**République Algérienne Démocratique et Populaire**

Ministère de l'enseignement supérieur et de la recherche scientifique

**Université El Hadj Lakhdar - Batna**

Faculté des sciences de l'ingénieur

Département d'informatique

Mémoire

**présenté en vu de l'obtention du diplôme de**

Magister en Informatique

Option : Informatique industrielle

# Intégration des moyens de modification dynamique des contenus sur le WEB

Samir ADOUANE

**Jury :**

| | | |
|---|---|---|
| Dr. S. Chikhi | Université de Constantine | Président |
| Pr. M. Benmohammed | Université de Constantine | Rapporteur |
| Dr. A. Zidani | Université de Batna | Examinateur |
| Dr. S. Zidet | Université de Batna | Examinateur |

**Remerciements**

Je remercie le Pr Mohamed BENMOHAMMED, Maître de conférence à l'université de Constantine d'avoir accepté de prendre en charge la direction et le suivi de ce travail, pour sa patience, son écoute et sa compréhension.

Ma gratitude s'adresse tout particulièrement aux membres du jury, qui ont accepté de juger ce modeste travail.

Je ne permette pas d'oublier les collègues à ECS Schindler où la plupart des applications ont été développées et testées ; Driss Elkhalouki, Bertrand PARMENTIER, Fabrice TOMACITE et en générale tout le personnel d'ECS-Shindler Mulhouse.

Je tiens aussi à présenter mes remerciements à tous ceux qui m'ont, de près ou de loin, permis de mener à bien ce travail, qui s'est révélé une expérience de grande ampleur et à la hauteur de ses promesses.

Je remercie ma famille pour son soutien continu et son encouragement pour mener à terme ce travail.

## RESUME

Le monde de conception des applications est enrichi chaque jour par de nouveaux designs architecturaux, répondant à des problèmes d'implémentation récurrents, la majorité de ces solutions tournent autours de la programmation orientée objet qui facilite leur intégration et mise en œuvre.

Dans ce mémoire nous proposons une approche basée sur le modèle MVC que nous appellerons « les délégués des packages », cette approche vise à faciliter et à accélérer le processus de développement web. Elle tente d'apporter des éléments pratiques, facilitant le déploiement du modèle MVC dans un contexte de la programmation orientée objet.



**Mots-clés :** Modélisation, Méthode de conception, UML, Design Patterns, XML, XMI, application web, Entrepôt de données.



## ABSTRACT

The world of applications design is enriched every day by new architectural designs that answer recurring implementation problems. The majority of these solutions address the Object-Oriented Programming (OOP) which facilitates their integration and implementation.

In this report we propose an approach based on the MVC model which we shall call "packages delegates", this approach aims to facilitate and to accelerate the process of web development. moreover it tries to come up with practical elements, facilitating the deployment of the MVC model in an OOP context.



**Keywords :** Modeling, Conception method, UML, Design Patterns, XML, Web application, Data Warehouse.

# Liste des Figures


**Figure 1.1** Un processus itératif [Kruchten00] ..... 08
**Figure 1.2** Le modèle, un artefact Central du processus De développement . ... 10
**Figure 1.3** Traitement d'une page statique ..... ... 15
**Figure 1.4** Traitement des pages dynamiques ..... ... 16
**Figure 1.5** Accès à une base de données ..... ... 18
**Figure 1.6** Problèmes de conception des applications Web ..... ... 24
**Figure 1.7** Une approche centralisée de la validation des entrées ..... ... 26
**Figure 1.8** Stratégie de validation des entrées : contraignez, rejetez et assainissez la saisie de la validation des entrées ..... ... 27
**Figure 2.1** Séparation des données et de la présentation avec XML ..... ... 32
**Figure 2.2** Exemple de fichier XML avec sa DTD ..... ... 37
**Figure 2.3** Transformation avec XSL ..... ... 42
**Figure 3.1** Historique d'UML ..... ... 51
**Figure 3.2** Décomposition fonctionnelle ..... ... 54
**Figure 3.3** Décomposition Objet ..... ... 54
**Figure 3.4** Communication des objets par des messages ..... ... 56
**Figure 3.5** Les diagrammes UML par axe de modélisation ..... ... 56
**Figure 3.6** Héritage des diagrammes UML ..... ... 56
**Figure 3.7** Extrait du métamodèle. Caractéristiques de la classe ..... ... 57
**Figure 3.8** Un ou plusieurs automates peuvent être associés à chaque classe du modèle ..... ... 59
**Figure 3.9** Niveaux d'abstraction de la méta-modélisation ..... ... 67
**Figure 3.10** Package EMOF (MOF 2.0) ..... ... 68
**Figure 3.11** Le design pattern Singleton ..... ... 72
**Figure 3.12** Le design pattern Decorator ..... ... 72
**Figure 3.13** Le design pattern Observer ..... ... 73
**Figure 4.1** Structure de classe MVC ..... ... 76
**Figure 4.2** Comportement du modèle passif ..... ... 77
**Figure 4.3** Utilisation de l'observateur pour dissocier le modèle de la vue dans le modèle actif ..... ... 78
**Figure 4.4** Comportement du modèle actif ..... ... 79
**Figure 4.5** Modèle MVC avec les délégués des Packages ..... ... 82
**Figure 5.1** Sources de données MERLIN ..... ... 90
**Figure 5.2** Diagramme des cas d'utilisations ..... ... 91
**Figure 5.3** Détail du cas d'utilisation : CommandeVente ..... ... 91
**Figure 5.4** Présentation dynamique avec le diagramme de séquence ..... ... 92
**Figure 5.5** Diagramme des packages MERLIN 0..... ... 94
**Figure 5.6** Diagramme de classe partiel du package CONTROLE (MVC avec les délégués des packages) ..... ... 94
**Figure 5.7** Diagramme de classe partiel du package MODELE (MVC avec les délégués des packages) ..... ... 95
**Figure 5.8** Diagramme de classe partiel du package VUE (MVC avec les délégués des packages) ..... ... 95
**Figure 5.9** Base de données classiques et Data Warehouse ..... ... 97
**Figure 5.10** Modèle physique de donnée Modèle en étoile ..... ... 98
**Figure 5.11** Modèle physique de donnée Modèle en flocon "neige" ..... ... 98
**Figure 5.12** Plusieurs tables de fait dans un modèle en étoile ..... ... 100
**Figure 5.13** Eclatement des tables de fait ..... ... 101
**Figure 5.14** Profils d'accès à la base de données ..... ... 102
**Figure 5.15** Ecran : Affichage de plusieurs indicateurs ..... ... 104
**Figure 5.16** Ecran : Détail d'un indicateur ..... ... 105
**Figure 5.17** Ecran : Statistiques de consultation ..... ... 106
**Figure 5.18** Architecture générale ..... ... 111

**Figure 5.19** Une partie du diagramme de classe de l 'éditeur des contenus …… … 111
**Figure 5.20** Aperçu du schéma XML des fichiers XML générés par l'éditeur des contenus …… … 112
**Figure 5.21** Exemple d'un fichier XML générés par l'éditeur des contenus ……. … 113
**Figure 5.22** Ecran : L'éditeur des contenus ……. … 113
**Figure A.1** Vue Multidimensionnelle des Data Warehouse……. … 118
**Figure A.2** Architecture d'un Data warehouse…… … 119
**Figure A.3** *Représentation graphique d'un* RELAX NG …… … 121

# Liste des Tableaux

**Tableau 1.1** Langages de script ……. … 20
Tableau 1.2 Vulnérabilités des applications Web et problèmes potentiels dus à une conception médiocre ……. … 13
**Tableau 3.1** Description textuelle d'un Cas d'Utilisation ……. … 61
**Tableau 3.2** Outils de modélisation UML ……. … 63
**Tableau 3.3** Liste des Design Patterns ……. … 73
**Tableau A.1** Comparaison OLTP/OLAP ……. … 120

# SOMMAIRE

**INTRODUCTION ---------- 2**
**LES APPLICATIONS WEB ---------- 6**
1.1 Introduction ---------- 6
1.2 Définition d'une application WEB ---------- 6
1.3 Sites ou applications web? ---------- 6
1.4 Utilisations courantes des applications Web ---------- 7
1.5 Le processus de développement ---------- 7
1.5.1 Le modèle ---------- 9
1.5.2 Le workflows ---------- 10
1.5.3 Les risques ---------- 14
1.5.4 Les itérations ---------- 14
1.6 Contenu des sites web ---------- 14
1.6.1 Page dynamique et page statique ---------- 14
1.6.2 Traitement des pages Web statiques ---------- 15
1.6.3 Traitement des pages dynamiques ---------- 16
1.6.4 Accès à une base de données ---------- 16
1.6.5 Création de pages dynamiques ---------- 18
1.6.6 Choix d'une technologie de serveur ---------- 20
1.7 Patterns architecturaux pour les applications WEB ---------- 20
1.8 Sécurité des application web ---------- 21
1.8.1 Introduction ---------- 21
1.8.2 Principes fondamentaux de la sécurité ---------- 21
1.8.3 Menaces, vulnérabilités et attaques : définitions ---------- 22
1.8.4 Comment créer une application Web sécurisée ? ---------- 23
1.8.5 Aspects architecturaux pour la conception des applications Web ---------- 23
1.9 Critères ergonomiques pour l'évaluation des interfaces Web ---------- 27
1.9.1Guidage ---------- 27
1.9.2 Charge de travail ---------- 28
1.9.3 contrôle explicite ---------- 28
1.9.4 Adaptabilité ---------- 29
1.9.5 Gestion des erreurs ---------- 29
1.9.6 Homogénéité/Cohérence ---------- 29
1.9.7 Signifiance des codes et dénominations ---------- 30
1.9.8 Compatibilité ---------- 30
Conclusion ---------- 30
**TECHNOLOGIE XML ---------- 31**
2.1 Introduction ---------- 31
2.2 Pourquoi utiliser XML ? ---------- 32
2.3 Structure d'un document XML ---------- 33
2.3.1 Les composantes des documents XML ---------- 33
2.3.2 Unicode ---------- 34
2.3.3 Unicode et XML ---------- 34
2.3.4 La question de l'encodage ---------- 34
2.4 XML opposé à HTML ---------- 35
2.4.1 Exemple de HTML/XML : ---------- 35
2.5 Les DTD ( Document Type Définition/Définition de Type de Document) ---------- 36
2.6 Les schémas XML ---------- 38
2.6.1 langages de schéma ---------- 39
2.7 DTD ou XML-Schema ? ---------- 40
2.8 Feuilles de style ---------- 40
2.8.1 CSS feuilles de style en cascade ---------- 43
2.9 Les règles du jeu XML ---------- 43
2.10 Les espaces de nommage (*namespaces*) ---------- 44
2.10.1 Problématique: espaces de nommage et validation ---------- 44
2.11 XML et les bases de données ---------- 45

2.11.1 XML est-il une base de données ?----46
2.12 Résumé ----47
Conclusion----49
**MODELISATION UML----50**
3.1 Introduction----50
3.2 Les principes de la modélisation ----53
3.3 Pourquoi l'approche Objet ?----53
3.4 Les objets ----55
3.5 Les diagrammes UML----56
3.5.1 Vue fonctionnelle : ----57
3.5.2 Vue statique : ----57
3.5.3 Vue dynamique : ----58
3.6 Nouveauté de UML 2.0 ----59
3.7 UML est un langage pour communiquer----60
3.8 Etude basée sur les cas d'utilisation----60
3.9 Le choix d'un outil de modélisation ----62
3.10 UML et la Méta-modélisation----65
3.11 Les Design Patterns ----69
3.11.1 Définition ----70
3.11.2 Exemple de design Patterns----71
**APPROCHE DES DELEGUES DES PACKAGES ----74**
4.1 Le Modèle MVC----74
4.1.1 Introduction ----74
4.1.2 Problématique ----75
4.1.3 Facteurs à prendre en compte----75
4.1.4.Solution ----76
4.1.5 Variations----77
4.1.6 Comportement du modèle actif----77
4.1.7 Considérations relatives au test ----79
4.2. Approche des délégués des packages----81
4.2.1 Introduction :----81
4.2.2 Approche des délégués des packages : ----81
4.2.3. Amélioration du faible couplage M-V-C----82
4.2.4 Avantages de Approche des délégués des packages ----83
4.2.5 Inconvénients Approche des délégués des packages----84
4.2.6 Exemple d'implémentation de l'approche des délégués des packages -----84
Conclusion----85
**APPLICATION & IMPLEMENTATION----86**
5.1 Introduction----86
5.2 Le projet MERLIN ----86
5.2.1 Contexte ----86
5.2.2 Dénomination du projet----87
5.2.3 Problématique ----87
5.2.4 Solution ----87
5.2.5 Technologies employées ----88
5.2.6 Notion d'indicateur ----88
5.2.7 Alimentation du Data Warehouse----89
5.2.8 Diagramme des cas d'utilisation----90
5.2.9 Diagramme de séquence : Comportement dynamique du système --------92
5.2.10 Diagramme des packages ----94
5.2.11 Utilisation de l'approche des délégués des packages ----94
5.2.12 Modèle de données :----96
5.2.13 Interface (Ecrans)---- 103
5.2.14 Avantages constatés avec l'utilisation de l'approche des délégués des packages ---- 106
5.3 Site WEB AFPLemans---- 107

5.3.1 Préface ---------- 107
5.3.2 Contexte ---------- 107
5.3.3 Technologies utilisées ---------- 107
5.3.4 Choix de la technologie XML ---------- 108
5.3.5 Structure du site ---------- 108
5.3.6 Intégration des moyens de modification dynamique ---------- 108
5.3.7 Exemple d'outils facilitant l'édition des contenus : ---------- 109
5.3.8 Editeur des contenus intégré ---------- 109
5.3.9 Le fichier XML---------- 112
5.3.10 Interface de l'éditeur des contenus ---------- 113
**CONCLUSION ---------- 115**
**ANNEXE A : DATA WAREHOUSE---------- 117**
**ANNEXE B : TUTORIAL RELAX NG [CLARK01] ---------- 121**
**ANNEXE C : GLOSSAIRE DE VOCABULAIRE ---------- 127**
**REFERENCES BIBLIOGRAPHIES ---------- 135**

## Introduction

La préoccupation des professionnels de l'informatique est de recruter de nouveaux utilisateurs, cet objectif est vital pour la survie de ce métier, la voie à la réalisation de cet objectif est la simplicité et l'ouverture de portes franchissables par tout le monde. Les dernières années ont marqué la croissance exponentielle de l'introduction de l'outil informatique dans les divers métiers de la vie quotidienne de l'humanité, notamment Internet, ce phénomène qui ne cesse de croître et de prendre de la puissance. Internet doit sont succès à l'efficacité et à la fiabilité des solutions informatiques proposées dans le marché, et l'adoption des processus de développement logiciels bien élaborés.

Cette thèse s'inscrit dans un processus de développement de nouvelles approches pour la gestion des sites web et le trafic d'informations sur Internet. Plusieurs éditeurs de logiciels, notamment Microsoft avec le framework .Net [Microsoft], Adobe avec ses produits multimédias (GoldFusion Dreamweaver) [Adobe], et plus récemment SUN Microsystems avec JSF (JavaServer Faces) [SUN] ont lancé, depuis plusieurs années, des projets de recherche pour promouvoir cette nouvelle tendance, qui vise à réduire la distance entre la conception et l'implémentation des applications web. On trouve de plus en plus des EDI intégrant des ateliers de génie logiciel couvrant tout le processus de développement logiciel.

Les travaux que nous présentons dans ce mémoire visent à faciliter et à accélérer le processus de développement web. Ils tentent d'apporter des éléments de réponses à des questions théoriques dans le développement web mais surtout, par la même occasion, de proposer une approche pratique, facilitant le déploiement du modèle MVC dans un contexte de POO.

Le développement d'un programme se résume souvent pour beaucoup de personnes à une phase de codage, bien que cette étape soit essentielle, elle ne représente qu'une petite partie du cycle de vie d'un logiciel. Il est tout à fait possible de réaliser un logiciel sans utiliser d'autres étapes de conception, et tout le monde peut prétendre être en mesure de concevoir des interfaces ergonomiques, mais peu sont les développeurs en mesure d'apporter une preuve formelle (au sens mathématique) que leurs œuvres sont garanties et évolutives.

La conception orientée objets repose sur une solide assise théorique ayant déjà obtenu des résultats concrets, elle permet la maîtrise complète du processus, non seulement au niveau de la modélisation (analyse, conception) mais également en conduite de projet, définition des activités, règles de production, démarche qualité, réutilisation des modèles…

Plusieurs approches ont vu le jour, permettant de structurer l'architecture des systèmes informatiques, l'architecture trois-tiers étant un exemple dont le principe est relativement simple : Il consiste à séparer la réalisation des trois parties classiques (stockage des données, logique applicative, présentation). Les éléments permettant la réalisation classique d'un système en architecture trois-tiers sont les suivants: système de base de donnée (SGBD) pour le stockage des données, serveur applicatif pour la logique applicative, navigateur web pour la présentation.

Il y a aussi le modèle MVC (Modèle, Vue, Contrôle) bien qu'il ait été créé en 1979 par Trygve Reenskaug, reste toujours d'actualité, car il répond à une architecture récurrente utilisée dans la plupart des systèmes informatiques. Cependant, malgré sa popularité, le modèle MVC présente l'inconvénient de la difficulté que trouvent les développeurs pour l'implémenter dans leurs applications.

De la description théorique au codage existe une distance importante rendant équivoque l'implémentation du modèle MVC. Le langage Java, tente de faciliter cette étape, en mettant à la disposition des développeurs la classe *Observable* et l'interface Observer, mais ce moyen reste inadapté au développement web sous d'autres langages de programmation et incapable de profiter de toute la puissance de ce modèle.

Dans ce contexte nous proposons une nouvelle approche que nous appellerons « les délégués des packages » visant à réduire la charge des concepteurs d'architecture web et leur donner la possibilité de créer des applications web plus rapidement et plus efficacement, suivant un modèle générique qui constitue une sorte de design pattern. Cette approche fournit une démarche pratique pour les programmeurs pour implémenter le modèle MVC et exploiter sa modularité.

L'approche des délégués des packages est un design architectural pour implémenter le modèle MVC exploitant la nature fonctionnelle de ces trois composantes. L'approche proposée met en clair le passage de la théorie vers la pratique en donnant une démarche méthodologique d'insertion de nouveaux modules dans les applications WEB.

Structure du document

Le document s'organise en cinq chapitres principaux, les trois premiers chapitres exposent les technologies utilisées dans le cadre de ces travaux, et traitent leurs particularités d'un angle de vue théorique et pratique. Ainsi le premier chapitre présente les applications web, un domaine qui ne cesse d'évoluer. Devant cette expansion, le développement web se voit –inévitablement- adopter des processus logiciels à l'image des applications classiques, processus capables de supporter la taille et la rigueur imposées par la complexité de la nouvelle génération des applications web.

Dans le deuxième chapitre nous parlons du XML, technologie qui fait rêver les informaticiens, utilisée largement dans notre éditeur des contenus (voir chapitre 5). Ce chapitre traite le sujet XML en détail : Définitions, Standards, XML et bases de données, domaines d'applications, XML en pratique, DTD… Nous verrons l'utilité de cette technologie et ce qu'elle apporte pour le développement web.

Le troisième chapitre présente le langage de modélisation UML, un langage largement adopté dans la communauté des développeurs informatiques, grâce à sa simplicité et son utilité à représenter des systèmes robustes sous une séries de diagrammes graphiques, représentant chacun une vue différente du système. UML fournit, à la fois aux concepteurs et aux développeurs de systèmes informatiques, une puissante abstraction de haut niveau, permettant un meilleur suivi durant toutes les phases de réalisation et de déploiement de ces systèmes.

Le quatrième chapitre débute par une présentation du modèle MVC et ses différentes variantes, puis dans une deuxième partie, nous proposons l'approche des délégués des packages basée sur ce modèle. Cette approche peut être considérée comme un design pattern ou un modèle d'architecture type, différenciant clairement les trois composantes classiques d'un modèle MVC à savoir *Vue*, *Contrôle* et *Modèle*. Cette approche facilite visiblement le déploiement de ce modèle sous un langage de programmation orienté objet.

Le cinquième chapitre est consacré pour la présentation de deux projets que nous avons réalisés, le premier intitulé MERLIN, pour le compte de la sociétés ECS

Schindler spécialisée dans la fabrication et la commercialisation d'ascenseurs et d'escaliers mécaniques, le projet est une implémentation effective de l'approche des délégués des packages décrite dans le quatrième chapitre, la réalisation de ce projet nous donne l'occasion d'illustrer et de clarifier son processus de déploiement et de sa mise en application. Après avoir présenté la problématique, nous décrirons en détail le modèle conceptuel de l'architecture de notre système.

Le second projet consiste en la réalisation d'un site web de l'AFP ( Association pour la Formation du Personnel ) intégrant un éditeur des contenus destiné à améliorer et à faciliter la création et l'édition des sites web.

Enfin nous concluons sur nos travaux avant de dégager des perspectives d'évolutions inspirées d'autres technologies.

A la fin de ce mémoire on trouvera des annexes, détaillant des sujets non étoffés dans les chapitres principaux. Ainsi l'utilisateur intéressé peut les consulter pour de plus amples informations.

# 1

# Les applications WEB

## 1.1 Introduction

Le développement des applications WEB présente certaines particularités, au niveau technique et ergonomique. Cette spécificité nous oblige, au moment de la conception, à préconiser des méthodes de conception et des méthodes de travail dédiées à ce genre d'applications.

Ce chapitre met en claire cette particularité des applications WEB en mettant l'accent sur leurs caractéristiques.

## 1.2 Définition d'une application WEB

Une application Web est un ensemble de pages qui interagissent avec les utilisateurs, les unes avec les autres, ainsi qu'avec les différentes ressources d'un serveur Web, notamment les bases de données.

Une application Web est un site Web qui contient des pages et dont le contenu est partiellement ou totalement indéterminé. Le contenu final d'une page est déterminé uniquement lorsque l'utilisateur requiert une page depuis le serveur Web. Le contenu final d'une page variant d'une requête à une autre en fonction des actions de l'utilisateur, ce type de page est appelé page dynamique. Les applications Web sont construites de manière à répondre à différents types de défis et de problèmes.[Microsoft]

## 1.3 Sites ou applications web?

Les applications web reposent sur des technologies sous-jacentes qui rendent leur contenu dynamique et qui permettent à l'utilisateur de modifier l'état applicatif sur le serveur. La distinction entre sites web et applications web est subtile, puisqu'elle réside dans le dernier cas en la possibilité d'affecter l'état applicatif sur le serveur *via* un navigateur. Sans cela, il est inapproprié de parler d'application web. Seuls les systèmes qui offrent cette possibilité, et les serveurs d'application qui utilisent un serveur web pour l'interaction utilisateur, peuvent être considérés comme des applications web. Pour toutes les applications web, aussi simples soient-elles, l'utilisateur doit transmettre plus que de simples requêtes de navigation ; en

général, il peut saisir du texte simple, sélectionner des boutons d'options, voire donner des informations binaires ou de fichier.

La distinction devient encore plus subtile dans le cas des moteurs de recherche, pour lesquels des critères de recherche relativement sophistiqués sont saisis. Les sites web que sont les moteurs de recherche ne font qu'accepter ces critères, qu'ils emploient dans une espèce de requête sur une base de données afin de renvoyer des résultats. Quand l'utilisateur en a terminé avec le système, il n'y a pas de différence notable dans l'état du moteur de recherche, excepté, bien sûr, dans les fichiers journaux et les compteurs d'accès. Il en va tout autrement pour les applications web qui, par exemple, acceptent en ligne des informations d'inscription. Ainsi l'état d'un site web dans lequel un utilisateur s'inscrit à une formation sera-t-il modifié quand ce dernier en aura fini avec l'application.

L'architecture d'un site web est tout ce qu'il y a de simple. Elle contient trois composants principaux : un serveur web, une connexion réseau et des navigateurs clients. Les applications web contiennent, en outre, un serveur d'application qui rend possible le traitement d'une logique et d'un état applicatif.

### 1.4 Utilisations courantes des applications Web

Les applications Web peuvent être utilisées de diverses façons par les visiteurs d'un site et les développeurs, notamment pour :

Permettre aux utilisateurs de trouver rapidement et facilement des informations sur un site Web riche en contenu. Ce type d'applications Web permet aux visiteurs du site de rechercher, d'organiser et de parcourir le contenu à leur convenance.

Collecter, enregistrer et analyser des données fournies par les visiteurs du site. Auparavant, les données saisies dans des formulaires HTML étaient envoyées sous forme de courriel aux employés ou sous forme d'applications CGI pour le traitement. Une application Web peut enregistrer les données d'un formulaire directement dans une base de données et créer des rapports Web pour l'analyse. Les exemples incluent des pages de banques en ligne, de contrôle des stocks, des sondages et des formulaires de commentaires.

Mettre à jour des sites Web dont le contenu change souvent. Une application Web évite au créateur d'avoir à mettre fréquemment à jour le code HTML du site. Les fournisseurs de contenu tels que les rédacteurs en chef alimentent l'application Web et celle-ci met automatiquement le site à jour.

### 1.5 Le processus de développement

On ne développe pas une application comme on suit une recette. Les compétences et les aptitudes des acteurs impliqués sont pleinement sollicitées. Cela ne diminue en rien l'importance de la définition du projet. Mais si les efforts consentis par une équipe de développement peuvent suffire pour mener un projet à terme, ils seront mieux encore rétribués si dès l'abord le processus a été convenablement défini.

Un processus de développement assure quatre fonctions [Booch96] :

- Déterminer l'ordre des activités d'une équipe,
- Spécifier les artefacts à développer,
- Guider la tâche des développeurs et de l'équipe dans son ensemble,

- Offrir des critères pour le contrôle et l'évaluation des produits et des activités du projet.

Un processus de développement logiciel peut prendre la forme d'un ensemble de documents ou de fichiers hypertexte. Le processus définit les chaînes de processus métier *(workflows),* les activités, les artefacts et les rôles des acteurs impliqués dans le processus de développement. Un acteur, dans ce sens, est un rôle joué par un individu au cours du processus. Dans beaucoup d'équipes, des individus assurent les rôles de plusieurs acteurs.

Un *workflow* est un ensemble d'activités qui produit à terme un résultat concret et observable. Ainsi, la détermination des besoins, la modélisation par cas d'utilisation, l'analyse, la conception.

Un workflow définit un ensemble d'activités dans lesquelles des acteurs s'investissent. Les activités représentent que font les acteurs pour produire les artefacts de sortie du workflow. Un artefact désigne tout élément d'information qui est produit par les acteurs du processus. Les artefacts peuvent être du code, graphique, schémas de base de données, documents texte, diagrammes, modèles, etc.… Ils ont une propriété importante : ils peuvent être soumis à un contrôle de versions car ils évoluent souvent tout au long du processus ; un historique précis de ces suites de modifications est en effet critique.

Le processus décrit ici est basé sur le processus unifié de Rational (RUP) pour *Rational Unified Process* [Krachten00] et le *ICONIX Unified Process* [Rosenberg99]*.* Tous deux découlent de processus et de méthodologies orientés objet fondés sur les travaux de Grady Booch, Ivar Jacobson et Jim Rumbaugh [Booch99]. Ces deux processus ne sont pas les seuls processus formels utilisés aujourd'hui, il existe bien d'autres processus.

Comme d'autres, ces deux processus sont itératifs (voir figure 1.1). Cela veut dire que chaque phase du processus, analyse des besoins, analyse et conception, implémentation, test et évaluation, est répétée et raffinée jusqu'à ce que le processus réponde aux besoins et qu'il puisse être déployé. C'est une orientation toute différente du processus traditionnel de cycle de vie en cascade, dans lequel chaque phase du projet était achevée avant que l'équipe ne passe à la suivante. Les processus itératifs proviennent de cette constatation que, bien souvent, le développement de logiciels ne se déroule pas ainsi.

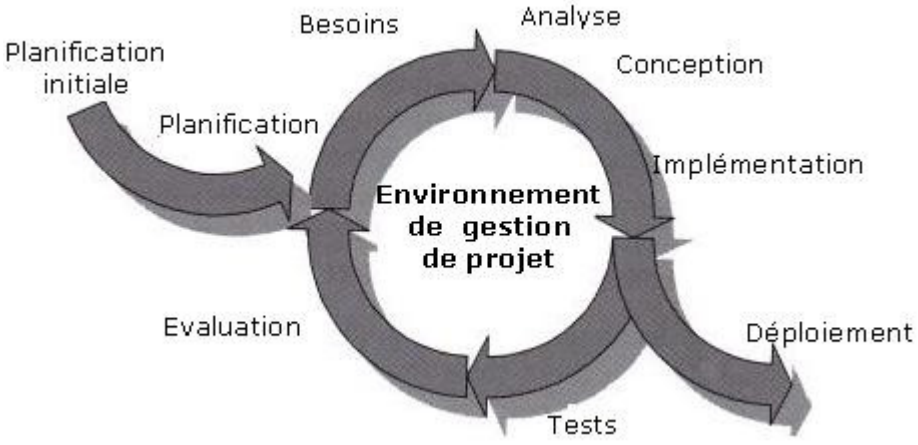



**Figure 1.1**
*Un processus itératif [Kruchten00]*

Alors que les processus décrits dans les manuels sont abstraits, la réalisation même d'un processus de développement est un processus en soi. Or, on ne saurait imposer une méthodologie à une équipe tout en espérant qu'elle se conforme à une

organisation, sans qu'un effort d'adaptation soit entrepris. Jim Conallen [Conallen00] défini un certain nombre de particularités qui peuvent influencer la qualité d'un processus de développement logiciel :

1. *Constitution de l'entreprise et organisation :* Les grandes entreprises avec en leur sein des groupes conséquents de talents spécialisés seront plus efficaces en suivant un processus rigoureux. Dans de telles organisations, les membres de l'équipe ne jouent qu'un rôle et ne réalisent qu'un ensemble restreint d'activités bien en accord avec leur compétence et leur expérience. Le nombre important de personnes impliquées renforce la nécessité de communications formelles entre les membres de l'équipe, ce qui se traduira par des artefacts détaillés et complets.
2. Les petites équipes, quant à elles, pourront préférer un processus moins contraignant. Ainsi, des équipes dont les membres ont déjà collaboré avec succès sur d'autres projets et dans lesquelles une dynamique de groupe a instauré une qualité de communication satisfaisante n'auront peut-être pas besoin de toute la rigueur d'un processus surchargé d'artefacts. Que ces types de structures reposent sur des compétences éprouvées ne fera néanmoins pas l'économie d'un processus bien défini. Simplement, certains aspects du processus, tels que les réunions officielles, n'auront pas la même importance.
3. *Nature de l'application.* Le domaine de l'application peut influencer considérablement la structure du processus de développement. Ainsi le contrôle de qualité deviendra-t-il une priorité dans des secteurs qui engagent la vie humaine, tels que les appareillages médicaux, les engins spatiaux ou les contrôles de centrales nucléaires. Dans tous ces cas, le rythme des itérations sera avant tout défini par la réalisation des objectifs fixés par l'assurance qualité.
4. *Les compétences de l'équipe de développement.* Le processus doit être adapté afin de tenir compte de la compétence de l'équipe. Des équipes peu expérimentées pourront tirer profit d'un processus plus encadré où les revues de groupe seront essentielles. Pour ce genre d'équipe, le processus de développement doit aussi devenir un processus d'apprentissage.
5. *Les priorités relatives entre les fonctionnalités, le délai de livraison, le taux de défauts acceptable, etc.* C'est l'importance relative de tout ce qui caractérise la livraison du système final qui détermine les éléments importants du processus de développement. Par exemple, dans le cas où le but prioritaire du projet est d'être le premier sur le marché, les tests qualité pourront être allégés, au profit de la capacité à s'adapter à de nouveaux besoins qui sera déterminante - quitte à ne répondre qu'à un minimum de besoins dans la première version. Le processus ainsi adapté répondra au mieux à la priorité accordée aux délais.

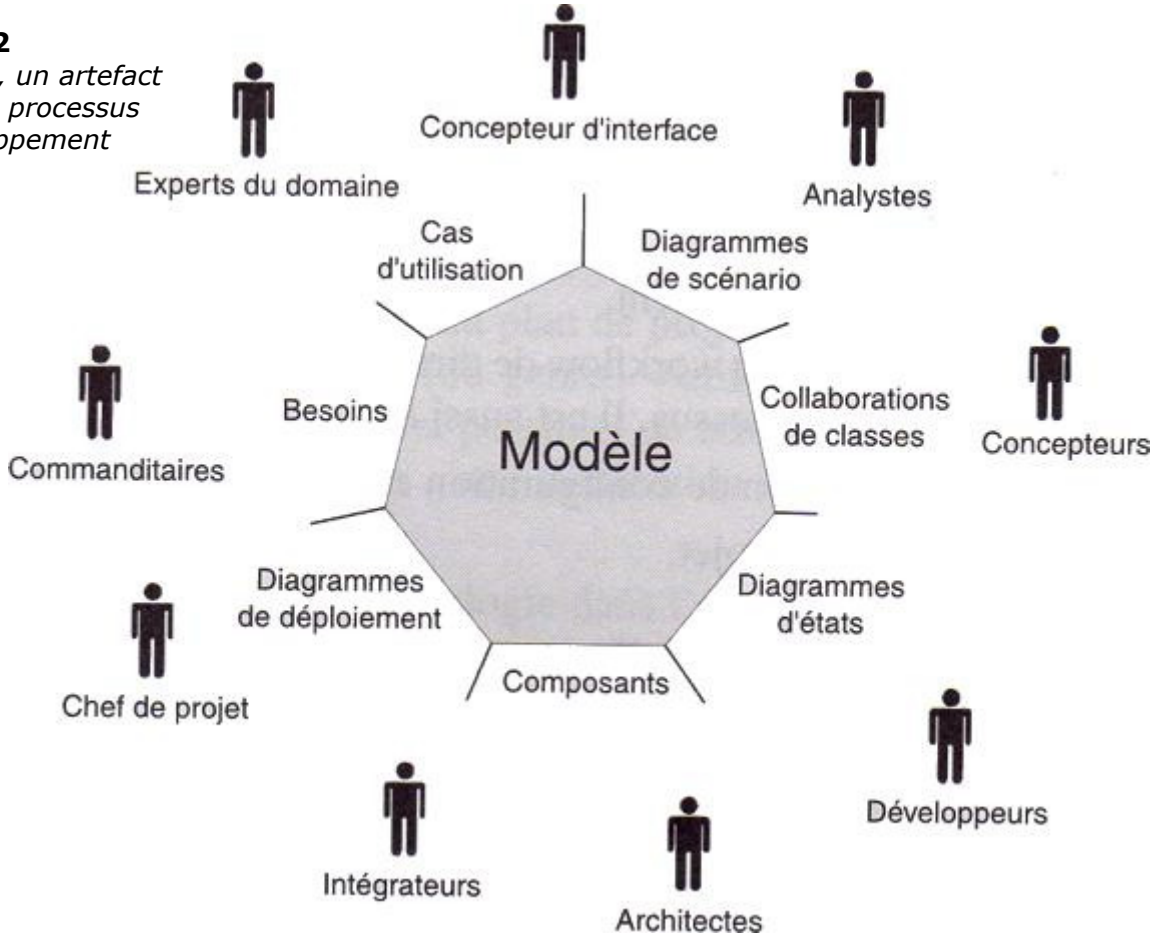
### *1.5.1 Le modèle*

La communication est essentielle dans le processus; c'est précisément une des attributions du modèle du système que de servir de mécanisme de communication. Le modèle est une représentation abstraite du système à développer, du système en développement et du système développé. Le modèle évolue donc avec le système en jouant un rôle majeur dans chaque phase du projet.

Le modèle est un ensemble d'artefacts représentant chacun une vue du système. Il est utilisé par presque tous les membres de l'équipe, depuis les commanditaires dont les conditions de travail seront améliorées par le système, jusqu'aux développeurs responsables de l'implémentation de ses composants. Chaque participant dans le processus de développement utilise ou enrichit le modèle différemment (voir figure 1.2) La communication est facilitée parce que tous les membres sont partie prenante à

l'élaboration du même modèle, même si chaque participant en a une vision particulière.

**Figure 1.2**
*Le modèle, un artefact*
*Central du processus*
*De développement*




Comme les systèmes deviennent de plus en plus complexes, leur compréhension dépasse les capacités d'un seul individu. La construction d'un modèle abstrait permet d'y remédier. Un bon modèle pourra ainsi servir à déterminer quels composants sont associés à quels cas d'utilisation et en quelle qualité. Le modèle pourra aussi aider à prédire les conséquences que des modifications de besoins pourraient avoir sur le système.

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. Par exemple, les cas d'utilisation dans le modèle sont associés aux réalisations de cas d'utilisation et aux diagrammes de séquence (scénario), qui à leur tour sont associés aux classes de base et aux collaborations qui implémentent le scénario. La traçabilité permet ainsi aux chefs de projet de naviguer dans le modèle pour trouver les solutions aux problèmes rencontrés pendant le processus de développement.

### *1.5.2 Le workflows*

Les principaux workflows du processus de développement logiciel sont les suivants : la direction de projet, les besoins, l'analyse, la conception, l'implémentation, les tests, le déploiement, la configuration et la gestion des changements. [Conallen00]

### *la direction de projet*

Le workflow de direction de projet est responsable de la gestion globale de l'application, incluant la gestion des ressources et des budgets, et assume aussi le rôle de porte-parole du projet. Le workflow de direction de projet produit les artefacts suivants :

6. Planning du projet,
7. Planning des itérations,
8. Gestion du risque,
9. Contrôle de l'avancement.

Le principal acteur du workflow de direction de projet est le chef de projet, responsable du bon déroulement du processus.

### *La collecte des besoins*

On appelle besoin un énoncé de ce que le système devrait faire. L'ensemble des besoins est défini dans la spécification des besoins. Définir les besoins pour un système ne va pas de soi : même en partant d'une vision commune, les représentations mentales d'un système diffèrent d'un individu à l'autre. Pire encore, les divergences de conception peuvent aussi se manifester parmi les développeurs.

L'expression des besoins doit donc proposer ce que le système devrait accomplir - non pas comment, mais quoi. Ce sont l'analyse et la conception qui répondront à la question du «comment» Les cas d'utilisation sont un outil clé pour exprimer clairement les besoins à tous les individus concernés et pour qu'ils puissent les comprendre d'une manière univoque.

### *L'analyse*

L'analyse et la conception sont souvent fortement associées ; nous préférons les séparer dans deux workflows distincts. En effet, bien que, dans la majorité des cas, les mêmes individus participent aux deux workflows, les activités et les objectifs sont très différents. L'analyse est le processus qui, en partant de l'examen des besoins, produit un modèle conceptuel du système à développer. Les artefacts de l'analyse incluent les classes et les collaborations détaillées, les diagrammes de séquence, les diagrammes d'états et les diagrammes d'activités. La conception opère sur les mêmes artefacts, mais seulement après que l'architecture y a été intégrée. Au moment de l'analyse, ces artefacts peuvent encore être discutés avec des experts du domaine et des représentants des utilisateurs, alors qu'une fois l'architecture appliquée au moment de la conception, ils deviennent le plus souvent trop techniques pour eux.

### *La conception*

Le workflow de conception traite les artefacts produits durant l'analyse en leur appliquant l'architecture. Le but principal de la conception est de rendre le modèle de l'analyse réalisable sous forme de logiciel. C'est alors la première fois que les concepts abstraits du métier sont confrontés au monde logiciel. Dans certains cas, l'introduction d'une architecture affecte tant le modèle que deux modèles séparés du système sont maintenus : un modèle d'analyse et un modèle de conception. Ces modèles ne sont que deux vues différentes du même système. Dans le cas où deux

modèles doivent coexister, le workflow de gestion de configuration et des changements a une nouvelle responsabilité : assurer la cohérence des deux modèles.

### *L'implémentation*

Bien que l'écriture et la compilation du code ne soient pas les seules tâches afférentes à l'implémentation d'un système logiciel, elles en représentent une grande part. L'implémentation applique des outils de développement logiciel aux artefacts de la conception. En plus des éditeurs et compilateurs courants, des environnements de développement intégrés peuvent générer du code à partir d'éléments du modèle ; cependant, la génération de code avec les environnements intégrés se limite pour l'essentiel à des éléments structuraux, laissant totalement à la charge du développeur le codage de la dynamique de l'application.

L'implémentation d'applications web implique souvent la maîtrise simultanée de nombreuses technologies. Pour le développement côté client, les langages et les techniques sont principalement HTML, JavaScript, Java, ActiveX et d'éventuelles technologies objet distribuées. Sur le serveur, les langages et les technologies sont plus variés avec des langages de troisième génération et des langages orientés objet (C/C++, Java, Smalltalk, Ada, Eiffel, PHP), mais aussi des technologies à base de composants (JavaBeans, COM), et enfin des technologies traditionnelles de bases de données. On retrouve ainsi sur le serveur toutes les technologies d'un système client-serveur classique.

### *Les tests*

Le workflow des tests consiste pour l'essentiel à évaluer les artefacts exécutables du système. Les tests sont dissociés de l'assurance qualité, laquelle affecte toutes les parties du système et non pas uniquement les exécutables.

De nombreux tests différents sont effectués sur le système; chacun d'entre eux essaie de déterminer une propriété du système. Un test de performance évalue la capacité du système à fonctionner rapidement sous la pression d'une forte charge. Un test de charge soumet le système à une utilisation intense pour établir son point de rupture ou simplement sa courbe de performance relative. Les tests fonctionnels vérifient que des fonctions particulières ont bien été implémentées telles qu'elles sont définies dans les spécifications des besoins. Les tests fonctionnels découlent souvent directement des cas d'utilisation.

Outre les tests d'évaluation de propriétés du système, d'autres tests permettent de valider certaines étapes du processus :

10. Test unitaire : il est effectué par le développeur sur une partie limitée du système qu'il a développé. Bien souvent, un test unitaire ne porte que sur un seul composant ou sur une petite collaboration de composants. Chaque développeur est censé effectuer les tests unitaires qui concernent son propre travail.
11. Test d'intégration : il valide la compatibilité des différentes interfaces des composants. Il est effectué après l'interconnexion de certaines parties du système mais avant l'assemblage de l'ensemble du système.
12. Test système : il vérifie que l'ensemble des besoins est satisfait. Ce test est effectué une fois tous les composants du système assemblés afin de valider le système dans son ensemble.

13. Test d'acceptation : c'est le test officiel par lequel la communauté des utilisateurs valide le système. Si le système est accepté, il est prêt à être déployé.

Parallèlement aux tests de validation de certaines étapes, les tests de non-régression prennent une importance plus sensible dans un processus de développement itératif. Ce sont les tests de non-régression qui contrôlent à nouveau un système qui a potentiellement changé. Bien que la traçabilité dans le modèle facilite la détermination des cas d'utilisation affectés par une modification du code, la possibilité qu'un changement effectué sur une partie du système en affecte une autre, apparemment indépendante, n'est jamais exclue. L'application systématique de tests de non-régression limite ce risque. Si un lest de non-régression détecte un dysfonctionnement, cela peut révéler une dépendance inconnue, qu'il convient de clarifier dans le modèle.

Les tests d'une application web se déroulent à peu près de la même manière que pour un autre système. L'équipe de test procède souvent avec l'application comme s'il s'agissait d'une boîte noire : seul le comportement extérieur du système compte. La nature de l'application web pourra amener à effectuer les tests sur un grand nombre de configurations de plates-formes et de navigateur client. Les applications destinées à l'Internet doivent considérer toutes les variétés de navigateurs qui pourraient être employées, ainsi que les plates-formes sur lesquelles ils fonctionnent. Pour les applications Internet, cela représente une lourde contrainte qui à elle seule peut limiter leur interface à leur plus simple expression.

***Le déploiement***

Selon les cas, le déploiement d'une application web peut s'avérer très facile ou au contraire très complexe. Ainsi, le déploiement d'une application intranet simple qui fonctionne sur un serveur et bénéficie d'un réseau existant pourra être très aisé. Seul le serveur doit être configuré car les clients possèdent probablement des navigateurs adaptés. Si l'application est conçue pour n'utiliser que les possibilités de base du client, il n'y a rien de plus à faire.

Mais si l'application doit gérer des aspects de sécurité et de gros volumes d'accès sur l'Internet, un investissement conséquent en termes de planification du déploiement est nécessaire. Les applications Internet doivent aussi soigneusement planifier la disponibilité des ressources et des accès réseau. La plupart des grosses applications Internet possèdent des connexions Internet redondantes et des systèmes de sauvegarde hors site. Le déploiement de telles applications exige une planification et une gestion rigoureuses.

***La gestion de configuration et des changement***

Si la gestion de configuration et des changements est un workflow en soi, c'est parce que son rôle est vital dans un processus itératif. La raison en est que la possibilité d'incorporer des changements dans le processus en cours, d'une façon contrôlée, est la condition même du succès de la convergence des itérations. Chaque changement est un petit ajustement par rapport à l'orientation fixée au projet.

Le contrôle des changements fournit au chef de projet des informations utiles qui lui permettent de valider la qualité des parties du projet. Des changements fréquents et non prévus dans certains secteurs indiqueraient probablement que ce sont là des secteurs à haut risque, qui n'ont pas fait l'objet d'une analyse suffisante en amont.

### *1.5.3 Les risques*

Le processus de développement vise avant tout à cerner au plus tôt tout risque qu'encourt un projet. Le risque, c'est l'inconnu ; et ce, aussi bien dans les secteurs de l'application qui reposent sur des technologies non éprouvées que dans des domaines où l'équipe de développement manque d'expérience. C'est le plus tôt possible dans le processus que le chef de projet, l'architecte et les membres clés de l'équipe doivent examiner l'ensemble des artefacts de l'analyse des besoins, incluant les cas d'utilisation, à la recherche de sources de risques.

La mise en œuvre du processus permet d'anticiper sur les risques potentiels du système et de délimiter ses zones à risque en les implémentant en premier. Le mécanisme consiste à laisser les cas d'utilisation jouer leur rôle central, qui est de conduire le processus. En effet, les cas d'utilisation sont une ressource pour presque toutes les activités du processus et chaque collaborateur se réfère aux cas d'utilisation pour valider les décisions prises durant les activités de développement. Cette approche centrée sur les cas d'utilisation favorise l'éclaircissement des risques en insistant sur le développement.

### *1.5.4 Les itérations*

La planification des itérations est essentielle dans la réussite du processus. Aucune place ne doit être laissée à l'improvisation. Une fois que les secteurs à risque ont été déterminés, le chef de projet dresse, avec l'architecte et les membres clés de l'équipe, le plan d'itération initial. Ce plan détaille les objectifs à court terme de chaque membre de l'équipe et définit les dates de livraison des artefacts.

La première itération permet en grande partie de définir la vision du projet. Elle aboutit principalement à deux artefacts, l'énoncé de la vision du projet et son cas d'utilisation. Cette première itération peut également inclure la mise en place de procédures pour la gestion de configuration et des changements, ainsi que l'analyse de l'environnement. Elle peut en outre contenir une modélisation préliminaire de cas d'utilisation ou la définition des paquetages des cas d'utilisation de haut niveau. Dans certains cas, le choix d'une architecture d'application web est évident. Par exemple, l'entreprise qui souhaite vendre sur l'Internet des produits doit opter pour un système de commerce électronique basé sur le Web.

Si la majorité des applications web de la première génération ont pu être développées hors de tout vrai processus sans trop de difficultés, c'est parce qu'elles étaient les produits de petites équipes (voire d'une seule personne) et qu'elles ne possédaient qu'une fonctionnalité limitée. Les applications web actuelles, qui deviennent de plus en plus complexes et qui jouent des rôles critiques en faisant appel à des équipes très étoffées, en nombre et en spécialistes, ne peuvent se soustraire, sous peine de dérive incontrôlée, à un processus fortement structuré.

## 1.6 Contenu des sites web

### *1.6.1 Page dynamique et page statique*

Un site Web est un ensemble de pages statiques et dynamiques. Une *page statique* n'est pas modifiée lorsqu'un visiteur la consulte : le serveur Web transmet la page au navigateur qui la sollicite sans la modifier. A l'inverse, une *page Web dynamique* est modifiée (construite) par le serveur avant d'être transmise au navigateur qui la sollicite. C'est pourquoi cette page est dite dynamique.

Vous pouvez par exemple créer une page pour afficher les résultats du programme de mise en forme et faire en sorte que certaines informations (telles le nom et les résultats de l'employé) soient déterminées lorsqu'une page est sollicitée par un employé donné.

### *1.6.2 Traitement des pages Web statiques*

Un site Web statique comprend un jeu de pages et de fichiers HTML associés hébergés sur un ordinateur exécutant un serveur Web.

Un serveur Web est un logiciel qui fournit des pages Web en réponse à des requêtes de navigateurs Web. Une requête de page est générée lorsqu'un utilisateur clique sur un lien d'une page Web ou saisit une URL dans le champ Adresse du navigateur.

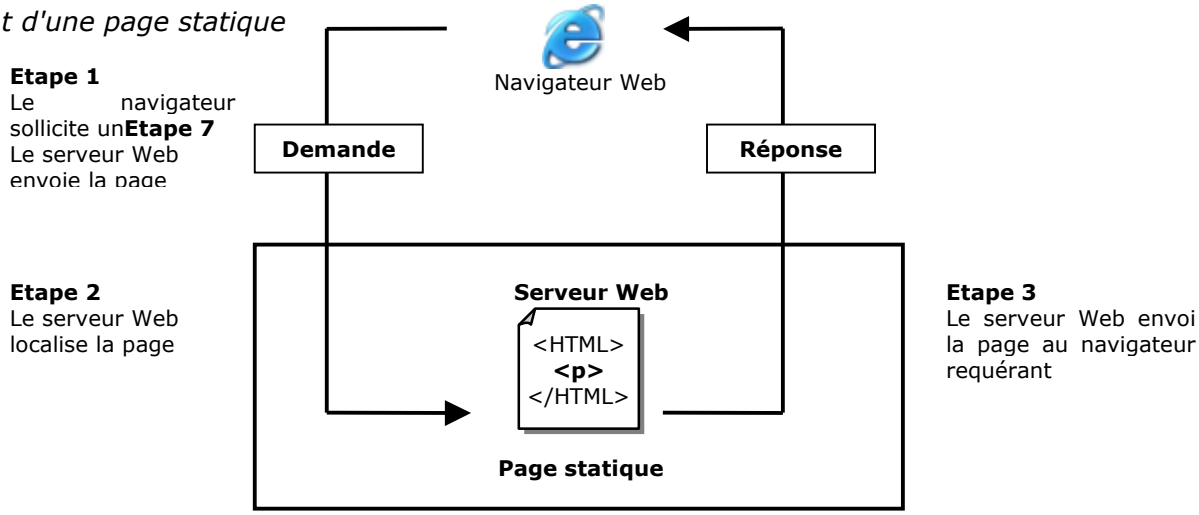
Le contenu final d'une page Web statique est déterminé par le créateur de la page et n'est pas modifié lorsqu'un utilisateur la sollicite. Exemple :

Chaque ligne du code HTML de la page est rédigée par le créateur avant que la page ne soit placée sur le serveur. Ce code HTML n'étant pas modifié une fois la page sur le serveur, cette page est dite statique.

```
<html>
      <head>
            <title>Informations relatives à la page</title>
      </head>
      <body>
            <h1>A propos de cette page</h1>
            <p>Cette page est une simple page HTML.</p>
      </body>
</html>
```

Remarque :
Lorsqu'un serveur Web reçoit une requête de page statique, il lit la requête, localise la page et la transmet au navigateur qui l'a sollicitée, tel qu'indiqué dans la figure 1.3 ci-dessous :

**Figure 1.3**
*Traitement d'une page statique*



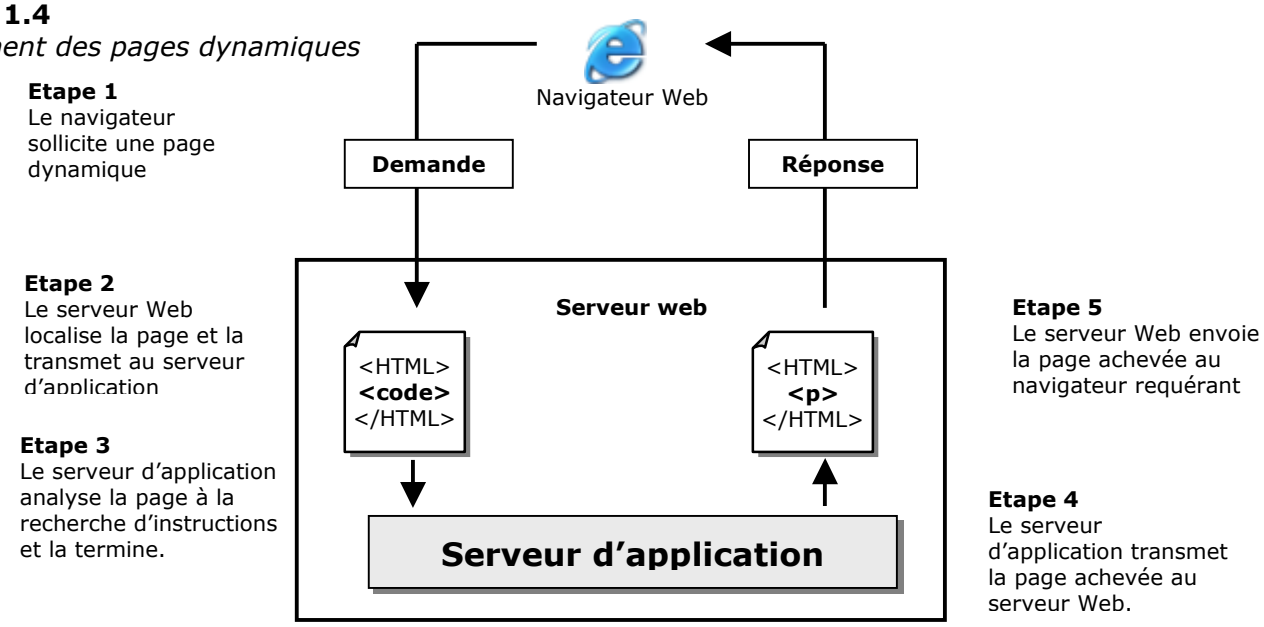
Dans le cas des applications Web, certaines lignes de code ne sont pas déterminées au moment où l'utilisateur sollicite la page. Ces lignes doivent être déterminées via un mécanisme avant que la page ne soit transmise au navigateur. Ce mécanisme est présenté dans la section suivante.

### *1.6.3 Traitement des pages dynamiques*

Lorsqu'un serveur Web reçoit une requête de page Web statique, il transmet simplement cette page au navigateur requérant. En revanche, lorsque le serveur Web reçoit une requête de page dynamique, il réagit de manière différente : il transmet cette page à une extension logicielle spéciale chargée d'achever la page. Ce logiciel spécial est appelé serveur d'application.

Le serveur d'application lit le code de la page, termine cette page en fonction des instructions figurant dans le code, puis en retire le code. Il en résulte une page statique que le serveur d'application renvoie au serveur Web, lequel transmet alors cette page au navigateur requérant. Le navigateur reçoit uniquement du code HTML pur lorsque la page lui est transmise. La figure 1.4 affiche un aperçu du processus :

**Figure 1.4**
*Traitement des pages dynamiques*




### *1.6.4 Accès à une base de données*

Un serveur d'application vous permet de travailler avec des ressources côté serveur telles que les bases de données. Une page dynamique peut, par exemple, ordonner au serveur d'application d'extraire des données de la base de données et de les insérer dans le code HTML de la page.

L'instruction d'extraction des données de la base est nommée requête de base de données. Une requête est composée de critères de recherche rédigés dans un langage de base de données appelé SQL (Structured Query Language) La requête SQL est rédigée dans les scripts ou les balises côté serveur de la page.

Un serveur d'application ne peut pas communiquer directement avec une base de données car le format propriétaire de cette dernière rend les données

indéchiffrables, de la même manière qu'un document Microsoft Word ouvert dans Bloc-notes ou dans BBEdit est indéchiffrable. Le pilote de base de données constitue l'unique moyen de communication entre le serveur d'application et la base de données : ce logiciel agit comme un interprète entre le serveur d'application et la base de données.

Une fois que le pilote a établi la communication, la requête est exécutée par rapport à la base de données et un jeu d'enregistrements est créé. Un jeu d'enregistrements est un ensemble de données extraites d'une ou de plusieurs tables de base de données. Le jeu d'enregistrements est renvoyé au serveur d'application et les données sont utilisées pour compléter la page dynamique.

Voici un exemple simple de requête de base de données rédigée en SQL :

```
SELECT nom, prenom, Email
FROM Employees
```

[retourne le nom et le prénom et l'email de tous les employés contenus dans la table Employees]

Et en voici une autre plus complexe :

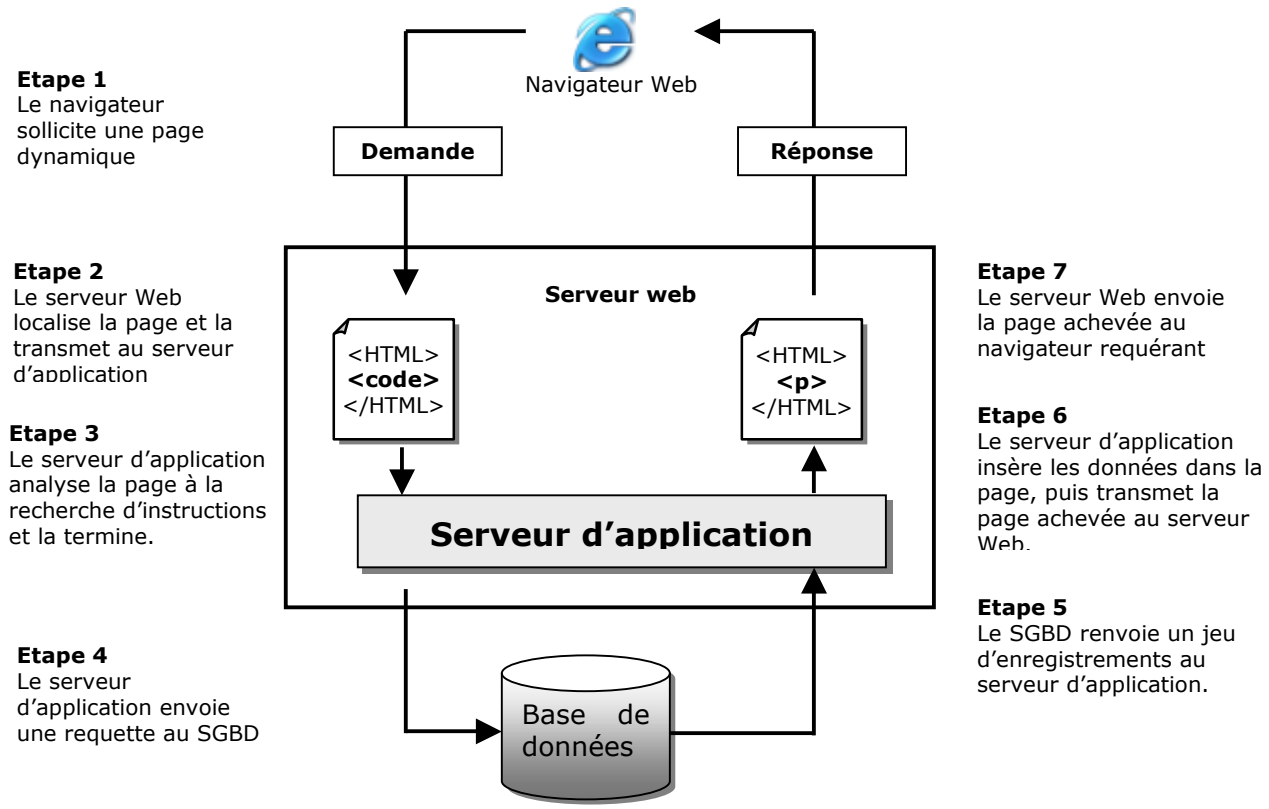
```
SELECT nom, prenom, Email, adressEntreprise
FROM Employees a , Entreprise b
 Where a.idEnreprise = b. idEnreprise
 And a.email like '%@monDomaine.com'
```

[retourne le nom et le prénom, email et l'adresse de l'entreprise de tous les employés dont l'e-mail contient la chaîne '@monDomaine.com' en dernière position]

Le mot clé SELECT regroupe tout un langage d'interrogation de base de données, avec sa propre grammaire il constitue un outil très puissant de sélection et de filtrage des données. D'un éditeur de base de données à un autre, la syntaxe diffère, mais il existe un standard SQL/92 qui offre la plupart des fonctionnalités de base d'interrogation et de manipulation des données. Les SGBD actuellement sur le marché fournissent de nombreux outils « requêteurs » permettant de construire des requêtes graphiquement sans nécessiter des connaissances du langage SQL. Le but est de se rapprocher d'une large tranche d'utilisateurs, qui dans leurs majorité ne sont pas des spécialistes du domaine

La figure 1.5 est une illustration du processus d'interrogation de la base de données via des requêtes et de la transmission des données au navigateur

**Figure 1.5**
*Accès à une base de données*




A partir du moment où les pilotes de base de données requis sont installés sur le serveur, vous pouvez utiliser pratiquement toutes les bases de données avec votre application Web.

Si vous prévoyez de créer de petites applications peu onéreuses, vous pouvez utiliser une base de données fichier, créée par exemple sous Microsoft Access. Si vous prévoyez de créer des applications stratégiques robustes, vous pouvez utiliser une base de données client/serveur, créée par exemple avec Microsoft SQL Server, Oracle 9i ou MySQL.

Si votre base de données réside sur un système autre que votre serveur Web, assurez-vous qu'il existe une connexion rapide entre les deux systèmes pour un fonctionnement efficace et rapide de votre application Web.

### *1.6.5 Création de pages dynamiques*

La création d'une page dynamique consiste à écrire d'abord le code HTML, puis à ajouter les scripts et les balises côté serveur au code HTML pour rendre la page dynamique. Lorsque vous visualisez le code obtenu, le langage apparaît incorporé dans le code HTML de la page. Par conséquent, ces langages sont appelés langages de programmation HTML intégrés. L'exemple de base ci-dessous utilise le code PHP :

```
<html>
	<head>
		<title>Une Simple page dynamique avec PHP </title>
	</head>
	<body>
		<h3>Ceci est un exemple de page dynamique</h3>
		<p>Cette page a été crée le :<b>
		<!-- début du script -->
		<?php
			$date = Date("l F d, Y");
			echo($date);
		?>
		<!-- Fin du script -->
		</b></p>
		<h3>Fin de la page</h3>
	</body>
</html>
```

Les instructions incorporées dans cette page exécutent les actions suivantes :

- Création d'une variable appelée *date* et assignation de la date courante à cette variable.
- Insertion de la valeur de la variable, « date », dans le code HTML.

Le serveur d'application renvoie la page suivante au serveur Web :

```
<html>
	<head>
		<title>Une Simple page dynamique avec PHP </title>
	</head>
	<body>
		<h3>Ceci est un exemple de page dynamique</h3>
		<p>Cette page a été crée le :<b>
		<!-- début du script -->
			Samedi 10 Mars 2006
		<!-- Fin du script -->
		</b></p>
		<h3>Fin de la page</h3>
	</body>
</html>
```

Le serveur Web transmet la page au navigateur requérant, lequel l'affiche de la manière suivante :

Le langage de script ou de balise à utiliser dépend de la technologie de serveur disponible sur votre serveur. Voici une liste des langages les plus utilisés pour les cinq technologies de serveur :

**Tableau 1.1**
*Langages de script*

| Technologie de serveur | Langage |
|---|---|
| ColdFusion | CFML (ColdFusion Markup Language) |
| ASP.NET | Visual Basic<br>C# |
| Pages ASP (Active Server Pages) | VBScript<br>JavaScript |
| JSP (JavaServer Pages) | Java |
| PHP | PHP |

### *1.6.6 Choix d'une technologie de serveur*

Le choix de la technologie de serveur dépend également du serveur d'application que vous souhaitez utiliser pour votre application Web. Un serveur d'application est un logiciel qui aide un serveur Web à traiter des pages Web contenant des scripts ou des balises côté serveur. Par exemple, si vous disposez de ColdFusion MX Server, vous pouvez choisir ColdFusion comme technologie de serveur. Si vous avez accès au serveur Microsoft Internet Information Server 5 (IIS) avec la plate-forme .NET, vous pouvez choisir ASP.NET. Faites appel à PHP si vous avez accès à un serveur Web avec un serveur d'application PHP.

- Pour en savoir plus sur ASP.NET, visitez le site Web Microsoft à l'adresse suivante : www.asp.net/ (en anglais).
- Pour en savoir plus sur ASP, visitez le site Web Microsoft à l'adresse suivante : msdn.microsoft.com/library/psdk/iisref/aspguide.htm.
- Pour en savoir plus sur JSP, visitez le site Web de Sun Microsystems à l'adresse suivante : `http://java.sun.com/products/jsp`.
- Pour en savoir plus sur PHP, visitez le site Web de PHP à l'adresse suivante : www.php.net/.

## 1.7 Patterns architecturaux pour les applications WEB

Un pattern architectural est l'expression d'un schéma fondamental d'organisation pour des systèmes logiciels. Il inclut un ensemble de sous-systèmes prédéfinis, précisent leurs responsabilités et prescrit des règles et des conseils pour organiser leurs relations

Les trois patterns les plus courants sont les suivants :

- **Le pattern du client web léger** *(Thin Web Client),* employé surtout pour les applications destinées à l'Internet pour lesquelles la configuration du client n'est pas contrôlée. Le client ne nécessite qu'un navigateur web standard compatible avec les formulaires. La logique métier est intégralement exécutée sur le serveur.
- **Le pattern du client web lourd** *(Thick Web Client),* permettant à une partie significative de la logique métier d'être exécutée sur le poste client. En général, le client met en œuvre HTML dynamique, des applets Java ou des

contrôles ActiveX pour exécuter la logique métier. Les communications avec le serveur passent toujours par HTTP.

- **Le pattern livraison web** *(Web Delivery),* pour lequel, en plus de l'emploi de HTTP pour la communication client-serveur, d'autres protocoles, tels IIOP ou DCOM, peuvent être mis en œuvre dans un système d'objets distribués. Le navigateur web agit principalement comme un conteneur et comme un périphérique de livraison dans le système d'objets distribués.

## 1.8 Sécurité des application web

### *1.8.1 Introduction*

Lorsqu'il est question de sécurité des applications Web, on pense immédiatement à ces pirates qui endommagent les sites Web, volent des numéros de cartes de crédit et bombardent les sites d'attaques de refus de service. Il est fréquent aussi de penser aux virus, aux chevaux de Troie et aux vers. Ce sont là les types de problèmes le plus largement évoqués car ils représentent quelques-unes des menaces les plus importantes auxquelles sont confrontées les applications Web actuelles.

Mais ils ne sont pas les seuls. D'autres problèmes essentiels sont fréquemment sous-estimés. Les menaces internes dues à des administrateurs malfaisants, des employés hostiles et des utilisateurs occasionnels qui tombent par erreur sur des données sensibles représentent un risque non négligeable. Le plus gros problème est peut-être l'ignorance.

La solution pour assurer la sécurité d'une application Web dépasse la technologie. Elle suppose un processus progressif d'implication des personnes et de recours à certaines pratiques.

Pour construire une application Web sécurisée, une architecture et une conception adaptées sont indispensables. Le coût et les efforts d'adaptation de la sécurité après le développement sont trop élevés. L'examen de l'architecture et de la conception vous aide à valider les fonctions de conception de votre application liées à la sécurité, avant le démarrage de la phase de développement. Ceci vous permet d'identifier et de corriger les vulnérabilités avant qu'elles ne soient exploitées, et avant que la correction ne requière des efforts inconsidérés de réingénierie.

**Quel sens donner au mot « sécurité » ?**

La sécurité est essentiellement destinée à protéger les ressources. Celles-ci peuvent être des éléments tangibles, comme une page Web ou votre base de données de clients ou plus abstraits, tels que la réputation d'une société.

La sécurité n'est pas une destination, elle est un chemin à parcourir. À mesure que vous analysez votre infrastructure et vos applications, vous identifiez des menaces potentielles et vous réalisez alors que chacune d'elles présente un degré de risque. La sécurité consiste à gérer le risque et à mettre en œuvre des contre-mesures efficaces.

### *1.8.2 Principes fondamentaux de la sécurité*

La sécurité s'appuie sur les éléments suivants :

**Authentification**

L'authentification pose la question : qui êtes-vous ? Il s'agit du processus visant à identifier de façon unique les clients de vos applications et services, qui peuvent être des utilisateurs finals, d'autres services, des processus ou des ordinateurs. En termes de sécurité, les clients authentifiés sont appelés *entités.*

**Autorisation**

L'autorisation pose la question : que pouvez-vous faire ? Il s'agit de déterminer les ressources et les opérations que le client authentifié est autorisé à consulter ou à effectuer. Les ressources englobent des fichiers, des bases de données, des tables, des lignes, etc., ainsi que des ressources dites système telles que des clés de registre et des données de configuration. Les opérations incluent l'exécution de transactions telles que l'achat d'un produit, le transfert d'argent d'un compte à un autre ou l'augmentation du taux de crédit d'un client.

**Confidentialité**

La confidentialité permet de s'assurer que les données restent privées et confidentielles et qu'elles ne peuvent pas être vues ou détournées par des utilisateurs non autorisés ou des indiscrets qui surveillent le flux du trafic sur un réseau. Le cryptage est souvent employé pour renforcer la confidentialité. Les listes de contrôle d'accès (ACL) sont un autre moyen d'imposer la confidentialité.

**Intégrité**

L'intégrité garantit que les données sont protégées contre toute modification accidentelle ou délibérée (malveillante) Comme la confidentialité, l'intégrité est une préoccupation majeure, notamment pour les données transmises sur des réseaux. L'intégrité des données en transit est généralement assurée par des techniques de hachage et des codes d'authentification des messages.

**Disponibilité**

En termes de sécurité, on entend par disponibilité la capacité pour des systèmes de rester disponibles pour les utilisateurs légitimes. Le but de nombreux auteurs d'attaques par refus de service est de détériorer une application ou de créer une saturation telle que les utilisateurs ne puissent plus y accéder.

### *1.8.3 Menaces, vulnérabilités et attaques : définitions*

Une menace est un événement potentiel, malveillant ou autre, qui pourrait nuire à une ressource. En d'autres termes, toute opération préjudiciable à vos ressources est une menace.

Une vulnérabilité est une faiblesse qui rend possible une menace. Cette possibilité peut être due à une mauvaise conception, à des erreurs de configuration ou à des techniques de codage inappropriées et non fiables. Une validation de la saisie insuffisante est un exemple de vulnérabilité de la couche application susceptible d'aboutir à des attaques au niveau des entrées.

Une attaque est une action qui exploite une vulnérabilité ou exécute une menace. Il s'agit par exemple d'envoyer des données d'entrée malveillantes à une application ou de saturer un réseau en vue d'entraîner un refus de service.

Pour résumer, une menace est un événement potentiel qui peut avoir un impact négatif sur une ressource, tandis qu'une attaque réussie exploite les vulnérabilités de votre système.

### *1.8.4 Comment créer une application Web sécurisée ?*

Il n'est pas possible de concevoir et de créer une application Web sécurisée tant que vous ignorez les menaces. Un processus toujours plus important qu'il est conseillé d'intégrer dans la phase de conception de votre application est la modélisation de la menace. Son objectif est d'analyser l'architecture et la conception de l'application et d'identifier les domaines potentiellement vulnérables qui permettraient à un utilisateur, même par erreur ou à un pirate mal intentionné, de compromettre la sécurité de votre système.

Une fois que vous connaissez les menaces, gardez à l'esprit, lors de la phase de conception, l'objectif de sécurité en appliquant des principes éprouvés de longue date. En tant que développeur, vous devez appliquer des techniques de codage sûres afin d'élaborer des solutions sécurisées, robustes et résistantes aux attaques. La conception et le développement de la partie logicielle de la couche application doivent être étayés par une configuration fiabilisée du réseau, de l'hôte et de l'application sur les serveurs où le logiciel sera déployé.

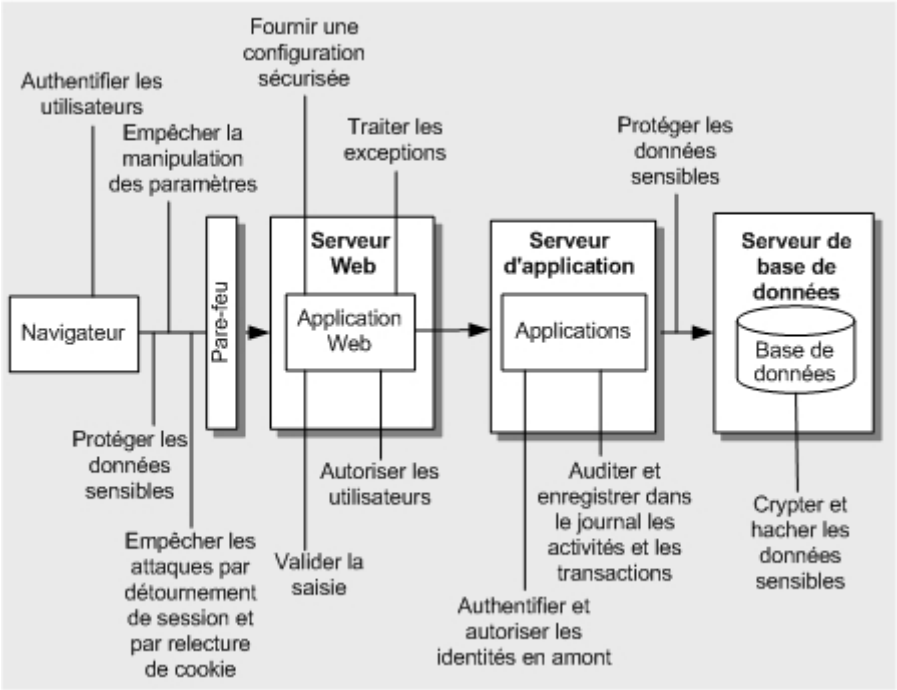
### *1.8.5 Aspects architecturaux pour la conception des applications Web*

Dans le monde des entreprises, la sécurité est un élément clé dans la mise en place des réseaux informatiques, l'architecture doit répondre rigoureusement à des normes de qualité et d'audit pour assurer le niveau de protection le plus élevé possible. Certaines entreprises, par crainte de divulguer leurs secrets professionnels préfèrent s'isoler dans des réseaux Intranet garantissant la protection contre les requêtes malveillantes des tiers. Mais l'ouverture du marché mondial pousse les sociétés à se doter –inévitablement- d'ouvertures sur le monde extérieur, via des applications Web de plus en plus nombreuses.

Les concepteurs et développeurs des applications Web doivent faire face à un grand nombre de difficultés. Au préalable, l'application doit être capable d'identifier l'utilisateur en faisant appel à une forme ou une autre d'authentification. Sachant que toutes les décisions d'autorisation ultérieures sont basées sur l'identité de l'utilisateur, il est essentiel que le processus d'authentification soit sécurisé et que le mécanisme de gestion de session utilisé pour suivre les utilisateurs authentifiés bénéficie du même niveau de protection. La conception de mécanismes d'authentification et de gestion de session ne constitue qu'une partie des difficultés auxquelles les concepteurs et développeurs des applications Web sont confrontés. D'autres difficultés apparaissent du fait que les données d'entrée et de sortie sont transmises sur des réseaux publics. La prévention de la manipulation des paramètres et la divulgation des données sensibles constituent d'autres problèmes majeurs. [Microsoft]

Certains des principaux problèmes devant être résolus en ayant recours à des pratiques de conception sécurisée sont illustrés dans la figure 1.6.

**Figure 1.6**
*Problèmes de conception des applications Web*




Le tableau 1.2 énumère les catégories de vulnérabilité, et pour chacune d'entre elle, met en évidence les problèmes potentiels pouvant découler d'une mauvaise conception.

**Tableau 1.2**
*Vulnérabilités des applications Web et problèmes potentiels dus à une conception médiocre*

| Catégorie de vulnérabilité | Problème potentiel dû à une conception médiocre |
|---|---|
| Validation des entrées | Attaques réalisées en insérant des chaînes malveillantes dans des chaînes de requête, des champs de formulaires, des cookies et des en-têtes HTTP. Il peut s'agir d'attaques d'exécution de commande, de script inter-site (XSS), d'injection SQL et de dépassement de la capacité de la mémoire tampon. |
| Authentification | Usurpation d'identité, décodage de mot de passe, élévation des privilèges et accès non autorisé. |
| Autorisation | Accès à des données confidentielles ou restreintes, falsification et exécution d'opérations non autorisées. |
| Gestion de la configuration | Accès non autorisé aux interfaces d'administration, capacité à mettre à jour les données de configuration et accès non autorisé aux comptes utilisateurs et aux profils de compte. |
| Données sensibles | Divulgation d'informations confidentielles et falsification des données. |

| Catégorie de vulnérabilité | Problème potentiel dû à une conception médiocre |
|---|---|
| Gestion des sessions | Capture des identificateurs de session se traduisant par le piratage de session et l'usurpation d'identité. |
| Cryptographie | Accès à des données confidentielles ou à des informations d'identification de compte ou les deux. |
| Manipulation des paramètres | Attaques de pénétration de répertoire, exécution de commande et contournement des mécanismes de contrôle d'accès (entre autres), se traduisant par la divulgation d'informations, l'élévation des privilèges et le refus de service. |
| Gestion des exceptions | Refus de service et divulgation d'informations sensibles de niveau système. |
| Audit et journalisation | Absence de détection des signes d'intrusion, incapacité à authentifier les actions d'un utilisateur et difficultés à diagnostiquer les problèmes. |

**Validation des entrées**

La validation des entrées est un problème délicat et la responsabilité principale de sa solution repose sur les épaules des développeurs d'applications. Néanmoins, une validation correcte des entrées est l'une des meilleures mesures de défense contre les attaques actuelles visant les applications. Elle constitue une contre-mesure efficace pour empêcher ses attaques.

La validation des entrées est délicate du fait qu'il n'existe pas de réponse unique vis-à-vis de ce qui constitue une saisie valide à travers des applications, voir au sein d'une même application. De même, il n'existe pas de définition unique de ce qu'est une entrée malveillante. Pour ne rien simplifier, l'utilisation que fait votre application de ces entrées influe sur le risque lié au détournement. Par exemple, stockez-vous des données à l'intention d'autres applications ou votre application consomme-t-elle les entrées provenant de sources de données créées par d'autres applications ?

Les pratiques suivantes visent à améliorer la validation des entrées de vos applications Web :

- Partir du principe que toute entrée de données est malveillante.
- Centraliser votre approche.
- Ne pas faire confiance à la validation côté client.
- Contraindre, rejeter et assainir vos entrées

**Partir du principe que toute entrée de données est malveillante**

La validation des entrées repose sur la supposition fondamentale que toute entrée est malveillante jusqu'à preuve du contraire. Que les entrées émanent d'un service, d'un partage de fichier, d'un utilisateur ou d'une base de données, validez-les si la source se trouve hors de votre frontière de sécurité. Par exemple, si vous appelez un service Web externe qui retourne des chaînes, comment pouvez-vous savoir si des commandes malveillantes ne s'y trouvent pas ? De plus, si plusieurs

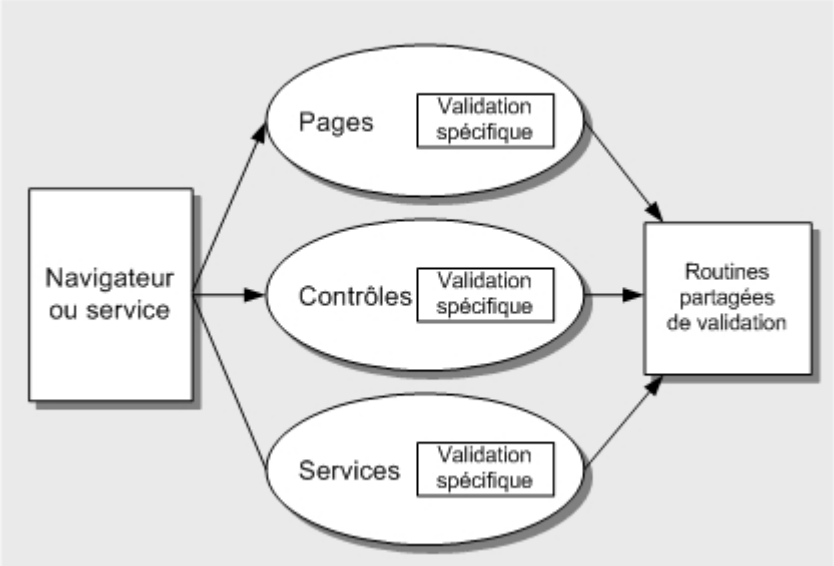
applications écrivent dans une base de données partagée, lorsque vous lisez les données, comment savoir si elles sont saines ?

**Centraliser votre approche**

Faites de votre stratégie de validation de saisie un élément central de votre conception d'application. Envisagez une approche de validation centralisée, par exemple en utilisant un code commun de validation et de filtrage dans des bibliothèques partagées. Ceci garantit que les règles de validation sont appliquées en permanence. Ceci réduit en outre les efforts de développement et favorise la maintenance future. Imaginez un modèle où l'ensemble des cas d'utilisation de votre application implémentent un unique package de validation et de contrôle.

Cette approche est illustrée dans la figure 1.7.

**Figure 1.7**
*Une approche centralisée de la validation des entrées*




**Ne pas faire confiance à la validation côté client**

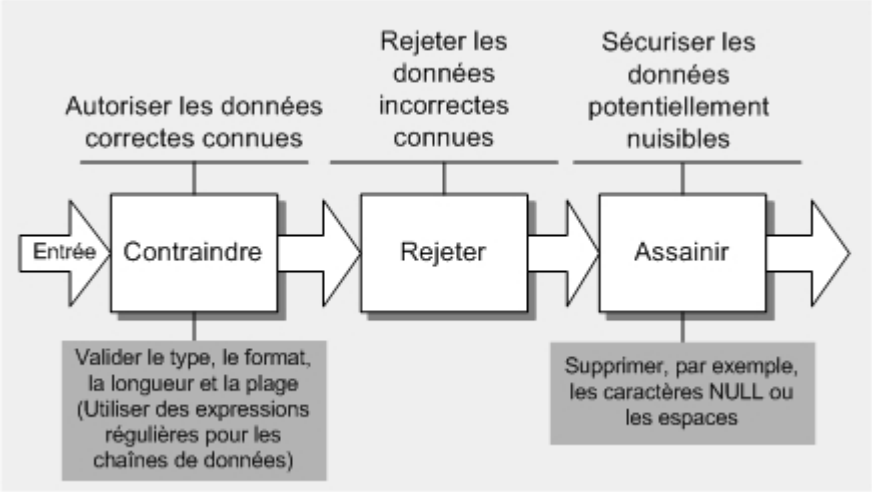
Le code côté serveur doit effectuer sa propre validation. Que se passe-t-il si un attaquant contourne votre client ou neutralise vos routines de script côté client, par exemple, en désactivant JavaScript ? Utilisez la validation côté client pour aider à réduire le nombre de boucles vers le serveur mais ne vous basez pas dessus pour la sécurité. Ceci est un exemple de défense renforcée.

**Forcer, bloquer et purifier la saisie**

L'approche de prédilection en matière de validation des entrées consiste à limiter ce que vous autorisez dès le début. Il est beaucoup plus facile de valider des données pré-filtrées que de valider des données en recherchant des caractères réputés incorrects. Lorsque vous concevez votre application, vous savez ce qu'elle attend. La plage de données valides est généralement un ensemble fini d'éléments, dans lequel une entrée potentiellement malveillante n'a pas sa place. Néanmoins, pour assurer une défense renforcée, vous pouvez également rejeter les entrées connues comme incorrectes, puis assainir la saisie. Cette approche nécessite la mise en

place de plans de test permettant de vérifier tous les cas de figure possibles et de prévoir des procédés appropriés à chaque cas. La stratégie recommandée est illustrée dans la figure 1.8.

**Figure 1.8**
*Stratégie de validation des entrées : contraignez, rejetez et assainissez la saisie de la validation des entrées*




## 1.9 Critères ergonomiques pour l'évaluation des interfaces Web

Les critères ergonomiques, qu'on les voit comme des règles ou des guides, sont à la base un des meilleurs outils que l'on puisse utiliser pour voir, comprendre et expliquer des problèmes ergonomiques. Ils permettent aussi d'orienter des choix de conception vers des pistes ergonomiquement solides, sans pour autant sacrifier l'aspect esthétique d'un site. Ils contribuent à éviter les pièges de la subjectivité et des goûts personnels en donnant un cadre de travail neutre et efficace.

Tel que souligné dans l'article de Scapin et Bastien [Bastin93], les critères ergonomiques ont trois caractéristiques qui les distinguent d'autres activités ergonomiques et en font un outil de choix:

1. Ils sont basés sur une analyse de l'interface, activité plus rapide et moins dispendieuse que les tests d'utilisabilité;
2. Ils sont utilisables par des non-spécialistes du domaine de l'utilisabilité;
3. Ils sont suffisamment explicites pour permettre des mesures précises, puis suffisamment standardisés pour donner des résultats reproductibles.

Voir le document original des auteurs publié chez l'INRIA [Bastin93]

### *1.9.1Guidage*

***Définition :***

Le *Guidage* est l'ensemble des moyens mis en œuvre pour conseiller, orienter, informer, et conduire l'utilisateur lors de ses interactions avec l'ordinateur (messages, alarmes, labels, etc.), y compris dans ses aspects lexicaux.

Quatre sous-critères participent au Guidage : Incitation, Groupement/Distinction entre Items, Feedback Immédiat et Lisibilité.

> L'objectif du *guidage est de f*aciliter l'utilisation du système et son apprentissage. [Nogier01]

***Justification :***

Un bon guidage facilite l'apprentissage et l'utilisation du système en permettant à l'utilisateur : de savoir, à tout moment, où il se trouve dans une séquence d'interactions, ou dans l'accomplissement d'une tâche ; de connaître les actions permises ainsi que leurs conséquences ; et d'obtenir de l'information supplémentaire (éventuellement sur demande). La facilité d'apprentissage et d'utilisation qui s'ensuivent conduisent à de meilleures performances et occasionnent moins d'erreurs

### *1.9.2 Charge de travail*

***Définition :***

Le critère *Charge de Travail* concerne l'ensemble des éléments de l'interface qui ont un rôle dans la réduction de la charge perceptive ou mnésique des utilisateurs et dans l'augmentation de l'efficacité du dialogue. Deux sous-critères participent au critère *Charge de Travail : Brièveté* (qui inclut les critères *Concision* et *Actions Minimales),* et *Densité Informationnelle.*

***Justification :***

Plus la charge de travail est élevée, plus grands sont les risques d'erreurs. De même, moins l'utilisateur sera distrait par des informations non pertinentes, plus il pourra effectuer sa tâche efficacement. Par ailleurs, plus les actions requises seront courtes, plus rapides seront les interactions.

### *1.9.3 contrôle explicite*

***Définition :***

Le critère *Contrôle Explicite* concerne à la fois la prise en compte par le système des actions explicites des utilisateurs et le contrôle qu'ont les utilisateurs sur le traitement de leurs actions.

Deux sous-critères participent au Contrôle Explicite : Actions Explicites et Contrôle Utilisateur.

***Justification :***

Quand les entrées des utilisateurs sont explicitement définies par eux-mêmes et sous leur contrôle, les ambiguïtés et les erreurs sont limitées. De plus, le contrôle qu'ont les utilisateurs sur le dialogue est un facteur d'acceptation du système.

### *1.9.4 Adaptabilité*

***Définition :***

L'adaptabilité d'un système concerne sa capacité à réagir selon le contexte, et selon les besoins et préférences des utilisateurs.

Deux sous-critères participent au critère *Adaptabilité : Flexibilité* et *Prise en Compte de l'Expérience de l'Utilisateur.*

***Justification :***

Plus les façons d'effectuer une même tâche sont diverses, plus les chances que l'utilisateur puisse choisir et maîtriser l'une d'entre elles, au cours de ses apprentissages, sont importantes. Il faut donc fournir à l'utilisateur des procédures, options, et commandes différentes leur permettant d'atteindre un même objectif. Par ailleurs, une interface ne peut convenir à la fois à tous ses utilisateurs potentiels. Pour qu'elle n'ait pas d'effets négatifs sur l'utilisateur, cette interface doit, selon les contextes, s'adapter à l'utilisateur.

### *1.9.5 Gestion des erreurs*

***Définition :***

Le critère *Gestion des Erreurs* concerne tous les moyens permettant d'une part d'éviter ou de réduire les erreurs, et d'autre part de les corriger lorsqu'elles surviennent. Les erreurs sont ici considérées comme des saisies de données incorrectes, des saisies dans des formats inadéquats, des saisies de commandes avec une syntaxe incorrecte, etc.

***Justification :***

Les interruptions provoquées par les erreurs ont des conséquences négatives sur l'activité des utilisateurs. De manière générale, elles rallongent les transactions et perturbent la planification. Plus les erreurs sont limitées, moins il y a d'interruptions au cours de la réalisation d'une tâche et meilleure est la performance.

### *1.9.6 Homogénéité/Cohérence*

***Définition :***

Le critère *Homogénéité/Cohérence* se réfère à la façon avec laquelle les choix de conception de l'interface (codes, dénominations, formats, procédures, etc.) sont conservés pour des contextes identiques, et sont différents pour des contextes différents.

***Justification :***

Les procédures, labels, commandes, etc., sont d'autant mieux reconnus, localisés et utilisés, que leur format, localisation, ou syntaxe sont stables d'un écran à l'autre, d'une session à l'autre. Dans ces conditions le système est davantage prévisible et les apprentissages plus généralisables; les erreurs sont réduites. Le manque d'homogénéité peut augmenter considérablement le temps de recherche.

Le manque d'homogénéité est aussi une raison importante du refus d'utilisation.

### *1.9.7 Signifiance des codes et dénominations*

***Définition :***

Le critère *Signifiance des Codes et Dénominations* concerne l'adéquation entre l'objet ou l'information affichée ou entrée, et son référent. Des codes et dénominations "signifiants" disposent d'une relation sémantique forte avec leur référent.

***Justification :***

Lorsque le codage est signifiant, le rappel et la reconnaissance sont meilleurs. De plus, des codes et dénominations non significatifs pour les utilisateurs peuvent leur suggérer des opérations inappropriées et ainsi conduire à des erreurs.

### *1.9.8 Compatibilité*

***Définition :***

Le critère *Compatibilité* se réfère à l'accord pouvant exister entre les caractéristiques des utilisateurs (mémoire, perceptions, habitudes, compétences, âge, attentes, etc.) et des tâches, d'une part, et l'organisation des sorties, des entrées et du dialogue d'une application donnée, d'autre part.

De plus, la *Compatibilité* concerne également le degré de similitude entre divers environnements ou applications.

***Justification :***

Le transfert d'information d'un contexte à un autre est d'autant plus rapide et efficace que le volume d'information à recoder par l'utilisateur est réduit. L'efficacité est accrue lorsque : les procédures nécessaires à l'accomplissement de la tâche sont compatibles avec les caractéristiques psychologiques des utilisateurs ; les procédures et les tâches sont organisées de manière à respecter les attentes ou les habitudes des utilisateurs ; les traductions, les transpositions, les interprétations ou références à la documentation sont minimisées.

**Conclusion**

Le développement web constitue un domaine en expansion, les entreprises, les particuliers tout le monde est concerné. En parallèle, les technologies évoluent vers des modèles facilitant visiblement la tâche des développeurs web, des solutions d'intégration et de conception d'architectures web sont proposées pour accélérer le processus de développement. Ce chapitre nous a présenté quelques aspects de ce domaine. Nous nous sommes focalisés sur le coté fonctionnel pour permettre une appropriation avec le thème de ce mémoire à savoir le rapprochement des méthodes de conception des applications web afin de proposer des solutions fonctionnellement exploitables.

# 2

# Technologie XML

## 2.1 Introduction

XML (*Extensible Markup Language* ou Langage Extensible de Balisage) dérive d'un langage développé dans les années 80, le SGML (Standard Generalized Markup Language) Ce langage était complexe à apprendre et utiliser quotidiennement, En particulier, il était inadapté à l'écriture de documents pour Internet. En pratique XML tente de se servir des principes de simplicité du HTML et de la souplesse SGML.

XML a été développé par un groupe de travail (GT) XML (XML Working Group) (initialement connu sous le nom de comité d'examen éditorial SGML [SGML Editorial Review Board]) constitué sous les auspices du Consortium du World Wide Web (W3C) en 1996. Le GT était présidé par Jon Bosak de Sun Microsystems avec la participation active d'un groupe d'intérêt XML [XML Special Interest Group] (auparavant connu sous le nom de groupe de travail de SGML [SGML Working Group]) également organisé par le W3C.

> XML est considéré comme 'simplification' du SGML, car il ne présente que 35 pages de spécification, dans sa 1ère version, contre 155 pages pour SGML.

> Formellement, on peut dire que XML est un sous-ensemble du SGML(ISO 8879). Qui dit sous-ensemble dit nécessairement *moins* de fonctionnalités. [W3C]

> HTML a permis la mise en place rapide du Web, le plus important système d'information jamais conçu.

La technologie XML, depuis son apparition, a connu un succès sans précédent, ceci est dû à sa crédibilité et sa subordination au monde réel, effectivement XML réduit le fossé entre la réalité et le support informatique. Chaque élément de la nature est englobé dans un autre, en partant du plus grand jusqu'au plus petit, c'est toujours le même principe et chaque niveau se réserve le contrôle de ses descendants et se distingue par certaines propriétés. XML est tout simplement une manière, bien réfléchie, de présenter cette vue le l'univers.

Une des caractéristiques, les plus importantes, du XML est que la mise en forme du contenu est totalement séparée des données. Cela permet de séparer complètement l'information (le contenu) de son apparence (présentation), et donc de fournir plusieurs types de sortie pour un même fichier de données, en fonction de l'utilisateur ou de l'application demandeuse (tableau, graphique, image, animation multimédia, fichier HTML, fichier PDF...) (figure 2.1)

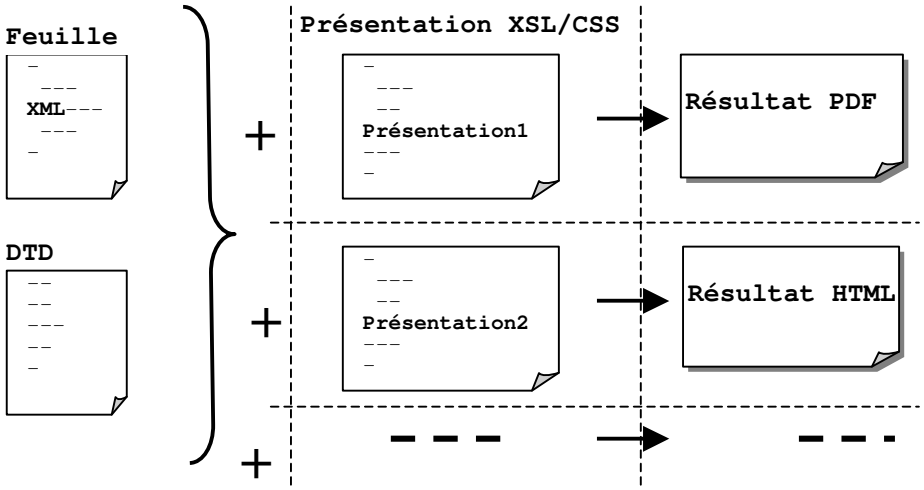



**Figure 2.1**
*Séparation des données et de la présentation avec XML*

À la différence d'HTML, mais comme SGML, XML n'a pas de balise prédéfinies : celles-ci sont fixées par l'application. Les balises sont donc extensibles, c'est-à-dire que l'on peut toujours ajouter des balises; elles doivent être correctement enchaînées, avec une balise de début et une balise de fin, comme des parenthèses. En résumé, les balises permettent de composer les éléments des documents. Les éléments peuvent être imbriqués autant qu'on le veut. Un document est simplement une suite d'éléments correctement imbriqués précédée d'une ligne d'en-tête. XML s'appuie donc sur un modèle d'échange par éléments balisés.

Un document XML est fondamentalement de type **texte**, par opposition à d'autres structures informatiques qui peuvent être de type binaire (programmes exécutables, base de données) Un simple éditeur de texte est donc capable de créer, d'ouvrir ou de modifier un document XML stocké sur disque. Mais un document XML n'existe pas uniquement sous forme de fichier. Il peut exister aussi temporairement en mémoire comme format de transfert de données entre deux programmes ou comme requête html POST.

### 2.2 Pourquoi utiliser XML ?

Beaucoup de développements tournent aujourd'hui autour du web, des assistants de poche, des WAP et autres, il devenait nécessaire de définir une norme universelle de description des données, quelque soit la plate-forme cible. XML

répond à cette exigence, en théorie du moins, car le phénomène a encore du chemin à parcourir sur la voie de la standardisation.

L'intérêt d'XML réside dans l'existence de bibliothèques qui implémentent les API standards d'XML (DOM, SAX etc.) et ce sur de nombreuses plateformes. Même une technologie propriétaire comme Flash (de Macromédia) embarque un parseur XML. Du coup, il devient réellement intéressant de concevoir ses paquets de données au format XML, lorsqu'elles doivent être communiquées à un autre service ou système.

Les objectifs de conception de XML sont les suivants :

1. XML devrait pouvoir être utilisé sans difficulté sur Internet.
2. XML devrait soutenir une grande variété d'applications.
3. XML devra être compatible avec SGML.
4. Il devrait être facile d'écrire des programmes traitant les documents XML.
5. Le nombre d'options dans XML doit être réduit au minimum, idéalement à aucune.
6. Les documents XML devraient être lisibles par l'homme et raisonnablement clairs.
7. La conception de XML devrait être préparée rapidement.
8. La conception de XML sera formelle et concise.
9. Il devrait être facile de créer des documents XML.
10. La concision dans le balisage de XML est de peu d'importance.

## 2.3 Structure d'un document XML

« Chaque document XML a une structure logique et une structure physique. Physiquement, le document se compose d'unités appelées entités. Une entité peut appeler d'autres entités pour causer leur inclusion dans le document. Un document commence à la « racine » ou entité document. Logiquement, le document se compose de déclarations, d'éléments, de commentaires, d'appels de caractère et d'instructions de traitement, qui sont indiqués dans le document par du balisage explicite. Les structures logiques et physiques doivent s'imbriquer correctement, tel que précisé dans les spécifications » [SpecXML]

### *2.3.1 Les composantes des documents XML*

On peut distinguer trois parties dans un document XML bien formé :

1. Le prologue : permet d'indiquer la version de la norme XML utilisée pour créer le document (cette indication est obligatoire à partir de la version XML 1.1) ainsi que le jeu de caractères (encoding) utilisé dans le document (élément facultatif)

   Exemple de prologue : `<?xml version="1.0" encoding="ISO-8859-1"?>`

2. La déclaration de type de document XML : contient ou désigne des déclarations de balisage qui fournissent la grammaire d'une classe de document. Cette grammaire est connue sous le nom de définition de type de document (DTD). La déclaration de type de document peut pointer vers un sous-ensemble externe (un type d'entité externe particulier), contenant des déclarations de balisage ou peut contenir directement les déclarations de balisage dans un sous-ensemble interne ou encore les deux. Le DTD d'un document consiste en la réunion des deux sous-ensembles.]

3. L'arbre des éléments XML : est constitué d'une hiérarchie de balises comportant éventuellement des attributs.

   Un attribut est une paire clé valeur écrit sous la forme *Cle="Valeur"*, ainsi une balise affectée d'un attribut aura la syntaxe suivante :

```
<element Attrib0="valeurAttribut1"> ValeurElement</element >
```

### *2.3.2 Unicode*

Unicode est un standard définissant un jeu de caractères. Sans entrer dans les détails, mentionnons que l'Unicode de base permet le codage des caractères sur 2 octets, soit plus de 65 000 caractères différents. En plus des techniques de codage, Unicode définit un jeu de caractères qui en compte plusieurs dizaines de milliers (la version actuelle d'Unicode est la 5.00 et elle définit près de 99 000 caractères). Tous les caractères des langues vivantes s'y retrouvent, de même que certaines langues mortes et des symboles. Unicode ne règle pas tous les problèmes de multilinguisme, mais il constitue la meilleure solution que l'on connaisse actuellement.

L'incorporation d'Unicode dans les applications client-serveur, les applications distribuées et les sites Internet permet une simplification de l'architecture et une réduction des coûts par rapport à l'utilisation des systèmes de codage traditionnels. Grâce à Unicode, un seul logiciel ou site Internet peut satisfaire simultanément et sans modification les demandes de plusieurs plateformes, langues et pays. Unicode permet aussi à des logiciels de provenance variée d'échanger des caractères sans pertes de données. [Unicode]

### *2.3.3 Unicode et XML*

Par défaut, les documents XML sont des documents *texte* dont le jeu de caractères est l'Unicode. En quelque sorte, il y a un seul jeu de caractères standard pour l'XML, et ce jeu de caractères est le plus important que l'on connaisse. D'autres jeux de caractères (comme l'ISO Latin 1) peuvent être utilisés, mais une application XML n'est pas obligée de les supporter, et donc de les traiter correctement.

Cette intime association entre Unicode et XML fait en sorte que les systèmes d'information basés sur XML peuvent gérer (en théorie) des documents ou des informations de toutes les langues, y compris dans les mêmes documents.

### *2.3.4 La question de l'encodage*

Il existe plusieurs façons d'encoder des caractères Unicode dans des fichiers informatiques :

- **UTF-8** : consiste à encoder les caractères ASCII sur un octet, puis les autres caractères sur 2, 3 ou 4 octets, sans ambiguïté. Il s'agit d'un encodage intéressant pour le français, car la plupart des caractères pourront s'écrire sur un seul octet (les lettres non accentuées), alors que les autres sont définis de telle sorte qu'en UTF-8 ils sont codés sur deux octets seulement.
- **UTF-16** consiste à représenter tous les caractères sur deux octets. Cet encodage est plus efficace pour des documents en langues asiatiques par exemple, mais inefficace pour des documents en langues européennes.

> Les deux encodages **Unicode** (UTF-8 et UTF-16) sont ainsi les seuls encodages que la recommandation **XML** 1.0 impose à tous les parseurs **XML**.

## 2.4 XML opposé à HTML

Pour comparer XML et HTML, les points les plus importants à considérer sont les suivants :

- XML ne remplace pas HTML ; il en serait plutôt un complément. XML et HTML poursuivent différents objectifs : HTML est conçu pour afficher des données et s'attache plutôt à leur aspect, tandis que XML est conçu pour décrire les données et s'attache plutôt à leur nature.
- À l'instar de HTML, XML ne fait rien. En effet, les balises XML servent à décrire la structure d'un élément tel qu'un bon de commande ; toutefois, ce bon de commande ne contient pas de code permettant de l'envoyer, de le traiter et de garantir qu'il sera rempli. D'autres gens doivent écrire du code pour que toutes ces opérations soient effectuées sur vos données au format XML.
- À la différence de HTML, les balises XML sont définies par l'auteur d'un schéma ou d'un document et sont illimitées. À l'inverse, les balises HTML sont prédéfinies ; il n'est possible d'utiliser que les balises prises en charge par la norme HTML en vigueur.

### *2.4.1 Exemple de HTML/XML :*

Pour comprendre la philosophie de XML, rien de mieux qu'un petit exemple. Prenons pour ce faire une page énumérant la bibliographie en langue française sur XML (à rapprocher des dizaines d'ouvrages existant en anglais sur le même sujet). Voici comment une telle page pourrait se présenter (avant interprétation par le logiciel de navigation) si elle était codée selon le langage actuel du Web, HTML :

```
<H2>Bibliographie XML</H2>
  <UL>
    <LI> JeanChristophe Bernadac et François Knab,
    <I>
     Construire une application XML</I>, Paris, Eyrolles,1999
    </LI>
    <LI> Alain Michard,
   <I>
    XML, Langage et Applications</I>, Paris, Eyrolles, 1998
    </LI>
 </UL>
```

Résultat dans un navigateur web :

Nous constatons dans cet exemple que les balises HTML fournissent des indications qui vont permettre au butineur de mettre en page (ou en écran) le texte qu'elles encadrent : <H2> et </H2> signalent un titre de niveau 2, <UL> et </UL> une liste non numérotée, <LI> et </LI> un élément de cette liste.

Voici ce que donnerait la même page si elle était codée en XML :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<BIBLIO SUBJECT="XML">
   <BOOK ISBN="9782212090819" LANG="fr" SUBJECT="applications">
      <AUTHOR>
         <FIRSTNAME>Jean-Christophe</FIRSTNAME>
         <LASTNAME>Bernadac</LASTNAME>
      </AUTHOR>
      <AUTHOR>
         <FIRSTNAME>François</FIRSTNAME>
         <LASTNAME>Knab</LASTNAME>
      </AUTHOR>
      <TITLE>Construire une application XML</TITLE>
      <PUBLISHER>
         <NAME>Eyrolles</NAME>
         <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1999</DATEPUB>
   </BOOK>
   <BOOK ISBN="9782212090529" LANG="fr" SUBJECT="général">
      <AUTHOR>
         <FIRSTNAME>Alain</FIRSTNAME>
         <LASTNAME>Michard</LASTNAME>
      </AUTHOR>
      <TITLE>XML, Langage et Applications</TITLE>
      <PUBLISHER>
         <NAME>Eyrolles</NAME>
         <PLACE>Paris</PLACE>
      </PUBLISHER>
      <DATEPUB>1998</DATEPUB>
   </BOOK>
</BIBLIO>
```

Ici aucune des balises ne décrit la présentation finale. On remarque, en revanche, que maintenant les balises ont un sens et une hiérarchie. Par exemple, l'élément "author" comprend le prénom (balise "firstname") et le nom (balise "lastname") de l'auteur. On remarque aussi que des informations supplémentaires (langue, sujet, n° ISBN), ont pu être ajoutées sous forme d'"attributs" situés à l'intérieur même des balises.

### 2.5 Les DTD ( Document Type Définition/Définition de Type de Document)

XML étant un métalangage, la définition de la structure attendu est nécessaire afin de s'assurer de la cohérence des données manipulées. La DTD d'un document décrit

l'ensemble des éléments admissibles au sein du document (< !ELEMENT…), ainsi que les attributs possibles pour chaque élément (< !ATTLIST…). (voir la figure 2.2)

DTD permet d'utiliser un fichier afin de vérifier qu'un document XML est conforme à une syntaxe donnée. La norme XML définit ainsi une définition de document type appelée DTD (*Document Type Definition*), c'est-à-dire une grammaire permettant de vérifier la conformité du document XML. La norme XML n'impose pas l'utilisation d'une DTD pour un document XML, mais elle impose par contre le respect exact des règles de base de la norme XML.

Ainsi on parlera de :

- **document valide** pour un document XML comportant une DTD
- **document bien formé** pour un document XML ne comportant pas de DTD mais répondant aux règles de base du XML.

Une DTD peut être définie de 2 façons :

- sous **forme interne**, c'est-à-dire en incluant la grammaire au sein même du document.
- sous **forme externe**, soit en appelant un fichier contenant la grammaire à partir d'un fichier local ou bien en y accédant par son URL. (figure 2.2)

**Figure 2.2**
*Exemple de fichier XML avec sa DTD*

**annuaire.xml**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE annuaire SYSTEM "annuaire.dtd">
<annuaire>
  <personne profession =" profession_1">
    <nom>nom_1</nom>
    <prenom>prenom_1</prenom>
    <email>prenom_1.nom_1@domaine_1.ext_1</email>
  </personne>
  <personne type=" profession_1">
    <nom>nom_2</nom>
    <prenom>prenom_2</prenom>
    <email>prenom_2.nom_2@domaine_2.ext_2</email>
  </personne>
</annuaire>
```

**annuaire.dtd ( fichier DTD associé)**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT annnuaire (personne*)>
<!ELEMENT personne (nom,prenom,email+)>
<!ATTLIST personne profession(profession_1|profession_2) profession_1">
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

**Remarques:**

- Pour chacun des éléments, on défini sa composition par <!ELEMENT nom_element (structure)>
- Pour chacun des attributs, on défini sa composition par <!ATTLIST nom_element nom_attribut (structure)>
- L'attribut profession est, ici, défini comme étant un choix imposé entre "profession_1", " profession_2", par défaut on met profession_1.
- `personne`***** signifie que dans un élément `annuaire` on peut trouver de 0 ou plusieurs éléments de type `personne`.
- `nom,prenom,email+` signifie que dans un élément `personne` on doit trouver une élément `nom`, un élément `prenom` et un ou plusieurs élément(s) `email` dans cet ordre.
- `nom`, `prenom` et `email` sont des éléments qui peuvent contenir du texte, les contenus #PCDATA sont dits contenus mixtes

### 2.6 Les schémas XML

Les définitions de type de document (DTD) ont rapidement laissées apparaître des carences dans le domaine des contraintes de structures et de données. L'universalité du langage XML requiert la création d'un ensemble d'applications verticales allant de la définition des documents (langage de schéma), en passant par leur structuration et composition (langage XML) et jusqu'à leur présentation à l'utilisateur final (langage XSL). L'adoption d'un langage de définition des documents basé sur XML, était donc inévitable afin de conserver une homogénéité dans la sphère XML déjà fort bien pourvue. [Laltruiste]

> XML Schema est un langage de description de format de document XML permettant de définir la structure d'un document XML. Un schéma XML est lui-même un fichier XML
>
> Une instance d'un *XML Schema* est un peu l'équivalent d'une DTD

Les schémas XML sont des documents qui servent à définir et valider le contenu et la structure des données XML, comme le schéma de base de données définit et valide les tables, les colonnes et les types de données constituant une base de données. Un schéma XML définit et décrit certains types de données XML à l'aide du langage de définition de schéma XML (XSD, *XML Schema Definition*). Les éléments du schéma XML (éléments, attributs, types et groupes) servent à définir une structure et un contenu de données valides ainsi que des relations entre certains types de données XML. Les schémas XML peuvent également fournir des valeurs par défaut pour les attributs et les éléments. Vous pouvez utiliser un schéma XML pour garantir la cohérence de certains types de données XML que se partagent plusieurs applications et organisations. Un schéma XML peut être utilisé comme un contrat régissant l'échange des données entre deux applications. Les organisations peuvent publier des schémas décrivant le format du code XML généré et utilisé par leurs applications. D'autres organisations et applications souhaitant échanger des données peuvent alors générer leurs applications à partir de ces schémas de sorte que leurs messages XML soient compris.

Par exemple, un bon de commande en XML peut être validé à l'aide d'un schéma XML avant d'être envoyé de l'acheteur au vendeur. Cette validation vérifie que tous les éléments de données existent, qu'ils sont ordonnés selon la séquence adéquate et que leur type de données est correct. Elle garantit ainsi que le destinataire du bon de commande sera en mesure d'interpréter correctement les données lors de leur réception.[MSDN]

Les schémas XML (xsd) sont une évolution des DTD pour la validation des documents XML. Ils sont standardisés sous forme de recommandation du W3C (XML Schema 1.0) depuis mai 2000. Malgré quelques critiques pointant surtout la complexité des schémas. Ils tendent aujourd'hui à remplacer les DTD car ils permettent de définir des modèles beaucoup plus complets.

La définition d'un schéma XML se réalise par l'intermédiaire de plusieurs éléments chargés de représenter l'arborescence et les informations d'un document XML.

L'extensibilité des définitions des documents apportée par les schémas XML, facilite l'échange, la fusion ou la réutilisation de données provenant d'une ou plusieurs sources. Effectivement, à l'instar des documents XML, les schémas XML pourront désormais se combiner en se fusionnant complètement ou partiellement afin de créer un modèle de contenu composite. D'ailleurs, plusieurs schémas peuvent être référencés à l'intérieur d'un unique document XML et donc valider la structure et le contenu de ce dernier.

La génération dynamique de documents XML et du modèle de contenu associé, devient alors beaucoup plus commode. Evidemment, cela demande une certaine compatibilité entre les schémas susceptibles d'être combinés. Grâce à cette normalisation, des schémas XML provenant d'un fournisseur pourront être fusionnés sans peines avec ceux d'un distributeur dans le but de créer un modèle de contenu spécifique destiné à la validation de documents XML générés à la volée.

### *2.6.1 langages de schéma*

Les principaux langages de schéma sont :

- **W3C XML Schema** est le choix naturel pour publication dans les répertoires de schémas et lorsque des informations de typage doivent être communiquées à des applications.
- **Relax NG** est plus simple à maîtriser (et donc à générer). Il dispose d'un meilleur pouvoir expressif que **W3C XML Schema** et permet d'exprimer plus de contraintes. Pour des fonctions de validations, c'est donc un langage qui permettra de coller au plus près aux documents décrits.
- Schematron est l'outil de finition qui peut tester ce qui ne peut pas être testé avec les autres langages. Il est généralement utilisé en complément de **W3C XML Schema** ou **Relax NG** (on n'utilise pas un outil de finition pour faire le travail de dégrossissage!). Dans la mesure où **Relax NG** décrit mieux les documents que **W3C XML Schema**, il y aura fréquemment plus à faire avec **Schematron** lorsqu'il est associé à **W3C XML Schema** que lorsqu'il est associé à **Relax NG**.
- Il ne faut pas oublier les **DTD**s qui sont un outil privilégié pour toute une génération de logiciels.

A côté de ces principaux langages, il existe d'autres langages moins connus qui peuvent avoir leur rôle à jouer.

## 2.7 DTD ou XML-Schema ?

Bob DuCharme écrit : « Beaucoup de développeurs XML ne sont pas satisfaits de la syntaxe des déclarations de balises décrites dans les spécifications, pour deux raisons. Premièrement, ils pensent que si les documents XML sont si bons pour décrire la structure des informations, alors la description d'une structure de type de document (ses « schémas ») devrait être dans un document XML et non écrite avec sa propre syntaxe. En plus d'être plus consistent, ceci permettrait plus facilement d'éditer et manipuler le schéma avec des outils habituels de manipulation de documents. Deuxièmement, ils pensent que la notation traditionnelle des DTD ne donne pas aux définisseurs de schémas la puissance nécessaire pour imposer assez de contrainte sur les données -- par exemple, la possibilité de dire qu'un certain type élément doit toujours avoir une valeur positive, qu'il ne peut pas être vide ou qu'il peut faire partie d'une liste au choix. Ceci faciliterait le développement de logiciels utilisant ces données car le développeur aurait moins de code de détection d'erreurs à écrire. » [DUCHARME01]

Un document XML se conforme à sa DTD, certes, mais une DTD ne se conforme pas à une DTD (puisque qu'une DTD décrit du XML mais que la DTD n'est pas elle-même décrite en XML). Du coup, impossible de passer à la méta-DTD.

Solution : le XML-Schema, il décrit du XML et se décrit lui même en XML; la réflexion devient donc possible. On peu désormais utiliser un méta-Schéma. Le XML devient ainsi un espace technique au même titre que les modèles et les langages.

Les schémas XML présentent les avantages suivants par rapport au DTD :

- Ils utilisent la syntaxe XML ; vous n'avez donc pas besoin d'apprendre une nouvelle syntaxe pour définir la structure de vos données.
- Ils prennent en charge des types réutilisables et vous permettent de créer de nouveaux types par le biais de l'héritage.
- Ils vous permettent de grouper des éléments en vue de contrôler la récurrence des éléments et des attributs.

## 2.8 Feuilles de style

Comme il a été dit, la philosophie d'XML consiste à bien séparer les données/documents (le fichier XML proprement dit) des traitements/présentations. Un document donné sera, lors de sa création, balisé uniquement en fonction de son contenu (sa sémantique) intrinsèque et indépendamment de sa restitution future (papier, écran, terminal Braille, synthèse vocale ou autre) — comme d'ailleurs de tout autre traitement automatique qui pourra lui être appliqué.

> XML/XSL apporte l'aspect « multi-représentation de document »

Cette indépendance par rapport aux applications qui vont le traiter en général, et par rapport à celles chargées de sa restitution en particulier, va lui conférer :

- Une très grande *interopérabilité* : le *même* document XML va pouvoir être affiché sur le Web et/ou produit en version papier, alimenter un SGBD, etc.
- Une grande *durabilité/réutilisation* : le document ne deviendra pas obsolète avec l'évolution des techniques informatiques ; il pourra sans difficulté être incorporé, en tout ou partie, dans des documents de nature très différente, être traité par des applications non prévues.

### *2.8.1 XSL( Extensible stylesheet language )*

XSL est le langage de description de feuilles de style du W3C associé à XML. A partir de fichiers de données XML, arbitrairement structurés, les auteurs utilisent une feuille de style XSL pour déterminer la façon dont ce contenu structuré devrait être représenté ; c'est-à-dire la façon dont le contenu source devrait être stylé, disposé et paginé pour un support de présentation donné, comme par exemple dans une fenêtre de navigateur web, un appareil portatif ou un jeu de pages physiques dans un catalogue, un rapport. [SpecXMLT].

Une feuille de style XSL est un fichier XML qui décrit comment doivent être présentés (c'est-à-dire affichés, imprimés, épelés) les documents XML.

La spécification est divisée en trois parties :

- XSLT, le langage de transformation
- XPath, le langage de navigation dans un document XML
- XSL-FO, le vocabulaire XML de mise en forme

#### *2.8.1.1 Le traitement d'une feuille de style*

À la différence de HTML, les noms des éléments en XML n'ont aucune signification de présentation intrinsèque. En l'absence d'une feuille de style, un processeur n'aurait aucun moyen à sa disposition pour rendre le contenu d'un document XML, autrement que sous la forme d'une chaîne indifférenciée de caractères. XSL fournit un modèle complet et un vocabulaire pour l'écriture d'une telle feuille de style avec une syntaxe XML.

XSL est bâti sur un travail précédent effectué sur les feuilles de style en cascade (CSS) et le langage de sémantique et de spécification du style du document (DSSSL).

Un processeur de feuille de style XSL reçoit un document, ou des données, XML avec une feuille de style XSL et produit une présentation de cette source XML de la manière voulue par l'auteur de la feuille de style. Ce procédé de présentation revêt deux aspects : premièrement, la construction d'un arbre élaboré à partir de celui de la source XML et, deuxièmement, l'interprétation de cet arbre résultant pour produire les sorties adaptées pour la représentation voulue. On appelle la première opération **transformation de l'arbre** et la seconde **mise en forme**. L'opération de mise en forme est accomplie par un **formateur**. Ce formateur peut simplement être le moteur de rendu d'un navigateur.

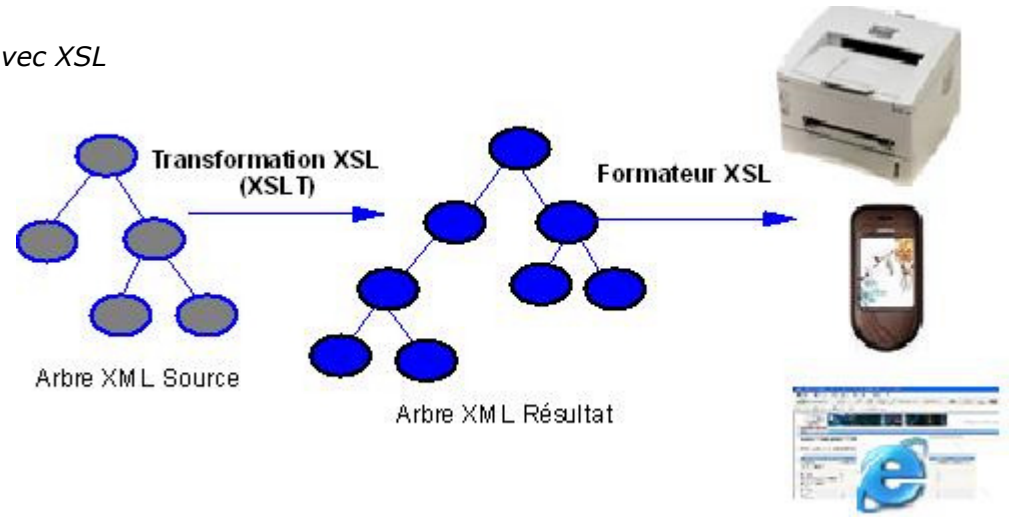
La transformation de l'arbre autorise une différenciation significative de la structure de l'arbre résultant par rapport à celui de la source. Par exemple, on pourrait ajouter une table des matières qui agirait comme un filtre de sélection sur un document source original ou on pourrait réarranger les données de la source en une présentation tabulaire ordonnée. Lors de l'élaboration de l'arbre résultant, l'opération de transformation de l'arbre inclut également les instructions nécessaires pour la propre mise en forme de celui-ci.

La mise en forme est activée par l'incorporation d'une sémantique de formatage dans l'arbre résultant. Cette sémantique de formatage s'expriment sous la forme d'un catalogue de classes d'**objets de mise en forme**. Les nœuds de l'arbre résultant sont des objets de mise en forme. Les classes d'objets de mise en forme

représentent des abstractions typographiques comme une page, un paragraphe, une table et ainsi de suite. Un contrôle plus fin de la représentation de ces abstractions est réalisé par un jeu de propriétés de mise en forme, ainsi celles régissant les indentations, les espacements entre les mots et les lettres. En XSL, les classes des **objets de mise en forme** et les **propriétés de mise en forme** fournissent un vocabulaire pour exprimer une présentation souhaitée.

Le modèle de traitement XSL n'est que conceptuel. On ne requiert pas d'une implémentation qu'elle reproduise ces opérations de manière distincte. En outre, les implémentations sont libres de traiter un document source à leur façon, pourvu que le résultat de ce traitement soit le même que celui qui aurait été obtenu selon le modèle de traitement conceptuel de XSL. La figure 2.3 montre le modèle conceptuel en détails.

**Figure 2.3**
*Transformation avec XSL*




La transformation de l'arbre construit l'arbre résultant. En XSL, on appelle ce dernier l'**arbre des éléments et des attributs**, avec des objets principalement dans l'espace de nommage de « l'objet de mise en forme ». Dans cet arbre, un objet de mise en forme est représenté comme un élément XML dont les propriétés se présentent sous la forme d'un jeu de paires XML attribut-valeur. Le contenu d'un objet de mise en forme est tout simplement le contenu de l'élément XML. La transformation de l'arbre est définie par la recommandation XSLT (figure 2.3)

La mise en forme est une interprétation de l'arbre résultant, dans sa forme d'arbre des objets de mise en forme, pour la production de la présentation souhaitée par l'auteur de la feuille de style, à partir de laquelle l'arbre des éléments et des attributs XML dans l'espace de nommage « fo » a été construit.

XSL a été développé pour donner aux auteurs un contrôle des fonctions nécessaires autant pour des documents paginés que pour fournir une structure équivalente aux « cadres » pour la navigation sur le web. Pour obtenir ce contrôle, XSL a étendu le jeu des objets et des propriétés de mise en forme. De plus, la sélection des composants d'une source XML, ceux pouvant recevoir un style (éléments, attributs, nœuds de texte, commentaires et instructions de traitement), se base sur XSLT et XPath, ce qui offre à l'utilisateur un mécanisme de sélection très puissant.

Une feuille de style XSL est appelée à partir d'un document XML par une "processing instruction" (PI) selon l'exemple suivant :

```
<?xml-stylesheet href="biblio.xsl" type="text/xsl" ?>
```

### *2.8.1 CSS feuilles de style en cascade*

CSS est un Langage utilisé pour décrire la présentation d'un document structuré écrit en HTML ou en XML. CSS est utilisé pour définir les couleurs, les polices, et d'autres caractéristiques liées à la présentation d'un document. L'objectif est de bien séparer la structure et la présentation du document.

CSS à été originellement définit, et optimisé, pour être utilisé avec le langage HTML. Les travaux les plus récents sur cette norme visent à l'ouvrir à tout document dont la structure est basée sur un jeu d'éléments quel qu'il soit (soit n'importe quelle application xml)

A la différence de XSL, CSS est très limité en terme de possibilités de traitement et de transformation sur les documents XML.

Une feuille de style CSS est appelée à partir d'un document XML par une "processing instruction" (PI) selon l'exemple suivant :

```
<?xml-stylesheet href="biblio.css" type="text/css" ?>
```

## 2.9 Les règles du jeu XML

Les informations doivent être :

- Soit encadrées par des balises ouvrantes(ex. <LIVRE>) *et* fermantes (ex. </LIVRE>) — contrairement à HTML où ces dernières n'étaient pas toujours obligatoires. On parle alors d'*éléments*. Les éléments doivent s'imbriquer proprement les uns dans les autres : aucun chevauchement n'est autorisé. Les éléments vides sont permis, selon le format <ELEMENTVIDE/>.

- Soit incluses à l'intérieur même des balises : on parle alors d'*attributs*. Exemple : <LIVRE SUJET="XML">. Ici l'attribut *SUJET* de l'élément *LIVRE* a la valeur "XML" . En XML, contrairement à HTML, les valeurs des entités doivent toujours être encadrées par des guillemets (simples ou doubles).

- Soit encore définies sous forme d'*entités*. Les entités sont des abréviations. Par ex; si "Extensible Markup Language" est déclaré comme une entité associée à la notation "xml"; cette chaîne de caractères pourra être abrégée en "&xml;" dans tout le fichier XML. Une entité peut aussi représenter un fichier XML externe tout entier. Ainsi un même fichier XML (par exemple notre bibliographie) pourra être utilisé par plusieurs pages XML différentes (par exemple une page spécifiquement consacrée à XSL pourra présenter à la fin la bibliographie spécifique d'XSL -quand elle existera- extraite automatiquement de notre bibliographie XML)

- La DTD (Définition de Type de Document). La structure arborescente du document XML (intitulé des balises, imbrications des balises, caractère obligatoire ou facultatif des balises et de leur ordre de succession…) peut être déclarée formellement dans le corps du document XML ou dans un fichier à part. Cette déclaration s'appelle une Définition de Type de Document (DTD). Elle s'effectue selon un formalisme particulier défini lui-aussi dans la spécification XML. En XML cette déclaration est *facultative*, ce qui donne une grande souplesse aux développeurs. On n'écrira donc une DTD que lorsqu'il y aura vraiment intérêt à le faire, par exemple pour contraindre la saisie/mise à jour du document XML.

- Lorsqu'un document XML possède une DTD associée et la respecte, on dit qu'il est *valide*. Lorsqu'il respecte seulement les règles de la grammaire XML (balises fermées, correctement imbriquées…) on dit qu'il est *bien formé*.

La spécification XML se trouve sur le site du W3C [SpecXML].

## 2.10 Les espaces de nommage (*namespaces*)

Les *namespaces*, ou "espaces de nommage", sont un mécanisme destiné à lever les ambiguïtés éventuelles des intitulés de balise. Par exemple "titre" peut aussi bien désigner le titre d'un ouvrage que celui d'une personne. Pour lever ces ambiguïtés, le mécanisme des *namespaces* consiste à utiliser des préfixes, par ex. "perso:titre" et "biblio:titre". Ces préfixes doivent être déclarés comme associés à une URL qui peut être fictive, mais qui le plus souvent fera référence à l'organisme garant du vocabulaire en question.

Les espaces de nommage XML visent à éviter les conflits de noms et les ambivalences dans des documents XML contenant des éléments d'une ou de plusieurs définitions de type de document (DTD). En les rattachant à une Uniform Resource Identifier (URI) des noms potentiellement ambivalents sont reliés à un espace de nommage et sont ainsi qualifiés. Un espace de nommage n'est ni plus ni moins qu'une série de noms de types d'éléments et d'attributs pouvant être identifiés de façon univoque par le biais d'une URI. Les espaces de nommage utilisés dans un document doivent être déclarés dans le document XML. À cette fin, l'URI de l'espace de nommage est affectée à ce qu'on appelle un préfixe d'espace de nommage. Un élément ou attribut d'un espace de nommage est alors qualifié par le préfixe d'espace de nommage. Le préfixe d'espace de nommage fait alors office d'abréviation de l'URI.

### *2.10.1 Problématique: espaces de nommage et validation*

L'URI de l'espace de nommage ne doit pas forcément pointer sur une adresse existant réellement sur la toile, en particulier sur une DTD affectée à l'espace de nommage. La déclaration d'espace de nommage n'a pas la moindre fonction de lien et l'URI est en fin de compte un identificateur univoque formel pour l'espace de nommage. En déclarant l'espace de nommage, aucune liaison n'est établie entre un espace de nommage et les déclarations pour les éléments et les attributs de l'espace de nommage. Plus généralement: Le concept d'espace de nommage et le concept de validation de XML sont bien distincts.

Ce qui signifie concrètement qu'il faut veiller à certaines choses pour créer des documents XML valides utilisant des espaces de nommage.

## 2.11 XML et les bases de données

XML devient le standard d'échange des systèmes d'information ouverts au-dessus du protocole HTTP. Il devient donc nécessaire d'une part, de stocker les documents XML reçus dans une base de données et d'autre part, de publier les données des bases en XML directement sur le Web. XML est issu de SGML, le langage de la gestion de documents. Il pourrait donc être naturel d'utiliser un système documentaire pour faire persister et interroger des documents XML. Mais les bases de données intègrent de plus en plus souvent des documents, par exemple sous forme de colonnes dans les systèmes objet-relationnel. Les bases e données XML se situent donc à la croisée des chemins des systèmes documentaires et des bases de données.

Stocker des documents XML directement en bases de données ou produire des documents XML directement depuis des bases de données présentent de nombreux avantages, outre celui de gérer la persistance, le partage et la sécurité des documents XML :

- Pouvoir saisir directement les données en base depuis des formulaires XML (XForms du W3C est prévu pour cela).
- Autoriser la publication directe des données sur le Web suite à une requête produisant le document XML désiré, sans manipulation textuelle, simplement en ajoutant une feuille de style pour la présentation des résultats.
- Pouvoir échanger des données avec des partenaires commerciaux, par exemple des commandes, des factures, etc., en suivant des DTD ou des schémas conventionnels , et ceci directement entre systèmes d'information d'entreprises.
- Permettre le développement de système d'interrogation multi-bases, véritables moteurs de recherche de la mémoire de l'entreprise étendue.

Depuis 1998, l'idée de bases de données XML a émergé. Deux approches se sont opposées : l'**approche *middleware*** consistant à utiliser un SGBD relationnel ou objet et à décomposer le document XML en éléments simples stockés dans des tables et l'**approche native** consistant à garder le document entier en base et à lui joindre des accélérateurs d'accès rapide (index). La première approche présente l'avantage de permettre une intégration douce de XML au cœur du système d'information existant par un *mapping* XML-relationnel approprié. Elle peut plus ou moins profiter des extensions objet des systèmes relationnels. La seconde approche nécessite au minimum le développement d'un nouveau type de données avec fonctions d'accès spécifiques et au plus d'un SGBD complet orienté vers la description et la manipulation de documents XML. [Gardarin02]

Ces deux approches, qui se subdivisent en de multiples sous-classes, nécessitent de toute manière de choisir un langage de description de données et langage de manipulation pour les données XML. La description s'appuie en général sur les schémas XML. La manipulation peut s'effectuer par des extensions de SQL3, la version objet-tionnel normalisée en 1999, avec des fonctions pour XML; c'est ainsi construit le langage de requête SQL/XML, proposé en particulier par Oracle. Cependant, SQL étant plutôt orienté vers les tables et quelque peu limité pour les données semi-structurées, le W3C a lancé un groupe de travail afin de normaliser un langage de requêtes pour XML. Celui-ci s'appelle Xquery. Aujourd'hui arrivé à maturité.

### *2.11.1 XML est-il une base de données ?*

Un document XML peut être considéré comme une base de donnés uniquement au sens le plus strict de l'expression. À savoir, une collection de données. XML en cela n'est guère différent des autres fichiers, car après tout, tous les fichiers contiennent d'une certaine manière des données. En tant que format de "base de données", XML présente cependant certains avantages. Il est par exemple auto-descriptif car les balises décrivent la structure et le type des noms des données ; il est portable puisque codé en Unicode, et il peut décrire les données sous la forme d'un arbre ou d'un graphe. Il présente aussi quelques inconvénients. Il est verbeux par exemple, et l'accès aux données est lent à cause de l'analyse [*parsing*] et de la conversion du texte.

Une question plus intéressante consiste à se demander si XML et les technologies qui lui sont associées constituent une "base de données" dans un sens plus large, c'est-à-dire au sens d'un système de gestion de base de données (SGBD). La réponse à cette question est : « XML est une sorte de SGBD ». Dans le sens d'une réponse positive, XML fournit plusieurs des caractéristiques que l'on retrouve dans les bases de données : le stockage (les documents XML), les schémas (DTD, XML Schemas, RELAX NG, etc.), des langages de requête (XQuery, XPath, XQL, XML-QL, QUILT, etc.), des interfaces de programmation (SAX, DOM, JDOM), et ainsi de suite. Dans le sens d'une réponse négative, de nombreuses caractéristiques présentes dans les bases de données lui font défaut : un stockage efficace, les index, la sécurité, les transactions et l'intégrité des données, l'accès multiutilisateur, les déclencheurs [*triggers*], les requêtes sur plusieurs documents, etc.

Ainsi, bien qu'il soit possible d'utiliser un ou plusieurs documents XML comme une base de données dans les environnements qui présentent une faible quantité de données, qui comportent peu d'utilisateurs, et qui se satisfont de performances modestes, cela échouera dans la plupart des environnements de production qui disposent de nombreux utilisateurs, requièrent une stricte intégrité des données et ont besoin de bonnes performances.

Un fichier .ini -- c'est-à-dire un fichier contenant les informations de configuration d'une application -- constitue un bon exemple de type de "base de données" pour lequel un document XML est approprié. Il est bien plus facile d'inventer un petit langage XML et d'écrire une application SAX permettant d'interpréter ce langage que d'écrire un analyseur de fichier au format texte délimité par des virgules. De plus, XML permet d'imbriquer les entrées, ce qu'il est difficile de réaliser avec des fichiers au format texte délimité par des virgules. Il ne s'agit pourtant pas vraiment d'une base de données puisque le document XML est lu et écrit linéairement au début et à la fin de l'application.

Il existe d'autres exemples plus sophistiqués de collections de données pour lesquelles un document XML pourrait être utilisé en tant que base de donnée: les listes de contacts personnels (noms, numéros de téléphone, adresses, etc.), les liens favoris dans un navigateur… Cependant, étant donné le coût réduit et la facilité d'utilisation des bases de données telles que Access, il semble, même pour les cas évoqués, qu'il existe peu de raisons d'utiliser un document XML en tant que base de données. Le seul véritable avantage d'un document XML est sa portabilité, et d'ailleurs, c'est un avantage moins crucial qu'il n'y paraît quand on considère le grand nombre d'outils disponibles pour sérialiser sous forme XML une base de données. [Bourret03]

## 2.12 Résumé

Pour finir avec XML, ce résumé en 10 points présente les concepts de base et donne un aperçu global des importantes applications dérivées d'XML.

### 1. XML est une méthode pour structurer des données

On entend par " données structurées " des éléments tels que des feuilles de calcul, des carnets d'adresses, des paramètres de configuration, des transactions financières, des dessins techniques, etc. XML est un ensemble de règles, de lignes directrices, de conventions pour la conception de formats texte permettant de structurer des données. XML facilite la réalisation de fichiers qui ne soient pas ambigus, et qui évitent les pièges courants, tels que la non-extensibilité, l'absence de prise en charge de l'internationalisation/localisation et la dépendance par rapport à certaines plates-formes.

### 2. XML ressemble un peu à HTML

Comme HTML, XML utilise des balises (des mots encadrés par '<' et '>') et des attributs (de la forme non = « valeur »). Mais alors que HTML définit la signification de chaque balise et de chaque attribut (et souvent la manière dont le texte qu'ils encadrent apparaîtra dans un navigateur), XML utilise les balises seulement pour délimiter les éléments de données et laisse l'entière interprétation des données à l'application qui les lit. En d'autres termes, si vous voyez "<p>" dans un fichier XML, ne supposez pas qu'il s'agit d'un paragraphe. Selon le contexte, cela peut être un prix, un paramètre, une personne, un p ... (d'ailleurs, qui a dit qu'il devait s'agir d'un mot commençant par "p" ?).

### 3. XML est du texte, mais il n'est pas destiné à être lu

Les programmes qui produisent de telles données les stockent souvent sur disque, dans un format binaire ou un format texte. Ce dernier format vous permet, si nécessaire, de consulter les données sans le programme qui les a produites. Les fichiers XML sont des fichiers texte, mais ils sont encore moins destinés à être lus par des individus que les fichiers HTML. Ce sont des fichiers texte, car ils permettent à des experts (tels que les programmeurs) de *déboguer* plus facilement des applications, et en cas d'urgence, d'utiliser un simple éditeur de texte pour corriger un fichier XML endommagé. Mais les règles des fichiers XML sont beaucoup plus strictes que celles des fichiers HTML. Une balise oubliée ou un attribut sans guillemets rendent le fichier inutilisable, alors qu'avec HTML, de telles pratiques sont souvent explicitement permises, ou au moins tolérées. Les spécifications officielles de XML indiquent que les applications ne sont pas *autorisées* à essayer de deviner ce qu'a voulu faire le créateur d'un fichier XML endommagé ; si le fichier est endommagé, une application doit s'arrêter immédiatement et émettre une erreur.

### 4. XML est bavard, mais ce n'est pas un problème

Comme XML est un format texte et qu'il utilise des balises pour délimiter les données, les fichiers XML sont presque toujours d'une taille plus importante que les formats binaires équivalents. Il s'agit là d'une décision prise en toute conscience par les développeurs de XML. Les avantages d'un format texte sont évidents (voir le point 3 ci-dessus), et ses inconvénients peuvent être généralement compensés à un autre niveau. L'espace disque n'est plus aussi coûteux qu'auparavant, et les programmes tels que winzip compressent très bien et très rapidement les fichiers.

De plus, les protocoles de communication tels que les protocoles de modem et HTTP/1.1 (le protocole de base du Web) peuvent compresser des données à la volée, ce qui économise de la bande passante aussi efficacement qu'un format binaire.

### 5. XML est une famille de technologies

Il existe *XML 1.0*, la spécification qui définit ce que sont les "balises" et les "attributs", mais autour de cette spécification, un nombre de plus en plus important de modules facultatifs fournissant des ensembles de balises et d'attributs ou des lignes directrices pour des tâches particulières ont été définis. C'est, par exemple, le cas de *Xlink*, qui décrit une méthode standard pour ajouter des liens hypertextes à un fichier XML. *XPointer* et *XFragments* sont des syntaxes pour pointer sur des parties d'un document XML. Un XPointer ressemble à un URL, mais au lieu de pointer sur des documents du Web, il pointe sur des éléments de données au sein d'un fichier XML. *CSS*, le langage des feuilles de style, s'applique à XML de la même façon qu'à HTML. *XSL* est le langage évolué pour la définition de feuilles de style. Il est basé sur *XSLT*, un langage de transformation utilisé pour réorganiser, ajouter ou supprimer des balises et des attributs. Le *DOM* est un ensemble d'appels de fonctions standard pour manipuler des fichiers XML (et HTML) à partir d'un langage de programmation. *Les Schémas XML 1* et *2* aident les développeurs à définir précisément leurs propres formats basés sur XML. Plusieurs autres modules et outils sont disponibles ou en cours de développement. Consultez régulièrement la page des rapports techniques du W3C.

### 6. XML est nouveau, mais pas si nouveau que ça

Le développement de XML a commencé en 1996, et XML est une norme du W3C depuis février 1998, ce qui peut laisser supposer qu'il s'agit d'une technologie plutôt immature. En fait, il ne s'agit pas d'une technologie très nouvelle. Avant XML, il existait SGML, développé au début des années 80, devenu norme ISO depuis 1986 et largement utilisé dans des projets de documentation de taille importante. Et il existait bien sûr HTML, dont le développement a commencé en 1990. Les concepteurs de XML ont simplement pris les meilleures parties de SGML, profité de l'expérience de HTML, et produit une technologie qui n'est pas moins puissante que SGML, mais infiniment plus régulière et plus simple à utiliser. Certaines évolutions, cependant, peuvent être assimilées à des révolutions... Il faut également savoir que SGML est principalement utilisé pour des documentations techniques et beaucoup moins pour d'autres types de données, alors que c'est exactement l'inverse avec XML.

### 7. XML conduit HTML à XHTML

Une application importante d'XML est le langage XHTML comme successeur de HTML. On retrouve dans XHTML beaucoup d'éléments du langage HTML. La syntaxe a été légèrement modifiée afin de se conformer aux règles d'XML. Plus généralement, un document fondé sur XML hérite sa syntaxe d'XML modulo quelques exceptions : par exemple, XHTML autorise la balise "<p>" mais pas "<r>". XML ajoute aussi un sens à cette syntaxe : par exemple, XHTML dit que "<p>" signifie "paragraphe" et non "prix" ou "personne" ou etc.

### 8. XML est modulaire

XML permet de définir un nouveau format de document en associant et en réutilisant d'autres formats. Cependant, deux formats développés indépendamment

peuvent avoir des éléments ou attributs de même nom. Il faut alors être très attentif à ne pas confondre les noms : "<p>" signifie-t-il "paragraphe" ou "personne" ? Pour éviter toute confusion lors de l'association de noms identiques, XML fournit un mécanisme d'espaces de nom. XSL et RDF sont de bons exemples de technologies fondées sur XML qui utilisent les espaces de nom. XML Schema a été conçu pour répercuter cette fonctionnalité modulaire pour la définition des structures XML, puisqu'il est facile de combiner deux schémas pour en produire un troisième qui sera associé au document fusionné.

### 9. XML est le fondement de RDF et du Web Sémantique

RDF, la norme du W3C pour les métadonnées, est un texte au format XML qui autorise la description de ressources et les applications métadonnées, tels que recueils de musiques, albums photos, et bibliographies. Par exemple, RDF pourrait vous laisser identifier des personnes dans un album photo sur le Web en utilisant des informations puisées dans une liste de contacts ; puis, votre messagerie pourrait automatiquement envoyer un message prévenant les personnes que leurs photos sont sur le Web. Tout comme les éléments HTML intégrés, comme les menus et formulaires, RDF intègre les applications et les agents en un Web Sémantique. De la même manière que les humains ont besoin de se mettre d'accord sur les mots qu'ils utilisent en communicant entre eux, les machines ont aussi besoin de mécanismes pour communiquer efficacement. Les descriptions formelles de terminologies dans un domaine particulier (la grande distribution ou l'industrie, par exemple) sont appelées ontologies, et sont une partie nécessaire du Web Sémantique. RDF, ontologies, et la représentation des connaissances sont tous des sujets décrits dans l'activité Web Sémantique.

### 10. XML est libre de droits, indépendant des plates-formes et correctement pris en charge

En choisissant XML pour un projet, vous bénéficiez d'un ensemble important et sans cesse croissant d'outils. Opter pour XML, c'est un peu comme choisir SQL pour des bases de données : vous devez encore construire votre propre base de données et vos propres programmes ou procédures qui la manipulent, mais un grand nombre d'outils et d'individus peuvent vous y aider. Et comme XML est libre de droits, vous pouvez vous en servir pour construire votre propre logiciel sans avoir à payer quoi que ce soit à qui que ce soit.

### Conclusion

XML tire sa popularité de sa puissance à cerner une multitude de problèmes de représentation sémantique et syntaxique sous une bibliothèque de vocabulaires riche et paramétrable à la demande. XML constitue un moyen incontournable pour l'échange des données formatées. A travers ce chapitre nous avons pu aborder les différents aspects du XML, nous avons constaté l'avantage de faire recours à cette technologie dans l'implémentation d'un outils d'édition des contenus pour la sauvegarde des données et de leur sémantique.(Chapitre 5). La plupart des langages de programmation évolués intègre désormais dans leurs bibliothèques de subroutines des parseurs XML puissants et couvrant pratiquement l'ensemble des besoins de navigation et de manipulation des objets XML.

# 3

# Modélisation UML

## 3.1 Introduction

L'objectif de ce chapitre est de donner une petite introduction à UML (Unified Modeling Language) il ne traite pas en détail tout les concepts d'UML. Il existe toute une bibliothèque de livre sur UML et les méthodes de son apprentissage ( voir la bibliographie ).

UML est apparu comme la notation de facto pour la modélisation orientée objet. Il résulte des travaux de Grady Booch et de Jim Rumbaugh qui, en 1994, ont combiné les notation de leurs deux méthodes ; les méthodes *Booch* et *OMT* (*Object Modeling Technique* ). Rejoints plus tard par Ivar Jacobson, créateur de la méthode *Objectory*, il ont acquis leur notoriété sous le nom des trois *amigos*. Bien que d'autres auteurs ont contribué à UML, en particulier Cris Kobryn, qui en a développé les versions les plus fines [Larman2002].(figure 3.1)

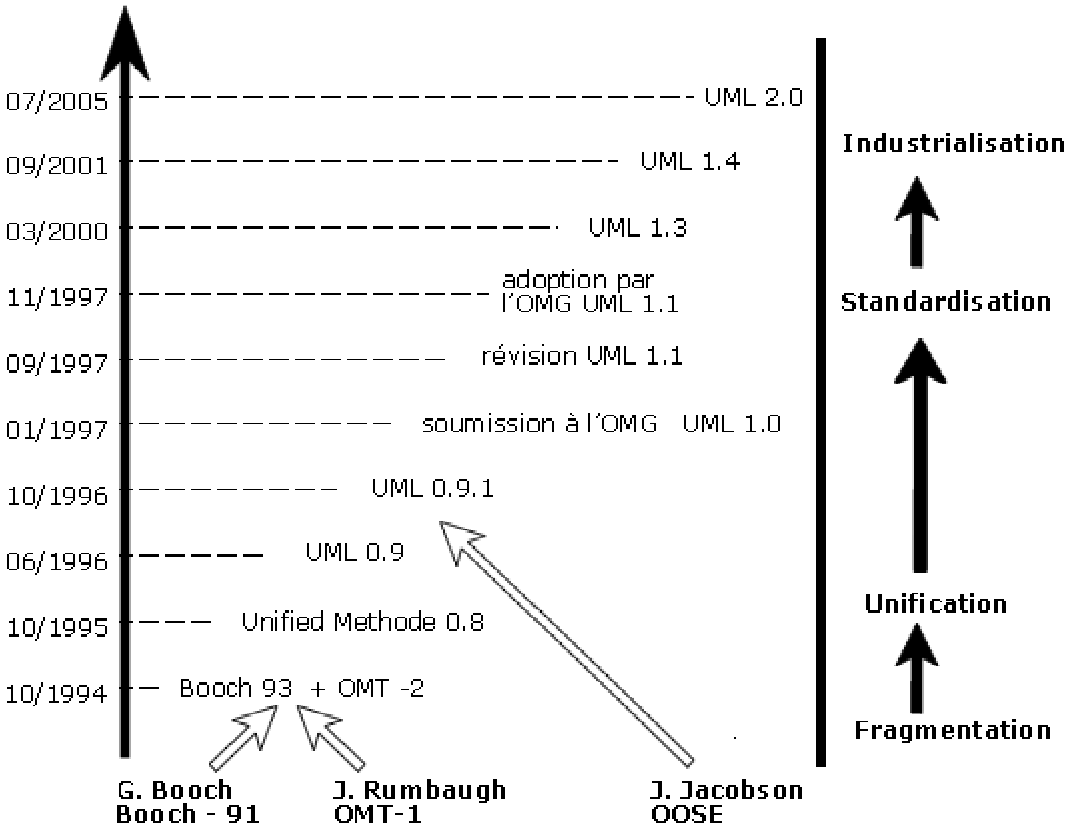



**Figure 3.1**
*Historique d'UML*

UML est un langage de spécification, de représentation graphique, de construction et de documentation des artefacts des systèmes logiciels ainsi qu'un langage de modélisation métier et de systèmes non logiciels [OMG01]. UML est adapté à la modélisation de systèmes, depuis les systèmes informatiques d'entreprise jusqu'aux applications distribuées basées sur le Web , en passant par les systèmes temps réel embarqués. [BooRumJac02]

UML unifie les notations nécessaire 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 des besoins jusqu'au codage.

*Pourquoi la modélisation ?*

> *Le fait de connaître un langage de programmation orienté objet comme Java est nécessaire mais insuffisant pour créer des systèmes à objets, il faut en outre savoir « penser en objet » [Larman02]*

> Les modèles permettent de mieux comprendre le système que l'on développe *[BooRumJac02]*

Un éditeur de solutions informatiques qui adopte des méthodes de travail fondées sur des techniques de conception solides rencontrera probablement du succès, car ses produits répondrons au mieux aux attentes des utilisateurs et constituerons des versions stables, évolutives et modulaires.

Les projets informatiques échouent chacun à leur manière, mais ceux qui réussissent présentent tous des caractéristiques communes. Ainsi, de nombreuses raisons expliquent le succès de certains fabricants de logiciels, mais le recours à la modélisation est un élément commun à tous. [BooRumJac02]

> Un modèle est une simplification de la réalité *[BooRumJac02]*

UML n'est qu'un langage de modélisation, il ne constitue qu'une partie d'une méthode de développement logiciel. UML est indépendant des processus, même si, idéalement, il doit être utilisé dans le cadre d'un processus fondé sur les cas d'utilisation, centré sur l'architecture, itératif et incrémental. [BooRumJac02]

Le recours à la modélisation est depuis longtemps une pratique indispensable au développement logiciel, car un modèle sert à anticiper les résultats du codage, il constitue une représentation abstraite d'un système qui permet d'en faciliter l'étude et de le documenter. Chaque intervenant dans le processus de développement contribue et enrichit le modèle différemment. La modélisation synthétise les systèmes et facilite leur compréhension et permette la réalisation de projet volumineux sans perdre le contrôle de leur gestion et évolutivité. La modélisation permet entre autre de responsabilisé clairement les entités du système en représentant leurs relations avec les autres entités et leurs rôles dans des contextes différents.

> 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 modèle doit présenter le contexte étudié, et non une vue plus globale qui, sans doute, va l'encombrer. Prenez l'exemple d'une carte routière où il est inutile d'utiliser la carte d'un payé pour circuler dans une de ses villes, par contre cette utilité ce manifeste s'il s'agit de circuler au niveau national. « A chaque voyage correspond une bonne carte »

La modélisation permet de formuler au mieux notre perception du monde réel en employant des abstractions orientées entités-métiers. Ainsi ces abstractions fusionnent toute notre connaissance du système et représentent notre compréhension d'une manière claire et exploitable.

La modélisation peut être bénéfique pour n'importe quel projet, parce que tout les systèmes utiles et intéressants tendent à devenir plus compliqués avec le temps et, même s'il semble inutile de les modéliser aujourd'hui, cette décision pourra être mise en cause et regrettée demain quand ces systèmes auront évolué.

La modélisation permet d'atteindre quarte objectifs : [BooRumJac02]

- Les modèles aident à visualiser un système tel qu'il est ou tel que nous voudrions qu'il soit.
- Les modèles permettent de préciser la structure ou le comportement d'un système.
- Les modèles fournissent un canevas qui guide la construction d'un système.
- Les modèles permettent de documenter les décisions prises.

> Nous construisons des modèles pour les systèmes complexes parce que nous ne somme pas en mesure d'appréhender de tels systèmes dans leur intégralité. *[BooRumJac02]*

Grâce à la modélisation il est devenu possible d'envisager la réalisation des systèmes complexes sans autant perdre le contrôle, ceci est du à sa capacité de fractionner la complexité dans des entités élémentaires moins complexes et plus faciles à cerner par les facultés de compréhensions humaine.

### 3.2 Les principes de la modélisation

L'histoire de la modélisation dans tous les domaines technologiques est riche et l'expérience acquise dicte quatre principes de base : [BooRumJac02]

1. *Le choix des modèles à créer a une très forte influence sur la manière d'aborder un problème et sur la nature de sa solution :* En d'autres termes, les modèles doivent être choisis avec soin car, s'ils sont bons, ils feront ressortir les problèmes de développement les plus ardus et apporteront un éclaircissement inespéré, alors que s'ils sont mauvais, ils détourneront l'attention des développeurs des questions fondamentales.
2. *Tous les modèles peuvent avoir différents niveaux de précision :* Dans certains cas, le modèle exécutable simple et rapide d'une interface utilisateur est exactement ce qui convient, dans d'autres, il faut descendre au niveau à des niveaux beaucoup plus bas pour comprendre. Les meilleurs modèles sont ceux qui laissent le choix du niveau de détail, en fonction du point de vue envisagé. Un analyste ou un utilisateur final se concentrera sur les fonctions, alors qu'un développeur mettra l'accent sur leur réalisation. Tous veulent visualiser le système avec une précision variable et à des phases de développement différentes.
3. *Les meilleurs modèles ne perdent pas le sens de la réalité :* Il est donc préférable de disposer de modèles qui restent en contact avec la réalité, un décalage entre le modèle analytique et le modèle conceptuel d'un système peut engendrer un décalage fondamental entre la conception et la réalisation effective.
4. *Parce qu'aucun modèle n 'est suffisant à lui seul, il est préférable de décomposer un système important en un ensemble de petits modèles presque indépendants :*pour comprendre l'architecture d'un système, plusieurs perspectives complémentaires et imbriquées sont nécessaires.

### 3.3 Pourquoi l'approche Objet ?

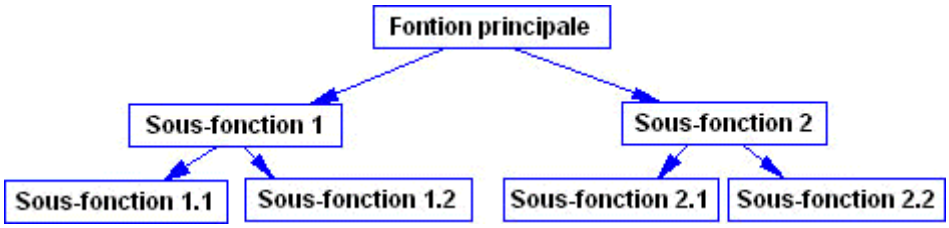
En quoi l'approche objet est-elle tellement attractive ? A cette question, les adeptes de l'objet répondent invariablement que les avantages de l'approche objet sont la stabilité de la modélisation par rapport aux entités du monde réel, la construction itérative facilitée par le faible couplage entre composants et la possibilité de réutiliser les éléments d'un développement à un autre. Tous ces points sont parfaitement exacts, mais ils ne sont que la conséquence de la fantastique capacité d'intégration ( ou d'unification) de l'approche objet.

En informatique, la modélisation peut être abordée sous plusieurs angles. Les deux principaux sont la perspective algorithmique et la perspective orientée objet. [BooRumJac02]

La construction d'un logiciel est, par conséquent, une suite d'itérations du genre division-réunion ; il faut décomposer - diviser - pour comprendre et il faut composer - réunir - pour construire. Cela conduit à une situation paradoxale : il faut diviser pour réunir.

Face à ce paradoxe, le processus de décomposition a été dirigé traditionnellement par un critère fonctionnel. Les fonctions du système sont identifiées, puis décomposées en sous-fonctions, et cela récursivement jusqu'à l'obtention d'éléments simples, directement représentables dans les langages de programmation (par les fonctions et les procédures).

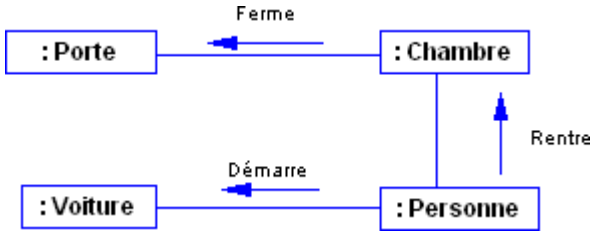
**Figure 3.2**
*Décomposition fonctionnelle*




L'architecture logicielle est alors le reflet des fonctions du système. Cène démarche, dont les mécanismes intégrateurs sont la fonction et la hiérarchie, apporte des résultats satisfaisants lorsque les fonctions sont bien identifiées et lorsqu'elles sont stables dans le temps. Toutefois, étant donné que la fonction induit la structure, les évolutions fonctionnelles peuvent impliquer des modifications structurelles lourdes, du fait du couplage statique entre architecture et fonctions.

L'approche objet repose à la fois sur le rationalisme d'une démarche cartésienne et sur une démarche systémique qui considère un système comme une totalité organisée, dont les éléments solidaires ne peuvent être définis que les uns par rapport aux autres. Elle propose une méthode de décomposition, non pas basée uniquement sur ce que le système fait, mais plutôt sur l'intégration de ce que le système est fait.

**Figure 3.3**
*Décomposition Objet*




Initialement basée sur l'utilisation d'objets en tant qu'abstraction du monde réel, l'approche objet a pour but une modélisation des propriétés statiques et dynamiques de l'environnement dans le quel sont définis les besoins, appelé le domaine du problème ou le champ d'application. [MullerGaertner00]. Elle traduit notre perception du monde réel et met en perspective des solutions rapprochées le plus possible de la logique qui définie la structure des objets et leurs relations entre eux mais aussi avec tout les intervenants.

Lorsque le problème est compris et la conception de notre système informatique est achevée, le passage à l'implémentation informatique de la solution (codage) devient facile et « mécanique » . La construction d'un système apparaît donc comme une démarche d'intégration d'un ensemble de composants

harmonieusement organisés sous une architecture progressivement complexe et qui reflète notre perception rapproché du monde réel. [MullerGaertner00]

En informatique, l'approche orientée objet est largement adoptée, tout simplement parce qu'elle a démontré son efficacité lors de la construction de systèmes dans les domaines métier les plus divers et qu'elle englobe toutes les dimensions et tous les degrés de complexité. De plus, la plupart des langages, des systèmes d'exploitation et des outils modernes sont au moins partiellement orientés objet, ce qui incite d'autant plus à voir les choses en termes d'objets.

### 3.4 Les objets

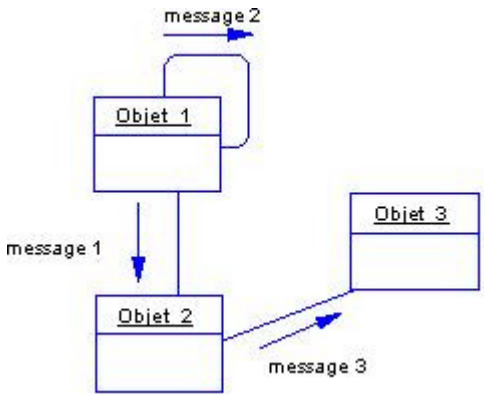
L'objet est une unité atomique formée de l'union d'un état et d'un comportement. Il fournit une relation d'encapsulation qui assure à la fois une cohésion interne très forte et un faible couplage entre le dit objet et l'extérieur. L'objet révèle son vrai rôle et sa vraie responsabilité lorsque, par l'intermédiaire de l'envoi de message, il s'insère dans un scénario de communication [MullerGaertner00]

*Objet = Etat + Comportement + Identité*

- **L 'état :** regroupe les valeurs instantanées de tous les attributs d'un objet, par exemple dans le cas d'un objet Moto, celui-ci pourrait être caractérisé par les attributs suivants : la marque, la couleur, le poids, la puissance, la quantité d'essence...
- **Le comportement** d'un objet indique toutes les compétences de celui-ci et décrit les actions et les réactions qu'il peut avoir. Chaque élément de base du comportement est appelé opération. Les opérations d'un objet sont déclenchées suite à une stimulation externe de l'utilisateur qui appuie sur un bouton ou du programmeur qui appelle une opération. Il existe un lien très étroit entre le comportement d'un objet et son état. Effectivement, l'état d'un objet peut être modifié par l'appel d'une opération et les actions effectuées par une opération dépendent de l'état de l'objet. Reprenons notre exemple de l'objet Moto. Nous pourrions définir les opérations Démarrer, Rouler, Stopper. L'opération Démarrer n'a de sens que si l'information "moteur éteint" est vérifiée. De même, l'opération Stopper n'aura d'effet que si les opérations Démarrer et Rouler ont déjà été exécutées modifiant ainsi l'état de l'objet moto de "moto stoppée" à "moto en déplacement".
- **L 'identité** : En plus de son état et de son comportement, un objet possède une identité qui caractérise son existence propre. L'identité permet de distinguer tout objet de façon non ambiguë, et cela indépendamment de son état. Cela permet, entre autres, de distinguer deux objets dont toutes les valeurs d'attributs sont identiques. L'identité est souvent construite à partir d'un identifiant issu naturellement du domaine du problème. Ainsi une moto est identifiée par son numéro de série, un compte en banque par son numéro de compte. [MullerGaertner00]

Les objets communiquent à travers les `messages`, un message est le support d'une relation de communication qui relie, de façon dynamique, les objets qui ont été séparés par le processus de décomposition. Il permet l'interaction de manière flexible et c'est lui qui assure la délégation des tâches et garantit le respect des contraintes.

**Figure 3.4**
*Communication des objets par des messages*




### 3.5 Les diagrammes UML

Un diagramme donne à l'utilisateur un moyen de visualiser et de manipuler des éléments de modélisation. UML s'articule autour de plusieurs types de diagrammes, chacun d'eux étant dédié à la représentation des concepts particuliers d'un système logiciel. UML modélise le système suivant trois mode de représentation, la figure 3.5 montre ces trois vues de modélisation. [Roques02]

**Figure 3.5**
*Les diagrammes UML par axe de modélisation*

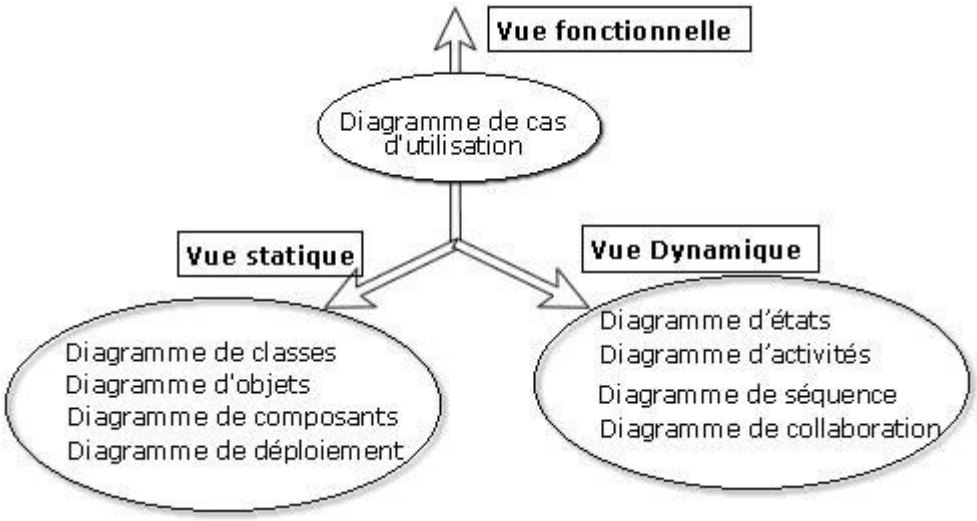
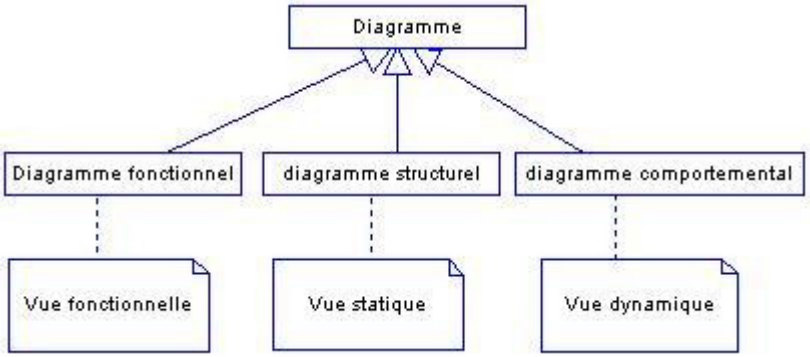



**Figure 3.6**
*Héritage des diagrammes UML*



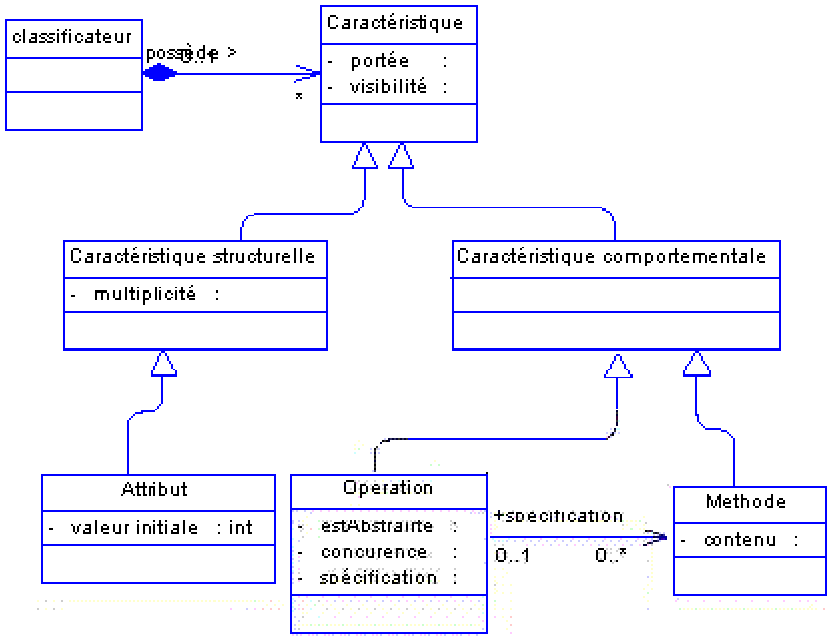
### *3.5.1 Vue fonctionnelle :*

- Décrit les services et les fonctionnalités assurées par le système, elle est représentée par le diagramme de cas d'utilisation :
    - **Diagramme de cas d'utilisation :** représentation des fonctions du système du point de vue de l'utilisateur, il est utilisé dans phase d'analyse des besoins, les cas d'utilisation servent à saisir le comportement attendu d'un système en cours de développement sans avoir à préciser la façon dont ce comportement est réalisé. Les cas d'utilisation permettent aux développeurs d'atteindre une compréhension commune avec les utilisateurs finaux du système et les experts du domaine. Les cas d'utilisations bien structurés représentent uniquement le contexte étudié il ne doivent pas être trop généraux, ni trop spécifiques. Le diagramme des cas d'utilisation constitue l'apport original de la méthode OOSE (I.Jacobson, 1992) à la notation UML.

### *3.5.2 Vue statique :*

- **Diagramme de classes :** est le diagramme le plus utilisé dans la modélisation orienté objet et représente le point centrale dans un développement orienté objet. Il décrit la structure des entités manipulées par les utilisateurs. Il représente un ensemble de classes, d'interfaces et de collaborations ainsi que leurs relations. Le diagramme de classes est important non seulement pour visualiser, construire, spécifier et documenter des modèles structurels, mais également pour construire des systèmes exécutable grâce au reverse engineering. [BooRumJac02]

**Figure 3.7**
*Extrait du métamodèle. Caractéristiques de la classe*




- **Diagramme d'objets :** ou diagramme d'instance, montre des objets et des liens. Un diagramme d'objet est une instance d'un diagramme de classes qui montre l'état du système modélisé à un instant donné. La notation retenue

pour les diagrammes d'objets est ainsi dérivée de celle des diagrammes de classes.

- **Diagramme de composants :** un diagramme de composants décrit les composants et leurs dépendances dans l'environnement de réalisation. En pratique, les diagrammes de composant ne sont utilisés que pour des architectures complexes.

> Un composant est une partie physique remplaçable d'un système qui fournit la réalisation d'un ensemble d'interfaces et s'y conforme.[BooRumJac02]

- **Diagramme de déploiement :** Les diagrammes de déploiement montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels. Les ressources matérielles sont représentées sous forme de nœuds. Les diagrammes de déploiement peuvent montrer des instances de nœuds (un matériel précis), ou des classes de nœuds.

  Un **nœud** (*node*) est un objet physique correspondant à une ressource de traitement (mémoire et processeur). Il s'agit d'ordinateurs, de ressources humaines, de traitements mécaniques.

> Un diagramme de déploiement est un diagramme qui montre la configuration de nœuds de traitement en phase d'exécution ainsi que les composants qui se trouvent sur ces nœuds. [BooRumJac02]

  Pour certains types de systèmes, le diagrammes de déploiement ne sont pas nécessaire. Si l'on développe un logiciel qui se trouve sur une seule machine et n'est en interface qu'avec des périphériques standards déjà gérés par le système d'exploitation hôte ( exp : clavier, modem ) on peut se passer des diagrammes de déploiement. En revanche, si l'on développe un logiciel en interaction avec des périphériques que le système d'exploitation de l'hôte ne gère pas habituellement ou qui est réparti physiquement sur plusieurs processeurs, l'utilisation de diagrammes de déploiement permet alors de réfléchir au mapping entre le logiciel et le matériel du système. [BooRumJac02]

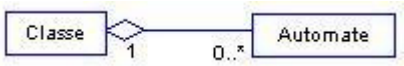
### *3.5.3 Vue dynamique :*

- La phase d'analyse dynamique intervient après la phase d'analyse statique, elle s'appuie sur les objets décrits dans la vue statique, ainsi que sur les événements provenant de l'extérieur du système décrits dans la vue fonctionnelle, La modélisation dynamique est plus difficile à mettre en œuvre que la modélisation statique car elle nécessite une réelle compréhension du fonctionnement global du système.

  - **Diagramme d'états-transitions :** est un automate à états finis, composé d'états, de transitions, d'événements et d'activités. Ils présente la vue dynamique d'un système, particulièrement important dans la modélisation du comportement d'une interface, d'une classe ou d'une collaboration et met l'accent sur le comportement d'un objet ordonnancé par les événements, ce qui est particulièrement utile dans la modélisation des systèmes réactifs.

Le diagramme d'états-transitions représente le cycle de vie commun aux objets d'une même classe. Ce diagramme permet de compléter la connaissance des classes en analyse et en conception.

**Figure 3.8**
*Un ou plusieurs automates peuvent être associés à chaque classe du modèle.*




- **Diagramme d'activités** : est une variante des diagrammes d'états-transitions qui décrit la succession des activités au sein d'un système. Ils présentent la vue dynamique d'un système ; ils sont particulièrement importants dans la modélisation de la fonction d'un système et mettent l'accent sur le flot de contrôle entre les objets. Dans un diagramme d'états-transitions, les états et les transitions sont mis en avant alors que dans un diagramme d'activités, ce sont les activités et les transitions qui sont mises en avant.
  Les diagrammes d'activités utilisent beaucoup d'éléments du métamodèle définis pour les diagrammes d'états-transitions.
  Un diagramme d'activités représente l'état de l'exécution d'un mécanisme, sous la forme d'un déroulement d'étapes regroupées séquentiellement dans des branches parallèles de flot de contrôle. Le début et la fin (si elle existe) d'un mécanisme sont définis respectivement par un état initial et un état final. [MullerGaertner00]

- **Diagramme de séquence** et **Diagramme de collaboration :** sont deux types de diagrammes d'interaction. Les *diagrammes d'interaction* représentent une interaction, c'est-à-dire un ensemble d'objets et leurs relations, y compris les messages qu'ils peuvent s'échanger. Ils présentent la vue dynamique d'un système. Les *diagrammes de séquence* sont des diagrammes d'interaction qui mettent l'accent sur le classement chronologique des messages alors que les *diagrammes de collaboration* sont des diagrammes d'interaction qui mettent l'accent sur l'organisation structurelle des objets qui envoient et reçoivent des messages. Les diagrammes de séquence et les diagrammes de collaboration sont isomorphes, c'est-à-dire que l'un peut être transformé en l'autre.

### 3.6 Nouveauté de UML 2.0

Avec la nouvelle version d'UML ( UML 2.0 ) 4 nouveaux diagrammes viennent compléter la collection des diagrammes UML, ces diagrammes sont :

- *Diagramme de packages* : permet de représenter la hiérarchie des modules du projet, leur organisation et leurs interdépendances. Cela simplifie les diagrammes, et les rend donc plus simple à comprendre
- *Diagramme de structure composite :* permet de décrire la structure interne d'un objet complexe lors de son exécution (au *run-time* - décrire l'exécution du programme), dont ses points d'interaction avec le reste du système.
- *Diagramme global d'interaction* : permet d'associer les notations du diagramme de séquence à celle du diagramme d'activité, ce qui permet de décrire une méthode complexe. C'est une variante du diagramme d'activité.

- *Diagramme de temps* : permet de modéliser les contraintes d'interaction entre plusieurs objets, comme le changement d'état en réponse à un évènement extérieur.

### 3.7 UML est un langage pour communiquer

Souvent dans le jargon des programmeurs on entends l'expression « telle personne parle bien UML » c'est parce que UML est un moyen de communication très efficace et utilisé à grande échelle dans la communauté des développeurs informatiques. Il substitue largement au langage naturel ( parlé ) ceci est du à sa simplicité, sa capacité à présenter des systèmes assez complexes sur des schémas expressifs. D'ailleurs les fondateurs d'UML le qualifie comme un langage pour visualiser [BooRumJac02].

UML comporte un riche vocabulaire et une arsenal de règles régissant la manière de concevoir et de comprendre des modèles correctement structurés. UML est bien plus qu'un simple ensemble de symboles graphiques, en fait chaque symbole véhicule une sémantique stricte. Cette rigueur permet entre autre de :

- Rendre son modèle universel : la maîtrise d'UML permet de lire n'importe quel modèle respectant les règles définies dans le standards OMG.
- L'utilisation des outils de génération automatique du code natif depuis les schéma UML.
- Reverse engineering

### 3.8 Etude basée sur les cas d'utilisation

Le principe des cas d'utilisation a été introduit en 1986 par Ivar Jacobson [Jacobson92] il permet de décrire des besoins fonctionnels, cette technique a rapidement trouvée sa place dans le milieu des concepteurs d'applications et même devenue une école, ceci et du à sa simplicité et surtout son utilité [Larman2002]. Parmi tout les travaux réalisés dans ce domaine, on trouve l'œuvre de Alistair Cockburn, qui constitue une référence dans la matière avec son exhaustivité « Rédiger des cas d'utilisation efficaces » [Cokburn01]

« Un cas d'utilisation établit, entre les différents intervenants, un contrat régissant le comportement d'un système. Il décrit ce comportement sous diverses conditions » [Cokburn01] cette définition ne laisse penser que les cas d'utilisation est la première étapes du processus d'analyse, elle doit permettre de voir notre futur système sous plusieurs angles, mais tous centrés sur l'utilisateur final, ce dernier doit assister à la rédaction des cas d'utilisation car sa contribution est essentielle.

Ce qui différencie l'étude basée sur les cas d'utilisation et une autre qualifiée fonctionnelle c'est la manière d'aborder le problème pour arriver au résultat. Les cas d'utilisation visent les objectifs (utilisation effective du système) cependant l'étude fonctionnelle traite plus le processus ramenant au résultat, le modèle des cas d'utilisation permet d'aboutir à des réponses novatrices et optimisées aux besoins et non pas au processus, en plus concevoir une solution fonctionnelle nous pousse, sans conscience, à reproduire des systèmes existants ce qui présente une lacune majeure pour le profit de l'utilisateur final.

La détermination et la compréhension des besoins sont souvent difficiles car les intervenants sont noyés sous de grandes quantités d'information. Les besoins sont souvent exprimés de manière non structurée, sans forte. Les cas d'utilisation est une

perception de haut niveau du système, ils décrivent la relation de notre système avec le monde extérieur, sans se préoccuper des détails ou se laisser distraire par les contraintes technologiques.

> Les cas d'utilisation recentrent l'expression des besoins sur les utilisateurs, en partant du principe qu'un système est avant tout construit pour ses utilisateurs. [BooRumJac02]

Pour un système, il existe trois aspects importants : la fonction, l'interface et l'implémentation (c'est le cas d'un objet). La fonction justifie le pourquoi du projet, l'interface représente les voies de communication avec le système depuis l'extérieur pour exploiter ces services, tandis que l'implémentation est l'existence technique du système.

Une étude basée sur l'utilisateur focalise l'effort sur les deux premiers aspects c à d la fonction et l'interface car elle présente le seul intérêt pour l'utilisateur final. En plus le faite de les bien définir, permet de cerner le contexte et le champ d'application de notre système.

Tracer un diagramme des cas d'utilisation est une tâche souvent difficile et équivoque et pleine de pièges, pourquoi ? Les analystes et les utilisateurs n'ont pas forcément les mêmes définitions, ceci est dû au faite que chacun d'entre eux vise un objectif différent, mais il reste à dire que c'est le rôle de l'analyste de trouver un compromis et de piloter les discutions pour aboutir à des informations exploitables. Décrire un cas d'utilisation par un petit schéma conventionnel est très utile mais il reste insuffisant pour comprendre l'emplacement du cas d'utilisation dans le système et surtout dans le workflow. Ceci revient à définir certains éléments qui sont communs dans la plupart des cas d'utilisation [Cokburn01]

**Tableau 3.1**
*Description textuelle d'un Cas d'Utilisation*

| | |
|---|---|
| **Présupposés** | L'état dans le quel le système doit être avant d'appeler le cas d'utilisation, c à d tous les présupposés sont vérifiés.<br>Les présupposés, à la différence des pré-conditions, doivent être vérifier par d'autres cas d'utilisation. |
| **Pré-conditions** | L'état dans lequel le système doit être avant d'utiliser le cas d'utilisation. Mais contrairement aux présupposés, les pré-conditions sont vérifier par le cas d'utilisation, si les conditions ne sont pas remplies, l'acteur se voit refuser l'usage du cas. |
| **Démarrage** | Où et quand le cas démarrera ? Trouver les déclencheurs du cas est important pour l'aspect de réutilisation, chaque utilisation est particulièrement distinguée par une manière de déclencher le cas. |
| **Processus ou dialogue** | La description de la communication entre le cas d'utilisation et son utilisateur (acteur, cas d'utilisation) depuis la demande jusqu'à un point d'arrêt du cas. |
| **Arrêt** | Souvent le cas d'utilisation dispose de plusieurs signaux d'arrêt, lister toute les possibilités pour éviter d'oublier le traitement des exceptions et des erreurs. |

| | |
|---|---|
| **Post-condition** | L'état dans lequel le système doit se retrouver après la fin du cas d'utilisation. Ceci permet de vérifier la stabilité du système après l'achèvement de l'exécution du cas (normale ou anormale exp : Erreur). |

Notez bien, que cette description textuelle d'un cas d'utilisation ne fait pas partie du standard UML, en effet il s'agit plus d'une méthodologie de travail.

### 3.9 Le choix d'un outil de modélisation

Il existe de nombreux outils de modélisation graphique UML allant du plus simple jusqu'aux ateliers de génie logiciel. Mais l'OMG ne préconise aucun outil de modélisation UML, l'organisation précise que la plupart des outils du marché ont été conçus pour mettre en œuvre certaines méthodologies(langages), du fait ce n'est pas judicieux d'utiliser un outil et essayer ensuite de l'adapter à une méthodologie à la quelle il ne s'adhère pas [URL_OMG] car ceci peut constituer un grand handicape pour le processus de développement. Cependant il existe des outils qui intègrent plusieurs méthodologies (langages de programmation) il y a même d'autres qui permettent le passage d'une méthodologie à une autre.

L'OMG à pris cette position neutre pour éviter de créer une concurrence polémique non souhaitée entre éditeurs de solution software car elle va orienter la recherche en dépit de l'innovation.

Néanmoins, L'utilisation des outils de modélisation à montré qu'un certain nombre de critère sont très important dans l'évaluation d'un outil de modélisation UML :

- **Génération automatique du code** : permet de réduire visiblement la charge du développeur en créant –automatiquement – depuis des modèles UML des squelettes de code dans un langage précis.
- **Reverse engineering** : c'est l'opération inverse de la génératrice du code, elle consiste à créer des modèles UML à partir du code. Beaucoup d'outils proposent cette fonctionnalité mais peu sont ceux qui atteignent un niveau de succès acceptable. Le Reverse engineering est très utile aussi bien pour transformer le code dans un modèle quand aucun modèle n'a existé précédemment, que pour synchroniser de nouveau un modèle avec le code à la fin d'une itération.
- **Génération de la documentation** ( format HTML ou autre ) avec des images du modèle.
- **Compatibilité avec les versions UML** : avant de commencer à utiliser UML, il faut choisir une version ( 1.1, 1.3, 2.0 … ) ce critère va déterminer le choix de l'outil de modélisation car tout les outils ne supportent pas toutes les versions d'UML. Ou faire l'inverse c à d choisir l'outil suivant d'autres critères que la version d'UML et s'aligner ensuite à la version d'UML utilisée par cette outil.
- **Les Données d'Intégration du Modèle** : L'outil de modélisation d'objet doit permettre l'intégration avec des fonctionnalités de modélisation des données. Il y a beaucoup de façons de fournir ces fonctionnalités. L'une des possibilités est que l'outil UML comprenne une fonction qui permet au modèle d'objet d'être transformé en DDL. DDL est le SQL nécessaire à la création de tables pour les classes. L' outil UML peut également exporter les méta-data vers l'outil de modélisation des données qui peut les importer et les utiliser comme base d'un modèle de données. Un ensemble d'outils

intégré avancé doit permettre la synchronisation des modèles de données et des modèles d'objet après chaque itération du dessin.

- **Gestion des versions** : permettre la sauvegarde de plusieurs version du modèle et la possibilité de naviguer entre ces versions.
- Ergonomie conviviale et interface intuitive.
- **Plate-forme** : Afin d'optimiser votre investissement dans un outil de modélisation, considérez attentivement les plates-formes sur lesquelles s'exécutera cet outil. Allez-vous développer des produits pour Windows, pour Unix ou pour les deux ? Sur quelle plate-forme développerez-vous ?
- **Exportation vers le format XMI :** XMI standard de l'OMG est l'un des développements récents les plus passionnant pour la communauté des développeurs de l'UML. XMI est un format d'échange qui a la capacité d'autoriser le partage en continu des modèles entre les outils de développement les meilleurs de la nouvelle génération. Par exemple, plutôt que d'écrire les scripts dans un outil de modélisation UML pour créer des rapports, un utilisateur pourrait exporter le modèle en cours de conception en utilisant XMI et simplement importer le modèle dans un outil spécialisé en écriture de rapports.

Pour finir voici une liste, non exhaustive, d'outils de modélisation :

**Tableau 3.2**
*Outils de modélisation UML*

| **Produit** | **Description** |
|---|---|
| Rational Rose<br>Société :Rational software Corporation<br>URL : http://www.rational.com | Rational Rose est une des leaders mondiaux des outils de modélisation UML. La société Rose a dans ces membres les pères des méthodes objet : Rumbaugh, Jacobson, Booch. Rose est orienté programmation |
| PowerDesigner<br>(PowerAMC version française)<br>Société :Sybase<br>URL :http://www.sybase.com/ | Version d'évaluation 1 mois.<br>Outil professionnel très puissant et très complet de modélisation de données jouissant d'une très bonne réputation. Support de multiples modèles(OO), modèle conceptuel, modèle physique, traduction de modèle, reverse engineering de base de données et de programme. UML à partir de la version 7.<br>Prix (approx.) : 5000 €. |
| Visio<br>Société : Microsoft<br>URL :http://www.microsoft.com/office/visio | Gamme de produit qui va du simple diagramme (Standard) à un véritable outil de modélisation de base de données et pour la conception de programmes (entreprise). Fournit un environnement de développement Visual Basic pour application ce qui permet de le personnaliser facilement. Support du reverse-engineering, génération de rapports. Supporte tous les diagrammes UML |

| Produit | Description |
|---|---|
| PoseidonPE<br>Société : gentleware<br>URL :http://www.gentleware.com | Outil de modélisation UML, génération de code java, export XMI, interface agréable et intuitive, version CE gratuite, supporte les 9 diagrammes de la norme UML. Néanmoins, les diagrammes d'objets, de composants et de déploiement sont tous rassemblés dans un seul éditeur ("Deployment diagram") car les éléments de notation graphique sont similaires |
| Oracle Designer 2000<br>Société : Oracle<br>URL : http://www.oracle.com/ | Outil très complet de développement. Support du formalisme UML. La version complète permet de couvrir tous les besoins de modélisation de base de données, programmation. |
| Perceptory<br>Société : Université Laval<br>URL :http://sirs.scg.ulaval.ca/perceptory | Nécessite Visio (sauf la version Java). Simple, gratuit et très complet. Outil spécifiquement développé pour la modélisation de données géographiques. Développement possible avec Visual Basic pour Application ou d'autres langages. Code disponible. Outil gratuit. Génération de dictionnaire de données. Génération de rapport compatible ISO TC211. Diagramme de classe UML |
| ArgoUML<br>Société : Tigris<br>URL :http://www.tigris.org | Outil de modélisation UML programmé en Java. Il s'agit d'un produit Open Source. |
| MyUML<br>Société : Softwarefarm<br>URL : http://www.swfm.com | Outil de modélisation UML en Java. Version gratuite disponible. Utilisation de XMI pour import et export. Support de Java JDK versions 1.1.x et 1.2.x. |
| Objecteering<br>Société : Softeam<br>URL :http://www.objecteering.com | Atelier de modélisation UML, un outil puissant |
| SmartDraw<br>Société : Hemera Technologies Inc<br>URL :http://www.smartDraw.com | Outil de modélisation UML, ainsi que d'autre fonction de présentation graphique. L'originalité de cet outil c'est l'existence d'une bibliothèque dédiée à la modélisation WEB |
| Enterprise Architect<br>Société : Sparx Systems<br>URL :http://www.sparxsystems.com | Outil de modélisation UML très puissant et très complet. |

| Produit | Description |
|---|---|
| ModelMaker<br>URL :http://www.modelmakertools.com | ModelMaker est maintenant directement intégré dans Borland Delphi Entreprise, exclusivement conçu pour la modélisation UML pour Delphi, synchronisation bidirectionnelle entre le modèle UML et le code Delphi, Reverse engineering. |
| Borland Together<br>Borland<br>URL :<br>http://www.borland.com/fr/products/together | Environnement de conception agile et intégré, Synchroniser le modèle et le code source du projet, Générer la documentation de l'application, S'interfacer avec tous les EDI majeurs du marché (C++BuilderX, Eclipse, JBuilder, ect) |
| Visual Paradigm<br>http://www.visual-paradigm.com/ | Atelier de génie logiciel, riche en fonctionnalités et c'est aussi un outil de conception UML facile à utiliser qui dispose d'excellentes possibilités d'interopérabilité avec d'autres applications. Son environnement de développement intelligent (SDE) est disponible dans de nombreuses versions, chacune étant conçue pour s'intégrer aisément à un EDI. La version d'évaluation contient la version Community Edition qui est gratuite. |

### 3.10 UML et la Méta-modélisation

Le développement dirigée par les modèles, est une approche de conception qui vise à garder les modèles comme point de référence tout au long du processus de développement logiciel. L'OMG cherche à apporter de nouvelles fonctionnalités aux différents modèles qui sont utilisés pour concevoir des applications, à faciliter la création de nouveaux espaces de modélisation plus adaptés aux besoins de leurs utilisateurs, et à faciliter les différentes étapes de modélisation nécessaires à l'élaboration d'un produit fini.

Un des points clés dans cette approche est la possibilité de transformer les modèles, d'un espace de modélisation ou d'un niveau d'abstraction vers un autre. En effet, la gestion manuelle des modèles s'avère aujourd'hui coûteuse et les modèles sont en conséquence bien souvent laissés de côté. Il devient donc nécessaire de proposer des outils flexibles de gestion automatique des modèles, permettant, tout au long de leur cycle de vie, de les porter aisément sous des formes dans lesquelles ils répondront mieux aux attentes de leurs utilisateurs à un moment donné. Cette partie du chapitre tente d'établir un état de l'art sur les outils de transformations de modèles, en analysant en particulier les possibilités qui ont déjà été mises en œuvre à l'heure actuelle.

#### *3.10.1 Les besoins liés aux modèles :*

Depuis longtemps, divers langages et techniques de modélisation sont utilisés lors des phases d'analyse et de conception, entre autres en développement logiciel. Se situant à un niveau intermédiaire entre l'expression des besoins et le code développé pour répondre à ces besoins, ils permettent une meilleure

communication entre informaticien et client. Du point de vue du développeur, ils permettent de garder tout au long du cycle de vie du logiciel une vision globale sur le système à traiter, en offrant plusieurs niveaux d'abstraction.

De ce point de vue, le langage UML offre une gamme d'outils étendue, permettant de représenter sous forme graphique divers aspects d'un système à traiter, notamment l'organisation des données, les flots de contrôle, les interactions entre plusieurs modules, l'organisation des applications réparties.

Pourtant, ce standard souffre actuellement de nombreuses critiques : orienté vers un développement objet, basé sur un formalisme relâché, il ne permettrait pas de tenir compte de la diversité des plates-formes logicielles, ni d'exprimer certains points clé de la définition d'un système.

En effet, les différentes phases du cycle de vie du logiciel sont ponctuées de prises de décisions, et il apparaît nécessaire que les modèles fournissent une aide efficace dans ces décisions. Il s'agit notamment des questions liées au choix d'une plate-forme, de validation des choix de développement en amont de l'implémentation, de l'élaboration de données de tests, de vérification de composants.

Pour répondre à ces besoins et offrir une réelle assistance dans ces phases de décision, il est nécessaire qu'un modèle puisse être traité automatiquement. Cela permettrait notamment l'animation d'un modèle, la génération de données de tests, ou encore la simulation d'ajout de nouveaux composants, offrant ainsi un réel gain en temps et en moyens.

Les besoins étant très divers en matière de modélisation, il apparaît à l'heure actuelle impossible de définir un langage de modélisation permettant de répondre à tous ces besoins de manière équitable. En réalité, des outils et espaces de modélisation variés sont utilisés et continueront à l'être en fonction des besoins qui interviennent lors du cycle de développement, et du domaine métier concerné. De même, les phases de modélisation passent par divers niveaux d'abstraction, et le retour sur les choix pris à un niveau précis est très coûteux : de nombreuses améliorations pourraient être apportées de ce point de vue. En conséquence, pour rendre la démarche de modélisation productive, il est nécessaire de fournir des outils de transformation permettant de passer d'un espace de modélisation ou d'un niveau d'abstraction à un autre. [TATIBOUET06]

Tout comme un programme, un modèle destiné à être traité par une machine ne doit pas permettre d'ambiguïtés d'interprétation, et doit être exprimé en respectant des règles bien définies. Il s'agit là de l'objet de la démarche de méta-modélisation, qui s'attache à définir des formalismes pour les langages de modélisation. Une fois le formalisme défini, on peut garantir que tout modèle conforme à ce formalisme pourra être traité correctement.

Il s'agit ici de bien distinguer les différents niveaux d'abstraction liés au concept de méta-modélisation, qui sont résumés dans le tableau figure 3.9.

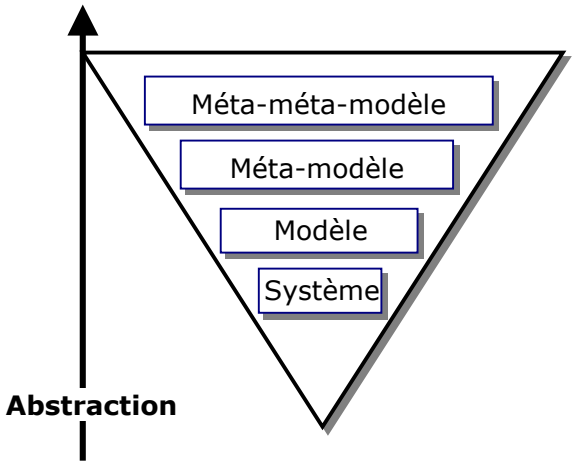



**Figure 3.9**
*Niveaux d'abstraction de la méta-modélisation*

- Au premier niveau, se trouve le système étudier : il peut s'agir selon le cas du système d'information d'une entreprise, ou des tâches que l'on souhaite automatiser.
- Au second niveau, se trouve le modèle qui décrit certains aspects du premier niveau que l'on veut étudier : il peut s'agir d'un diagramme de classes UML, d'un modèle conceptuel de traitement MERISE, d'une machine B, ou de tout schéma qui représente une vue abstraite des objets modélisés.
- Au troisième niveau se trouve le langage de modélisation ou méta-modèle : par exemple les définitions d'un MCD MERISE d'un diagramme d'objet UML ou le langage B. C'est ce langage qui doit faire l'objet d'une spécification formelle.
- Au quatrième niveau se trouve le méta-méta-modèle : il s'agit d'un langage qui doit être assez générique pour définir les différents langages de modélisation existants, et assez précis pour exprimer les règles que chaque langage doit respecter pour pouvoir être traité automatiquement.

Dans l'architecture proposée par l'OMG, UML se situe au troisième niveau, et un langage générique de description des modèles est proposé : le MOF (Meta Object Facility), que nous décrivons dans les pages qui suivent.

Dans le formalisme XML, on pourrait situer le langage des DTD ou des schémas au quatrième niveau, puisqu'il s'agit de langages servant à décrire un format de document. Une DTD ou un schéma particulier se placerait alors au niveau 3, décrivant le format que doit respecter un certain type de document. Un document particulier se situerait alors au niveau 2, conforme au précédent et représentant certaines données d'un domaine particulier. Comme nous l'avons vu dans le chapitre 2, XML étant lui-même utilisé en tant que langage de description de modèle, ces niveaux peuvent varier suivant l'utilisation qu'on en fait. L'établissement d'un méta-modèle est une condition nécessaire au traitement automatisé. Le méta-modèle doit définir une syntaxe qui permettra à un programme non seulement de vérifier qu'un modèle est décrit correctement, mais surtout d'identifier les différents éléments de ce modèle afin de leur appliquer un certain traitement. [TATIBOUET06]

C'est notamment sur un méta-modèle que s'appuiera la définition de transformations. Par exemple, si on désire obtenir à partir d'un modèle objet son équivalent dans le formalisme relationnel, le programme qui réalise la transformation doit être capable d'identifier chaque élément du modèle source pour générer son équivalent dans le modèle cible. Il est donc nécessaire d'exprimer

formellement les langages de définition des deux modèles source et cible, afin qu'ils soient reconnus par notre programme.

l'OMG a proposé trois standards qui sont respectivement le MOF (langage permettant de décrire des métamodèles), le format XMI (XML Metadata Interchange) et QVT (Query / View / Transform).

### *3.10.2 MOF*

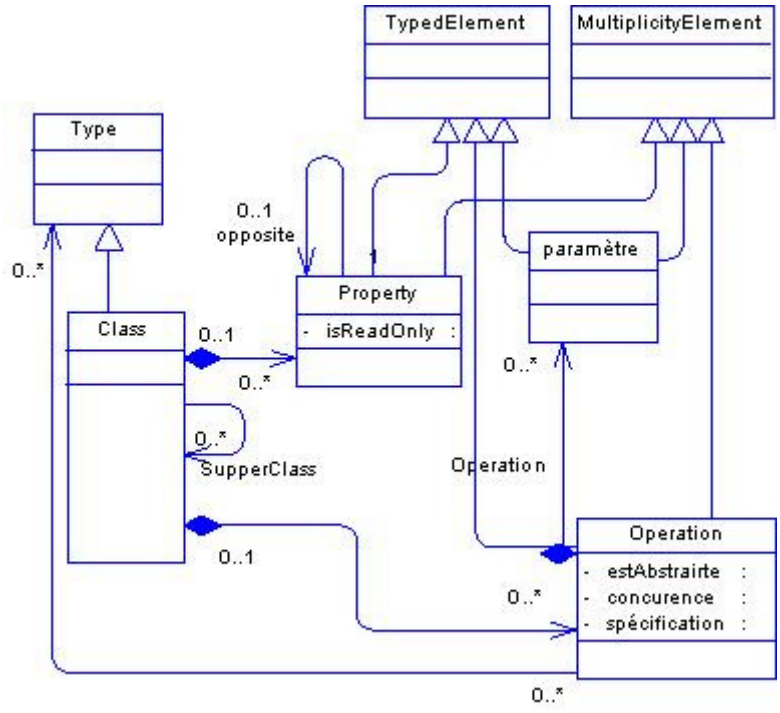
Le MOF (Meta Object Facility) est une extension du modèle objet, qui permet à celui-ci de représenter non plus des entités du monde réel, mais des entités de description de modèles. La figure 3.10 extraite des spécifications MOF 2.0 présente la partie du méta-modèle sur laquelle s'appuient les procédés de transformation.

On retrouve le concept de classe, qui représente ici un élément du méta-modèle décrit. Les relations entre ces éléments sont exprimées par la référence «opposite» de la classe « Property » : pour dire qu'il existe une relation bidirectionnelle entre une classe A et une classe B, la représentation de chaque classe possédera une propriété dont l'attribut « opposite » fera référence à l'autre classe.

Si l'on veut par exemple représenter une partie du méta-modèle UML en MOF, les concepts de classe ou d'attribut seront chacun représentés par une classe du MOF, sachant qu'ils représentent une entité de description de modèles. Une association entre une classe et les attributs qu'elle possède sera représentée dans la classe « Class » par une propriété dont l'attribut « opposite » fera référence à la classe «Attribute».

Le MOF est donc un langage de description des langages de modélisation (ou métamodèles). En utilisant des éléments du MOF, on peut décrire un langage tel que UML, le langage de description des bases de données relationnelles, XML ou encore le MOF lui-même. Ce langage permet d'abord de faire une représentation graphique d'un langage de modélisation particulier. Mais surtout il définit une grammaire selon laquelle un programme peut reconnaître et traiter des éléments d'un modèle décrit dans le langage spécifié.

**Figure 3.10**
*Package EMOF (MOF 2.0)*



### *3.10.3 XMI*

Le MOF restant avant tout une représentation graphique, il est nécessaire de disposer d'un format de stockage des méta-modèle MOF ou des modèles décrits avec un outil de modélisation quelconque, afin que ces modèles puissent être récupérés et traités par un programme.

Le standard XMI (XML Metadata Interchange) a été mis en place en 2000 pour permettre la sérialisation des modèles, afin de rendre possible l'échange de modèles entre différents logiciels (de modélisation, de développement).

Le standard XMI, est actuellement utilisé, sous de nombreuses variantes par des plates-formes de modélisation et de développement logiciel, notamment Rational Rose ,PowerAMC et Poseïdon pour permettre la transmission de modèles entre ces plateformes.

L'intérêt du standard XMI dans les outils de modélisation est qu'il propose un format idéal pour décrire à la fois les méta-modèles et les modèles conformes à ceux-ci, qui vont permettre à l'outil de traiter automatiquement les modèles à transformer, et de fournir en sortie un modèle directement exploitable par d'autres logiciels.

### *3.10.4 QVT :*

Les standards MOF et XMI étant largement utilisés dans le cadre de la méta-modélisation, ils ne suffisent toutefois pas à définir le procédé de transformation qui permet de passer d'un modèle à un autre sur la base des méta-modèles de départ et d'arrivée. En effet, des APIs ont été définies pour passer d'un formalisme à l'autre, mais l'enjeu des outils de transformation est de pouvoir transformer plusieurs classes de modèles, définies suivant plusieurs méta-modèles différents.

L'identification ces dernières années de nouveaux besoins en matière de modélisation donne lieu à une recherche active dans le domaine de la méta-modélisation. En effet, cette démarche offre la perspective de pouvoir opérer des traitements informatiques avancés sur les modèles, amenant ainsi une gestion plus efficace des ceux-ci, leur permettant d'occuper une place plus importante dans les procédés de développement logiciel. La possibilité de transformer des modèles est au centre des préoccupations de ceux qui traitent de méta-modélisation. Une partie importante des recherches qui se font sur ce thème est centrée sur l'appel à propositions QVT de l'OMG.

## 3.11 Les Design Patterns

Les *design patterns* sont apparus dans les années 1990. Les travaux de la « bande des 4 '» [Gamma 95] en ont produit le document fondateur. Cet ouvrage ainsi que des publications ultérieures constituent depuis des catalogues de référence pour les concepteurs objet.

### *3.11.1 Définition*

Plusieurs définitions nous aideront à comprendre ce concept :

« En génie informatique, un motif de conception (Design pattern en anglais) est un concept issu de la programmation orientée objet, destiné à résoudre les problèmes récurrents »

« Solution récurrente décrivant et résolvant un problème général dans un contexte particulier ».

« une solution de conception commune à un problème | récurrent dans un contexte donné [UML-UG 98] ».

«…, un Design Pattern décrit un problème récurrent dans un environnement donné puis décrit une solution à ce problème de telle manière que cette solution soit réutilisable à chaque fois qu'on rencontre le problème sans cependant qu'elle soit deux fois exactement la même. »

Le principal intérêt des patterns est de permettre la manipulation d'artefacts plus élaborés que les objets et les classes. Ils accroissent la force d'expression des langages de modélisation. Le concepteur d'un pattern propose une solution générique pour un problème, plutôt de détail, qui est souvent rencontré dans divers développements. Cette solution est présentée sous une forme indépendante d'un langage de programmation particulier »

L'usage des design patterns apporte donc évolutivité, lisibilité et efficacité aux développements. C'est pourquoi leur emploi améliore sensiblement le respect des prescriptions d'architecture [Bushmann 96]. Par ailleurs, ils offrent un transfert de compétence rapide en conception orientée objet, dans la mesure où ils représentent pour les débutants un catalogue des meilleures pratiques à adopter.

Un design pattern est généralement formé de quatre parties essentielles, à savoir :

- Le nom du *pattern,* point d'accroché qui permettra de facilement retrouver le problème et sa solution. Ce nom doit donc être choisi avec soin.
- Le problème lui-même, qui décrit le plus synthétiquement possible le contexte dans lequel s'applique ce design pattern.
- La solution, qui décrit l'ensemble des éléments en jeu, leurs relations et leurs responsabilités. La solution est exprimée d'une façon générique et ne repose sur aucun cas concret pour être facilement transposable. La solution doit être abstraite pour mieux être concrétisée, un peu comme une classe abstraite permet de créer par héritages des classes concrètes.
- Les conséquences qu'entraîné l'application de la solution. En matière de développement, cela peut concerner la mémoire utilisée, le temps de calcul, les contraintes particulières que le langage choisi, pour mettre en œuvre la solution, doit pouvoir respecter.

Les deux points centraux (description du problème traité et solution apportée) justifient en général l'utilisation de UML pour illustrer le propos et le rendre plus universel.

Les diagrammes les plus souvent utilisés en la matière sont les cas d'utilisation, les diagrammes de classes, les diagrammes de séquences et de collaboration ainsi que les diagrammes d'états de transitions.

Les *design patterns* sont l'antithèse de la réinvention de la roue. Et comme finalement le monde est partout le même, et que les problèmes proviennent de souvent de causes similaires, de situations « classiques », il est raisonnable de vouloir réutiliser les solutions qui ont fait leurs preuves face à des problèmes finalement identiques ou proches dans leur nature.

C'est ce que permettent de faire les *design patterns.* Lorsqu'elles sont appliquées au développement, elles permettent à la fois de gagner du temps et de s'assurer qu'on utilise une « bonne » solution qui ne causera pas ultérieurement plus de problèmes qu'elle n'en réglait.

La POO a introduit la notion de réutilisation du code en créant des acteurs autonomes que sont les objets, les design patterns inventent *la réutilisation des idées.* Après avoir résolu un problème original on peut généraliser cette solution en créant un design pattern.

### *3.11.2 Exemple de design Patterns*

#### *3.11.2.1 Le design pattern « singleton »*

Le « singleton» [Gamma 95] est l'une des techniques les plus utilisées en conception orientée objet. Il permet de référencer l'instance d'une classe devant être unique par construction. Certains objets techniques prennent en effet une responsabilité particulière dans la gestion logique d'une application. C'est par exemple le cas d'objets comme le « contrôleur des objets chargés en mémoire » ou le « superviseur des vues », qui sont les seuls et uniques représentants de leur classe. Ces objets sont le plus souvent publiquement accessibles. De tels cas de figure sont fréquents en conception, le singleton est requis pour les concevoir.

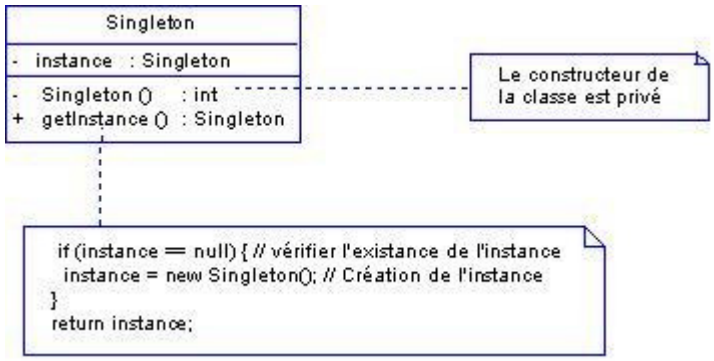
> Le singleton permet de s'assurer qu'une seule instance d'une classe sera instanciée pendant toute la durée d'exécution d'une application.

Implémentation

Pour s'assurer de l'unicité de l'instance d'une classe, il faut mettre les constructeurs de la classe en accessibilité limitée (privée). Cette mesure va garantir qu'aucun objet externe à la classe en question peut appeler directement le constructeur.

Du moment que les constructeurs sont inaccessibles depuis l'extérieur, il faut fournir un point d'appel accessible par tous les objets, une méthode *static* getInstance() qui va vérifier si une instance existe déjà avant de la retourné à l'appelant. (figure 3.11)

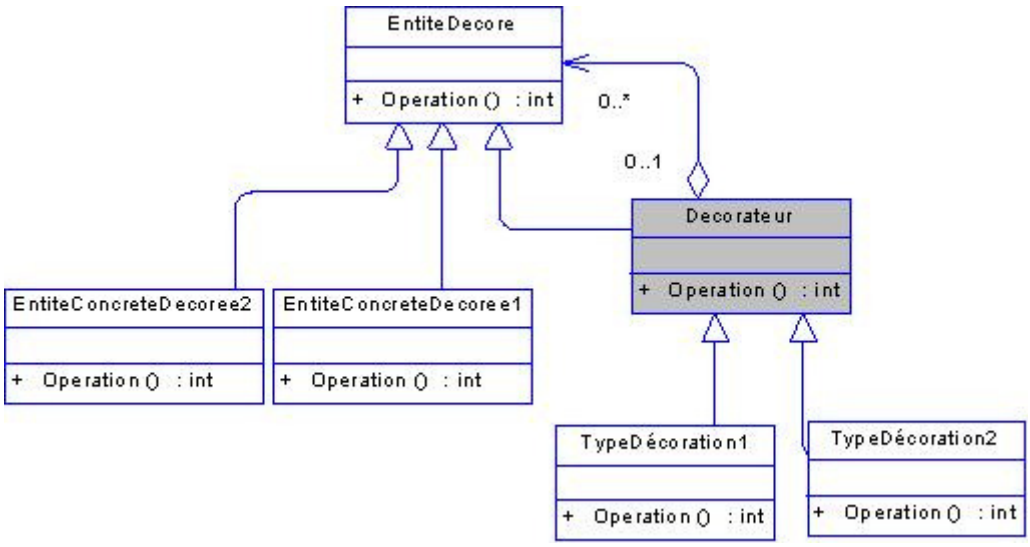
**Figure 3.11**
*Le design pattern Singleton*




### *3.11.2.2 Le design pattern « Decorator »*

Ce pattern permet d'attacher des responsabilités supplémentaires à un objet de façon dynamique. Avec ce pattern on dispose d'un moyen efficace pour accrocher de nouvelles fonctions à un objet dynamiquement lors de l'exécution. Cela ressemble à la fois un peu aux interfaces et à l'héritage mais c'est une troisième voie offrant des avantages différents. Les nouvelles fonctionnalités sont contenues dans des objets, l'objet parent étant dynamiquement mis en relation avec ces derniers pour en utiliser les capacités.

**Figure 3.12**
*Le design pattern Decorator*

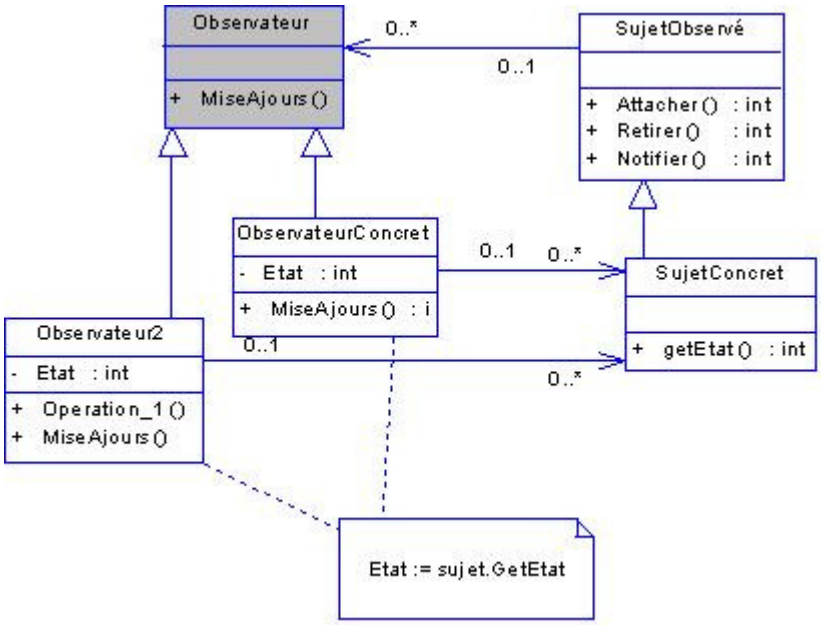



### *3.11.2.3 Le design pattern « Observer »*

La design pattern observer (observateur) définit une relation un-à-plusieurs entre des objets, de telle sorte que lorsque l'objet observé est modifié, tous les observateurs de ce dernier en soit notifiés

De nombreux exemples illustrent ce besoin, l'un des plus classiques est celui d'une information qui se trouve visualisée sous diverses formes. Supposons une gestion de stock offrant diverses vues sur chaque article : des graphiques, des fiches textuelles... Si l'utilisateur affiche plusieurs de ces fenêtres, ainsi que celle qui permet de modifier la fiche article, nous pouvons désirer que dès qu'une saisie modifie la fiche article, cela soit reflété immédiatement par toutes les fenêtres ouvertes affichant des informations relatives à l'article en question

**Figure 3.13**
*Le design pattern Observer*




#### *3.11.2.4 Autres exemples*

Le tableau suivant illustre une liste des design patterns les plus connus classés par catégorie, notez que cette liste n'est pas exhaustive, en effet il existe beaucoup d'autre, d'ailleurs vous pouvez vous-même en inventer des nouveaux.

**Tableau 3.3**
*Liste de Design Patterns*

| Catégorie | Design patterns |
|---|---|
| création | Abstract Factory, Builder, Factory Method, Prototype, Singleton |
| Structure | Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy |
| Comportement | Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor |

Comme le précise la définition d'un design pattern, l'application d'un design pattern dépend fortement de la nature du problème. Avec l'expérience, les développeurs cumulent une sorte de bibliothèque mentale, qui les aides à sélectionner rapidement le design pattern adéquat à leur problématique. Il est très utile d'avoir un aperçu sur un grand nombre de design pattern et leur utilisation pour s'en souvenir le moment venu.

# 4

# Approche des délégués des packages

## 4.1 Le Modèle MVC

### *4.1.1 Introduction*

L'objectif de la majorité des systèmes informatiques est de récupérer les données depuis un entrepôt de données et de les présenter à l'utilisateur. Une fois que l'utilisateur a modifié les données, le système stocke les mises à jour dans l'entrepôt. Comme le principal flux d'informations s'effectue entre le l'entrepôt de données et l'interface utilisateur, vous pourriez être tenté de lier ces deux éléments afin de réduire le volume de code et améliorer les performances de l'application. Cette approche qui peut vous paraître naturelle n'est pas sans présenter certains problèmes non négligeables. L'un d'eux est que l'interface utilisateur a tendance à changer plus fréquemment que le système de stockage des données. Autre problème du couplage des données avec les éléments d'interface utilisateur, les applications métier tendent à intégrer une logique métier qui va bien au-delà de la transmission de données.

L'architecture MVC (M*odèle Vue Contrôleur)* est un design pattern (motif de conception) pour le développement d'applications logicielles qui sépare le modèle de données, l'interface utilisateur et la logique de contrôle. Ce motif a été mis au point en 1979 par Trygve Reenskaug [Reenskaug], qui travaillait alors sur Smalltalk dans les laboratoires de recherche Xerox PARC. [wikipedia]

### *4.1.2 Problématique*

Comment normaliser l'interface utilisateur d'une application Web de sorte que chaque partie puisse être aisément modifiée individuellement ?

### *4.1.3 Facteurs à prendre en compte*

Les facteurs suivants agissent sur le système dans le contexte présent et doivent être pris en compte dans la recherche de la solution au problème :

La logique d'interface utilisateur est modifiée plus fréquemment que la logique métier, particulièrement dans les applications Web. Il est possible, par exemple, d'ajouter de nouvelles pages d'interface ou de réorganiser les pages existantes. Après tout, l'un des avantages d'une application cliente légère Web réside bien dans le fait que vous pouvez modifier l'interface utilisateur à tout moment sans avoir à redistribuer l'application. Si le code de présentation et la logique métier sont associés dans un seul et même objet, vous devrez modifier l'objet contenant la logique métier chaque fois que vous souhaiterez changer l'interface utilisateur. Cette démarche risque d'introduire des erreurs et nécessite à chaque modification de l'interface, même minime, de tester de nouveau l'ensemble de la logique métier.

Dans certains cas, l'application affiche les mêmes données sous plusieurs formes. Par exemple, un analyste préférera visualiser les données dans une feuille de calcul alors que ses managers choisiront les graphiques à secteur pour afficher les mêmes données. Certaines interfaces utilisateur clientes riches présentent simultanément plusieurs vues des même données. Si l'utilisateur modifie les données dans une vue, le système doit mettre à jour automatiquement toutes les autres vues des données.

La conception de pages HTML visuellement efficaces et attirantes nécessite généralement des compétences autres que celles requises pour le développement d'une logique métier complexe. Et il est bien rare qu'une même personne présente cette double compétence. Il est donc préférable de séparer l'effort de développement de ces deux parties.

L'activité de l'interface utilisateur se compose généralement de deux éléments : la présentation et la mise à jour. La partie présentation récupère les données depuis un entrepôt de données et les met en forme avant de les afficher. Lorsque l'utilisateur effectue une action sur les données, la partie mise à jour passe la main à la logique métier pour mettre à jour les données.

Dans les applications Web, une simple requête de page combine le traitement de l'action associée au lien que l'utilisateur a sélectionné et le rendu de la page cible. Il n'est pas rare que la page cible ne soit pas directement liée à l'action. Imaginez, par exemple, une simple application Web qui présente une liste d'articles. L'utilisateur revient à la page de liste principale après avoir ajouté un article à la liste ou supprimé un article de la liste. L'application doit donc rendre la même page (la liste) après exécution de deux commandes plutôt différentes (ajout ou suppression) — le tout dans le cadre de la même requête HTTP.

Le code de l'interface utilisateur dépend généralement plus de la machine que la logique métier. Si vous souhaitez faire migrer l'application d'une application sur navigateur à une application sur assistant personnel ou sur téléphone mobile compatible Web, vous devez réécrire une bonne partie du code de l'interface utilisateur alors que la logique métier pourra rester inchangée. Une séparation

claire de ces deux éléments permet d'accélérer la migration et de réduire le risque d'introduction d'erreurs dans la logique métier.

La création de tests automatisés pour les interfaces utilisateur est généralement plus longue et difficile que les mêmes tests pour la logique métier. On comprend donc que réduire le volume de code directement lié à l'interface utilisateur permet d'améliorer les capacités d'évaluation de l'application.

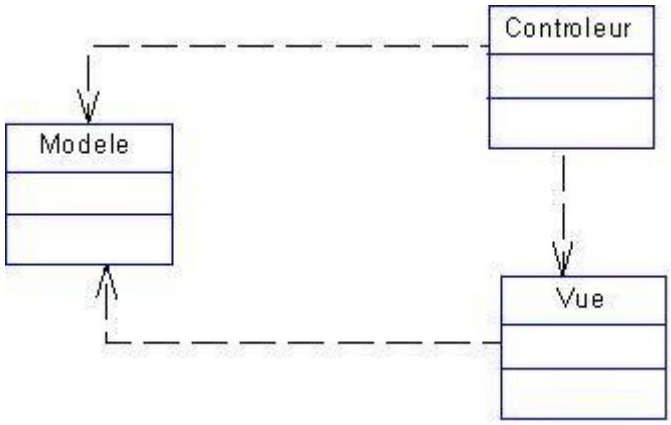
### *4.1.4.Solution*

Le modèle *Model-View-Controller (MVC)* sépare la modélisation du domaine, la présentation et les actions reposant sur l'entrée utilisateur en trois classes distinctes [Burbeck92] :

- **Modèle**. Le modèle gère le comportement et les données du domaine d'application, répond aux demandes d'informations sur son état (souvent issues de la vue) ainsi qu'aux instructions de changement d'état (souvent issues du contrôleur).
- **Vue**. La vue gère l'affichage des informations.
- **Contrôleur**. Le contrôleur interprète les entrées clavier et souris de l'utilisateur et informe le modèle et/ou la vue des changements nécessaires.

La figure 4.1 présente les relations structurelles entre les trois objets.

**Figure 4.1**
*Structure de classe MVC*




Il est important de noter que la vue et le contrôleur dépendent tous deux du modèle. En revanche, le modèle ne dépend ni de la vue ni du contrôleur. C'est l'un des principaux avantages de la séparation. Cette séparation des tâches permet de créer et de tester le modèle indépendamment de la représentation visuelle. La séparation entre la vue et le contrôleur est secondaire dans la plupart des applications clientes riches et, en effet, de nombreuses infrastructures d'interface utilisateur implémentent les rôles comme un seul objet. Dans les applications Web, par contre, la séparation entre la vue (le navigateur) et le contrôleur (les composants côté serveur gérant les requêtes HTTP) est clairement définie.

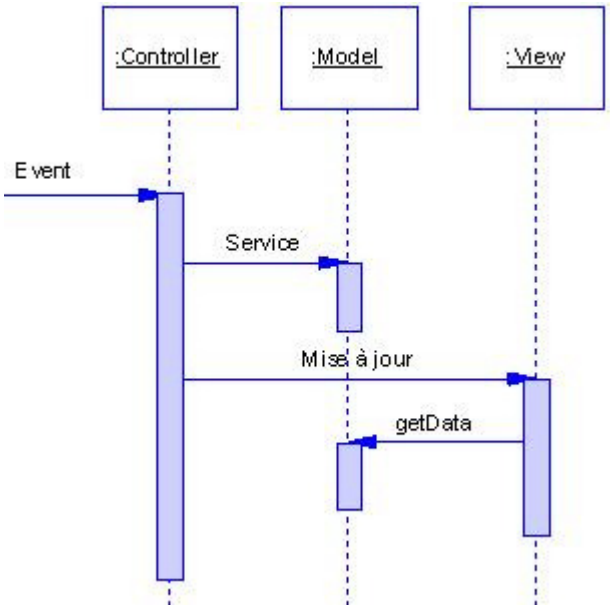
Le modèle *Model-View-Controller* est un modèle de conception fondamental en matière de séparation de la logique de l'interface utilisateur et de la logique métier. Malheureusement, la popularité de ce modèle a eu pour conséquence un certain nombre de descriptions erronées. En particulier, le terme « contrôleur » a été utilisé pour signifier différentes choses dans différents contextes. Fort heureusement, l'avènement des applications Web a permis de résoudre certaines ambiguïtés, tant la séparation entre la vue et le contrôleur est évidente.

### *4.1.5 Variations*

Dans son livre, *Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)* [Burbeck92] (Programmation d'applications Smalltalk80: utilisation du modèle Model-View-Controller (MVC)), Steve Burbeck décrit deux variantes du MVC : un modèle passif et un modèle actif.

Le modèle passif s'utilise lorsqu'un contrôleur manipule le modèle exclusivement. Le contrôleur modifie le modèle, puis informe la vue que le modèle a changé et doit être actualisé (voir figure 4.2). Dans ce scénario, le modèle est entièrement indépendant de la vue et du contrôleur, ce qui signifie que le modèle n'a aucun moyen de rapporter ses changements d'état. Le protocole HTTP constitue un exemple de ce mode de fonctionnement. Le navigateur ne dispose pas de moyen simple pour obtenir des mises à jour asynchrones du serveur. Il affiche la vue et répond à l'entrée utilisateur mais il ne détecte pas les modifications apportées aux données sur le serveur. Le navigateur interroge le serveur sur les modifications uniquement lorsque l'utilisateur demande explicitement une actualisation.

**Figure 4.2**
*Comportement du modèle passif*




### *4.1.6 Comportement du modèle actif*

Le modèle actif est utilisé lorsque le modèle change d'état sans impliquer le contrôleur. Ceci se produit lorsque d'autres sources modifient les données et que les modifications doivent être prises en compte dans les affichages. Considérons l'affichage d'un marché boursier. Vous recevez des informations sur les actions d'une source externe et vous souhaitez mettre à jour les vues (par exemple un téléscripteur et une fenêtre d'alerte) lorsque les données sur les actions changent.

Comme le modèle est le seul à détecter les modifications de son état interne lorsqu'elles se produisent, il doit indiquer aux vues d'actualiser l'affichage.

Toutefois, l'une des raisons qui motivent l'utilisation du modèle MVC est de conserver un modèle indépendant des vues. Si le modèle devait notifier les changements aux vues, vous réintroduiriez la dépendance que vous cherchiez à éviter. Heureusement, le modèle ***Observer*** [Gamma95] fournit un mécanisme pour alerter les autres objets des changements d'état sans pour autant introduire de dépendances entre eux. Chaque vue implémente l'interface *Observer* et s'enregistre auprès du modèle. Le modèle tient à jour la liste de tous les observateurs qui s'abonnent aux changements. Lorsqu'un modèle change, il effectue une itération sur tous les observateurs enregistrés et les notifie du changement. Cette approche est généralement appelée « éditer-s'abonner ». Le modèle ne nécessite jamais d'informations spécifiques sur les vues. En fait, dans les scénarios où le contrôleur doit être informé des changements de modèle (pour activer ou désactiver des options de menu, par exemple), les seules tâches du contrôleur consistent à implémenter l'interface *Observer* et à s'abonner aux changements du modèle. Lorsqu'il y a plusieurs vues, il peut s'avérer utile de définir plusieurs sujets, chacun d'entre eux décrivant un type spécifique de changement de modèle. Chaque vue peut ensuite s'abonner uniquement aux types de modifications pertinents pour la vue.

La figure 4.3 illustre la structure du MVC actif utilisant l'*observateur*. Elle présente en outre la façon dont l'observateur isole le modèle afin d'éviter qu'il ne fasse directement référence aux vues.

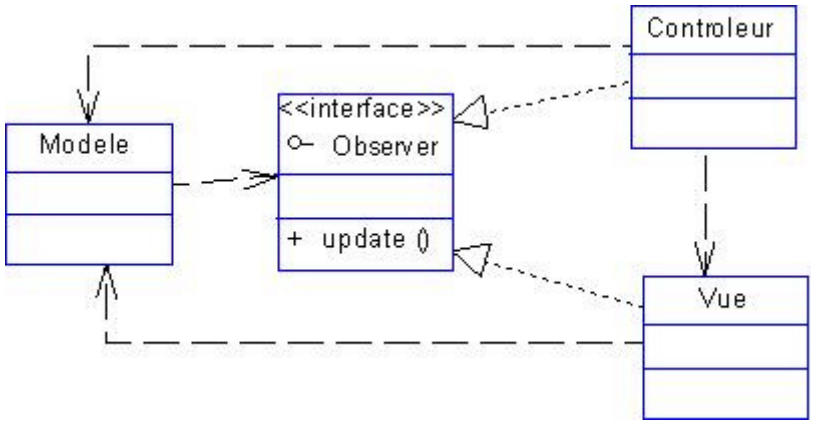



**Figure 4.3**
*Utilisation de l'observateur pour dissocier le modèle de la vue dans le modèle actif*

La figure 4.4 illustre la façon dont l'*observateur* signale aux vues les modifications du modèle. Malheureusement, il n'est pas évident de démontrer la séparation du modèle et de la vue dans un diagramme de séquences UML car le diagramme représente des instances d'objets au lieu de classes et d'interfaces.

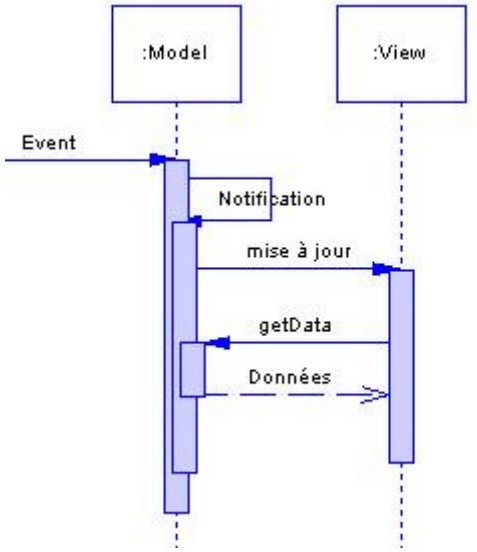



**Figure 4.4**
*Comportement du modèle actif*

### *4.1.7 Considérations relatives au test*

La capacité de test est fortement améliorée lorsqu'on utilise un modèle *Model-View-Controller*. Le test des composants se complique lorsque ceux-ci sont interdépendants et particulièrement lorsqu'il s'agit de composants d'interface. En effet, ce type de composant nécessite souvent une configuration complexe pour tester une simple fonction. Pire, lorsqu'une erreur se produit, il est difficile d'isoler le problème sur un composant spécifique. C'est la raison pour laquelle la séparation des problèmes revêt une telle importance du point de vue architectural. *MVC* sépare le problème de stockage, d'affichage et de mise à jour des données entre trois composants pouvant êtres testés individuellement.

Outre les problèmes posés par les interdépendances, les infrastructures d'interface utilisateur s'avèrent intrinsèquement difficiles à tester. Le test des interfaces utilisateur nécessite soit un test manuel, fastidieux (et sujet aux erreurs), soit des scripts qui simulent les actions de l'utilisateur. Le développement de ces scripts est généralement long et délicat. Le modèle *MVC* n'évite pas d'avoir à tester l'interface utilisateur mais, du fait que la logique de présentation est séparée du modèle, ce dernier peut être testé en toute indépendance par rapport à la présentation, ce qui réduit le nombre des combinaisons de test de l'interface utilisateur.

**Contexte final**

L'organisation architecturale de la couche présentation autour du modèle MVC fait apparaître les avantages et les inconvénients suivants :

**Avantages**

**Prise en charge de plusieurs vues.** Comme la vue est séparée du modèle et qu'il n'y a pas de dépendance directe entre eux, l'interface utilisateur peut afficher simultanément plusieurs vues des même données. Ainsi, plusieurs pages d'une application Web peuvent utiliser les mêmes objets de modèle. L'application Web permettant à l'utilisateur de modifier l'apparence des pages constitue un autre exemple des avantages de cette séparation. Ces pages affichent les mêmes

données à partir du modèle partagé mais ces données s'affichent de différentes manières.

**Adaptation aux changements.** Les exigences en matière d'interface utilisateur ont tendance à changer beaucoup plus rapidement que les règles métiers. Les utilisateurs peuvent préférer utiliser différentes couleurs, polices, disposition à l'écran et plusieurs niveaux de prise en charge des nouveaux périphériques tels que les téléphones mobiles ou les assistants personnels. Comme le modèle ne dépend pas des vues, l'ajout de nouveaux types de vues au système ne l'affecte généralement pas. En conséquence, le changement porte uniquement sur la vue.

**Inconvénients**

**Complexité.** Le modèle *MVC* introduit de nouveaux stades d'indirection et augmente de ce fait légèrement la complexité de la solution. Il accroît également la nature événementielle du code de l'interface utilisateur qui peut alors être plus difficile à déboguer.

**Coût des mises à jour fréquentes.** Dissocier le modèle de la vue ne signifie pas que les développeurs du modèle peuvent ignorer la nature des vues. Par exemple, si le modèle subit des modifications fréquentes, il peut inonder les vues de demandes de mise à jour. Certaines vues, telles que les affichages graphiques, peuvent prendre du temps à s'afficher et la vue peut alors être décalée par rapport aux demandes de mise à jour. Il est donc important de penser à la vue lors du codage du modèle. Le modèle pourra, par exemple, regrouper plusieurs requêtes en une seule notification à la vue.

## 4.2. Approche des délégués des packages

### *4.2.1 Introduction :*

Nous avons vu dans la première partie de ce chapitre, l'utilité de ce modèle et ce qu'il apporte comme avantages tout au long du processus du développement logiciel. Cependant il reste un grand travail à la charge des concepteurs, pour implémenter réellement cette architecture. L'étude faite dans cette partie vise à réduire cette charge et à proposer une architecture fidèle au MVC ainsi facile à implémenter pour la plupart des applications WEB.

Cette approche se base sur le principe des *délégués des packages* et la programmation orientée objet, qui consiste à centraliser les connaissances d'un package MVC dans une seule classe constituant l'interface du package et le moyen unique pour le pénétrer.

### *4.2.2 Approche des délégués des packages :*

Supposons que vous êtes chargés d'entreprendre une négociation avec un prestataire de service. A priori vous savez ce que cette entreprise peut vous fournir comme produit, mais vous ne savez pas qui en particulier se chargera réellement de la réalisation du travail demandé. La question est : à qui devez vous vous adresser ?

Dans ce genre de situation, les responsables de l'entreprise sont les plus qualifiés pour vous répondre. En effet, car ils ont la connaissance sur l'organisation de l'entreprise et les fonctions du personnel. Vous allez, naturellement entamer les négociations avec eux et ils se chargeront ensuite de demander aux personnes concernées de réaliser le travail demandé.

L'approche **des délégués des packages** s'inspire fortement de cet exemple. Le terme package fait référence aux trois unités qui constituent le modèle MVC [Modèle –Vus –Contrôle]. J'ai choisi le mot « *délégué* » pour cette approche car toutes les connaissances d'un package sont centralisées dans une seule classe de délégation. Cette classe s'occupe du dispatching des évènements en provenance de différentes sources vers les classes internes du package.

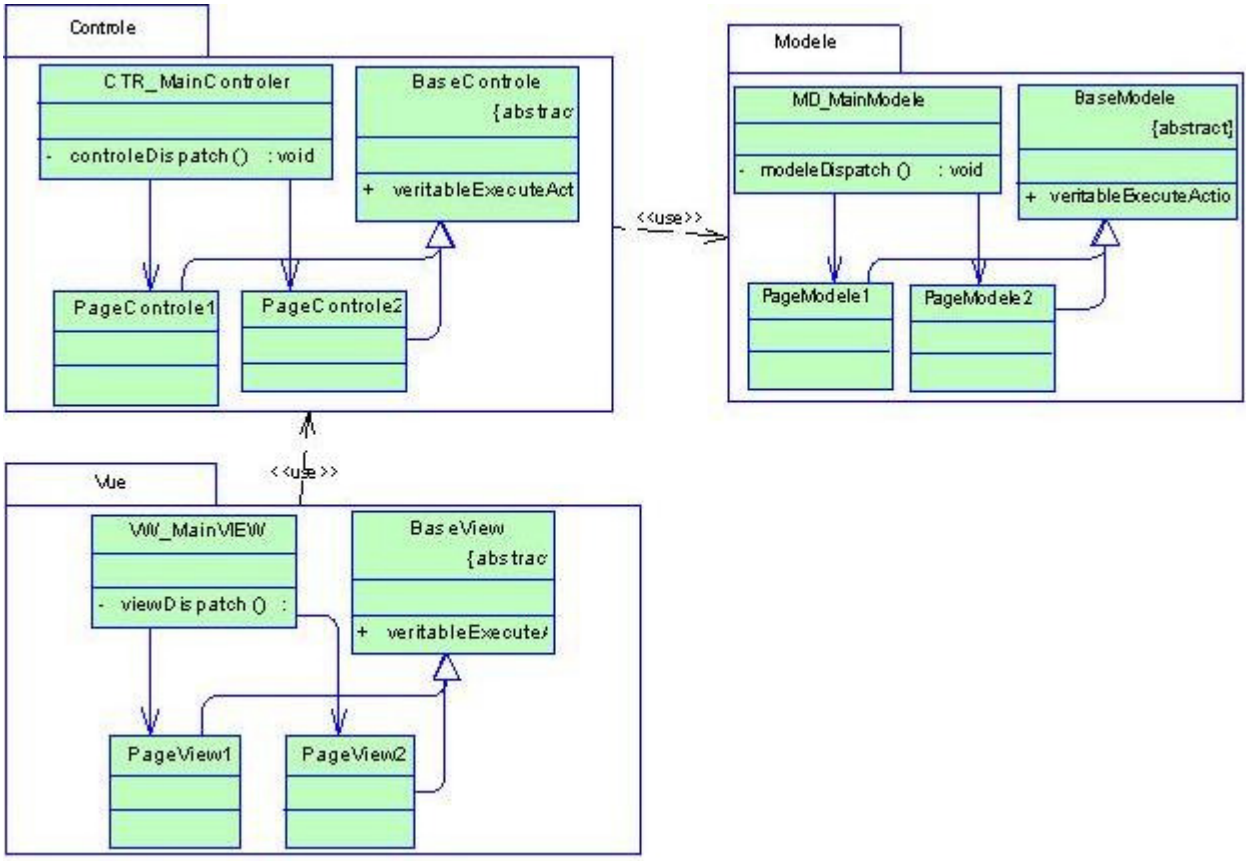
On peut résumer le principe de cette approche dans les points suivant :

1. Pour chaque package on crée une classe de délégation.
2. La communication avec les autres classes du package passe exclusivement par l'intermédiaire de cette classe de délégation.
3. Les classes de délégation disposent d'une connaissance exacte sur les autres classes du même package.
4. A la réception d'un événement (demande d'exécution d'une méthode) par une classe de délégation. Cette dernière peut déterminer la cible de l'évènement au sein de son package puis lui rediriger automatiquement cette demande d'exécution.

### *4.2.3. Amélioration du faible couplage M-V-C*

La figure 4.5 ci-dessous montre l'architecture globale du modèle MVC amélioré suivant l'approche des délégués des packages.

**Figure 4.5**
*Modèle MVC avec les délégués des Packages*




Les classes commençant avec le préfixe `Main` à savoir `MainControle`, `MainView` et `MainModele` correspondent respectivement aux délégués des packages `Modele`, `Vue`, `Contrôle`.

Les classes commençant par le préfixe `Base`, représentent des classes abstraites additionnelles. Elles regroupent toutes les fonctionnalités communes aux classes de chaque package, on peut imaginer une architecture plus complexe avec une hiérarchie à plusieurs niveaux, suivant les besoins de chaque projet et application WEB.

Les classes commençant par le préfixe `page` donnent un exemple sur ce que peut contenir effectivement l'application WEB. Je préconise avec cette approche, de créer une classe de contrôle et une classe de vue ainsi q'une classe de modèle pour chaque page de l'application, susceptible de contenir des données dynamiques.

Il reste à la charge de chaque développeur de définir le contenu fonctionnel (propriétés, méthodes ...) des classes suivant ses besoins. Néanmoins, il est important de noter que les classes du modèle représentent l'interface de l'application coté base de données. Il est donc inutile de créer et d'associer de telles classes à des pages qui ne demandent pas des données de la base de données. L'exemple caractéristique, est celui des pages à contenu *dérivé* calculé avec des scripts chargés sur le poste du client.

### *4.2.4 Avantages de Approche des délégués des packages*

1. Modularité : cette approche se base sur le principe d'éclatement des applications WEB en composants autonomes. Le modèle MVC propose déjà dans sa version standard trois modules (modèle, vue, contrôle). Cette approche améliore cette version en décrivant une meilleure manière de procéder pour arriver à une implémentation simple et claire.

2. Faible couplage entre modules: nous avons vu que les trois packages disposent d'une seule interface de communication avec l'extérieur, ce qui permet d'avoir une dépendance menée au minimum entre eux.

3. Réduire la charge des concepteurs : souvent la conception des applications (web et classiques), demande un effort considérable et des compétences en terme d'analyse et de méta modélisation. Si vous avez participé à la réalisation d'un projet informatique (sûrement), vous avez pu constater qu'une erreur de conception a souvent des implications, plus ou moins, importantes au moment du passage au codage. En effet, une fois impliqué dans le processus de codage, vous risquez de faire une erreur si vous essayer de modifier une décision déjà prise dans la phase de conception de l'application.

   Pourquoi ? Parce que le codage consiste à traduire ce qui a été conçu en un programme qui tourne sur un support informatique. Or la conception possède l'avantage de visualiser la solution depuis un niveau d'abstraction plus élevé.
   L'intérêt de l'approche des délégués des packages est de permettre de faire ce retour en arrière, car il y a peu de chose à modifier dans les concepts, mis à part l'attribution des rôles à des classes spécifiques, la création de nouvelles classes dans les différents packages et d'en informer les délégués.

4. Evolution rapide : supposez que vous travaillez sur une application WEB conçue suivant cette approche et que l'on vous demande d'ajouter d'autres modules à l'application (ex. : ajouter de nouvelles rubriques ou de nouvelles pages) La démarche est simple, après avoir défini l'information à afficher et la source de cette information, l'implémentation devient automatique en respectant les règles définies.

5. Sécurité : rediriger les évènements vers une seule cible permet une meilleure sécurité, ainsi le contrôle d'identification et des droits d'accès est centralisé, ce qui éloigne le risque d'oubli (ex. : oublier de sécuriser le point d'entrer d'un nouveau module).

6. Maintenance aisée : travailler sur des modules fonctionnellement séparés permet de réaliser des tests précis et facilite la détection des sources d'erreurs. S'il s'agit d'un problème d'affichage vous savez pertinemment que c'est dans la vue qu'il faut chercher.

7. Génération automatique du code natif : la génération automatique du code natif depuis des schémas de modélisations UML est devenue désormais une pratique courante dans la plupart des outils de modélisation. Cependant, il n'est pas courant de trouver de tels outils pour les langages de développement WEB (PHP, ASP, …) mais on s'y dirige petit à petit car il y a une forte tendance vers POO. Le langage PHP implémente dans la version 5 les concepts de la programmation orientée objet. D'autres outils commencent à intégrer ces techniques, tel que CodeCharge3, Dreamweaver.

8. Distribuer les responsabilités suivant les compétences : cet avantage hérité du modèle MVC, comme il s'agit de travailler sur 3 aspects fondamentalement différents (Modèle, Vue, Contrôle) il permet de diviser l'équipe de travaille en trois grands groupes ; chacun se voyant affecter la charge d'un module. En outre, il est rare de trouver des compétences du codage et d'interface réunies chez une même personne.

### *4.2.5 Inconvénients Approche des délégués des packages*

1. Demande des compétences de POO : j'ai évoqué ce point en tant qu'inconvénient, bien que l'on puisse le considérer comme un avantage. En effet, le développement WEB est ouvert le plus souvent, à des concepteurs non confirmés. Le développement objet reste à présent réservé aux professionnels du métier.

2. Rigoureuse au moment de la conception : le modèle MVC est un moyen théorique pour mieux organiser l'architecture des applications afin d'améliorer la productivité. Cette approche perfectionne ce modèle et ajoute de la clarté à son implémentation technique. Mais elle exige une bonne conception c'est à dire son adaptation aux besoins de chaque application WEB.

**Notes :**

Cette approche est valable pour la plupart des applications WEB. Par contre il est inutile d'y faire recours dans deux cas :

1. Quand il s'agit de site web à contenu statique.

2. Applications dynamiques de petites taille et non évolutives : d'ailleurs c'est le cas du MVC. L'objectif est de réduire la charge du concepteur et non pas de compliquer sa tâche. Mélanger tous les aspects (logique métier et application) révèle un intérêt crucial quand le contenu de l'application est restreints.

### *4.2.6 Exemple d'implémentation de l'approche des délégués des packages*

Ce chapitre était l'occasion pour présenter notre approche des délégués des packages, qui vise à optimiser et à simplifier l'architecture conceptuelle des applications orientées objet en général et les en particulier les application web.

Nous avons pu réaliser tout un projet suivant cette approche. C'est le projet Merlin décrit dans le chapitre 5 suivant, où l'architecture de l'application a été soigneusement réalisée pour respecter les concepts définis dans ce chapitre.

## Conclusion

Le modèle MVC a fait preuve d'efficacité il y a bien longtemps. L'approche des délégués des packages vise à renfoncer la segmentation de différentes composantes d'un système informatique, et de fournir une méthode d'implémentation pratique.

L'avantage de faire recours à cette approche, est de pouvoir suivre l'aspect dynamique d'une application. En effet, déléguer la communication d'une composante architecturale à une seule classe, garantie la centralisation des messages entrants/sortants et offre ainsi une meilleurs traçabilité des trames échangées.

Le chapitre suivant est consacré à l'implémentation de cette approche dans un projet d'informatique décisionnelle.

# 5

# Application & Implémentation

## 5.1 Introduction

Nous avons pu concrétiser les concepts étudiés dans ce mémoire dans plusieurs projets informatiques de grandes ampleurs, il s'agit précisément d'un site web et d'une série d'applications Intranet implémentées dans un environnement industriel pour gérer le système d'information d'une entreprise.

Nous allons nous contenter d'exposer deux projets qui ont étés développés respectivement pour le compte des sociétés ECS-Schindler[1] et AFP[2]. Le premier projet, est une implémentation effective de l'approche des délégués des packages pour une solution de l'informatique décisionnelle avec un Data Warehouse. Le deuxième est un site web (www.afp-lemans.fr) intégrant un éditeur des contenus destiné à améliorer la création et l'édition des sites web.

## 5.2 Le projet MERLIN

### *5.2.1 Contexte*

Ce projet s'inscrit dans la volonté de l'entreprise ECS-Schindler[1] à renforcer son système décisionnel, afin de permettre aux décideurs de disposer d'informations et de réponses à des questions cruciales, pour la prise de décision et la planification d'une stratégie marketing ciblée.

1 : ECS-Schindler (Elevator Car System) : est une unité de production rattachée au groupe multinational Suisse Schindler spécialisé dans la fabrication et la commercialisation d'ascenseurs et d'escaliers mécaniques. Leader mondial des escaliers mécaniques et leader européen des ascenseurs, le groupe Schindler est implanté dans plus de 140 pays.
ECS est une usine de production de cabine d'ascenseur Schindler située à la ville de Mulhouse en France. Elle reçoit ses commandes à travers un système de gestion intégré ERP/SAP en provenance des autres entités Schindler à l'échelle mondiale. Dès réception un système d'import automatique récupère le détail de ces commandes dans une base interne oracle pour réaliser des traitements spécifiques, Chaque commande suit un flux de production et en parallèle un circuit informatique de validation très strict avant d'être expédiée au client.

2 : AFP (Association pour la Formation du Personnel) est spécialisée dans l'audit conseil en ressources humaines et la formation du personnel : Management, Communication, Technique, Productique, Mécanique, Maintenance, Production. Elle prépare aussi les jeunes aux différents diplômes professionnels. Les demandeurs d'emploi aussi peuvent acquérir des compétences au sein de l'AFP.

### *5.2.2 Dénomination du projet*

Pour simplifier le référencement de ce projet, nous nommerons « MERLIN » l'ensemble des entités de celui-ci. [Ainsi ce mot désigne le projet en question. Ce choix a été pris seulement dans le seul but de simplification.]

### *5.2.3 Problématique*

L'entreprise d'ECS-Schindler dispose d'un nombre important de sources de données (ERP/SAP, Notes/GSM, Base oracle, Base Access, fichiers Excel). Avant la mise en place de ce projet il n'existait pas de moyens pour fusionner toutes ces sources de données afin d'en extraire de l'information.

Souvent la dimension d'une réponse fait appel à plusieurs sources de données, il n'est pas impossible de rechercher dans ces sources pour trouver la réponse mais il est rare de disposer du temps nécessaire pour le faire. Il faut pouvoir donner une réponse immédiate lors d'un appel téléphonique d'un client ou d'un responsable hiérarchique.

L'ambition de l'entreprise est d'arriver à éviter des erreurs de décision dues au manque d'informations aux moments opportuns.

### *5.2.4 Solution*

Avant d'arriver à élaborer une solution à une problématique il faut faire un effort considérable afin de comprendre le besoin et faire le point sur l'état de l'art des technologies et des solutions existantes.

Dans notre cas, il est question d'un traitement de grande masse de données en provenance de plusieurs sources de données non homogènes. Typiquement la solution à cette problématique est la mise en place d'un data warehouse (Annexe A) (entrepôt de données). Il s'agit d'une solution de l'informatique décisionnelle qui exploite les données des entreprises dans le but de faciliter la prise de décision (par les décideurs).

> A ce titre, le Data Warehouse doit être rapproché de tous les concepts visant à établir une synergie entre le système d'information et sa stratégie.

Objectifs de MERLIN :

14. Permettre une prise de décision rapide.
15. Tableaux de bord disponible de manière dynamique.
16. Accès direct par l'utilisateur final via un navigateur Web.
17. Un seul système interrogé assurant performance et cohérence.
18. Mettre en valeur les connaissances et le savoir-faire de l'entreprise.
19. Archivage et accessibilité des données.
20. Croiser les données des différentes sources pour en extraire de l'information.

Ce projet était l'occasion pour implémenter l'approche des délégués des packages exposée dans le chapitre 4. Nous avons respecté l'architecture de cette approche ; chaque page de notre système est représenté par les trois composantes MVC et

toutes les transactions des événements transitent via les délégués des packages pour s'aligner aux principes de l'approche.

### *5.2.5 Technologies employées*

Plusieurs technologies ont été utilisées :

- DELPHI pour l'importation des données et le paramétrage de ETL.
- Serveur d'application : PHP 4.3.10 pour la consultation web.
- Serveur http : Apache.
- C /Labwindow : pour l'application des bancs de test : l'utilisation de cet environnement était secondaire,
- PL/SQL : pour les déclencheurs et les procédures stockées.
- SGBD :
    - Oracle : comme source et cible de données.
    - Access : source de donnée
    - Notes : utilisation de SQLNotes avec ODBC
    - SAP : utilisation de l'ActiveX

Pour des contraintes de performances et budgétaires liées à la politique de la société, le choix d'Oracle apparaît le plus évidant, en effet, la masse d'informations traitées nécessite un SGBD client/Serveur robuste, en plus l'entreprise dispose d'une licence Oracle ce qui évite un investissement assez important. En plus oracle a l'avantage de faciliter l'implémentation de plusieurs profils de connexion pour assurer une sécurité renforcée (voir *type de requête* plus loin dans ce chapitre 5.2.12.3).

### *5.2.6 Notion d'indicateur*

« Un indicateur est un ensemble d'informations structurées qui permet à un décideur de prendre des décisions ou de prévoir des mesures à mettre en place pour assurer un niveau de qualité prévu dans la politique d'une organisation ».

La détermination des indicateurs pertinents est une étape clé dans la réussite de n'importe quel système décisionnel. Elle détermine son éventuelle fréquence d'utilisation ainsi sa valeur rajoutée au sein d'une organisation. Souvent cette étape s'élabore par plusieurs réunions de travail, avec les décideurs et les potentiels utilisateurs du futur système, pour accueillir un maximum d'informations sur le fonctionnement du processus décisionnel de l'organisation et cela tout au long du projet.

Pour l'utilisateur final le résultat doit être instantané et exploitable pour la prise de décision, ce qui implique une présentation intuitive et la possibilité de visualiser les résultats suivant plusieurs façons dont la plus utilisée est la notion de tableaux de bord.

Les objectifs de tous les systèmes décisionnels tournent autour du principe des indicateurs ; meilleurs sont les indicateurs implémentés, meilleurs sont les réponses fournies et en dépend la position stratégique de ces systèmes au sein de l'entreprise.

#### *5.2.6.1 Indicateurs ECS-Schindler*

Chaque entreprise dispose de ses propres indicateurs, il est vrai que certains d'entre eux peuvent être communs à plusieurs entreprises, mais souvent le piège invisible, c'est d'essayer de calquer ou de projeter les indicateurs d'une entreprise sur une autre sans aucune étude fonctionnelle auprès des utilisateurs concernés. Cette démarche engendre une fuite en masse de ces potentiels utilisateurs, et la solution demeure obsolète car elle ne répond plus à leurs attentes.

Après plusieurs réunions de travail avec les futurs utilisateur du système, nous avons pu définir un certain nombre d'indicateurs importants pour l'entreprise, en voici une liste non exhaustive :

- Suivi des commandes : à chaque fois savoir le nombre de commande suivant son statut :
  - Statut lancement : Entrée, Traitée, Bloquée
  - Statut production : En production, En quarantaine, Produite
  - Statut expédition : Expédiée, Non Expédiée.
- Suivi chiffre d'affaires : Permet de connaître le chiffre d'affaires réalisé avec un client (ou l'ensemble des clients) par produit dans une période donnée.
- Suivi des retards : Permet de connaître la liste des commandes en retard par type de produit.
- Paramètre commande : Permet de consulter les paramètres techniques d'une commande.

### *5.2.7 Alimentation du Data Warehouse*

Comme nous l'avons annoncé dans la problématique, les sources de données sont multiples. La présence des données nécessaires pour la prise d'une décision dans ces multiples sources en même temps implique des recherches fastidieuses et des temps de traitement très importants, ce qui complique la tâche des décideurs.

Pour prendre la décision d'augmenter le chiffre d'affaire avec un client il faut connaître :

- Type des commandes : base de production
- Ponctuation du payement des redevances : base comptabilité
- Solvabilité : donnée juridique
- ….

C'est là où interviens les ETL.

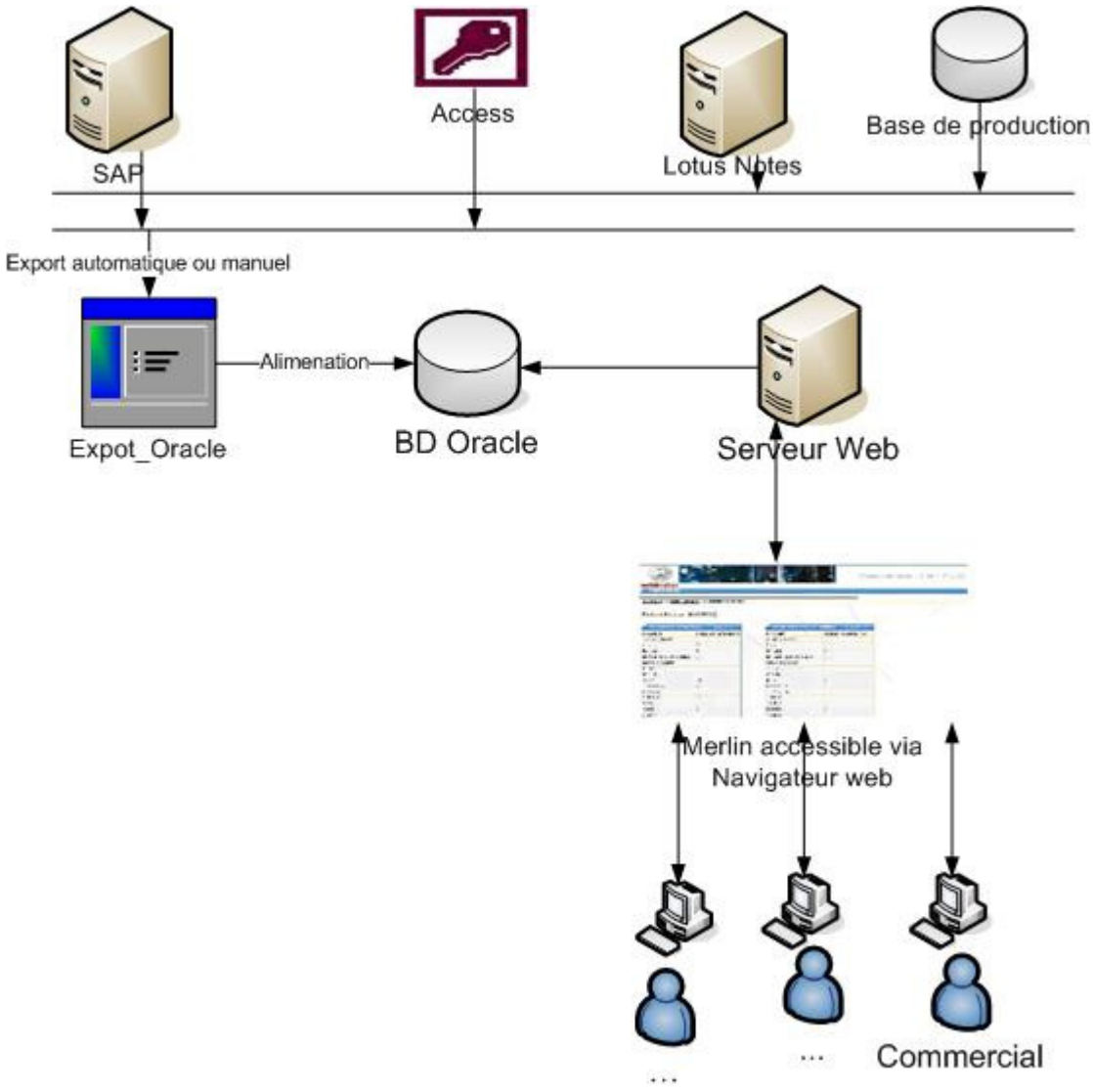
ETL (Extract-Transform-Load ou extraction-transformation-chargement) ?

A l'origine les données peuvent être de différentes sources de donnée de l'entreprise, à ce stade elles se présentent de manière brute. Afin qu'elles soient exploitables, une étape de filtrage et de prétraitement est nécessaire (rôle des ETL).

L'ETL est un logiciel programmé pour tourner périodiquement (ex. Une fois chaque nuit) en se connectant aux différentes sources, récupérer les données et effectuer les traitements nécessaires avant d'alimenter le Data Warehouse.

La figure 5.1 montre une vision globale de l'architecture du projet MERLIN.

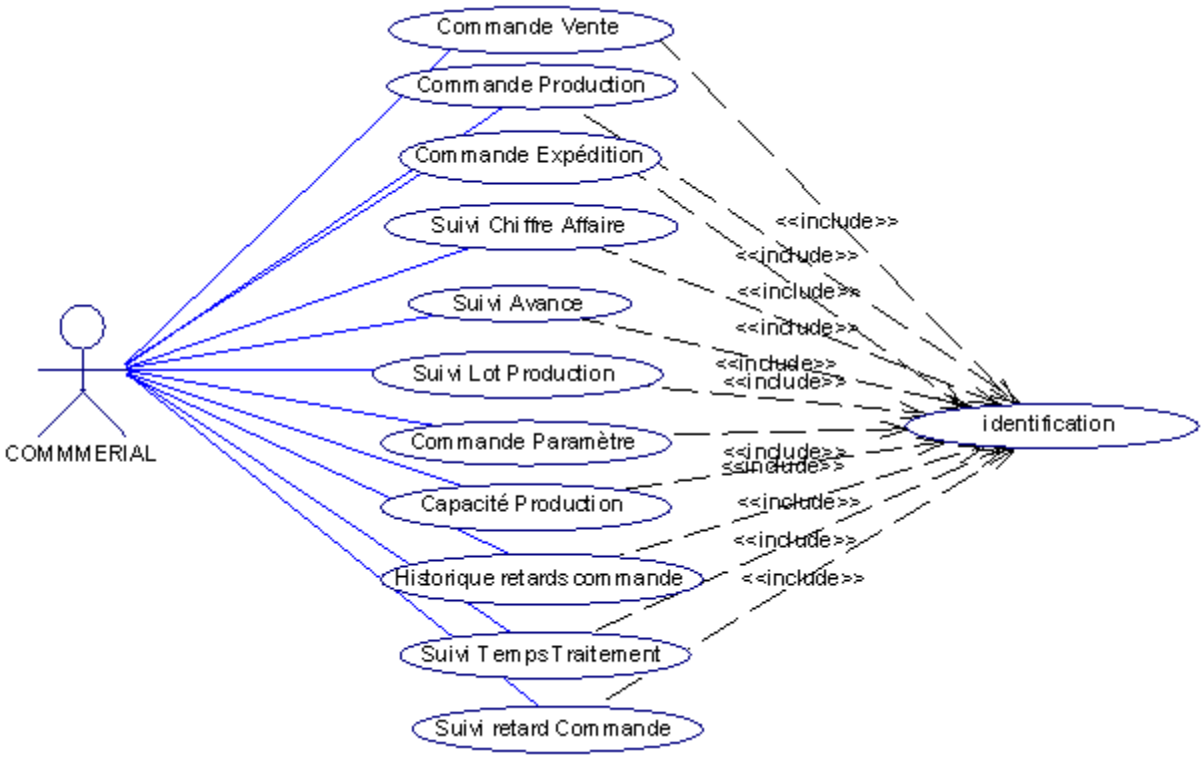
**Figure 5.1**
*Sources de données MERLIN*




### *5.2.8 Diagramme des cas d'utilisation*

Pour simplifier le diagramme des cas d'utilisation nous présenterons un seul niveau d'héritage. (figure 5.2)

**Figure 5.2**
*Diagramme des cas d'utilisation*

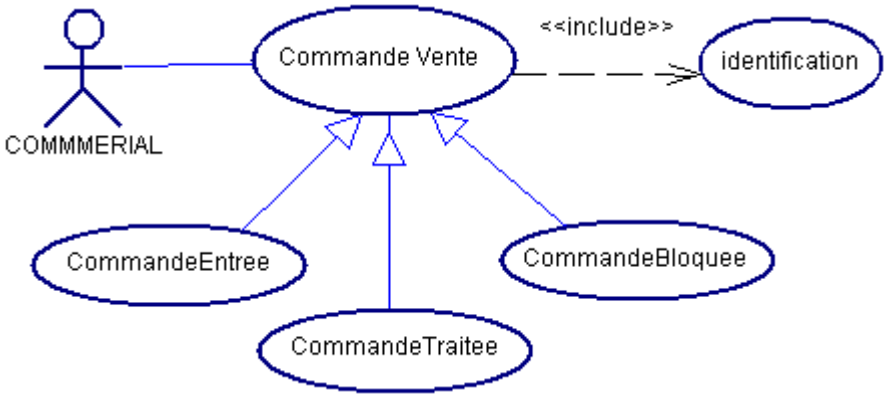



Il n'est pas lieu ici de détailler toute l'implémentation technique de notre système, nous nous contentons de présenter un seul cas d'utilisation parmi les plus importants et les plus complets à savoir `commandeVente`.

Pour décrire notre cas d'utilisation nous utilisons la description textuelle exposée dans le chapitre 3.

**Description textuelle du cas d'utilisation CommandeVente**

**Figure 5.3**
*Détail du cas d'utilisation CommandeVente*




| CU | Description |
|---|---|
| Commande vente | Permet à un commercial de connaître le nombre de commandes entrées, traitées ou bloquées par type de produit dans une période donnée. (figure 5.3) |

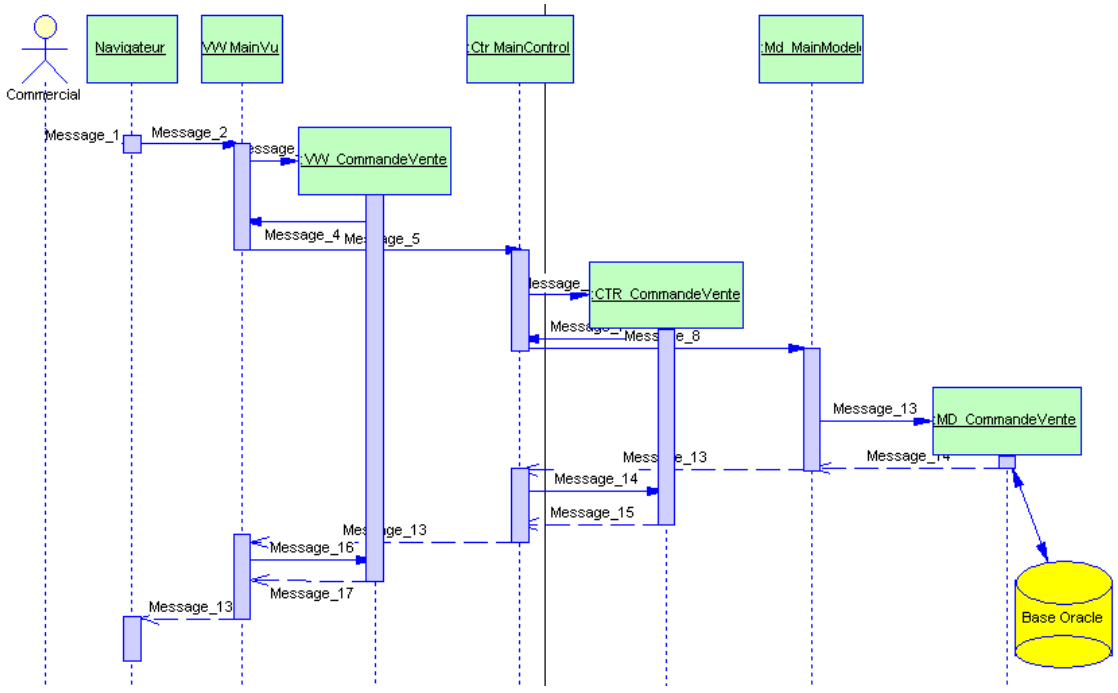
**Fiche descriptive**

| Fonctionnalité | Description |
|---|---|
| Type | Cas d'utilisation |
| Nom | Commande vente |
| Numéro séquentiel | CU002 |
| Acteurs concernés | COOMERCIAL |
| Présupposition | Utilisateur identifié avec succès |
| Pré-conditions | Saisir une période valide **ou rien** |
| Démarrage | A la demande |
| Arrêt | Informations affichées |
| | Echec de connexion à la base de données |
| | L'utilisateur annule sa demande |
| | Le délai d'attente est écoulé |
| Post-condition | Informations affichées |

### *5.2.9 Diagramme de séquence : Comportement dynamique du système*

Le diagramme de séquence de la figure 5.4 constitue une vue dynamique associée au cas d'utilisation commandeVente.

**Figure 5.4**
*Présentation dynamique avec le diagramme de séquence*



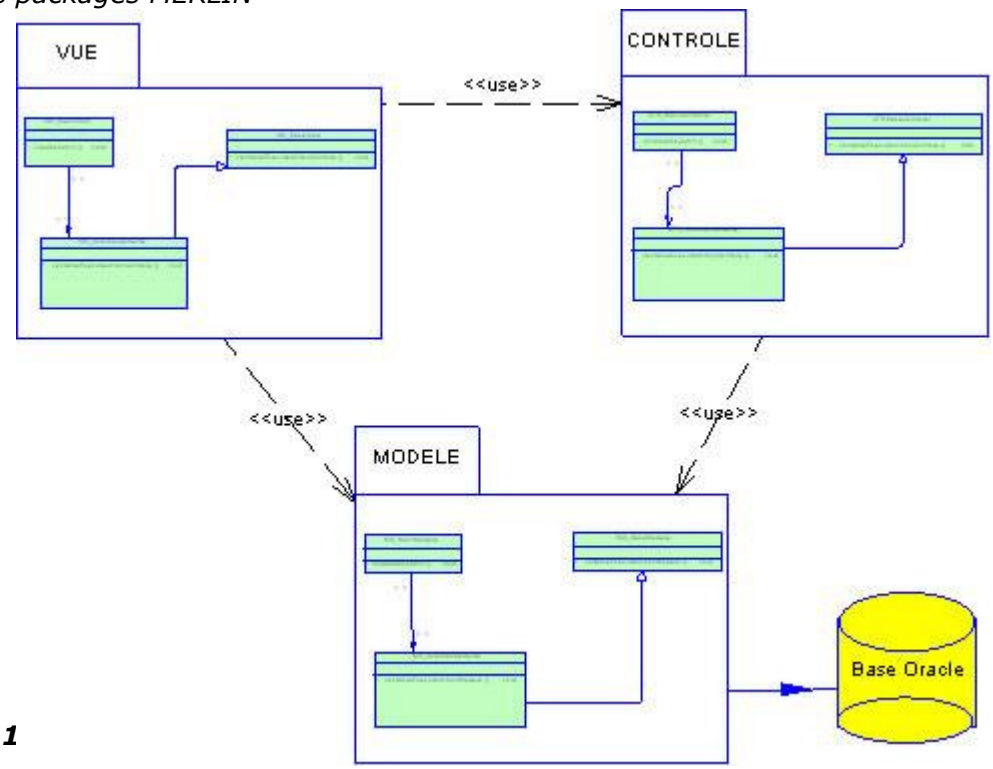
Le diagramme de la figure 5.4 implémente le cas d'utilisation commandeVente, cependant, comme nous l'avons expliqué dans le chapitre 4, l'approche des délégués des packages – utilisée dans ce projet – impose d'utiliser le même scénario d'exécution pour l'ensemble des pages, ce qui change ce sont les implémentations des méthodes métiers et l'utilisation des classes utilitaires.

Ce qui suit est une description textuelle d'un diagramme de séquence valable à tous les flux respectant l'approche des délégués des packages.

- ➔ Utilisateur : Demande d'une page
- ➔ View :
    1. Création d'une instance de la classe mainView.
    2. Exécution de la méthode viewDispatch qui récupère les paramètres utilisateurs.
    3. Création de la classe view pour la page sollicitée par l'utilisateur.
- ➔ Contrôle :
    4. Création d'une instance de la classe mainControle.
    5. Evaluation des paramètres d'entrée par la méthode dispachControle
    6. Création de la classe contrôle pour la page sollicitée par l'utilisateur
- ➔ Modèle :
    7. Création d'une instance de la classe mainModele.
    8. Evaluation des paramètres d'entrées par la méthode dispachModele
    9. Création de la classe modèle pour la page sollicitée par l'utilisateur
    10. Exécution de requêtes SQL pour interroger la base de données
    11. Transmettre les données à la classe de contrôle.
- ➔ Contrôle :
    12. Traitement des données avec les processus métiers implémentées dans les classes de contrôles.
    13. Envoyer le résultat des traitements à la classe view
- ➔ view :
    14. Formatage des données reçues de la classe de contrôle.
    15. Affichage des données dans l'interface utilisateur.

### *5.2.10 Diagramme des packages*

**Figure 5.5**
*Diagramme des packages MERLIN*

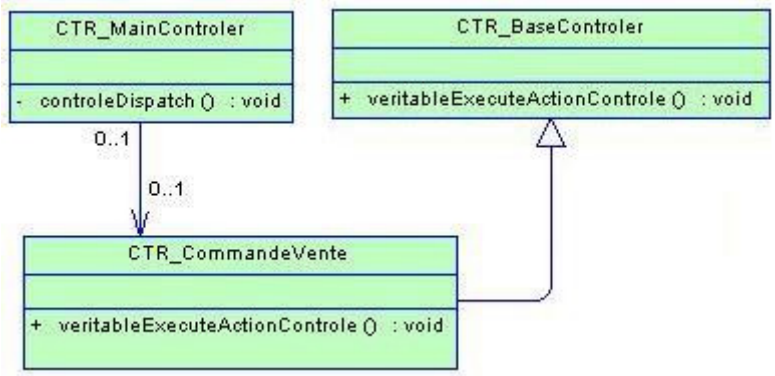



### *5.2.1*

L'implémentation de l'approche des délégués des packages dans ce projet était simple, chaque indicateur se voit attribuer un triplé de classe MVC, et chaque classe de chaque package implémente la méthode principale de la classe de base à savoir veritableExecuteAction et la méthode d'abonnement auprès de la classe main (voir chapitre 5).

#### *5.2.11.1 Diagramme de classe*

**Figure 5.6**
*Diagramme de classe partiel du package CONTROLE (MVC avec les délégués des packages)*



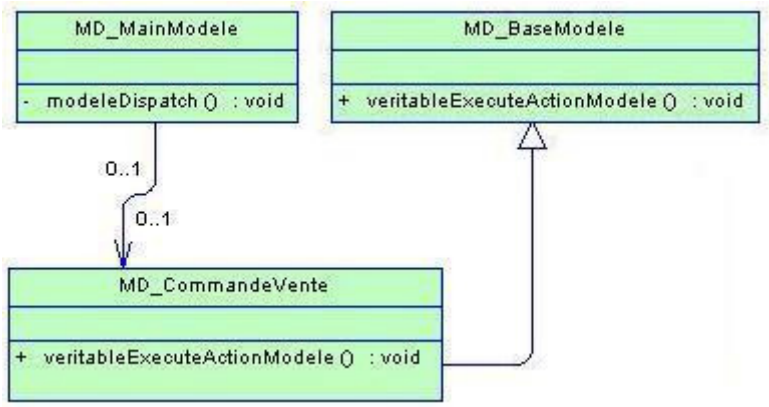

Classe `CTR_MainControler` : Classe déléguée du package CONTROLE.

La méthode `controleDispatch()` dispatche les évènements en provenance des vues vers les différentes classes du package CONTROLE pour implémenter des traitements métiers.

La classe CTR_BaseControler est une classe de référence pour l'ensemble des classes du package, elle implémente des méthodes communes à plusieurs classes du package.

De la même manière les deux autres packages implémentent les mêmes classes.

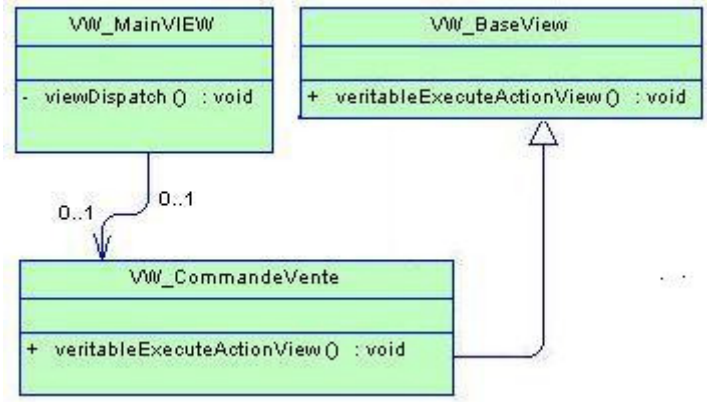
**Figure 5.7**
*Diagramme de classe partiel du package MODELE (MVC avec les délégués des packages)*




Classe `CTR_MainModele` : Classe déléguée du package MODELE.

La méthode modele`Dispatch()` dispatche les évènements en provenance des classes externes vers les différentes classes du package MODELE pour réaliser des opérations sur la base de données.

**Figure 5.8**
*Diagramme de classe partiel du package VUE (MVC avec les délégués des packages)*




Classe `CTR_MainView` : Classe déléguée du package VIEW.

La méthode view`Dispatch()` dispatche les évènements en provenance des classes externes vers les différentes classes du package VIEW pour essayer de comprendre les actions de l'utilisateur et préparer l'affichage des données retournées par le package CONTROLE.

Les classes de délégation sont :

1. `VUE.VW_MainVIEW`
2. `CONTROLE.CTR_MainControler`
3. `MODELE.MD_MainModele`

### *5.2.11.2 Principe d'abonnement des classes d'un package :*

Les classes de délégation reçoivent tous les événements en direction d'un package, l'exécution de la méthode dispatche permet de connaître le destinataire de l'évènement externe. Une fois que le destinataire est connu, la méthode crée une instance de la classe désignée par l'événement externe et lui transmet le contrôle pour réaliser le traitement demandé. Le constructeur de la classe de contrôle doit s'abonner auprès du délégué du package pour que ce dernier accepte le résultat du traitement achevé.

Ce principe est optionnel, néanmoins, il permet de mieux contrôler et de suivre les instances des classes créées, son utilité se manifeste clairement dans le débogage du système.

Une classe abonnée : est une classe dont le délégué garde un pointeur (vers) sur son adresse mémoire.

L'abonnement des classes permet de s'assurer :

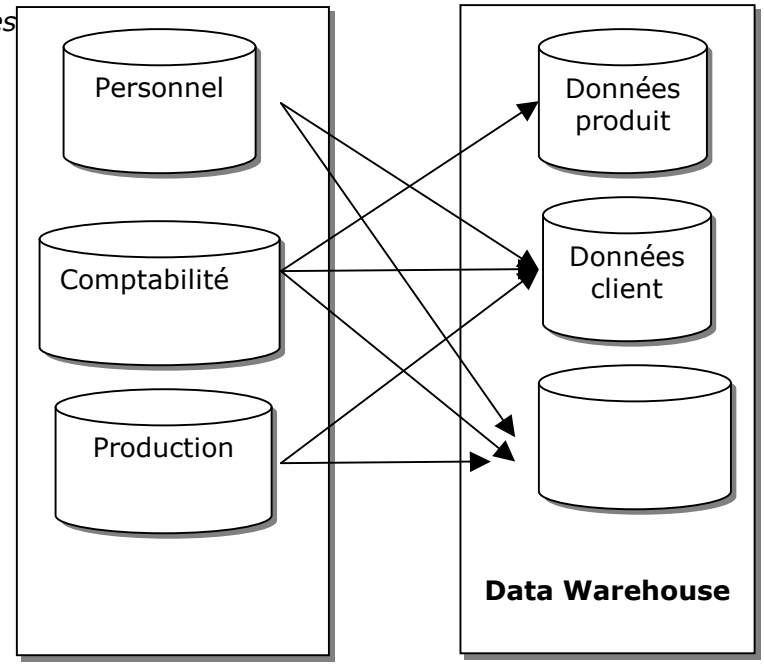
- Un meilleur déboguage du système
- Une meilleure gestion de la mémoire (garbage Collector )

### *5.2.12 Modèle de données :*

Pour pouvoir exploiter facilement les données, nous avons implémenté une modélisation par sujet fonctionnel plutôt que par application. Malgré que pour l'instant il n'existe qu'une seule application utilisant ce modèle cette solution s'aligne sur l'ambition de l'entreprise pour généraliser cette application pour d'autres services. Ainsi l'intégrité des données est garantie et l'accès est réalisé plus efficacement aux données.

Cette solution est commune aux data warehouse car leurs objectifs est de répondre à des questions stratégiques orientées sujet. Et c'est bien ce qui les différencient des bases de données ordinaires ( figue 5.9)

**Figure 5.9**
*Base de données classiques et Data Warehouse*




### *5.2.12.1 Modèle physique de données MPD*

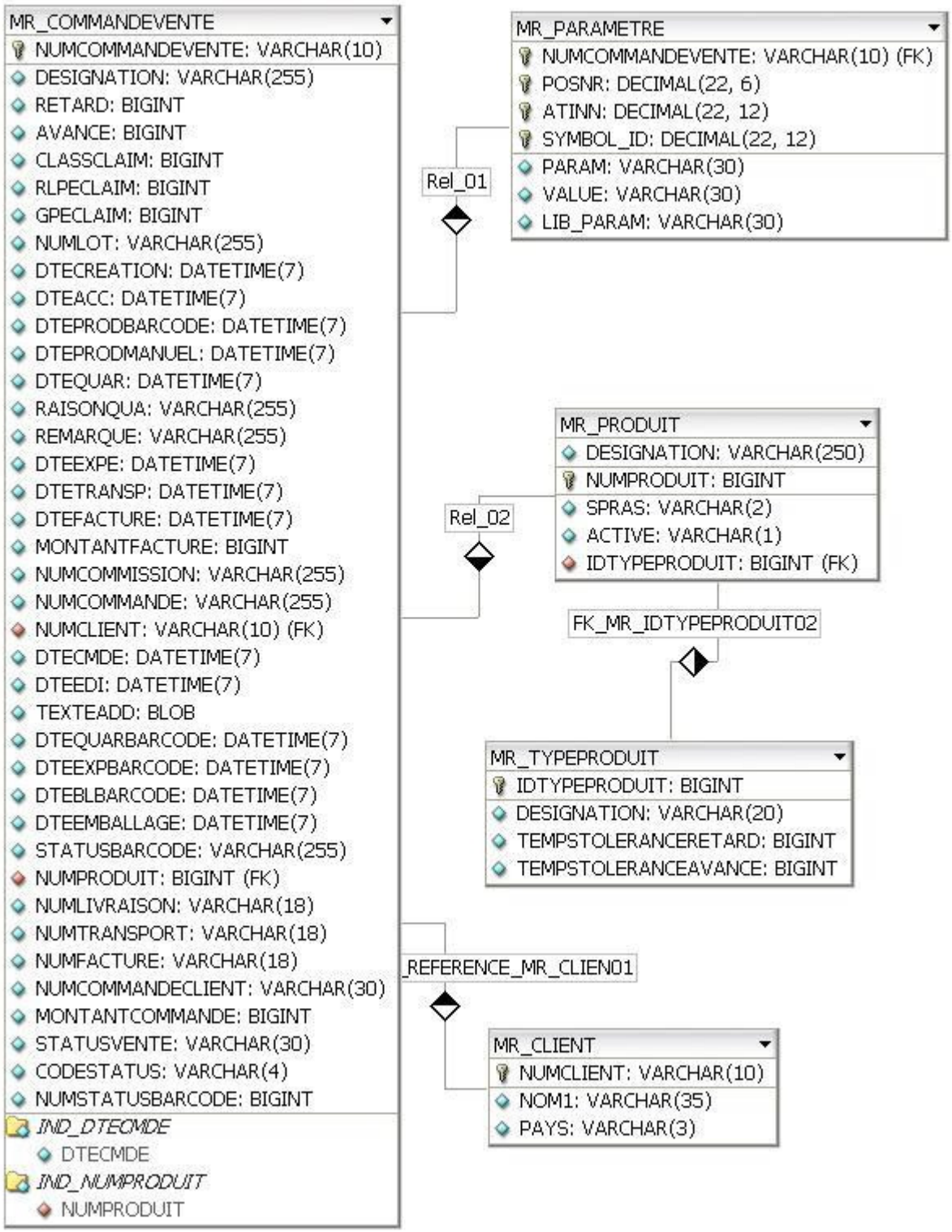
La figure 5.10 montre l'utilisation du modèle en étoile, qui consiste à centraliser l'information dans une table centrale (MR_COMMANDEVENTE) appelée table de *fait* et la relier à des tables secondaires appelées table de dimensions. Le modèle montre la simplicité de la solution adoptée pour implémenter la consistance des données.

Remarquez qu'il existe une table "gigantesque" *MR_COMMANDE*, en fait, nous aurions pu éclater cette relation sur plusieurs relations suivant le modèle relationnel pour respecter les formes normales canoniques (exemple créer une entité *lot)* cependant, nous avons considéré d'autres facteurs dans notre choix que la normalisation :

1. La simplicité des requêtes
2. La fréquence des requêtes sur les attributs d'une entité : exemple : il est inutile de créer une entité pour les lots s'il n'existe pas des demandes de consultation des détails des lots.
3. Respecter le modèle en étoile qui répond au mieux aux besoins des applications Data warehouse.

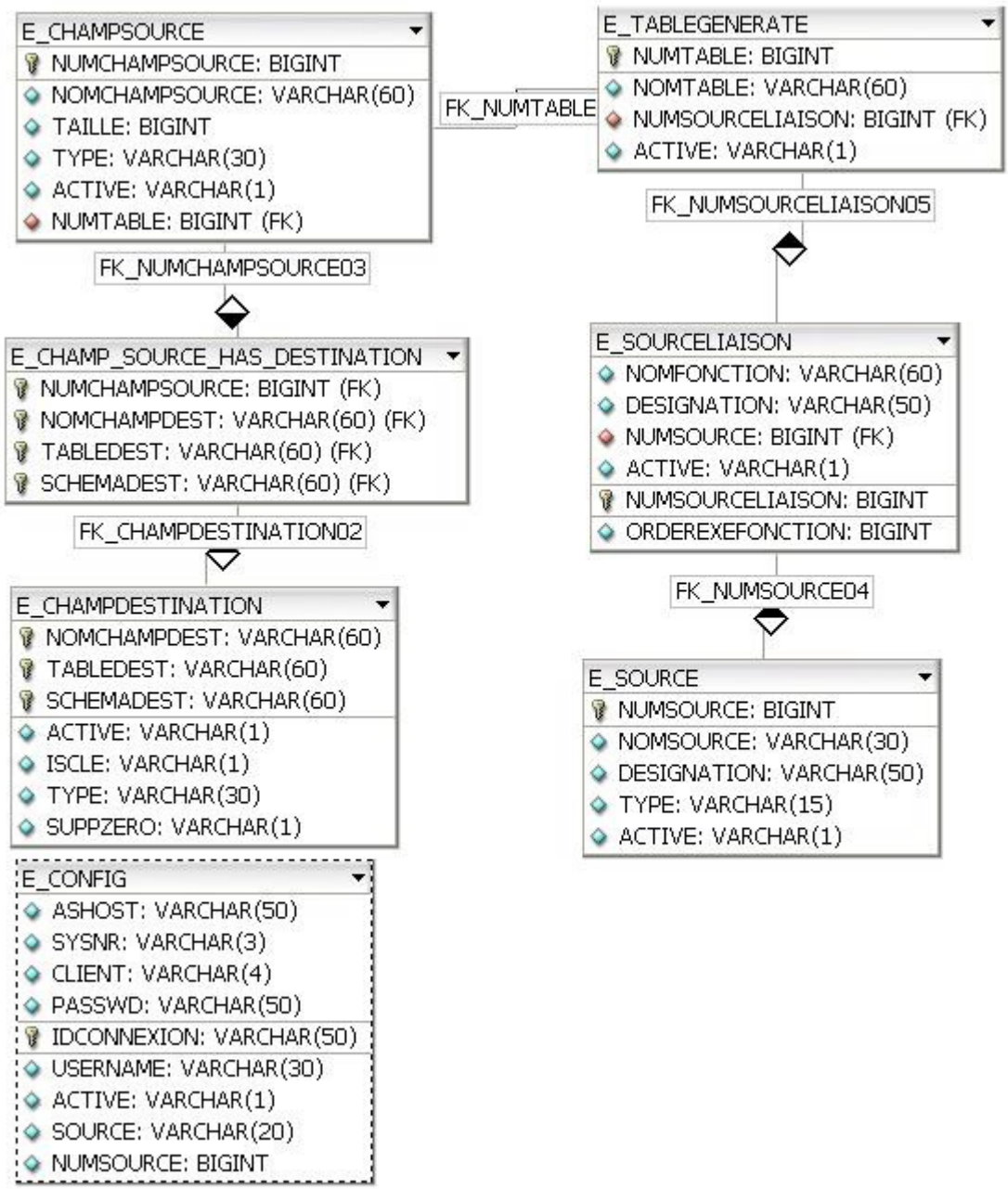
Le modèle en étoile est optimisé pour les requêtes d'analyse et souvent considéré, à tort, comme un modèle dénormalisé, parce que ses objectifs sont plus spécifiques et plus pointus que ceux des bases de données ordinaires.

**Figure 5.10**
*Modèle physique de donnée*
*Modèle en étoile*



La figure 5.11 montre la deuxième partie du modèle physique de données, consacrée à l'ETL (Extract-Transform-Load)

**Figure 5.11**
*Modèle physique de donnée*
*Modèle en flocon "neige"*

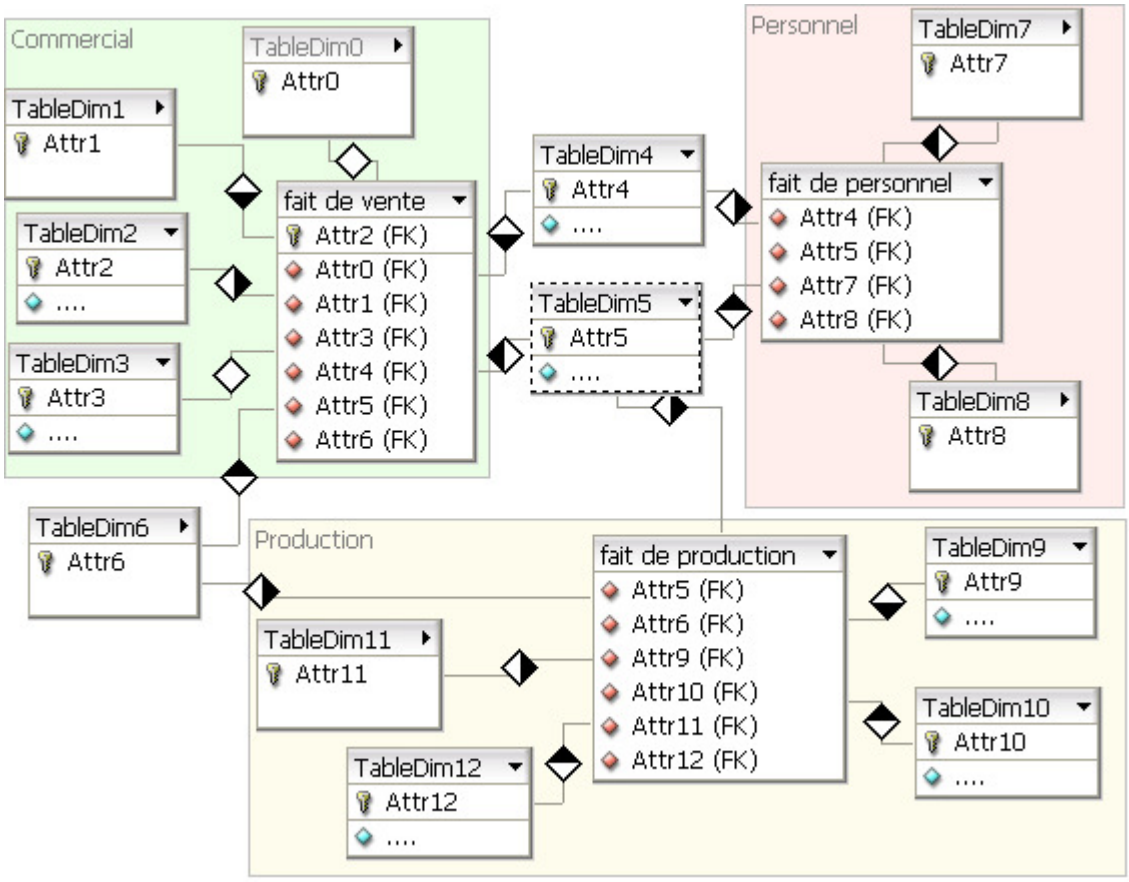


### *5.2.12.2 Evolution du modèle*

Comme nous l'avons mentionné précédemment cette application est fortement sollicitée pour évoluer et prendre forme afin d'inclure d'autres services et couvrir un secteur plus vaste des données de l'entreprise, à ce titre il est indispensable de revoir notre modèle de donnée afin de supporter ces éventuelles évolutions.

Le modèle en étoile vise à réduire au maximum le nombre des tables de fait, cependant l'insertion d'une table de fait s'avère inévitable lorsque l'on se retrouve devant des entités fonctionnement distinctes.

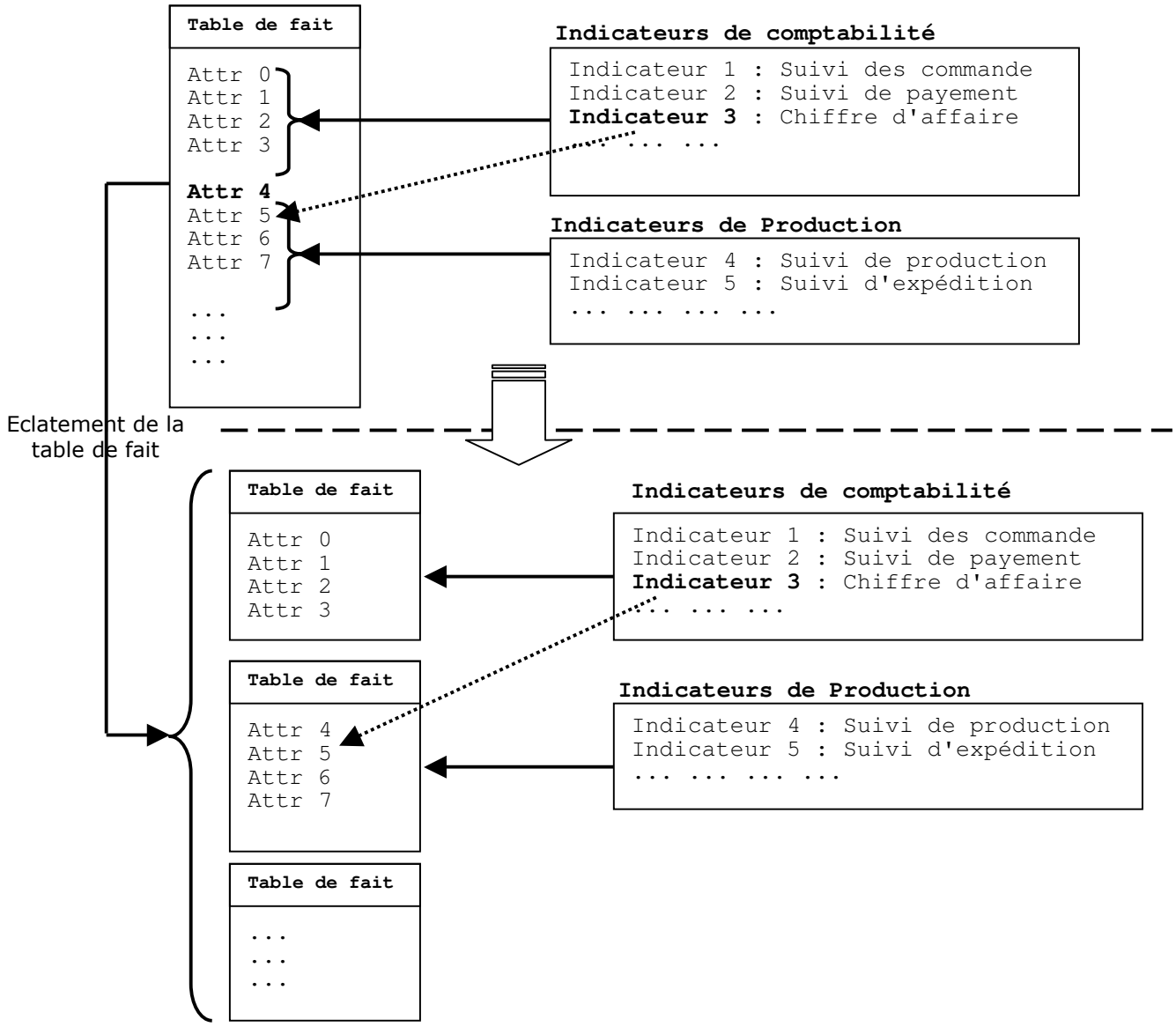
**Figure 5.12**
*Plusieurs tables de fait dans un modèle en étoile*




La figure 5.12 montre l'utilisation de plusieurs tables de fait pour modéliser des vues fonctionnelles différentes de notre modèle et de regrouper les entités suivant les modèles d'analyses établis dans les perspectives du Data warehouse pour répondre au mieux aux attentes des utilisateurs.

La fusion ou l'éclatement des tables de fait dépend, en premier, du regroupement des indicateurs par liste d'attribues intervenant dans l'exécution des requêtes SQL. Ceci revient à essayer de regrouper dans chaque table de fait les attributs clés correspondants à des indicateurs fonctionnellement liés. La figure 5.13 montre cette interdépendance entre groupes d'indicateurs et les attributs des tables de fait.

**Figure 5.13**
*Eclatement des tables de fait*



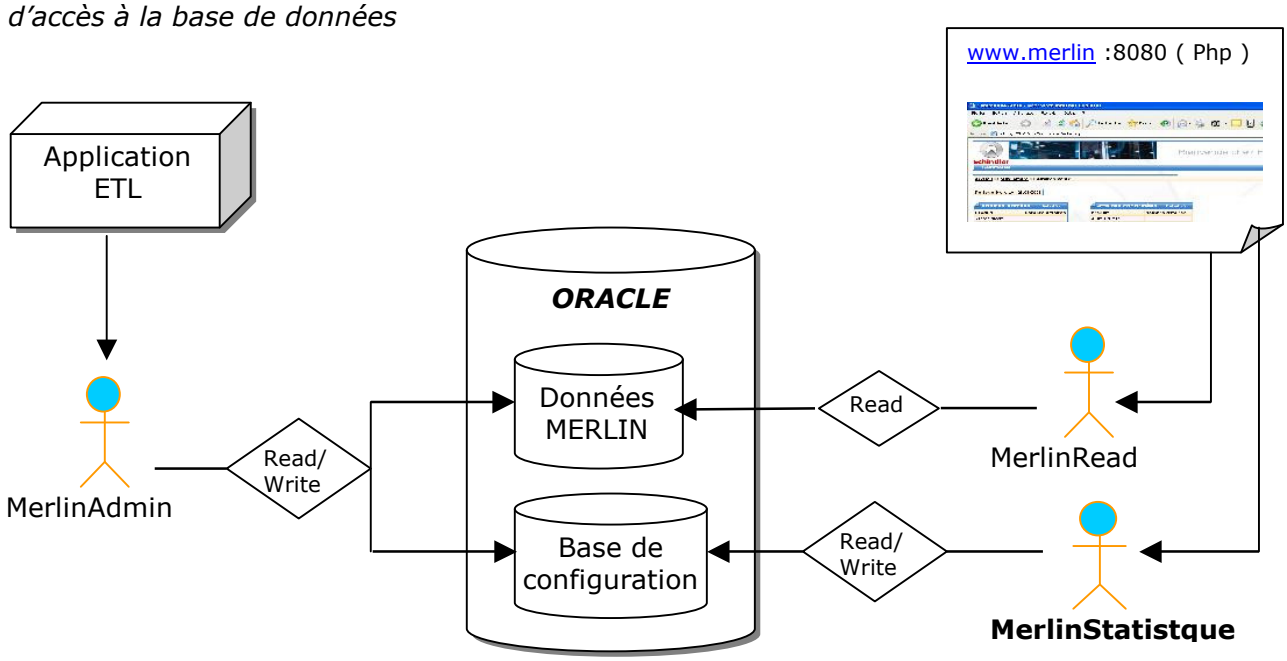

Ce modèle améliore nettement l'efficacité de notre base de données en terme de temps d'exécution pour la plupart des requêtes. Mais cette amélioration souvent est réalisée en dépit de d'autres indicateurs (indicateur 3 de la figure 5.13) qui se voient compliquer les requêtes de sélection, dans ce genre de situation, il faut arriver à trouver un compromis de juste milieu, de manière à isoler les indicateurs secondaires et à favoriser l'efficacité des requêtes pour les principaux indicateurs. A noter que la valeur d'un indicateur se traduit par sa fréquence d'utilisation et sa valeur stratégique dans l'entreprise.

#### *5.2.12.3 Type des requêtes*

Une des principales caractéristiques des Data warehouse, est que les données sont non volatiles et leur enregistrement est définitif. En pratique, il est strictement interdit d'exécuter des requêtes de mise à jours ou de suppression sur les données du data warehouse par l'utilisateur, mis à part les requêtes de sélection qui sont autorisées. Pour remédier à ce problème nous avons utilisé une technique ORACLE

en créant plusieurs profils utilisateurs spécialisés, la figure 5.14 montre l'utilisation de ces profils

**Figure 5.14**
*Profils d'accès à la base de données*




Trois profils sont implémentés pour moduler les droits d'accès suivant le besoin et suivant l'application hôte :

**MerlinAdmin** : dispose d'un contrôle total sur la base de données, utilisé par l'application ETL pour alimenter la base de données de différentes sources et pour modifier la configuration de la base de l'ETL.

**MerlinRead :** dispose d'un droit de lecture sur la base Merlin contenant les données de notre Data warehouse. Accéder à la base à travers ce profil n'ouvre le droit qu'à des requêtes de sélection. Ce profil est utilisé par l'application de consultation via un navigateur web.

**MerlinStatistique :** Dispose d'un contrôle total sur la base de configuration de l'ETL qui contient des statistiques de consultation (nombre de connexion, répartition des tâches, fréquence d'utilisation des indicateurs…)
Pourquoi ce profil ?
On aurait pu utiliser le profil MerlinAdmin pour accéder à cette base de données, mais cette application est accessible de l'extérieur par des utilisateurs multiples ce qui augmente le risque d'une utilisation malveillante.

Cette technique a l'avantage de déléguer l'assurance de la sécurité des données de notre système à l'SGBD ORACLE qui constitue une référence en la matière.

### *5.2.13 Interface (Ecrans)*

L'interface graphique représente aux yeux des utilisateurs un élément d'évaluation très présent, ainsi un système bien élaboré au sens du métier, mais qui néglige sa présentation, est soumis inévitablement à l'échec.

Nous avons essayé de respecter les critères ergonomiques exposés dans le premier chapitre à travers un processus de validation des maquettes auprès des utilisateurs et nous avons établi aussi un questionnaire type pour vérifier l'application de ces critères.

Comme l'application est destinée à un usage purement professionnel, nous avons adopté une interface sobre à caractère professionnel visant à mettre en avant les attentes des utilisateurs. Le moteur de recherche change de critère d'une page à l'autre pour répondre à des recherches personnalisées.

Etant donné que le résultat de l'ensemble des indicateurs sont des chiffres calculés à base des données de l'entreprise, nous avons intégré une fonctionnalité d'export vers Excel pour réaliser des éventuels traitements sur ces résultats.

Un seul menu de navigation, permet d'atteindre n'importe quelle page des indicateurs, et depuis ces pages l'utilisateur peut accéder au résultat de ces recherches.

La figure 5.15 affiche un certain nombre d'indicateurs (Statut lancement : Entrées, traitées, bloquées). Les indicateurs sont calculés sur des périodes paramétrables par l'utilisateur.

**Figure 5.15**
*Ecran : Affichage de plusieurs indicateurs*

Cet écran est un aperçu de la capacité de notre système à agréger de grande quantité de données en provenance de plusieurs sources de données pour présenter le résultat sous forme d'informations très utiles à la prise de décision.

Dans le même esprit de sobriété et de professionnalisme l'écran suivant (figure 5.16) représente un masque de recherche des « commandes entrées » et le détail d'une recherche.

**Figure 5.16**
*Ecran : Détail d'un indicateur*

### *5.2.13.1 Module des statistiques*

Comme nous l'avons mentionné dans le modèle de donnée, il existe une base de données dédiée à l'administration du site et à la sauvegarde des statistiques sur la fréquence d'utilisation du site par les utilisateurs. Ici aussi, on retrouve le principe des indicateurs pour présenter cette information (La figure 5.17)

Le principal objectif de garder ces statistiques est de promouvoir les indicateurs peu consultés en s'approchant des utilisateurs pour comprendre et écouter leurs propositions d'améliorations.

**Figure 5.17**
*Ecran : Statistiques de consultation*

### *5.2.14 Avantages constatés avec l'utilisation de l'approche des délégués des packages*

Comme nous l'avons mentionné dans le chapitre 4 l'approche des délégués des packages est rigoureusement exigeante dans la phase de conception, chose qui consomme beaucoup de temps alors que le planning de ce projet été très serré. Le seul argument qui a fait pencher la balance en faveur de cette approche est la perspective d'évolution de ce projet pour inclure d'autres services de l'entreprise. Effectivement, avec cette approche - purement orientée objet – l'insertion d'un nouvel indicateur devient (après sa validation au sens métier) presque une étape techniquement mécanique, il suffit de respecter les recommandations techniques de l'approche, d'où la perspective de concevoir un EDI dédié qui permettrait l'encapsulation de la mise en place d'une architecture MVC respectant l'approche des délégués des packages.

## 5.3 Site WEB AFPLemans

### *5.3.1 Préface*

Le deuxième projet que nous présentons dans ce chapitre est un site Web réalisé pour le compte de l'entreprise AFP Le Mans. Vous trouverez dans cette partie une brève description de la structure du site et plus en détail le fonctionnement de l'éditeur des contenus intégré dans le site pour faciliter la mise à jour du contenu du site.

Nous ne nous concentrerons pas sur le contenu du site mais surtout sur le coté technique au sens informatique, c'est à dire les technologies utilisées et l'implémentation d'un « éditeur des contenus » qui sera l'objet principal de cette partie.

### *5.3.2 Contexte*

L'AFP menée par le souci de donner une nouvelle image, qui reflète les changements structurels au sein de son organisation et un nouveau moyen à ses clients de bénéficier des opportunités de communication sur le Web. Les clients de l'AFP, sont des demandeurs d'emplois, des employés d'entreprises, des étudiants ayant différents niveaux d'études désirant acquérir une formation professionnelle ; le défi est de répondre aux besoins de toute cette variété de clients potentiels.

Le site devra présenter l'institution mais surtout répondre aux attentes des internautes cherchant une solution à leur problématique de formation. L'objectif fondamental du site de l'AFP, est d'augmenter son efficacité en permettant aux diverses catégories de clients de trouver rapidement les formations qui répondent le mieux à leurs attentes et de pouvoir télécharger des formulaires d'inscriptions à ces formations.

Le site est destiné à être actualisé fréquemment pour mettre à jour la liste des formations et le contenu informationnel, cependant, les personnes chargées de cet entretien et évolution continue du site, ne sont pas forcément des informaticiens c'est pourquoi on doit leur fournir un moyen facile et intuitif ne nécessitant aucune intervention au niveau du code.

### *5.3.3 Technologies utilisées*

Nous avons utilisé plusieurs technologies pour mettre en place ce site :

- PHP : pour les scripts de création et de navigation dans les fichiers XML.
- Flash : avec son langage de programmation propriétaire orienté objet *ActionScript dérivé du langage C* dédiée pour la manipulation et le contrôle des animations Flash. Flash constitue le principal support d'affichage du site, le noyau de l'éditeur des contenus est développé sous FLASH.
- XML : pour enregistrer les contenus dynamiques du site édités par l'éditeur des contenus.
- MySQL : la quantité des données traitées dans le cadre de ce projet en prenant en compte les éventuelles évolutions prévues est relativement modeste, dans cette perspective nous avons choisi d'utiliser MySQL car il est gratuit et assez puissant pour gérer une telle quantité de données et de fréquence d'utilisation.

### *5.3.4 Choix de la technologie XML*

Effectivement le choix d'XML n'était pas un hasard, comme nous l'avons vu dans le chapitre 2 qui lui a été consacré, le langage XML offre une palette de nouveautés efficacement exploitées et testées pour réaliser diverses applications robustes et de grandes ampleurs. Notre expérience avec XML avait débuté avec des applications classiques sous le langage Java (qui ne sont pas l'objet de ce thème), elle nous a permis de découvrir sa puissance et son utilité. Nous avons su après l'étude des besoins qu'XML répondra au mieux à l'implémentation des attentes des utilisateurs envers leur futur site web.

### *5.3.5 Structure du site*

Nous avons repris un classique de structuration des applications à contenus dynamiques, à savoir, les deux modules *Back Office et* Front office :

- Back Office (le module d'administration) : regroupe toutes les fonctionnalités d'administration du site et de la base de données. Cette partie du site est destinée à être utilisée en interne pour la mise à jour du site, elle est protégée par une étape d'identification. La mise en place de cette partie était la plus longue dans les phases de réalisation de ce projet.
- Front office : constitue le site proprement dit aux yeux des internautes, c'est l'interface WEB du site sans restriction d'accès.

Ce choix se justifie par l'existence - bien clair - de ces deux catégories de fonctionnalités dans notre site web, cette séparation entre les deux parties est à l'origine de l'implémentation de l'éditeur des contenus pour la partie back office.

### *5.3.6 Intégration des moyens de modification dynamique*

Nous détaillerons dans cette partie quelques principes en relation avec le sujet de ce mémoire à savoir l'*intégration des moyens de modification dynamique.*

En quoi consiste un moyen de modification dynamique ?

Il n'existe aucune définition formelle à cette expression, cependant, on peut classer dans cette catégorie tous les moyens et techniques utilisées en vue de faciliter et d'améliorer la modification et la mise à jour des sites web. Plusieurs firmes de software concentrent des efforts importants pour innover de nouvelles approches visant à augmenter le potentiel du web.

Exemples de moyens de modification dynamique :

- Structuration conventionnelle de site
- Normalisation des interfaces utilisateurs.
- Suppression ou diminution du recours à la modification du code pour la mise à jour de la structure ou du contenu des sites web
- Génération automatique des scripts de modifications.
- ….

Le marché des outils destinés au développement web constitue un des plus grands marchés en terme du nombre d'utilisateurs et il est en constante évolution en gagnant chaque jour un nouveau public. Cette constatation s'affirme devant le

nombre colossal des sites web développés à travers le monde. Plusieurs explications peuvent être données pour cette tendance, mais la plus pertinente est celle qui met la lumière sur l'effort réalisé par les concepteurs de ces outils pour s'approcher de leurs utilisateurs les plus modestes au sens technique, en leurs offrants des environnements intégrés allant jusqu'aux ateliers du génie logiciel intégrant des outils de conceptions graphiques encapsulant la complexité des langages de programmation.

L'utilisateur (développeur web) se voit dégager de l'effort d'apprendre la syntaxe et la sémantique des langages de programmation, ainsi il concentrera son effort sur les règles du métier et la matière de son site web. Cette productivité encourage les utilisateurs à continuer et à attirer de nouveaux utilisateurs.

Il existe un autre point très important pour la survie des sites web. Un contenu peut devenir très vite désuet, en effet, le contenu et la présentation d'un site web sont souvent sujettes à des modifications pour assurer leur pertinence et cohérence. Cette étape et très lassante et coûteuse si l'on doit (fouiller) dans les piles de code. Les développeurs des sites préconisent des outils de développement intégrant la possibilité de modifier le site dynamiquement, sans passer par des transferts FTP.

Ce que nous proposons dans le site développé est d'intégrer directement sur le serveur un éditeur permettant la modification en ligne du contenu sans passer par des transferts FTP.

### *5.3.7 Exemple d'outils facilitant l'édition des contenus :*

Macromedia Dreamweaver, Macromedia Fireworks, SUN Java Studio Creator, CodeCharge, WebEdit. Tous ces outils fournissent beaucoup de facilité pour la création et la mise à jours des sites web.

### *5.3.8 Editeur des contenus intégré*

Dans la première version, nous avons proposé de séparer l'administration du site dans une application à part, mais avec l'avancement de la phase de conception nous nous sommes aperçus que les modules de modifications se chevauchent fortement avec les modules de lecture, c'est pourquoi nous avons commencé à penser autrement d'où la naissance de l'idée de fusionner les deux modules (Administration et Affichage) en une seule et unique application.

C'est quoi l'éditeur intégré des contenus ?

L'éditeur des contenus est un programme informatique, pour la génération dynamique des sites web. Sa spécificité est son intégration avec le site web, c'est à dire physiquement, le programme est transféré sur le serveur hôte du site, ainsi pas besoin de faire des transferts FTP pour apporter des modifications au contenu du site. Accessible directement avec un navigateur web et sécurisé par un mot de passe il permet d'administrer le site web et la base de donnée, utilise la technologie XML pour sauvegarder les données et la description de l'interface du site.

La puissance de l'éditeur des contenus :

- Sa capacité à traiter plusieurs formes de fichier (animation, image, texte)

- Basé sur la technologie XML, avec toute son ouverture sur plusieurs langages de programmation.
- La création rapide des sites web
- Possibilité de réaliser facilement des modèles de sites génériques, destinés à êtres paramétrés par les utilisateurs pour créer leurs propres sites web.
- Faciliter l'entretien des sites web.
- Permettre interopérabilité avec d'autres systèmes de gestion de sites web en définissant et respectant un protocole d'échange avec les schémas XML.
- Archivage facile des versions, il suffit de garder une copie des fichiers XML générés.

Inconvénient

- L'éditeur, avec sa version actuelle n'est pas riche en terme de fonctionnalité, cependant, la perspective d'enrichir la bibliothèque utilitaire par de nouvelles fonctionnalités et l'ouverture sur d'autres formats de fichier met en valeur l'idée de cet éditeur.
- La mise en place de nouvelles fonctionnalités est très lourde et demande beaucoup de temps pour prendre en compte les variétés des langages de programmation et des normes du développement web.

#### *5.3.8.1 Ce qui est possible avec l'éditeur des contenus*

Modifier trois types de contenu :

- Les zones de texte : avec l'éditeur des contenus, il est possible de créer/modifier autant de zones de texte que l'on veut. Changer leur position sur les pages du site.
- Les animations : possibilité de changer la position, le contraste et la transparence.
- Les images : possibilité de changer la position, le contraste et la transparence.

#### *5.3.8.2 Fonctionnement de l'éditeur des contenus*

L'éditeur est accessible dans le mode administration du site (Back office) protégé par mot de passe. Ce mode reprend la même interface de visualisation du site, ainsi l'administrateur peut visualiser un aperçu de ses modifications avant de les valider.

### *5.3.8.3 Architecture générale*

**Figure 5.18**
*Architecture générale*

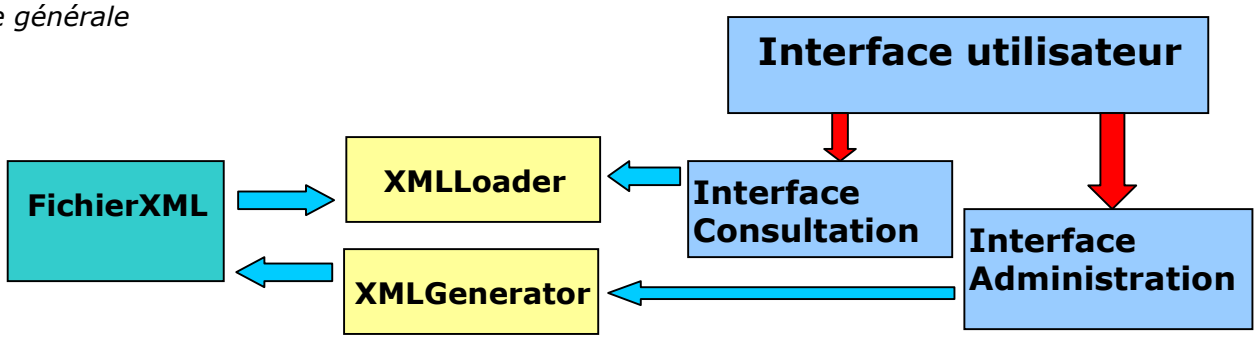
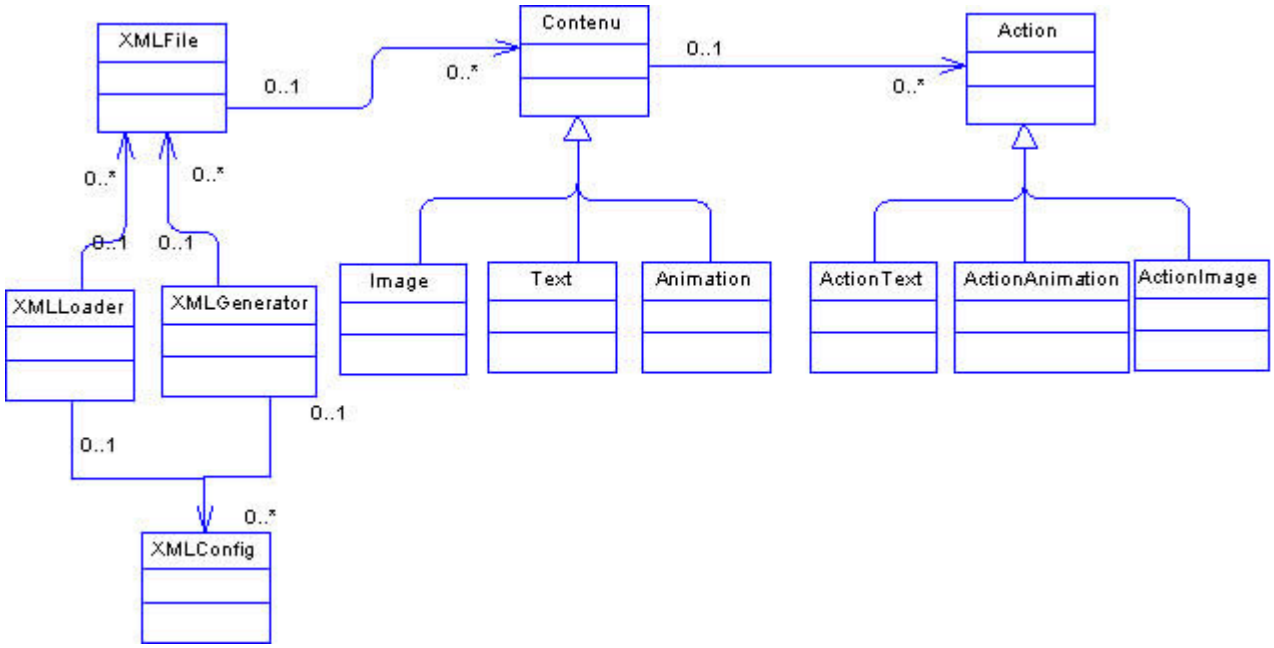



**Figure 5.19**
*Une partie du diagramme de classe de l 'éditeur des contenus*




### *5.3.8.4 Description des principales classes*

**package org.merlin.xmlManager**

**Classe XMLFile**

Classe mère des classes des traitements liés aux fichiers XML.

**Classe XMLLoader**

Cette classe a pour rôle de charger les fichiers XML associés aux pages du site, le chargement du fichier XML consiste à parcourir tous les nœuds et de les charger en mémoire.

**Classe XMLGenerator**

Utilisé par l'éditeur des contenus pour générer de nouveaux fichiers XML ou de modifier des existants

**Classe Contenu.**

Classe mère des types de contenus que *l'éditeur des contenus* peut manipuler. Toute insertion de nouveaux types doit hériter de cette classe.

**Classe Action**

Classe mère des actions (lecture, déplacement, suppression …) que peuvent réaliser les objets de type contenu.

Remarque :

Pour la modélisation de cette application nous avons utilisé PowerMAc de Sysbase.

### 5.3.9 *Le fichier XML*

L'éditeur des contenus doit sa flexibilité à l'utilisation du langage XML pour enregistrer la description de notre site web. Pour assurer un éventuel traitement conventionnel des données des pages du site, tous les fichiers XML respectent le schéma de la figure 5.20. Ainsi la figure 5.21 donne un aperçu de ce que peut être une partie de notre fichier XML.

(voir le chapitre 2 et annexe B pour plus d'informations sur les schémas XML )

**Figure 5.20**
*Aperçu du schéma XML des fichiers XML générés par l'éditeur des contenus*

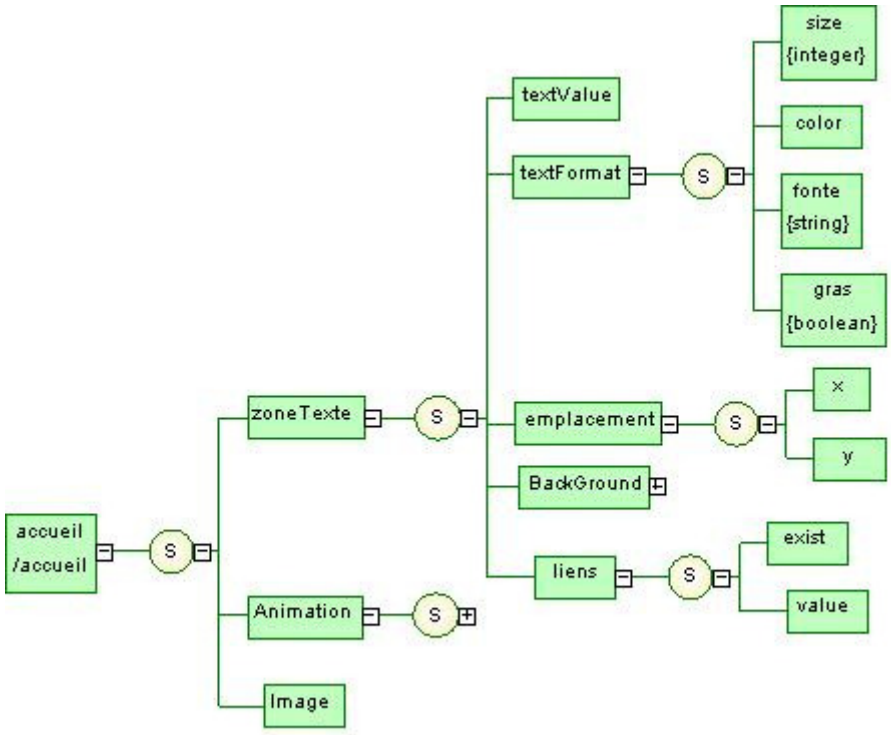
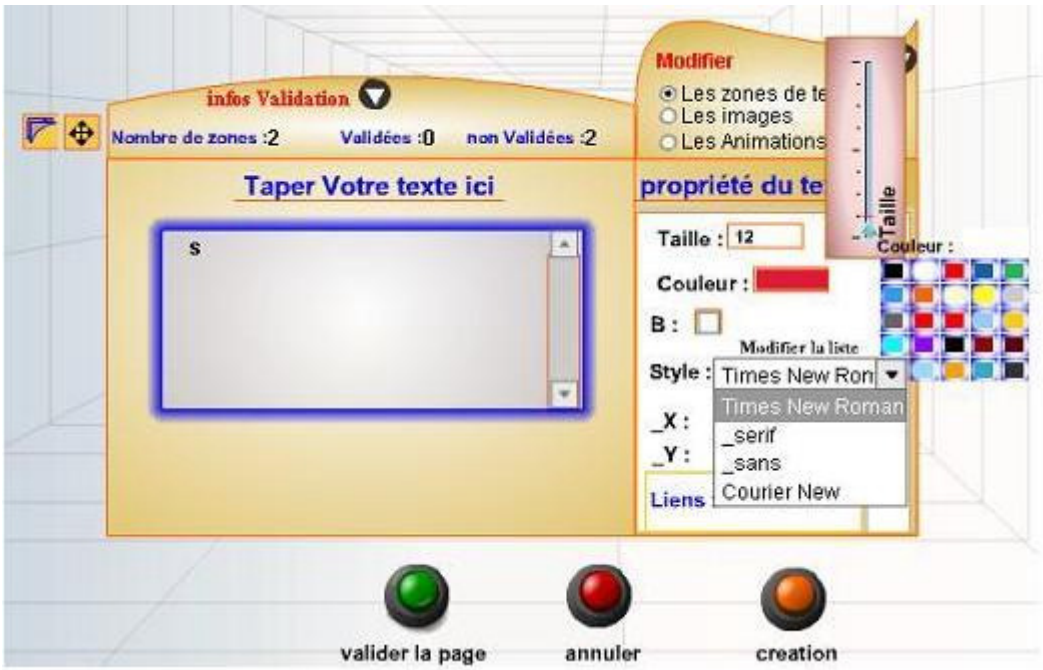


**Figure 5.21**
*Exemple d'un fichier XML générés par l'éditeur des contenus*

```
<accueil>        <---------- Racine du document
    <zoneTexte>  <----------  Englobe les éléments zone de texte
        <zone><----------  Nouvelle zone
            <textValue>Page D&apos;accueil</textValue><----  Contenu textuelle/
            <textFormat><----------  Format du texte
                <size>24</size><----------  Taille
                <color>3284170</color><----------  Couleur du texte
                <fonte>Times New Roman</fonte><----------  Style de la police
                <format>false</format><----------  Type gras
            </textFormat><----------  Fin de la balise « Format »
            <emplacement><----------  Emplacement de la zone du texte
                <x>116.8</x><----------  Position X de la zone du texte
                <y>16.2</y><----------  Position Y de la zone du texte
            </emplacement><----------  Fin de la balise emplacement
            <BackGround>><----------  Background de la zone de texte
                <arrPlan>false</arrPlan><----------  existe il un arrière plan
                <scrollBar>false</scrollBar><----------  existe il un une scrolbare
            </BackGround><----------  Fin de la balise background
            <liens><----------  Possibilité de mettre le texte comme hyper lien
                <exist>false</exist><----------  nous sommes ou non en présence d'un lien
                <valeur/><----------  si oui le quel
            </liens><----------  Fin de la balise lien
        </zone><----------  Fin de la balise zone
    </zoneTexte><----------  Fin de la balise
</accueil><----------  Fin du document
```

### *5.3.10 Interface de l'éditeur des contenus*

**Figure 5.22**
*Ecran : L'éditeur des contenus*




La figure 5.22 montre l'architecture globale du site pour la partie Front office. Le cœur de tout le système est l'animation principale, qui gère toutes les transactions en faisant appel à des scripts de configuration « *.as »(ActionScript) et des scripts PHP, pour l'interrogation de la base de données.

Cette structure constitue un privilège pour la création dynamique des contenus. Le faite de fractionner les éléments sur des modules séparés fournis un moyen de contrôle sélectif. Selon le besoin et la position des requêtes, des chargements partiels se déclenchent.

Les fichiers XML, chargés à la demande, contiennent les données et les informations nécessaires à l'affichage des contenus, utilisés pour la création des zones de textes au moment du chargement.

## Conclusion

Dans ces travaux, nous nous sommes intéressés au domaine de conception et de développement des applications web, pour répondre à la problématique exposée dans l'introduction : *réduire la distance entre la théorie et la pratique dans les modèles de conception.* Nous avons introduit deux concepts, à savoir l'approche des délégués des packages et l'éditeur des contenus, deux propositions qui s'alignent sur la tendance essentielle de l'informatique à faciliter l'utilisation de l'outil informatique et d'accroître le potentiel des utilisateurs, en leur offrant des solutions logiciels répondant au mieux à leurs besoins et garantissant, en même temps, fiabilité et professionnalisme.

Depuis le début de cette thèse, un nombre important de projets ont vu le jour, lancés par des éditeurs de logiciels notamment Microsoft avec le framework .Net, SUN avec la nouvelle technologie JSF (JavaServer Faces successeur de STRUTS et JSP) CodeCharge de Yes Software, chaque solution visant à répondre à certaines exigences techniques. Tous ces produits implémentent des méthodes différentes d'édition dynamique des contenus.

Le contexte industriel pratique dans lequel les projets ont été réalisés bénéficie de l'approbation et des critiques des utilisateurs, ainsi que de leurs propositions d'amélioration et d'évolution. Coupler recherche et application rapporte des résultats exploitables et rapproche la théorie des applications du terrain, ce qui instaure une motivation soutenue et une contribution complémentaire. Le contact continu avec les utilisateurs potentiels des solutions proposées, permet de les impliquer dans les projets et d'éviter des spirales inutiles dans le processus de développement et assure une validation progressive des étapes de réalisation.

Nous avons pu constater l'utilité de l'implémentation de l'approche des délégués des packages à travers la mise en place d'un Data Warehouse. Dès la première version du système, nous étions amenés à apporter des modifications, fonctionnellement séparées. La clarté et l'homogénéité de l'architecture de notre système, nous à permis de cibler rapidement les zones concernées par les modifications, contribuant ainsi à la maintenance et à l'évolution du système.

A travers ce modeste travail mené sur plusieurs années de recherche et d'application dans différent domaine, l'approche d'édition dynamique a donné un résultat très satisfaisant dans deux projets de grande ampleur ; le premier avec l'utilisation d'un éditeur de contenu intégré avec la technologie XML, et le second avec l'utilisation intégrale de l'approche des délégués des packages. Nous avons constaté l'utilité de cette approche dans la facilité des évolutions des premières versions.

*Perspectives*

Ce projet s'ouvre sur des perspectives prometteuses ; jusqu'à présent les applications développées ont été réalisées sous des éditeurs non dédiés comme PHPEdit, Dreamweaver, codeCharge, nous espérons mettre en place un EDI avec un framework intégrant l'approche des délégués des packages pour faciliter et accélérer la génération du code natif. Une autre perspective consiste à proposer des modèles prédéfinis à partir de cette approche pour servir de point de départ à des solutions logiciels complexes dans différents domaines ( Système d'information, workflow, data warehouse ...)

Nous espérons mener une étude plus approfondie des designs patterns existants afin de dégager leurs points forts et de les mettre à profit dans l'architecture de notre approche des délégués des packages.

L'éditeur des contenus dans sa version actuelle ne contient pas beaucoup de fonctionnalités et ne supporte que quelques formats de fichiers. L'objectif est d'essayer d'enrichir cette bibliothèque en gardant toujours le principe d'édition dynamique.

# Annexe A : Data warehouse

## Présentation

*Data Warehouse (DW)* ou **entrepôt de données**, est un concept spécifique de l'informatique décisionnelle, issu du constat suivant : les données de l'informatique de production (également appelée « informatique transactionnelle »), notamment les progiciels de gestion intégrés (ou ERP, *Enterprise Resource Planning*) ne se prêtent pas à une exploitation dans un cadre d'analyse décisionnelle. Les systèmes de production sont en effet construits dans le but de traiter des opérations individuelles qui peuvent impliquer différents métiers de l'entreprise et surtout, ne se préoccupent pas de leur compilation ou historisation dans le temps. À l'inverse, les systèmes décisionnels doivent permettre l'analyse par métiers ou par sujets et le suivi dans le temps d'indicateurs calculés ou agrégés. Il est donc souvent indispensable de séparer ces deux mondes et de repenser les schémas de données, ce qui implique l'unification des différents gisements de données de l'entreprise en un **entrepôt de données** global (*data warehouse*) ou dédié à un sujet/métiers (*datamart*).

> « Le Data WareHouse est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision. » [WH.Inmon96]

Un **datamart** est un *magasin* de données. Comme le data warehouse c'est un entrepôt de données, mais spécialisé pour un métier ou un sujet donné. En règle générale, il se situe *en aval* d'un data warehouse plus global et est alimenté à partir de ce data warehouse, dont il constitue en quelque sorte un extrait. Le datamart est un point de vue spécifique, déterminé par des critères fonctionnels métier. Le datamart est conçu d'après un besoin d'affaire identifié et pour une portée clairement définie. Ainsi plusieurs datamarts pourront être conçus pour répondre chacun à un besoin différent. Le datamart est une base de données décisionnelle structurée et formatée en fonction d'un métier précis ou d'un usage particulier (datamart commercial, datamart financier, ...). L'information y est préparée pour être « consommée » telle quelle : c'est une structure plus spécifique qu'un data warehouse qui reste, lui, à vocation plus générale et plus transversale au regard des différents métiers de l'entreprise. [wikipedia]

Le datamart sert au reporting et à l'analyse décisionnelle, en bout de la chaîne de l'informatique décisionnelle.

L'entrepôt de données de type datamart a une structure de données :

- Soit multidimensionnelle stockée dans un cube ou hypercube : M-OLAP, R-OLAP ... (notamment lorsqu'il s'agit de données agrégées)
- Soit représentée par un modèle de données en étoile ou en flocon dans un SGBD relationnel (notamment lorsqu'il s'agit de données élémentaires ou unitaires non agrégées).

Exemples d'informations décisionnelles :

- Chiffre d'affaire par client
- Progression des ventes dans un intervalle de temps

- Evolution de la part du marché d'un produit particulier
- Evolution de la part du marché par région
- Taux d'absentéisme des employés par secteur de production
- …

« *Un data warehouse ne s'achète pas, il se construit.* » [WH.Inmon96]

Un DW est une structure informatique dans laquelle est centralisé un volume important de données consolidées à partir des différentes sources de renseignements d'une entreprise (notamment les bases de données internes) et qui est conçue de manière que les personnes intéressées aient accès rapidement à l'information stratégique dont elles ont besoin. Si, dans le passé, l'entrepôt de données servait davantage à l'archivage, aujourd'hui il est devenu une pièce maîtresse de l'informatique décisionnelle (ou informatique d'aide à la décision). Il représente l'un des éléments essentiels d'un ensemble matériel et logiciel dynamique de recherche d'informations. Dans un entrepôt de données, les données sont : sélectionnées et préparées (pour répondre aux questions vitales de l'entreprise), intégrées (à partir des différentes sources de renseignements) et datées (elles gardent la trace de leur origine). Le terme entrepôt de données, employé très fréquemment, semble vouloir supplanter ses concurrents dépôt de données et centrale de données. Bien que le terme magasin de données soit utilisé comme équivalent de data warehouse par certains auteurs, il convient mieux au concept d'« operational data store ». D'après les renseignements inscrits dans la base de données IBMOT d'IBM, Information Warehouse est une marque de commerce d'International Business Machines Corp. Le terme lui-même est pourtant répertorié comme nom commun dans plus d'un dictionnaire unilingue anglais spécialisé. Cette définition est tirée de celle de Bill Inmon (William H. Inmon ) que l'on nomme le père des entrepôts de données et l'inventeur de ce qu'on appelle EDW pour Entreprise Data warehouse ou CIF pour Corporate Information Factory . [GrdDiction]

Le but d'un DW est de transformer des données avec vacation de production en un ensemble d'informations décisionnelles alimentant la source de données d'un système d'informations décisionnels.

**Les principes**

Un Data Warehouse est un entrepôt de données (une base de données) qui se caractérise par des données :

- Orientées « **métier** » ou *business* (par exemple, pour une entreprise de production, les commandes clients sont traitées différemment entre le service de comptabilité et le service des achats)
- Présentées selon différents **axes** d'analyse ou « dimensions » (par exemple : le temps, les types ou segments de clientèle, les différentes gammes de produits, les différents secteurs régionaux ou commerciaux, etc.)

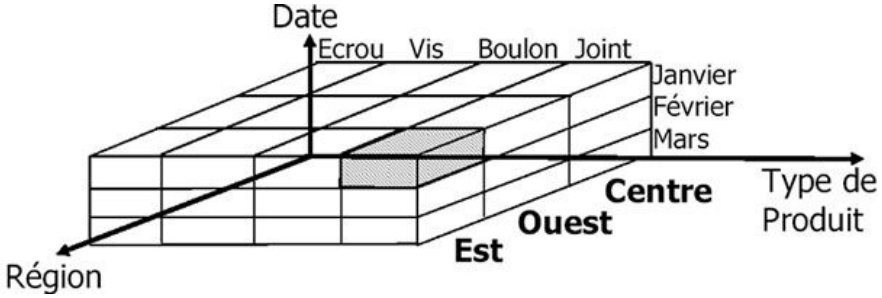



**Figure A.1** *Vue Multidimensionnelle des Data Warehouse*

- **Non volatiles** : stables, en lecture seule, non modifiables,
- **Intégrées** en provenance de sources hétérogènes ou d'origines diverses (y compris des fichiers externes de cotation ou de string)
- **Archivées** et donc *datées* : avec une conservation de l'historique et de son évolution pour permettre les analyses comparatives (par exemple, d'une année sur l'autre, etc.).

## Critères d'évaluation des Data Warehouse

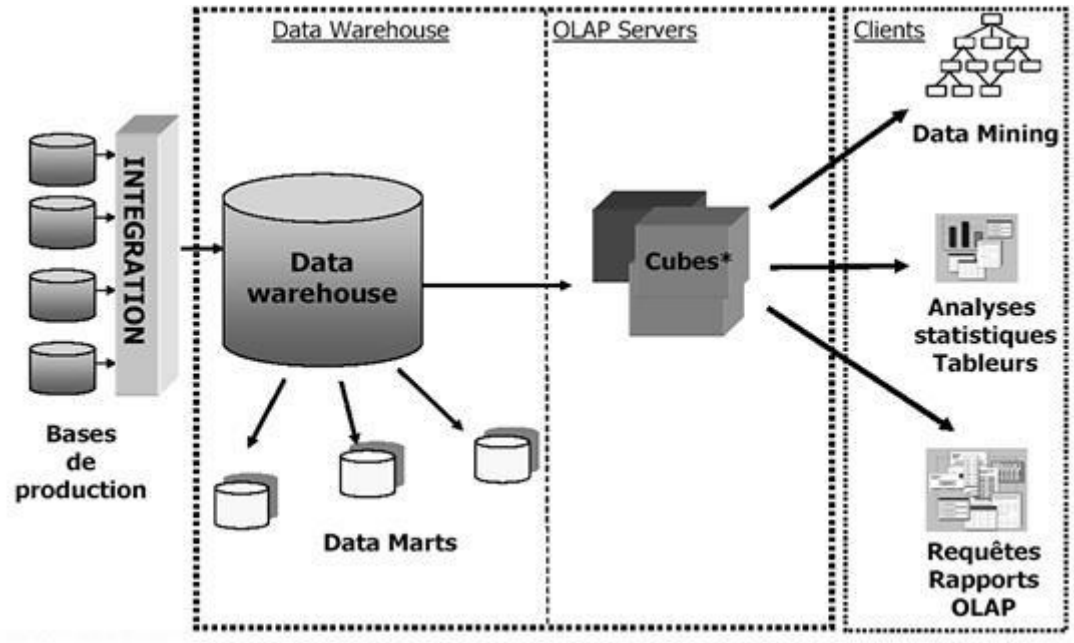
- Les temps de réponse.
- Facilité d'utilisation.
- Visualisation multidimensionnelle des données.

### Architecture d'un Data Warehouse

Un entrepôt de données est généralement construit selon une architecture en 3 niveaux, tel qu'indiqué dans la figure A.3 ci-dessous :

- d'un serveur d'entrepôt (serveur de données)
- d'un serveur OLAP (de type HOLAP/MOLAP ou ROLAP)
- d'un client
  - outil pour l'exécution des requêtes
  - outil pour l'analyse des données

**Figure A.2**
*Architecture d'un Data warehouse*




### *Alimentation*

L'alimentation des données à partir des bases de production est une phase primordiale d'un Data Warehouse. Des outils logiciels sont alors nécessaires pour

intégrer les données dans le data warehouse. On parle d'ELT (Extract, Transform, Load). Les phases de l'alimentation d'un data warehouse sont les suivantes :

**Découverte des données** : Il s'agit d'identifier dans les systèmes sources les données à importer dans le data warehouse. Il faut prendre les données les plus judicieuses. Un mauvais choix peut considérablement compliquer les phases suivantes de l'alimentation.

**Extraction des données** : Il s'agit de collecter les données utiles dans les systèmes de production. Il faut identifier les données ayant été modifiées afin d'importer le minimum de données dans le data warehouse.

**Transformation des données** : Il faut rendre les données cohérentes avec la structure du data warehouse. On applique alors des filtres sur les données. Il peut être nécessaire de convertir le format des données (EBCDIC vers ASCII par exemple) ou d'harmoniser les formats de dates (jj/mm/aaaa). Il faut également associer les champs source avec les champs cibles. Un champ source « adresse » pourra ainsi par exemple être décomposé en « numéro », « rue », « code postal » , « ville » ou l'inverse. Enfin des données des systèmes de production doivent être agrégées ou calculées avant leur chargement.

## OLAP

Les data warehouses eux reposent sur le système OLAP (On Line Analytical Processing). Ce système travail en lecture seulement. Les programmes consultent d'importantes quantités de données pour procéder à des analyses. Les objectifs principaux sont regrouper, organiser des informations provenant de sources diverses, les intégrer et les stocker pour donner à l'utilisateur une vue orientée métier, retrouver et analyser l'information facilement et rapidement. Cela nécessite de consulter des versions historiques de la base et peut se permettre d'ignorer temporairement les dernières mises à jour. Ces bases sont souvent d'un ordre de grandeur nettement supérieur à celle des bases OLTP, du fait de la conservation de l'historique.

**Comparaison :** Voici un tableau récapitulatif des différences entre OLTP et OLAP :

**Tableau A.1**
*Comparaison OLTP/OLAP*

| **Caractéristiques** | **OLTP** | **OLAP** |
|---|---|---|
| Utilisation | SGBD (base de production) | Data Warehouse |
| Opération typique | Mise à jour | Analyse |
| Type d'accès | Lecture écriture | Lecture |
| Niveau d'analyse | Elémentaire | Global |
| Quantité d'information échangées | Faible | Importante |
| Orientation | Ligne | Multidimension |
| Taille BD | Faible (max qq GB) | Importante (pouvant aller à plusieurs TB). |

L'objectif des bases OLTP est de pouvoir répondre rapidement à des réponses simples, exemple : les ventes du produit X.

Les bases OLAP permettent des requêtes plus complexes : les ventes du produit X par vendeur, région et par mois.

## Annexe B : Tutorial RELAX NG [CLARK01]

Cette annexe est une copie intégrale d'une partie d'un très bon tutorial [CLARK01] traitant les schémas XML avec le langage RELAX NG. Nous vous conseillons de le consulter pour découvrir toutes les fonctionnalités de ce puissant langage.

RELAX NG est un langage de schéma pour XML, basé sur RELAX et TREX. Un schéma RELAX NG spécifie des motifs qui décrivent la structure et le contenu d'un document XML. Un tel schéma RELAX NG identifie une classe de documents XML qui correspond aux documents qui se conforment à cet ensemble de motifs. Un schéma RELAX NG est en soit un document XML.

### 1. Premiers pas

Considérons une représentation XML basique d'un répertoire d'adresses électroniques :

```
<répertoire>
  <carte>
    <nom>John Smith</nom>
    <courriel>js@exemple.com</courriel>
  </carte>
  <carte>
    <nom>Fred Bloggs</nom>
    <courriel>fb@exemple.net</courriel>
  </carte>
</répertoire>
```

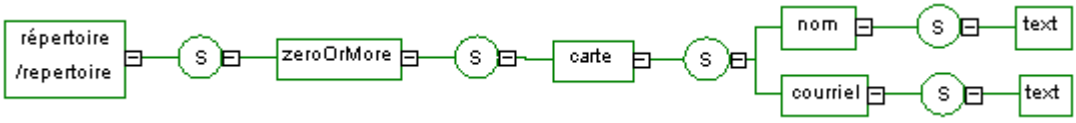
La DTD en serait la suivante :

```
<!DOCTYPE répertoire [
<!ELEMENT répertoire (carte*)>
<!ELEMENT carte (nom, courriel)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT courriel (#PCDATA)>
]>
```

Un motif RELAX NG correspondant pourrait être rédigé ainsi :

```
<element name="répertoire"
xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="carte">
      <element name="nom">
        <text/>
      </element>
      <element name="courriel">
        <text/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

**Figure A.3**
*Représentation graphique d'un RELAX NG*



Si répertoire ne doit pas être vide, alors on utilise oneOrMore (NdT : un ou plus) au lieu de zeroOrMore (NdT : zéro ou plus):

```
<element name="répertoire"
 xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="carte">
      <element name="nom">
        <text/>
      </element>
      <element name="courriel">
        <text/>
      </element>
    </element>
  </oneOrMore>
</element>
```

Changeons maintenant ce motif pour permettre à chaque carte de contenir un élément note optionnel :

```
<element                                        name="répertoire"
xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="carte">
      <element name="nom">
        <text/>
      </element>
      <element name="courriel">
        <text/>
      </element>
      <optional>
      <element name="note">
        <text/>
      </element>
      </optional>
    </element>
  </zeroOrMore>
</element>
```

Notez que le motif `text` correspond à n'importe quel texte, y compris un texte vide. Notez aussi qu'un espace séparant des balises est ignoré quand ces dernières sont comparées à un motif. Tous les éléments définissant un motif doivent appartenir à un espace de noms identifié par l'URI : http://relaxng.org/ns/structure/1.0

L'exemple ci-dessus utilise la déclaration d'espace de noms par défaut `xmlns="http://relaxng.org/ns/structure/1.0"`. Il est également autorisé d'associer un préfixe à un espace de noms :

```
<rng:element name="répertoire"
xmlns:rng="http://relaxng.org/ns/structure/1.0">
  <rng:zeroOrMore>
    <rng:element name="carte">
      <rng:element name="nom">
        <rng:text/>
      </rng:element>
      <rng:element name="courriel">
        <rng:text/>
      </rng:element>
    </rng:element>
  </rng:zeroOrMore>
</rng:element>
```

## 2. Choix

Supposons maintenant que nous voulions permettre, comme alternative à nom, une distinction entre un prénom et un nom-de-famille, selon un répertoire suivant :

```
  <répertoire>
    <carte>
      <prénom>John</prénom>
      <nom-de-famille>Smith</nom-de-famille>
      <courriel>js@exemple.com</courriel>
    </carte>
    <carte>
      <nom>Fred Bloggs</nom>
      <courriel>fb@exemple.net</courriel>
    </carte>
  </répertoire>
```

On peut utiliser le motif suivant :

```
      <element name="courriel">
        <text/>
      </element>
      <optional>
      <element name="note">
        <text/>
      </element>
      </optional>
    </element>
  </zeroOrMore>
</element>
……
```

```
……
       <element name="courriel">
         <text/>
       </element>
       <optional>
       <element name="note">
         <text/>
       </element>
       </optional>
     </element>
   </zeroOrMore>
 </element>
```

Cela correspond à la DTD suivante :

```
<!DOCTYPE répertoire [
<!ELEMENT répertoire (carte*)>
<!ELEMENT carte ((nom | (prénom, nom-de-famille)), courriel,
note?)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT courriel (#PCDATA)>
<!ELEMENT prénom (#PCDATA)>
<!ELEMENT nom-de-famille (#PCDATA)>
<!ELEMENT note (#PCDATA)>
]>
```

### 3.Typage de données

RELAX NG permet aux motifs de faire référence à des types de données conformes à des définitions externes, comme celles définies par [W3C XML Schema Datatypes]. Les implémentations RELAX NG peuvent différer dans les types de données qu'elles supportent. Vous devez utiliser les types de données qui sont supportés par l'implémentation que vous souhaitez utiliser.

Le motif data valide une chaîne de caractères qui représente la valeur d'un type de données désigné. L'attribut datatypeLibrary prend pour valeur l'URI identifiant la librairie de types de données utilisée. La librairie de types de données de [W3C XML Schema Datatypes] doit être identifiée par l'URI http://www.w3.org/2001/XMLSchema-datatypes. L'attribut type spécifie le nom d'un type de données de la libraire identifiée par l'attribut datatypeLibrary. Par exemple, si une implémentation RELAX NG supportait les types de données de [W3C XML Schema Datatypes], vous pourriez utiliser :

```
<element name="nombre">
  <data type="integer"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"/>
</element>
```

Il ne serait pas commode de devoir spécifier systématiquement un attribut datatypeLibrary pour chaque élément data, c'est pourquoi RELAX NG permet à ses éléments d'hériter de la valeur de l'attribut datatypeLibrary. L'attribut datatypeLibrary peut-être spécifié pour n'importe quel élément RELAX NG. Si un élément data n'a pas d'attribut datatypeLibrary, la valeur de l'attribut datatypeLibrary de l'élément parent le plus direct sera utilisée. Typiquement, l'attribut datatypeLibrary dans l'élément racine d'un schéma RELAX NG. Par exemple :

```
<element name="point"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <element name="x">
    <data type="double"/>
  </element>
  <element name="y">
    <data type="double"/>
  </element>
</element>
```

Si le fils d'un élément ou attribut se conforme à un motif data, alors tout le contenu de cet élément ou attribut doit se conformer à ce motif data. Un motif ne peut pas permettre à une partie de son contenu de se conformer à un motif data donné tout en permettant à une autre partie de se conformer à un autre. Par exemple, le motif suivant n'est pas autorisé :

```
<element name="incorrect">
  <data type="int"/>
  <element name="note">
    <text/>
  </element>
</element>
```

Cependant, celui-ci serait correct :

```
<element name="correct">
  <data type="int"/>
  <attribute name="note">
    <text/>
  </attribute>
</element>
```

Notez que ces restrictions ne s'appliquent pas au motif text.

Les types de données peuvent avoir des paramètres. Par exemple, une chaîne de caractères d'un type de données peut avoir un paramètre contrôlant sa longueur. Les paramètres applicables à des types de données particuliers sont déterminés par le vocabulaire de typage des données. Les paramètres sont spécifiés en insérant un ou plusieurs éléments param comme éléments fils d'un élément data. L'exemple suivant limite la chaîne de caractères contenue dans un élément courriel à une longueur maximum de 127 caractères :

```
<element name="courriel">
  <data type="string">
    <param name="maxLength">127</param>
  </data>
</element>
```

**Comparaison avec les DTD XML**

RELAX NG propose des fonctionnalités qui dépassent le cadre des DTD pour XML. En particulier, RELAX NG :

- utilise une syntaxe XML pour représenter des schémas ;
- supporte le typage des données ;
- intègre les attributs dans les modèles de contenu ;
- supporte les espaces de noms XML ;
- supporte les contenus non ordonnés ;
- supporte la dépendance au contexte des modèles de contenu.

**Syntaxe du schéma XML** [MSDN]

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://tempuri.org/XMLSchema1.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="addressType">
   <xs:sequence>
      <xs:element name="street1" type="xs:string"/>
      <xs:element name="street2" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="zip" type="xs:integer"/>
   </xs:sequence>
</xs:complexType>

<xs:element name="purchaseOrder">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="shipTo" type="addressType" />
            <xs:element name="billTo" type="addressType" />
            <xs:element name="shipDate" type="xs:date" />
            <xs:element name="item" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Exemple d'un schéma XML

L'exemple de schéma XML ci-dessus contient les éléments suivants :

- La première instruction, `<?xml version="1.0" encoding="utf-8"?>`, spécifie quelle version de XML est utilisée.
- La seconde instruction se compose de plusieurs parties :
  - La déclaration `xs:schema` indique qu'il s'agit d'un schéma et que le préfixe xs: sera placé devant ses éléments.
  - La déclaration `xmlns:xs="http://www.w3.org/2001/XMLSchema"` indique que toutes les balises de ce schéma doivent être interprétées en fonction de l'espace de noms par défaut du consortium W3C (World Wide Web Consortium), fondé en 2001.
  - La déclaration `targetnamespace` attribue au schéma le nom XMLSchema1.xsd et indique sa position par défaut, laquelle se trouvera dans un URI (Universal Resource Identifier) de votre serveur de développement appelé tempuri.org.
- Un type complexe sans nom appelé « addressType » est défini ; il contient cinq éléments de types de données divers. Notez que chacun de ces éléments est un type nommé simple.
- Un élément complexe appelé « purchaseOrder » est défini. Il contient des éléments utilisant le type addressType précédemment défini. Une relation imbriquée est ainsi créée entre l'élément purchaseOrder et ces éléments.
- L'élément final de purchaseOrder, Item, contient une facette appelée **maxOccurs**. Les facettes servent à limiter le type de contenu qu'un type simple peut contenir. **maxOccurs** est utilisé pour limiter le nombre d'occurrences d'un même élément dans les documents créés à l'aide de ce schéma. Par défaut, il est égal à un. Dans ce cas, la valeur **unbounded** définie pour **maxOccurs** indique qu'il peut exister autant d'occurrences de l'élément Item que nécessaire. Annexe.

## Annexe C : Glossaire de vocabulaire

**A/COO** : Analyse et Conception Orientées Objet.

**CWM** :Common Warehouse Metamodel, Une interface servant à faciliter les échanges de métadonnées entre outils, plates-formes et bibliothèques de métadonnées dans un environnement hétérogène.

**CODEC** : Coding decoding, Codage-decodage, algorithmes de compression et décompression

**Cookie** : Petit fichier envoyé du serveur vers le client et utilisé comme marqueur pour suivre l'utilisateur et enregistrer des paramètres à son sujet

**CSS / feuille de style :** Cascading Style Sheet, est un concept qui a été développé pour HTML et contient les propriétés attachées à un élément (les styles). Les styles définissent comment visualiser une page HTML (police de caractère, taille de police, etc.)

**Consortium Unicode** : est une organisation sans but lucratif, ayant pour mission de développer, d'étendre et de promouvoir Unicode. Les membres du consortium sont issus d'un large éventail d'entreprises et d'organisations dans le domaine de l'informatique et des technologies de l'information. Le consortium est financé uniquement par les cotisations de ses membres. L'adhésion au consortium Unicode est ouverte à toutes les organisations et individus qui soutiennent le standard Unicode et souhaitent favoriser son extension et son utilisation. [Unicode]

**DTD** : Document Type Définition, Définition de Type de Document

**Data Mining** : Dans le cadre de leurs activités, les organisations (privées ou publiques) constituent de nombreuses bases de données. Le data mining consiste à extraire et analyser certaines de ces données, notamment dans le cadre du marketing. Par exemple, lors d'un achat sur un magasin en ligne, on recommande un produit au consommateur en fonction de l'analyse de ses habitudes d'achat

**Data warehouse :** entrepôt de données, est une sorte de base de données de bases de données! Structurée de manière spécifique, elle permet l'analyse de toutes les données stockées par une organisation et offre ainsi aux responsables des outils stratégiques d'aides à la décision

**DNS** : Domain Name Server, nom de domaine

**EDI :** Environnement de Développement Intégré : est un programme regroupant un éditeur de texte, un compilateur, des outils automatiques de fabrication, et souvent un débogueur. Bien que des EDI pour plusieurs langages existent, bien souvent (surtout dans les produits commerciaux) un EDI est dédié à un seul langage de programmation. On peut également trouver dans un EDI un système de gestion de versions et différents outils pour faciliter la création de l'interface graphique

**ERP :** *Enterprise Resource Planning,* selon le grand dictionnaire terminologique [GDT], un « Progiciel qui permet de gérer l'ensemble des processus d'une entreprise, en intégrant l'ensemble des fonctions de cette dernière comme la gestion des ressources humaines, la gestion comptable et financière, l'aide à la

décision, mais aussi la vente, la distribution, l'approvisionnement et le commerce électronique.»

**ETL** : Extract-Transform-Load / *datapumping*, Il s'agit d'une technologie informatique middleware permettant d'effectuer des synchronisations massives d'information d'une banque de données vers une autre. Selon le contexte, on traduira par « alimentation », « extraction », « transformation », « constitution » ou « conversion », souvent combinés. Elle est basée sur des *connecteurs* servant à exporter ou importer les données dans les applications (Ex : connecteur Oracle ou SAP...), des *transformateurs* qui manipulent les données (agrégations, filtres, conversions...), et des *mises en correspondance* (*mappages*). Le but est l'intégration de l'entreprise par ses données.

**GDT :** est un dictionnaire terminologique de l'Office québécois de la langue française contenant plus de 3 millions de termes français, anglais et latin dans 200 domaines d'activité. http://www.granddictionnaire.com.

**ISO** : International Organization for Standardization,Organisation internationale de normalisation.

**IIS** : Internet Information Services, est un ensemble de services TCP/IP dédiés à l'Internet, Il comprend les serveurs suivants : Web, FTP, Gopher, SMTP, NNTP et les outils d'administration associés.

**JSF** :JavaServer Faces. JSF est un framework Java, ou en français cadre de travail, pour les développeurs d'interface Web, utilisant l'architecture J2EE. JSF a été mis au point par *Craig McClanahan*, le père de Struts et de Catalina le conteneur de servlet de Tomcat. La première version a vu le jour en 2003. Le but de JSF est de fournir au développeur d'applications de type Web, une interface de programmation lui permettant de manipuler l'interface Web dans un environnement J2EE sans avoir à recourir à du code HTML ou JavaScript. JSF est implémenté sous l'EDI SUN Java Studio Creator.

**LAN** : Local Area Network, Réseau local d'ordinateurs qui partagent des ressources (serveur, imprimantes, scanners, ...). Typiquement le bâtiment d'une entreprise. Le serveur héberge des applications et partage l'espace de stockage de données pour les utilisateurs.

**LDAP** : Lightweight Directory Access Protocol, Protocole d'accès normalisé aux services d'un annuaire (base de données avec une structure spécifique arborescente). LDAP est aujourd'hui très largement utilisé par les applications visant à offrir un accès à un ou des annuaire(s), par exemple dans un réseau local ou sur Internet.

**NTIC / TIC :** Nouvelles Technologies de l'Information et de la Communication

**Middleware** *:* En informatique, désigne les logiciels servant d'intermédiaire entre d'autres logiciels. On utilise généralement du *middleware* comme intermédiaire de communication entre des applications complexes, distribuées sur un réseau informatique.

**Metamodeling** : méta-modélisation, Modélisation Récursive des Eléments de modélisation à partir d'eux-mêmes.

**MVC :** *Modèle Vue Contrôleur :* est un design pattern (motif de conception) pour le développement d'applications logicielles qui sépare le modèle de données, l'interface utilisateur et la logique de contrôle. Ce motif a été mis au point en 1979 par Trygve Reenskaug, qui travaillait alors sur Smalltalk.

Les éléments permettant la réalisation classique d'un système en architecture trois tiers sont les suivants:

- système de base de donnée relationnel (SGBDR) pour le stockage des données.
- serveur applicatif pour la logique applicative.
- navigateur web pour la présentation

**MOF** : Meta Object Facility, http://www.omg.org/, est un standard de l'OMG adressant la représentation des métamodèles et leur manipulation. Le langage MOF s'auto-définit.

**OMT** : Object Modeling Technique, Technique de modélisation Objet

**OMG :** Object Management Group, est un groupe d'industriels créé en 1989 dont l'objectif est de standardiser et promouvoir le modèle objet sous toutes ses formes. l'OMG comprend actuellement plus de 800 membres, dont les principaux acteurs de l'industrie informatique ( SUN, IBM, Microsoft, … ) [Roques02]

**OCL** : Object Constraint Language, langage développé à l'origine pour exprimer des contraintes sur des éléments de diagrammes UML, il offre un ensemble de fonctions efficaces pour naviguer entre les éléments d'un modèle UML ou MOF et récupérer les informations liées à ces éléments.

**ODBC** : Open DataBase Connectivity, Interface d'accès aux bases de données développée par Microsoft pour en savoir plus: www.microsoft.com/data/

**RDF :** Ressource Description Framework

**RELAX** : REgular LAnguage description for XML.

**RELAX NG :** est un langage de schéma pour XML, basé sur [RELAX] et TREX]. Un schéma RELAX NG spécifie des motifs qui décrivent la structure et le contenu d'un document XML. Un tel schéma RELAX NG identifie une classe de documents XML qui correspond aux documents qui se conforment à cet ensemble de motifs. Un schéma RELAX NG est en soit un document XML.

**RUP** : Rational Unified Process, est un processus de développement logiciel élaboré et commercialisé par la société Rational Software géré actuellement par IBM. Il se présente sous forme d'un guide méthodologique au format HTML, couplé à une base de connaissances et capable de s'interfacer avec divers outils d'expression des besoins, de modélisation UML, d'automatisation des tests, de gestion de configuration, de production de documents, de gestion de projet, etc.

**SAX :** Simple API for XML. Interface standard pour les parseurs XML. SAX est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML.

**SMTP** : Simple Mail Transfer Protocol, Protocole utilisé pour l'envoi de messages électroniques sur Internet

**SGML** : Standard Generalized Markup Language, Langage normalisé ( ancêtre du langage XML ) permettant de décrire les relations entre le contenu d'un document informatique et sa structure

**Shareware / Partagiciel :** Logiciel sans copyright. On en trouve des milliers sur le Net . On peut le télécharger gratuitement et l'utiliser un certain temps pour l'évaluer. Si l'on est satisfait, il suffit de contacter l'auteur pour payer et recevoir une version complète et sans limite d'utilisation dans le temps

**SGBD** : Système de gestion de bases de données.

**SGBDR** : Système de gestion de bases de données relationnelles (exemple: Oracle, Informix, Sybase, SQLServer, etc)

**TREX** : Tree Regular Expressions for XML

**TCP/IP** : Transmission Control Protocol/Internet Protocol

**URI** : Universal Resource Identifier

**URL** : Unified Resource Locator

**UWE** : UML-based Web Engineering, approche de conception orienté objet proposée par N.Koch [Koch00]. Elle peut être classée comme une extension de la notation UML pour le développement WEB.

**Web services :** Services Web, Développés sur base d'XML et des standards UDDI, WSDL et SOAP, les services Web permettent la mise à disposition d'applications prêtes à l'emploi via le Web, sans contrainte de plate-forme de développement. Ils permettent la communication entre applications et entreprises sur le Web

**W3C** : World Wide Web Consortium, Organisation internationale créée en 1994 et dont l'objectif est de permettre au Web d'atteindre son meilleur potentiel par le développement de protocoles communs et universels. www.w3c.org

**XSL** : eXtensible Style Language, Langage normalisé par W3C définissant l'utilisation de feuilles de styles pour transformer des documents XML en documents XML, HTML, XHTML, texte, etc

**XSL-FO :** XSL Formatting Objects

**XML** : eXtensible Markup Language
Langage de description et d'échange des documents structurés défini par le Consortium W3C en 1998. XML est en passe de devenir le format standard de stockage et d'échange de données, notamment pour les applications d'e-business pour en savoir plus: www.xml.org.

**XMI** : XML Metadata Interchange, http://www.omg.org/.

**Annexe E Liste des Logiciels utilisés :**

Les logiciels mentionnés dans cet annexe ont été utilisés dans le cadre des travaux pratiques liées au thème de ce mémoire et d'autres à figurent titre d'information.

**1**

| | |
|---|---|
| **NOM DU LOGICIEL** | **DBDesigner 4** |
| **Entreprise** | |
| **Licence** | Libres GNU GPL |
| **Version** | 4.0.5.6 BETA |
| **Description** | • Modéliser les base de données<br>• Synchroniser avec BD [testé avec MySql, avec oracle il faut passer un ODBC ]<br>• Reverse ingenering : Généré le modèle physique de la base de données, et possibilité d'exportation des scripts SQL |
| **Remarque** | Interface agréable et facile à utiliser. |
| **Source WEB** | http://www.fabforce.net/ |

**2**

| | |
|---|---|
| **NOM DU LOGICIEL** | **TOAD** |
| **Entreprise** | QUEST SOFTWARE |
| **Licence** | Payant |
| **Version** | 8.6.1 |
| **Description** | ▪ Gérer des base de données Oracle<br>▪ Requêteur<br>▪ Administration de base de données<br>▪ Gestion des tables, des vues, des Triggers, des procédures stockées …<br>▪ Export/import des données.<br>▪ Import/export<br>▪ Et bien d'autres fonctionnalités très pratique |
| **Remarque** | ▪ Compatible seulement avec Oracle<br>▪ Ergonomiquement bien conçu<br>▪ Multi connexion<br>▪ Rapide<br>▪ Très pratique pour le teste des de requêtes (à l'image de SQL+)<br>▪ Je le conseil vivement. |
| **Source WEB** | www.quest.com |

**3**

| | |
|---|---|
| **NOM DU LOGICIEL** | **PowerAMC** |
| **Entreprise** | Sybase |
| **Licence** | Payant [version d'essai 30 jours] |
| **Version** | 10.0 |
| **Description** | ▪ Modélisation UML<br>▪ Modéliser les base de données<br>▪ Génération du code natif pour application JAVA<br>▪ Génération des données de teste<br>▪ Reverse ingenering. |
| **Remarque** | ▪ Ergonomiquement bien conçu<br>▪ Un produit complet et stable |

| | |
|---|---|
| | ▪ Multi connexion<br>▪ Coûte très cher |
| **Source WEB** | www.sysbase.com |

**4**

| | |
|---|---|
| **NOM DU LOGICIEL** | **Poseidon for UML** |
| **Entreprise** | Gentleware |
| **Licence** | Version Standard gratuite<br>Version Professionnel payante |
| **Version** | 1.3 |
| **Description** | ▪ Modélisation UML<br>▪ Génération de la documentation HTML des classes et des diagrammes UML<br>▪ Génération du code natif JAVA |
| **Remarque** | ▪ Interface basique.<br>▪ La version payante semble intéressante. |
| **Source WEB** | http://www.gentleware.com |

**5**

| | |
|---|---|
| **NOM DU LOGICIEL** | **Dreamweaver MX 2004** |
| **Entreprise** | Macromedia |
| **Licence** | Payant (Version d'essai gratuite) |
| **Version** | MX 2004 |
| **Description** | ▪ Création de site WEB<br>▪ Usage de modèle prédéfinie<br>▪ Fonctionnalités avancées pour la création de page HTML<br>▪ Débogueur PHP<br>▪ Connexion à une base de données<br>▪ … |
| **Remarque** | ▪ Un logiciel Complet et stable<br>▪ Ergonomiquement bien conçu<br>▪ La même ergonomie des logiciels STUDIO MX de Macromedia |
| **Source WEB** | www.macromedia.com |

**6**

| | |
|---|---|
| **NOM DU LOGICIEL** | **Flash MX 2004** |
| **Entreprise** | Macromedia |
| **Licence** | Payant (Version d'essai gratuite) |
| **Version** | MX 2004 |
| **Description** | ▪ Création d'animation<br>▪ Ouvert sur plusieurs langages de technologie Internet (JAVA, HTML, PHP, JSP, JavaScript, XML… )<br>▪ Langage objet propriétaire (ActionScript ) pour la création et le contrôle d'animations et du déroulement de scénarios.<br>▪ Site Web interactifs |
| **Remarque** | ▪ Intègre des fonctionnalités avancées pour la manipulation des objets XML<br>▪ Animation à base de dessins vectoriels |
| **Source WEB** | www.macromedia.com |

**7**

| | |
|---|---|
| **NOM DU LOGICIEL** | **Fireworks MX 2004** |

| | |
|---|---|
| **Entreprise** | Macromedia |
| **Licence** | Payant (Version d'essai gratuite) |
| **Version** | MX 2004 |
| **Description** | ▪ Traitement d'images<br>▪ Dessin vectoriel<br>▪ Fusion des images<br>▪ Exportation des images vers plusieurs formats<br>▪ Création des images Interactives avec possibilité de génération du code HTML intégrant les évènement d'interaction en JavaScript |
| **Remarque** | ▪ plein de fonctionnalités d'effets de traitement d'images<br>▪ Pratique pour la génération des squelettes des pages HTML |
| **Source WEB** | www.macromedia.com |

**8**

| | |
|---|---|
| **NOM DU LOGICIEL** | **PHPEDit** |
| **Entreprise** | Waterproof Software |
| **Licence** | Payant de puis la version 1.0.3 |
| **Version** | 2.8 |
| **Description** | ▪ Editeur de script PHP<br>▪ Génération de la documentation depuis les commentaire<br>▪ Débogage<br>▪ Des centaines de raccourcis<br>▪ Possibilité de création des modèles de templates avec des raccourcis pour la génération des squelettes de code (exemple Class avec constructeur, fonction, variable)<br>▪ Facilite la programmation orientée objet |
| **Remarque** | ▪ Interface agréable et facile à utilisée<br>▪ indispensable pour un développeur PHP<br>▪ je le conseil vivement |
| **Source WEB** | http://www.waterproof.fr/ |

**9**

| | |
|---|---|
| **NOM DU LOGICIEL** | **CodeCharge Studio** |
| **Entreprise** | Yessoftware |
| **Licence** | Payant ( version d'essai gratuite ) |
| **Version** | 3.1 |
| **Description** | ▪ Environnement de programmation graphique de multiples langage de programmation WEB<br>▪ Génération automatique de modèle de sites web à partir de base de donnée<br>▪ Supporte plusieurs langages de programmation web ( PHP4/PHP5, ASP, ASP.Net, CFML,JSP, PERL)<br>▪ Programmation événementielle |
| **Remarque** | ▪ Permet de réaliser des sites web rapidement<br>▪ je le conseil vivement<br>▪ L'originalité de ce logiciel c'est la **Programmation événementielle** dans du développement web. |

| | |
|---|---|
| **Source WEB** | www.yessoftware.com |

**10**

| | |
|---|---|
| **NOM DU LOGICIEL** | **MySQL to Oracle** |
| **Entreprise** | Intelligent Converters software |
| **Licence** | Payant (Version d'essai gratuite) |
| **Version** | 2.3 |
| **Description** | ▪ Migration des bases de données MySQL vers Oracle (Schéma + données) |
| **Remarque** | ▪ Exige un serveur MySQL installé et un serveur Oracle<br>▪ Pratique pour la migration d'une base MySQL vers une base Oracle<br>▪ L'entreprise présente un grand nombre de logiciels de conversion pour d'autres SGBD |
| **Source WEB** | http://www.convert-in.com/ |

**11**

| | |
|---|---|
| **NOM DU LOGICIEL** | **ModelMaker 6.2** |
| **Entreprise** | |
| **Licence** | Payant, version d'essai fournie avec le cd Delphi |
| **Version** | |
| **Description** | ▪ Modélisation graphique des Objets pour DELPHI<br>▪ Synchronisé le code avec l'environnement de développement DELPHI |
| **Remarque** | ▪ Nécessite des connaissances avancées sur la programmation Objet sous Delphi<br>▪ Très pratique pour les projets de taille, et surtout dans la phase de maintenance. |
| **Source WEB** | http://www.modelmakertools.com |

**12**

| | |
|---|---|
| **NOM DU LOGICIEL** | **ECLIPSE** |
| **Entreprise** | Eclipse |
| **Licence** | Open source, gratuit |
| **Version** | 3.2 |
| **Description** | ▪ Un puissant outil pour réaliser des applications orientées sous Java |
| **Remarque** | ▪ Intègre depuis la version 2.0 le langage de programmation PHP |
| **Source WEB** | www.eclipse.org |

**13**

| | |
|---|---|
| **NOM DU LOGICIEL** | **Java Studio Creator** |
| **Entreprise** | SUN |
| **Licence** | Gratuit |
| **Version** | 2004 |
| **Description** | ▪ Programmation java .Net<br>▪ Technologie JSF (Java Server Face ) |
| **Remarque** | ▪ Permet de faire de la programmation Web visuel |
| **Source WEB** | www.sun.com |

# Références bibliographies

## Ouvrages


| | | |
|---|---|---|
| [Conallen00] | Concevoir des applications WEB avec UML. J.Conallen. Eyrolles | 2000 |
| [Larman02] | UML et les Design Patterns. C.Larman CampusPress | 2002 |
| [Lopez98] | Intégrer UML dans vos projets. N.Lopez, J.Migueis, E.Pichon Eyrolles | 1998 |
| [Cokburn01] | Rédiger des cas d'utilisation efficaces. A.Cockburn Eyrolles | 2001 |
| [Roques02] | UML Modéliser un site e-commerce. P.Roques. Eyrolles. | 2002 |
| [Jacobson02] | Le guide de l'utilisateur UML. G.Booch, J.Rumbaugh, I.Jacobson. Eyrolles. | 2002 |
| [MullerGaertner00] | Modélisation objet avec UML. P-A.Muller, N.Gaertner. Eyrolles. | 2000 |
| [Gamma 95] | Design Patterns : Elements of reusable Object-Oriented Software. E.Gamme, et R. Helm, Ralph Johnson, J.Vlissides. Addison-Wesley | 1995 |
| [UML-UG 98] | The Unified Modeling Language Reference Manual. 1999. G.Booch, J.Rumbaugh, I.Jacobson. Addison-Wesley | 1998 |
| [Bushmann96]. | Pattern-Oriented Software Architecture : A System of Patterns. F.Bushmann R.Meunier, H.Rohnert, P.Sommerlad, M.Stal. Wiley. | 1996 |
| [Bordage06] | Conduite de projet Web. 2006 S.Bordage. Eyrolles | 2006 |
| [Booch96] | Object Solutions - Managing the Object-Oriented Project. G.Booch. Pearson Education | 1996 |
| [Kruchten00] | Introduction au Rational Unified Process, Ph.Krachten . Eyrolles | 2000 |
| [Monnier02] | Flash MX : animation, interactivité, ActionScript. G.Monnier. Dunod | 2002 |
| [Booch99] | The Unified Software Development Process. G.Booch, J.Rumbaugh, I.Jacobson. Addison-Wesley | 1999 |
| [Rosenberg99] | Use Case Driven Object Modeling with UML. D.Rosenberg, K.Scott. Wesley Longman | 1999 |
| [Penner03] | Creative Studio programmation Falsh MX. R.Penner. First interactive | 2003 |
| [Sehan02] | Flash MX et ActionScript. J-F.Sehan, B.Underdahl. First interactive, | 2002 |
| [Goldfarb03] | The XML Handbook, 5the edition, C-F.Goldfarb, P.Prescod. Prentice Hall PTR | 2003 |
| [Gardarin02] | XML : Des bases de données aux services WEB. G. Gardarin. Dunod | 2002 |

[Harold2000] XML Le guide de l'utilisateur.Elliotte Rusty Harold (Adap fr G.Picarde). 2000 Eyrolles

[DUCHARME01] Xslt Quickly: A Tutorial and Concise User's Guide. B.Ducharme. 2001 Manning Publications

[Franco00] Piloter l'entreprise grâce au data Warehouse. J-M.Franco .Eyrolles,

[Nogier01] De l'ergonomie du logiciel au design des sites web : J.F. Nogier. 2001 Dunod

[Nogier05] Ergonomie du logiciel et design Web : Le manuel des interfaces 2005 utilisateur J.F. Nogier. Dunod.

[Kimball00] Concevoir et déployer un data warehouse Guide de conduite de 2000 projet : R.Kimball, L.Reeves. Eyrolles.

[WH.Inmon96] Building the Data Warehouse : W.H.Inmon. John Wiley & Sons 1996

[Kimbel03] Entrepôts de données. Guide pratique de modélisation 2003 dimensionnelle. R.Kimball, M.Ross. Vuibert.

## Articles

[Koch01] The Authoring Process of the UML-based Web Engineering Approach. International : 2001, Institute of Computer Science, Ludwig-Maximilians University of Munich, http://www.pst.informatik.uni-muenchen.de/publications/uwe_iwwost01.pdf

[Koch00]. Software Engineering for Adaptive Hypermedia Applications : PhD. Thesis.

[Bastien93] Ergonomic Criteria for the Evaluation of Human-Computer interfaces 1993 : Ch.Scapin, L.Bastien. INRIA

[Poo01] Model-Driven Architecture, Vision, Standards And Emerging 2001 Technologies : John D. Poole

[Tatibouet06] Etude et comparaison d'outils de transformation de modèles : 2006 B.Tatibouët. LIFC

[Burbeck92] How to use Model-View-Controller. Steve Burbeck. 1992

Comment choisir les contenus de votre site web ? La méthode "Actions-Déclencheurs" de veblog.
http://www.veblog.com/fr/2001/0513-determiner-contenus.html

## Internet

[W3C] http://www.w3.org

[Bourret03] http://www.campusxml.org : XML and Databases, R.Bourret. 2003 Traduction française P.Peccatte.

[CLARK01] http://xmlfr.org/ : RELAX NG Tutorial. J.CLARK, M.Makoto. Oasis. 2001 Traduction française A.Arcouteil.

[Laltruiste] http://www.laltruiste.com/ : Ressources de plusieurs langages de programmation

[MSDN] http://msdn.microsoft.com/ : Divers Ressource informatique

[Microsoft] http://www.microsoft.com/ : Site officiel de l'entreprise Microsoft

[Unicode] http://www.unicode.org/ : Le site Web du consortium Unicode, où vous trouverez les tableaux de caractères et beaucoup d'informations techniques sur la norme

[SpecXML] http://www.w3.org/TR/2004/REC-xml11-20040204/ ou http://www.w3.org/TR/REC-xml.
La recommandation XML 1.1 du w3c

[SpecXMLT] http://www.w3.org/TR/2001/REC-xsl-20011015/
La recommandation XML 1.0 du w3c

[SUN] http://fr.sun.com Site officiel de l'entreprise SUN Microsystems

[Adobe] http://www.adobe.com site officiel de Adobe
Macromedia fait désormais partie de Adobe puisqu'elle a été rachetée en avril 2005

[wikipedia] http://fr.wikipedia.org : encyclopédie librement distribuable

[Reenskaug] http://heim.ifi.uio.no/~trygver/ Page web personnelle de Trygve M. H. Reenskaug inventeur du modèle MVC

[GrdDiction] http://www.granddictionnaire.com/ e grand dictionnaire une excellente référence qui nous donne accès à près de 3 millions de termes français et anglais du vocabulaire industriel, scientifique et commercial, dans 200 domaines d'activité

[XMLFR] http://xmlfr.org/ Ressource XML en français

[UML] http://www.uml.org, http://uml.free.fr

[OMG] http://www.omg.org/

http://www.cyberzoide.net/java/javastyle/ Guide francophone des conventions de codage pour la programmation en langage Java

http://directory.google.com/ Répertoire de liens vers des references informatqiues classées par thème

http://www.valtech.fr/fr/index/hot/agile_methodology.html :
La méthode Agile pour le développement informatique