AlkoWeb: Un outil pour modéliser l'architecture des applications Web riches

Reda Kadri, Chouki Tibermacine, Régis Fleurquin, Salah Sadou, François Merciol

AlkoWeb: Un outil pour modéliser l'architecture des applications Web riches

Reda Kadri*, Chouki Tibermacine**
Régis Fleurquin, Salah Sadou, François Marciol***

*Alkante SAS 31A rue des Landelles 35510 Cesson-Sévigné
r.kadri@alkante.com,
http://www.alkante.com
**UMR 5506 161 rue Ada 34392 Montpellier Cedex 5 - France
tibermacin@lirmm.fr
http://www.lirmm.fr
***VALORIA - Université de Bretagne Sud, Campus de Tohannic
Bât. Yves Coppens, BP 573, 56017 Vannes Cedex
sadou,fleurquin,merciol@univ-ubs.fr

Résumé. La couche présentation des architectures n-tiers a besoin d'être conçue avec des modules correctement structurés et réutilisables. Dans ce article, nous présentons un modèle de composants hiérarchiques qui permet aux développeurs de modéliser, de générer du code et de réutiliser ce niveau logiciel des applications Web riches. Dans ce modèle, les composants peuvent être inter-connectés via leurs interfaces pour construire des composants plus complexes. Ces modèles peuvent être réutilisés avec leur code grâce à un mécanisme d'association. Comme nous le montrons dans cet article, ceci est une propriété intéressante pour assister les développeurs pour naviguer dans le code source et les modèles de leurs applications. Ceci permet par ailleurs de maintenir une cohérence entre les artefacts de ces deux étapes du processus de développement (conception et implémentation).

1 Introduction

De nos jours, les systèmes d'information sont de plus en plus distribués et déployés sur Internet. Le Web représente actuellement la plate-forme standard pour l'hébergement de tels systèmes. Dans ce contexte, l'utilisation d'une architecture multi-tiers est souvent la meilleure décision de conception qui satisfasse les objectifs de qualité comme le passage à l'échelle, la maintenabilité et la fiabilité. Un gros travail a été fait pour améliorer les parties traitement et données des applications multi tiers. La partie cliente (présentation) a souvent été, quant à elle, assimilée à un navigateur Web. Or, ces dernières années celle-ci est devenue de plus en plus complexe. C'est l'article publié par James Garett(Garrett (2005)), co-fondateur d'*Active Path*, qui semble avoir suscité la prise de conscience et l'intérêt des développeurs vis-à-vis

de cette partie cliente[1] (souvent appelée Web riche). Plusieurs technologies, avec leurs outils associés, sont à ce jour proposées pour le développement de clients "Web riche" (AJAX, Flash, XUL ou Eclipse RPC). Pour le moment, Elles ne sont destinées qu'à mener à bien l'étape de codage. De ce fait, l'architecture de ces applications est rarement un sujet de préoccupation, on conséquence, elle pose de sérieux problèmes de maintenance et de d'évolution.

Dans cet article, nous proposons une approche, supportée par un outil, appropriée pour modéliser d'une part les architecture des clients Web riche et d'autre part pour générer le code associé. Cette approche repose sur la technologie des composants logiciels. Elle consiste en une interprétation du diagramme de composants UML2.0, appelée AlkoWeb. Le choix de ce type de diagramme UML2.0 est, entre autres, dicté par le fait que les applications Web riches sont basées sur des éléments organisés hiérarchiquement. En effet, une simple page HTML peut inclure un formulaire, qui peut être composé de différents composants (`input`,`select` ou `text area`), lui-même réutilisable pour composer d'autres applications clientes. Les techniques existantes de composition d'applications Web ne prennent en compte que les aspects statiques et structurels des composants. Cela occasione un travail supplémentaire aux développeurs afin d'implémenter les liaisons entre ces dernières (liaisons non dynamiques). Les composants que supporte notre approche vont des simples éléments HTML (formulaires, liens hypertexte, etc..) aux éléments plus complexes tels que ceux concernant l'authentification, l'interface mail ou la gestion éditoriale. L'objectif que nous nous sommes fixé est d'offrir aux développeurs la possibilité de définir interactivement les liens de composition entre ces composants, afin de faciliter la conception d'applications Web riches.

Dans la section suivante, nous présentons la sémantique des éléments architecturaux de notre implémentation. Un exemple sera ensuite détaillé dans la section 3. Dans la section 4 nous allons décrire les contraintes architecturales et le déploiement de nos applications. Dans la section 5, nous présentons un outil développé dans le but d'implémenter notre approche. Avant de conclure cet article nous discutons quelques travaux connexes.

2 Les éléments architecturaux d'AlkoWeb

Au lieu de proposer un nouveau méta-modèle de composant et donc éviter l'introduction d'un nouveau langage de modélisation architecturale, nous avons choisi de nous appuyer sur un standard existant et largement adopté : UML2.0. Nous trouvons dans les spécifications du diagramme de composants de la version 2.0 d'UML toutes les abstractions dont nous avons besoin pour la modélisation d'applications Web riches à l'aide d'entités hiérarchiques. L'interprétation du méta-modèle de composants UML2.0 proposée dans cet article est cependant orientée application Web. 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

[1] Une discussion sur le sujet est exposée sur le blog de James Garett (http ://www.adaptivepath.com/publications/essays/archives/000385.php)

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 évènements (appelées aussi interfaces d'évènements). Chaque service est exécuté seulement si un évènement 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 3. 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.

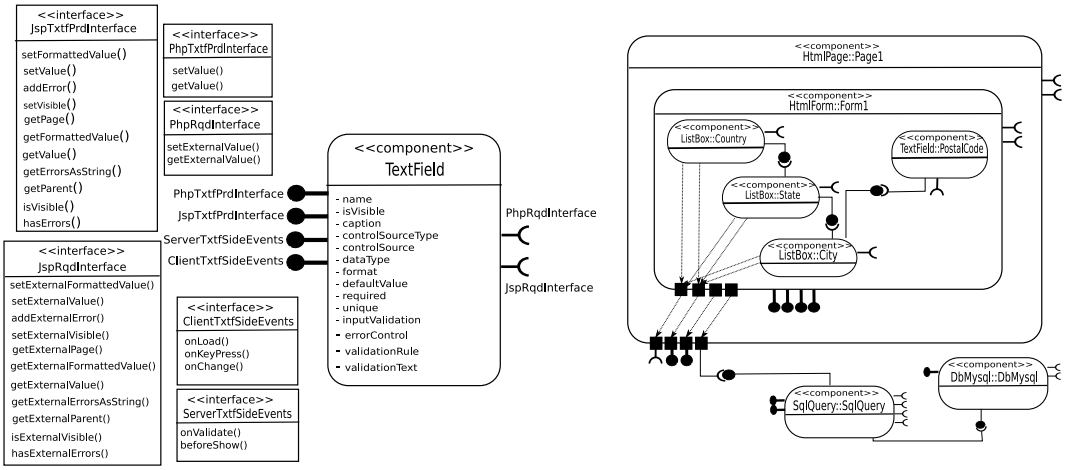

FIG. 1 – *Un exemple d'architecture pour un composant Page*

3 Exemple Illustratif

Nous avons développé à l'aide d'AlkoWeb 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, PostgresSQL/Postgis, etc .), des composants d'accès aux service Web cartographique (WMS, WFS,etc.), des Widgets Ajax basés sur Dojo(DOJO), Scriptaculos (ScriptAculos), Rico(RICO), Google et Ajax Yahoo APIs. Pour des raisons de brièveté et de limitation d'espace, nous ne détaillerons qu'un seul de ces composants qui illustre l'intérêt de notre approche . La Figure 1 donne un exemple d'utilisation du diagramme de composants pour la modélisation d'applications Web riches. La partie gauche de la figure 1 décrit un composant `TextField`. Ce composant fournit et requiert un certain nombre d'interfaces synchrones et possède quelques interfaces d'évènement. Les interfaces fournies et requises par ce composant sont séparées en deux groupes :
PHP-specific (`PhpTxtfPrdInterface` et `PhpTxtfReqInterface`) et JSP-specific (`JspTxtfPrdInterface` et `JspTxtfReqInterface`). Les interfaces d'évènements sont également séparées en deux groupes :
Le premier dont la source est la partie cliente (`ClientTxtfSideEvents`), le second dont la source provient du serveur(`ServerTxtfSideEvents`).

La partie droite de la figure 1 montre une page HTML d'une application Web. Dans cette page figure un formulaire pour sélectionner un pays, puis une région suivie d'une ville, et enfin le code postal de la ville choisie. Cette page est modélisée par un composant(`HTMLPage`) qui possède de multiples interfaces requises, fournies et d'évènements. Ce composant contient un sous composant de nom `HTMLForm`. Des connecteurs hiérarchiques lient les interfaces du composant composite à celles de son sous composant. Ces liaisons sont représentées par des flèches sur la figure liant le sous composant au port de son super composant.

Le composant `HTMLForm` contient quatre sous-composants. Le premier sous composant

(à droite) 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 correspondent à des listes de pays, régions, et villes. Nous supposons que toutes les informations géographiques de ces localités sont stockées dans une base de données représentée par les deux sous composants `SqlQuery` et `DbMysql`. Ces composants sont reliés entre eux par des connecteurs via leurs interfaces fournies et requises respectivement *(sockets and lollipops d'UML2.0)*.

Le composant `ListBox` représentant la liste des pays se connecte à la base de données pour obtenir la liste disponible des pays. Dès que ce composant reçoit l'évènement de sélection de l'utilisateur exécute l'opération `onChange()` qui met à jours le deuxième composant `ListBox`. Ce dernier se connecte alors à la base via XMLHtttprequest pour récupérer la liste des régions qui correspondent au pays choisi. La mêmes séquence d'exécution se produit pour le dernier composant `ListBox`. Ce mode fonctionnement basé sur Ajax est modélisé ici par une architecture à base de composants hiérarchiques.

4 Contraintes architecturales et déploiement

Nous venons de montrer que le diagramme de composants UML 2.X, convenablement projeté dans le domaines des applications Web riches, est un outil performant pour modéliser simplement la couche présentation de ce type d'application. Nous proposons 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 évoluer. On peut par exemple vouloir imposer qu'un composant page HTML contenant deux onglets ne pourra dans l'avenir 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 architecturale, nous avons usé du 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, nous l'exprimerons 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). Nous avons donc exprimé à l'aide de contraintes OCL l'intégralité des contraintes émises par le W3C de manière à pouvoir contrôler dynamiquement leur respect pendant l'activité de modélisation.

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 AlKoWeb-Builder

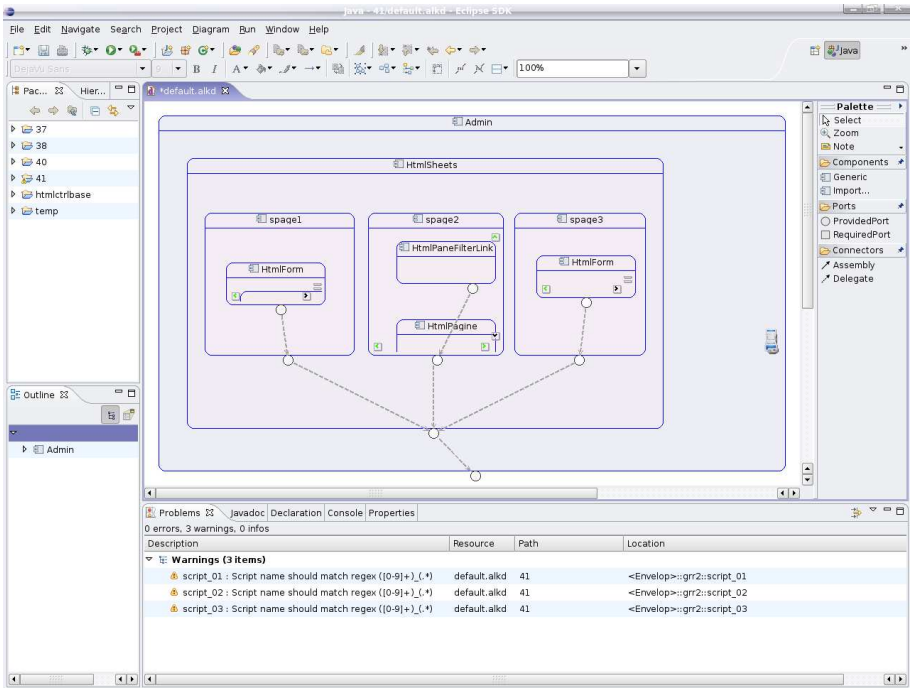

FIG. 2 – *Capture écran d'AlKoWeb-Builder*

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, nous avons 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. Nous trouvons que ce mécanisme est une bonne pratique dans la création de liaison entre la vue architecturale et la vue physique d'une application Web. Nous appliquons en cela le principe défendu par Clemetset al en 2003 (Clements et al. (2003)) : maintenir la relation entre la vue physique et la vue architecturale d'une application.

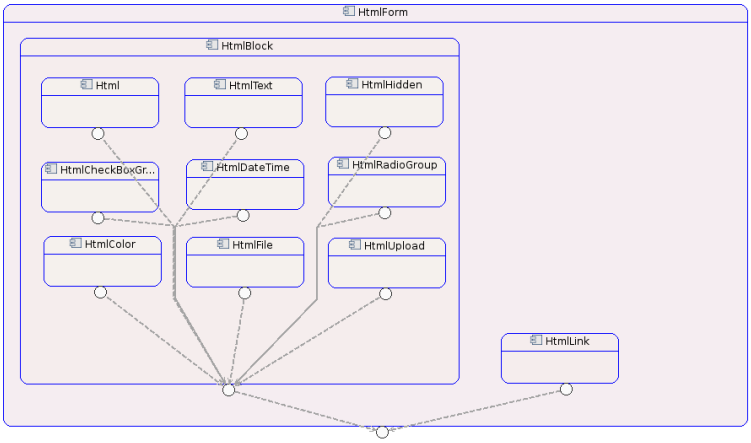

FIG. 3 – *Structure interne du composant HTMLForm*

L'environnement que nous avons développé correspond à différents frameworks offerts par la platforme 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. La section suivante présente d'une part les frameworks utilisés pour le développement de cet outil et d'autre part comment nous générons le code de l'application depuis les modèles architecturaux.

Eclipse est maintenant suffisamment mature et correspond à une plate-forme productive. Il est composé de différents projets offrant des frameworks pour le développement logiciel ; des 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. Trois principaux frameworks ont été utilisés pour le développement d'AlkoWeb-Builder.

- **GMF** (Graphical Modeling Framework (Eclipse (2007))) offre une infrastructure pour la production d'éditeurs graphiques complets basés sur EMF (Eclipse Modeling Framework) et GEF (Graphical Editing Framework). D'un coté 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 coté, GMF enrichi GEF avec de nouvelles fonctionnalités graphique.
- **MDT** (Model Development Tools (Eclipse (2007))) offre deux frameworks : UML2 et OCL. Le framework UML2 offre deux frameworks : UML2 et OCL. Le framework UML2 est utilisé pour l'implémentation des spécifications UML^{TM} 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 (Eclipse (2007))) est une partie du projet M2T (Eclipse (2007)) (Model To Text). JET est utilisé comme un générateur de code à partir de modèles. Des templates à la JSP peuvent être transformées vers n'importe quel type d'artefacts source (Java, PHP, Javascript ...).

5.1 Fonctionnement

La première version du prototype AlKoWeb-Builder a été développée en quatre principaux plug-ins : modeleur, éditeur, contraintes et transformation. Les plug-ins de modélisations et d'éditions 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 2). 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, ...).

Dans un premier temps, notre implémentation a été modélisé grâce à EMF. Cette implémentation est une instance du metamodèle 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 prèintegrées, 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 ficher 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.

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 language 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 3 montre la structure interne du composant HTMLForm vu dans la figure précédente. Les composant sont modélisé hiérarchiquement et incrémentalement. Un double clic sur l'un d'eux offre, d'un coté, la possibilité d'accéder à son architecture, si cette dernière existe, et d'un autre coté 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 4.

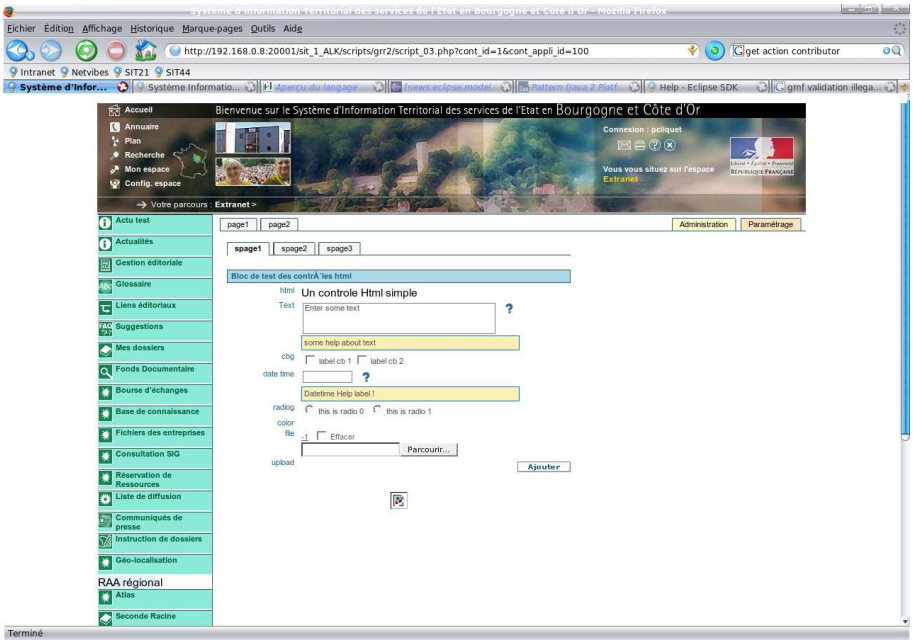

FIG. 4 – *Un exemple de la vue de l'interface d'une application*

6 Travaux connexes

La modélisation des architectures logicielles avec UML a été discutée dans de multiples travaux (Medvidovic et al. (2002); Kandé et Strohmeier (2000)). Différents types d'extensions ont été proposées 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 cet article, nous avons montré 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. De plus, nous avons utilisé UML dans sa version 2.0 et non 1.5.

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 (Conallen (2002)). 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 notre article, elle ne traite que les applications Web traditionnelles et non les applications Web riches. Ces dernières permettent dans certaines situations une communication directe entre les composants du tiers présentation et les composants du tiers données (comme illustré dans la section 3). 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 c'est simplement d'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 le tiers présentation est par nature hiérarchique.

Dans (Hennicker et Koch (2001)), 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 (Ginige et al. (2005)), les auteurs présentent une approche pour les utilisateurs finaux afin de développer des applications Web utilisant des composants. L'approche proposée 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 exécutés à la fin. Dans cette approche, les composants varient des générateurs de formulaires aux moteurs de requêtes de bases de données. La préoccupation est donc pour des applications Web entières ; tous les tiers et leurs environnements d'exécution sont ciblés. Dans le travail que nous avons présenté dans ce papier, les composants sont spécifiques au tiers présentation. Nous nous focalisons sur la modélisation d'architectures d'applications basées sur des composants de libraire collaboratifs. Comme précisé précédemment, notre approche est centrée-produit et non orientée-organisation comme dans (Ginige et al. (2005)).

7 Conclusion

Alkante développe des applications cartographiques Web riches pour des organismes gouvernementaux en Bretagne et dans toute la France[2]. Le travail présenté dans cet article est dans la continuité d'un précédent travail (Kadri et al. (2006)) réalisé par cette société dont l'objectif était de définir un processus de développement dédié aux applications Web et basé sur la conception et la réutilisation de composants logiciels. Dans ce processus, la réutilisation de modules hiérarchiques est une préoccupation récurrente. Afin de faciliter cette tache, nous avons introduit une notation basée sur le langage UML2.X et un outil support. Cet environnement présente en particulier l'originalité d'user d'OCL pour adjoindre de la documentation inter-

[2] Alkante Web site : www.alkante.com

prétée dynamiquement lors du travail de modélisation. Cette documentation peut par exemple servir à contraindre les évolutions futurs d'un composant ou permettre la vérification en temps réel du respect du standard W3C. Cet environnement est également accompagné d' un système de gestion de versions dont le but est d'organiser et de maintenir les différents composants d'AlKoWeb et son Dépôts. Ce système est basé sur un répository de composants utilisé à la manière d'un CVS (avec gestion des branches et marquage de modules).

Nous prévoyons dans un futur proche de mettre à disposition l'environnement AlkoWeb-Builder et notre dépôt de composants afin qu'ils puissent être utilisé et enrichi par la communauté dans un contexte open source. AlkoWeb offrirait à la communauté des développeurs de systèmes d'information Web un langage architectural souple dans un environnement convivial et transparent. La base des composants déjà développés pourrait également servir de base de test à la communauté des chercheurs en ingénierie de logiciel dans le paradigme composant. Nous allons également introduire le langage ACL (Tibermacine et al. (2005)) dans l'outil. Ce langage dédié à la documentation de choix architecturaux et de contraintes d'évolution offrira de nouvelles possibilités d'une part sur le plan documentaire et d'autre part sur le plan des contrôles dynamiques effectués lors des activités de modélisation.

Références

Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, et J. Stafford (2003). *Documenting Software Architectures, Views and Beyond*. Addison-Wesley.

Conallen, J. (2002). *Modeling Web Applications with UML, 2nd Edition*. Addison-Wesley Professional.

DOJO. The dojo toolkit. http ://dojotoolkit.org/.

Eclipse (Last access : June 2007). Eclipse web site. http ://www.eclipse.org/.

Garrett, J. J. (February 18, 2005). Ajax : A new approach to web applications. Archived at http ://www.adaptivepath.com/publications/essays/archives/000385.php/.

Ginige, J. A., B. De Silva, et A. Ginige (2005). 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)*, Sydney, Australia, pp. 489–499. LNCS 3579, Springer-Verlag.

Hennicker, R. et N. Koch (2001). Systematic design of web applications with uml. In *Unified Modeling Language : Systems Analysis, Design and Development Issues*, pp. 1–20. Hershey, PA, USA : Idea Group Publishing.

Kadri, R., F. Merciol, et S. Sadou (2006). Cbse in small and medium-sized enterprise : Experience report. In *Proceedings of the 9th ACM SIGSOFT International Symposium on Component-Based Software Engineering (CBSE'06)*, Vasteras, Sweden. Springer LNCS.

Kandé, M. M. et A. Strohmeier (2000). 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.

Medvidovic, N., D. S. Rosenblum, D. F. Redmiles, et J. E. Robbins (2002). Modeling software architectures in the unified modeling language. *ACM Transactions On Software Engineering and Methodology 11*(1), 2–57.

RICO. Jacascript for rich internet applications. http ://openrico.org/.

ScriptAculos. Javascript toolkit. http ://script.aculo.us/.

Tibermacine, C., R. Fleurquin, et S. Sadou (2005). Preserving architectural choices throughout the component-based software development process. In *Proceedings of the 5th IEEE/IFIP Working Conference on Software Architecture (WICSA'05)*, Pittsburgh, Pennsylvania, USA, pp. 121–130. IEEE Computer Society Press.

Summary

Software in the presentation-tier of a multi-tiered architecture needs in practice to be designed with structured and reusable library modules. In this paper, we present a hierarchical component model which allows developers to build (model, generate code and then reuse) this software level of rich Web applications. In this model, components can be connected via their interfaces to build more complex components. These architecture design models can be reused together with their corresponding code using an association mechanism. As shown in this paper this is a valuable feature in assisting developers to position their developed documents within the overall software design and thus enable maintaining the consistency between artifacts of these two stages of the development process.