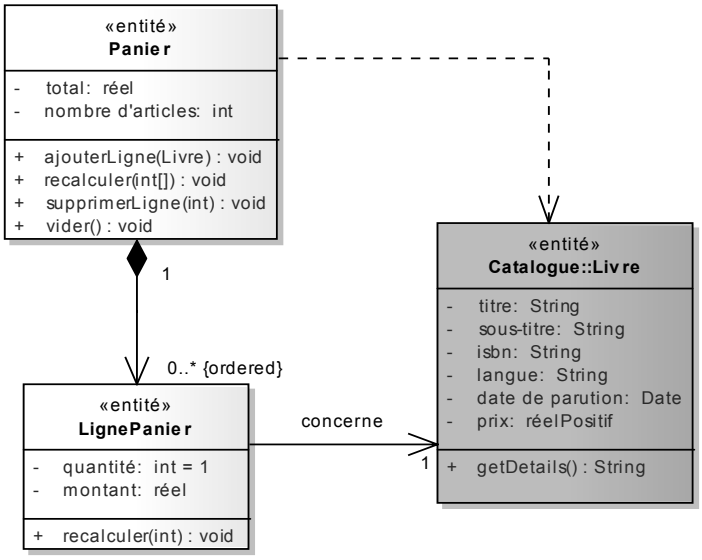


Figure B–29
Diagramme de classes complété du package Panier

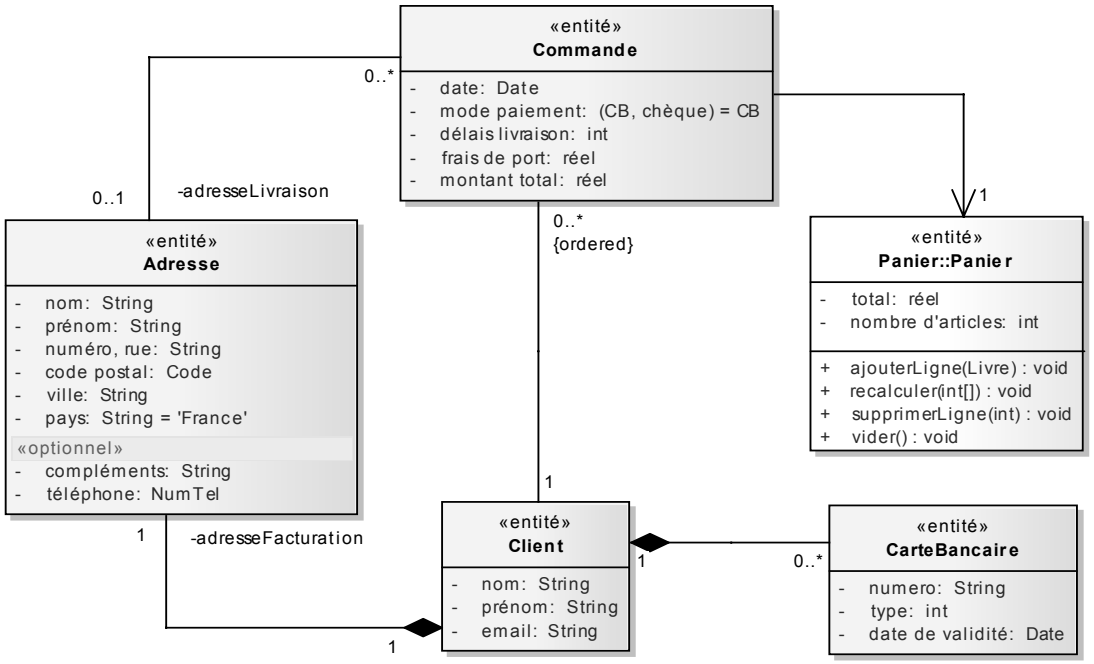


Figure B–30
Diagramme de classes complété du package Gestion

Modèle de conception détaillée

Solution à base de scripts (PHP)

Diagrammes d'interaction

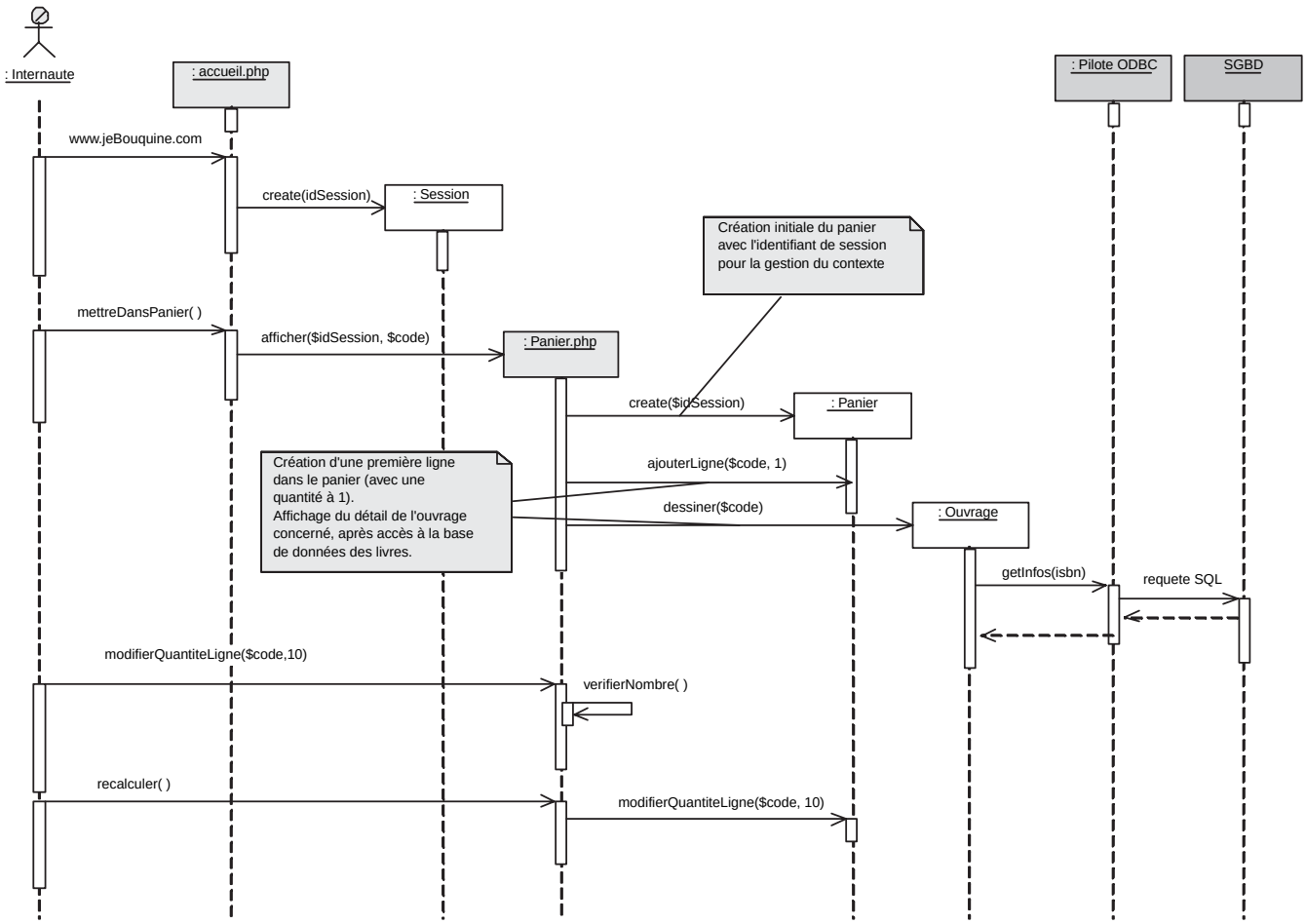


Figure B–31 Exemple de diagramme de séquence de conception détaillée PHP pour la gestion du panier

Diagrammes de classes

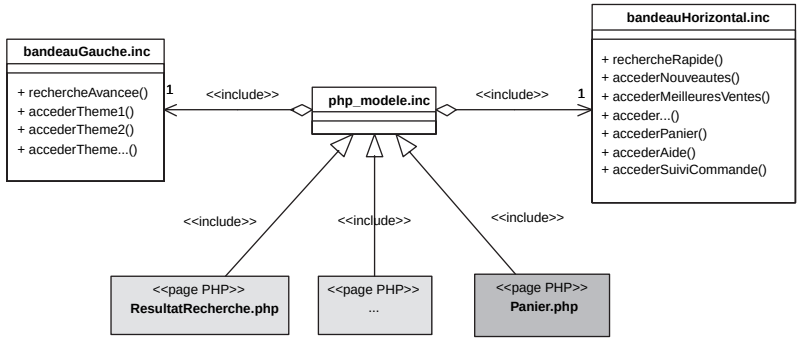


Figure B–32
Diagramme de classes des pages PHP

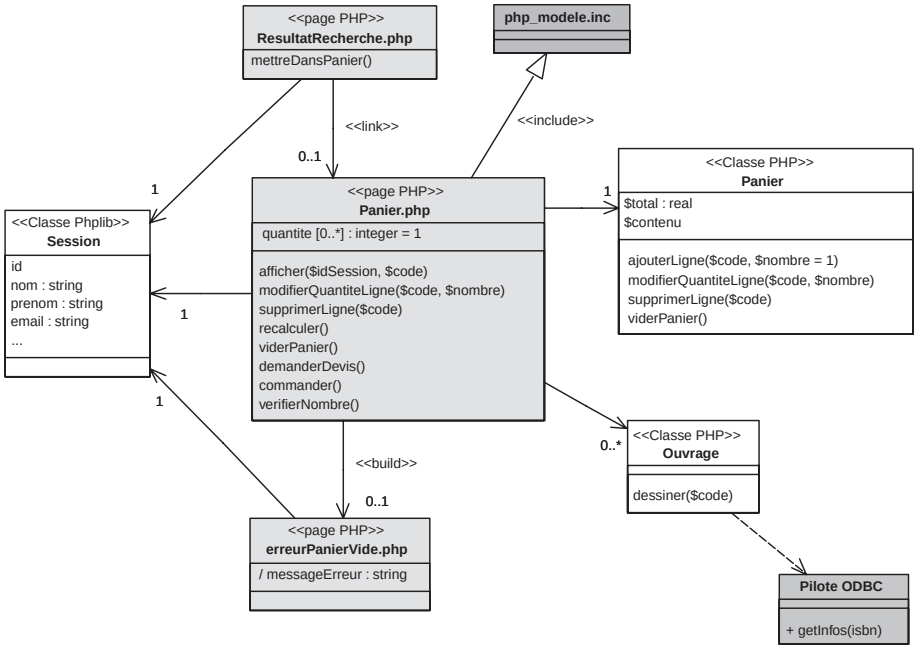


Figure B–33
Diagramme de classes de conception détaillée PHP de la gestion du panier

Solution technique J2EE (Struts)

Architecture logique

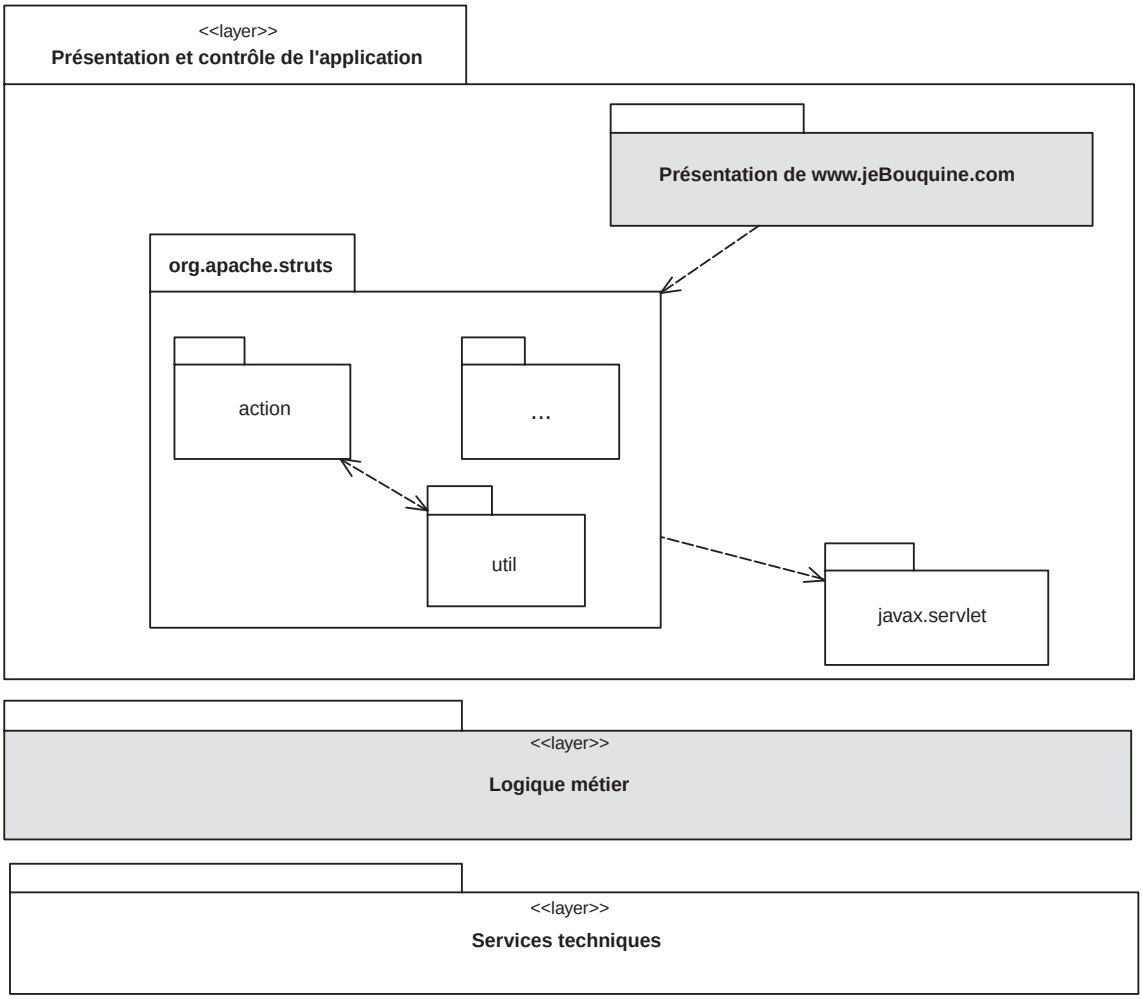


Figure B–34
Packages et couches notables reliés à Struts

Diagrammes de séquence

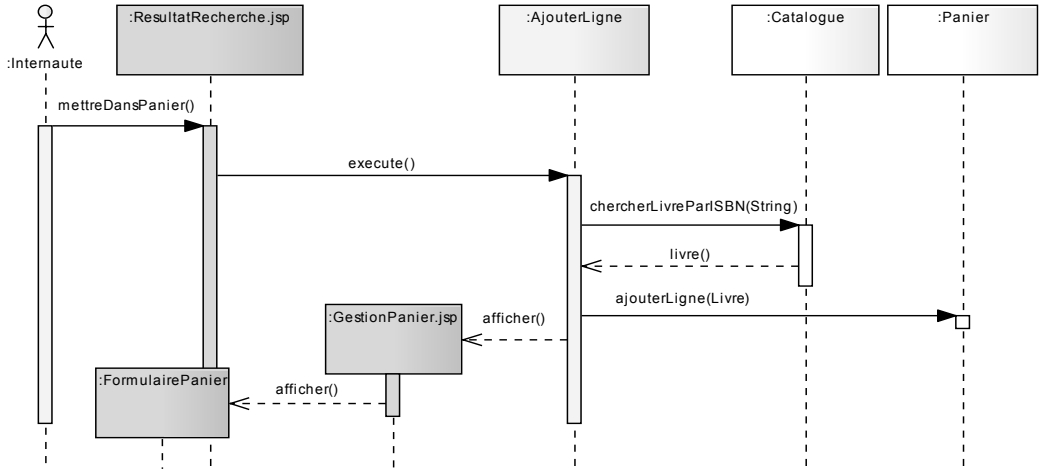


Figure B–35
Diagramme de séquence de la création d'une ligne du panier avec Struts

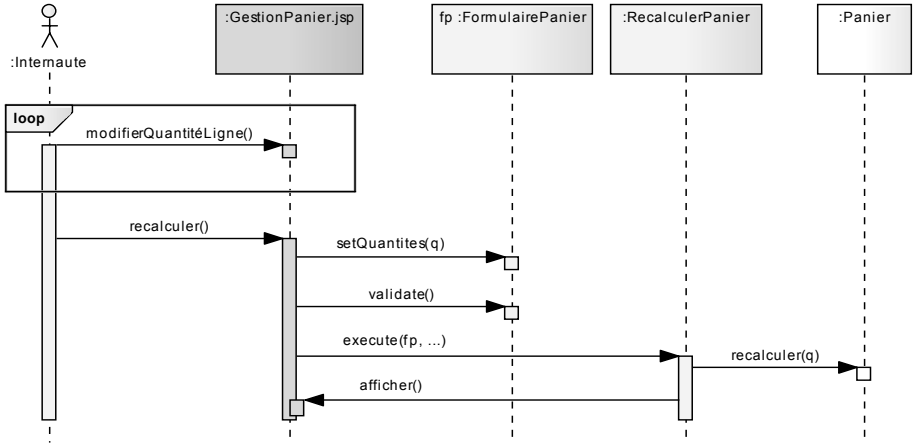


Figure B–36
Diagramme de séquence de la gestion du panier avec Struts

Diagramme de classes

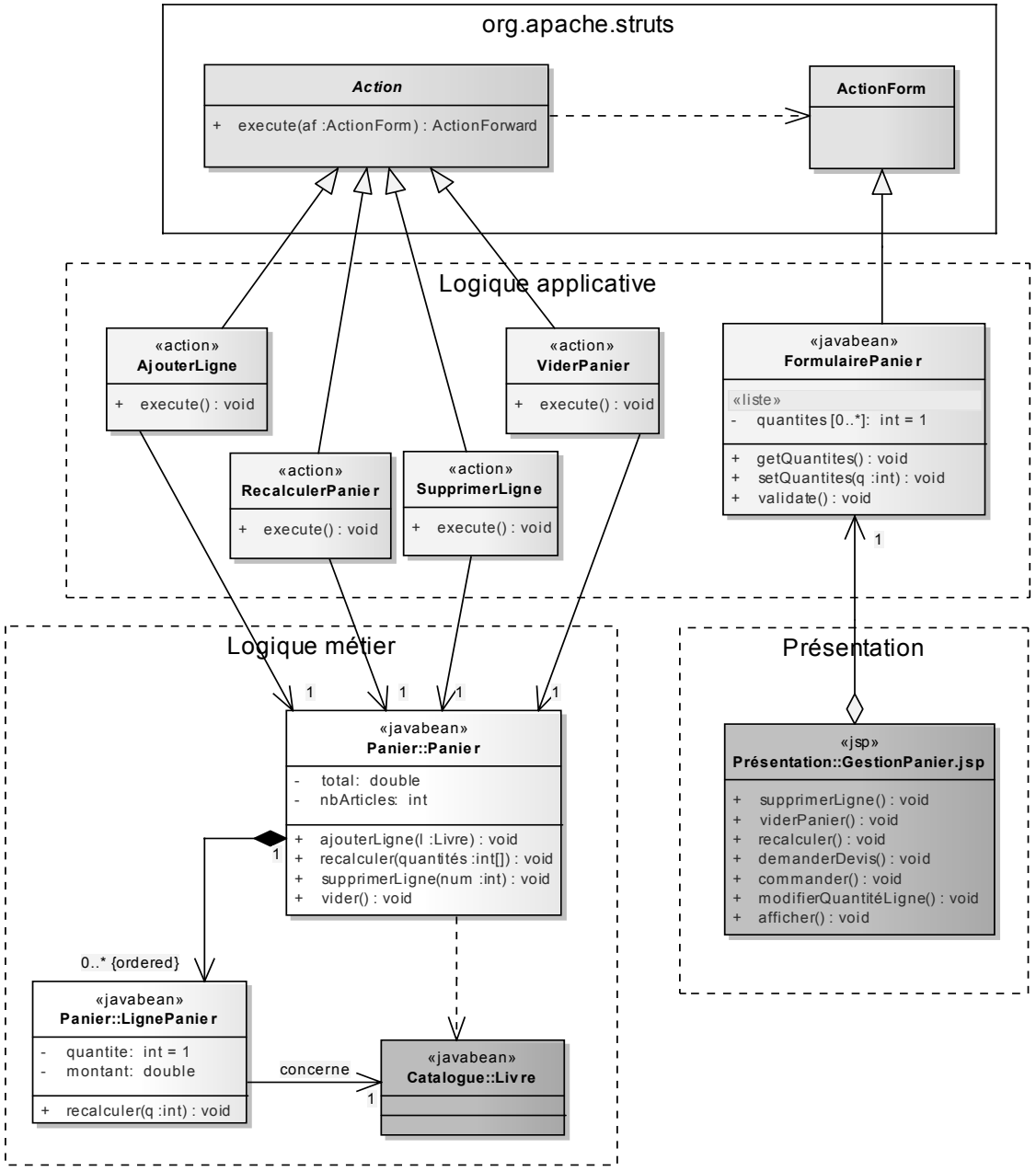


Figure B–37
Diagramme de classes de conception détaillée Struts de la Gestion du panier

Solution technique .NET

Diagrammes de séquence

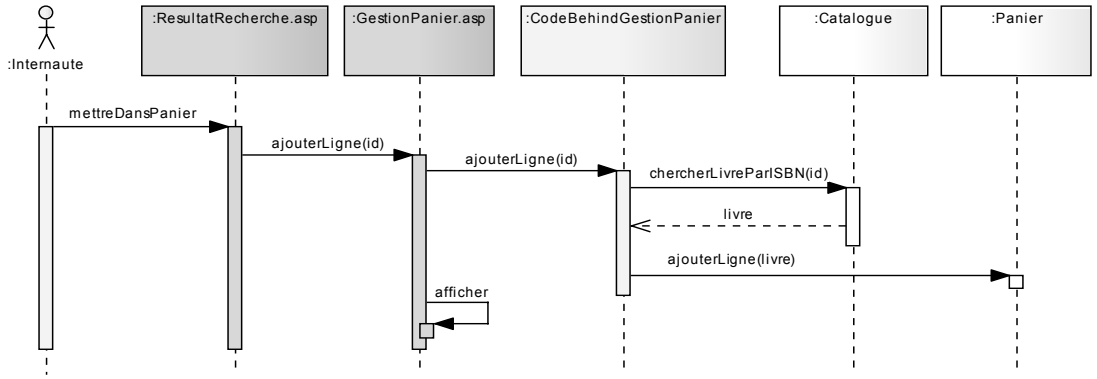


Figure B–38 Diagramme de séquence
de la création d'une ligne du panier avec .NET

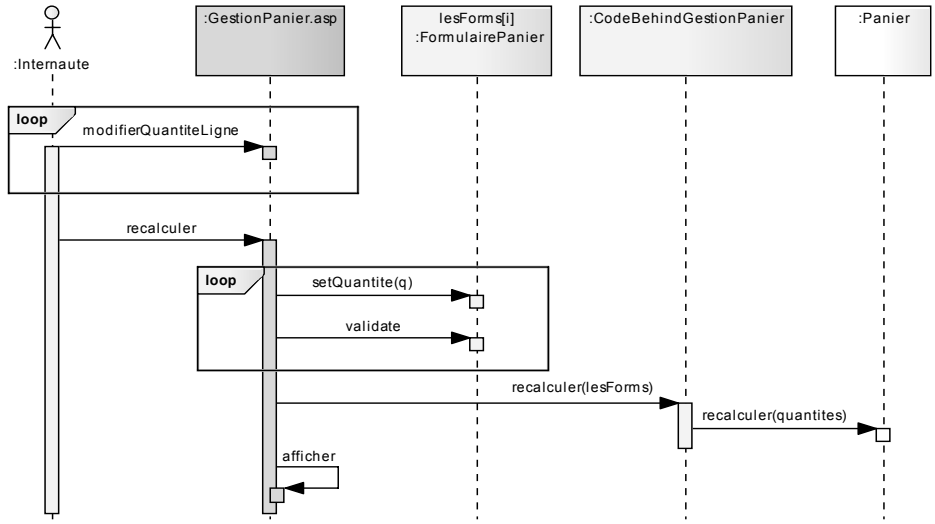


Figure B–39
Diagramme de séquence de la gestion du panier avec .NET

Diagramme de classes

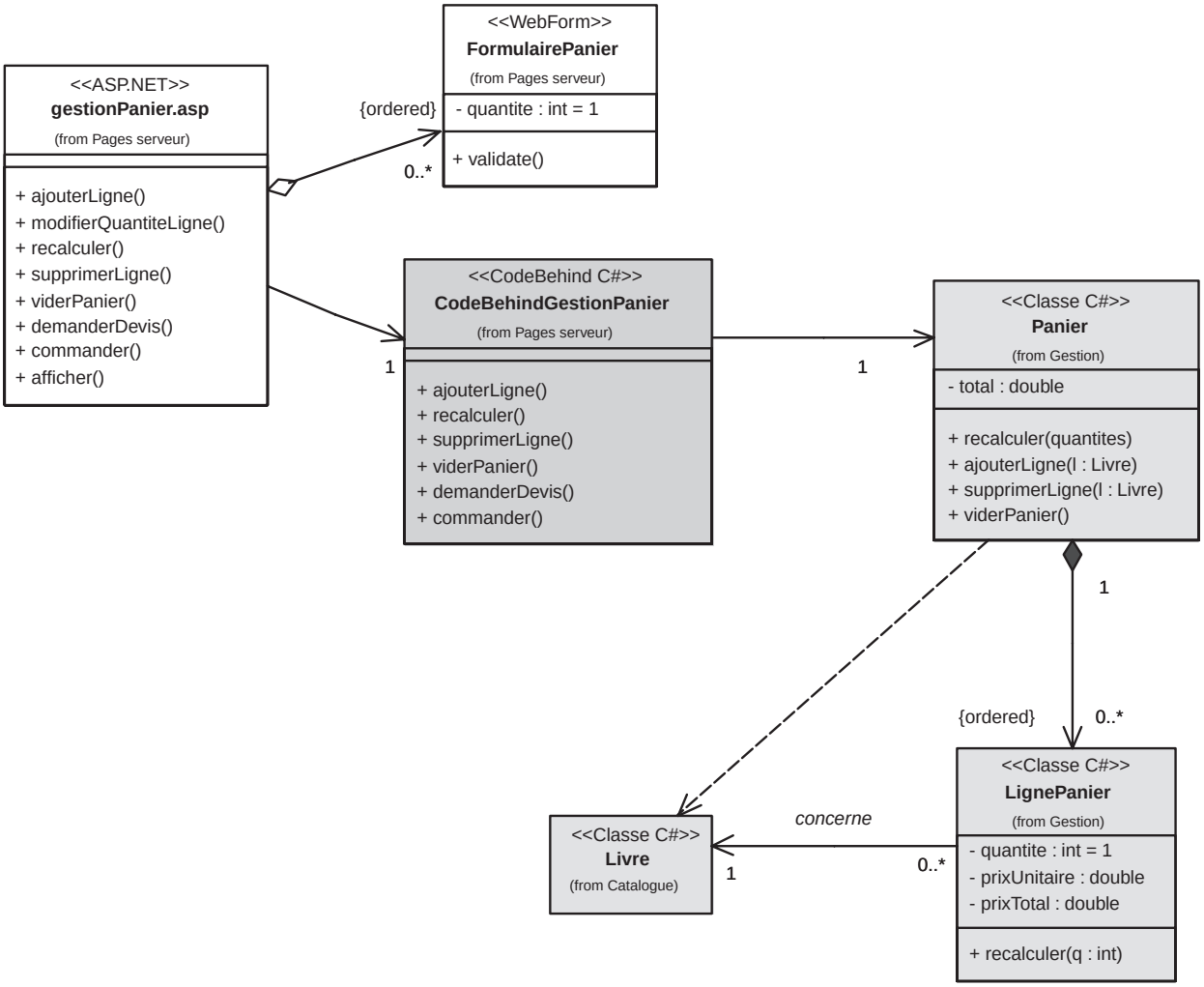


Figure B–40
Diagramme de classes de conception détaillée .NET de la gestion du panier

annexe

C

Modèle UML 1.4 de la première édition (réalisé avec Rational/Rose 2002)

Modèle des cas d'utilisation

Modèle du domaine

Modèle de navigation

Modèle de conception préliminaire

Modèle de conception détaillée

Modèle des cas d'utilisation

Structuration en packages

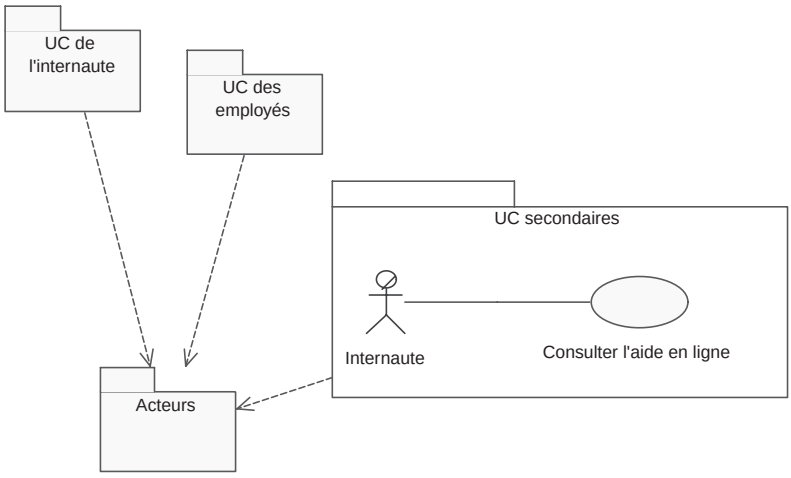


Figure C–1
Packages de la vue des cas d'utilisation

Package Acteurs

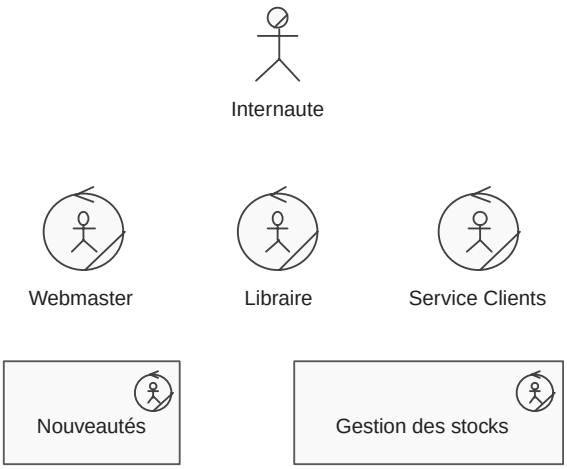


Figure C–2
Acteurs du site jeBouquine.com

Package des cas d'utilisation de l'internaute

Diagramme de cas d'utilisation

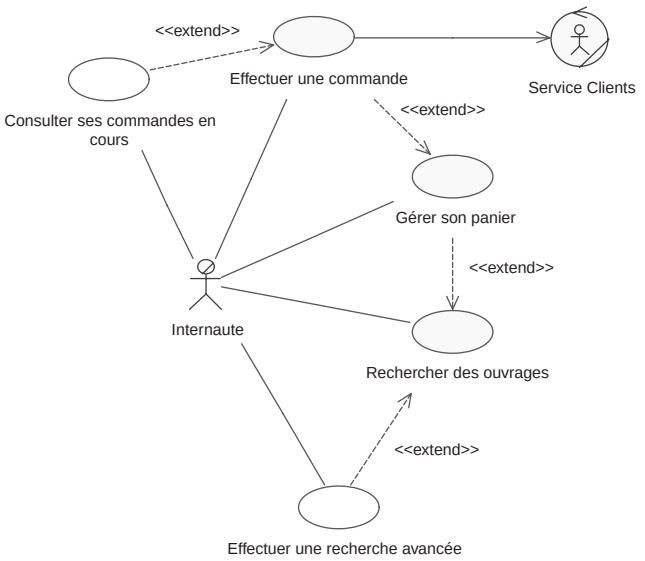


Figure C–3
Cas d'utilisation de l'internaute

Rechercher des ouvrages

Diagramme de Séquence Système (DSS)

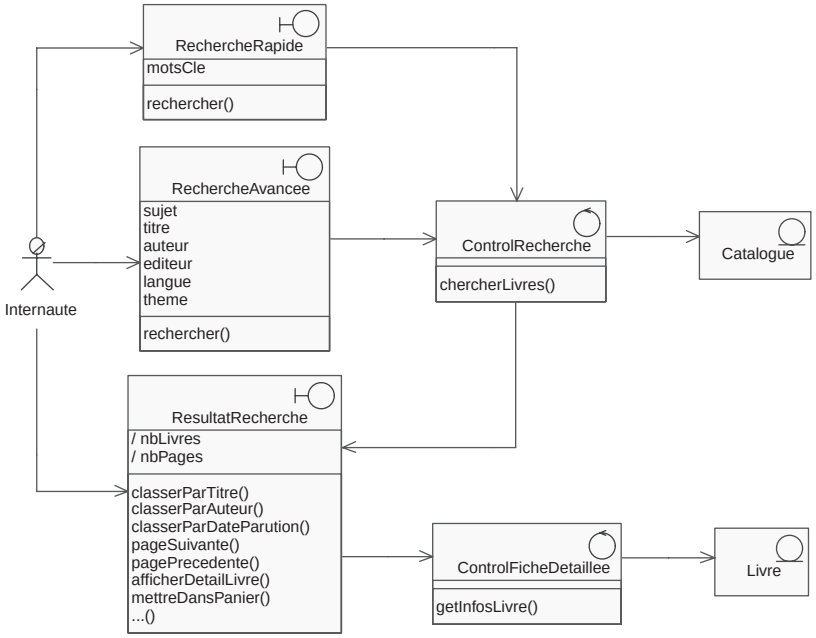


Figure C–4
Diagramme de séquence système de Rechercher des ouvrages

Diagramme de Classes Participantes (DCP)

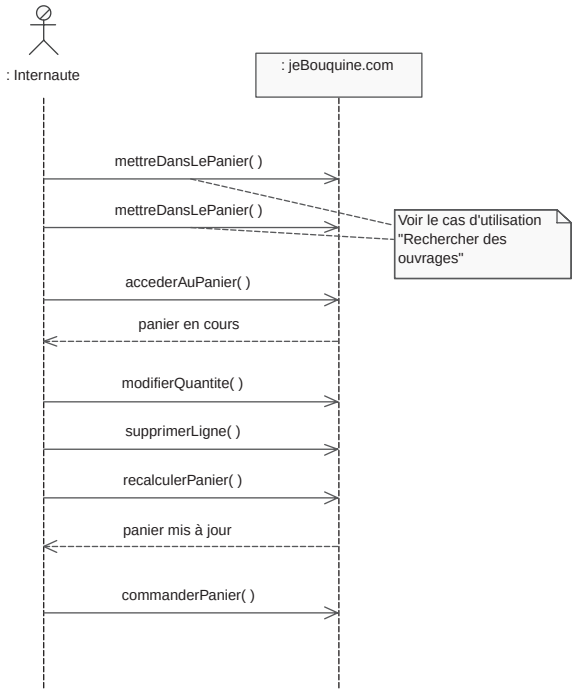


Figure C–5
DCP de Effectuer une recherche

Gérer son panier

Diagramme de Séquence Système (DSS)

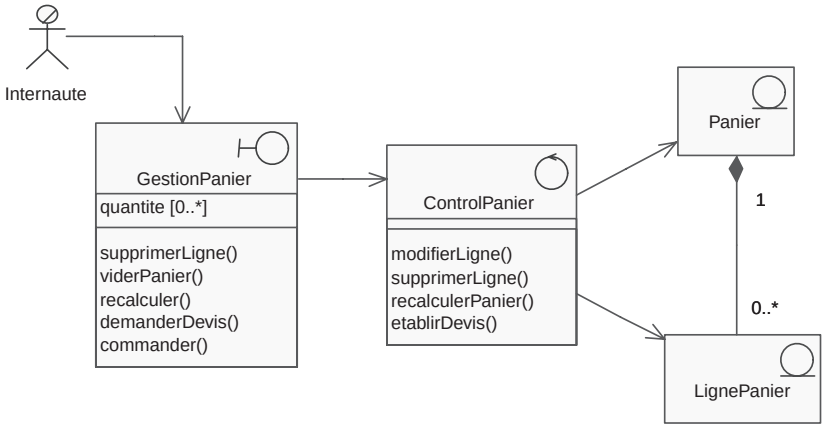


Figure C–6
Diagramme de séquence système de Gérer son panier

Diagramme de Classes Participantes (DCP)

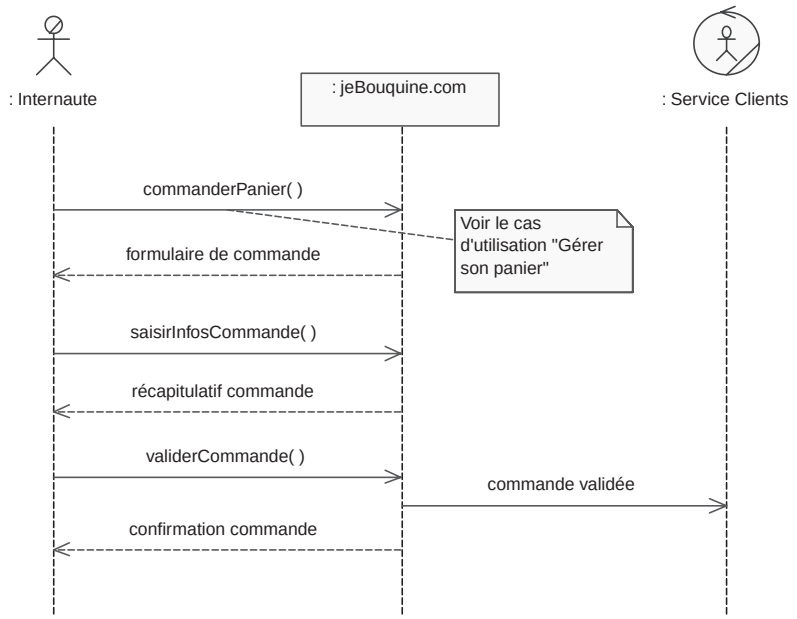


Figure C–7
DCP de Gérer son panier

Effectuer une commande

Diagramme de Séquence Système (DSS)

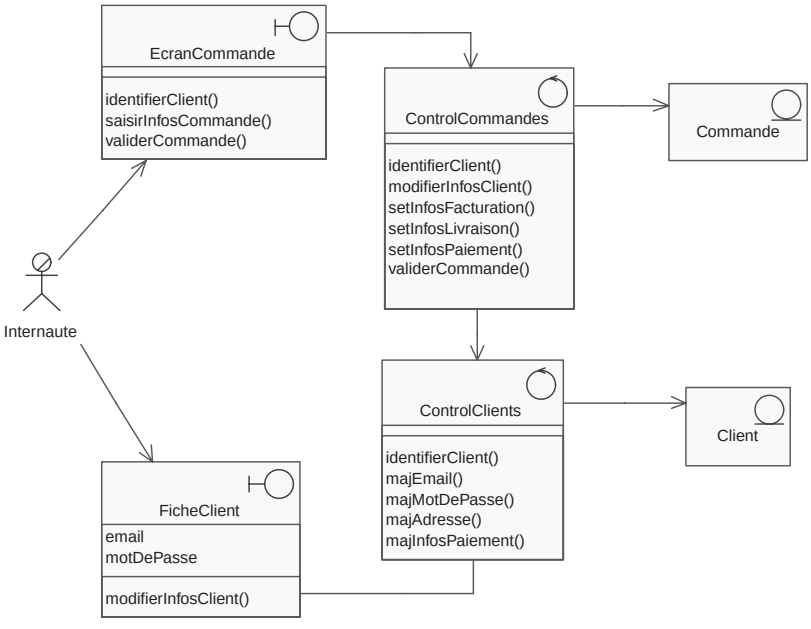


Figure C–8
Diagramme de séquence système de Effectuer une commande

Diagramme de Classes Participantes (DCP)

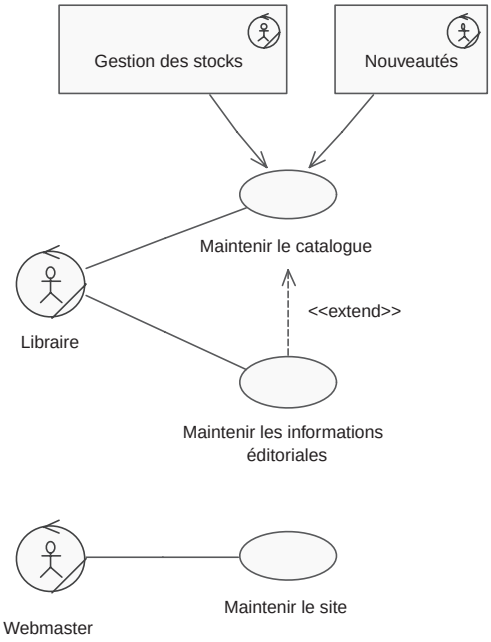


Figure C–9
DCP de Effectuer une commande

Package des cas d'utilisation des employés

Diagramme de cas d'utilisation

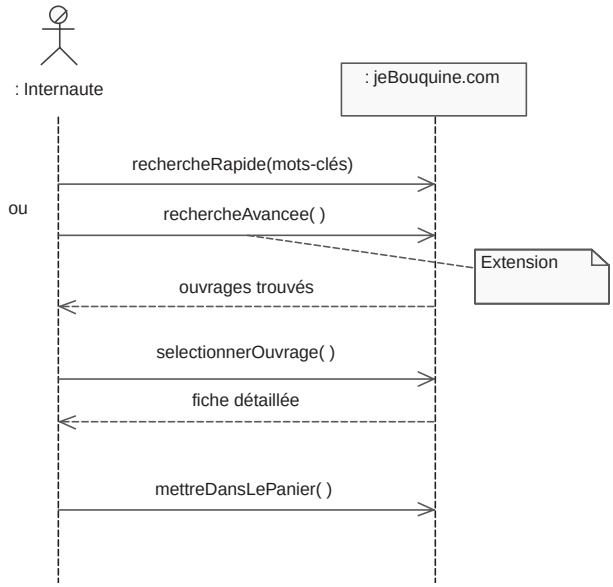


Figure C–10
Cas d'utilisation des employés

Maintenir le Catalogue

Diagramme de Séquence Système (DSS)

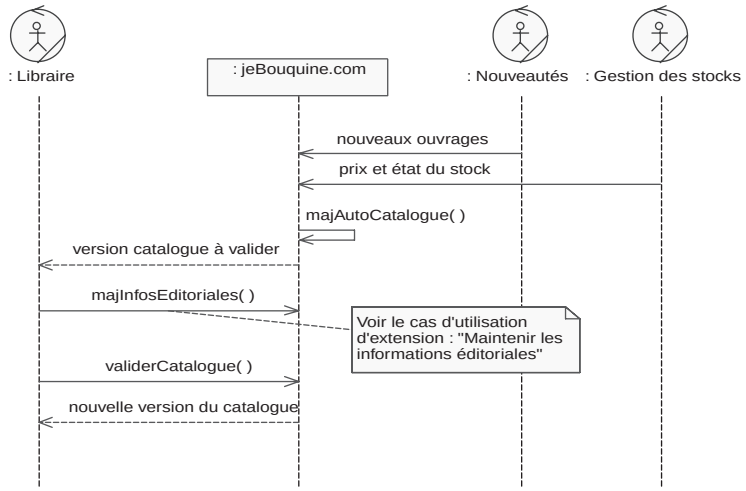


Figure C–11
Diagramme de séquence système de Maintenir le catalogue

Diagramme de Classes Participantes (DCP)

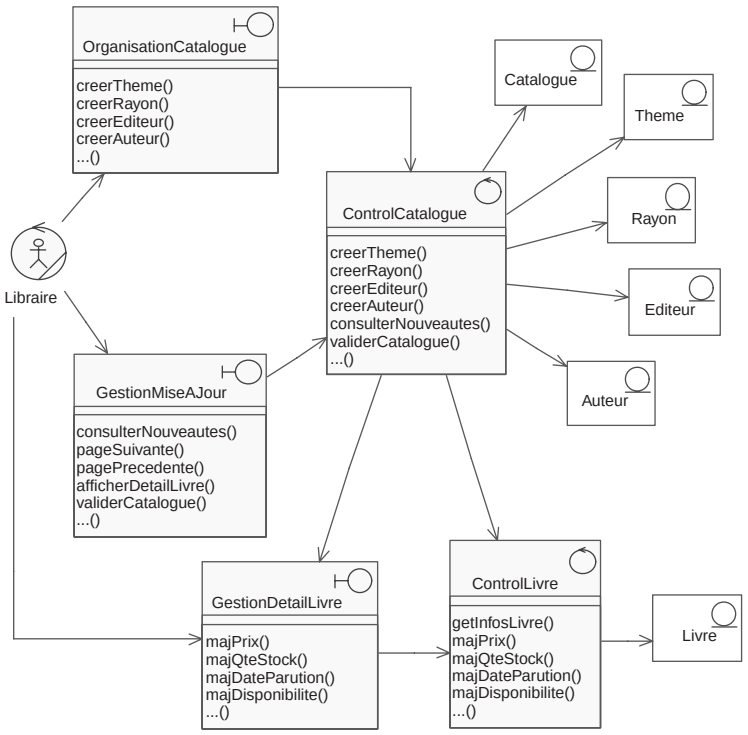


Figure C–12
DCP de Maintenir le Catalogue

Modèle du domaine

Structuration en packages

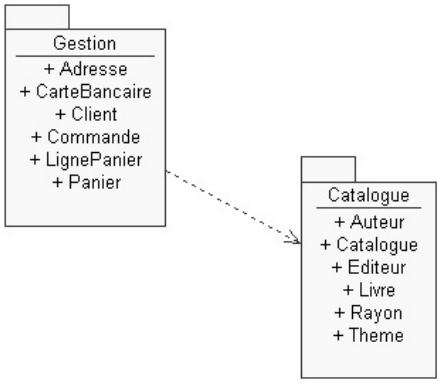


Figure C–13
Représentation synthétique des packages d'objets métier

Package Catalogue

Diagramme de classes

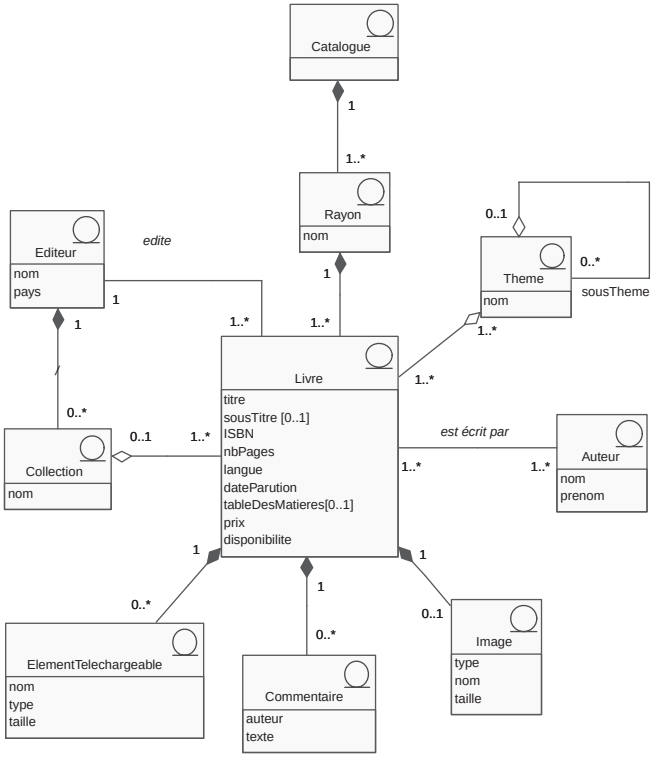


Figure C–14
Diagramme de classes du package Catalogue

Package Gestion

Diagramme de classes

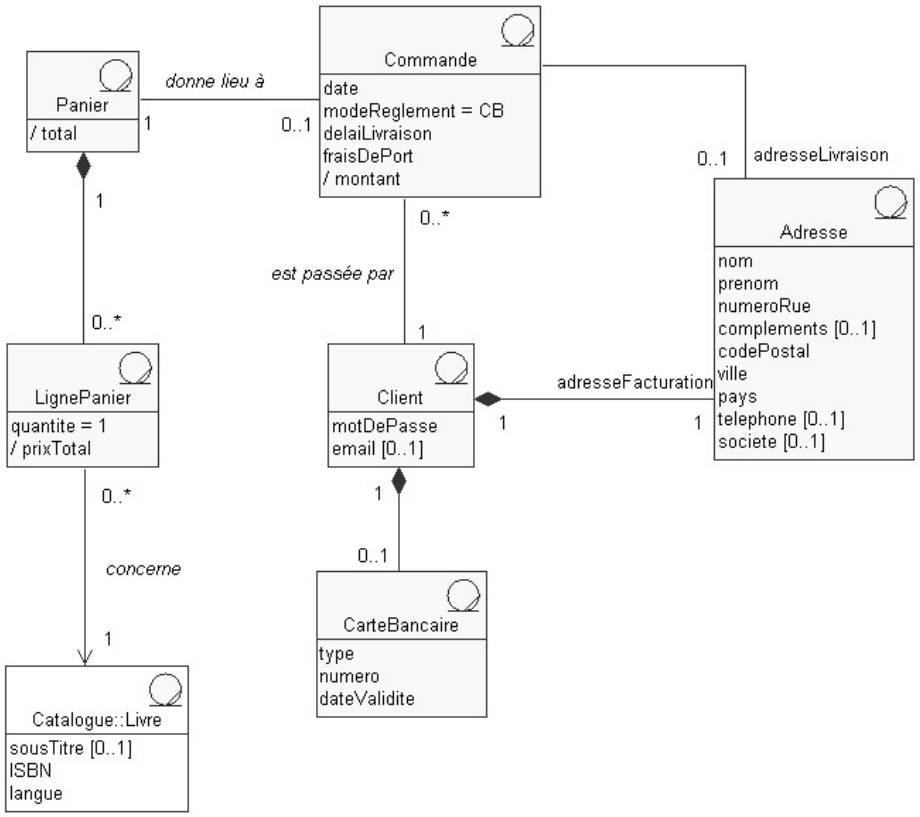


Figure C–15
Diagramme de classes complété du package Gestion

Modèle de navigation

Navigation de l'internaute

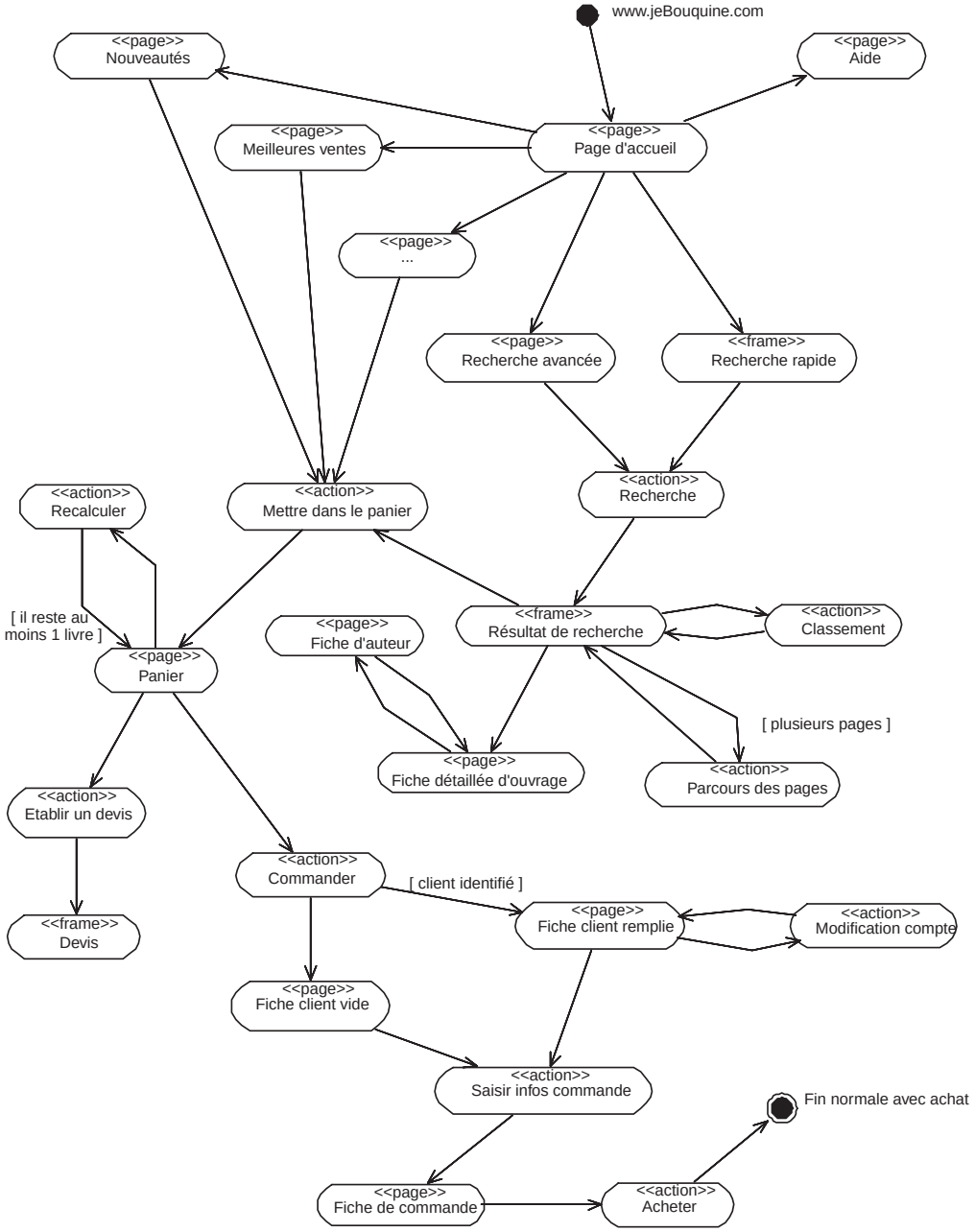


Figure C–16 Diagramme global de navigation de l'internaute

Modèle de conception préliminaire

Diagrammes d'interaction

Rechercher des ouvrages

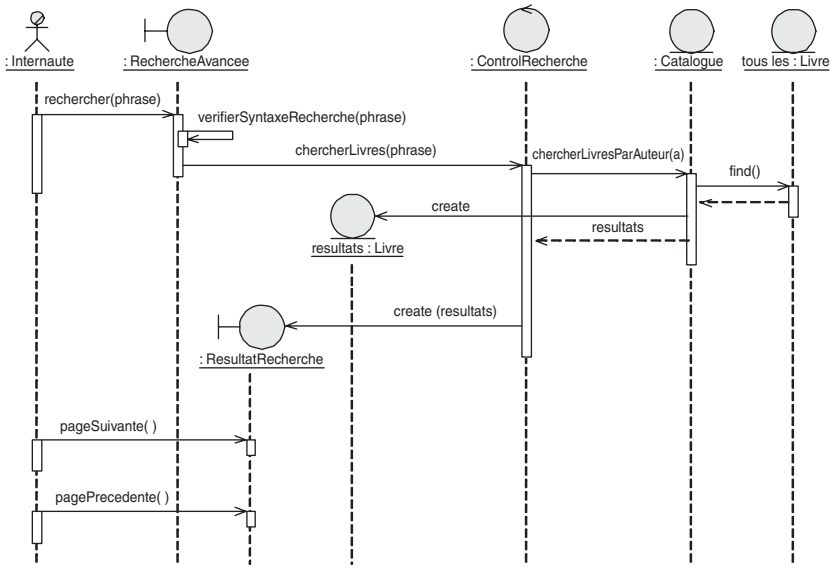


Figure C–17
Diagramme de séquence du scénario nominal de recherche avancée

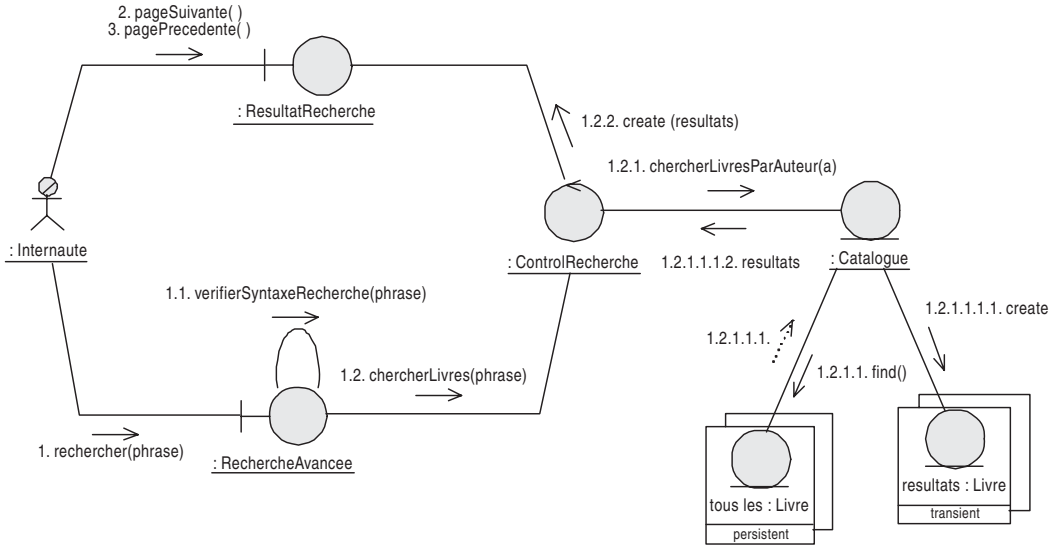


Figure C–18 Diagramme de collaboration du scénario nominal de recherche avancée

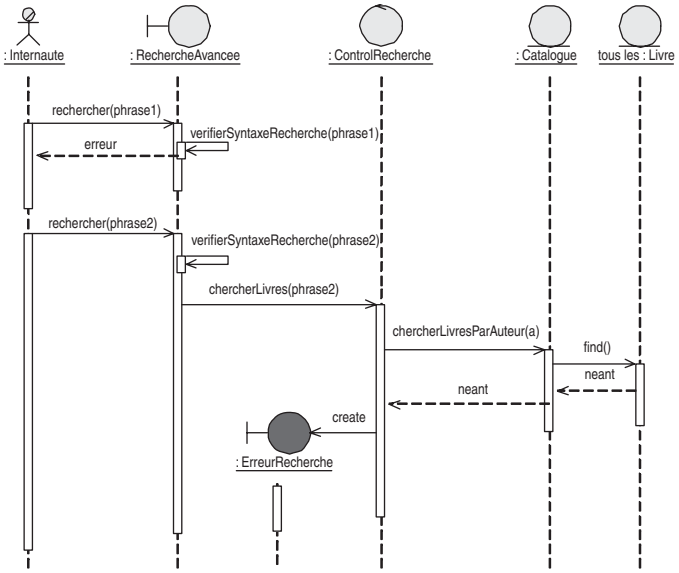


Figure C–19
Diagramme de séquence des scénarios d'erreur de recherche avancée

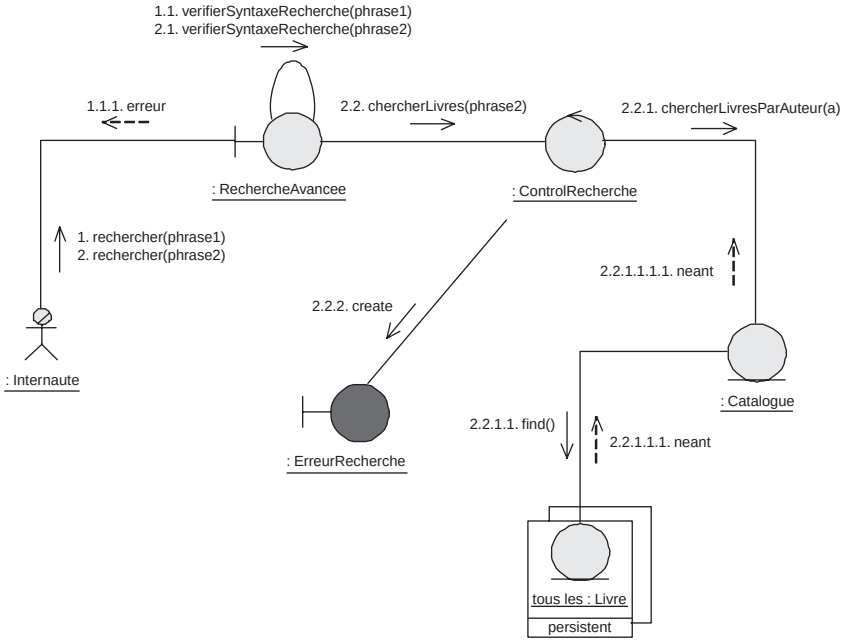


Figure C–20
Diagramme de collaboration des scénarios d'erreur de recherche avancée

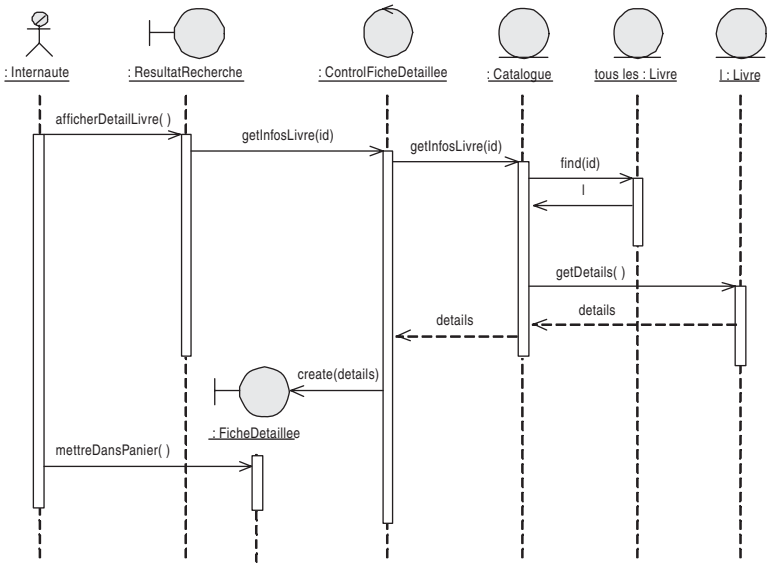


Figure C–21
Diagramme de séquence de la suite du scénario nominal de recherche avancée

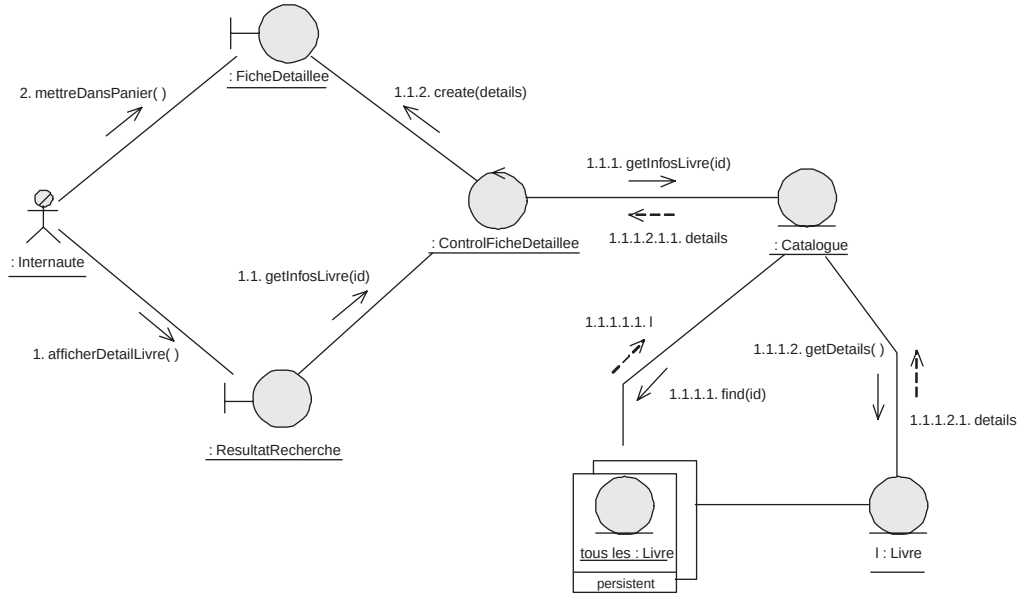


Figure C–22 Diagramme de collaboration de la suite du scénario nominal de recherche avancée

Gérer son panier

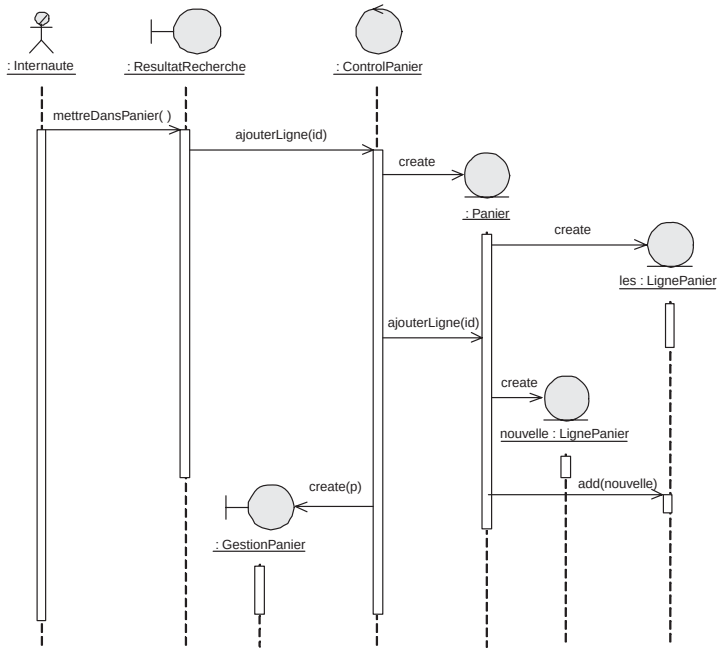


Figure C–23
Diagramme de séquence du scénario nominal de création de la première ligne du panier

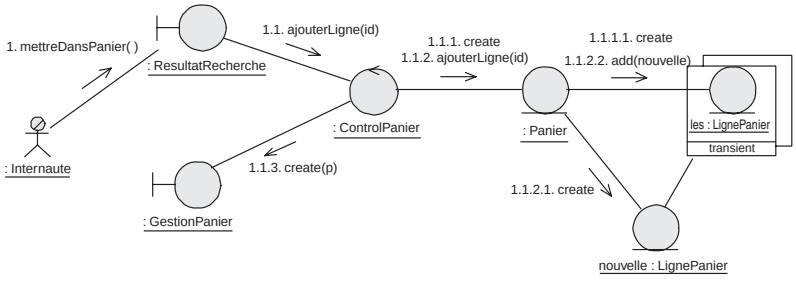


Figure C–24
Diagramme de collaboration du scénario nominal de création de la première ligne du panier

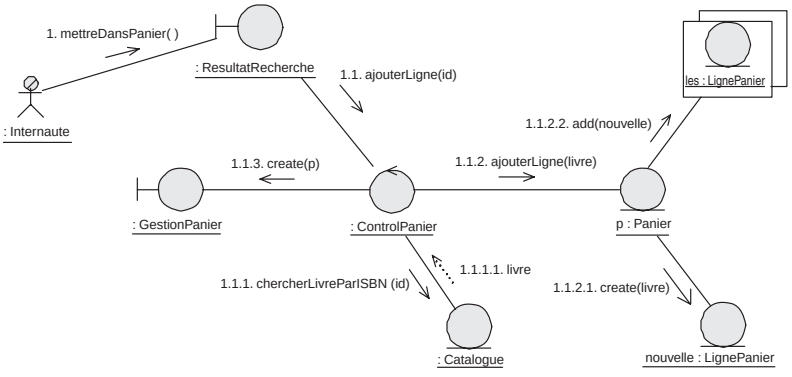


Figure C–25 Diagramme de collaboration détaillé de la création d'une ligne de panier

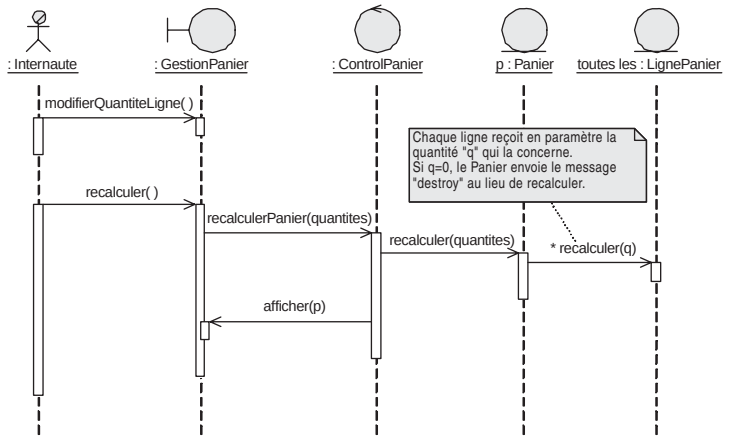


Figure C–26 Diagramme de séquence du scénario de recalcul du panier

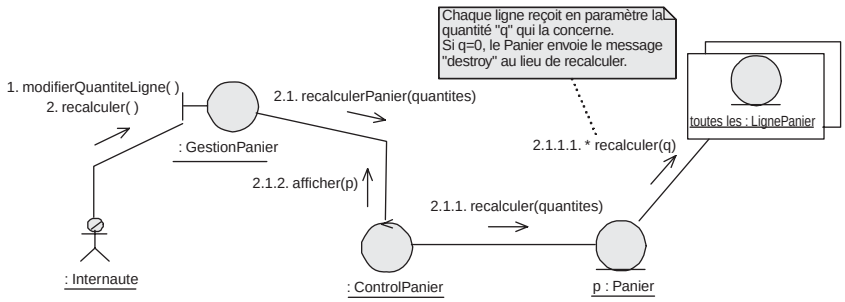


Figure C–27 Diagramme de collaboration du scénario de recalcul du panier

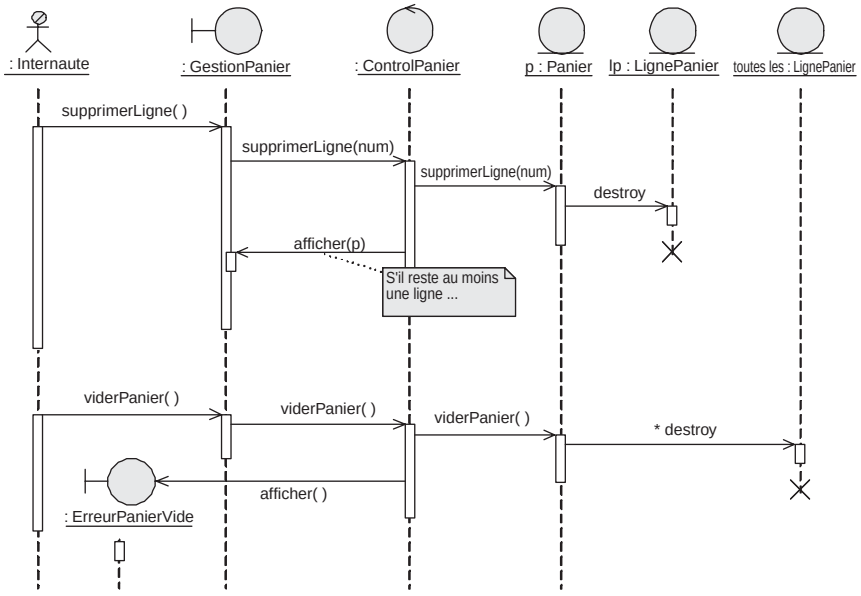


Figure C–28 Diagramme de séquence d'un scénario de vidage du panier

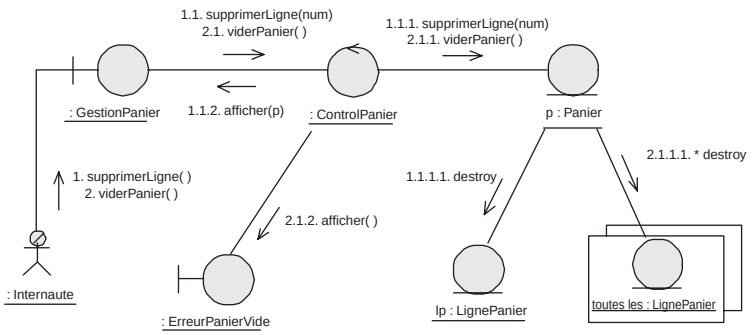


Figure C–29
Diagramme de collaboration d'un scénario de vidage du panier

Diagrammes de classes de conception préliminaire

Rechercher des ouvrages

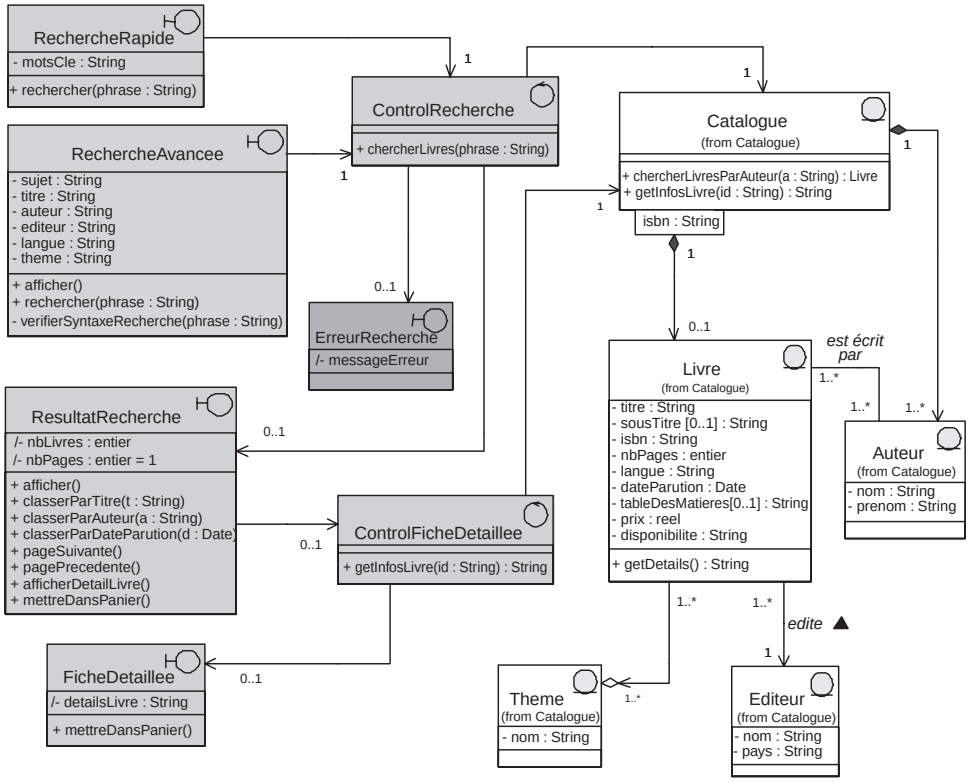


Figure C–30 Diagramme de classes de conception de Rechercher des ouvrages

Gérer son panier

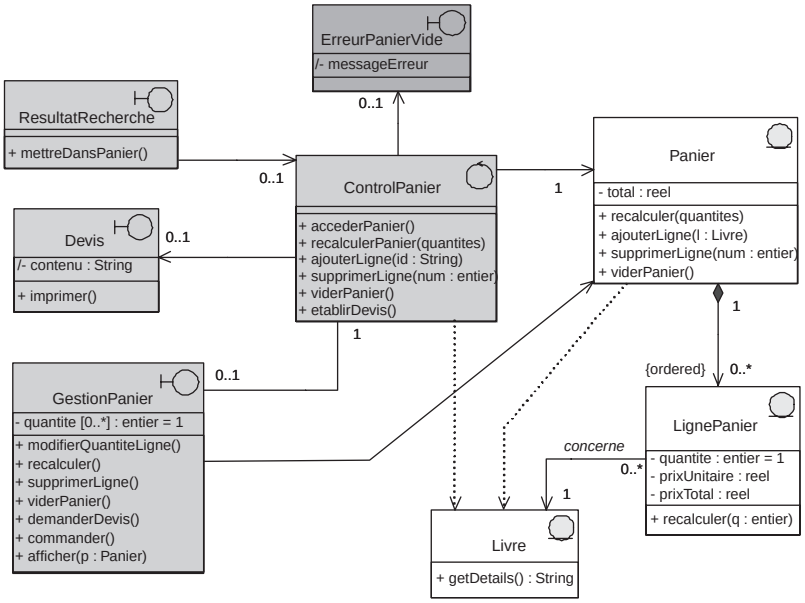


Figure C–31 Diagramme de classes de conception complété de Gérer son panier

Modèle de conception détaillée

Architecture logique

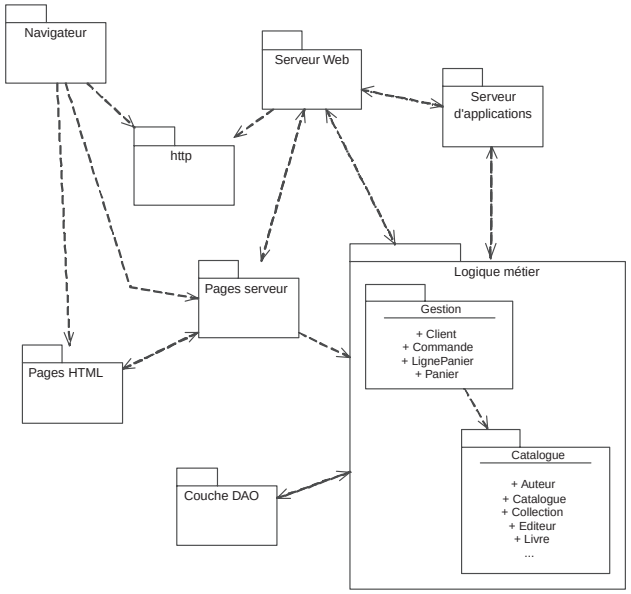


Figure C–32
Vue logique complète du pattern client web léger

Solution à base de scripts (PHP)

Diagramme d'interaction

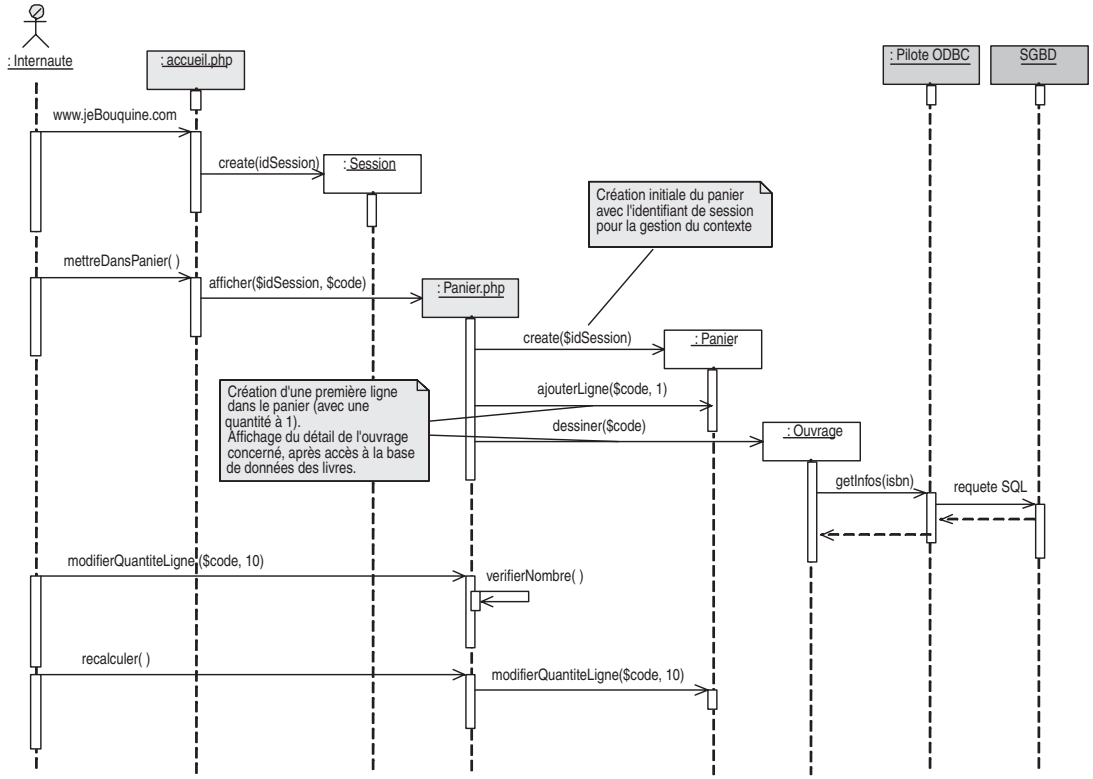


Figure C–33
Exemple de diagramme de séquence de conception détaillée PHP pour la gestion du panier

Diagramme de classes

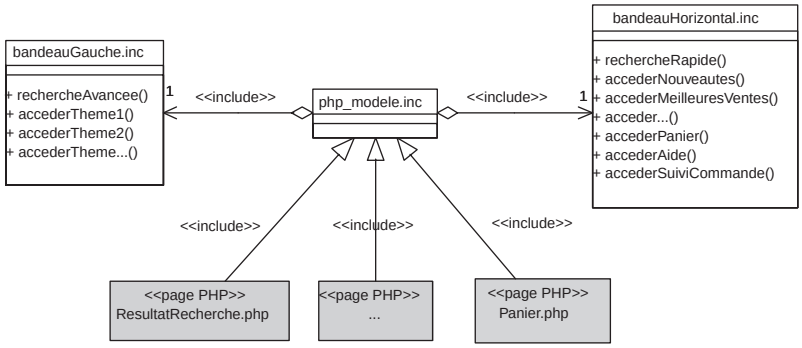


Figure C–34
Diagramme de classes des pages PHP

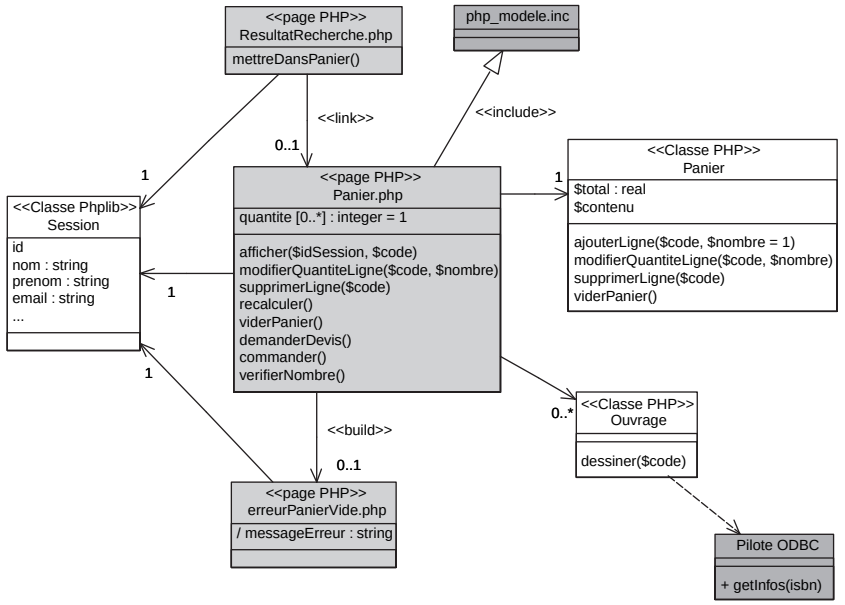


Figure C–35
Diagramme de classes de conception détaillée PHP de la gestion du panier

Solution technique J2EE (Struts)

Architecture logique

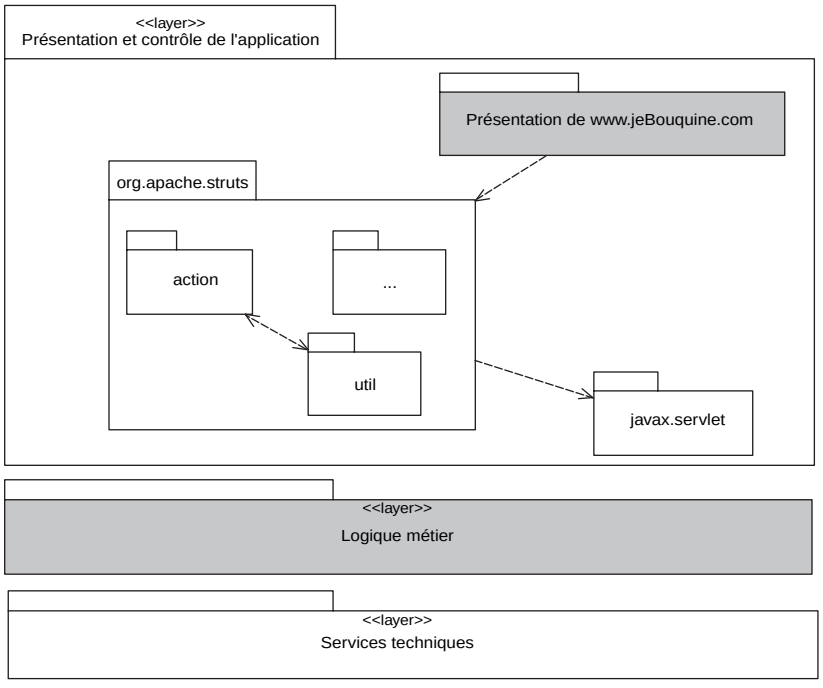


Figure C–36
Packages et couches notables reliés à Struts

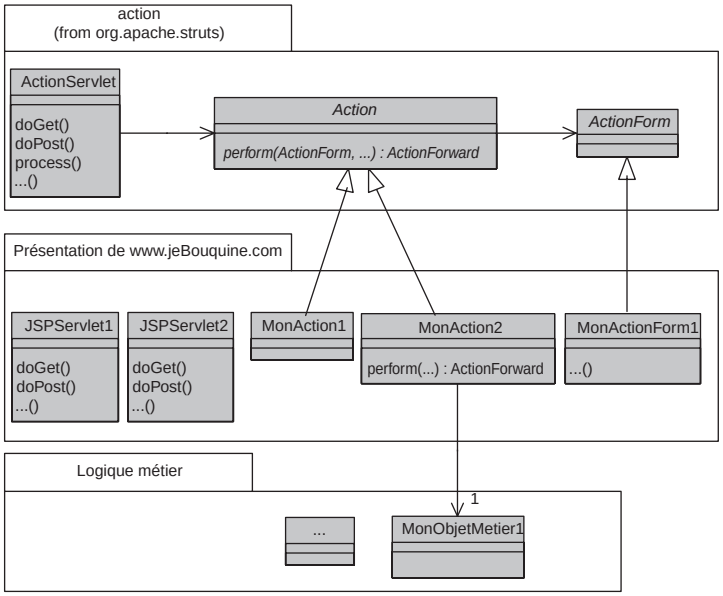


Figure C–37
Paradigme MVC dans Struts

Diagrammes d'interactions

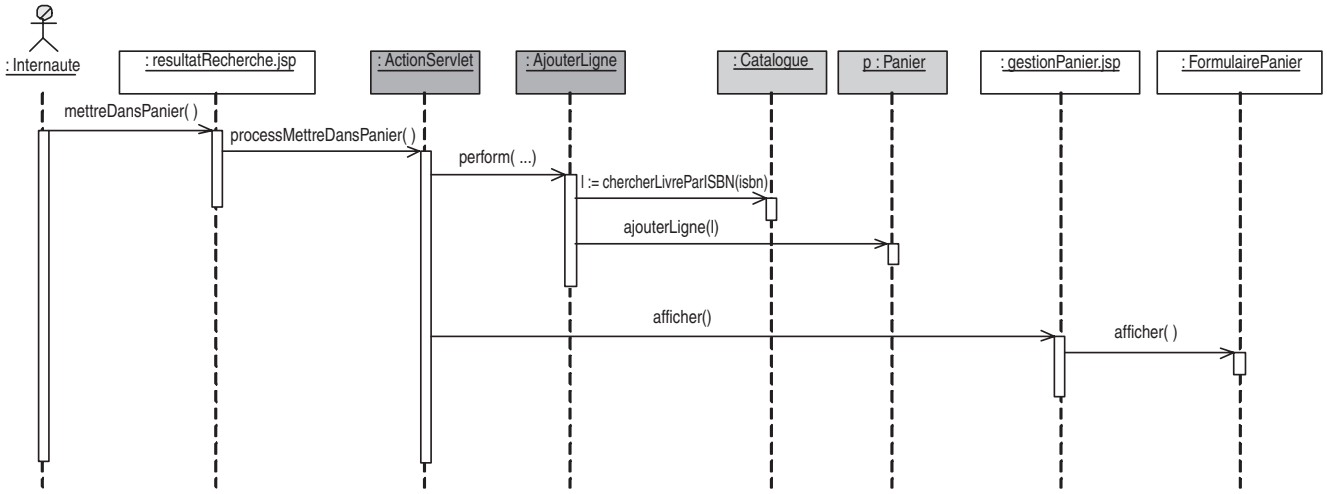


Figure C–38
Diagramme de séquence de la création d'une ligne du panier avec Struts

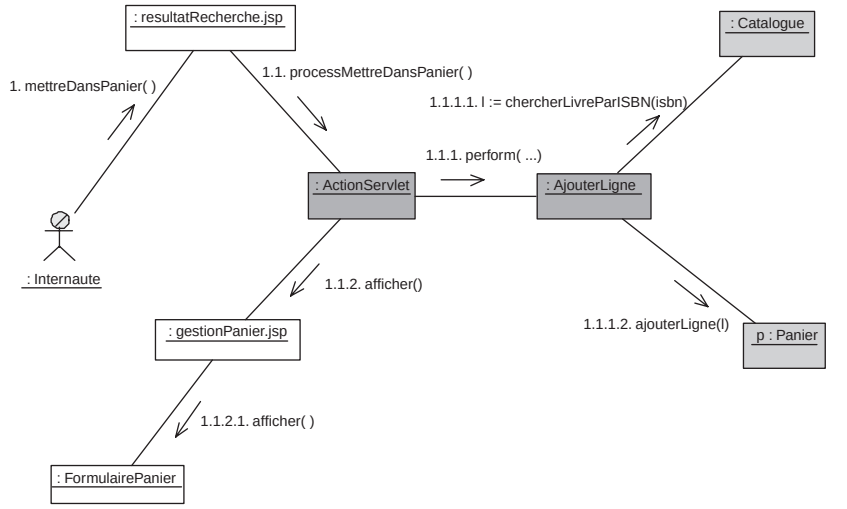


Figure C–39
Diagramme de collaboration de la création d'une ligne du panier avec Struts

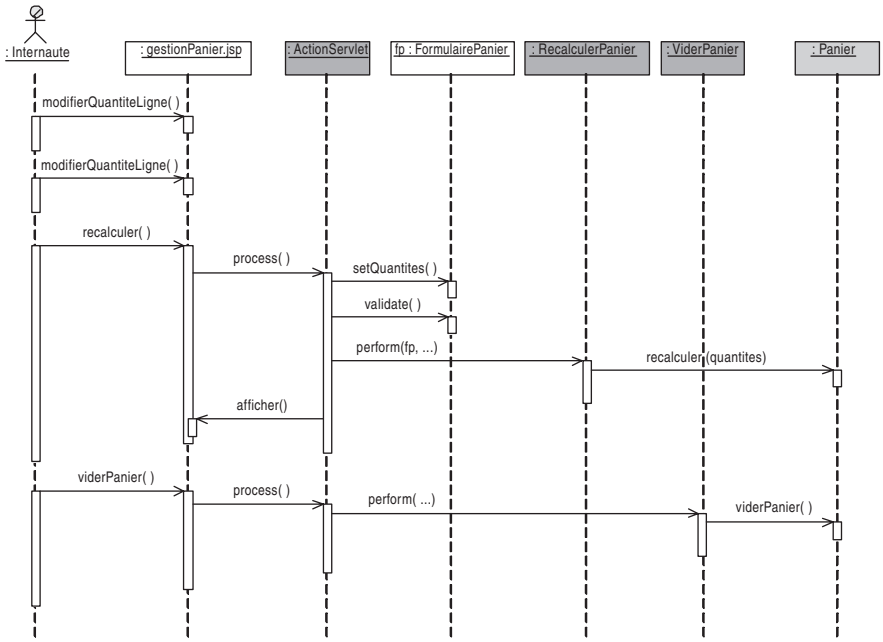


Figure C–40
Diagramme de séquence de la gestion du panier avec Struts

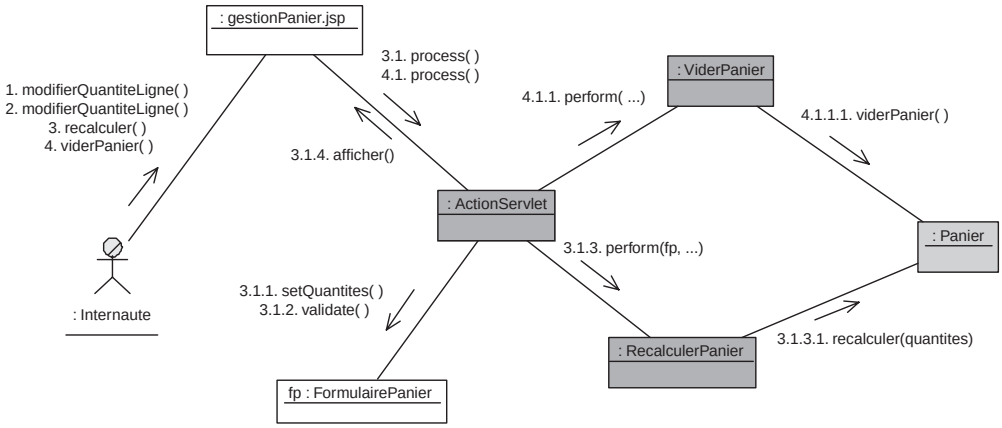


Figure C–41 Diagramme de collaboration de la gestion du panier avec Struts

Diagramme de classes

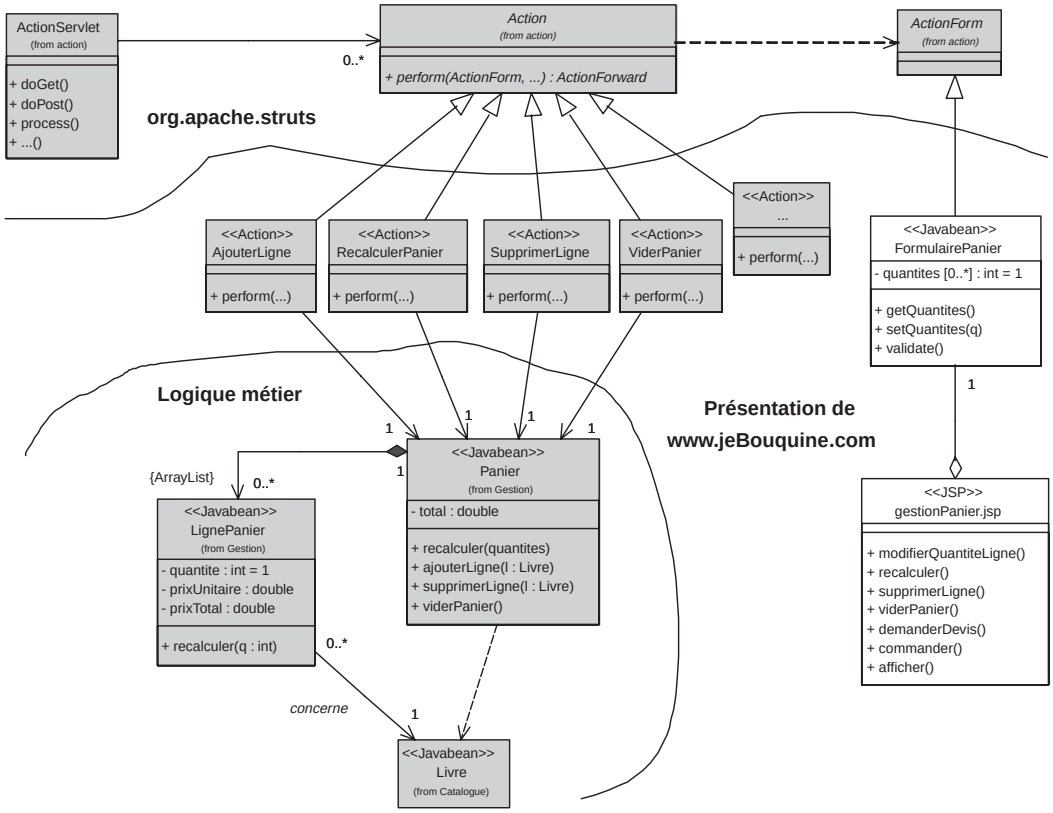


Figure C–42 Diagramme de classes de conception détaillée Struts de la gestion du panier

Solution technique .NET

Diagrammes d'interactions

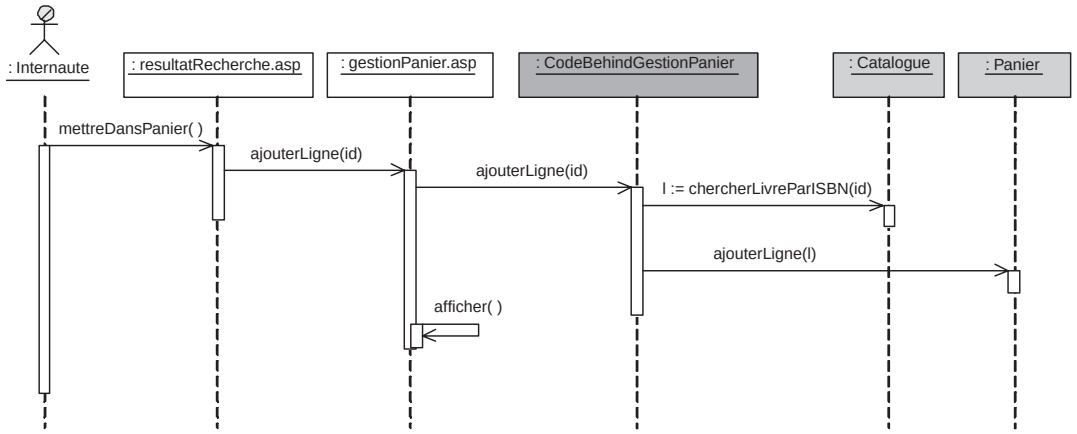


Figure C–43
Diagramme de séquence de la création d'une ligne du panier avec .NET

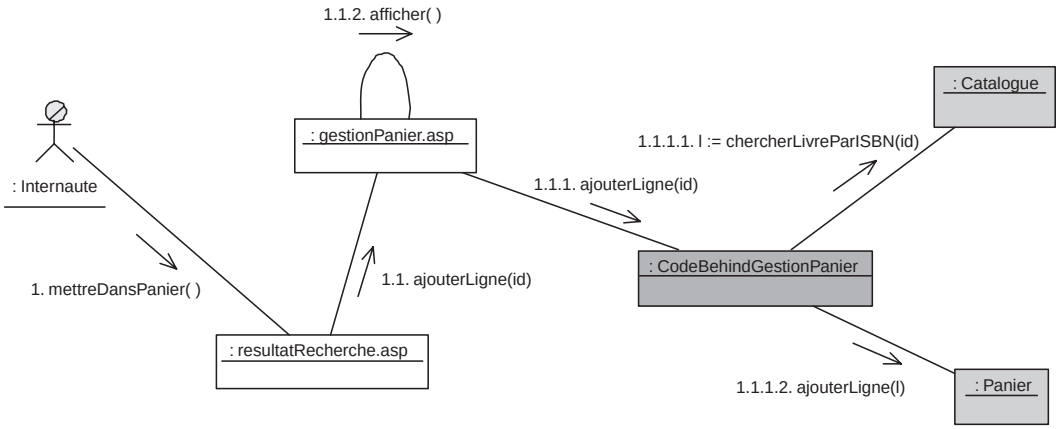


Figure C–44
Diagramme de collaboration de la création d'une ligne du panier avec .NET

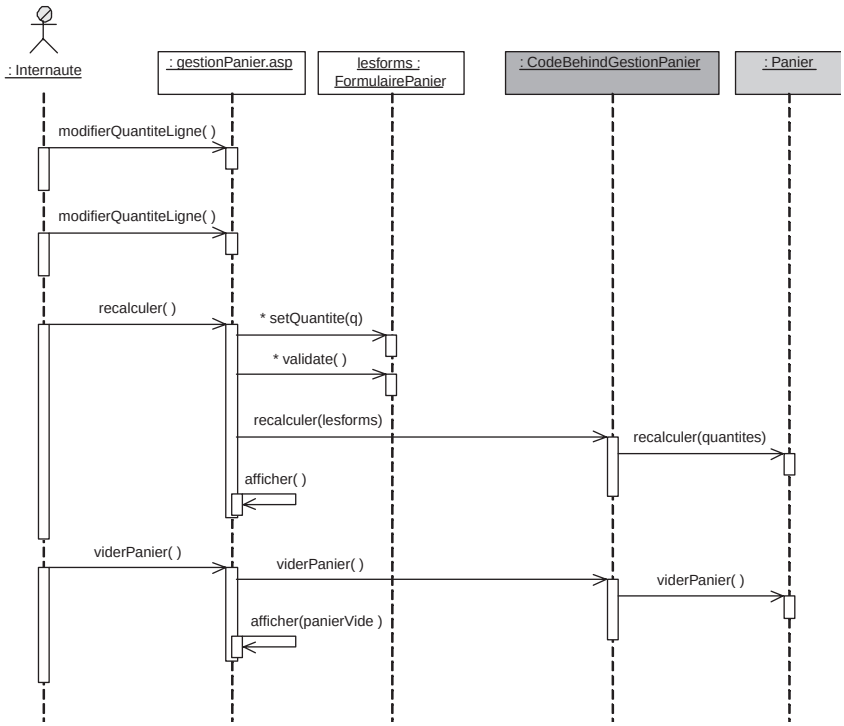


Figure C–45
Diagramme de séquence de la gestion du panier avec .NET

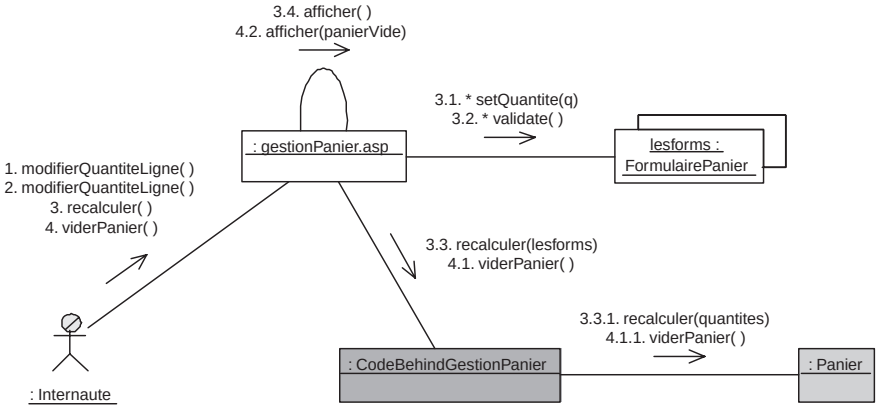


Figure C–46
Diagramme de collaboration de la gestion du panier avec .NET

Diagramme de classes

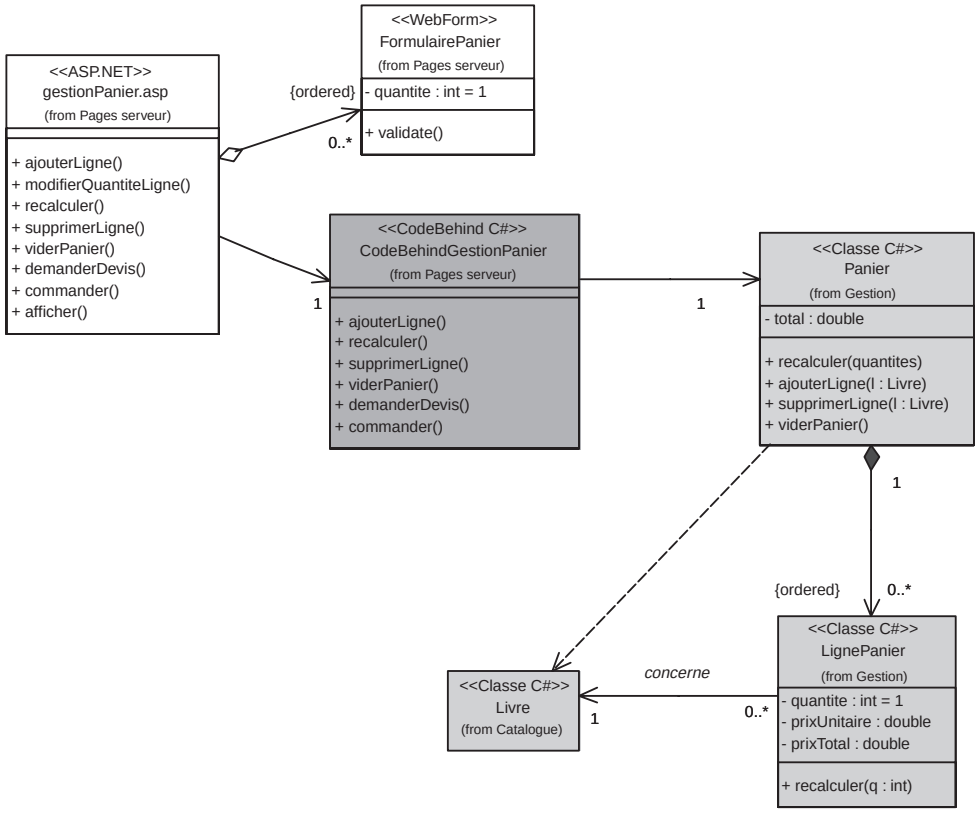


Figure C–47
Diagramme de classes de conception détaillée .NET de la gestion du panier

Index

Programmez intelligent avec les Cahiers du Programmeur

UML 2

4e édition

Ce cahier montre à tous les programmeurs combien UML est un outil simple et universel : nullement réservé aux applications Java, C++ ou C#, il s'applique parfaitement à des applications web telles que des sites marchands en PHP 5, dont la complexité en fait des candidats naturels à la modélisation.

Du cahier des charges au code, ce livre vous offrira les meilleures pratiques de modélisation avec UML 2 sous la forme d'une étude de cas complète. Toutes les étapes d'analyse et conception sont décrites, abondamment illustrées et expliquées, à travers une démarche située à mi-chemin entre processus lourd et démarche agile. *Cette quatrième édition traite de la gestion des exigences avec l'outil UML Enterprise Architect (EA).*

Consultant senior et formateur dans le groupe Valtech depuis 1995, **Pascal Roques** a plus de vingt ans d'expérience dans la modélisation de systèmes complexes, d'abord avec les techniques d'analyse structurée (SADT…), puis avec les approches objet (OMT, UP…). Il travaille à promouvoir l'utilisation d'UML dans des domaines variés (aéronautique, espace, banques, etc.) et est actuellement responsable de l'ensemble des formations catalogue Valtech Training sur le thème « Modélisation avec UML ».

Pascal Roques est l'auteur des livres *UML 2 par la pratique*, *UML 2 en action*, et du *Mémento UML* chez Eyrolles.

Sommaire

Conception couverture : Nordcompo