Une méthodologie de test pour la validation des services Web

Ana Cavalli, Stéphane Maag
Institut National des Télécommunications
CNRS UMR 5157
9, rue Charles Fourier, F-91011 Evry Cedex
Email : {Ana.Cavalli,Stephane.Maag}@int-evry.fr

Fatiha Zaïdi
Université Paris-Sud XI
LRI, CNRS UMR 8623
Bat. 490 Université Paris Sud, F-91405 Orsay Cedex
Email : Fatiha.Zaidi@lri.fr

***Abstract*— Cet article présente une méthodologie de test de conformité pour des services web, et en particulier, son application à des services d'enseignement à distance sur une plate-forme open source. La méthodologie inclut la spécification en UML des systèmes étudiés, principalement en utilisant les concepts de diagrammes de séquence, diagrammes d'activité, diagrammes de classes et la carte de navigation. La méthodologie inclut aussi la génération de tests à partir de ces spécifications, leur exécution sur le système à tester et le traitement des données en proposant deux approches : une basée sur le langage OCL et l'autre sur les profils UML. La plate-forme open source étudiée est la plate-forme dotLRN, qui est un environnement de développement de services web et un portail de support de l'enseignement à distance, de communautés en ligne et de travail collaboratif. Le travail présenté dans cet article fait partie du projet E-LANE[1], une collaboration Européenne et Latino-américaine pour la création de plates-formes open source intégrées pour l'enseignement à distance.**

I. Introduction

Ces dernières années, des progrès considérables ont été réalisés dans la conception et le développement des environnements de services web, et en particulier, dans le développement des méthodologies et des plates-formes pour l'enseignement à distance. Plus récemment, cette évolution a intégré l'idée de développer des plates-formes logicielles open source utilisant des applications déjà existantes. Dans ce contexte, des institutions éducationnelles peuvent prévoir l'enseignement d'un ensemble de cours en utilisant des techniques d'enseignement à distance, en dépassant les barrières géographiques, s'adressant à une plus vaste audience, tout en réduisant les coûts. Pour réduire ces coûts, il leur suffit de développer un serveur web puissant, avec pour objectif de permettre aux utilisateurs d'accéder à ces services en utilisant une plate-forme ouverte connectée à Internet. De plus, ces plates-formes sont conçues de telle sorte qu'elles encouragent les utilisateurs à utiliser non seulement les services proposés mais aussi à en créer de nouveaux.

Cet article présente le travail développé dans le cadre du projet E-LANE (European -Latin American New Education), une collaboration Européenne et Latino-américaine pour la création de plates-formes intégrées open source pour l'enseignement à distance. Ce projet fait partie du programme de la Commission Européenne @LIS Demonstration Projects. La plate-forme de base pour E-LANE est dotLRN, qui est une plate-forme open source composée par un ensemble d'applications web et un portail pour un système d'enseignement à distance.

Ces plates-formes ouvertes, du fait qu'elles intègrent divers services développés par différentes équipes, doivent être testées afin de garantir que ces services soient intégrés correctement et de manière souple. L'intégration de nouveaux services par les utilisateurs doit aussi être testée, afin de garantir que les fonctionnalités du système sont préservées. En particulier, ces systèmes sont confrontés à de nombreux problèmes : un serveur web doit répondre à des requêtes provenant de n'importe quel noeud du réseau. Par exemple, le test devra garantir l'impossibilité pour un utilisateur d'obtenir des autorisations d'accès qu'il n'est pas censé obtenir. De plus, on doit considérer que les nouveaux services web deviennent de plus en plus complexes et critiques. Cette situation est aggravée en raison du manque de normalisation pour les clients web (navigateurs). Le test doit aussi garantir l'usage du système. Par exemple, un utilisateur peut ne pas être capable de compléter une activité car le contenu de la page web n'apparaît pas correctement, ou bien à cause d'une connexion trop lente, il se trouve dans l'impossibilité de se connecter en raison d'un "timeout" trop rapide. Notre approche prend en compte la manière dont ces aspects fonctionnels on été implantés, en vérifiant sa conformité à la spécification.

Dans cet article, nous proposons une méthodologie de test pour les services web, et plus particulièrement son application aux services web de la plate-forme dotLRN. Cette méthodologie est basée sur une méthode de génération de tests à partir de spécifications UML (Unified Modelling Language) et leur exécution sur le système à tester. Nous introduisons également une nouvelle approche pour le test de nouveaux services web, créés par les utilisateurs, complémentaire de la méthode de génération de tests. Dans cette nouvelle approche il s'agit de décrire les propriétés souhaitées du service web dans le langage OCL [1] (Object Constraint Language) et de les vérifier sur les traces d'exécution du service.

D'autres contributions présentées dans cet article sont les suivantes. En effet nous illustrons la conception d'une méthodologie composée par des techniques et des outils utilisés pour

[1]http ://www.e-lane.org/

démontrer la conformité des services web de la plate-forme. La méthodologie proposée inclut les étapes suivantes : 1) spécification des services web en UML, en particulier des cas d'utilisation utilisant les diagrammes d'activités, diagrammes de séquence, diagrammes de classes et la carte de navigation ; 2) l'exportation des diagrammes dans le format XMI (XML Metadata Interchange) ; 3) le parsing du XMI pour générer des scripts de test ; 4) la génération des scripts de test (dans différents langages de script). Cet article présente aussi deux modèles des données, un en utilisant des contraintes OCL et l'autre des profils UML.

Dans les sections qui suivent, nous présentons avant tout les travaux existants dans ce domaine. Puis, l'article décrit les services web disponibles de la plate-forme dotLRN (section 3). La section 4 décrit comment spécifier les services web de cette plate-forme en utilisant les modèles UML comme spécification. Dans la section 5, nous décrivons les outils pour automatiser la génération de tests à partir de ces modèles. La section 6 introduit deux approches pour la modélisation des données et la section 7 expose les perspectives de ce travail. Enfin, la section 8 présente les conclusions de ce travail.

II. Travaux existants

Très peu de travaux sont proposés pour la génération automatique de cas de test à partir de l'analyse des modèles UML [2], [3]. L'une des raisons est que les modèles UML varient de manière significative d'une technique de développement à une autre. En effet, il n'y a pas de méthodes standardisées pour le développement de modèles UML. Par exemple, les auteurs de [2] ne font pas d'hypothèses spécifiques quant à l'emploi des cas d'utilisations, des diagrammes de séquences, des gardes, etc., ce qui complique la généralisation d'une méthode automatisée des tests. Dans [4], les auteurs adaptent des critères de couvertures traditionnels de flots de données (e.g., definition-utilisations de variables) au contexte des diagrammes de collaboration mais ne l'appliquent pas à la génération de cas de test. Les auteurs de [3] proposent une méthodologie nommée TOTEM, pour développer des modèles UML et en dériver des cas de test. Cependant, aucun de ces travaux n'est adapté au test d'applications web dans un environnement d'enseignement à distance. Dans notre cas, nous devons tester les interfaces graphiques mais aussi le contenu des pages générées. Il existe une extension d'UML pour des applications web mais elle est davantage centrée sur le " Design View ". Aussi, afin de produire les diagrammes qui transportent l'information nécessaire à la production de nos tests, nous considérons le travail présenté dans [5] et [6]. Dans notre travail, nous répondons à certaines des interrogations susmentionnées.

III. Services Web de la plate-forme dotLRN

La plate-forme dotLRN[2] est une plate-forme open source d'enseignement à distance sur le web et un portail web conçu pour gérer des cours en ligne, des communautés en ligne et des collaborations. Il y a trois types de portails : utilisateurs, classes et communautés. Le portail utilisateur est un espace privé qui appartient à chaque utilisateur, alors que les portails classes et communautés contiennent toutes les pages qui concernent une communauté particulière (par exemple "communauté sur la photographie") ou une classe (par exemple "cours d'informatique"). Chacun de ces trois types de portails est divisé en quatre sections : l'espace principal, le calendrier, les fichiers, le panneau de contrôle. Les pages sont composées de *portlets*. Ces derniers sont de petits portails qui ont des fonctionnalités particulières, comme les forums, les news, le calendrier. Un portail utilisateur est créé automatiquement lorsqu'un nouvel utilisateur s'est enregistré dans dotLRN, par contre les portails classes et communautés sont eux créés par l'administrateur selon les besoins de l'utilisateur. Lorsqu'une nouvelle classe ou communauté est créée, l'administrateur affecte à ce portail un ou plusieurs utilisateurs administrateurs. Par exemple, pour le portail relatif aux cours d'informatique, les administrateurs peuvent être le professeur et ses professeurs assistants, alors que pour un portail de communauté sur la photographie les administrateurs peuvent être un ou plusieurs étudiants. La responsabilité d'un administrateur de portail est d'ajouter du contenu, de personnaliser l'affichage et de décider de la politique du portail. Lorsqu'une classe ou une communauté a une politique dite "ouverte", n'importe quel utilisateur peut les rejoindre. Si la politique est au contraire "fermée", seul l'administrateur peut ajouter des utilisateurs. Une troisième politique existe, la politique "d'attente", où un utilisateur peut demander à rejoindre le portail et ensuite l'administrateur décide soit de l'accepter ou soit de lui refuser l'accès. Les services web liés aux trois types de portails que l'on vient de décrire sont les services que l'on souhaite tester et que l'on modélise en UML telle que l'explique la section suivante.

IV. Modèles UML pour le test

Afin de dériver des tests, nous devons comprendre et décrire les fonctionnalités du système de manière formelle. Une approche de développement à base de modèles pour les applications web peut être réalisée en utilisant le langage UML. Nous utilisons une méthodologie de conception qui est basée sur une extension de UML pour les applications web [7]. Elle se décompose en trois étapes principales à savoir le modèle conceptuel, la carte de navigation et le modèle de présentation. Le modèle conceptuel est construit en considérant les besoins fonctionnels capturés par des cas d'utilisation [8]. Nous devons identifier les cas d'utilisation du système considéré et les documenter dans le cahier des charges par un document d'expression des besoins qui contient le but du système, les acteurs principaux, le diagramme des cas d'utilisation et la description textuelle de chacun d'eux.
Ce que nous obtenons de cette étape n'est pas directement utilisé en entrée de notre outil de tests mais il est important afin de modéliser les diagrammes UML qui constituent nos entrées.

[2]http ://www.dotlrn.org

A partir du modèle conceptuel, le modèle de l'espace de navigation, qui est représenté aussi par un modèle de classes statiques, est construit. Il définit une vue du modèle conceptuel montrant les classes qui peuvent être visitées par navigation dans l'application web. Finalement, un modèle de présentation dynamique est représenté par des diagrammes de séquence UML. Ces derniers décrivent les collaborations et les comportements des objets navigationnels et leurs primitives d'accès. Afin de spécifier notre modèle global, les modèles suivants sont utilisés :

- le diagramme de classes qui introduit les principales classes du système ;
- le diagramme d'activités de chaque acteur qui met en évidence les dépendances au sein des cas d'utilisation ;
- la carte de navigation qui fournit des informations sur le contenu dynamique des pages web ;
- le diagramme de séquence pour chaque cas d'utilisation qui décrit les scénarios principaux et optionnels représentant le modèle de présentation dynamique.

Au sein de notre laboratoire, nous avons développé un outil qui implante notre méthodologie. Les composants qui constituent notre outil sont décrits dans la Figure 1. Avant tout, il analyse les dépendances trouvées dans les diagrammes d'activités. A partir de ces dépendances et de données d'entrée, il génère les séquences de test. Ces dernières sont ensuite utilisées dans le module d'exécution de tests. De la même manière, les informations contenues dans les cas d'utilisation et dans la carte de navigation sont utilisées par ce module pour exécuter les requêtes courantes sur le serveur dotLRN. Le dernier module, quant à lui, analyse les réponses du serveur et les compare à la spécification UML de la page web attendue. Enfin, il fournit le verdict qui permet d'établir si le test a été exécuté avec succès ou s'il a échoué.

A. D'un modèle conceptuel à une carte de navigation

Afin de spécifier les services web de la plate-forme dotLRN, nous devions tout d'abord construire un modèle conceptuel du domaine de l'application en prenant en compte les besoins fonctionnels capturés par les cas d'utilisation. Le modèle conceptuel a été construit en trouvant les classes et leurs associations et en définissant les structures d'héritage utilisées. Ensuite, la carte de navigation de l'application web introduite dans [5] est utilisée car elle nous fournit des informations sur le contenu dynamique de chaque page web qui font partie intégrante du système au même titre que les liens entre les différentes pages web. Ces informations sont essentielles lors de l'analyse syntaxique des pages HTML (voir section V-C). La carte de navigation est un diagramme de classes où chaque page web est une classe et chaque lien entre deux pages est une association entre les deux classes respectives. Cette extension d'UML pour les applications web introduit un nombre conséquent de *stéréotypes* (tels que les indexes, les requêtes, les menus, etc.), de *valeurs étiquetées* et de *contraintes*.
La Table I résume les associations que nous faisons entre les entités web et les éléments du meta-modèle dans un diagramme de classes.

B. Modélisation des dépendances entre cas d'utilisation

Les cas d'utilisation d'un système ne sont pas indépendants. Pour qu'un cas d'utilisation soit exécuté, il faut qu'au préalable le cas d'utilisation précédent l'ait été. Par exemple, pour les services web de dotLRN, l'utilisateur doit s'être « logué » avant de pouvoir faire n'importe quelle autre action. La procédure d'automatisation des tests doit prendre cela en considération, et donc décrire les dépendances qui existent entre les cas d'utilisation. Pour ce faire, nous utilisons des diagrammes d'activités où les sommets représentent les cas d'utilisation et les arêtes les liens séquentiels entre eux (figure 2). Une arête, dans un tel diagramme, signifie que le cas d'utilisation situé en queue de spécification (*tail*) doit être exécuté après le cas d'utilisation en tête de spécification (*head*).
Pendant la phase de test, avant l'analyse des scénarios des diagrammes de séquence, ces diagrammes d'activités doivent être parcourus pour obtenir la séquence qui permettra de tester les cas d'utilisation.

C. Diagramme de séquence

En UML, un diagramme de séquence représente des interactions entre objets par échanges de messages et suivant un axe temporel. Les objets, comme dans les diagrammes d'activités, sont des instances de classes décrites dans le diagramme de classes. Généralement, les diagrammes de séquence décrivent un seul scénario. Dans notre approche, on énumère les messages comme cela est décrit dans [3], de telle sorte que l'on peut illustrer plusieurs scénarios conditionnels dans le même diagramme de séquence. De ce fait, les lettres en majuscules représentent les alternatives (les messages d'erreur). Ce choix, nous permet de dériver plus facilement les différentes séquences de messages correspondant à un même cas d'utilisation. La figure 3 montre le diagramme de séquence pour le cas d'utilisation `"Login"`.

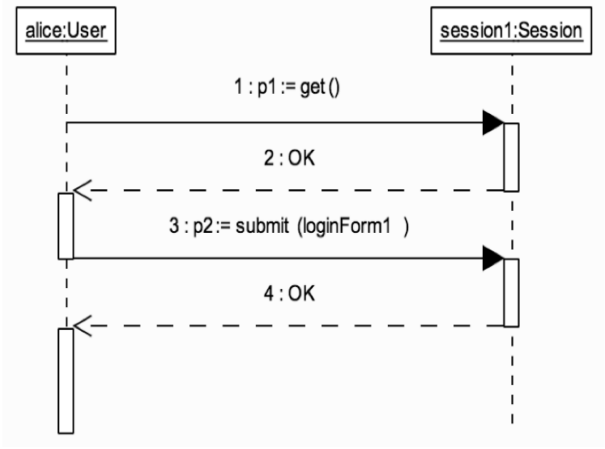

Fig. 3. Diagramme de Séquence pour le cas d'utilisation "Login".

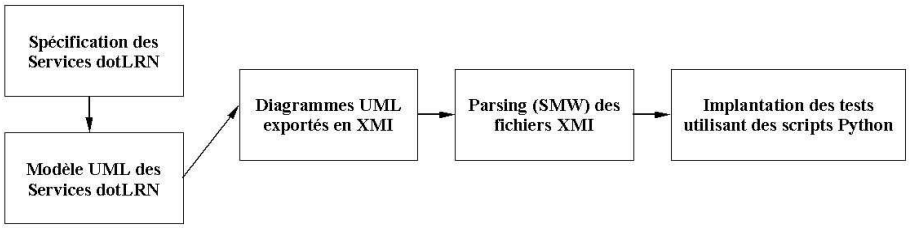

Fig. 1. Aperçu de notre méthodologie.

Entité Web	Méta-modèle UML	Valeurs étiquetées
page web	≪web page≫ class	**TitleTag :** le titre de la page **RelativeURL :** l'URL relative de la page **BodyTag :** l'ensemble des attributs pour le tag <body>
page de scripts	≪web page≫ class operations	
page des variables	≪web page≫ class attributes	
lien	≪link≫ association	
formulaire	≪form≫ class	**Method :** GET or POST
types de formulaires	≪form≫ class attributes stéréotypé par ≪form input≫	
portail	≪portal page≫ class	**TitleTag :** le titre de la page **RelativeURL :** l'URL relative de la page **BodyTag :** l'ensemble des attributs pour le tag <body>
éléments du portail	≪portal element≫ class	**BelongTo :** le ≪portal page≫ auquel il appartient

TABLE I

ASSOCIATION DES APPLICATIONS WEB AUX DIAGRAMMES DE CLASSES UML. LES STÉRÉOTYPES SONT REPRÉSENTÉS ENTRE ≪ ≫.

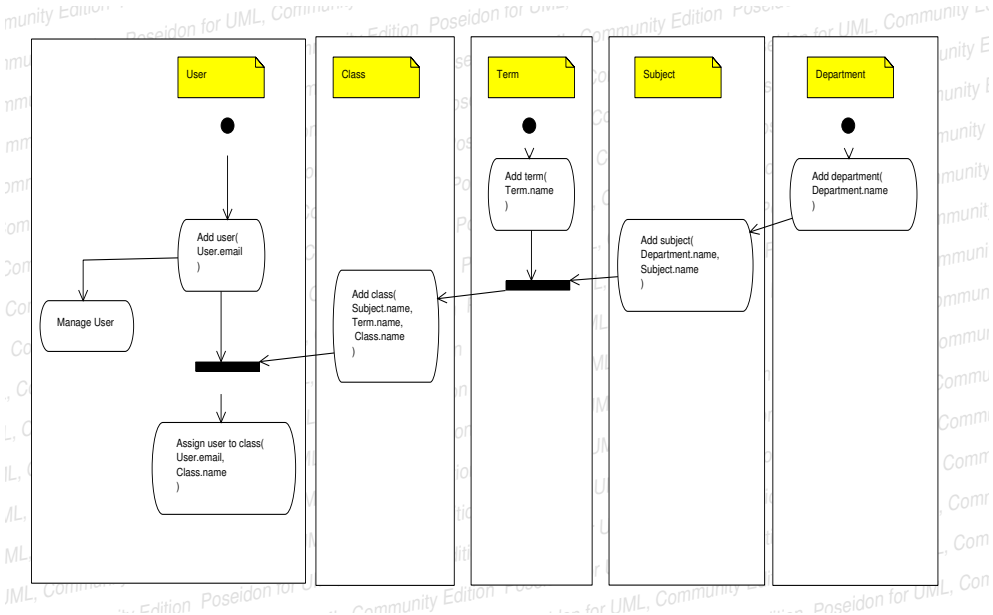

Fig. 2. Diagramme d'Activités des dépendances pour le cas d'utilisation"Assign user to class".

Notre diagramme de séquence est également paramétré, car les paramètres d'entrée peuvent avoir une influence pendant l'exécution et former des choix séparés [2]. Un exemple de paramètre peut être une adresse email d'un utilisateur. Soit cet email est celui d'un utilisateur enregistré (il existe dans la base de données), soit il correspond à un nouvel utilisateur (il n'apparaît pas dans la base de données), ce paramètre va déterminer ce qui se produira par la suite. Dans le premier cas, la page d'accès aux services de dotLRN est affichée sinon un avertissement apparaît dans la page de "Login". Pendant

requêtes HTTP	réponses HTTP possibles
navigate(url :String) : L'utilisateur fait une requête HTTP pour une URL	**display(page :WebPage)** : Le serveur Web retourne la page souhaitée
link(target :String) : L'utilisateur clique sur un lien web	**display(edge :WebPage)** : Le serveur Web retourne la page cible
submit(page :WebPage, form :Form, data :List) : L'utilisateur soumet un formulaire	**display(page :WebPage)** : Dans le cas d'une entrée valide, le serveur web répond avec une nouvelle page Web **error(msg :String)** : En cas de mauvaises entrées, le serveur retourne la page précédente avec un warning

TABLE II
ACTIONS DU DIAGRAMME DE SÉQUENCE.

la procédure de test, s'il existe de telles alternatives et paramètres, le programme produit devra prendre en considération toutes ces possibilités.
La Table II résume les actions utilisées par le diagramme de séquence contenant des requêtes HTTP de l'utilisateur et des possibles réponses retournées à l'utilisateur par le serveur, car le succès ou l'échec de nos tests dépend de ces requêtes et des réponses respectives. Puisque le système sous test est une application web, il y a trois possibilités pour l'utilisateur : soit il navigue vers une URL, soit il fait appel à une page à partir d'une autre page (en cliquant sur un lien), ou il soumet des informations en remplissant un formulaire HTML. Le système répond soit directement en retournant la page demandée (display) ou en retournant un message d'erreur.

V. AUTOMATISER LA GÉNÉRATION DES SCÉNARIOS DE TEST

Afin d'automatiser la génération des tests, notre but est avant tout de parser les diagrammes UML obtenus à partir des phases précédentes. Ainsi, en se basant sur ces diagrammes, nous générons les requêtes nécessaires vers le serveur de services Web (dotLRN) et vérifions alors si ses réponses sont conformes aux réponses présentes dans les modèles présentés ci-dessus. Puisque dotLRN est une plate-forme d'applications web, les requêtes sont des requêtes HTTP qui simulent un utilisateur naviguant sur le site à travers un navigateur. Les actions possibles sont le remplissage (*fill*), la soumission (*submit*) d'un formulaire, le click sur un lien ou la navigation vers une URL donnée. De même, les réponses du serveur sont des réponses HTTP qui peuvent contenir soit une page HTML, une redirection vers une autre URL ou un message d'erreur. En supposant que nous traitons le premier cas, la page HTML de la réponse doit être parsée afin de savoir si son contenu est celui attendu dans la spécification.
Selon ces besoins, nous devons choisir les composants nécessaires à la construction de notre suite de tests. Pour cela, nous présentons dans la suite les instruments permettant l'analyse syntaxique et l'exécution des diagrammes UML. Puis nous illustrons le parsing des pages HTML reçues de notre serveur. Enfin nous proposons une solution complète permettant la validation des services web de dotLRN.

A. Le langage de programmation pour le parsing

Afin d'implanter un analyseur syntaxique et d'exécuter les diagrammes UML, il est judicieux de choisir un langage orienté objet. Le langage choisi se doit également de permettre une gestion facilitée des messages et ayant un haut niveau d'abstraction pour les communications réseaux. Ainsi, le langage le plus approprié que nous avons choisi est Python [9]. Python est un langage orienté objets qui combine une sémantique importante et une syntaxe claire. Les modules utilisés dans Python fournissent de nombreuses fonctions qui facilitent la gestion des chaînes de caractères et du réseau.

B. Analyse syntaxique et exécution des diagrammes UML

Afin de parser les diagrammes UML, nous pouvions utiliser les API (Application Programming Interface) d'un outil UML ou exporter les diagrammes dans un format XMI qui permettrait de les parser en utilisant un parser XML (eXtensible Markup Language). XMI est un standard créé par l'Object Management Group (OMG) pour représenter les diagrammes UML en XML. La solution retenue est alors d'exporter les diagrammes en XMI afin de ne pas être dépendants d'un outil spécifique.
Bien que nous puissions utiliser un parser XML quelconque pour parser les fichiers XMI, en raison de la complexité du standard nous avons décidé d'utiliser un parser XMI particulier. De fait nous avons utilisé le parser inclus dans l'outil SMW[3] . Ce dernier est gratuit, open-source et également écrit en Python, permettant ainsi de l'intégrer plus facilement à notre code. Puisqu'il est open source, il nous permet de résoudre des problèmes de compatibilité qui apparaissent quand nous produisons le code XMI avec notre outil de génération XMI Poseidon[4] .

C. Parser les pages HTML

Puisque les pages HTML mélangent des données de présentation et des données de contenu, le code HTML en sortie du serveur ne nous permet pas d'extraire les informations souhaitées sans regarder avant tout les détails d'implantation. Pour éviter cela, il est nécessaire de modifier la page des templates des services Web afin de fournir les données de manière clairement définie. Ceci est réalisé en ajoutant des attributs *id* aux étiquettes que nous souhaitons parser. Par exemple, dans certaines pages, à l'étiquette *td* qui contient le nom de l'utilisateur, nous insérons un attribut *id=" username "*. Ainsi, nous pouvons questionner n'importe quelle page indépendamment de sa structure.

[3]System Modelling Workbench, http ://www.abo.fi/ iporres/html/smw.html
[4]http ://www.gentleware.com/products/descriptions/se.php4

D. Exemple d'implantation

Dans cette section, nous donnons le squelette de notre solution permettant de parser le code XMI et les pages HTML retournées par le serveur, puis de les comparer pour valider ou non la conformité des services selon la spécification. Dans le code qui suit, nous avons exclu certaines parties (essentiellement des fonctions) afin de réduire la taille et de clarifier le code.

```
1  from urllib import urlopen
2  from smw.io import XMIStreamer
3  from smw.metamodel import UML14
4
5  class TestSuite:
6      returnedPage = None
7
8      def validateLink(self, link):
9          operation = self.getOperation(link)
10
11         if operation.name == "navigate":
12           url = self.getOperationParameters(operation)
13           fd = urlopen(serverbase + url)
14           self.returnedPage = fd.read()
15           fd.close()
16
17         elif operation.name == "display":
18           params = self.getOperationParameters(operation)
19           pageTemplate = generatePageTemplate(params)
20
21           parser = dotHTMLParser(pageTemplate)
22           parser.feed(returnedPage)
23
24     def execute(self, source):
25         xmi = XMIStreamer(UML14)
26         fd = open(source, "r")
27         model = xmi.loadFromStream(fd)
28         fd.close()
29
30         sequenceDiagram = self.getSequenceDiagram(model)
31         for link in self.getLinks(sequenceDiagram):
32             self.validateLink(link)
```

La fonction *execute* (ligne 24) est la fonction principale et elle lit le code XMI à partir du fichier défini dans la variable *source* donnée en paramètre. Elle isole de fait le diagramme de séquence (dans cet exemple nous supposons qu'un seul diagramme existe) et valide alors un à un tous les messages (*link*). Toutes les fonctions *getX* (comme *getSequenceDiagram* et *getLinks*) sont triviales et consistent à naviguer à travers les structures générées par SMW pour obtenir une donnée spécifique souhaitée. Il est supposé qu'elles sont définies dans la classe. C'est à travers cette fonction *execute* que notre suite de tests est générée. Ce sont tous les liens obtenus du code XMI ainsi que les opérations qui y sont liées qui permettent la validation des fonctionnalités du service web. Par ailleurs, comme nous le notons dans la suite, la génération des tests et leur exécution sur les pages retournées par le serveur sont entièrement automatisés.

La validation de chaque lien dépend de l'opération. Si l'opération est *navigate* (ligne 11-15) alors nous devons extraire l'URL de destination à partir des paramètres et obtenir la page demandée. Nous supposons que la destination est une URL relative, de fait nous utilisons la variable *serverbase* (ligne 13) pour la rendre absolue. La page retournée est ainsi conservée par la variable *returnedPage* afin d'être utilisée par les commandes qui suivent.
Dans le cas d'une opération *display* (ligne 17-22), nous créons un modèle de la page basé sur les paramètres des opérations et nous utilisons alors notre parser HTML pour comparer *returnedPage* avec le modèle. Le squelette du parser HTML est comme suit.

```
1  from HTMLParser import HTMLParser
2
3  class dotHTMLParser(HTMLParser):
4      pageTemplate = None
5
6      def __init__(self, pageTemplate):
7          self.pageTemplate = pageTemplate
8
9      def handle_starttag(self, tag, attrs):
10         for attr in attrs:
11             if "id" in attr:
12                 validateElement(tag, attr)
```

La classe *dotHTMLParser* hérite de la classe *HTMLParser* et annule la fonction $handle\_starttag$ pour rechercher les éléments qui ont un attribut *id*. Chaque élément sera validé selon la *pageTemplate* donnée lors de l'instantiation (les détails de la validation ne sont pas donnés ici).
De manière similaire aux deux opérations illustrées dans l'exemple, nous pouvons écrire le code pour traiter le reste des opérations supportées.

VI. Insertion de données dans notre modèle UML

Afin d'exécuter nos tests et d'observer le comportement du système, il est nécessaire de substituer les paramètres formels par les valeurs réelles. Par exemple, dans le cas où nous voulons tester si un utilisateur avec un login et un mot de passe valide peut correctement se connecter au système, nous devons fournir au test une paire valide d'email et de mot de passe. Dans les travaux existants où l'exécution se fait manuellement, le testeur fournit cette entrée de manière active i.e. à travers l'interface avant l'exécution de chaque test. Dans notre cas, puisque nous nous intéressons à la génération et l'exécution automatique de scripts, il nous est nécessaire d'intégrer ces informations dans notre modèle.
Nous présentons dans ce qui suit deux approches permettant de résoudre cette difficulté.

A. Utiliser OCL pour modéliser les données

Dans [10], l'OMG définit un ensemble de données comme étant une collection de valeurs qui est utilisée par des procédures de test comme une source de valeurs pour l'exécution de cas de tests. Malheureusement, il n'est pas mentionné comment ces valeurs peuvent être intégrées dans le modèle mais il est juste proposé qu'elles peuvent être représentées comme des contraintes. Dans [11], les auteurs utilisent des contraintes OCL exprimées dans un modèle statique afin d'identifier les domaines de test à partir desquels les cas de test sont sélectionnés. Cependant cette approche nécessite des efforts importants pour trouver ces domaines et aussi pour y extraire des valeurs aléatoires. Cependant, nous pensons qu'il est possible d'utiliser des expressions OCL afin

d'affecter des valeurs aux attributs et paramètres et les relier aux entités du modèle. En OCL, grâce à des expressions telles que *invariants* et *init*, nous pouvons affecter des valeurs à un attribut mais puisque ces contraintes doivent toujours être vraies, toutes les instances d'une entité prendront ces valeurs. Ceci est différent de ce dont nous avons besoin car nous souhaitons définir différentes valeurs pour différentes instances. Par exemple, nous avons besoin de deux instances d'une classe utilisateur ayant des valeurs différentes pour le login et le mot de passe.

Cependant, dans une expression "let", nous pouvons définir une variable qui contient toutes les valeurs. Puisque nous souhaitons que cette variable soit visible hors de nos contraintes spécifiques, nous donnons les valeurs dans une contrainte "definition". Selon la spécification, une contrainte "definition" doit être attachée à un classifieur. Toutes les variables et opérations définies dans cette contrainte sont connues dans le même contexte où les propriétés du classifieur peuvent être utilisées. De plus, OCL 2.0 introduit la notion de Tuples dans lesquels plusieurs valeurs peuvent être composées. En les combinant, nous pourrions associer un classifieur avec un nombre de valeurs de différents types qui représentent une instance. Par exemple, dans le cas où nous avons défini une classe "User" avec deux attributs "email" et "âge" et nous voulons représenter deux instances d'utilisateur "user1" et "user2", nous pourrions définir les expressions OCL suivantes :

```
context User
def: let user1 = Tuple{email: String = 'Fatiha',
                       age: Integer = 12}
     let user2 = Tuple{email: String = 'Stephane',
                       age: Integer = 5}
```

L'un des avantages d'OCL est qu'il est lié aux types des diagrammes UML, ce qui convient fortement à notre modèle. De plus, il est aisé de vérifier la correction syntaxique et la cohérence du modèle en utilisant soit des outils OCL intégrés aux outils CASE, ou en utilisant des parsers OCL plus sophistiqués. Par ailleurs, les expressions OCL apparaissent dans la représentation XMI du modèle. Cependant, les Tuples sont seulement introduits dans OCL 2.0 et d'autres outils CASE comme Poseidon[5], mais pas dans ArgoUML[6].

B. Modéliser les données avec les Profils UML

Une autre approche pour représenter les données est d'utiliser un Profil UML. Les profils UML sont des extensions des mécanismes fournis par UML qui nous permettent d'ajouter des services spécifiques. Les profils UML sont implantés en utilisant des stéréotypes et des valeurs étiquetées.
Différentes solutions peuvent être proposées pour la conception de profils UML appropriés, mais notre solution est orientée vers la simplicité et la facilité d'implantation. Quand nous souhaitons représenter une instance **I** d'une classe **C**, nous pouvons définir **I** comme une nouvelle classe ayant les mêmes attributs et opérations que **C**. Chaque attribut de **I** aura une valeur étiquetée contenant la valeur actuelle de l'attribut de **C** respectif. Si nous voulons une valeur étiquetée en référence à une instance **J** d'une autre classe, alors nous utilisons la notation **/J**. Par ailleurs, afin de montrer que **I** est du type de **C**, il aura le stéréotype $\ll$C$\gg$.
Cette méthode nécessite des efforts de développement supplémentaires pour vérifier la correction des valeurs données. Cependant, elle peut être utilisée avec d'autres versions d'UML qui sont mieux supportées par la plupart des outils open source CASE actuellement disponibles.

VII. PERSPECTIVES

L'outil logiciel présenté dans les sections précédentes réalise le parsing des spécifications UML, la génération et la traduction des tests dans le format XMI et son exécution sur la plate-forme dotLRN. La méthode que nous proposons analyse et exécute automatiquement les diagrammes UML. Cependant, il aurait été possible aussi de générer le code du test spécifié. Le code généré aurait pu s'écrire dans un langage quelconque et aurait eu l'avantage de pouvoir être utilisé pour exécuter les tests à partir de tout ordinateur sans utiliser notre logiciel. Evidemment, un effort de développement beaucoup plus important aurait été nécessaire, mais comme il est indépendant de la conception globale, il peut être laissé pour des étapes ultérieures de ce travail.

Concernant le test, dans une première étape, nous commençons par le test des interfaces de communication et les besoins des utilisateurs. L'étape suivante sera le test des mécanismes d'authentification et les contenus des applications (par exemple, tester le contenu d'une page web demandée). Pour ce faire, nous avons besoin de traiter les données au niveau de la spécification. En réalité, pour tester si un utilisateur a l'autorisation d'accéder au système, nous avons besoin de tester s'il entre son propre " login " et son mot de passe. Pour traiter les données, nous avons proposé deux approches : une basée sur OCL et l'autre sur les profils UML. Cependant, les deux approches présentent des avantages et des inconvénients et nous envisageons d'étudier plus en détail les deux solutions.

Une autre question, que nous avons étudiée, est en relation avec une nouvelle méthode de test. La méthode, présentée dans cet article, est basée sur une spécification UML du système à tester. Les résultats d'expérimentations avec cette approche sont très prometteurs. L'automatisation de la procédure de test réduit le temps et le coût, et permet de produire des outils plus fiables pour la conception et le développement des services web. De plus, les tests des besoins des utilisateurs contribuent à la conception des outils avec des interfaces plus conviviales et faciles à utiliser.

Cependant, nous voulons proposer une autre approche pour l'automatisation de la génération de tests et leur exécution. Cette méthode n'a pas d'interactions avec le système sous test comme la méthode présentée dans cet article. Cette nouvelle méthode est bien adaptée pour les concepteurs de services qui veulent gagner du temps dans la phase d'analyse. En outre,

[5]http ://www.gentleware.com/index.php
[6]http ://argouml.tigris.org/

elle permet d'éviter de mobiliser le serveur web pour la phase d'exécution des tests car elle est non intrusive. L'approche est basée sur la vérification des propriétés des services directement sur les traces d'exécution de ces services. Avec cet objectif, nous installons un "sniffer" entre le client et le serveur, afin d'obtenir de manière dynamique toutes les pages html de chaque requête de service. Au niveau de la spécification, nous associons à chaque classe et à chaque opération, des propriétés qui doivent être vérifiées une fois que les services ont été exécutés. Ces propriétés sont décrites en utilisant le langage OCL et représentent les besoins du service web que nous voulons tester. Ce langage ne peut pas être utilisé directement. Nous avons besoin de le traduire en un autre format afin de le tester sur les traces. Ainsi, nous traduisons les propriétés exprimées en OCL en expressions régulières. Par la suite, ce sont ces expressions que nous allons évaluer sur les traces d'exécution réelles du système. A l'INT, nous avons développé un outil pour effectuer cette vérification [12].

Cette approche, qui utilise des contraintes exprimées en OCL, permet de définir de manière formelle des fonctionnalités que les services doivent fournir. De plus, il peut être utilisé pour le test de nouveaux services. En l'utilisant, nous pouvons tester si un service qui vient d'être ajouté (i.e. une nouvelle fonctionnalité) travaille correctement par rapport aux contraintes exprimées en OCL. De plus, nous sommes capable de tester la préservation des services existants, c'est-à-dire, de réaliser des tests de non régression en réutilisant des contraintes OCL déjà testées.

VIII. Conclusion

Nous avons présenté dans cet article une méthodologie pour le test de services web, en particulier des services d'enseignement à distance (de la plate-forme dot LRN). Cette plate-forme présente comme avantage le fait de proposer un ensemble d'outils logiciels open source, qui permet de construire des applications web de taille réelle. L'application de notre méthodologie à cette plate-forme illustre son passage à l'échelle.

Par ailleurs, même si la méthodologie de test et l'outil logiciel développés dans le cadre de ce travail ont été appliqués à un serveur de services web dotLRN, ils sont génériques et peuvent donc s'appliquer facilement à d'autres environnements de services web. La méthodologie proposée couvre toutes les étapes du test : la spécification du service en UML, la génération de tests et leur exécution sur le serveur. Les tests sont sélectionnés en tenant compte des avis des experts et des concepteurs ; ils sont basés sur les cas d'utilisation et couvrent tous les aspects relevants du comportement du système. Nous pouvons également garantir une couverture complète des erreurs des tests sélectionnés. De plus, deux approches ont été proposées pour le traitement des données, que nous avons commencées à intégrer dans notre outil.

Les résultats obtenus dans le cadre du projet E-LANE ont montré l'intérêt de l'utilisation des méthodologies basées sur des méthodes formelles : elles facilitent la génération de tests et permettent de réutiliser les tests pour tester de nouveaux services. En effet, des tests pour les nouveaux services peuvent être obtenus en modifiant les tests déjà générés pour des services présentant des traits communs. Ces méthodologies facilitent également l'exécution de tests de non- régression.

References

[1] "UML Object Constraint Language (2.0 submission)," http ://www.omg.org/cgi-bin/apps/doc ?ad/03-01-07.pdf.

[2] F. Basanieri, A. Bertolino, and E. Marchetti, "The Cow_suit Approach to Planning and Deriving Test Suites in UML Projects," *Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002, LNCS 2460, Dresden, Germany*, pp. 383–397, September 2002.

[3] L. C. Briand and Y. Labiche, "A UML-Based Approach to System Testing," *Software and Systems Modeling*, vol. 1, no. 1, pp. 10–42, 2002.

[4] A.J.Offutt and A.Abdurazik, "Using UML Collaboration Diagrams for Static Checking and Test Generation," *Proc. Third International Conference on the UML (UML'00), York, UK*, pp. 383–395, October 2000.

[5] J. Conallen, *Building Web Applications With UML*, 2nd ed. Addison-Wesley, 2002.

[6] J.Conallen, "Modelling Web Application Architectures with UML," *Communications of the ACM*, vol. 42, no. 10, pp. 63–70, 1999.

[7] N. Koch, H. Baumeister, R. Hennicker, and L. Mandel, "Extending UML to Model Navigation and Presentation in Web applications," in *Proc. of Modeling Web applications in the UML Workshop, UML2000*, 2000.

[8] A. Cockburn, *Writing Effective Use Cases*, 1st ed. Addison-Wesley, 2000.

[9] J. D. Christopher A. Jones, Fred L., *Python & XML*, 1st ed. O'Reilly & Associates, 2001.

[10] "UML Testing Profile (Draft Adopted Specification)," http ://www.omg.org/docs/ptc/03-07-01.pdf.

[11] M. Benattou, J.-M. Bruel, and N. Hameurlain, "Generating Test Data from OCL Specification," in *Proceedings of the ECOOP'2002 Workshop on Integration and Transformation of UML models (WITUML'2002)*, 2002.

[12] E. Bayse, A. Cavalli, M. Nunez, and F. Zaidi, "A passive Testing Approach based on Invariants : Application to the WAP," *Computer Networks journal*, no. 48, pp. 247–266, 2005.