



HAL
open science

Proposition d'une méthode et d'un outil pour le développement d'applications

Reda Kadri

► **To cite this version:**

Reda Kadri. Proposition d'une méthode et d'un outil pour le développement d'applications. Génie logiciel [cs.SE]. Université de Bretagne Sud, 2009. Français. NNT : . tel-00511828

HAL Id: tel-00511828

<https://theses.hal.science/tel-00511828>

Submitted on 26 Aug 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre: 000

THÈSE

à soutenir

devant l'Université de Bretagne Sud

pour obtenir le grade de :
DOCTEUR DE L'UNIVERSITÉ DE BRETAGNE SUD

Mention :
SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

par

Réda KADRI

Équipe d'accueil : Laboratoire VALORIA

Titre de la thèse :

*Une approche pour la Modélisation d'Applications Web à base de
Composants Logiciels*

Soutenance prévue pour le 12 Janvier 2009 devant la commission d'examen composée de :

Pr	Hafedh	MILI	LATECE, Université du Qubec Montréal	Président
Pr	Hafedh	MILI	LATECE, Université du Québec à Montréal	Rapporteur
Pr	Yamine	AIT-AMEUR	LISI, Ecole Nationale Supérieure d'Aérotechnique	Rapporteur
Pr	Thérèse	Rougé	LIBOUREL LIRMM, Université MontpellierII	Examinatrice
Mfc-Hdr	Salah	SADOU	VALORIA, Université de Bretagne Sud	Directeur
Mfc	Régis	FLEURQUIN	VALORIA, Université de Bretagne Sud	Encadrant
Mfc	François	MERCIOL	VALORIA, Université de Bretagne Sud	Encadrant

Table des matières

I	Introduction	9
1	Problématique du développement des applications Web	11
1.1	État actuel du marché des application Web	11
1.2	Les pratiques de développement actuelles et leur inadéquation au marché	12
1.3	Les promesses de l'approche à base de composants	13
1.4	Contexte de la thèse	14
1.5	Travail présenté dans cette thèse	14
1.6	Organisation du mémoire	15
II	État de l'art	17
2	Historique et évolution des applications Web	19
2.1	Histoire du développement Web	19
2.2	Application Web de cinquième génération	22
2.3	Classification des applications Web	23
2.3.1	Application Web 1.0	23
2.3.2	Application Web 2.0	23
2.3.3	Application Web mobile	24
2.3.4	Application Web Sémantique	24
2.3.5	Application Web riche	24
2.4	Caractéristiques et Complexité des applications Web	26
3	Ingénierie des applications Web	29
3.1	Un diagnostic sur l'ingénierie du développement Web	29
3.2	Évolution des méthodes de modélisation des applications Web	30
3.3	Ingénierie Web basée sur UML	33
3.3.1	Phases de modélisations avec UWE	34
3.3.2	Méta modèle UWE	36
3.4	Modélisation des applications Web avec WebML	36
3.4.1	Historique de WebML	37
3.4.2	WebML	37
3.4.3	Modélisation WebML avec UML	39
3.5	En résumé	40

III	Contribution de la thèse	43
4	Un protocole de migration vers un processus de développement à base de composants	45
4.1	Un protocole de migration	46
4.2	Le processus de développement	47
4.3	Illustration du PCF et du CCF	49
4.4	Expérimentation de l'approche	53
4.4.1	Coûts de développement	54
4.4.2	Discussion	55
4.5	Conclusion	57
5	AlkoWeb, une plate-forme pour le développement Web riche à base de composants	59
5.1	À La recherche d'un modèle de composants logiciels	59
5.2	Le modèle de composant UML2.0	61
5.3	Les éléments architecturaux du modèle AlkoWeb	63
5.4	Exemple Illustratif	64
5.5	Contraintes de modélisation et de déploiement	67
5.6	Déploiement de l'application	68
5.7	AlKoWeb-Builder	68
5.7.1	Fonctionnement	70

Table des figures

2.1	L'ingénierie du Web	20
2.2	Les cinq générations de l'ingénierie du développement Web	21
2.3	Aperçu du méta modèle UWE	26
2.4	Complexité des applications Web	27
3.1	UWE [61]	33
3.2	Aperçu du méta modèle UWE	37
3.3	Processus de développement avec WebML	38
4.1	La plate-forme de développement	47
4.2	Architecture de développement	48
4.3	Un exemple d'une application Alkante	49
4.4	Un exemple d'une sous application Alkante	50
4.5	Exemple d'un PCF	51
4.6	Application de déploiement des composants d'Alkante	52
4.7	Liste de quelques composants Alkante	53
4.8	Programme de livraison des sous applications	54
4.9	Livraison de sous applications après la transition	56
5.1	Méta modèle UML2	62
5.2	Un exemple de composant graphique de type Texte	64
5.3	Un exemple de composant de gestion d'annuaire LDAP	65
5.4	Un exemple illustratif de l'utilisation d'AlkoWeb	66
5.5	Capture écran d'AlKoWeb-Builder	68
5.6	Un exemple de la vue de l'interface d'une application	70
5.7	Un exemple de la vue de l'interface d'une application	70
5.8	Un exemple de la vue de l'interface d'une application	71

Liste des tableaux

4.1	Coûts des différentes Versions	54
4.2	Coûts réels liés aux développement des sous applications	55
4.3	Coûts d'assemblages.	55
4.4	Les coûts de compositions des sous applications.	56
4.5	Les coûts d'assemblages	56

Première partie

Introduction

Chapitre 1

Problématique du développement des applications Web

Sommaire

1.1	État actuel du marché des application Web	11
1.2	Les pratiques de développement actuelles et leur inadéquation au marché	12
1.3	Les promesses de l'approche à base de composants	13
1.4	Contexte de la thèse	14
1.5	Travail présenté dans cette thèse	14
1.6	Organisation du mémoire	15

1.1 État actuel du marché des application Web

La diffusion de l'information par l'intermédiaire d'applications Web en modes Intranet, Extranet et Internet se généralise actuellement. Les avantages de ces applications ne sont plus aujourd'hui à démontrer et l'adoption massive par les entreprises ou par les administrations de ces modes de diffusion de l'information est un fait indéniable. Nous remarquons, par exemple, la forte émergence d'applications et de systèmes collaboratifs de gestion de contenus en ligne, de syndication, d'importation et d'exportation de données. En plus de développer ces nouvelles applications métiers, les entreprises spécialisées dans le développement d'applications Web doivent maintenant faire face aux demandes de transformations ou de pseudo migration des applications de type Desktops vers le Web. Ainsi, des milliers d'applications de différents domaines sont appelées à être déployées sur le Web. Dans un futur proche, nous n'aurons plus besoin d'installer de logiciels sur nos ordinateurs ni même d'y stocker des fichiers. En effet, d'ores-et-déjà, tous ces services peuvent être rendus par le Web 2.0. Chaque application ou presque a son pendant en ligne, qu'il s'agisse de retoucher une image, de faire du montage vidéo, de convertir des fichiers, de créer des documents, etc.

Les applications Web sont actuellement en train de remplacer les applications traditionnelles de type Desktop. L'intérêt est de cette migration est double. Tout d'abord, beaucoup

d'utilisateurs sont à l'aise et préfèrent la navigation à travers un navigateur. Il y a peu, concevoir des applications Web avec des interfaces qui offraient les mêmes possibilités d'interactions, la même fluidité et autant d'ergonomie qu'une application classique paraissait impossible. Avec l'avènement du Web riche, la conception et le déploiement de telles applications sont devenus possibles. Grâce à de nouvelles technologies et standards, les développeurs d'applications Web peuvent dorénavant concevoir des applications ayant les mêmes aspects et offrant les mêmes avantages et fonctionnalités que les applications Desktops. Enfin, deuxième intérêt, les administrateurs de système se retrouvent à gérer quelques serveurs à la place d'une multitude de machines. Les applications ne sont plus déployées sur les postes clients.

L'arrivée en masse du concept de Web 2.0 laisse donc supposer une forte croissance du nombre et de la complexité des applications sur le Web, quel que soit le contexte de leur utilisation.

1.2 Les pratiques de développement actuelles et leur inadéquation au marché

Face à cette croissance, nous trouvons aujourd'hui des spécialistes centrés, soit sur la conception et la programmation, soit sur l'ergonomie des systèmes Web. Malheureusement, avec l'émergence de la notion du Web 2.0 et des clients riches, les métiers de l'ergonomie et de la programmation se télescopent. En effet, la partie IHM des applications Web ne se limite plus à la simple mise en place d'une charte graphique. Elle intègre de plus en plus de traitements avec pour objectif d'augmenter la convivialité et de limiter le nombre des transactions.

Le domaine du Web connaît chaque jour de nouvelles normes, standards et technologies. Pour rester compétitive et pour offrir de meilleurs services, les éditeurs s'obligent à les intégrer. Pour continuer à respecter les délais de livraisons, exigés par les clients, les éditeurs ont souvent tendance d'une part à sacrifier les phases de tests [57] [56] et d'autre part à négocier une livraison en plusieurs étapes. Le découpage est un compromis entre les impératifs architecturaux et les priorités fonctionnelles exigées par le client. Les différentes parties seront ensuite livrées selon un calendrier faisant objet d'un contrat entre les deux parties. Malheureusement, cette solution peut occasionner plus de problèmes qu'elle n'en résout. Ces difficultés sont liées à l'utilisation de techniques de découpage modulaire sur des technologies qui s'y prêtent mal. Au final, cette approche peut engendrer des coûts d'assemblage plus importants et un non respect des délais de livraisons. Parfois, les coûts d'assemblage et d'intégration deviennent plus importants que ceux nécessaires aux développements de nouveaux composants métiers [33]. On retrouve fréquemment cette problématique dans les petites et moyennes structures.

Dans les entreprises ayant atteint un niveau de maturité conséquent (le plus souvent de grosses sociétés), la conception et le développement d'applications Web riches, s'appuient sur une architecture multi-tiers. Ce type d'architecture est souvent la meilleure décision de conception satisfaisant le passage à l'échelle, la maintenabilité et la fiabilité. Un gros travail a été réalisé ces dernières années pour améliorer les parties traitement (WebSphere Studio Application Developer, Rational Software Architect, etc.) et données (SQL Manager for Oracle, DbDesigner, etc.) des applications multi-tiers. La partie cliente (présentation) a souvent été, quant à elle, assimilée à un navigateur Web. Ces outils ne prennent en compte que les aspects

statiques et structurels des composants graphiques. Cela occasionne un travail supplémentaire aux développeurs afin d'implémenter les liaisons entre ces composants (liaisons non dynamiques). Ces solutions ne sont pas orientées sur les problématiques liées au développement de solution Web dynamique compte tenu de la complexité des interactions entre les composants du Web (traitement serveur, navigateur, librairie et accès aux composants métiers nécessite un effort considérable pour le suivi des évolutions Web). Plusieurs technologies, avec leurs outils associés, sont à ce jour proposées pour le développement de clients "Web riche" dynamiques (AJAX, Flash, XUL ou Eclipse RPC). Pour le moment, elles ne sont destinées qu'à mener à bien l'étape de codage. Une réflexion sur l'architecture de la partie cliente est rarement un sujet de préoccupation. En conséquence, ce *tier* pose de sérieux problèmes de maintenance et de d'évolution.

On le voit, les méthodes et technologies actuellement utilisées pour le développement des applications Web ne sont pas suffisantes. L'organisation des projets nécessite également d'être repensée. Les équipes de développement doivent être pluridisciplinaires [1]. De nos jours une application Web peut nécessiter, en plus des compétences de spécialistes de certaines technologies, l'expertise d'un géomaticien pour la partie cartographique ou d'un psychologue ergonomiste pour le choix des couleurs et des styles. Il faut également prendre en compte la répartition généralement matricielle adoptée pour les projets. Les développeurs interviennent souvent simultanément sur plusieurs projets. De nouvelles méthodes de développement Web doivent émerger et prendre en considération toutes ces contraintes. Il est également nécessaire de disposer d'un outillage supportant ces méthodes pour optimiser la qualité des productions tout en minimisant les coûts et les délais.

1.3 Les promesses de l'approche à base de composants

Toute méthode de développement fait l'hypothèse d'un certain paradigme d'analyse, de conception ou de programmation. Ce paradigme met en avant une philosophie de développement que l'on pense utile au domaine concerné par la méthode. Le concept de composants réutilisables est apparu au milieu des années 80. Du fait de ses résultats encourageants, ce paradigme tend depuis quelques années à s'imposer dans le domaine du développement Web. Plusieurs entreprises proposant des solutions pour le développement ont conçus une gamme de produits basée sur cette technologie. Ces environnements de développement sont maintenant mis en oeuvre dans les projets menés par de grandes sociétés de services.

Le nombre des serveurs d'application disponibles montre clairement les efforts réalisés dans ce domaine. Actuellement, la différence entre ces différentes solutions à base de composants se situe dans les parties service après vente et formation. En effet, suite à des années de recherche et d'optimisation, les performances techniques (fiabilité, occupation mémoire, temps de réponse, ...) de ces solutions sont proches. Elles facilitent la conception d'applications possédant des architectures optimisées, sûres et robustes, et cela en rendant possible l'utilisation de patrons ou de modèles réutilisables étudiés et adaptés à pour chaque type de besoin. Grâce à ces serveurs, les concepteurs et les développeurs d'applications voient leurs efforts de développement soulagés.

Malheureusement, pour une entreprise qui dispose d'une masse importante de codes capi-

talissant plusieurs années de savoir-faire, le passage à une approche par composant représente une difficulté majeure. Des craintes justifiées au sujet de cette migration apparaissent concernant ses coûts et sa fiabilité, mais surtout la manière de procéder. Les entreprises ne veulent pas perdre des années de travail. Elles veulent également optimiser les coûts de formation de leur personnel et mettre en place d'une manière progressive des processus de développement adaptés.

1.4 Contexte de la thèse

La société Alkante est spécialisée dans le conseil et l'ingénierie des technologies de l'information. Elle développe pour ses clients des systèmes d'information et plus spécialement des systèmes d'information géographique orientés Web. Ces dernières années, Alkante observait une croissance de la complexité des applications Web qu'elle développait. Elle était confrontée aux difficultés identifiées dans les sections qui précèdent.

A la recherche d'une réponse à ses difficultés, la société Alkante choisit de s'associer à un partenaire académique. Elle proposa, donc, à l'équipe Software Evolution (SE) du laboratoire VALORIA, un travail de recherche portant sur **le développement et l'évolution d'applications Web à base de composants**. Elle profitait ainsi des compétences de l'équipe SE dans le domaine des méthodes et des outils facilitant les activités de développement et de maintenance des logiciels à base de composants. La collaboration se concrétisa par l'établissement de ma thèse en convention CIFRE. Dans le cadre de cette convention, l'équipe SE se chargeait de mon encadrement sur les parties théoriques de l'étude. La société Alkante, quant à elle, participait très activement à l'expression des besoins et à l'évaluation des solutions proposées.

Le travail présenté dans ce mémoire de thèse est le résultat de cette collaboration université-entreprise qui a permis à Alkante de se voir proposer des réponses théoriques à ses problèmes de développement et à sa stratégie. Elle a aussi donné l'opportunité à l'équipe SE de connaître les attentes des entreprises dans le domaine de l'évolution, la maintenance et le développement de logiciels à base de composants. Le travail fut en définitive très bénéfique pour les deux parties et ouvrit des projets de nouvelles collaborations entre elles.

1.5 Travail présenté dans cette thèse

L'objectif de cette thèse dans le cadre de la collaboration était d'offrir un cadre (outils et méthodes) pour faciliter, dans le paradigme composant, le développement et l'évolution d'applications Web.

Bien que la société Alkante ait fait le choix stratégique d'user du paradigme composant logiciel, elle ne pouvait pas se permettre d'abandonner son patrimoine très important de codes. Mon premier travail a donc consisté à proposer un protocole de migration permettant un passage en douceur d'un processus de développement classique à un processus de développement à base de composants logiciels [33] [32]. Ce protocole a rendu possible la réutilisation par l'entreprise de ses codes existants et a permis à ses développeurs de se familiariser avec les concepts et les bases du développement à base de composants logiciels [33, 32].

Une fois cette migration effectuée, la suite de mon travail a consisté à proposer une nouvelle approche pour la modélisation d'applications Web dans un contexte purement composant. Cependant, une autre question liée à la maintenance et aux coûts d'évolution se posait. Les applications Web sont particulièrement sujettes à de nombreuses évolutions, très régulières dans le temps. Afin de maîtriser l'évolution des applications développées et pour réduire les coûts de maintenance, j'ai utilisé une solution appelée *contrats d'évolution*. Cette solution a fait l'objet d'une thèse [?] et a été validée dans un contexte purement académique [?]. Ainsi, l'intégration du *contrats d'évolution* dans le processus de développement d'Alkante valide le concept dans un contexte industriel.

1.6 Organisation du mémoire

Les sections précédentes ont présenté la problématique et le contexte de la présente thèse. Le reste du mémoire est organisé comme suit :

Partie II : L'état de l'art des travaux de recherche en relation avec cette thèse est présenté dans cette partie afin de cerner plus précisément les problématiques traitées. Partant du principe que le développement à base de composants logiciels est à présent suffisamment mature, cette partie est entièrement consacrée à l'ingénierie des applications Web. Cette partie est organisée en deux chapitres :

Chapitre 2 : Le premier chapitre retrace l'histoire et l'évolution des applications Web. Je présente dans ce chapitre, les différentes générations d'applications Web. Je donnerai plus d'importance à la cinquième génération car elle constitue la cible de mes travaux et représente la génération actuelle des applications Web. Dans ce chapitre, je décris les différentes classifications des applications Web. Ces classifications sont nécessaires à la compréhension des concepts et technologies traités dans cette thèse. La classification d'une application Web est différente de son évolution et ne correspond pas à une génération précise. Il peut y avoir différentes générations d'applications Web dans la même classification. Je terminerai ce chapitre par une synthèse sur les caractéristiques des applications Web et ce qu'il les différencie du reste des applications informatiques.

Chapitre 3 : Dans ce chapitre je présente les travaux réalisés dans le domaine de l'ingénierie des applications Web. Je porterai plus d'attention aux deux principaux courants de modélisation d'applications Web. Le premier regroupe les techniques de modélisation utilisant UML ; Le deuxième regroupe les techniques de modélisation basées sur un langage appelé WebML. À la fin de ce chapitre, je me positionnerai par rapport aux travaux existants et mettrai en évidence la contribution de mon approche.

Partie III : Cette partie détaille la contribution de cette thèse. Elle est composée de trois chapitres présentant mes travaux afin de répondre aux problématiques introduites dans la partie I.

Chapitre 4 : Ce chapitre est consacré à la présentation de l'approche proposée pour la migration d'un processus de développement classique vers un autre à base de composants logiciels. J'introduis ce chapitre par une synthèse sur les composants

logiciels et le développement à base de composants logiciels afin de fixer la lexique utilisé par la suite. J'illustre mon approche qui est basée sur une architecture dite transitoire. Ainsi, je propose un processus de développement logiciel organisé autour de cette architecture. Je conclus ce chapitre par une expérimentation dans le contexte de développement d'Alkante.

Chapitre 5 : Dans ce chapitre, je présente mon approche pour la modélisation et le développement d'applications Web à base de composants logiciels. Je l'introduis par une synthèse et une discussion sur les modèles de composants logiciels existants. Je finis ce chapitre par la présentation du prototype développé pour concrétiser mon approche.

Chapitre ?? : Ce chapitre est consacré à l'intégration du concept de *contrat d'évolution* dans mon processus de développement. Une expérimentation des avantages de l'utilisation du *contrat d'évolution* conclura ce chapitre.

Partie ?? : La dernière partie contient deux chapitres qui clôturent ce mémoire.

Chapitre ?? : Synthétise les apports de cette thèse sur les plans conceptuel et pratique.

Chapitre ?? : Ce chapitre vise à mettre en avant les limites de l'approche a proposée, et a proposer des pistes d'amélioration possibles. Ces dernières constituent un travail important que le temps imparti à cette thèse n'a pas permis d'approfondir. Elles constituent des ouvertures pour des travaux futurs.

Deuxième partie

État de l'art

Chapitre 2

Historique et évolution des applications Web

Sommaire

2.1	Histoire du développement Web	19
2.2	Application Web de cinquième génération	22
2.3	Classification des applications Web	23
2.3.1	Application Web 1.0	23
2.3.2	Application Web 2.0	23
2.3.3	Application Web mobile	24
2.3.4	Application Web Sémantique	24
2.3.5	Application Web riche	24
2.4	Caractéristiques et Complexité des applications Web	26

Cette partie de l'état de l'art sera consacré à l'historique et à l'évolution des applications Web. Je commencerais ce chapitre par l'histoire de l'ingénierie du développement Web, je présenterai les différentes générations et particulièrement la cinquième génération d'applications Web, je classifierai ensuite ces applications par type et enfin, je conclurai ce chapitre par la présentation des caractéristiques et des complexités liées aux développement de ce genre d'applications.

2.1 Histoire du développement Web

L'ingénierie du développement Web (Web engineering) est une discipline du développement logiciel relativement nouvelle . L'expression *développement Web* n'était pas largement utilisée avant 1998. Cette année correspond à la neuvième conférence d'ACM sur les Hypertext et les Hypermedia [9]. Bien que cette définition ait été introduite par Murugesan en 1997, elle reste en contraste avec celle du développement d'hypermedia (Hypermedia engineering) définie bien avant les années 90.

L'ingénierie du Web peut être définie comme : *“the use of scientific, engineering, and management principles and systematic approaches with the aim of successfully developing, deploying and maintaining high quality Web-based systems and applications”* [51].

N'accordant pas d'importance au développement des applications Web, la plupart des travaux menés ces dernières années ne sont que de simples aperçus ou de simples prises de position liées à la qualité dans le domaine du Web. Beaucoup de ces travaux avaient décrit les besoins exprimés par l'ingénierie Web en terme de processus et de réutilisabilité. Ils avaient aussi clairement défini les limites des sites Web. D'autres travaux comme ceux de Powell [54] ont constaté le changement rapide dans la nature des projets Web et avaient tenté de codifier certains rôles principaux propres à ce type de projets.

La Figure 2.1 illustre les travaux consacrés à l'ingénierie du développement Web. Elle représente différents domaines reliés par des flèches continues. Les lignes discontinues illustrent les connexions de chacun de ces domaines à l'ingénierie du développement Web.

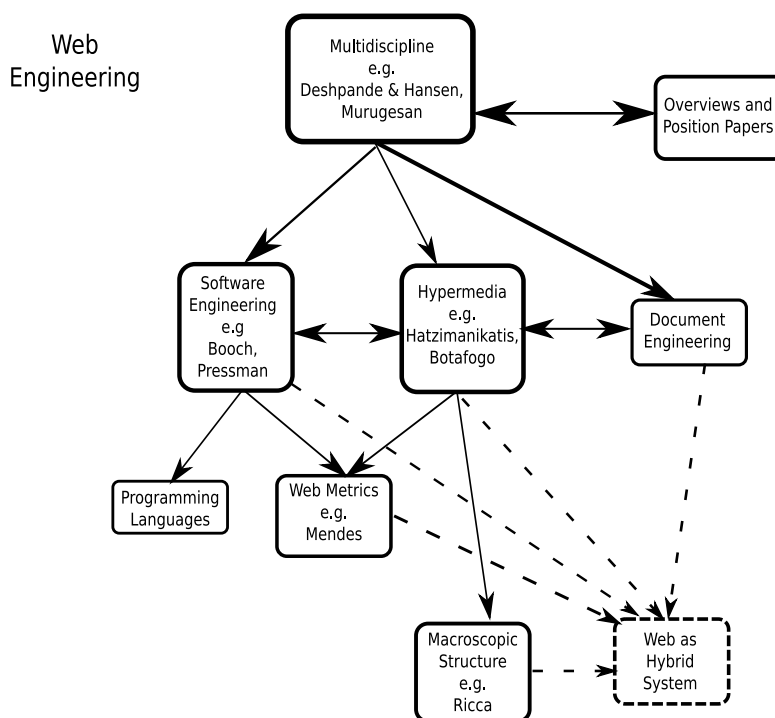


FIG. 2.1 – L'ingénierie du Web

Afin de comprendre le besoin en processus et méthodes structurées pour le développement d'applications Web, il sera préférable de se référer aux travaux de Powell [54], qui a décomposé l'évolution des applications Web en cinq générations illustrées dans la Figure 2.2.

Ces cinq générations peuvent être illustrées de la manière suivante :

- La première génération était basée sur des navigateurs entièrement textuels, la partie multimédia ne faisait pas partie de ses applications. Les efforts étaient concentrés sur le

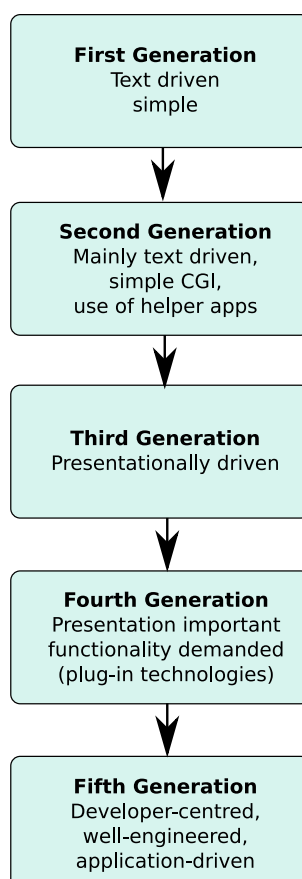


FIG. 2.2 – Les cinq générations de l'ingénierie du développement Web

contenu mais non sur la présentation. Le premier exemple de ce type d'application est le site du CERN en Suisse (le premier serveur Web publique)¹.

- La seconde génération, comme la première, fut très simple, ses applications représentaient un ensemble de pages Web statiques ou des technologies de type CGI très basiques. Les projets initiaux étaient souvent ad hoc et possédaient de très faibles budgets.
- La troisième génération a vu le jour grâce à la naissance de la première version du navigateur Web Netscape, la partie présentation prenait de plus en plus d'importance. Dans cette génération beaucoup d'applications ou de sites Web étaient plus orientés apparence que contenu. L'utilisabilité, quant à elle, était très négligée par les concepteurs. Cette opinion est partagée par Jakob Nielsen [50] lors de ses enquêtes sur l'utilisabilité dans les applications Web.
- Dans la quatrième génération, les utilisateurs exigeaient plus de fonctionnalités. Cela s'était traduit par de nouveaux contenus, par l'introduction de nouveaux médias et par l'utilisation de nouveaux langages de programmation (comme par exemple Java) pour offrir plus d'interactivité. La plupart de ces fonctionnalités étaient disponibles sous forme

¹<http://info.cern.ch>

de plug-ins ou de scripts interprétés par le client (le navigateur).

- De nos jours, les applications Web sont en train de rentrer dans une cinquième génération, cette génération n'est plus dirigée que par les besoins des utilisateurs mais aussi par ceux des développeurs. La croissance de la taille et la complexité de ces applications sont la cause de cette évolution.

2.2 Application Web de cinquième génération

Une application Web est une application déployée sur un serveur Web pour que son contenu soit délivré sur un réseau. La caractéristique essentielle de cette forme d'application est qu'elle repose sur l'utilisation d'un navigateur Web dit 'client léger'. De nos jours, le navigateur Web est présent sur de nombreux supports matériels (Smart-Phone, Téléphone mobile, Borne interactive, ...etc.) et pratiquement, tous les systèmes d'exploitation en proposent.

Ces dernières années, le coté application de cette génération s'est excessivement complexifié, le nombre de serveurs Web n'a cessé d'augmenter faisant ainsi croître le nombre de niveaux de l'architecture. Chacun de ces niveaux possède un rôle dans l'application. Il existe un niveau pour afficher l'interface utilisateur, un niveau pour contrôler et gérer les actions de l'utilisateur, un niveau pour gérer l'accès et la mise à jour des données, et un niveau pour assurer la communication entre les différents niveaux. Ainsi la nomination des niveaux correspondant aux différents rôles cités est la suivante :

- ***Niveau interface utilisateur***

A ce niveau, les technologies utilisées sont celles des pages Web classiques (HTML, Javascript, styles CSS). Il faudra alors manipuler des documents, des styles, des formulaires, des scripts et des balises.

- ***Niveau logique métier (logique business ou logique fonctionnelle)***

Ce niveau est implémenté par des technologies de génération dynamique de page Web, ces technologies sont souvent associées à un langage de programmation tel que PHP, JSP (Java) ou ASP (C#).

- ***Niveau stockage et accès aux données***

Ce niveau peut être, soit de type objet, soit de type relationnel. Les technologies de type objet disponibles sont celles des langages objet utilisés pour développer le niveau contrôle, à savoir : Java et J2EE, ASP et .NET et la version objet de PHP. Les éléments architecturaux utilisés sont ceux de la technologie objet : la classe, l'opération, la propriété, l'objet. Une base de donnée relationnelle est utilisée lorsque les données sont persistantes, elle l'est, soit via un serveur d'application gérant la persistance des objets, soit explicitement lorsque ce serveur n'est pas utilisé. Les technologies utilisées sont les SGBD relationnels du marché : MS-SQL-Server, Oracle, MySQL, PostgresSql. Quant au type relationnel, il représente le modèle des données dans lequel les notions de relations, attributs, types, clés sont utilisées. Pour faire communiquer ces technologies, des bibliothèques (API) existent.

Ces multiples technologies et modèles rendent les architectures des applications Web de plus en plus complexe. Concevoir et implémenter une application Web nécessite actuellement un minimum de compétences dans différents domaines technologiques (réseau, bases de données, programmation objet, HTML, ...etc.). Malgré l'existence de frameworks de développement, tel que Struts, Jfaces, Webform (.Net), ou encore Spring qui définissent une organisation logique des composants d'une application Web, sa conception continue à demander un investissement et un travail très importants.

2.3 Classification des applications Web

Nous pouvons classer les applications Web selon les technologies utilisées pour leur implémentation de la manière suivante :

2.3.1 Application Web 1.0

Les premières applications Web étaient statiques : ce n'était qu'un ensemble de pages de type HTML reliées entre elles par des liens hypertextes offrant de l'information. peu après, elles sont devenues dynamiques en offrant aux utilisateurs des pages générées à la volée. La possibilité de créer des pages Web à partir d'un contenu stocké dans une base de données avait donné ensuite aux développeurs la possibilité de personnaliser l'information offerte aux utilisateurs. Ces applications, dites dynamiques, offraient à l'utilisateur un seul sens d'interaction et limitaient l'interactivité. L'utilisateur n'avait aucun rôle dans l'édition du contenu. Les applications Web statiques et dynamiques ayant peu d'interactivité sont appelées, de nos jours, *application Web 1.0*

2.3.2 Application Web 2.0

Ces dernières années, nous avons assisté à l'émergence d'une nouvelle classe d'applications Web plus orientées services et appelées Web 2.0. Elles permettent aux utilisateurs de collaborer et d'échanger leurs informations en ligne. Cette nouvelle génération d'applications est aussi appelée *Widson Web*, *People-centric Web* ou *Read/Write Web*. Elles offrent à l'utilisateur des interfaces intelligentes conçues pour lui faciliter l'édition et la génération de nouveaux contenus. Elles lui offrent également la possibilité de les modifier ou de les supprimer. Ce nouveau type d'applications est aussi capable d'intégrer de multiples services dans une seule interface utilisateur. Avec l'apparition de nouvelles technologies comme AJAX (Asynchronous Javascript and XML), Ruby, blog ou wiki le Web est rapidement devenu plus dynamique et plus interactif. À présent, les utilisateurs ne sont plus limités à récupérer de l'information d'un site, ils peuvent également y contribuer. Ces technologies offrent à présent à l'utilisateur la possibilité de récupérer du contenu mis à jour dans l'application sans même visiter le site. L'autre point fort du Web 2.0 est la prolifération des applications Web possédant des API (Application Programming Interfaces). Une API de Web services offre par exemple aux développeurs la possibilité de récupérer des données d'une application et d'en créer une autre

avec ces mêmes données. Le Web 2.0 est une collection de technologies, de stratégies commerciales et de tendances sociales. Pour plus d'informations sur le Web 2.0, le lecteur pourra se référer à Murugesan [47] [53].

2.3.3 Application Web mobile

Les avancées technologiques que connaît le monde de la mobilité et du sans fil, ainsi que la miniaturisation spectaculaire des équipements tels que les PDA et les Smart phone, ne font qu'augmenter le nombre des utilisateurs des applications Web. Les téléphones mobiles pourront bientôt concurrencer les ordinateurs (fixes et portables) en ce qui concerne l'accès aux applications Web. Selon une enquête Ipsos 2008, 28% des mobiles possédant des navigateurs Web et une connexion Internet ont consulté des application Web en 2005. L'accès à Internet et aux applications Web est devenu quelque chose de quotidien pour les utilisateurs de certains pays développés. Conscient de cette émergence le Consortium World Wide Web a décidé une nouvelle ouverture appelée *Mobile Web*. Les applications Web mobiles peuvent offrir des nouvelles fonctions aux applications Web telles que le positionnement GPS.

2.3.4 Application Web Sémantique

Dans les applications Web actuelles, les informations sont représentées dans le langage naturel, ce langage est compréhensible et manipulable par l'humain et non par les ordinateurs. Le Web sémantique espère lever cette barrière. Il n'est que l'extension du Web actuel où l'information est restituée dans son sens le plus correct permettant ainsi une meilleure collaboration entre l'homme et la machine [8]. Elle tend à créer un médium d'échange d'information universel avec une sémantique pour le Web. L'ajout de la sémantique aux applications Web va radicalement changer la nature du Web, de la partie où l'information a été visualisée à celle où elle est interprétée, échangée et transformée. Associer le sens au contenu permettra un très haut degré d'automatisation, produira plus d'applications intelligentes et facilitera l'interopérabilité des services. Les trois principales clés du Web sémantique sont le marquage sémantique, l'ontologie et les systèmes à agents intelligents. Pour plus d'informations sur le Web sémantique le lecteur pourra se référer aux travaux d'Antonioni et Harmelen [4] et de Berners-Lee [8] et Shadbolt [64].

2.3.5 Application Web riche

Les applications Web riches AWR sont des applications qui fonctionnent grâce à des navigateurs internet et ne nécessitent aucune installation. Ces applications possèdent les mêmes fonctions et services que les applications desktop traditionnelles. Cette définition a été introduite pour la première fois dans un livre blanc de Macromedia en Mars 2002. AWR représente l'évolution du navigateur Internet, du stade d'interface requête-réponse statique à un stade d'interface asynchrone et dynamique.

Les AWR, selon les fonctionnalités pour lesquelles elles ont été développées et déployées, peuvent être classées selon trois catégories :

1. La première est dépendante d'un plug-in et donc dépendante de la plate-forme où elle a été déployée.

Les technologies Flex2 d'Adobe et Java Web Start de Sun font partie de cette catégorie. La première utilise une technologie Xml appelée MXML pour la génération d'application Web dans la technologie Flash, la deuxième utilise le Java Network Launch Protocol (JNLP) pour exécuter une application Java dans un navigateur Web. Dans cette catégorie les applications sont souvent embarquées à l'intérieur de pages HTML à l'aide de balises spécifiques.

L'avantage majeur de ce genre d'application *plugeable* est la simplicité de leur développement, elle assure aux développeurs un environnement de développement et d'exécution stable (sans surprise et multi-plateforme) à condition qu'il soit disponible chez le client. Il existe plusieurs outils pour le développement d'AWR plugeable tels que Microsoft Windows Smart Client ou encore Open Laszlo.

2. Le second type d'AWR est le plus connu. Il s'agit des applications dites AJAX. Ces applications emploient une combinaison de technologies existantes (XHTML/HTML, CSS, DOM et Javascript). L'idée est d'utiliser CSS et HTML pour les styles et la partie présentation et interface, utiliser Javascript pour faire des appels asynchrones aux serveurs et enfin DOM pour dynamiser le rendu des interfaces. Cette catégorie d'applications ne nécessite aucun plugin ou autre logiciels pré-installés chez le client. Cependant, ces applications réservent parfois de mauvaises surprises aux utilisateurs car elles ne sont pas compatibles avec tous les navigateurs et s'exécutent de manières différentes sur des systèmes différents. À cause de cette contrainte, les développeurs de ce genre d'applications doivent fournir plus d'efforts et de précautions pour les rendre multi-plateformes. La stratégie de développement la plus utilisée pour le bon développement de cette catégorie d'application est de considérer dans un premier temps que le navigateur n'interprète pas le javascript et de développer l'application dans un contexte d'application Web classique. Le Javascript est ensuite utilisé pour les pages qui nécessitent d'être dynamiques. Ce genre de développement est souvent long, complexe et nécessite de connaître toutes les incompatibilités pour toutes les plate-formes et navigateurs. Il existe cependant des frameworks développés pour alléger ce genre de traitement comme Dojo ou ScriptAculos.
3. La troisième catégorie d'AWR est basée sur les navigateurs. Elles utilisent un langage généralement basé sur XML et interprété par le navigateur pour bâtir des interfaces utilisateurs. Un exemple de cette catégorie d'application est le langage XUL de Mozilla. L'avantage de cette catégorie d'applications est qu'elles sont basées sur des standards W3C comme CSS 1 et 2, DOM 1 et 2 et Javascript 1.5. XUL est indépendant de la plate-forme et les applications conçues en XUL sont interopérables.

L'élément commun entre ces trois catégories est leur côté dynamique et l'excellent rendu de leur interface utilisateur. En effet, les actions et les traitements se font en tâche de fond, l'utilisateur n'est jamais handicapé par les temps de traitement des serveurs. De plus, les données sont affichées et mises à jour progressivement. Dans les applications Web riches et contrairement aux applications Web classiques, seules les parties de l'interface concernées par les

modifications sont rechargées et non pas la totalité de l'interface ou de la page. Grâce à ce système les applications génèrent moins de trafic dans le réseau.[7]

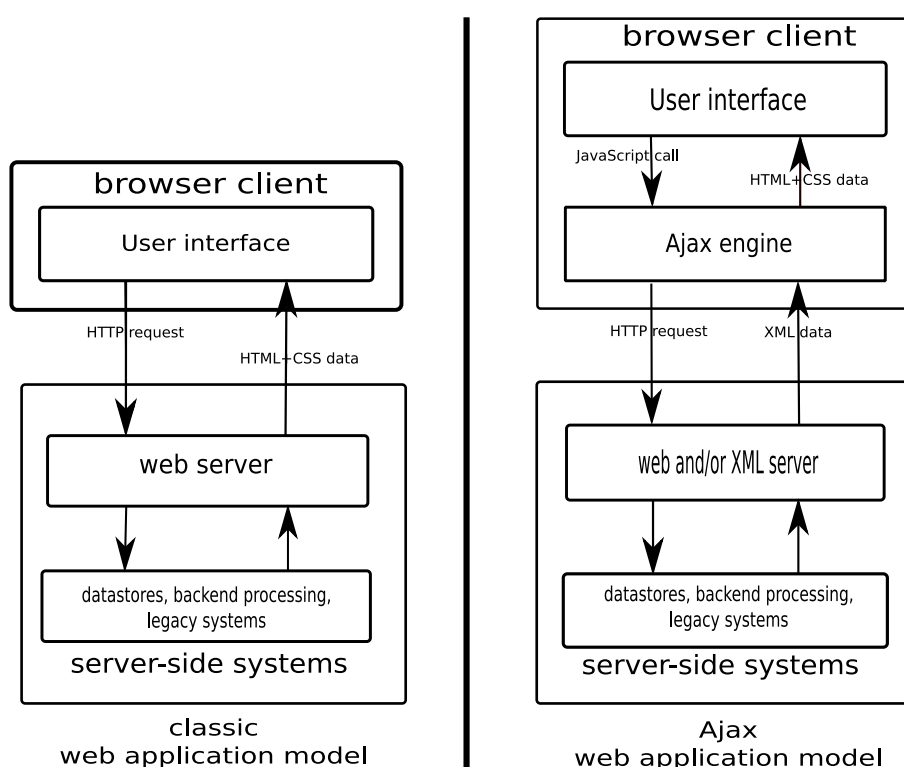


FIG. 2.3 – Aperçu du méta modèle UWE

2.4 Caractéristiques et Complexité des applications Web

Le Web est un domaine particulier de l'informatique. Par conséquent, la compréhension de ses caractéristiques est essentiel pour la conception et la modélisation de meilleurs systèmes et applications. Celles-ci en plus de satisfaire les besoins des utilisateurs, doivent être conformes aux différents standards de développement et de qualité logiciels (robustesse, maintenance,...etc.). Les applications Web possèdent certaines caractéristiques qui les rendent différentes des logiciels et applications classiques. Ces caractéristiques rendent leur développement différent et compliqué comparé à un développement traditionnel. L'environnement opérationnel, les approches de leur développement et la vitesse à laquelle ces applications sont développées et déployées les rendent déjà différentes des applications et logiciels informatiques traditionnels. Les aspects liés à la sécurité de ces applications est plus important et plus critique que ceux dédiés aux applications classiques.

Les principales caractéristiques de ces applications sont [48] :

- Les applications Web sont, par nature, sujettes à des évolutions fréquentes et constantes,

elles requièrent des changements fréquents de leur contenu, de leurs fonctionnalités, de leurs structures, de leur navigation et de leur présentation ou même de leur implémentation. Elles ont une particularité liée à l'instabilité de leurs besoins, spécialement quand le système est déployé et mis en service. Dans la majorité des applications Web, la fréquence et le degré de changement des spécifications sont plus élevés que dans les applications traditionnelles. Dans quelques systèmes, il arrive que celles-ci ne soient pas identifiables au début du projet.

- Les applications Web sont, dans la majorité des cas, conçues pour être utilisées par des utilisateurs possédant des profils différents. Leurs interfaces utilisateur doivent répondre à différents besoins, elle doivent également s'adapter à différentes personnes. En outre, le nombre des utilisateurs est imprévisible et peut être extrêmement variable, créant ainsi des problèmes de performance.
- Les applications Web nécessitent l'utilisation et la manipulation d'une grande variété de contenu (texte, image, audio, video, ...etc). Ce contenu peut être généré dynamiquement et doit être présenté de manière attractive. L'esthétique dans les applications Web possède une place très importante lors de la conception et même après son déploiement.
- Les applications Web sont accessibles depuis différents supports matériels, par différents navigateurs internet et à partir de différents réseaux possédant différents débits de connexion.

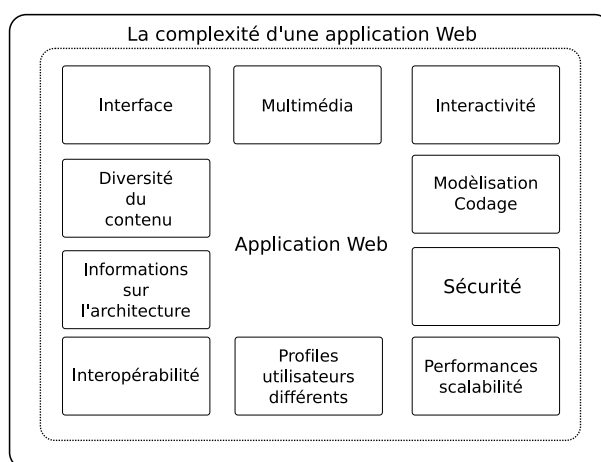


FIG. 2.4 – Complexité des applications Web

Beaucoup de personnes considèrent que le plus grand obstacle à l'adoption du développement Web est le contexte économique dans lequel ce type d'application s'insère. En effet, ce contexte est différent de celui dans lequel se situe une application informatique classique [20] :

- Temps de développement réduit,
- Budget peu important,
- Évolution rapide et constante possédant des cycles courts de mise à jour,
- Contenu, très important, souvent intégré dans le code de l'application,
- Spécification insuffisante des besoins,

- Technologies émergentes,
- Petites équipes de développement,
- Manque de procédure de test,
- Importance de la satisfaction de l'utilisateur et concurrence forte,
- Management de projet minimal,
- Aspect critique de performances,
- Nombreux standards et normes à utiliser,
- Variétés des disciplines à investir (hypertexte, conception graphique, ergonomie, ..),
- Gestion de la sécurité
- Aspects sociaux, légaux et éthiques à considérer
- Contexte social de l'équipe de développement (âge, expérience, savoir-faire)
- Évolution rapide des environnements de développement

La complexité est un phénomène omniprésent dans les applications Web. Les applications Web possèdent toutes les caractéristiques des systèmes complexes comme l'hétérogénéité, le très grand nombre de composants interactifs et une évolution rapide et constante. Plusieurs facteurs contribuent à la complexité des systèmes Web, ces facteurs sont illustrés sur la Figure 2.4. La complexité des applications Web n'est pas seulement liée à leur complexité technologique, mais également à leur complexité organisationnelle. Les développeurs doivent être conscients de la complexité de ces applications et de la manière dont elle affecte leurs projets. Cette complexité peut être réduite par l'utilisation d'approches et méthodes appropriées.

Chapitre 3

Ingénierie des applications Web

Sommaire

3.1	Un diagnostic sur l'ingénierie du développement Web	29
3.2	Évolution des méthodes de modélisation des applications Web	30
3.3	Ingénierie Web basée sur UML	33
3.3.1	Phases de modélisations avec UWE	34
3.3.2	Méta modèle UWE	36
3.4	Modélisation des applications Web avec WebML	36
3.4.1	Historique de WebML	37
3.4.2	WebML	37
3.4.3	Modélisation WebML avec UML	39
3.5	En résumé	40

Cette partie de l'état de l'art est consacrée à l'ingénierie Web. Je commencerai ce chapitre par l'établissement d'un diagnostic sur le développement d'applications Web, j'aborderai ensuite les méthodes de modélisation de ces applications et je décrirai leurs évolutions. Dans ce chapitre, je présenterai les deux principaux courants de modélisation des systèmes Web. Le premier regroupe les méthodes de modélisation basées sur UML et le deuxième décrit la méthode la plus utilisée pour la la conception de ces systèmes. Enfin, je conclurai ce chapitre par une synthèse des forces et faiblesses des méthodes présentées.

3.1 Un diagnostic sur l'ingénierie du développement Web

La plupart des développeurs d'applications Web ne prêtent qu'une petite attention à l'analyse des besoins, au processus développement, à la modélisation, à la qualité logicielle, à l'évaluation des performances, à la gestion de projet et à la maintenance [69]. Une grande partie du développement d'applications Web est liée aux connaissances et à l'expérience d'un individu ou d'un petit groupe de développeurs. Elle est beaucoup plus orientée expérience que standards.

Beaucoup de personnes considèrent que la problématique du développement Web est plus liée à la problématique d'édition de documents qu'à celle du développement de logiciels. Ils

considèrent le développement Web comme étant un art lié à la manipulation des médias et à la manière de les représenter. Certes, le développement Web comporte un côté artistique important mais il a également besoin d'autres disciplines. La complexité des applications Web s'est significativement accrue, elle est passée de la conception de simples applications de diffusion d'informations à celle de systèmes d'information d'entreprises, d'agendas, de systèmes de réservation et d'achats en ligne. Suite à cette complexité, plusieurs attributs qualité liés aux applications Web (navigabilité, accessibilité, évolutivité, maintenabilité, interopérabilité et sécurité) tendent à disparaître lors du développement.

Les développeurs d'applications Web utilisent souvent des approches ad hoc peu rigoureuses, ne possédant aucun modèle qualité. Les nouveaux problèmes soulevés par un grand nombre de développeurs d'applications Web sont liés aux progrès continus que connaît Internet actuellement, à la montée en charge des applications commerciales sur internet, à la volonté croissante des entreprises à avoir leur propre vitrine sur internet et à la nécessité de migrer des anciens systèmes vers des environnements Web. Dans ce nouveau contexte, les méthodes classiques de développement conduiront certainement à la production d'applications Web avec de faibles performances voire un échec.

Actuellement, la mauvaise modélisation et le mauvais développement peuvent avoir de graves conséquences. Une récente enquête sur les projets et les systèmes Web effectuée par le Cutter Consortium [17] a mentionné plusieurs problèmes :

- Les projets Web ne sont pas conformes aux besoins réels (84%),
- Les projets Web ne respectent pas les délais (79%),
- Les projets Web dépassent largement leur budget (63%),
- Les projets Web sont pauvres en spécifications fonctionnelles (53%),
- Les projets Web possèdent des livrables de mauvaise qualité (53%).

3.2 Évolution des méthodes de modélisation des applications Web

Conformément à la définition d'une architecture logicielle du standard IEEE 1471-2000, l'architecture d'une application Web peut être définie comme étant *l'organisation fondamentale d'un système décrivant un ensemble de composants, leurs interactions, leur environnement et le principe régissant leur modélisation et leur évolution*. Elle utilise des abstractions et des modèles pour décrire et simplifier la compréhension des structures complexes. L'architecture présente un Framework, décrit la structure et rend le système compréhensible. Elle facilite la transition de l'analyse à l'implémentation [23]. Des exemples d'architectures d'applications Web complexes sont donnés par Dantzing [18]. Lors de la conception de l'architecture d'une application Web, plusieurs composants du système sont décrits, ainsi que la manière dont ils sont assemblés. L'architecture logicielle d'une application Web décrit de manière générale le réseau, les serveurs (Web ou applications), les bases de données et leurs interactions. Elle décrit l'application et ses modules, ainsi que les fonctionnalités offertes par chacun. Elle identifie les différents modules et technologies nécessaires à son implémentation. Plusieurs facteurs peuvent influencer le choix d'une architecture logicielle :

- Les besoins fonctionnels,
- La qualité, la sécurité et les performances,

- Les aspects techniques,
- L'expérience.

La conception et la modélisation des applications Web a toujours été abordée sous plusieurs aspects. Chacun de ces aspects a été conçu par l'utilisation d'une méthode et d'un modèle appropriés. Chaque modèle décrit un élément spécifique à l'application Web (les données, les parcours, les interfaces). Le point commun à ces méthodes était de proposer des modèles pour représenter les différentes propriétés d'une application Web (information, navigation, interaction, esthétique, etc.). Associés à ces modèles, on trouve parfois des processus permettant le parcours d'un modèle à l'autre. L'une des premières méthodes préconisant un modèle et une démarche est RMM (Relationship Management Method) [30]. Elle repose sur un modèle entité association et a ultérieurement évolué pour s'adapter aux applications complexes exigeant une combinaison d'informations issues de plusieurs entités. RMM a ensuite servi de base à WebArchitect [67]. Cette méthode définit une version étendue des notations de RMM pour décrire l'architecture des systèmes d'information basés sur le Web.

En 1996, la méthode OOHDM [62] (*Object-Oriented Hypermedia Design Method*) a été la première à introduire les concepts orientés objet dans la conception des applications Web. Depuis, plusieurs autres méthodes orientées objet se sont succédées. Par exemple, (HFPM Hypermedia Flexible Process Modeling) [41] propose un processus d'analyse qui permet de couvrir la totalité du cycle de développement. (SOHDM Scenario-based Object-oriented Hypermedia Design Methodology) [42] se base sur les scénarios pour établir des diagrammes d'activités déterminant les structures d'accès aux noeuds. L'approche OO-pattern [68] se distingue par l'introduction de patrons de conception de navigation et de présentation.

A la fin des années 90s, plusieurs méthodes appropriées au développement d'applications Web ont été proposées. Comme par exemple WSDM (*Web Site Design Method*) [19] qui propose une conception d'applications Web en reprenant les différents niveaux de conception mais avec pour originalité de placer l'utilisateur au centre de l'approche. La méthode MoHyCan [28] qui préconise pour sa part l'utilisation de trois modèles objets qui se superposent comme des calques utilisés par les architectes pour concevoir les plans d'une maison, enfin MoHyCan qui elle, permet une prise en compte des données, des médias et de l'interactivité par une définition de parcours.

La méthode (*WebML Web Modeling Language*) [14] représente le résultat de travaux effectués autour des méthodes HDM et RMM. Elle se caractérise par l'utilisation du standard XML pour la description des modèles. Elle propose un processus sous la forme d'une itération de constructions de plusieurs modèles qui sont :

- Le modèle structurel : c'est le modèle entités-associations comportant un héritage simple et des relations binaires sans attribut. Deux entités prédéfinies, Group et User, sont disponibles pour la gestion des droits d'accès et de la personnalisation.
- Le modèle de composition : ce modèle a pour objectif de déterminer les pages Web à construire et ce qu'elles possèdent comme informations. Le concept de page est traduit par une entité graphique et une description XML. Le contenu des pages est formalisé par l'introduction d'unités de contenu. Elles sont de divers types et sont présentées également par un graphisme et une spécification XML.
- Le modèle de navigation : il s'agit du modèle qui permet d'établir les liens hypertextes

entre les unités et les pages du modèle de composition. Pour modéliser les liens autres que ceux utilisés pour la consultation des données, des concepts de saisie de données ou d'opérations de modification de contenu ont été mis en place.

- Le modèle de présentation : ce modèle permet de définir l'aspect des pages Web. Cette étape implémente des technologies de transformation XSL qu'elle rattache au contenu XML.

WebML est à la base de WebRatio [3]¹, un framework de conception d'applications Web. Cette suite logicielle comprend un IDE permettant la saisie des modèles et leur vérification, il permet également la génération d'une bonne documentation. Cette suite offre actuellement un générateur de code pour XSL, SQL et la plate-forme J2EE par l'utilisation du framework Struts². Sa mise en oeuvre via l'outil *WebRatio* a eu un certain succès dans le milieu industriel.

Avec le succès que connaît UML, la tendance est de l'intégrer aux processus de conception des applications Web. Un premier travail a été effectué par Conallen [31] qui a défini un ensemble de stéréotypes appropriés aux architectures Web. Ces stéréotypes sont particulièrement utiles à l'implémentation (génération des pages dynamiques JSP ou ASP). Cependant, ce travail ne peut pas être considéré comme une proposition de méthode puisqu'il ne couvre pas toutes les phases de la conception. *UML based Web Engineering Methodology, (UWE)* [49] se présente comme une proposition de conception plus complète car elle définit une extension UML (un méta-modèle) pour la construction de modèles de conception, de navigation et de présentation. Ces modèles sont complètement compatibles avec les mécanismes d'extensions préconisés par UML.

La méthode *OO-Hmethod* [27] repose sur l'utilisation d'UML pour la description conceptuelle des données. Elle propose deux autres niveaux de conception, navigation et présentation, le tout associé à un outil (*VisualWade*)³. Elle est capable de générer du PHP, du JSP ou du ASP ainsi que le modèle relationnel des données vers plusieurs SGBD.

Les nouvelles méthodes utilisent les nouvelles technologies tout en s'appuyant sur les concepts énoncés par les anciennes méthodes. Toutes proposent une modélisation du domaine, des parcours et de l'interface avec une meilleure prise en compte des différents types d'utilisateurs.

Plusieurs travaux ont été effectués par la communauté de l'ingénierie Web en exploitant la méta modélisation et les extensions UML pour définir des langages appropriés à la modélisation des applications Web [26] [31] [60] [46] [31] et pour proposer des extensions WAE (Web Applications Extensions) comportant les artefacts Web (pages Web, formulaires, frames, etc...). Le principal avantage du WAE est la simplicité avec laquelle le modèle reflète l'interface utilisateur. Malheureusement cette simplicité éprouve du mal à représenter la navigation tellement importante à ce genre d'applications. Cette méthode a vite atteint ses limites avec les applications Web complexes. Afin de résoudre ces limitations Gomez et Cahero ont proposé [26] un ensemble de vues qui étendent les diagrammes UML pour offrir un modèle d'interface pour les

¹<http://www.webratio.com>

²<http://struts.apache.org>

³<http://www.visualwade.com>

applications Web. Partant des diagrammes de cas d'utilisations et des diagrammes de classes, le modèle de navigation de l'application est décrit par un mélange d'UML et de propriétés primitives non implémentées par les outils de modélisation UML existants. Koch [38] et Baresi [5] ont indépendamment défini des extensions du méta-modèle UML 1.4 réduisant sa complexité pour la définition de nouveaux profils. Les profils ensuite définis étaient complexes à utiliser et leurs complexité augmentait avec la complexité des architectures des applications à modéliser. UML 1.4 a souvent été critiqué pour son manque de support dédié aux styles architecturaux. Ce n'est que dans la version 2 d'UML que les styles architecturaux ont été abordés [46]. Melia Gomez et Koch [46] ont été les premiers à utiliser les composants et les connecteurs UML 2.0 pour modéliser des applications Web. Ils les ont exploités pour définir les architectures Web de leurs applications. Le résultat fut une prolifération de modèles et stéréotypes rendant impossible leur intégration et leur implémentation dans un processus de génération de code.

3.3 Ingénierie Web basée sur UML

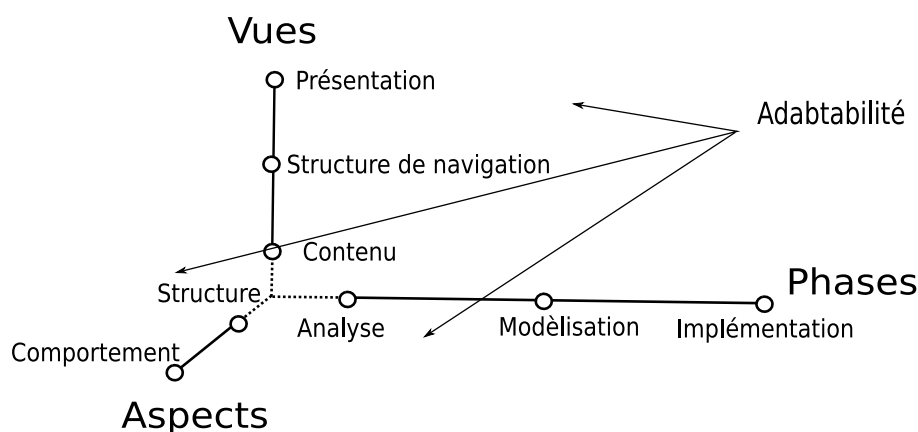


FIG. 3.1 – UWE [61]

L'ingénierie Web basée sur UML ou UWE, (UML-Based Web Engineering) est apparue à la fin des années 90s [6] [72] avec comme objectif la définition d'un standard pour la conception et la modélisation des applications Web. Ce nouveau standard devait implémenter les concepts des méthodes *OOHDM*, *RMM* et *WSDM*. Ce standard devait également utiliser un langage commun ou au moins être basé sur le même méta-modèle existant [40]. A cette époque, UML paraissait le standard idéal à cette initiative, du fait qu'il est le standard du développement logiciel de manière générale et les mécanismes d'extensions qu'il offre sont également les raisons de ce choix. L'idée véhiculée par l'UWE de définir un standard pour la modélisation Web n'est pas basée uniquement que sur UML. Elle utilise également XMI comme modèle de format d'échange, *MOF* pour la méta-modélisation, le principe de l'ingénierie dirigée par les modèles, le langage de transformation QVT et XML.

Pour faire face au phénomène de l'évolution et du changement constants des besoins propres

aux application Web et aux technologies utilisées, UWE a choisi une approche dirigée par les modèles et un processus basé sur les modèles et les transformations de modèles. Le modèle de l'application est enrichi à chaque étape du cycle de vie, de la spécification des besoins à l'implémentation en passant par l'analyse et la modélisation. Il permet la représentation des différentes vues de l'application Web correspondant à ses différentes propriétés (contenu, navigation, structure et présentation). Le modèle de contenu est utilisé pour spécifier le domaine de l'application et son type de contenu, La structure de navigation ou l'hypertexte sont modélisés séparément du contenu. Le modèle de navigation représente les différents chemins dans l'application Web. Le modèle de présentation prend en compte les utilisateurs et leurs interactions ainsi que leurs communications avec l'application. UWE propose au moins un type de diagramme UML pour la représentation de chacun de ces modèles. Plusieurs diagrammes d'activités et d'état-transition sont utilisés pour modéliser le comportement et l'aspect interaction des applications Web. La Figure 3.1 illustre les trois dimensions de modélisation dans l'UWE (vues du système, phases de développement et aspects).

La modélisation UML consiste dans la modélisation de l'aspect statique et dynamique du système par des objets et des diagrammes de classes, de composants, de déploiement, de cas d'utilisation, d'états et d'activité, de séquence et de communication. Les mécanismes d'extension offerts par UML sont utilisés pour définir des stéréotypes utilisés pour la modélisation des applications Web comme les noeuds et les liens. La notation UWE est définie comme étant un profil UML. L'avantage de l'utilisation des diagrammes UML est la compréhension commune et universelle de l'application. UWE a pour objectif la couverture et la modélisation de tout le cycle de vie d'une application Web, elle offre des techniques et des méthodes pour la modélisation des besoins et permet la navigation dans les différents modèles de l'application.

3.3.1 Phases de modélisations avec UWE

3.3.1.1 Spécifications des besoins

Comme pour toute application informatique, une application Web a pour première étape du développement la spécification et l'identification des besoins. UWE possède son propre modèle de spécification des besoins, elle propose deux niveaux de granularité pour les spécifier. Le premier, correspond à la simple description des fonctionnalités, il est modélisé par des diagrammes de cas d'utilisation UML. Le deuxième correspond à une description plus détaillée des fonctionnalités et des diagrammes de cas d'utilisation grâce à des diagramme d'activités UML par exemple. UWE distingue trois types de diagrammes de cas d'utilisation : le diagramme de navigation ; le diagramme processus et diagramme de cas d'utilisation personnalisé. Le diagramme de cas d'utilisation de navigation est utilisé pour décrire le comportement et les activités de l'utilisateur, telles que sa navigation dans l'application ou sa recherche par mots-clés. Le diagramme de cas d'utilisation processus décrit les tâches métier (transactions, opérations...), elles sont modélisées de la même manière que pour les applications classiques. Le troisième type impliquant la personnalisation de l'application Web est quant à lui modélise par des stéréotypes dénotés avec 'personalized'. La description détaillée des diagrammes de cas d'utilisation dépend de la complexité de l'application et des risque encourus par le projet. De manière générale, les diagrammes de cas d'utilisation ne sont pas suffisants pour la des-

cription de l'application [70]. Les concepteurs utilisent différents mécanismes pour raffiner ces diagrammes comme les workflows et les prototypes. En résumé, lors de la spécification des besoins dans UWE, trois buts sont visés : le but informationnel décrit le contenu des besoins, le but navigationnel décrit le type d'accès au contenu de l'application et le processus spécifie la capacité de l'utilisateur à exécuter quelques tâches dans l'application Web [55].

3.3.1.2 Définition du contenu

UWE utilise un modèle de contenu pour offrir une spécification visuelle du type d'information et du domaine traités par l'application. Lors de la modélisation, il est parfois nécessaire d'enrichir ce modèle par la définition de nouvelles entités comme les profils des utilisateurs, ces profils sont décrits par un modèle appelé modèle utilisateur ou également modèle de contexte. La séparation du contenu de l'utilisateur (du contexte) a bien su démontrer son importance en pratique. Les deux modèles sont représentés par des diagrammes de classes UML. Les relations entre le contenu et l'utilisateur sont modélisées par des associations UML.

3.3.1.3 Définition de la structure de navigation

La modélisation de la structure de navigation d'une application Web est basée sur l'analyse des besoins et sur le modèle de contenu. Les noeuds de la structure hypertexte sont modélisés par des classes appelées Classe de navigation, les liens hypertextes sont modélisés par associations et des liens reliant ces classes. D'autres alternatives de navigation telles que les menus sont représentées par des stéréotypes. Dans les applications Web possédant une logique applicative (business logic), des processus métiers doivent être intégrés dans le modèle de navigation. Les entrées et sorties du processus métier sont modélisées par des classes de processus. Les classes de processus, les menus et les autres accès primitifs sont hérités de la méta-classe UML Class, les liens de navigation et les liens entre processus sont hérités de la méta-classe UML Association. Pour plus d'information sur la modélisation de la structure de navigation des applications Web en UML, il faudra vous référer aux travaux de Koch et Kraus [38] ou à ceux de Knapp [37].

3.3.1.4 Processus métiers

La structure de navigation est étendue par des classes de processus qui représentent les points d'entrée et de sortie des processus métiers. Ces classes de processus sont conçues à partir des diagrammes de cas d'utilisation non navigables. L'intégration de ces classes à un modèle de navigation nécessite des mécanismes additionnels. Un seul diagramme de structure de navigation ne suffit pas à la modélisation d'applications complexes. Différentes vues de ce diagramme doivent être produites à partir du modèle de contenu se basant sur différents aspects de l'application telle que la navigation vers un contenu particulier. Chacune des classes de processus présentes dans le modèle de navigation est transformées en un modèle de processus. Les structures de contrôle et les données sont transformées en modèle de processus sous la forme des diagrammes d'activités UML. Les éléments de contrôle sont ajoutés à ce modèle pour modéliser la logique métier de l'application, les activités et les objets peuvent aussi être ajoutés à ces diagrammes d'activités. La structure du modèle de processus est sous la forme

de diagrammes de classe UML et décrit la relation entre les classes de processus et les autres classes utilisées pour modéliser les processus métier de l'application.

3.3.1.5 Modèle de présentation

Le modèle de présentation offre une vue abstraite de l'interface utilisateur de l'application Web. Il est basé sur le modèle de navigation et sur les éléments concrets de l'interface utilisateur comme les couleurs, les fontes de caractères et la localisation de ces éléments dans les pages Web. L'élément de base du modèle de présentation est la classe présentation, cette classe est basée sur les noeuds décrits dans le modèle de navigation, les classes de navigation, les menus, les accès primitifs et les classes de processus. La classe présentation se compose des différents éléments de l'interface utilisateur comme les textes, les boutons, les images et les formulaires.

3.3.2 Méta modèle UWE

UWE a défini un méta-modèle pour le développement Web orienté UML. Ce méta-modèle est l'extension du méta-modèle UML2.0, aucun élément du méta-modèle UML2.0 n'a été modifié. De nouveaux éléments ont été ajoutés, soit par héritage, soit par de nouvelles définitions. OCL a été utilisé pour spécifier la sémantique de ces nouveaux éléments. Le méta modèle UWE est profilable, ce qui veut dire qu'il est possible de le transformer en un profil UML [39]. Le méta-modèle UWE est compatible avec le méta-modèle MOF ce qui le rend compatible avec les outils basés sur le format d'échange standard XMI. L'avantage de ce méta-modèle est que tous les outils supportant les profils UML ou ses extensions sont utilisables pour la création des modèles UWE nécessaires au développement des applications Web. Il est alors possible d'étendre ces mêmes outils pour qu'ils supportent UWE. ArgoUML[37] possède une instance basée sur le méta-modèle UWE. L'extension UWE du méta-modèle UML consiste en l'ajout de nouveau package à UML, le premier appelé Core et le deuxième Adaptability illustrés dans la Figure 3.2.

3.4 Modélisation des applications Web avec WebML

Web Modeling Langage (WebML) fait partie de la troisième génération des méthodes de modélisation des applications Web. Il a été conçu en 1998 sur la base des méthodes de modélisation des hypermédias et des applications Web tels que HDM et RMM. Le but initial de WebML était la modélisation puis l'implémentation des applications Web dites data-intensive Web applications [14]. Ces applications étaient définies comme étant des sites Web permettant la maintenance et l'accès aux grandes masses de données. Ces données étaient souvent stockées dans des bases de données. Pour atteindre leur but, les concepteurs de WebML avaient repris des concepts existants pour la modélisation du Web et en avaient proposé de nouveaux pour exprimer la navigation et la composition des éléments hypertexte. Le modèle hypertexte de WebML est totalement différent des modèles existants. L'approche proposée par WebML consiste dans la définition d'un minimum de concepts pouvant être composés pour obtenir un nombre arbitraire d'applications configurables.

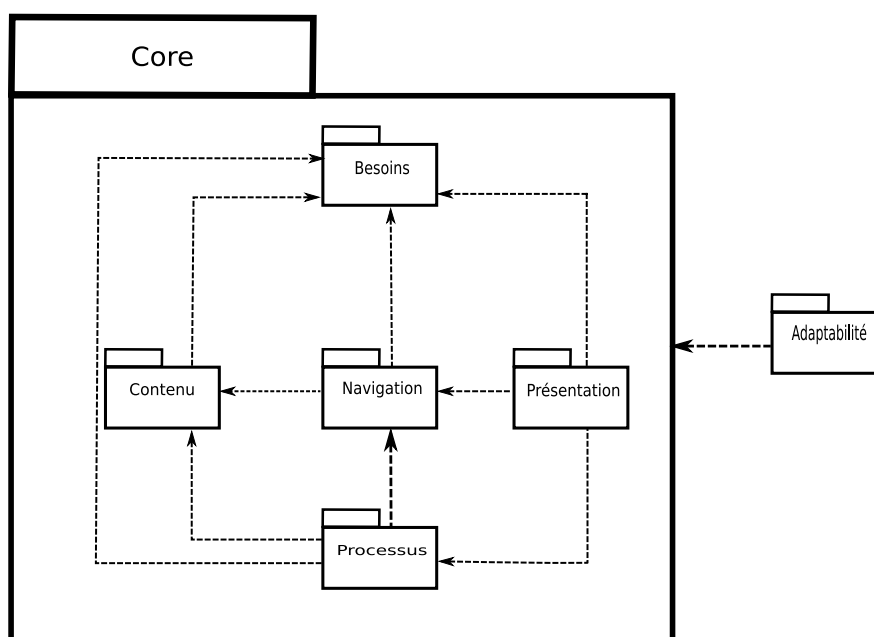


FIG. 3.2 – Aperçu du méta modèle UWE

3.4.1 Historique de WebML

La version originale ou la première génération de de WebML a juste défini un ensemble de primitives pour la représentation en lecture seule des données des applications Web. Elle s'est focalisée sur l'organisation modulaire de l'interface utilisateur, de la navigation et de l'extraction de contenu destiné à être publié dans les pages Web. Dans sa deuxième génération, les concepteurs avaient ajouté un support pour la représentation des actions métier de l'application déclenchées par la navigation de l'utilisateur. Ce support a servi à la gestion du contenu et à la création des mécanismes d'authentification ou d'autorisation. L'introduction du concept de modèle plugeable a transformé WebML en un langage extensible et a donné naissance à sa troisième génération. Il était devenu extensible, les concepteurs pouvaient à présent proposer leurs propres primitives, cela a été étendu WebML à de nouveaux domaines d'applications. Cette transition a mis l'accent sur le rôle des modèles à base de composants et fut la base des extensions à venir de WebML. L'originalité de WebML est plutôt le parallèle existant entre son utilisation dans l'industrie et sa présence dans la recherche académique. Le résultat majeur de la recherche industrielle dans ce sens fut la conception d'un environnement de développement supportant la modélisation d'applications Web avec WebML appelé WebRatio [71].

3.4.2 WebML

WebML est un langage visuel conçu pour spécifier le contenu et la structure des applications Web. Il sert également à l'organisation et la présentation de ce contenu dans un modèle

hypertexte [14] [13]. La Figure 3.3 illustre l'approche WebML. Cette approche comporte différentes phases inspirées du modèle spirale de Boehm [10] et d'autres méthodes de développement des applications classiques et des applications Web [11] [7] [31]. Le processus de développement de WebML est appliqué de manière itérative et incrémentale de manière à ce que ses phases soient répétées et raffinées tout au long de la conception. Le processus rentre alors dans différents cycles produisant ainsi différents prototypes correspondant à des versions partielles de l'application. À chaque itération le prototype de l'application est testé et évalué. Il est ensuite modifié ou amélioré afin de répondre aux besoins et aux spécifications de l'application. La spécification des besoins dans WebML n'est pas spécifique, les concepteurs sont libres d'utiliser les méthodes de leur choix. Ils peuvent les modéliser par des tableaux ou par la combinaison des diagrammes de cas d'utilisation et des diagrammes d'activités UML. Le modèle conceptuel de WebML consiste dans la définition des schémas conceptuels exprimant l'organisation de l'application avec un haut niveau d'abstraction indépendamment de leurs implémentations. Il sert à la modélisation des données et des liens hypertextes. La modélisation des données correspond à l'organisation du contenu identifié lors de la spécification et l'analyse des besoins. Il les modélise et les organise dans des schémas de données plus compréhensibles et plus utilisables par les développeurs. Le modèle hypertexte produit plusieurs vues de l'application à partir du modèle de données, il exprime les compositions du contenu et les liens hypertexte les reliant entre eux. Une description du langage de modélisation des données et des liens hypertexte accompagnée de sa notation formelle est illustrée dans les travaux de Ceri [14] [13].

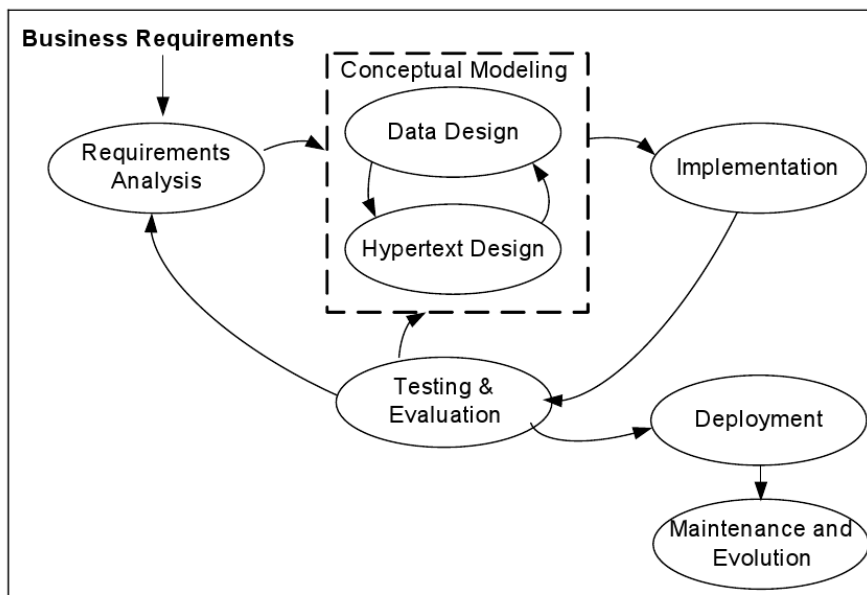


FIG. 3.3 – Processus de développement avec WebML

WebML décrit l'application Web à trois niveaux d'abstraction : le contenu, l'organisation et la présentation du contenu. Le contenu de l'application est modélisé à l'aide d'un modèle E/R équivalent à un diagramme de classe UML simplifié. L'organisation du contenu établit la struc-

ture hypertexte des pages Web. La présentation du contenu spécifie les différentes manières de représenter le site à l'utilisateur, elle associe les pages Web à des styles définis dans un format XML.

3.4.3 Modélisation WebML avec UML

Beaucoup de travaux ont tenté de créer des profils UML pour WebML. Les profils conçus avec la version 1.X d'UML ne correspondaient pas totalement aux spécifications de WebML. En effet, le manque de concepts et le manque de mécanismes propres aux applications Web, l'écart sémantique entre les concepts des applications Web et la structure d'UML, ne pouvaient pas permettre la définition d'un profil adapté à WebML. Avec l'aboutissement de la 2.0 d'UML, la situation a changé. Dans UML 2.0 les évolutions n'étaient pas liées uniquement à la sémantique, même si celle-ci a été modifiée. De nouveaux types de diagrammes destinés à modéliser les structures, les comportements et les activités propres aux applications Web ont été mis au point. Le progrès majeur qu'a connu UML avec sa version 2.0 est la possibilité de définir, grâce à un profil, son propre langage basé sur MOF. L'OMG offre trois approches pour définir un langage dédié DSL, la première solution est de développer son propre méta-modèle. Cela revient à créer un nouveau domaine de langage grâce à un méta-modèle MOF. La syntaxe et la sémantique de ses éléments sont clairement définies et reflètent les caractéristiques propres au domaine choisi. Le souci avec cette approche. C'est que les outils existants qui implémentent UML n'arrivent pas nativement à interpréter ce nouveau langage (édition de modèle à partir du nouveau méta-modèle, compilation du modèle, etc...). Cette approche a été suivie par des langages comme CWM (Common Warehouse Metamodel) ou W2000. La seconde et la troisième approches proposées par l'OMG sont basées sur des extensions d'UML. Ces extensions peuvent être soit légères soit lourdes. Les extensions lourdes sont basées sur la modification du méta-modèle UML avec un changement dans la sémantique de ses éléments, cette méthode implique que ce nouveau modèle ne soit plus interprété par les outils de modélisation basés sur UML. Les extensions légères sont les profils, elles exploitent les mécanismes existants d'extension proposés par UML (stéréotypes, contraintes...etc) pour spécialiser ces méta-classes sans altérer leur sémantique. Les profils UML peuvent imposer des nouvelles restrictions sur les méta-classes mais doivent respecter le méta-modèle UML sans modifier la sémantique de ces éléments de base (classes, associations, propriétés...etc) qui restent les mêmes, des nouvelles contraintes uniquement peuvent leur être ajoutées. *Un sucre syntaxique*⁴ peut aussi être défini dans le profil, en ajoutant des icônes et des symboles correspondant aux nouveaux éléments. L'avantage procuré par ces deux approches se résume dans la capacité des outils de modélisation à leur interprétation. Le meilleur exemple de profil UML pour le Web est décrit dans la section 3.3 [39].

⁴Le sucre syntaxique est une expression imaginée par Peter J. Landin pour désigner les extensions à la syntaxe d'un langage de programmation qui ne modifient pas son expressivité ; le rendent plus agréable à écrire comme à lire. Le sucre syntaxique exprime le fait de donner au programmeur des possibilités d'écriture plus succinctes ou plus proches d'une notation usuelle.

3.5 En résumé

La modélisation des architectures logicielles avec UML a été discutée dans de multiples travaux ([45, 36]). Différents types d'extensions ont été proposés afin de combler le manque d'expressivité du langage UML pour la description de certains aspects liés aux architectures logicielles. Comme dans ces approches, dans cette thèse, nous allons montrer comment nous pouvons utiliser UML pour modéliser des abstractions architecturales basées sur les composants, mais dans un domaine d'application particulier qui est l'ingénierie des logiciels Web.

Dans la littérature, plusieurs travaux ont contribué à la modélisation d'applications Web avec UML. Le travail le plus important dans ce sens est celui de Jim Conallen dans ([16]). L'auteur présente une approche qui utilise de manière particulière UML pour la modélisation des applications Web. Les pages Web sont représentées par des classes stéréotypées, les liens hyper-texte par des associations stéréotypées, les scripts de pages par des opérations, etc. Une sémantique additionnelle a été définie pour les éléments de modélisation UML pour distinguer les aspects côté client et ceux côté serveur (les scripts, par exemple). Alors que l'approche de Conallen ressemble à l'approche présentée dans cette thèse, elle ne traite que les applications Web traditionnelles et non les applications Web riches. L'auteur propose une extension du langage UML à travers les stéréotypes, les valeurs marquées et les contraintes, alors que ce que nous proposons dans notre approche consiste à utiliser UML tel quel. En effet, dans notre cas, la distinction entre les éléments côté serveur et les éléments côté client ne sont pas d'un grand intérêt. Par ailleurs, la représentation hiérarchique des éléments architecturaux n'est pas prise en considération dans ces travaux, alors que la tiers présentation est par nature hiérarchique.

Dans ([29]), Hennicker et Koch ont présenté un profil UML pour le développement d'applications Web. Le travail présenté par ces auteurs est plutôt orienté-processus que centré-produit. En effet, ils proposent une méthode de développement qui débute par la définition des cas d'utilisation comme spécifications des besoins. Ensuite, suivant plusieurs étapes, on peut déduire des modèles de présentation. Ces modèles sont représentés par des classes UML stéréotypés et des objets composés. Ils contiennent, entre autres, des éléments HTML de grain fin. Comme précisé ci-dessus, le travail de Hennicker et Koch est plus orienté-processus. Il montre comment on peut obtenir des applications Web à partir de spécifications de besoins. Dans notre travail, nous nous focalisons sur la définition des modèles de présentation qu'ils ont introduits. Nous ne nous intéressons pas à la méthode utilisée pour les obtenir. De plus, nous traitons les éléments hiérarchiques dans les applications Web comme leurs éléments hiérarchiques de présentation représentés par des objets composites. Cependant, les éléments de présentation que nous traitons sont interactifs et collaboratifs (comme les formulaires et les objets de formulaires). Leurs éléments de présentation sont particulièrement liés à la navigation Web (des pages HTML contenant du texte et des images).

Dans ([25]), les auteurs proposent une approche pour les utilisateurs finaux afin de développer des applications Web utilisant des composants. Cette approche se focalise sur la construction par les développeurs de composants génériques de haut niveau, qui peuvent être réutilisés par les utilisateurs finaux, assemblés ensuite, pour être déployés et finalement exécutés. Dans cette approche, les composants varient des générateurs de formulaires aux moteurs de requêtes de bases de données. La préoccupation s'étend donc à des applications Web entières ; tous les tiers et leurs environnements d'exécution sont ciblés. Dans le travail que je propose, les composants

sont spécifiques au tiers présentation. Je me focalise sur la modélisation d'architectures d'applications basées sur des composants de librairie collaboratifs. Mon approche est centrée-produit et non orientée-organisation comme dans ([25]).

Troisième partie

Contribution de la thèse

Chapitre 4

Un protocole de migration vers un processus de développement à base de composants

Clemens Szyperski [66] définit un composant logiciel comme : *a unit of composition that can be deployed independently and is subject to composition by a third party*. Bertrand Meyer [Mey00], quant à lui, définit les composants comme des logiciels orientés clients. Un composant est un élément de programme utilisable par d'autres éléments de programme. Ces derniers sont qualifiés de client pour le composant considéré. L'objectif de l'approche par composant est d'augmenter autant que possible les opportunités de réutilisation et en conséquence de diminuer les coûts, les délais et d'augmenter la qualité. Pour ce faire, un composant doit être aussi indépendant que possible du contexte d'exécution et offrir des services suffisamment généraux.

Un composant comporte deux parties : une implantation constituée du code exécutable, qui met en oeuvre les services proposés et une spécification qui décrit les interfaces (avec leur type) et les propriétés du composant (contraintes, propriétés non fonctionnelles). Cette distinction claire a pour objectif le masquage d'information ; diminuer au maximum le couplage entre les composants en limitant les dépendances aux seuls services offerts et non à la façon dont ces services sont réalisés. Les interfaces définissent les modalités d'interaction d'un composant avec son environnement. Ils expriment à la fois les services requis et les services fournis par le composant. Certains modèles de composant donnent également la possibilité à d'exporter des attributs permettant de le configurer (lors de son déploiement et/ou en cours d'exécution). La spécification est utilisée au niveau modélisation pour construire l'application ; elle sert aussi lors du déploiement du composant : par exemple un descripteur de déploiement est une spécification qui indique la façon dont le composant doit être déployé (les services dont il a besoin, son cycle de vie, etc.).

L'usage par un composant d'autres composants est rendu possible par l'existence d'un mécanisme de composition logicielle entre composants. La composition peut se faire au moyen de connecteurs. Ces derniers modélisent de manière explicite les interactions entre les composants en définissant les règles qui gouvernent ces interactions : un appel de procédure, l'accès à une variable partagée, ou un protocole d'accès à des bases de données avec un système de

gestion de transactions, etc. Comme pour un composant, un connecteur comporte deux parties : une interface et une implantation.

Il existe diverses formes de composants répondant à différents besoins. Par exemple, certains modèles de composant (e.g. EJB [65], CCM [2]) fournissent des supports d'exécution prenant en charge diverses propriétés non-fonctionnelles : persistance, transactions, protection, duplication, etc. Ces modèles sont destinés au développement d'applications nécessitant la présence de ces services comme les applications Web. D'autres modèles ont pour but de faciliter le déploiement d'applications patrimoniales. C'est notamment le cas de Jiazzi [59] et OSGi [52] qui permettent de gérer les dépendances entre classes Java patrimoniales via une notion de paquetage étendue. Ces modèles ne considèrent pas les composants comme des entités en cours d'exécution. Certains modèles (e.g. DCUP [24], Fractal [12], OpenCOM [43]) sont plutôt orientés vers la conception de systèmes dynamiquement configurables ; ces modèles possèdent des caractéristiques évoluées en termes de structuration des applications et offrent la possibilité d'associer des contrôleurs aux composants pour pouvoir procéder à leur reconfiguration en cours d'exécution.

Les composants sont aujourd'hui largement utilisés pour développer aussi bien des applications, que des intergiciels, ou encore des systèmes d'exploitation.

4.1 Un protocole de migration

Pour répondre à la problématique de migration citée dans 1.3 et face au constat de l'absence de méthodes clairement définies dans la littérature, j'ai défini un protocole ad hoc. Ce protocole s'appuie sur une étape intermédiaire et transitoire qui préserve les compétences et le code existant de l'entreprise. Je propose le temps de cette étape d'embarquer le code des applications dans des pseudo-composants possédant des interfaces requises et d'autres fournies. Ce choix d'un passage progressif d'un paradigme objet ou impératif à un paradigme composant est d'ailleurs fortement recommandé par [SYP 02]. L'idée défendue est que la pratique des principaux concepts du paradigme composant dans le cadre rassurant et connu du paradigme source habituel participe non seulement à une prise de recul salutaire lors des travaux réalisés mais également à une assimilation progressive de ces concepts. Une raison supplémentaire m'a conduit à préconiser cette étape intermédiaire : le souhait de pouvoir reporter à plus tard le choix d'une technologie de composants précise (FRACTAL [BRU 04], EJB [SUN 03], ...) en attendant leur maturation.

Sur le plan organisationnel, lors de cette étape, le développement est organisé de manière à séparer les développeurs en deux équipes. La première se destine aux développements des modules (développement pour la réutilisation). Elle se nomme *Component Development Team* (CDT). La seconde se charge du développement des applications en exploitant les modules déjà développés (développement par la réutilisation). Elle s'appelle *Application Development Team* (ADT). La Figure 4.1 illustre cette organisation.

Une plate-forme de développement supportant cette organisation a été mise en place au sein de l'entreprise. Elle propose un espace de travail pour les développeurs comprenant un gestionnaire de versions de fichiers pour le travail d'équipe et un espace dédié à la mise à jour distante des applications des clients et de leurs pseudo-composants. Chaque pseudo-composant

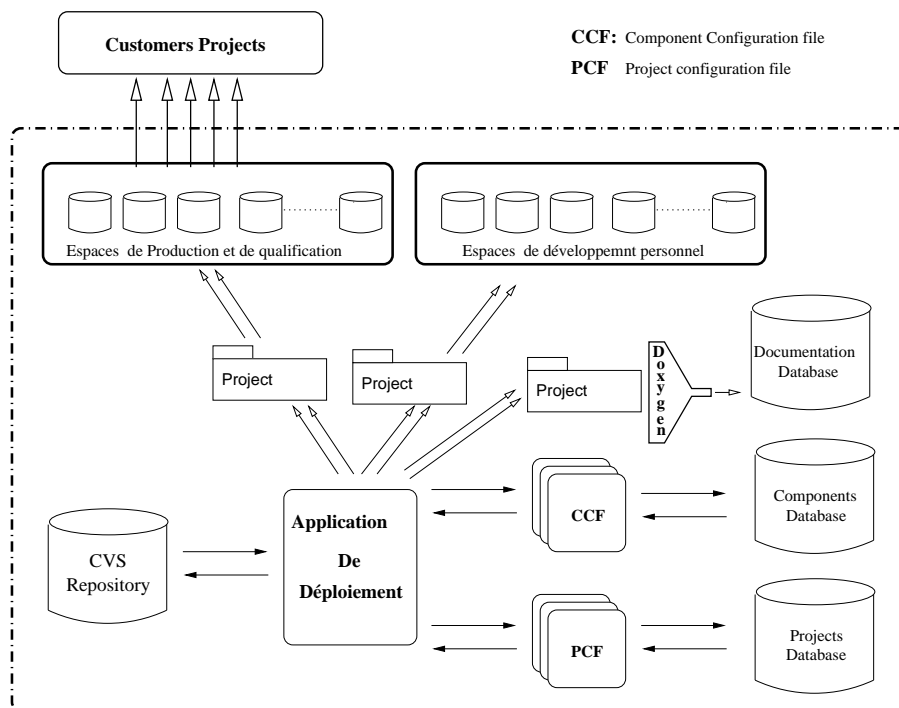


FIG. 4.1 – La plate-forme de développement

est défini par un *Configuration Component File* (CCF) écrit en XML et stocké dans une base ad hoc. Les CCFs décrivent les interfaces requises, offertes et de contrôle. Pour plus de flexibilité, le CCF ne doit pas contenir de code source mais référence la pièce logicielle correspondante présente dans le dépôt CVS. Dans cette approche, le CCF est considéré comme l'enveloppe d'un pseudo-composant. Chaque pseudo-composant embarque sa documentation de code et ses spécifications fonctionnelles. Pour des raisons de clarté, la documentation du composant n'est pas représentée dans la Figure 4.1. Le pseudo-composant est l'unité de base à partir de laquelle sont construites par agrégation les applications. Chaque application est décrite dans un *Project Configuration File* (PCF) et est stockée dans une base ad hoc de projets (dépôt de projets).

4.2 Le processus de développement

L'exploitation de cette plate forme est facilitée par une application de déploiement et d'assemblage de composants. D'un côté, elle permet aux développeurs de la CDT d'accéder à la liste des composants et leur permet d'en ajouter. D'un autre côté, elle permet à ceux de l'ADT de créer de nouveaux projets et de choisir leurs composants adéquats. Pour les développeurs de l'ADT cette plate-forme commune est vue comme étant un accès en lecture seul aux composants sur étagère. Les tâches des développeurs de l'ADT sont la personnalisation et le respect des spécifications du projet (voir Figure 4.2). Celles des développeurs de la CDT sont

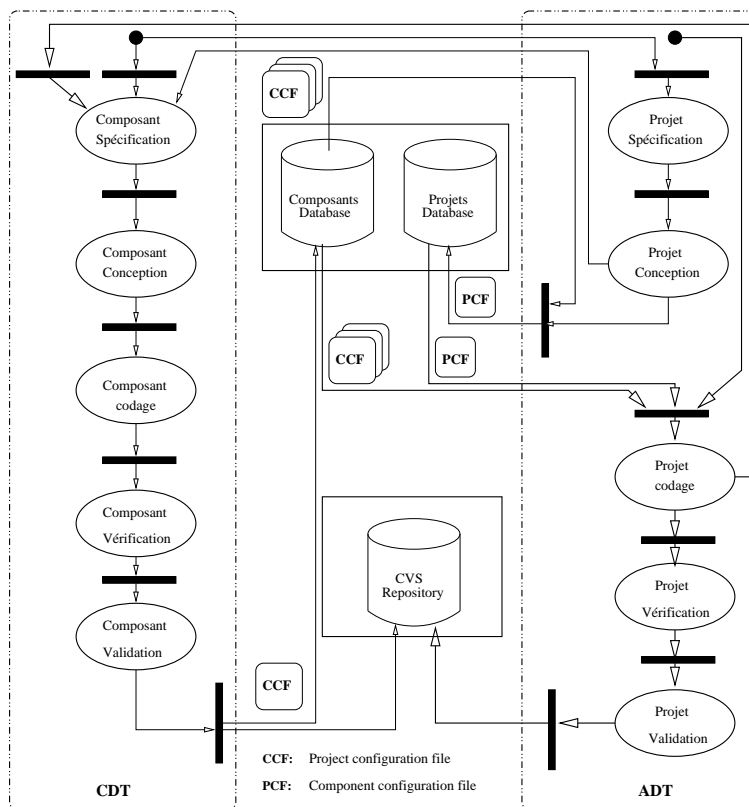


FIG. 4.2 – Architecture de développement

l'implémentation et le respect des spécifications des composants.

L'organisation du développement et la plate forme s'associent pour former un processus de développement très efficace :

- Le client et l'entreprise valident les spécifications à la réception du cahier des charges.
- Un chef de projet est désigné pour le piloter.
- Il forme deux équipes de développeurs, une CDT et l'autre de Production.
- Il crée le projet via l'application de déploiement.
- Il choisit les composants nécessaires et rédige les spécifications des nouveaux composants.
- Il notifie, enfin, les développeurs concernés par le nouveau projet.

La mission des membres de la CDT est le développement des composants non existants ainsi que leurs intégration dans la base. Les membres de la Production déploieront le projet dans leurs espaces de travail et devront satisfaire les spécifications de l'application. Les développeurs de production ont la possibilité, si nécessaire, de rédiger les spécifications des nouveaux composants. Les deux équipes respectent les mêmes étapes de développement, des spécifications à la validation et aux tests. Ce ne sont pas les composants que l'on trouve sur l'étagère, mais leurs enveloppes. Les outils d'assemblages utilisent ces enveloppes pour produire une application.

```
+ sit21/                                # racine du projet
+ sit_1_ALK/
  + api/                                # fichiers d'inclusion de la nouvelle API
  + classes/                             # modules de classes génériques
    + appli/                             # classes génériques de base du framework
    + form/                               # classes génériques des contrôles Html
    + ...
  + lib/
  + libconf/                             # fichiers pour les configurations
  + media/
  + scripts/                             # modules spécifiques aux sous applications
    + actu/                              # module de la sous application Actualités
    + admin/                             # module de la sous application Administration
    + fdoc/                              # module de la sous application Fonds Documentaires
    + sig_admin/                         # module d'administration de la sous application SIG
    + ...
  + services/
  + styles/                              # fichiers de styles CSS
  + upload/                              # répertoire pour l'upload de fichiers
```

FIG. 4.3 – Un exemple d'une application Alkante

4.3 Illustration du PCF et du CCF

Je présenterai dans cette section la structure du PCF et d'un CCF correspondant à une sous application. Des captures écrans de l'application de gestion des pseudo-composants et de leur déploiement seront également présentés pour illustrer le processus de developpemnt.

Reprenons l'exemple d'une application Alkante qui est composée de plusieurs sous applications. Ces sous applications sont réparties en deux catégories : les pseudo-composants

```

+ scripts/
+ actu/                                     # le module d'actualités
+   _admin/
+     20_grr.sql                             # scripts sql (création de tables...)
+   api/
+     gen_con.php
+ classes/
+   alkappliactu.class.php                   # classe d'application
+   queryactu.class.php                     # classe de requêtes sql
+   queryactu_action.class.php              # classe de requêtes sql
+   alkdataformreserver.class.php           # classe de gestion de formulaire
+ lib/
+   00_actu.php                              # script d'entrée de l'application
+   01_reserver_form.php                     # script d'affichage d'un formulaire

```

FIG. 4.4 – Un exemple d'une sous application Alkante

(sous-application) génériques réutilisés dans chaque application (préfixés de *mcg_*), et ceux spécifiques à une application (préfixés de *map_*). L'organisation d'une application type est présentée sur la Figure 4.3. Une sous application est également présentée dans la Figure 4.4.

La Figure 4.5 représente le PCF de l'application présentée dans la Figure 4.3. Les PCFs et Les CCFs possèdent la même structure. Dans l'architecture proposée, une sous application est aussi représentée par un PCF si celle-ci est livrée toute seule. La Figure 4.5 représente l'application SIT21 *Système d'information territoriale du département 21*, elle est constituée de plusieurs sous applications dont les sous applications d'actualités et d'administration cartographique. Elles correspondent aux fichiers CCF *mcg_actu_1_0_1.xml* et *mcg_sigadmin_2_2_1.xml*. La sous application d'actualités est un composant composite de l'application SIT21 d'où la balise `<controller desc="composite"/>`. Ce composant possède deux types d'interfaces : des interfaces fournies spécifiées par le rôle *Server* comme par exemple *"OutputHtmlPage"* ; Et des interfaces requises spécifiées par le rôle *Cleint* comme par exemple *"InputRSS"*. La première fournie aux autres composants un flux de type HTML et la deuxième requière d'un composant un flux de type RSS. Les connecteurs d'interfaces sont représentés par les balises `binding`, comme par exemple `<binding client="this.InputRSS" server="ReadRSSGoogle.OutputRSS"/>`. Ce connecteur relie le composant d'actualités au composant *ReadRssGoogle* qui lui fournie des flux RSS provenant de google actualités. L'application intranet de déploiement de la plateforme utilise le PCF et les CCFs associés pour générer le code de l'application SIT21 par extraction CVS.

Deux captures écran (les figures 4.6 et 4.7) de l'application de déploiement illustrent respectivement, la génération d'un PCF à partir de composants (CCFs) existants et un aperçu de la liste des composants disponibles pour sa génération. Dans la Figures 4.7 nous remarquons que les listes des composants disponibles sont regroupés selon le langage avec lequel ils sont implémentés. Dans notre cas, PHPDEV pour les composants écrits en PHP4, ASPDEV pour ceux écrits en ASP, LINUXDEV pour ceux écrit avec de langages de script (python, shell, etc.) ou en C++,JAVA ou JSP, WINDEV pour ceux écrits dans des langages supportés par des plate-formes Windows (c#, VisualBasic, etc) et enfin PHP5DEV pour ceux écrits en PHP5. Ces regroupement correspondent aux différents dépôts CVS. Les icônes représentées quant à elles par des horloges qui correspondent aux temps passé pour le développement de chaque

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE definition PUBLIC "-//alkante.com//DTD Alkante//EN"
"classpath://org/alkanet/alkoweb/adl/xml/standard.dtd">
<definition extends="com.alkanet.alkoweb.SIT1_ALK23658" name="SIT_1_ALK23658">
<component name="SIT21" pcf_file="pcf_sit_21.xml">
<component name="SIT_1_ALK">
<component name="Actu" ccf_file="mcg_actu_1_0_1.xml">
<interface signature="" role="client" name="InputRSS"/>
<interface signature="" role="client" name="InputAlkXML"/>
<interface signature="" role="client" name="InputJSON"/>
...
<interface signature="" role="server" name="OutputHtmlPage"/>
<interface signature="" role="server" name="OutputHtmlFrame"/>
<interface signature="" role="server" name="OuputHtmlSheet"/>
<interface signature="" role="server" name="OuputHtmlSubSheet"/>

<component name="ReadRSSGoogle" ccf_file="mcg_google_rss_1_1_1.xml">
<interface signature="" role="SERVER" name="OutputRSS"/>
....
</component>

<binding client="this.InputRSS" server="ReadRSSGoogle.OutputRSS"/>
<binding client="this.InputXML" server="ReadRSSDatabe.OutputXML"/>
...
<controller desc="composite"/>
</component>

<component name="sig_admin" ccf_file="mcg_sigadmin_2_2_1.xml">
...
<controller desc="composite"/>
</component>
...
</component>
</component>

```

FIG. 4.5 – Exemple d'un PCF

Modules liés au projet pour son extraction:
Gestion du projet associé à ces modules
Vous pouvez ajouter ici de nouveaux modules ou modifier les liens existants.

Projet : CRBN
 Module racine du projet :
 Path racine d'extraction :

29 modules pour l'extraction du projet			Supprimer
	Module - Version	Path/nom d'extraction	Ajouter
▼	43 mcg_db_v1 - db_1_1_0	CRBN/classes/db	Modifier
▲▼	45 mcg_dberr_v1 - dberr_1_0_0	CRBN/classes/err	Modifier
▲▼	48 mcg_html2pdf_v1 - html2pdf_2_0_0	CRBN/classes/html2pdf	Modifier
▲▼	50 mcg_mail_v1 - mail_1_0_0	CRBN/classes/mail	Modifier
▲▼	49 mcg_html_v1 - html_1_1_0	CRBN/classes/form	Modifier
▲▼	55 mlp_alkanet_lib_v1 - lib_1_1_0	CRBN/lib	Modifier
▲▼	31 map_sialke_fdoc_v1 - fdoc_1_3_0	CRBN/scripts/fdoc_01	Modifier
▲▼	22 map_sialke_annuaire_v1 - annu_2_1_0	CRBN/scripts/annu	Modifier
▲▼	28 map_sialke_espace_v1 - espace_1_1_0	CRBN/scripts/espace	Modifier
▲▼	53 mcp_sialke_appli_v1 - appli_1_1_0	CRBN/classes/appli	Modifier
▲▼	217 map_alkanet_sigadmin_v1 - sigadmin_1_1_0	CRBN/scripts/sig_admin	Modifier
▲▼	214 mcg_smarty_v2 - smarty_2_6_0	CRBN/classes/template	Modifier

FIG. 4.6 – Application de déploiement des composants d'Alkante

composant.






























Projets	Modules	CheckOut	Déployer	Suivi projets	Suivi déploiements	Projets interne	Créer projet interne	Modèles
Liste des modules dans les différents dépôts CVS : <i>Cliquez sur le nom du module pour consulter le suivi de son utilisation</i>								
PHPDEV	ASPDEV	WINDEV	LINUXDEV	PHP5DEV				
map_alkanet_actu_v1 		mpr_accidents_v2 	mpr_maplink_v1 	mpr_sitmi_v1 				
map_alkanet_annonce_v1 		mpr_incendies_v2 	mpr_maplink_server_v1 	mcq_pattern_v3 				
map_alkanet_cvs_v1 		mpr_accidents_V2 	mpr_maplink_qvsig_v1 	mcq_excel_v3 				
map_alkanet_editeur_v1 		mcq_plugin 	mpr_mondrian_v1 	mcq_form1_v3 				
map_alkanet_faqs_v1 			mpr_alksaveto_v1 	mcq_html2pdf_v3 				
map_alkanet_fichecom_v1 				mcq_template_v3 				
map_alkanet_form_v1 				mip_alkanet_lib_v3 				
map_alkanet_qedit_v1 				map_alkanet_annu_v3 				
map_alkanet_glos_v1 				map_alkanet_espace_v3 				
map_alkanet_lien_v1 				map_alkanet_atlas_v3 				

FIG. 4.7 – Liste de quelques composants Alkante

4.4 Expérimentation de l'approche

Pour l'expérimentation de l'approche proposée, j'ai pris le cas d'une application de chez Alkante. Un problème spécifique à l'ingénierie Web concerne la décomposition de la livraison des applications. Pour être compétitive, Alkante, comme les autres entreprises, découpe ses applications en différentes parties. Ce découpage est généralement lié aux contraintes et aux besoins de ses clients. Les différentes parties sont livrées selon un programme ou un agenda mis en place avant le début du projet. C'est à dire, qu'au lieu de livrer à ses client une application complète à échéance, Alkante livre une version minimum. Cette version possède des sous application existantes. Elle sera ensuite enrichi au fur et à mesure par de nouvelles sous applications.

La figure 4.8 illustre la problématique générée par la décomposition de la livraison d'une application. Elle montre la livraison d'une des applications à quatre clients (C1, C2, C3, C4) sur deux ans. Cette application est assemblée à partir de différentes sous applications. Certaines existent déjà alors que d'autres seront développées durant cette période. A la première livraison toutes les applications s'appuient sur les mêmes sous applications (A, B, C et D). Les sous applications A,B,C et D représentent respectivement, l'authentification, l'exploration de dossiers, la navigation cartographique et la gestion des utilisateurs.

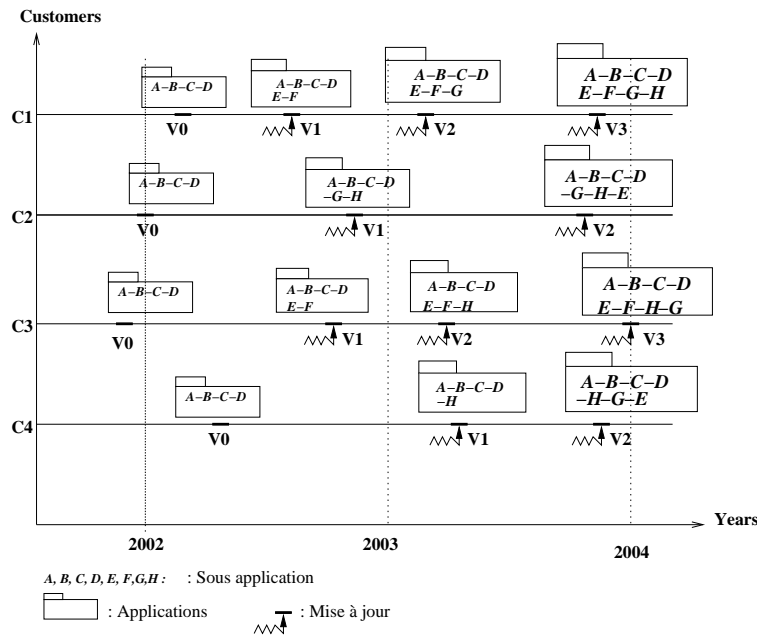


FIG. 4.8 – Programme de livraison des sous applications

4.4.1 Coûts de développement

Comme illustré dans le Tableau 4.1, les coûts les plus importants apparaissent lors du premier développement des sous applications (C3xV0, C1xV1, etc). Lorsque l'on réutilise une sous application existante dans une nouvelle application (ex ; C3xV1), les coûts sont moindres mais non nuls. En effet la réutilisation d'une sous application dans une nouvelle livraison engendre deux types de coûts : l'un liée à l'adaptation et l'autre à l'intégration. Nous les appellerons « coûts d'assemblages ».

	V0	V1	V2	V3	TOTAL
C1	63	1496	168	336	2081
C2	58	495	932		1485
C3	1910	38	327	183	2458
C4	49	321	1150		1520

TAB. 4.1 – Coûts des différentes Versions

Les mesures faites ont permis de calculer avec précision les coûts de développement de nouvelles sous applications, comme le montre le Tableau 4.2.

J'ai utilisé les éléments du Tableau 4.1 et du Tableau 4.2 pour extraire les coûts d'assemblages. Le Tableau 4.3 les illustre pour chacune des versions de chaque client. Nous remarquons que les coûts augmentent avec le nombre de sous applications ajoutées. Par exemple :

- Pour le client C1 à la version V1, les sous applications E et F (un gestionnaire de données

Sous applica- tions	A	B	C	D	E	F	G	H
Coûts (heures)	90	140	750	930	900	540	115	300

TAB. 4.2 – Coûts réels liés aux développement des sous applications

	V1	V2	V3	Total
C1	56	31	36	123
C2	40	32		72
C3	38	27	28	93
C4	21	95		116
TOTAL (Heures)	155	185	64	404

TAB. 4.3 – Coûts d'assemblages.

et un second type de navigateur cartographique) ont été ajoutées et ont coûtées 56 heures de développement, alors que la sous application G (générateur de formulaire) lui a coûtée 32 heures à la version V2.

- Pour le client C4 à la version V1, la sous application H (courrier et lettre de diffusion) a été ajoutée et a coûté 21 heures de développement, alors que les sous applications G et E lui ont coûtées 95 heures à la version V2.

4.4.2 Discussion

Dans cette étude de coûts, nous n'avons pas illustré ceux liés à la maintenance. Nous les étudierons dans le chapitre dédié à l'évolution des applications Web (cf chapitre évolution). Ici, le travail a été centré sur les coûts d'assemblage. Comme nous pouvons le constater dans le tableau 4.8 et 4.1, les clients C1 et C3 possèdent la même application (V3), mais assemblée dans un ordre différent. À la différence d'ordre d'assemblage correspond une différence significative du coût global, soit dans ce cas 337 heures. Quelques entreprises tendent par expérience à réduire leurs coûts d'assemblage, en livrant leurs sous applications dans un même ordre (le moins coûteux). Mais n'importe quelle modification du calendrier de livraison nécessitera une remodelisation et engendrera de nouveaux coûts supplémentaires. Dans notre étude, les coûts d'assemblage sont équivalents à ceux du développement d'une nouvelle sous application. A ce stade, une architecture traditionnelle ne peut pas nous aider à minimiser ces coûts. En accord avec les études d'Ommering [OMM 02] et de Shilagui [RAU 03], les architectures à base de composants permettent de créer une variété de produit compliqués en un minimum de temps. Par exemple, Nokia maintient une large librairie de composants qu'elle utilise pour produire sa famille de téléphone portable [JAN 05]. Koala, le modèle composant de Phillips, est aussi un exemple de la minimisation de coûts d'assemblages [OMM 00].

Cependant, comment valider les bénéfices de cette architecture composants transitoire ? Le

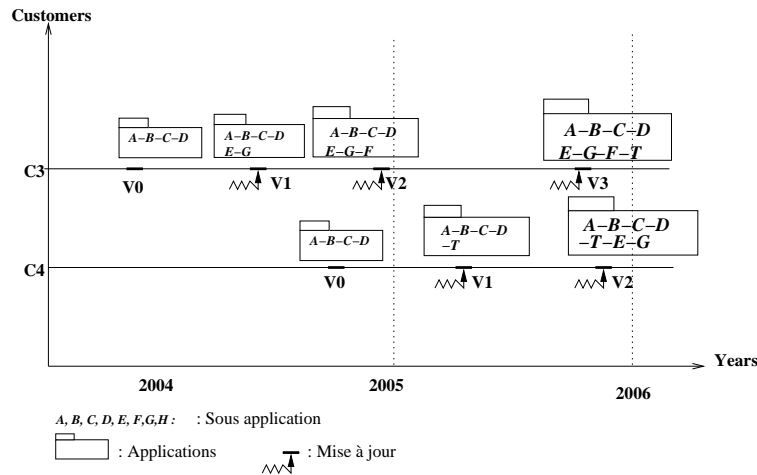


FIG. 4.9 – Livraison de sous applications après la transition

plus simple est de réitérer le même processus de développement pour les mêmes clients avec les mêmes développeurs. Mais cette solution n'est pas économiquement acceptable pour des entreprises et surtout pas pour des PME comme Alkante. Pour l'évaluation des nouveaux coûts d'assemblages nous avons répété notre même étude pour deux autres clients C5 et C6 comme le montre la figure 4.9. La même méthode nous a permis de mesurer les coûts de composition des sous applications, illustrés dans le tableau 4.4, et nous a permis de mesurer ceux de la plateforme évalués à 1260 heures. Cette composition concerne uniquement les huit sous applications et non ceux qu'elles encapsulent. Nous avons simplement défini une membrane autour de ces sous applications et nous les avons rendu interdépendantes.

Sous applica- tions	A	B	C	D	E	F	G	H
Coûts(heures)	12	40	68	69	54	46	23	16

TAB. 4.4 – Les coûts de compositions des sous applications.

	V1	V2	V3	Total
C5	16	11	14	41
C6	14	18		32
TOTAL				73

TAB. 4.5 – Les coûts d'assemblages

4.5 Conclusion

Après avoir attesté et certifié la minimisation des coûts de développement avec la plate forme transitoire, nous sommes à présent conscients des bénéfices que l'on pourra faire grâce aux architectures à bases de composants. Parmi ces bénéfices, nous avons remarqué la familiarité qu'ont acquis nos développeurs avec les concepts des architectures à bases de composants, rien qu'en utilisant cette architecture transitoire. Cette migration doit permettre dans un premier temps a renforcé et optimisé le processus de développement. Cette architecture permet un développement avec un minimum de coûts et permet la livraison de différentes applications en un temps minimum. Elle permet également une migration en douceur vers une architecture à base de composants et permet aux développeurs de vite acquérir les concepts de telles architectures. Elle les rend surtout capable d'isoler les parties composables du code existant.

Chapitre 5

AlkoWeb, une plate-forme pour le développement Web riche à base de composants

Partant du fait qu'au chapitre précédent, le choix d'un modèle de composants a été retardé volontairement. Il faut maintenant, choisir un modèle de composants qui correspond aux critères de modélisation propres aux nouvelles applications Web. Ce modèle, doit être conforme au processus de développement à base de composant mis en place précédemment et doit être implémenté par un outil intégrable à cet environnement de développement. J'introduis ce chapitre par la présentation des principaux modèles de composants (industriels et académiques) existants. Dans cette présentation, j'argumente mon choix d'UML2.0 comme modèle pour la modélisation des applications Web et je fais une bref description de son modèle de composants. Cette description est également appuyé par un métamodèle de composants UML2.0 simplifié. Je présente dans la section 5.3, une interprétation et ces éléments architecturaux d'UML2.0 pour la modélisation des applications Web appelée AlkoWeb. Cette interprétation est illustrée par un exemple dans la section 5.4. Un mécanisme basé sur le langage OCL est présenté dans la section 5.6. Ce mécanisme est dédié à vérifier la conformité de l'assemblage des composants. Je conclue ce chapitre par la section 5.7. Je présente dans cette section l'implémentation d'AlkoWeb [35] par un outil basé sur la plate forme eclipse appelé AlkoWeb-builder [34].

5.1 À La recherche d'un modèle de composants logiciels

Afin de modéliser l'architecture d'une application Web de nouvelle génération, il fallait être capable de modéliser toutes ses couches, sa couche présentation, sa couche métier, sa couche de données et éventuellement d'autres couches. Dans le cas spécifique des applications Web de dernière génération, les couche présentation et métier sont très liées. Le modèle de composants recherché doit alors être hiérarchique et permettre la navigation d'une couche à une autre. Il doit également, permettre la modélisation des liaisons entre chacune d'elles. La couche présentation des applications Web est de plus en plus dynamique et génère un nombre importants d'évènements(la plupart sont inter-éléments). Elle est également par

définition hiérarchique, que ce soit d'un point de vue conceptuel (Les éléments Web définies par les standard du W3C¹) ou d'un point de vue organisationnel (accès à l'information). La partie présentation répond à des normes rigoureuses définies par des spécifications du W3C. Ces normes définissent des règles d'accès et de manipulations des éléments de l'interface (DOM). Elles définissent également le modèle d'évènements générées par ces éléments.

Le modèle de composants recherché doit aussi rendre possible la modélisation de l'aspect évènement de cette couche, les éléments, les objets et les règles définies par les spécifications du W3C.

Il existe actuellement sur le marché des composants logiciels différents modèles accompagnés de leurs implémentations. Des implémentations et des modèles de composants industriels (EJB, .NET, CCM, etc) et des modèles non encore industrialisés (ou pourvus d'implémentations) développés par différentes équipes de recherche dans le monde (Fractal, OpenCom, etc.).

Les implémentations et les modèles que l'on trouve sur le marché sont généralement destinés à des domaines spécifiques. Le choix par une entreprise d'un modèle ou d'un autre dépend du contexte et du type de l'application à développer. Chaque modèle et chaque implémentation possède ses avantages et ses inconvénients. Par exemple, le modèle industriel CCM sert à la conception d'applications ayant besoin de divers services à l'exécution. L'avantage de ce modèle est qu'il simplifie le développement d'applications. La complexité de la prise en charge des services non fonctionnels est masquée par des interfaces simplifiées. Elle est prise en charge par les conteneurs et par les parties du composant qui sont générées par les compilateurs. Un des apports du modèle CCM est d'avoir proposé des outils nécessaires aux différentes étapes du cycle de vie d'une application. Malheureusement, CCM a un certain nombre de limites, la plupart sont citées dans [44]. Tout conteneur doit prendre en charge un certain nombre de propriétés non fonctionnelles. Cette dépendance vis-à-vis de services systèmes impose que les composants CCM soient exécutés sur des équipements ayant une certaine puissance. Le modèle CCM possède également un modèle de composition limité, il nous est impossible de créer des composants composites ou des composants partagés, essentiels pour notre approche. Les capacités d'administration sont limitées, les seules possibilités de reconfiguration sont fournies par les interfaces d'introspection et de gestion de ports implantées par les conteneurs. Une autre limitation du modèle CCM est que les outils fournis permettent uniquement de générer et déployer du code. Il n'existe aucun outil de vérification des structures déployées. CCM possède un niveau de configuration très faible, les conteneurs sont monolithiques et ne supportent qu'un ensemble borné de propriétés non-fonctionnelles.

De la même manière que le modèle CCM, le modèle industriel EJB a été conçu pour la conception des applications métier nécessitant des services non fonctionnels. L'avantage de ce modèle se trouve dans la facilité de développement rendu possible par la prise en charge d'un certain nombre de propriétés non-fonctionnelles par le conteneur. Le développeur se consacre qu'à l'écriture du code métier. Le modèle EJB souffre des mêmes limites que le modèle CCM, sa dépendance vis-à-vis des services système qui nécessitent de déployer le conteneur sur des équipements puissants, son modèle de composition est limité, il possède un faible niveau de configuration. Le modèle de composition EJB est moins évolué que celui du modèle CCM,

¹Le World Wide Web Consortium, abrégé par le sigle W3C, est un organisme de normalisation et un consortium chargé de promouvoir la compatibilité des technologies du World Wide Web telles que HTML, XHTML, XML, RDF, CSS, PNG, SVG et SOAP.

il ne permet pas, par exemple, de décrire de façon explicite les interfaces requises par les composants.

Un exemple de modèle composants non encore industriel proche de nos besoins est le modèle Fractal. Fractal est un modèle particulièrement flexible. Il définit un modèle de composition étendu dans lequel il est possible de créer des architectures hiérarchiques avec partage de composants. Par ailleurs, Fractal permet d'associer un méta-niveau arbitrairement complexe à chacun des composants. De fait, il est possible de configurer dynamiquement les architectures déployées. D'autre part, Fractal n'impose aucun service de base. Il est possible d'instancier des composants avec des contrôleurs minimaux, ayant une empreinte mémoire faible et très peu d'impacts sur les temps d'exécution, mais n'offrant pas de fonctions d'introspection et de reconfiguration. Il est également possible d'instancier des composants avec des contrôleurs plus évolués, permettant l'introspection et la reconfiguration dynamique, mais ayant un impact plus important sur les performances des applications. Fractal fournit un langage de description d'architectures extensible permettant de déployer des applications réparties. En revanche, Fractal ne fournit aucun outil pour procéder à la vérification des architectures à déployer.

A ce stade, ma recherche du modèle composants pouvant répondre aux problématiques de développements d'applications Web de nouvelle génération fut infructueuse. Proposer alors un nouveau modèle de composant pour le développement d'applications Web n'était pas une solution envisageable. Au lieu d'introduire un nouveau langage de modélisation architecturale, j'ai choisi de m'appuyer sur un standard existant et largement adopté : UML2.0. J'ai trouvé dans les spécifications du diagramme de composants de la version 2.0 d'UML toutes les abstractions dont j'ai besoin pour la modélisation d'applications Web riches à l'aide d'entités hiérarchiques. Il s'agit alors d'une interprétation du méta-modèle de composants UML2.0 orientée application Web.

5.2 Le modèle de composant UML2.0

Le modèle de composant UML2.0 est très récent. Il est issu de UML qui était déjà, dans sa version 1.5, un langage de modélisation largement utilisé dans l'industrie. De plus, la notation présente l'intérêt de pouvoir être modifiée pour intégrer des modèles abstraits spécifiques. C'est l'approche qui est en général effectuée dans les différents projets ayant besoin d'une modélisation particulière du problème traité. Il est à noter que parallèlement à la version 1.0 de UML, la méthodologie Catalysis [DW98] a vu le jour en 1998 ; ses auteurs, Desmond D'Souza et Alan Wills, ont d'ailleurs participé à la formalisation de UML1.0. Catalysis est une méthode de conception d'applications à base de composants abstraits ; elle fournit un processus de développement d'applications plus complet, et plus précis que UML. La méthodologie est basée sur le principe de raffinement, de types (pour spécifier la sémantique d'un objet), et de collaborations entre objets (pour spécifier les interactions entre ces objets) ; l'utilisation de canevas (ou frameworks) facilite la conception par patron d'interaction. La méthode Catalysis permet de typer les interfaces des composants, mais est beaucoup plus riche, car elle propose tout un modèle de conception.

Les composants étant un aspect de plus en plus important, notamment avec les dernières plates-formes réparties, l'OMG a doté la nouvelle version de son langage de modélisation

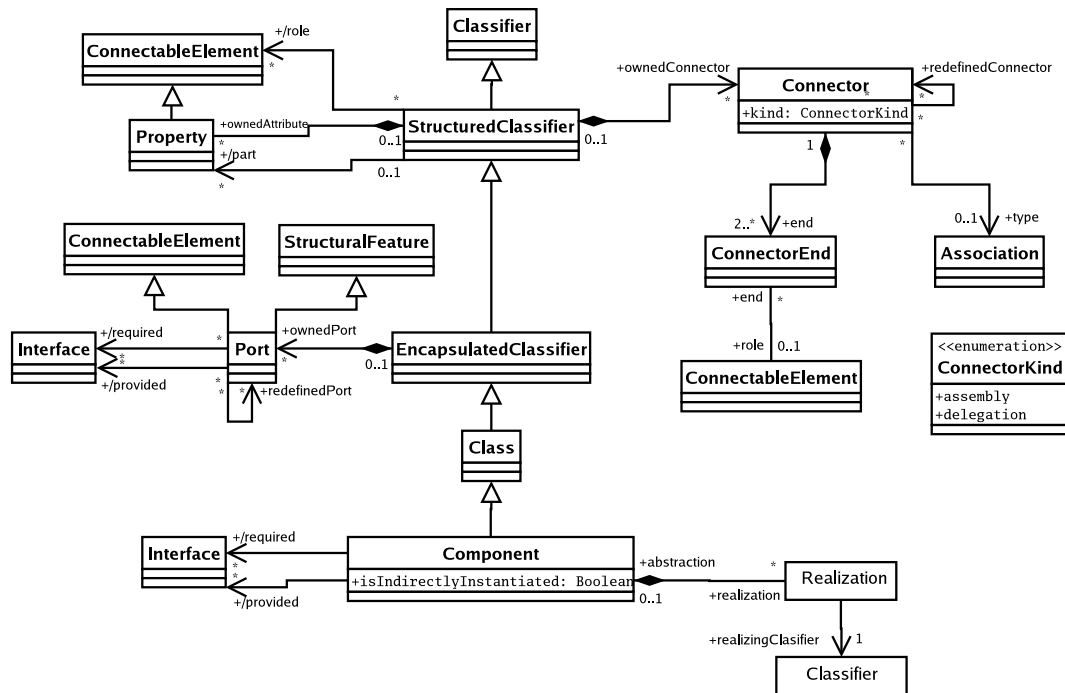


FIG. 5.1 – Méta modèle UML2

d'un modèle de composant. Le méta-modèle de composant de UML 2.0 [OMG03b] permet de définir les spécifications des composants, ainsi que l'architecture de l'application que l'on désire développer. UML2.0 spécifie un composant comme étant une unité modulaire, réutilisable, qui interagit avec son environnement par l'intermédiaire de points d'interactions appelés ports (voir Figure 5.1). Les ports sont typés par les interfaces : celles-ci contiennent un ensemble d'opérations et de contraintes ; les ports (et par conséquent les interfaces) peuvent être fournis ou requis. Le comportement interne du composant ne doit être ni visible, ni accessible autrement que par ses ports. Enfin, le composant est voué à être déployé un certain nombre de fois, dans un environnement a priori non déterminé lors de la conception (excepté au travers des ports requis). Il existe deux types de modélisation de composants dans UML2.0 : le composant basique et le composant composite. La première catégorie définit le composant comme un élément exécutable du système. La deuxième catégorie étend la première en définissant le composant comme un ensemble cohérent de parties. La connexion entre les ports requis et les ports fournis se fait au moyen de connecteurs. Deux types de connecteurs existent : le connecteur de délégation et le connecteur d'assemblage. Le premier est utilisé pour lier un port du composant composite vers un port d'un composant situé à l'intérieur de ce dernier (donc pour relier par exemple un port requis à un autre port requis). Le deuxième type de connecteur est utilisé pour les liens d'assemblage (donc relier un port requis à un port fourni). Les connecteurs sont vus comme des moyens d'assemblage et d'adaptation ; en réalisant cette connexion entre les ports (peut-être incompatibles), le connecteur offre l'avantage d'effectuer les assemblages sans modification des composants.

5.3 Les éléments architecturaux du modèle AlkoWeb

Ainsi, les abstractions architecturales UML2.0 auront le sens suivant :

Le composant : Il représente les éléments Web à différents niveaux d'abstraction. Il peut être atomique ou hiérarchique. Les composants atomiques sont considérés comme des boîtes noires (ne possédant pas une structure interne explicite). Les composants hiérarchiques quant à eux possèdent une structure interne explicite et sont décrits par un assemblage de composants hiérarchiques ou atomiques. Ainsi les zones de textes, les formulaires d'authentification et les cases à cocher sont autant d'exemples de composants atomiques. Toute composition de ceux-ci correspond à un composant hiérarchique.

L'interface : Elle représente les services définis par les composants. Les interfaces peuvent être de trois types :

Les interfaces synchrones : Elles contiennent les objets traditionnels orientés opérations.

Elles sont réparties en deux catégories :

- Les interfaces fournies : Elles définissent les services fournis aux autres composants. Par exemple, un composant *HTMLTextField* (zone de texte) définit une interface qui fournit des services comme *getFormattedValue*, correspondant aux formats de la valeur texte.
- Les interfaces requises : Elles décrivent le composant en terme de services requis, censés être fournis par les autres composants. Par exemple le composant *CheckBox* définit une interface requise dont l'opération est *setExternalValue*. Elle permet d'initialiser ou de modifier la valeur d'un attribut d'un autre composant (la valeur d'un champs texte par exemple).

Les interfaces asynchrones : Elles représentent les opérations basées sur les événements (appelées aussi interfaces d'évènements). Chaque service est exécuté seulement si un événement particulier se produit. L'implémentation de telles opérations est uniquement définie par des scripts interprétés côté client comme JavaScript. Un exemple de ce type de service en HTML est le *OnClick* d'un bouton, le *OnSelect* d'une liste, le *onFocus* d'un formulaire ou encore l'opération *mouseover*.

Les interfaces de contrôle : Elles regroupent les opérations nécessaires à la validation du contenu d'un composant. Comme pour les interfaces précédentes, l'implémentation de ces opérations est réalisée par un langage de script interprété par le client. Par exemple, dans un composant de type formulaire HTML, nous pouvons exécuter des contrôles pour vérifier que tous les champs d'un formulaire ont bien été remplis (dates bien formatées, URLs valides par vérification de domaine via un composant *DnsCheck*, etc..).

Dans AlkoWeb, tous les types d'interfaces sont modélisables par des interfaces UML2.X classiques. Nous les trouvons suffisantes et complètes pour concevoir et pour réutiliser nos composants sans ambiguïté.

Les Ports : Ils englobent un ensemble d'interfaces qui représentent un tout cohérent ou destinées aux mêmes fonctionnalités. Ils peuvent inclure des interfaces requises, fournies, de contrôle ou d'évènement.

Les connecteurs : Ce sont des éléments architecturaux orientés interaction. Ils lient les interfaces des différents composants et encapsulent les protocoles d'interactions. Un exemple sera donné à la section ???. Ces connecteurs peuvent être de différents types.

Les connecteurs hiérarchiques : Ils relient les composants hiérarchiques et leurs sous-composants.

Les connecteurs d'assemblage : Ils relient entre eux les composants d'un même niveau hiérarchique.

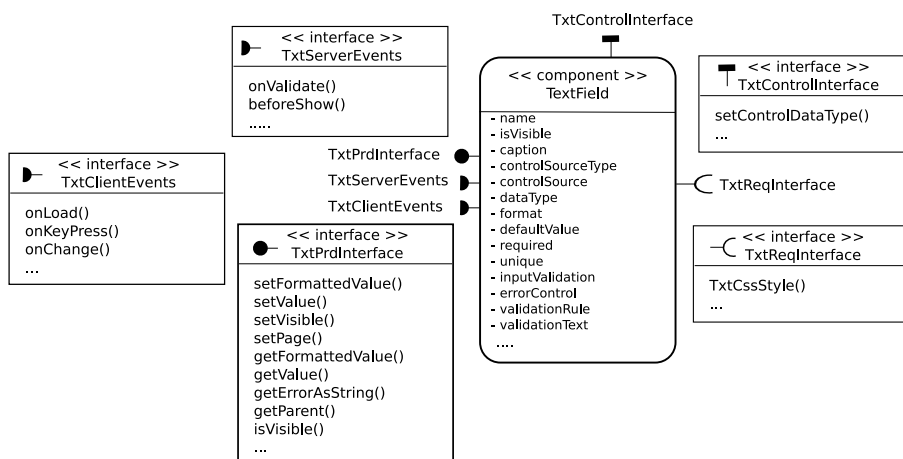


FIG. 5.2 – Un exemple de composant graphique de type Texte

5.4 Exemple Illustratif

AlkoWeb a servi pour le développement d'une large gamme de produits. Ces produits intègrent une multitude de technologies : des systèmes d'accès à des annuaires (OpenLdap, ActiveDirectory, etc.), des systèmes d'authentification, ou de cryptographie (MD5, DES, etc.), des accès aux bases de données (MySQL, PostgreSQL/Postgis, etc.), des composants d'accès aux service Web cartographique (WMS, WFS, etc.), des Widgets Ajax basés sur Dojo ([21]), Scriptaculos ([63]), Rico ([58]), Google et Ajax Yahoo APIs.

La Figure 5.4 donne un exemple d'utilisation du diagramme de composants pour la modélisation d'applications Web riches. Les figures 5.2 et 5.3 décrivent respectivement deux composants, **TextField** et **LdapManager**.

Ces composant fournissent et requièrent un certain nombre d'interfaces synchrones, ils possèdent également des interfaces d'évènements et de contrôles. Par exemple, dans la figure 5.2, les interfaces fournies et requises par le composant **TextField** sont *TxtPrdInterface* et *TxtReqInterface*. Ce composant possède des propriétés (name, format, etc.) comparables aux propriétés d'un JavaBean et accessible via différentes méthodes proposées par ses interfaces. Les interfaces fournies, requises, d'évènements et de contrôles, offrent un certain nombre de méthodes, elles sont illustrées dans les deux figures 5.2 et 5.3. L'interface de contrôle *TxtControlInterface* offre par exemple, une méthode *setControlDataType()* pour le contrôle du format

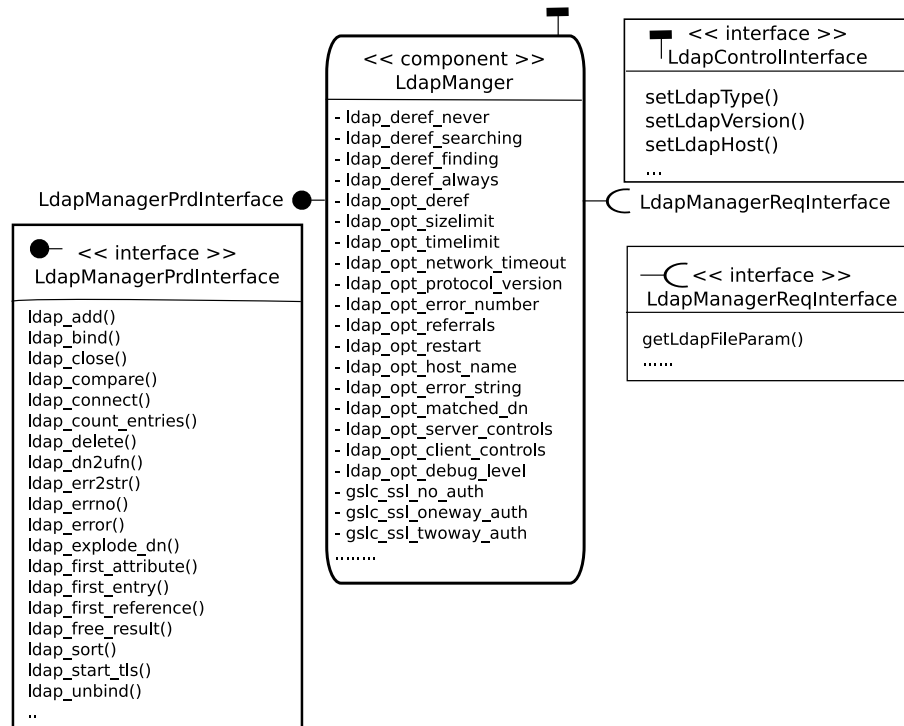


FIG. 5.3 – Un exemple de composant de gestion d'annuaire LDAP

de la donnée entrée. **TextField** possède également une interface de contrôle (*TxtControllInterface*) et deux autres d'évènements. Les interfaces d'évènements sont séparées en deux groupes :

Le premier dont la source est la partie cliente (*TxtClientEvents*), le second dont la source provient du serveur (*TxtServerEvents*).

La figure 5.4 montre une application Web qui correspond au composant **AlkAppliExemple**. Ce composant est composé de trois autres composants, **AlkPageIndex**, **AlkPageCarte** et **AlkPageExplorer**. Le premier, correspond à la page d'accueil de l'application (l'index), les deux autres correspondent respectivement à une application cartographique et à un explorateur Web de documents.

Le composant **AlkAppliExemple** illustre le premier niveau de la hiérarchie de l'application, lorsque nous naviguons dans l'architecture de l'application, nous accédons à ses différents niveaux. Dans la figure 5.4, nous distinguons trois niveaux de l'architecture de l'application **AlkAppliExemple** (il est impossible de donner une vision de tout ses niveaux dans une seule figure). Le composant **AlkPageIndex** fait partie du deuxième niveau de l'architecture. Il est composé de deux autres composants (**AlkFormListCountr** et **LinkExplorer**) appartenant au troisième niveau de l'architecture. **AlkFormListCountr** est un composant de type formulaire HTML. Il contient cinq sous-composants. Le premier sous composant **PostalCode** modélise le composant **TextField** précédemment décrit. Sa valeur fonctionnelle correspond au code postal d'une ville. Les trois autres sous composants sont des composants de type **ListBox**. Ils corres-

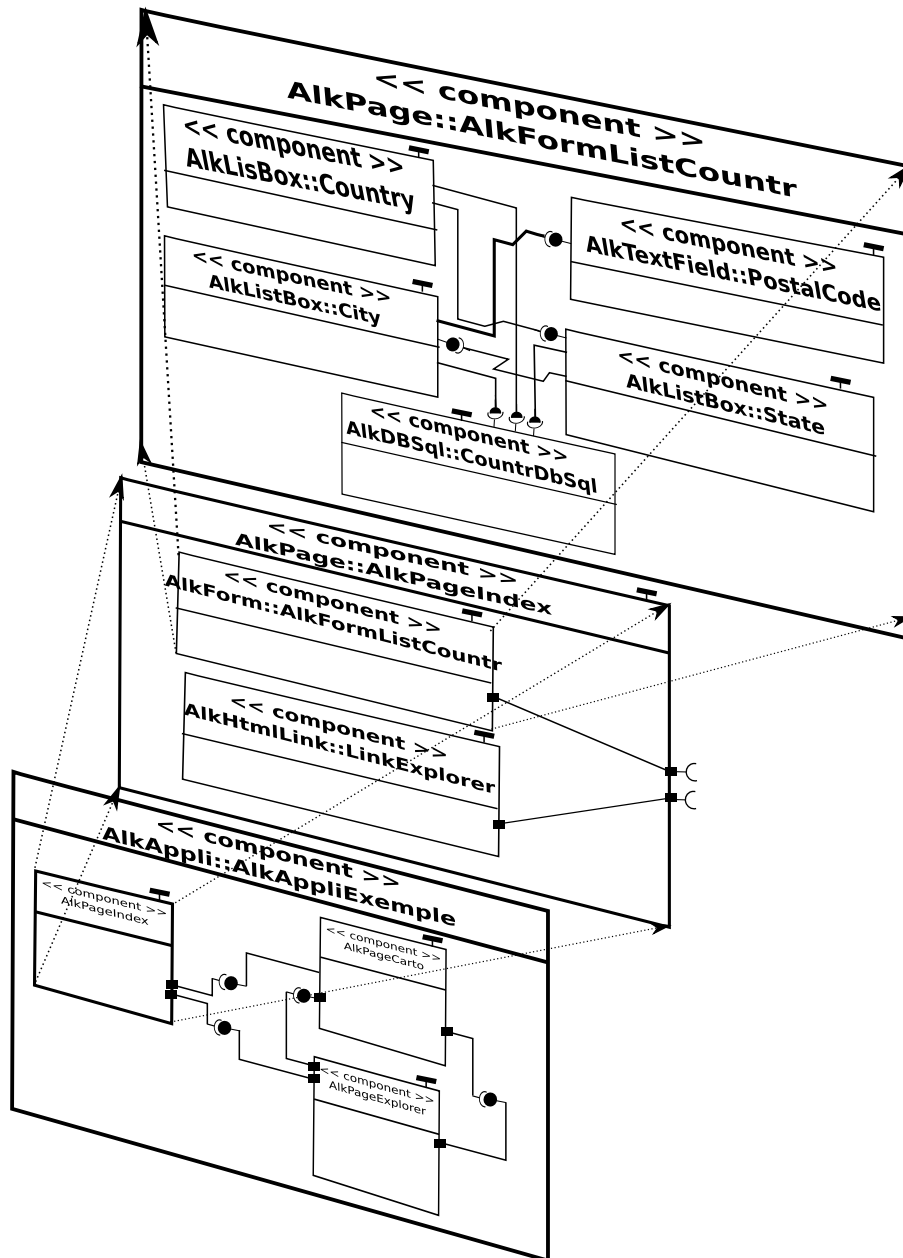


FIG. 5.4 – Un exemple illustratif de l'utilisation d'AlkoWeb

pondent à des listes de pays, de régions, et de villes. Ces informations sont stockées dans une base de données accessible par le composants **CountrDbSql**. Les composants sont reliés entre eux par des connecteurs via leurs interfaces fournies et requises respectivement (*sockets et lollipop* d'UML2.0). Des connecteurs hiérarchiques lient les interfaces du composant composite à celles de son sous composant. Ces liaisons sont représentées par des flèches liant

le sous composant au port de son super composant.

Les composant `ListBox` représentant les listes des pays, des régions et des villes se connectent à la base de données pour les obtenir. Dès que l'un des composants reçoit l'évènement de sélection d'un utilisateur, il exécute l'opération `onChange()` qui met à jours les autres composant `ListBox`. Ce dernier se connecte alors à la base via `XMLHttpRequest` pour récupérer la liste des régions qui correspondent au pays choisi. La même séquence d'exécution se produit pour les autres composants `ListBox`. Ce mode fonctionnement basé sur Ajax est modélisé ici par une architecture à base de composants hiérarchiques.

5.5 Contraintes de modélisation et de déploiement

Je viens de montrer que le diagramme de composants UML 2.X, convenablement projeté dans le domaine des applications Web riches, est un moyen performant pour les modéliser. Je propose d'aller plus loin encore et de profiter d'une facilité supplémentaire offerte par le standard UML, le langage OCL, pour adjoindre aux modèles de la documentation supplémentaire.

Lors du développement d'un composant Web on peut souhaiter documenter des contraintes qui décrivent de quelle façon celui-ci pourra être modifier. On peut par exemple vouloir imposer qu'un composant page HTML contenant deux onglets ne pourra pas en comporter plus. On impose ainsi qu'une nouvelle page soit créée si l'on souhaite rajouter un troisième onglet à cette page. Afin de documenter de manière rigoureuse ce type de contrainte, j'ai utilisé le langage de contrainte OCL 2.0. Par exemple, si nous avons besoin de définir une contrainte qui assure que le composant ayant comme nom **TextField** ne peut être connecté à plus de deux composants différents, je l'exprime à l'aide de la contrainte ci-dessous.

```
context TextField:Component inv:
TextField.interface.connectorEnd.connector.connectorEnd
.interface.component->asSet()->size() <= 3
```

Cette contrainte navigue à travers tous les connecteurs auxquels sont reliées les interfaces de **TextField**. Elle obtient alors tous les composants dont les interfaces sont reliées aux extrémités de ces connecteurs. Le résultat obtenu est alors transformé pour enlever les duplications. Le résultat englobe même le composant **TextField**, c'est la raison pour laquelle `size` est inférieure ou égale à 3. (au lieu de 2, comme cité dans les contraintes du paragraphe précédent).

Il est également souhaitable d'interdire dès la conception la création d'architectures qui ne respecteraient pas les spécifications W3C du langage HTML (par exemple une page HTML ne peut contenir qu'un seul formulaire). J'ai donc exprimé à l'aide de contraintes OCL des contraintes émises par le W3C de manière à pouvoir contrôler dynamiquement leur respect pendant l'activité de modélisation.

En effet, à chaque association de composition, on peut écrire une contrainte assurant la validité de cette association lors de la modélisation. Les contraintes assurant la validité du modèle de composition s'écrivent de la manière suivante :

```
context : Component
inv : self.stereotype = 'HtmlPage' implies
      self.components->forall(c | Set{'HtmlForm'}->includes(c.stereotype))
```

Cette contrainte s'applique sur les entités de type Component (context), et doit vérifier en permanence (inv) qu'un composant stéréotypé **HtmlPage** ne peut être composé que de composants de type **HtmlForm**. Cette contrainte n'est qu'un exemple, car à la vue de diagramme de composition établi, les contraintes sont de manière générale plus complexe que celle présentée. L'ensemble des contraintes associées au modèle de composition est données en annexes(ref).

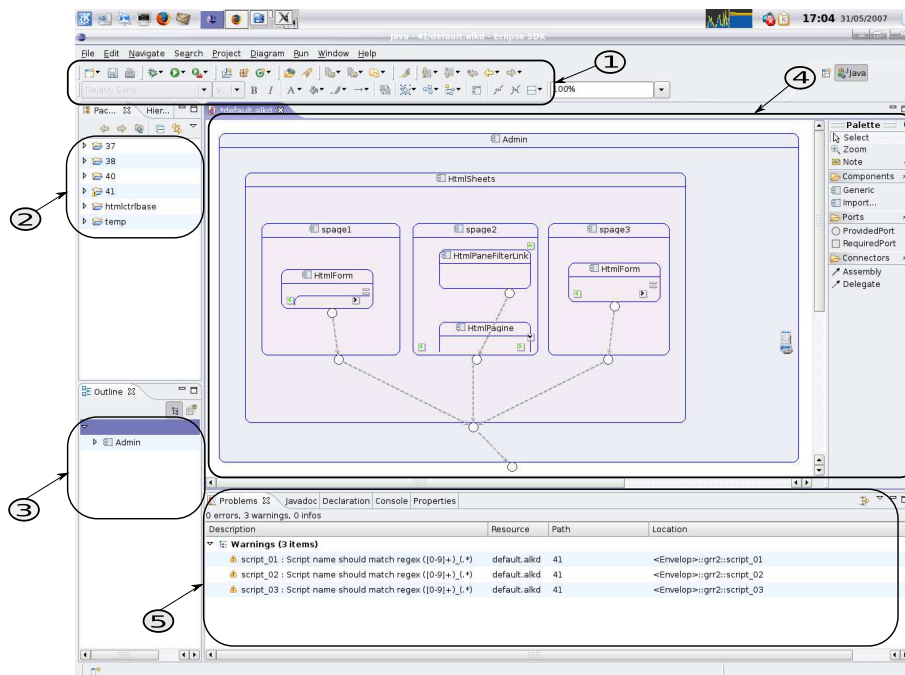


FIG. 5.5 – Capture écran d'AlkoWeb-Builder

5.6 Déploiement de l'application

A la fin du développement de l'application, nous procédons à son déploiement. L'application est accompagnée d'un fichier descripteur détaillant l'assemblage de ses composants. Il existe trois possibilités de déploiement.

Déploiement d'évaluation : Ce déploiement facilite le test des composants. Il peut être partiel afin de réaliser des tests sur une partie seulement des composants.

Le déploiement de qualification : L'application est déployée dans l'environnement du client.

Il est utilisé pour effectuer des tests avant recette.

Déploiement de production : permet la livraison finale de l'application.

5.7 AlkoWeb-Builder

Chaque artefact dans l'implémentation du modèle de l'application Web est associé au code qui lui correspond. Lorsque nous naviguons hiérarchiquement dans le modèle, nous pouvons

ainsi parcourir de la même manière le code implémenté. Pour ce faire, j'ai donc implémenté des liens d'associations permettant d'associer les éléments d'un modèle à leur code grâce à un mécanisme de marquage et de commentaire. Les associations peuvent aussi référencer des fichiers ou des répertoires. Je trouve que ce mécanisme est une bonne pratique dans la création de liaison entre la vue architecturale et la vue physique d'une application Web. J'applique en cela le principe défendu par Clemetset al en 2003 ([15]) : maintenir la relation entre la vue physique et la vue architecturale d'une application.

L'environnement développé, correspond à différents frameworks offerts par la plateforme Eclipse. AlKoWeb-Builder a été conçu comme un ensemble de plug-ins permettant de séparer l'implémentation du modèle de composant de son éditeur graphique.

Eclipse est un Environnement de Développement Intégré (EDI) issu de la communauté open source. Il est développé en Java et se focalise essentiellement sur l'extensibilité de sa plateforme. Il est formé d'un grand nombre de projets² de base qui permettent à la fois de développer des applications industrielles et d'enrichir la plate-forme Eclipse elle-même. Le choix d'Eclipse pour le développement d'un éditeur de composants est lié en majeure partie à la possibilité d'ajout de plug-ins. Elle permet d'étendre ses fonctionnalités afin de pouvoir entre autres gérer d'autres langages de programmation. AlKoWeb-Builder se présente donc sous la forme d'un plugin Eclipse, accessible via les menus et totalement fondu dans l'interface de la plate-forme, gagnant ainsi en ergonomie et en utilisabilité.

Trois principaux frameworks ont été utilisés pour le développement d'AlKoWeb-Builder.

- **GMF** (Graphical Modeling Framework ([22])) offre une infrastructure pour la production d'éditeurs graphiques complets basés sur EMF (Eclipse Modeling Framework) et GEF (Graphical Editing Framework). D'un côté le modèle de composants est modélisé par l'utilisation des fonctionnalités EMF et offre ainsi un ensemble de classes Java le représentant, de l'autre côté, GMF enrichi GEF avec de nouvelles fonctionnalités graphique.
- **MDT** (Model Development Tools ([22])) offre deux frameworks : UML2 et OCL. Le framework UML2 est utilisé pour l'implémentation des spécifications UMLTM 2.0. De la même manière, le framework OCL est l'implémentation de la norme OCL standard de L'OMG. Il offre une API pour le parcours et l'évaluation des contraintes OCL dans un modèle MOF.
- **JET** (Java Emitter Templates ([22])) est une partie du projet M2T ([22]) (Model To Text). JET est utilisé comme un générateur de code à partir de modèles. Des templates à la JSP peuvent être transformés vers n'importe quel type d'artefacts source (Java, PHP, Javascript ...).

Afin de mettre en place la navigation par niveaux dans l'éditeur, il a fallu retoucher au code métier généré par EMF. Malheureusement, si le framework fournit de nombreuses fonctionnalités il est aussi très restrictif quand à sa modification, et l'implantation de la navigation par niveaux a nécessité une sauvegarde de l'état global de la hiérarchie des composants, afin de pouvoir se déplacer dans les différents noeuds de celle-ci et revenir dans un état antérieur. Ce fonctionnement n'est sûrement pas optimal mais est pour l'instant la seule solution trouvée

²Des exemples de projets sont par exemple Eclipse (Rich Client Platform), BIRT (Business Intelligence & Reporting) ou WST (Web Standard Tools), qui offre un support pour le développement d'EJBs ou d'applications basé sur Ajax.

pour permettre la navigation dans les noeuds de la hiérarchie. La figure ?? illustre la navigation dans la hiérarchie des composants.

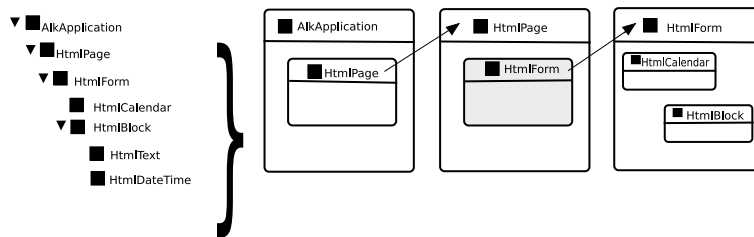


FIG. 5.6 – Un exemple de la vue de l'interface d'une application

5.7.1 Fonctionnement

La première version du prototype ALKoWeb-Builder a été développée en quatre principaux plug-ins : modèleur, éditeur, contraintes et transformation. Les plug-ins de modélisations et d'éditations représentent l'éditeur graphique avec toute la palette graphique de modélisation UML2, l'édition et l'enregistrement des différents artefacts (Comme le montre la figure 5.5). Ils représentent la partie générique de l'architecture liée à l'implémentation des composants décrite précédemment. Le plug-in de contrainte permet la validation des diagrammes des différents modèles. Par exemple en assurant que le composant `HtmlTextField` ne peut être ajouté aux composant `HtmlForm`. La génération de code est elle aussi matérialisée par un plug-in afin de permettre la génération de différents types de code source issus de différents langages (JSP, ASP, ...).

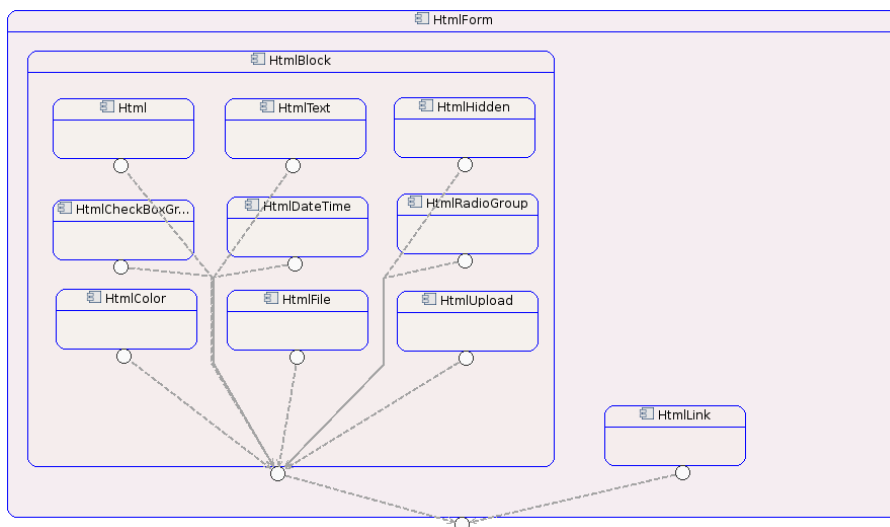


FIG. 5.7 – Un exemple de la vue de l'interface d'une application

Dans un premier temps, notre implémentation a été modélisée grâce à EMF. Cette implémentation

est une instance du metamodelle EMF (Ecore) qui est une implémentation JAVA d'un sous-ensemble noyau de MOF (Meta Object Facility). Dans les futurs releases, nous planifions d'utiliser l'implémentation Java du méta-modèle d'UML2 offerte par le framework MDT UM2 d'eclipse.

A partir de la description du modèle en XMI (XML Meta-data interchange), EMF produit un ensemble de classes Java. Ces classes servent de modèle de domaine dans l'architecture de GMF, principalement conçu dans un modèle architecturale MVC (Model View Controller). L'environnement d'exécution de GMF offre un ensemble intéressant de propriétés prèintégrées, comme les diagrammes de persistance, de validation et OCL. Dans AlkoWeb-Builder, la propriété de persistance permet de sauvegarder le diagramme en deux ressources XMI séparées : le fichier principal (contient une instance du modèle de composant) et le fichier de diagramme (contient les notations des éléments graphiques).

EMF offre aussi un langage support pour les contraintes. Dans notre outil, les contraintes OCL sont utilisées pour valider les diagrammes et assurer l'intégrité du modèle. Un éditeur OCL a été développé sous la forme d'un plug-in qui implémente l'assistance à l'évolution orientée qualité des diagrammes et de l'implémentation du modèle de composants définie plus haut. Cet éditeur OCL est basé sur un framework proposé par la plateforme Eclipse modifiée, nous avons ajouté un système d'auto-complétion de (capabilities) pour faciliter l'édition de contraintes par les développeurs.

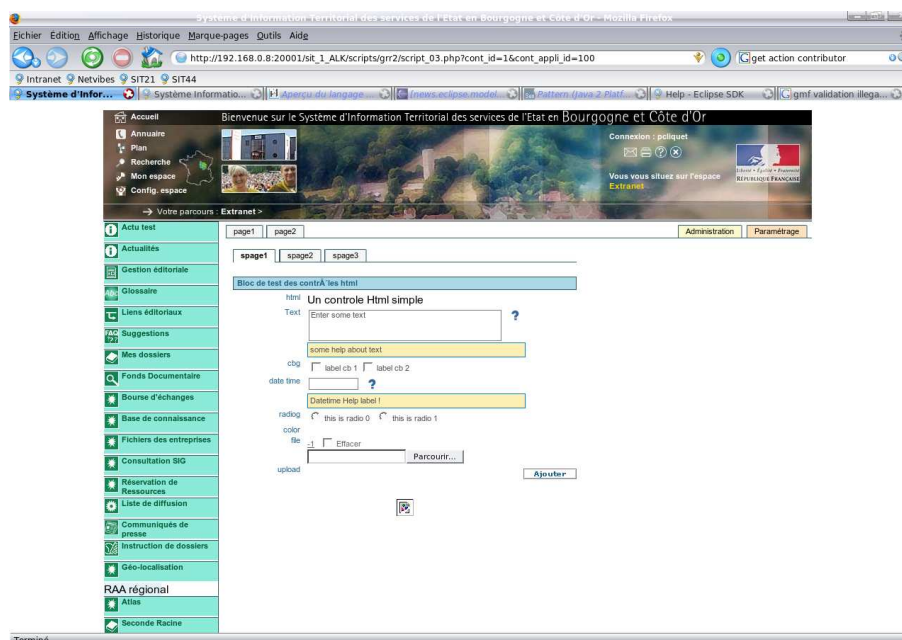


FIG. 5.8 – Un exemple de la vue de l'interface d'une application

Finalement, le framework JET2 nous a offert la possibilité de développer notre partie de génération de code. Il consiste en deux ensembles de fichiers : un modèle d'entrée et un ensemble de fichiers templates. Le fichier en entrée fourni dans un langage XML est dans notre cas l'instance du modèle précédemment modélisée avec l'éditeur GMF. Les templates utilisent

le langage Xpath pour atteindre les noeuds et les attributs du modèle en entrée, elles génèrent ensuite tout type de texte prédéfinis. La première version d'AlkoWeb-Builder utilise ces templates pour générer du code PHP.

La figure 5.5 présente une vue d'AlkoWeb-Builder. L'outil étant fourni sous forme de plugins, il permet d'exploiter directement diverses fonctionnalités, comme les barres d'outils et les menus (1), ou encore le Package Explorer (2) qui permet de naviguer dans les projets en cours de développement. L'éditeur se compose de la vue d'édition graphique et d'une palette (4), qui permettent la modélisation des composants, ports et connecteurs. La vue Outline / Overview (3) permet de visualiser le modèle sous la forme d'un arbre hiérarchique des composants, ou encore sous forme de vue miniature, pratique pour des modèles d'une taille importante. Enfin, la vue Properties (5) permet d'accéder aux attributs, graphiques et métiers, des éléments modélisables.

La figure 5.7 montre la structure interne du composant HTMLForm vu dans la figure précédente. Les composants sont modélisés hiérarchiquement et incrémentalement. Un double clic sur l'un d'eux offre, d'un côté, la possibilité d'accéder à son architecture, si cette dernière existe, et d'un autre côté permet de lui en définir une nouvelle si cette dernière n'existe pas. L'interprétation du code PHP, HTML, Javascript généré, est exposée dans la figure 5.8.

Bibliographie

- [1] Web engineering, software engineering and web application development. In San Murugesan and Yogesh Deshpande, editors, *Web Engineering*, volume 2016 of *Lecture Notes in Computer Science*. Springer, 2001.
- [2] Corba components. In *Forml/02-06-65.*, 2002.
- [3] Web engineering, 7th international conference, icwe 2007, como, italy, july 16-20, 2007, proceedings. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *ICWE*, volume 4607 of *Lecture Notes in Computer Science*. Springer, 2007.
- [4] G.F. Antoniou and F. Harmelen. A semantic web primer. In *The MIT Press*, 2004.
- [5] Garzotto F. Baresi, L. and M. Maritati. W2000 as a mof metamodel. In *6th World Multi-Conf. on Systemics, Cybernetics and Informatics - Web Engineering Track.*, 2002.
- [6] Koch N. Baumeister, H. and L. Mandel. Towards a uml extension for hypermedia design. In *Second International Conference on Unified Modeling Language (UML99)*, pages 614–629, 1999.
- [7] K. Beck. Embracing change with extreme programming. In *A spiral model of software development and enhancement. IEEE Computer*, pages 70–77, 1998.
- [8] T. et al. Berners-Lee. The semantic web. In *Scientific American*, pages 34–43, 2001.
- [9] Michael Bieber. Hypertext and web engineering. In *Hypertext*, pages 277–278, 1998.
- [10] Barry W. Boehm, Alexander Egyed, Julie Kwan, Daniel Port, Archita Shah, and Raymond J. Madachy. Using the winwin spiral model : A case study. volume 31, pages 33–44, 1998.
- [11] Rumbaugh J. Booch, G. and I. Jacobson. The unified modeling language user guide (object technology series). In *Addison-Wesley, Reading, MA.*, 1999.
- [12] E. Bruneton, T. Coupaye, and J.B. Stefani. The fractal component model specification. final release, version 2.0-3. [http ://fractal.objectweb.org/specification/](http://fractal.objectweb.org/specification/), 2004.
- [13] Fraternali P. Bongio A. Brambilla M. Comai S. Ceri, S. and M. Matera. Designing data-intensive web applications. In *Designing Data-Intensive Web Applications.*, 2002.
- [14] Bongio A. Ceri S., Fraternali P. Web modeling language (webml) : a modeling language for designing web sites. In *Ninth World Wide Web Conference*, 2000.
- [15] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford. *Documenting Software Architectures, Views and Beyond*. Addison-Wesley, 2003.

-
- [16] Jim Conallen. *Modeling Web Applications with UML, 2nd Edition*. Addison-Wesley Professional, October 2002.
- [17] Cutter Consortium. Poor project management number-one problem of outsourced e-projects. <http://www.cutter.com/research/2000/crb001107.html>, 2000.
- [18] P. Dantzing. Architecture and design of high volume web sites. In *International Conference on Software and Knowledge Engineering*, 2002.
- [19] Leune C. De Troyer O. Wsdm - a user-centered design method for web sites. In *7th International World Wide Web Conference.*, 1997.
- [20] Yogesh Deshpande, San Murugesan, Athula Ginige, Steve Hansen, Daniel Schwabe, Martin Gaedke, and Bebo White. Web engineering. *J. Web Eng.*, 1(1) :3–17, 2002.
- [21] DOJO. The dojo toolkit. <http://dojotoolkit.org/>.
- [22] Eclipse. Eclipse web site. <http://www.eclipse.org/>, Last access : June 2007.
- [23] C. Eichinger. The discipline of systematic development of web applications systems. In *Web application architecture*, pages 65–84, 2006.
- [24] D. Balek F. Plasil and R. Janecek. Dynamic component updating in java corba environment. In *Report 97/10, Department of Software Engineering, Charles University, Prague*, 1997.
- [25] Jeewani Anupama Ginige, Buddhima De Silva, and Athula Ginige. Towards end user development of web applications for smes : A component based approach. In *In proceedings of the 5th International Conference on Web Engineering (ICWE'05)*, pages 489–499, Sydney, Australia, July 2005. LNCS 3579, Springer-Verlag.
- [26] J. Gomez and C. Cachero. Information modeling for internet applications. In *Chapter OO-H method : extending UML to model web Interfaces, Idea Group Publishing.*, pages 144–173, 2003.
- [27] Cachero C. Gomez J. and Pastor O. Extending a conceptual modelling approach to web application design. In *12th International Conference CAiSE 2000*, 2000.
- [28] P. Humbert. Halin G., J.C. Bignon. Designing hypermedia : An experience in multimedia catalogue of building products. In *ACM Hypertext'99 Workshop on Hypermedia Development.*, 1999.
- [29] Rolf Hennicker and Nora Koch. Systematic design of web applications with uml. In *Unified Modeling Language : Systems Analysis, Design and Development Issues*, pages 1–20. Idea Group Publishing, Hershey, PA, USA, 2001.
- [30] Stohr E. A. et Balasubramanian P. Isakowitz T. A methodology for the design of structured hypermedia applications. In *Communications of ACM.*, pages 34–44, 1995.
- [31] Conallen J. Concevoir des applications web avec uml. In *Edition Eyrolles*, 2000.
- [32] Reda Kadri, François Merciol, and Salah Sadou. Cbse in small and medium-sized enterprise : Experience report. In *CBSE*, pages 154–165, 2006.
- [33] Reda Kadri, François Merciol, and Salah Sadou. Une expérience d'intégration d'une architecture à base de composants. In *CAL*, pages 132–140, 2006.

-
- [34] Reda Kadri, Chouki Tibermacine, Régis Fleurquin, Salah Sadou, and François Merciol. Alkoweb : Un outil pour modéliser l'architecture des applications web riches. In *CAL*, pages 107–118, 2008.
- [35] Reda Kadri, Chouki Tibermacine, and Vincent Le Gloahec. Building the presentation-tier of rich web applications with hierarchical components. In *WISE*, pages 123–134, 2007.
- [36] Mohamed Mancona Kandé and Alfred Strohmeier. Towards a uml profile for software architecture descriptions. In *Proceedings of UML'2000 - The Third International Conference on the Unified Modeling Language : Advancing the Standard -*, York, United Kingdom, October 2000.
- [37] Koch N. Moser F. Knapp, A. and G Zhang. Argouwe : A case tool for web applications. In *First International Workshop on Engineering Methods to Support Information Systems Evolution (EMSISE 03)*, 2003.
- [38] N. Koch and A. Kraus. The expressive power of uml-based web engineering. In *Second Internatioanl Workshop on Web-Oriented Software Technology (IWWOST 02)*, pages 105–119, 2002.
- [39] N. Koch and A. Kraus. Towards a common metamodel for the development of web applications. In *Third International Conference on Web Engineering (ICWE 03)*, pages 495–506, 2003.
- [40] Zhang G. Koch, N. and M.J. Escalona. Model transformations from requirements to web system design. In *Sixth International Conference on Web Engineering (ICWE 06)*, pages 281–286, 2006.
- [41] Olsina L. Building a web-based information system applying the hypermedia flexible process modeling strategy. In *Conference Hypertext*, 1998.
- [42] Olsina L. A scenario-based object-oriented methodology for developing hypermedia information systems. In *31st Annual Conference on Systems Science.*, 1998.
- [43] G. Coulson M. Clarke, G. Blair and N. Parlavantzas. An efficient component model for the construction of adaptive middleware. In *The Conference on Distributed Systems Platforms (Middleware'01)*, pages 160–178.
- [44] R. Marvie and P. Merle. Vers un modèle de composants pour cesure, le corba component model. In *Report Projet RNRT, LIFL*, 2000.
- [45] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling software architectures in the unified modeling language. *ACM Transactions On Software Engineering and Methodology*, 11(1) :2–57, 2002.
- [46] Gomez J. Melia, S. and N. Koch. Improving web design methods with architecture modeling. In *Electronic Commerce and Web Technologies, Copenhagen.*, 2005.
- [47] S. Murugesan. Understanding web 2.0. In *IEEE IT Professional*, 2006.
- [48] S. Murugesan and A. Ginige. Web engineering : Introduction and perspectives. In *Web Engineering : Principles and Techniques*, 2005.
- [49] Koch N. The authoring process of the uml-based web engineering approach. In *1st International Workshop on Web-Oriented Software Technology.*, 2001.

- [50] Jakob Nielsen. User interface directions for the web. volume 42, pages 65–72, 1999.
- [51] Griffiths G. Lockyer M. Oates, B. and B. Hebbbron. Empirical methodologies for web engineering, proceedings. In *ICWE*, 2004.
- [52] OSGi service gateway specification. Open Services Gateway Initiative. Open services gateway initiative. <http://www.osgi.org/>, 2003.
- [53] O'Reilly. What is web 2.0 In *Design patterns and business models for the next generation of software.*, 2005.
- [54] Thomas A. Powell, David L. Jones, and Dominique C. Cutts. *Web site engineering : beyond Web page design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.
- [55] R. Pressman. Software engineering-a practitioner's approach. In *edition, McGraw-Hill*, 2005.
- [56] Roger S. Pressman, Ted G. Lewis, Ben Adida, Ellen Ullman, Tom DeMarco, Thomas Gilb, Brent C. Gorda, Watts S. Humphrey, and Ray Johnson. Can internet-based applications be engineered ? volume 15, pages 104–110, 1998.
- [57] Donald J. Reifer. Web development : Estimating quick-to-market software. volume 17, 2000.
- [58] RICO. Javascript for rich internet applications. <http://openrico.org/>.
- [59] M. Flatt S. McDirmid and Jiazzi. W. Hsieh. New-age components for old-fashioned java. In *Conference on Object-Oriented Program- ming, Systems, Languages, and Applications (OOPSLA'01)*, ACM Press, 2001.
- [60] Wimmer M. Schauerhuber, A. and E. Kapsammer. Bridging existing web modeling languages to model-driven engineering : a metamodel for webml. In *Model Driven Web Engineering.*, 2003.
- [61] Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2) :25–31, 2006.
- [62] Rossi G. et Barbosa S. D. J. Schwabe D. Systematic hypermedia design with oohdm. In *Hypertext Conference*, 1996.
- [63] ScriptAculos. Javascript toolkit. <http://script.aculo.us/>.
- [64] N. Shadbolt. The semantic web revisited. In *IEEE Intelligent Systems*, pages 96–101, 2006.
- [65] Sun-Microsystems. Enterprise javabeans(tm) specification, version 2.1. <http://java.sun.com/products/ejb>, November 2003.
- [66] C. Szyperski. *Component Software : Beyond Object-Oriented Programming*. Addison-Wesley, 2002.
- [67] N. Takako and T. Tetsuo. Towards constructing a class evolution model. *IEEE Proceeding of the 4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC'97/ICSC'97)*, pages 131–138, 1997.
- [68] Greer J. Thomson J. and Cooke J. Algorithmically detectable design patterns for hypermedia collections. In *Workshop on Hypermedia development Process, Methods and Models. Hypermedia*, 1998.

-
- [69] Richard T. Vidgen. What's so different about developing web-based information systems? In *ECIS*, 2002.
- [70] Schwabe D. Vilain, P. and C.S. de Souza. A diagrammatic tool for representing user interaction in uml. In *Third International Conference on Unified Modeling Language (UML 00)*, pages 133–147, 2000.
- [71] WebModels. Webratio tool suite. <http://www.webratio.com>., 2006.
- [72] Koch N. Rossi G. Garrido A. Mandel L. Helmerich A. Wirsing, M. and L. Olsina. Hyperuml : Specification and modeling of multimedia and hypermedia applications in distributed systems. In *Second Workshop on German-Argentinian Bilateral Programme for Scientific and Technological Cooperation*, 1999.