UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

VALIDATION ET ENRICHISSEMENT DE LA QUALITÉ DES APPLICATIONS WEB

MÉMOIRE
PRÉSENTÉ
COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN INFORMATIQUE

PAR
ERIC LAFLEUR

AOÛT 2008

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS
Département d'informatique et d'ingénierie

Ce mémoire intitulé :

VALIDATION ET ENRICHISSEMENT DE LA QUALITÉ DES APPLICATIONS WEB

présenté par
Éric Lafleur

pour l'obtention du grade de maître en science (M.Sc.)

a été évalué par un jury composé des personnes suivantes :

Dr. Michal Iglewski Directeur de recherche
Dr. Luigi Logrippo Président du jury
Dr. Kamel Adi Membre du jury

Mémoire accepté le : 29 août 2008

Remerciements

J'adresse mes remerciements à ma famille pour leur support morale et à mon directeur de recherche professeur Michal Iglewski pour tous les précieux conseils, les suggestions et les encouragements qu'il m'a donnés tout au long de ce mémoire.

Résumé

Le contrôle de la qualité des produits et services est une phase importante que l'on retrouve dans la majorité des institutions. Dans les entreprises de conception de logiciels, un contrôle de la qualité des produits est toujours exercé. Ce contrôle est parfois fait de façon informelle surtout dans les petites organisations. Les entreprises de plus grande taille rédigent normalement un ensemble de documents qui accompagnent et décrivent les produits qu'ils conçoivent. Les critères de tests sont souvent définis à partir de ces documents. Ils utilisent, par exemple, les spécifications des besoins pour définir les critères de qualité de leurs logiciels. La documentation dans le domaine de la conception des applications Web (AW), lorsqu'elle existe, est en général élémentaire. Cette façon de faire complique la tâche des analystes de l'assurance de la qualité des AW.

Nous présentons dans notre mémoire une approche qui permet la validation et l'enrichissement de la qualité des AW. Nous favorisons les méthodes automatiques et semi-automatiques au détriment des méthodes manuelles. Nos méthodes se veulent assez générales pour être applicables à la majorité des AW. Nous sommes intéressés par la qualité des AW à tous les différents stades de leurs existences c.-à-d. avant, pendant et après leur création.

Abstract

Quality assurance control of products and services is an important step found in most of the institutions. In the software programming businesses, a quality control is always done. This control is sometimes done informally especially in the smaller organisations. The large businesses normally write documents describing the products they're creating. Testing criteria's are often defined based on those documents. For example, they sometimes use the specification to define the quality criteria's of there software's. When available, the documentation in the Web application domain is generally limited. This way of doing things complicates the task of the Web application quality assurance analysts.

In our thesis we present a method to validate and upgrade the quality of Web applications. We prefer automatic and semi-automatic methods over manual methods. Our methods are general enough so that they can be applied to the majority of Web applications. We're interested in the quality assurance of the Web applications at all the different steps of their life cycle including before, during and after their creation.

Table des matières

Table des figures

Chapitre 1

Introduction

1.1 Applications Web

La quantité d'applications Web (AW) augmente sans arrêt et ce à un rythme exponentiel. Elles occupent une place de plus en plus importante dans nos vies. Plusieurs d'entre nous ne pourrions pas fonctionner normalement si nous n'avions pas accès à ces AW. Nous dépendons de plus en plus de celles-ci dans l'accomplissement de nos tâches quotidiennes que se soit en tant qu'individu, entreprise, gouvernement ou autres. Voici quelques exemples d'AW qui ont un impact important sur nos vies : les services de messagerie électronique accessibles à l'aide d'un fureteur, les transactions bancaires faites en ligne, les achats fait par Internet, etc. Malgré le grand nombre d'AW déjà existantes, ce nombre devrait croître de façon importante à court terme puisque l'Internet, sur lequel la vaste majorité des AW sont utilisées, n'a commencé à être utilisé par le grand public qu'au début des années 1990. Depuis ces débuts la vitesse de transmission disponible pour un simple usager est passée de quelques kilobits par seconde à des dizaines de mégaoctets par seconde. Cette vitesse de transmission accrue fait augmenter les utilisations possibles d'Internet et par le fait même le nombre d'AW. Même les pays en voie de développement commencent eux aussi à utiliser de plus en plus les AW. Ceci est possible en partie grâce à des projets tel que celui de MIT de fournir un ordinateur portable par enfant. Ils ont conçu l'ordinateur portatif XO qui grâce à son faible coût, sa batterie rechargeable par manivelle et son interface sans fil devient très intéressant pour les pays les plus pauvres. En ce début du vingt et unième siècle, à l'ère

de la numérisation grandissante de l'information partout dans le monde, la validation et l'enrichissement de la qualité des AW deviennent une nécessité.

1.2 Définition du problème

La plupart des AW sont créées et modifiées alors qu'il n'y a aucun plan d'assurance de la qualité de mis en place. Normalement plusieurs personnes travaillent à la création d'AW et ces mêmes personnes font quelques tests fonctionnels informels pour ensuite publier ces AW. Il est difficile de prévoir, dans ces conditions, de quelle façon ces AW réagiront vis à vis différentes situations d'utilisation une fois qu'elles seront mises en fonction.

Pour ajouter au problème, les spécifications se font très rares dans le domaine des AW. Il est difficile voir impossible dans certain cas d'écrire des jeux de tests dans l'absence d'une spécification. Lorsqu'une spécification existe elle est souvent difficile à lire et primitive. De plus, bien que la majorité des AW de moyenne et grande envergure sont en constante évolution, les spécifications lorsqu'elles existent ne sont pas toujours maintenues à jour. Dans les cas où une spécification est inexistante ou désuète, il est difficile de faire des tests pour vérifier si une AW répond aux attentes puisque ces attentes ne sont pas clairement définies.

Nous expliquons dans le mémoire qu'il existe des modèles de conception d'AW qui permettent de construire des AW sans que l'écriture d'une spécification soit nécessaire. En fait, certains de ces modèles génèrent automatiquement l'écriture d'une spécification lorsque compilés pour créer une AW. Prenons l'exemple d'un des modèles de conception que nous présentons soit le modèle WebRatio. Au moment de la génération d'une AW ce modèle crée une spécification dans un fichier XML. C'est un concept intéressant, par contre, seules les personnes possédant une grande expertise du modèle WebRatio arrivent à bien comprendre le contenu de cette spécification.

Une plus grande place doit être faite à l'assurance de la qualité dans le développement des AW. Les répercussions de la négligence de la qualité des AW peuvent être, entre autres, des pertes économiques importantes. Une standardisation de la façon de concevoir et d'entretenir les AW, pour le maintien d'un certain niveau de qualité, serait dans le meilleur intérêt de nous tous en tant que société.

Plusieurs personnes se sont déjà intéressées à ce problème mais ce sujet de recherche en est encore au stade embryonnaire. La création d'AW est

un domaine récent où il n'existe pas de tradition d'assurance de la qualité comme celle qu'on retrouve dans la programmation traditionnelle de logiciels. Contrairement à ce qu'on voit dans l'industrie de conception de logiciels, on parle rarement de cycle de vie et encore moins de document des besoins lorsqu'on fait référence aux AW. La plupart du temps, les concepteurs d'AW ont une idée générale de ce qu'ils doivent concevoir et ils se fient à leurs talents et expériences pour y arriver. Très rare sont ceux qui se fixent des critères de qualité à atteindre avant de commencer à coder. Sans objectifs clairement définis, il est difficile d'évaluer la qualité d'une AW.

Dans ce mémoire nous proposons de traiter la validation et l'enrichissement de la qualité des AW. Plusieurs articles analysent et apportent des idées pour améliorer la qualité des AW. Différentes méthodes et outils sont exposés dans ces articles. Après s'être inspiré des articles les plus intéressants nous avons développé une méthodologie pour valider la qualité des AW. Nous expliquons de façon détaillée les différentes étapes à suivre pour valider une AW. Notre méthode tient aussi compte de l'enrichissement possible de la qualité des AW. Notre méthode est utilisable par la majorité des AW.

1.3 Structure du mémoire

Au deuxième chapitre nous discutons du contrôle de la qualité des logiciels. L'importance de ce type de contrôle est démontrée en examinant trois modèles de cycle de vie connus. Nous enchaînons en analysant la diversité présente entre AW et logiciel traditionnel et son impact sur le contrôle de la qualité. Au troisième chapitre nous parlons des critères de qualité applicables aux AW. Le chapitre débute par un exposé de quelques articles sur le sujet. Nous expliquons ensuite les critères de qualité retenus pour le mémoire et les raisons qui ont motivé nos choix.

Le quatrième chapitre est consacré aux différents types de tests. Nous introduisons ce chapitre par une présentation des différentes catégories de test. Suit une brève description de chacune des catégories de test accompagnées de leurs objectifs. Nous étudions par la suite les particularités des AW les distinguant des logiciels traditionnels.

Dans le cinquième chapitre nous discutons des modèles de conception d'AW. Nous examinons plus en détail trois modèles. Pour deux de ces modèles nous avons conçu une AW de vente de livres en ligne. Cet exercice nous a permis de mieux nous familiariser avec ces modèles.

Dans le sixième chapitre nous étudions différentes méthodes d'évaluation de la qualité des AW. Deux approches sont étudiées plus en détail : le modèle à cinq dimensions [1] et le modèle 2QCV3Q [2].

Au septième chapitre nous nous attardons aux spécifications et à leurs impacts sur la qualité des AW. Trois méthodes d'écriture de spécification Web sont analysées plus en détail.

Au huitième chapitre nous élaborons sur différents types de test permettant d'évaluer la qualité des AW. Les types de test sont séparés en trois groupes soit les tests de type boîte noire, boîte transparente et boîte grise.

Finalement nous présentons une discussion des principaux résultats et nous proposons une méthodologie pour valider et enrichir la qualité des AW.

Chapitre 2

Contrôle de la qualité des logiciels

2.1 Introduction

Dans l'industrie du développement des logiciels, le contrôle de la qualité occupe une place importante. Pour s'en convaincre nous n'avons qu'à regarder quelques-uns des modèles de cycle de vie de logiciels les plus connus. Dans les trois modèles présentés ici des tests sont exécutés pour s'assurer de la qualité des logiciels produits. Prenons pour commencer le modèle en cascade [3]. Sur la figure 2.1 nous voyons que l'avant dernière étape est consacrée aux tests. Nous parlons ici de tests unitaires et d'intégration.

Le modèle du cycle de vie en spirale [4] requière lui aussi l'exécution de plusieurs tests. Sont exécutés des tests unitaires, d'intégration, d'acceptation et de performance. Nous avons une représentation de ce modèle à la figure 2.2.

Le dernier modèle de cycle de vie que nous présentons est le modèle en V [5]. Ce modèle est un bon exemple de l'importance que peuvent prendre les tests dans le développement d'un logiciel pour contrôler la qualité de celui-ci. Dans ce modèle (figure 2.3) à chacune des étapes des tests sont exécutés.

2.2 Application Web vs logiciel traditionnel

Contrairement aux applications traditionnelles nous n'utilisons pas les modèles de cycle de vie pour la conception d'AW. L'approche différente, du

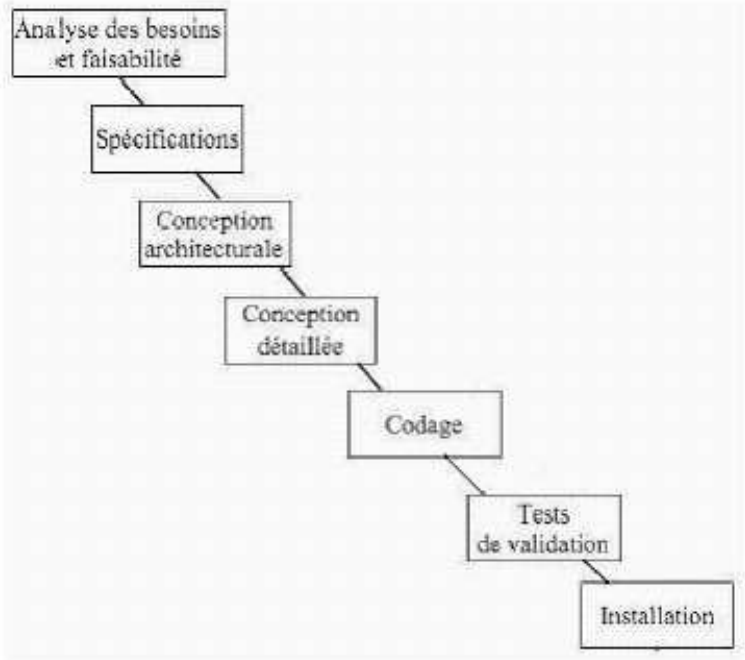

FIG. 2.1 – Modèle en cascade

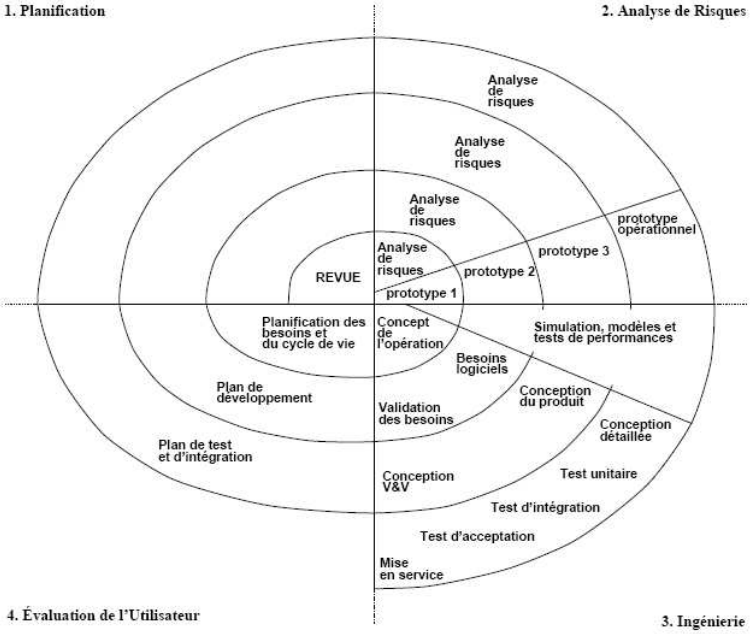

FIG. 2.2 – Le cycle de vie en spirale

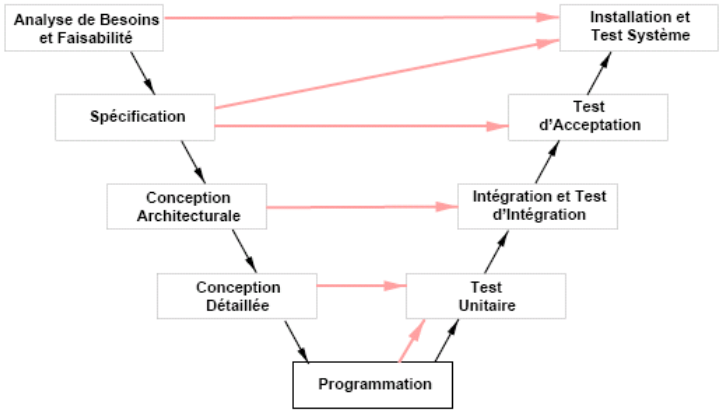

FIG. 2.3 – Le cycle de vie en V

point de vue du contrôle de la qualité, utilisée face aux AW versus la programmation traditionnelle de logiciels est due en partie aux disparités qui existent entre ces deux. Elbaum, Karre et Rothermel [6] notent trois différences principales entre ces deux méthodes de programmation. Premièrement le niveau d'utilisation d'une page Web peut changer très rapidement. Une page Web peut être visitée que par quelques personnes par jour et soudainement devenir très populaire, car nouvellement inscrite dans un engin de recherche par exemple, et être visitée par des centaines de millier d'utilisateurs. Une autre différence rapportée par les auteurs de cet article est le niveau de changement qui est normalement beaucoup plus grand dans l'univers des AW. Ce sont habituellement des petits changements graduels. Finalement dans l'article [6] les auteurs relèvent que dans les AW nous retrouvons normalement des architectures multiniveau (multi-tier) et hétérogènes qui incluent des serveurs Web, des serveurs d'application et des serveurs de bases de donnés.

Rappelons que les AW sont dépendantes des différents fureteurs qui ont chacun leurs particularités [7]. Les AW intègrent normalement plusieurs technologies différentes provenant de différents constructeurs. Il est alors difficile de prévoir comment ceux-ci vont réagir dans différentes situations.

Une autre différence notable est que les AW sont plus focalisées sur les documents et l'information tandis que la majorité des logiciels traditionnels sont focalisés sur les capacités informatiques [8]. Les auteurs de l'article [8] mentionnent aussi que même si les capacités informatiques des AW ont évolué ces dernières années, la recherche et la récupération des documents demeurent l'activité principale de la majorité d'entre elles.

Dans le chapitre deux du livre de Nguyen, Johnson et Hackett [9] les auteurs énumèrent une liste d'éléments à être testée dans la majorité des AW et que l'on retrouve plus rarement dans les logiciels traditionnels : l'interface usager, l'intégration du système, la mise en place du client et serveur, l'aide en ligne, la configuration et la compatibilité, base de données, sécurité, performance, charge et stress.

Toutes ces différences font qu'il est difficile d'utiliser un modèle de cycle de vie pour concevoir une AW. Elle est normalement en constante évolution et les modifications doivent être faites rapidement. Dans ce domaine nous voulons obtenir des résultats concrets immédiatement. Nous trouvons rarement des concepteurs d'AW qui s'attardent à élaborer la documentation qui devrait normalement précéder la conception d'un logiciel. Dans la majorité des cas aucune documentation normalisée n'accompagne les AW tout au long de leur développement. Nous ne retrouvons pas cette façon d'opérer

qui consiste à bien documenter les AW comme c'est le cas dans le milieu des logiciels traditionnels.

Chapitre 3

Critères de qualité

3.1 Introduction

Le choix des critères de qualité est une étape importante dans l'analyse de l'évaluation de la qualité d'un logiciel. Ils forment la fondation de l'analyse de la qualité de ceux-ci. Nous devons définir ceux-ci avant que ne débute la conception d'un logiciel. Nous ferons référence à ces critères tout au long du processus de création du logiciel concerné.

3.2 Applications Web

Différents auteurs définissent leurs critères de qualité générale qui devraient, selon eux, se retrouver dans toutes les AW. Le moment à partir duquel nous devrions tenir compte de la qualité des AW diffère aussi d'un auteur à l'autre. Par exemple le modèle WebML [10] nous impose de tenir compte des critères de qualité avant de débuter l'écriture du code d'une AW. C'est un modèle de conception pour la création d'AW. L'analyse du schéma de conception (Design Schema Analysis) vérifie au moment de la conception la consistance et l'exactitude des spécifications [11]. L'outil WebQEM_Tool [12] est un autre outil qui oblige à définir les critères de qualité avant de débuter la conception d'une AW. D'autres outils tel que WebTest [13] et les méthodes présentées par Sant et al. [14] analysent la qualité d'AW déjà existantes.

Avec l'outil TestWeb [14] les critères de test doivent être définis par le contrôleur. Par contre si nous utilisons l'outil WebRatio [18] certains critères de qualité du modèle sont définis par l'outil et peuvent être vérifiés automa-

tiquement au moment de la conception.

La méthode WebQEM [12] utilise une partie des critères de qualité définis dans le standard ISO/IEC 9126-1. Les critères de qualité du standard ISO/IEC 9126-1 sont les suivants :

1. convivialité
2. fonctionnalité
3. fiabilité
4. rendement
5. portabilité
6. maintenabilité

Les auteurs ont retenu les quatre premiers pour concevoir leur méthode. Le tableau Table 3.1 nous présente une vue d'ensemble des critères retenus et les sous groupes qui les composent.

1. Convivialité
Compréhensibilité
Rétroaction et fonctions d'aide
Interface et fonctions esthétiques
Fonctions diverses
2. Fonctionnalité
Recherche et récupération
Navigation
Fonctionnalités spécifiques au domaine et son contenu
3. Fiabilité
Les erreurs de liens
Les fautes d'orthographe
Les différentes erreurs de fiabilité
4. Rendement
Performance
Accessibilité de l'information

Tab. 3.1 – Critères de qualité de la méthode WebQEM

Au niveau de la convivialité les auteurs s'attardent entre autres à la facilité de comprendre l'AW à l'aide d'index et une désignation appropriée des pages. La rétroaction (feedback) et les fonctions d'aide disponibles aux usagers sont étudiées. Par exemple nous vérifions s'il y a une page contenant

les questions les plus souvent posées (FAQ). Toujours au niveau de la convivialité, la qualité de l'interface est évaluée. Nous regardons aussi si l'AW est disponible en plus d'une langue.

La fonctionnalité est un autre critère de qualité qui s'applique aux AW. Trois sous-catégories ont été définies pour ce critère. La recherche et récupération forment la première sous-catégorie. Nous analysons la qualité des fonctions de recherche globales et spécifiques de l'AW. Le niveau de personnalisation et le niveau de rétroaction des mécanismes de récupération seront évalués. Une autre sous-catégorie du critère de fonctionnalité est la navigation. Nous faisons référence ici, par exemple, à la facilité de se déplacer à l'intérieur d'un formulaire électronique, d'un champ à l'autre, d'une section à l'autre, etc. Plus spécifiquement nous observons si l'endroit où nous nous trouvons est toujours bien indiqué. Nous analysons si les boutons de contrôle sont toujours en place et standard d'une page à l'autre. La facilité de défiler horizontalement et verticalement sera mesurée. Les noms des hyperliens doivent être représentatifs. Nous vérifions aussi si les hyperliens affichent une aide lorsque cela s'avère utile. Les mécanismes de navigation sont testés pour s'assurer qu'ils sont adéquats. La troisième et dernière sous-catégorie pour le critère de qualité fonctionnalité est spécifique au domaine et son contenu. C'est ici que nous pouvons mettre les critères de qualité fonctionnels qui s'appliquent spécifiquement à l'AW sur laquelle nous travaillons.

Le critère de fiabilité se divise en trois sous-catégories : les erreurs de liens, les fautes d'orthographe et les différentes erreurs de fiabilité. Les erreurs de liens sont les liens brisés, invalides et ceux qui ne sont pas implémentés. Les fautes d'orthographe nuisent aussi à la fiabilité des AW. Finalement nous retrouvons ici les différentes erreurs de fiabilité dont les déficiences ou l'absence de richesse fonctionnelle de certains fureteurs. Les lacunes ou les résultats imprévus qui sont indépendants des fureteurs vont aussi être examinés. Les pages orphelines et les noeuds de destination en construction font parti des critères de fiabilité analysés par les auteurs.

Le dernier critère de qualité retenu par les auteurs pour l'exemple présenté dans l'article est le rendement. Les auteurs s'attardent à la performance et à l'accessibilité de l'information et des fenêtres des AW. La performance fait référence ici à la rapidité d'apparition des pages statiques. Analyser la performance des pages statiques n'est pas suffisant selon nous. On devrait aussi tenir compte des pages dynamiques. Elles sont les plus susceptibles d'avoir des problèmes au niveau de la rapidité d'apparition.

L'accessibilité de l'information est vérifiée lorsque les usagers peuvent

lire les pages en format texte seulement. Les règles d'accessibilité doivent être conformes à la norme W3C qui s'y rattache (Web Accessibility Initiative WAI). Il existe différents niveaux de conformité à la norme WAI. Chaque entreprise doit déterminer quel niveau est acceptable pour elle. À titre d'exemple pour les règles d'accessibilité mentionnons la possibilité pour les non-voyants de lire, à l'aide d'outils spécialisés, le texte, la description des images et le contenu des fenêtres avec cadres.

3.3 Critères de qualité utilisés

Nous avons pris nos critères de qualité pour notre mémoire dans le standard ISO 9126-1. Le standard ISO 9126-1 vise plus particulièrement la qualité des produits logiciels. ISO est une organisation internationale de normalisation connue et respectée mondialement. C'est la raison pour laquelle nous avons adopté l'idée d'Olsina et Rossi [12] d'utiliser celle-ci. Nous limitons nos critères de qualité à ceux que nous croyons essentiels afin d'éviter d'engendrer un projet de mémoire trop généraliste. La fonctionnalité, la fiabilité et la maintenabilité sont les trois critères de qualité qui nous intéressent. Nous devions faire des choix et ces trois là sont les plus importants selon nous. La convivialité est un critère de qualité qui peut être jugé très différemment d'une personne à l'autre. Il est très difficile d'établir des règles universelles pour la convivialité. Il y a place à beaucoup d'interprétation avec ce critère. Par exemple les goûts esthétiques peuvent varier grandement d'une personne à l'autre. Nous ne voyons pas l'intérêt de retenir ce critère. L'autre critère retenu par les auteurs et non par nous est le rendement. Cette donnée dépend en grande partie des logiciels et matériels utilisés par le serveur Web. Cela dépasse les objectifs d'analyse visés par notre mémoire. Finalement le critère de qualité portabilité n'est pas utilisé par nous pour valider la qualité des AW. Tout comme le critère rendement, le critère portabilité dépasse nos objectifs. Nous supposons dans notre mémoire que le matériel informatique utilisé est à jour et qu'il n'y a pas de plan à court et moyen terme de changer le matériel utilisé.

Chapitre 4

Types de test

4.1 Introduction

Pour augmenter notre niveau de confiance envers un logiciel il faut construire des jeux de tests auxquels celui-ci sera ensuite soumis. Ils servent à découvrir s'il y a des erreurs dans le programme. Ces tests peuvent être créés automatiquement à l'aide d'outils ou manuellement. Nous pouvons automatiser la génération de tests d'un logiciel lorsque celui-ci est conçu à partir d'un document des besoins. Les tests automatisés demandent moins d'effort et de temps que ceux créés manuellement. Lorsque les spécifications d'un logiciel changent, nous avons seulement qu'à relancer l'outil et des nouveaux jeux de tests qui tiennent compte de ces changements sont produits. Cette capacité de recréer des jeux de tests rapidement et efficacement est importante surtout dans l'univers des AW. Ceux-ci sont très souvent en constante évolution et leurs spécifications changent normalement plus souvent que ceux des logiciels traditionnels.

Un exemple de méthode qui permet de générer automatiquement des suites de tests est SCR (software cost reduction). SCR est une méthode formelle pour le développement de logiciels. Dans [15] les auteurs présentent un outil écrit en Java qu'ils ont conçu qui utilise un model checker pour construire une suite de séquences de tests à partir d'une spécification SCR. Cet outil traduit la spécification SCR dans le langage interne de SMV ou SPIN. Il exécute ensuite le model checker pour générer des tests satisfaisant certains critères.

4.2 Catégories de test

Il existe différentes techniques de test qui sont appliquées à différents moments dans le cycle de développement d'un logiciel. Dans [16] on mentionne trois catégories de tests : mise à l'essai des défauts (defect testing), test d'intégration et test orienté-objet.

4.2.1 Test de mise à l'essai

Le but de ce type de test est de faire ressortir les vices cachés d'un programme. Les tests de type boîte noire font partie de cette catégorie. Nous vérifions les entrées et sorties d'un système sans se préoccuper de son fonctionnement interne. Nous devons choisir des entrées qui peuvent causer des erreurs en sortie.

Les entrées de données dans un programme peuvent normalement être divisées en groupes qui sont équivalents selon un certain critère. Une fois ces groupes définis on peut composer des suites de tests pour chacun d'eux. Prenons par exemple un module qui prend en entrée un entier situé entre 0 et 100. Il retourne "p" si le chiffre est pair et "i" s'il est impair. Une façon de tester les différents groupes est de prendre des valeurs au milieu du groupe et de prendre des données situées aux frontières [16]. Dans notre exemple on pourrait fournir en entrée les entiers suivants : -1, 0, 1, 50, 51, 99, 100 et 101.

Les tests de type boîte transparente forment une autre catégorie de test qui fait partie des tests de mise à l'essai. Ces tests sont dérivés des connaissances de la structure du logiciel. Ils sont parfois appelés test de type boîte blanche. Ils sont normalement utilisés avec de petites parties de programmes tels que des sous-programmes associés avec un objet. Nous étudions le code et nous utilisons les connaissances à propos de la structure d'un composant du logiciel pour produire ces tests.

Les tests de chemin (path testing) sont des tests structurels. Ils vérifient le code d'un programme. Leur objectif est d'exécuter tous les chemins indépendants d'un programme. Pour qu'un chemin de programme soit considéré indépendant il doit traverser au moins une nouvelle arête dans l'organigramme représentant la partie du code testée. L'exécution de tous les chemins indépendants implique que toutes les instructions ont été exécutées au moins une fois et toutes les conditions ont été testées à vrai et faux. Les tests de chemin ne testent pas toutes les combinaisons possibles de tous les chemins d'un programme. Un organigramme sert de point de départ pour déterminer

les noeuds et les arêtes. Les noeuds représentent les conditions et les arêtes représentent le flot des instructions. Le nombre de chemins indépendants à tester peut être calculé avec cette formule :
CC (cyclomatic complexity) = Nombre d'arêtes - Nombre de noeuds + 2.
Le jeu de tests doit traverser toutes les branches. Le nombre minimum de tests sera égal au nombre obtenu avec la formule CC.

4.2.2 Test d'intégration

Une fois que les composantes indépendantes du programme ont été testées, elles doivent être intégrées pour créer un système partiel ou complet. Les tests d'intégration doivent être faits à partir de la spécification du système. Ces tests doivent débuter dès que des composantes du système peuvent être mises ensemble pour être testées.

Il existe deux approches différentes vis à vis l'intégration dans un système soit la méthode top-down et la méthode bottom-up. Avec la méthode top-down les composantes principales d'un système sont intégrées et testées avant qu'elles soient complétées. Dans la méthode bottom-up, les composantes de niveau plus bas sont intégrées et testées avant que les composantes principales soient développées. Les systèmes sont souvent développés en utilisant une combinaison des deux approches.

Lorsque les modules sont intégrés ensemble, les interfaces qui permettent aux modules de communiquer entre eux doivent aussi être testées. Il y a trois classes d'erreurs d'interface : mauvais emploi d'une interface, malentendu entre interfaces et erreurs de synchronisation. Sommerville [16] donne quelques conseils pour faire des tests d'interface. Nous commençons par vérifier le code et en faisant une liste de tous les appels aux composants externes. Un jeu de tests est ensuite construit avec des données situées aux extrémités de la plage de données possibles. Dans le cas où des pointeurs seraient transmis entre interfaces, un test avec le pointeur nul serait exécuté. On utilise des tests de stress c'est-à-dire que nous créons des tests qui engendrent plus de messages que ce qui devrait se produire en réalité. Les erreurs de synchronisation devraient ainsi se dévoiler. Lorsque plusieurs composantes interagissent en utilisant une mémoire partagée on doit créer des tests qui varieront l'ordre dans lequel ces composants sont activés.

Certaines classes de système ont été construites pour traiter un certain nombre de tâches. Avec le test de stress nous continuons de faire des tests jusqu'à temps que le système ne soit plus capable de poursuivre et qu'il

s'arrête suite à un échec. Nous analysons ensuite quel est l'impact de l'arrêt imprévu sur le programme.

4.2.3 Test orienté-objet

Il y a deux activités fondamentales dans le processus des tests des logiciels. Nous devons tester les composantes individuellement et ensuite joindre les composantes qui vont ensemble et faire des tests d'intégration. Ces activités s'appliquent aussi aux applications orientées objet. Il y a par contre des différences entre des systèmes orientés objet et ceux développés utilisant un modèle fonctionnel. Les objets, en tant que composantes individuelles, sont souvent plus gros qu'une simple fonction. Les objets intégrés dans des sous-systèmes sont jumelés de façon vague. Il n'y a pas de haut du système facilement identifiable. Si des objets sont réutilisés, les testeurs peuvent ne pas avoir accès au code source [16]. Les tests de type transparent doivent être adaptés pour couvrir de plus grand étendu et des approches alternatives aux tests d'intégration devront être utilisées.

4.3 Particularités des applications Web

Les AW sont des applications logicielles ayant certaines particularités qui les distinguent des applications logicielles traditionnelles. Nous devons prendre en considération ces particularités au moment d'écrire nos suites de tests. Nous énumérons et décrivons brièvement ici quelques unes de ces particularités.

4.3.1 Environnement hétérogène

Les AW sont exécutées dans des environnements matériels et logiciels hétérogènes. Elles sont généralement réparties sur des architectures à trois niveaux. Une représentation de la structure typique d'une AW est affichée sur la figure 4.1 [7].

Les logiciels traditionnels sont habituellement conçus pour fonctionner dans un environnement logiciel et matériel limité. Un logiciel typique est programmé pour fonctionner sur un ou quelque(s) système(s) d'exploitation spécifique(s). Il est normalement requis que l'ordinateur possède une certaine quantité de mémoire vive, d'espace disponible sur le disque dur et un certain

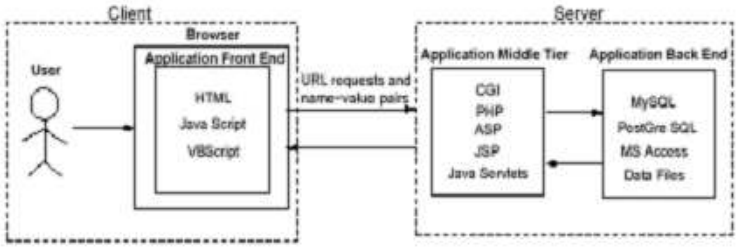

FIG. 4.1 – Structure d'une application Web

type de carte vidéo. Il en est tout autrement pour les AW. Elles doivent fonctionner sur tous les systèmes d'exploitation connus (Windows, Linux, Unix, Macintosh, etc.). Elles doivent fonctionner en utilisant les fureteurs les plus connus.

4.3.2 Standards d'écriture

Contrairement aux logiciels traditionnels, lorsque nous développons des AW nous pouvons nous permettre de ne pas respecter certains standards d'écriture HTML par exemple et notre application va fonctionner quand même. Cela est possible car les concepteurs des fureteurs ont prévu ces erreurs et ils ont programmé leurs fureteurs pour être capable de passer outre ces erreurs. Par exemple, en HTML toutes les balises d'ouverture devraient être suivies d'une balise de fermeture mais parfois les balises de fermeture sont omisses et les pages Web s'affichent et fonctionnent quand même comme prévu. Ce non-respect des standards n'est pas souhaitable car dans certaines situations, par exemple avec certains types de fureteur, les pages Web peuvent afficher des erreurs. Heureusement les plus récents standards W3C tel que XHTML, CSS et XML imposent des règles d'écritures strictes et qui ne peuvent pas être contournées comme c'est le cas présentement avec HTML.

4.3.3 Liens

La vérification du bon fonctionnement des liens est un autre type de test qui s'adresse aux AW. La presque totalité des AW contiennent des liens. Au minimum, elles disposent normalement d'au moins un lien qui permet

à l'usager de retourner à la page d'accueil. Ils permettent aux usagers de naviguer à l'intérieur et à l'extérieur d'une AW. Nous les retrouvons sous forme statique et dynamique.

4.3.4 Fureteurs

Un autre type de test qui s'adresse spécifiquement aux AW est la vérification de la compatibilité de l'application avec les différents fureteurs existants. Nous vérifiions les AW pour s'assurer qu'elles fonctionnent bien dans les différents fureteurs. Dans les cas où sont utilisés des langages de script comme JavaScript par exemple, nous devons vérifier que les fureteurs exécutent bien les commandes prévues. Concernant les fureteurs, dépendamment des applications, nous pourrions être intéressé à vérifier certaines capacités et paramètres du fureteur accédant l'AW. Nous pensons ici notamment aux témoins (cookie) et aux certificats de sécurité.

4.3.5 Cadres

Des cadres (frames) peuvent être utilisés dans les AW et lorsque c'est le cas nous devons contrôler leur bon fonctionnement. Une page Web pourrait être affichée au sommet d'un fureteur alors qu'elle doit toujours être affichée dans un cadre. Une page peut aussi s'afficher dans un cadre où elle ne devrait pas être affichée. Une autre possibilité d'erreur est que le contenu de la page Web qui s'affiche dans un cadre soit inconsistant avec le reste de la page Web. Par exemple un menu dans un cadre à gauche pourrait n'avoir aucun lien avec la page centrale située à droite. C'est à dire que l'index affiché [17] dans le cadre ne soit pas en harmonie avec la page principale située à droite.

4.3.6 Interfaces

Dans les applications traditionnelles les interfaces entre l'usager et le logiciel sont clairement définies. L'utilisateur ne peut pas modifier l'ordre par lequel il doit passer pour accéder aux différentes fonctions du logiciel. Prenons l'exemple du programme bloc-notes de Microsoft. Pour enregistrer une page nous devons activer *fichier* et ensuite activer *enregistrer* ou *enregistrer sous*. On peut aussi utiliser les raccourcis pour faire la même chose mais toujours en respectant l'ordre prévu. Dans une page Web l'usager peut utiliser à tout

moment les boutons *précédent* et *suivant* et ainsi modifier l'ordre dans lequel les interfaces sont accédés. L'usager peut écrire directement une adresse URL dans la barre d'adresse et ainsi se rendre directement à l'une des pages de l'AW sans passer par la page d'accueil. On doit prévoir de vérifier ceci lorsqu'on élaborera nos suites de tests.

4.3.7 Nombre d'usagers

Le nombre maximum d'usagers qui peuvent utiliser un même logiciel traditionnel en même temps est connu. Lorsqu'un logiciel est installé sur l'ordinateur d'un usager, il ne sera utilisé que par une personne à la fois. Dans le cas d'un logiciel qui se trouve sur un serveur, son utilisation sera limitée par le nombre de licences acquises. Dans le cas des AW le nombre d'utilisateurs peut varier grandement d'un instant à l'autre. Nous devons tenir compte de ce facteur lorsque nous mettrons au point nos différents tests d'assurance de la qualité.

4.3.8 Transactions

Les usagers sont amenés très souvent à utiliser des formulaires pour transmettre des données à partir du client (fureteur) vers les bases de données. Lorsque ces transferts de données sont des transactions, nous devons garantir le bon déroulement de ces transactions. Ceci est particulièrement vrai dans les cas de transactions bancaires ou commerciales.

4.3.9 Documents

Les logiciels traditionnels sont concentrés sur l'aspect informatique tandis que les AW sont majoritairement concentrées sur la recherche et l'acquisition d'information et de documents. Les AW peuvent souvent être vues comme étant des applications évoluées situées au-dessus de la base de données sous-jacente. Ils fournissent une interface graphique aux usagers pour rechercher de l'information ou des documents [4].

Chapitre 5

Modèles de conception d'applications Web

5.1 Introduction

Dans notre mémoire nous analysons la qualité des AW existantes mais aussi avant et pendant leur création. C'est ce qui nous a amené à s'intéresser aux modèles de conception d'AW. Il existe aussi des systèmes de gestion du contenu (CMS content management system) tel que joomla, mambo et wordpress par exemple qui permettent de créer rapidement et efficacement des AW possédant une interface graphique de grande qualité. Malgré les qualités (simples à utiliser, interfaces de qualité, etc.) que nous reconnaissons à ces outils elles n'offrent pas la même latitude que les modèles de conception d'AW. C'est la raison pour laquelle dans ce chapitre nous avons concentré nos recherches sur les modèles de conception.

Trois modèles en particulier ont retenu notre attention : SML, WebML et Rational Application Developer (RAD). Les auteurs qui ont utilisé le langage SML pour créer leur modèle de conception d'AW n'ont pas rendu leur outil disponible publiquement. Nous n'avons donc pas pu essayer cet outil. Par contre la méthode WebML a donné naissance à l'outil WebRatio dont une version d'essai est disponible gratuitement [18]. Le modèle RAD provient d'IBM qui fournit lui aussi une version d'essai gratuite [19]. Nous avons conçu une AW d'une dimension et complexité moyenne en utilisant la méthode WebML et la méthode RAD. Cette AW est en fait un site Web de vente de livres en ligne. La page d'accueil affiche la liste des livres à vendre qui sont

récupérés à partir d'une base de données. Une page administrative sert à la gestion des livres en vente sur le site. La création de cette AW nous a permis de mieux saisir les avantages et désavantages de ces modèles.

5.2 Conception d'applications Web avec SML et MSP

Le modèle de conception d'AW présenté dans [20] est intéressant de part son originalité et sa rigueur. Au départ on insiste sur l'importance de séparer la partie navigation de la partie non-navigation des AW. Cette séparation rend plus aisée l'exécution des tests. De plus, la séparation des préoccupations (concern) favorise la documentation, le développement et le maintien de l'AW. Cette approche sépare la création d'AW en trois grandes étapes : écriture de la spécification des exigences fonctionnelles, génération du schéma de la base de données et création du diagramme de navigation. Ce modèle de conception d'AW utilise le langage de programmation fonctionnel SML et le langage de script Web MSP (ML Server Pages) pour créer des AW.

La spécification des exigences fonctionnelles est donnée par des signatures. On y retrouve quatre catégories syntaxiques : noms de fonctions, noms de types, types de base et constructeurs de types. Les types de base sont int, bool et string. Pour les constructeurs on a les produits cartésiens '*' et les constructeurs de liste 'liste'. Une spécification de fonction peut avoir une de ces formes :
(1) f : typeName_1 * ... * typeName_n $\rightarrow$ typeName
(2) f : typeName_1 * ... * typeName_n $\rightarrow$ unit
Le type 'unit' nous indique que cette fonction peut avoir des effets secondaires.

Nous retrouvons à la figure 5.1 un exemple de spécification des exigences fonctionnelles pour une Université. Dans cet exemple deux types d'usagers existent : student et teacher. L'Université possède plusieurs curricula (type Curr) et plusieurs cours (type Course). Les deux premières lignes définissent des requêtes exécutables par les deux types d'usagers. Dans la première requête (ShowCurr) le nom d'un curriculum (Name) doit être fourni en entrée et en sortie nous obtenons le curriculum (Curr) qui lui comprend la liste des cours qui forment ce curriculum. Dans la deuxième requête (ShowCourse)

```
ShowCurr   : Name            -> Curr          [student, teacher]
ShowCourse : CourseId        -> Course        [student, teacher]

AddCourse  : Name * CourseId -> unit          [teacher]

Name       = string
CourseId   = int
Curr       = Name * Course list
CourseDesc = string
Course     = CourseId * CourseDesc
```

FIG. 5.1 – Spécifiation des exigences fonctionnelles de l'Université

l'entrée est le numéro d'un cours (CourseId) et la sortie est le cours (Course) qui s'y rattache. La fonction AddCourse énonce que seul les usagers de type teacher ont la possibilité d'ajouter des cours en fournissant en entrée de cette fonction le nom d'un curriculum (Name) accompagné du numéro d'un cours (CourseId).

Après la spécification des exigences il faut créer un plan de la base de données. On débute en construisant le diagramme entité relation. Celui-ci est généré automatiquement à partir des déclarations de types normalisés. Le diagramme entité relation pourra ensuite être modifié manuellement par l'usager. Certaines relations ne peuvent pas être exprimées par des types.

Une déclaration de types normalisés ne doit contenir que des expressions des types suivants : type de base, type de la forme NomDeType * list et NomDeType_1 *...* NomDeType_n. Ainsi dans l'exemple de l'Université la déclaration Curr = Name * Course list a été transformé en deux nouveaux types : Courses = Course list et Curr = Name * Courses.

Les types simples deviennent des attributs (ex. Name). Les types définis par un produit cartésien déviennent des entités (ex. Course). Finalement, les types définis par un constructeur de liste deviennent des entités faibles et un attribut pour l'index de la liste connectée à l'entité (ex Courses). Nous avons un exemple de diagramme entité relation créé par cette méthode à la figure figure 5.2 pour notre exemple d'Université.

Le schéma d'une base de données est généré à partir du diagramme entité relation. Nous retrouvons à la figure 5.3 le schéma de la base de données créé pour l'exemple d'Université.

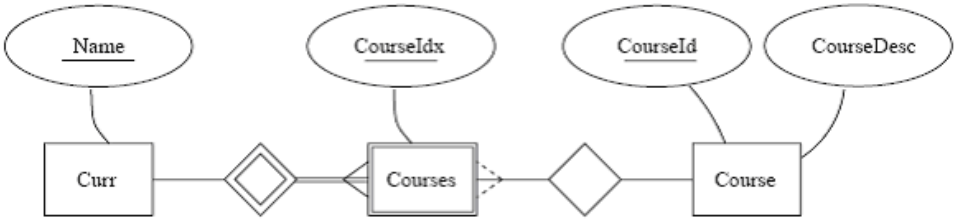

FIG. 5.2 – Diagramme entité relation pour l'Université

```
CREATE TABLE Curr{Name TEXT PRIMARY KEY};

CREATE TABLE Courses{
   Name TEXT REFERENCES Curr(Name),
   CourseIdx INT,
   CourseId INT REFERENCES Course(CourseId),
   PRIMARY KEY (Name, CourseIdx)};

CREATE TABLE Course{
   CourseId INT PRIMARY KEY, CourseDesc TEXT};

CREATE USER Student; CREATE USER Teacher;

GRANT SELECT
   ON Curr,CurrCourse,Course TO Student,Teacher;

GRANT INSERT, DELETE, UPDATE
   ON CurrCourse,Curr,Course TO Teacher;
```

FIG. 5.3 – Schéma de la base de données

L'usager doit ensuite construire le diagramme de navigation à partir d'un modèle de base fourni avec l'outil. Il doit respecter les exigences fonctionnelles spécifiées dans les signatures. C'est un graphe dans lequel les noeuds représentent des pages Web et les arêtes nous indiquent les liens entre ces pages Web. Les liens ont zéro, une ou plusieurs fonctions. On exerce un contrôle de la cohérence. Les fonctions dans le diagramme de navigation sont appliquées à des arguments du bon type en fonction des signatures. Toutes les fonctions disponibles à un groupe doivent être atteignables dans le diagramme de navigation. Les fonctions non disponibles pour un groupe selon les signatures ne sont pas disponibles dans le diagramme de navigation. Il n'y a pas d'hyperliens manquants et pas de partie du diagramme qui n'est pas accessible.

Le code est généré à partir d'une signature et un diagramme de navigation consistant. On veut que l'implémentation des exigences fonctionnelles soit faite dans un environnement fortement typé et que la partie navigation soit séparée de la partie fonctionnelle. On veut que le code HTML généré soit bien formé et finalement que le type des données d'entrée soit vérifié du côté client et serveur.

La partie fonction du code est générée à partir de la signature. On obtient ainsi une architecture modulaire. Le code de la partie navigation est généré en deux parties. Premièrement deux fichiers de configuration sont générés pour chaque page Web se trouvant dans le diagramme de navigation. Un fichier contient les définitions des éléments statiques nécessaires pour acquérir les arguments aux fonctions à partir des liens sur la page. Ce sont des champs d'entrée de donnée. L'autre fichier décrit le contenu visuel de la page Web et l'ordre des informations qui si retrouvent. Ce fichier peut être modifié par l'usager pour enrichir le contenu visuel et le contenu de la page Web concernée. Deuxièmement un fichier MSP est généré pour chaque page Web. L'ensemble des fichiers MSP forme le diagramme de navigation. L'outil génère ensuite le code HTML.

Ce modèle de conception est intéressant à plusieurs niveaux. Par contre la faiblesse de ce modèle est sa complexité d'utilisation. L'usager doit au départ écrire les signatures. Ensuite il doit normaliser les déclarations de types. À partir des signatures un diagramme entité relation est automatiquement créé. L'usager doit ensuite analyser s'il doit modifier manuellement ce diagramme pour exprimer certaines relations non-définies dans les signatures. Finalement, l'usager doit modifier les fichiers décrivant le contenu visuel des pages Web qu'il souhaite plus attirantes. Il est valable académiquement mais sans

intérêt au niveau pratique.

5.3 Modèle de conception WebML

Le modèle WebML [10] est en développement depuis 2001 et peut être utilisé pour construire des AW de grande taille et complexité. C'est une méthode très prometteuse. Ce modèle est associé avec le logiciel intégré (framework) WebRatio. WebML (Web Modeling Language) est un langage de conception pour les AW. Il est particulièrement utile pour la création de sites Web demandant l'accès à une grande quantité d'information. Il fournit une spécification formelle graphique. Les principaux objectifs de la méthode de création WebML sont les suivants :

1. Exprimer la structure d'une AW avec une description de haut niveau.
2. Fournir différentes prises de vue du même contenu.
3. Séparer les contenants d'information des composants qui utilisent ces informations.
4. Enregistrer l'information recueillie pendant le processus de création des AW dans un répertoire et le rendre disponible pour générer automatiquement des AW.
5. Modéliser les usagers et les groupes pour permettre la spécification de politiques personnalisées.
6. Permettre la manipulation des données pour mettre à jour ceux-ci.

Pour concevoir des AW, la méthode WebML utilise quatre modèles. Le **modèle de données** (Data Model) décrit l'organisation conceptuelle de l'information. Le modèle de données est compatible avec le modèle entité/relation et les diagrammes de classe UML. Le **modèle de dérivation** (Derivation Model) permet d'étendre le modèle structurel. Le **modèle hypertexte** (Hypertext Model) permet d'élaborer la spécification frontale (front-end) de l'application. L'application est composée de modules que l'on nomme site views. Ceux-ci sont divisés en sous-modules. Le contenu hypertexte de chaque module est défini en termes de pages et de liens. Le **modèle de gestion du contenu** (Content Management Model) permet le développement du modèle hypertexte.

On distingue deux types d'usagers dans WebML soit les usagers enregistrés et les non enregistrés. On a aussi deux types de sites. Nous avons

les sites publics qui sont accessibles par tous les usagers et les sites protégés qui sont accessibles seulement aux usagers enregistrés qui ont les droits pour accéder à ce site. Il y a des opérations dans WebML qui permettent de se connecter (login) à un site, se déconnecter (logout) d'un site, changer de groupe, etc.

WebML est un bon exemple de modèle de conception d'AW qui permet aux concepteurs de créer des AW de qualité. Une des qualités de cette méthode est la séparation entre les différentes composantes et relations qui misent ensemble forment une AW. L'outil WebRatio, en utilisant WebML comme modèle de conception, rend la création d'AW accessible à tous. Contrairement à la première méthode étudiée, celle-ci suscite de l'intérêt qui dépasse le milieu académique. WebRatio offre son modèle de conception depuis 2001. Des mises à jour sont offertes régulièrement.

5.4 Modèle de conception IBM Rational Application Developper

Au coeur de l'outil IBM RAD, comme pour tous les autres produits rational d'IBM, se trouve l'environnement de développement intégré de source libre eclipse. Notons au départ que RAD est un outil de taille imposante (plus de 5 Go) offrant plusieurs services. Il peut être utilisé pour développer, analyser et tester des AW, des portails et des programmes Java. En regard de l'outil RAD notre intérêt se limite à la capacité de création et de mise à l'essai d'AW dynamiques de celui-ci. L'environnement de développement de RAD fournit les outils nécessaires pour construire des AW contenant entre autres des pages JSP et des servlets Java. Un serveur nommé Websphere fait parti intégrant de RAD et permet ainsi de tester les AW dynamiques.

L'exemple que nous avons construit avec RAD nous a permis de constater que cet outil est relativement facile à utiliser. La rubrique d'aide contient beaucoup d'information et de plus plusieurs exemples sont inclus avec le produit RAD. Des exemples présentés sous forme d'animation flash permettent à l'usager de se familiariser plus rapidement avec cet outil.

L'outil IBM RAD est fourni avec un ensemble de fonctions de validation. Pour ce qui est du sujet qui nous intéresse dans IBM RAD, les AW dynamiques, nous retrouvons onze fonctions de validation par défaut. Nous n'entrerons pas dans les détails de chacune de ces fonctions de validation

mais nous allons plutôt nous intéresser aux buts visés par ceux-ci. Le point important à souligner c'est qu'avec IBM RAD nous retrouvons surtout des fonctions de validation du code de programmation qui permet la génération des différentes pages Web qui mises ensemble forment une AW dynamique. Mentionnons que ces fonctions permettent de valider du code DTD (document type definition), des fichiers XML, XSL et WSDL (Web Services Description Language). Le code JSP et JavaServer Faces (JSF) peuvent eux aussi être validés grâce aux fonctions de validation disponibles par défaut dans IBM RAD.

Chapitre 6

Méthodes d'évaluation de la qualité des AW

6.1 Introduction

Il existe plusieurs méthodes d'évaluation de la qualité des AW et parmi celles-ci nous en avons trouvées quelques unes dignes de mention. Nous présentons un résumé de ces méthodes ainsi que leurs forces et faiblesses.

Une approche intéressante pour évaluer la qualité AW est présentée dans [1]. Cet article et d'autres articles du même auteur sont publiés sur le site W3C italien. Dans [1] le but principal visé est de définir pour les AW un modèle de qualité et un ensemble de caractéristiques qui peuvent être mesurées automatiquement. ISO définit la qualité des logiciels selon trois points de vue différents : celui des usagers, des développeurs et des gestionnaires. Les usagers sont concernés par la qualité externe. Les développeurs et les gestionnaires sont davantage intéressés par la qualité interne des logiciels. Selon l'auteur les sites Web sont généralement évalués à partir du point de vue de l'usager. Ceci explique pourquoi il considère surtout les qualités externes.

Au départ dans [1] on signale et analyse quelques méthodes et critères d'évaluation de la qualité des AW. Nous citons celles qui nous apparaissent comme étant les plus intéressantes. On nous informe qu'il existe des critères de qualité Web spécifiques aux instances gouvernementales. Par exemple [21] est un document de plus de cent pages qui énumère en détails des critères de qualité spécifiques pour les AW gouvernementales.

D'autres approches définissent des critères de qualité Web applicables à

des ensembles plus restreints. Par exemple MINERVA est un organisme qui définit des critères de qualité suggérés pour les sites Internet culturels gouvernementaux européens. L'emphase est mise sur l'accessibilité et la facilité d'utilisation.

Dans [2] on parle d'une approche générique pour l'évaluation des AW. La méthode 2QCV3Q est présentée dans cet article. La qualité des AW est basée sur sept dimensions : qui, quoi, pourquoi, quand, où, comment et la faisabilité. Nous présentons cette méthode plus en détail un peu plus loin dans cette section.

L'auteur parle aussi de la méthode MiLE [22] qui permet d'évaluer les applications hypermédias. Cette méthode se concentre sur l'aspect utilisation. On cite aussi la méthode ETNOTEAM basée sur six attributs : communication, contenu, fonctionnalité, utilisation, gestion et accessibilité. Le modèle peut être personnalisé. L'importance des sous attributs est ajustée en fonction du type de site Web. Dans [23] la qualité des AW est analysée en utilisant plusieurs critères. Après analyse on donne une note en pourcentage pour la qualité des AW analysées. L'auteur mentionne aussi les standards de qualité ISO9241, ISO9126 et ISO13407 qui décrivent respectivement les standards d'utilisation, qualité des logiciels et de la production centrée sur les usagers.

6.2 Modèle à cinq dimensions

L'article [1] soulève le fait que l'évaluation de la qualité de plusieurs approches souffre de plusieurs limites. Les critères sont souvent qualitatifs donc sujets à erreur et discussion. Les critères sont fréquemment très généraux. Certaines caractéristiques sont évaluées plus d'une fois. Plusieurs critères d'évaluation n'évaluent que l'accessibilité ou l'utilisation. La distinction est floue entre qualité des pages Web et des AW. La perception de la qualité change en fonction du point de vue de l'usager (ex. client vs développeurs). Pour définir des unités de mesure on doit avoir des caractéristiques mesurables et une approche vigoureuse.

L'auteur propose un modèle à cinq dimensions : exactitude, présentation, contenu, navigation et interaction. La classification a été conçue pour pouvoir couvrir une certaine automatisation du processus d'évaluation de la qualité des AW. L'exactitude est un aspect technique interne tandis que les quatre autres dimensions sont reliées à la perspective des usagers. On veut évaluer les quatre dimensions en fonction du code interne. On veut définir un modèle

d'AW et une base de données de qualité. Les experts utiliseront la base de données pour évaluer les différents aspects que la qualité d'une AW qui demandent une intervention humaine.

Pour évaluer l'exactitude plusieurs outils existent. La présentation comprend la mise en page, la police des caractères, la présentation multimédia, le nombre et le bon fonctionnement des liens et l'accessibilité des formulaires. La dimension contenu enveloppe la lisibilité, architecture de l'information, structure de l'information, distinction entre auteur et webmestre, date de dernière mise à jour. La dimension navigation s'occupe de la topologie des liens. On retrouve des topologies de lien en forme d'arbre, horizontal, etc. Lorsqu'on parle de la dimension interaction celle-ci est majoritairement implémentée en utilisant des formulaires. On s'attarde à la transparence, capacité de récupération et possibilité de mettre des commentaires lorsque cela s'applique.

Plusieurs aspects intéressants ressortent de l'article [1]. Pour commencer, la revue de différentes méthodes d'évaluation démontre qu'il n'existe pas de définition unique d'une AW de qualité. Les critères de qualité diffèrent parfois grandement entre chacune des méthodes discutées. La décision de choisir les critères de qualité en fonction de leur possible automatisation est une bonne idée selon nous. Il est évident que les concepteurs d'AW seront toujours plus enclins à utiliser des outils automatiques versus des méthodes manuels. La création d'une base de données dans laquelle seraient récupérées les données qui demandent une évaluation d'expert nous apparaît aussi comme un bon plan. Un des points faibles de l'article selon nous est que l'auteur n'explique pas comment les différents critères qu'il propose de retenir, pourraient être évalués de façon automatique. À part quelques exemples simples, l'auteur ne donne pas de pistes de solution pour automatiser le tout. Dans notre méthode les critères de qualité ont aussi été retenus en tenant compte de leur possible automatisation. Lorsque la mise en place d'une méthode automatique d'évaluation n'est pas possible une méthodologie est alors présentée pour retrouver les données clefs le plus facilement possible. Pour chacun des critères de qualité retenus nous expliquons en détail comment faire pour analyser ceux-ci que se soit de façon automatique ou autre.

6.3 Modèle 2QCV2Q

Une autre méthode intéressante d'évaluation de la qualité des AW est suggérée dans [2]. Cette méthode est plus généraliste que celle exprimée dans [1]. Par contre cette méthode a le mérite d'apporter une réflexion intéressante sur ce qui forme une AW de qualité.

Les auteurs expliquent dans [2] qu'ils ont créé le modèle 2QCV3Q qui permet d'assurer la qualité des AW. Ce modèle prend en considération le point de vue des différentes personnes impliquées dans la conception d'une AW : propriétaire(s), utilisateurs et développeurs de l'AW.

Ils évaluent la qualité des AW en fonction des réponses obtenues suite aux différentes questions produites à partir du modèle 2QCV3Q. Ces questions couvrent sept catégories différentes : identité, contenu, services, localisation, gestion, utilisation et faisabilité. La catégorie identité (qui ?) fait référence à l'image de marque qui fait qu'une AW se distingue des autres AW. La catégorie contenu (quoi ?) prend en compte la qualité de l'information publiée sur le site. La catégorie services (pourquoi ?) demande que les AW répondent aux objectifs des propriétaires et usagers. Le contrôle de l'exactitude de l'information et l'aspect sécurité sont aussi analysés à ce niveau. La prochaine catégorie, soit la localisation (où ?), évalue la facilité d'interaction avec l'AW. On vérifie le nom des adresses URL pour voir s'ils sont représentatifs de l'AW en question. Une analyse est faite pour savoir si des outils sont en place pour permettre une interaction entre les usagers. La catégorie gestion (quand ?) vérifie que l'information sur l'AW est maintenue à jour. Les informations publiées devraient être accompagnées de leurs dates de publication. La catégorie utilisation (comment ?) vérifie l'accessibilité, la navigabilité et la facilité de compréhension d'une AW. La catégorie faisabilité (avec quels moyens et outils ?) est concernée par les ressources financières et humaines disponibles pour une AW.

Le processus d'évaluation 2QCV3Q est mis en place en trois étapes : mise en place initiale, conception et réalisation. Nous retrouvons dans la mise en place initiale le but de l'évaluation, le type d'AW à être évalué, le niveau de développement de l'AW, les objectifs du ou des propriétaire(s) et ceux des usagers. Cette étape produit en sortie la liste des critères de qualité. À l'étape de la conception nous devons mettre en place la méthode pour récupérer l'information voulue sur la qualité des AW en fonction des critères de qualité choisis à la première étape. Un plan d'évaluation est écrit. À l'étape de la réalisation nous évaluons la qualité des AW en utilisant les méthodes

d'analyse du plan d'évaluation.

Les auteurs concluent l'article en mentionnant que leur méthode peut être appliquée à tous les différents genres de domaines d'AW. Le modèle est extensible. Il permet d'évaluer les AW avec différents degrés de précision selon les besoins. Ils mettent l'emphase sur la facilité d'utilisation de leur modèle.

Nous reconnaissons certaines qualités au modèle 2QCV3Q. Les sept catégories sélectionnées par les auteurs sont assez générales pour s'appliquer à presque toutes les AW de moindre envergure. Les questions engendrées par ce modèle permettent de couvrir une grande partie de ce qui fait qu'une AW peut être considérée de qualité ou non. Une faiblesse évidente selon nous de cette méthode est en relation directe avec la qualité qu'elle possède d'être applicable à toutes les AW. Une des raisons principales qui fait que cette méthode peut être appliquée à toutes les AW est qu'elle manque de rigueur. Elle est trop généraliste. Puisque les auteurs présentent 2QCV3Q comme étant un modèle nous croyons qu'ils devraient expliquer en détail la façon d'utiliser celui-ci. Nous devrions retrouver une série d'étapes qui explique exactement comment utiliser ce modèle de façon concrète. Ils mentionnent dans l'article quelques étapes à suivre pour utiliser leur modèle mais encore une fois leurs explications sont très générales et n'aident pas vraiment le lecteur à comprendre exactement comment il pourrait utiliser ce modèle de façon concrète. Quelques exemples d'utilisation du modèle sont énumérés dans l'article mais encore une fois les auteurs ne donnent pas de détails sur le cheminement pris pour évaluer ces AW à l'aide du modèle 2QCV3Q. Nous avons retenu de cet article certains critères de qualité dont ceux-ci : publications accompagnées d'une date, accessibilité et la navigabilité.

Chapitre 7

Spécifications

7.1 Introduction

La spécification fonctionnelle qui accompagne une AW est un document pouvant nous aider à valider la qualité des AW. Elle nous aide, entre autres, à connaître les fonctions qui devraient être implémentées dans une AW. Suite à cette information nous pouvons définir les tests de fonctionnalité appropriés. Il existe différents langages formels d'écriture de spécification pour logiciel. Voici quelques exemples de ces langages : UML, Z et VDM. Il y a aussi des langages de spécification qui s'appliquent mieux aux AW tel que OOHDM, OOWS, etc. Une spécification devrait être écrite avant même qu'une seule ligne de code de l'AW ne soit écrite. Elle définit le squelette de l'AW et contient des informations cruciales tel que les différentes fonctionnalités, les écrans, les méthodes d'échange de données, les bases de données, etc. Au fur et à mesure qu'une AW est construite la spécification qui l'accompagne pourra être modifiée pour s'adapter aux changements qui peuvent survenir en cours de route. Normalement les changements devraient être mineurs et sans impact notable sur la forme générale de l'AW prévue au départ. En pratique dans le monde des AW la spécification est souvent inexistante. Dans les cas où une spécification est présente, elle est souvent écrite de façon informelle et une fois l'AW terminée. Rappelons quelques-unes des raisons qui expliquent en partie ce plus grand laxisme au niveau de l'écriture des spécifications dans le monde des AW en comparaison de celui des applications logiciels traditionnelles : temps de conception plus court, changements plus fréquents, durée de vie plus courte, etc.

En réponse à l'absence ou la piètre qualité d'une spécification fonctionnelle nous recommandons l'utilisation d'une spécification axée sur le contrôle de la qualité [13, 25, 26, 27, 28, 29]. Il existe quelques méthodes d'écriture de spécification pour AW qui méritent d'être mentionnées. Les méthodes d'écriture et de vérification des spécifications intéressantes pour nous sont celles qui peuvent être vérifiées de façon automatique ou semi-automatique. L'un des objectifs visés par notre méthodologie est de rendre le processus de contrôle de la qualité le plus automatisé possible. Nous présentons donc ici trois méthodes intéressantes d'écriture de spécification qui ont retenu notre attention.

7.2 Gverdi

Il existe plusieurs outils [30] qui permettent de faire différents types de test pour nous aider dans notre quête d'obtention d'AW de qualité. En contrepartie il existe peu d'outils permettant d'automatiser la conception et vérification d'une spécification pour AW. Le système Gverdi [25] permet ce genre de vérification. C'est un outil qui a été écrit en utilisant le langage Haskell qui lui-même est basé sur le lambda calcul et qui dispose d'une interface utilisateur graphique (GUI). L'usager écrit la spécification en respectant la grammaire proposée et ensuite il récupère le dossier contenant l'ensemble des pages Web et active un bouton qui fournit en sortie l'ensemble des erreurs identifiées. Cet outil nous permet de détecter les pages incomplètes, incorrectes, interdites ou manquantes de façon automatique en fonction des propriétés définies dans la spécification.

Pour les auteurs, une page Web est un document XHTML ou XML. Les pages Web sont représentées comme étant un ensemble fini de termes. Les auteurs définissent une spécification Web comme étant un triplet (R, I_n, I_m) dans lequel R, I_n et I_m sont des ensembles. R contient les définitions de certaines fonctions auxiliaires tel que par exemple add(X,Y) qui pourrait servir à retourner le résultat de l'addition de deux nombres naturels.

I_n décrit les contraintes (correctness rules) qui permettent de détecter les pages Web erronées. Ces règles ont la forme suivante :
l $\longrightarrow$ error | C. l est un terme, error est une constante réservée et C est une séquence d'équations et de tests d'appartenance en rapport avec un langage régulier (Ex. X $\in$ rexp). Voici un exemple de contrainte :
project(year(X)) $\longrightarrow$ error | X in [0-9]*, X<1990. Cette contrainte déclare

qu'une page Web est erronée si elle contient un projet plus vieux que 1990. Nous pouvons aussi définir des contraintes avec seulement un terme et la constante error comme dans l'exemple suivant : blink(X) $\longrightarrow$ error. Cette contrainte nous informe que dans toute l'AW nous ne devons pas retrouver du texte qui clignote.

L'ensemble I_m spécifie les propriétés de complétude (completeness rules) pour découvrir les pages Web incomplètes ou manquantes. Il définit certaines informations qui doivent être incluses dans certaines pages Web ou dans toutes les pages Web. Ces règles ont la forme suivante : l $\longrightarrow$ r [q], où l et r sont des termes et q $\in$ {E, A}. L'attribut A est utilisé pour identifier les règles universelles et E pour les règles qui s'appliquent seulement à certaines pages Web. Le côté droit des règles de pages complètes peut contenir des appels de fonctions définies dans R. Certains symboles du coté droit des règles peuvent avoir le symbole #. Le symbole # détermine le sous-ensemble qui doit être vérifié par la règle. Prenons l'exemple suivant :
member(name(X), surname(Y)) $\longrightarrow$ #hpage(fullname(append(X, Y)), status) [E]. C'est une règle qui s'applique seulement si une page Web contenant la liste des membres existe. Dans le cas où cette page des membres existerait nous devons retrouver pour chacun des membres au moins une page Web qui contient son nom complet et son état. Le symbole # placé devant hpage nous indique que cette propriété doit être vérifiée seulement sur les pages de type home page c.-à-d. les pages contenant la balise hpage.

Pour mieux comprendre le fonctionnement de cet outil nous avons créé une AW et une spécification en format Gverdi qui accompagne celle-ci. Nous ne pouvions pas utiliser notre AW de vente de livres en ligne puisque celle-ci est dynamique. Nous avons créé une petite page Web statique de vente de livres en ligne. L'AW contient cinq pages Web statiques. La page d'accueil affiche quatre livres à vendre. Pour obtenir plus de détail sur les livres à vendre l'usager peut activer le lien nommé détails situé à la droite de chacun des quatre livres à vendre. À l'aide de l'outil Gverdi nous vérifions un ensemble de propriétés qui nous semble être intéressant. Nous vérifions que pour tous les livres affichés sur la page d'accueil il existe une page Web contenant les détails sur ce livre à vendre.

Un aspect intéressant de cette méthode est que les auteurs définissent une AW comme étant un triplet (R, I_n, I_m). Ils s'attardent à deux grandes catégories d'erreur, les pages Web incorrectes et les pages Web incomplètes ou manquantes. Nous avons par contre noté quelques faiblesses avec la méthode GVerdi. Premièrement, l'écriture des spécifications demande l'apprentissage

d'un nouveau langage de programmation. La courbe d'apprentissage risque d'être grande pour les concepteurs d'AW qui ne sont pas familiers avec la programmation fonctionnelle. La lecture dans un fichier XML est limitée aux balises. L'outil ne nous permet pas de lire par exemple les attributs XML. Finalement, la plus grande faiblesse de ce système selon nous est qu'il ne s'applique seulement qu'aux pages Web statiques.

7.3 WAVE

Après avoir étudié la méthode Gverdi utilisée pour valider la qualité des AW statiques nous avons axé nos recherches sur des outils d'écriture de spécifications pour AW qui permettraient de vérifier automatiquement les propriétés dynamiques de celles-ci. Nous avons découvert un outil intéressant qui correspondait à nos critères de recherche soit l'outil WAVE [26, 27, 28]. Sur la page Web [31] les auteurs présentent leur système de création et vérification d'AW dynamiquement créées à partir de bases de données. Une des particularités de ce système est qu'il permet de vérifier automatiquement les propriétés temporelles des AW produites par ce système.

Nous pouvons voir sur la figure 7.1 que ce système est composé de quatre modules [26] : spécification, vérification, explication et générateur de code. Ce système fonctionne à l'aide de l'outil WAVE conçu par les auteurs. Le module générateur de code n'a pas encore été créé mais les auteurs projettent de l'ajouter éventuellement au programme WAVE. Le module de spécification prend un fichier texte, écrit avec la grammaire du langage de spécification WAVE, et le transmet au module générateur de code et au module de vérification. La spécification peut être écrite manuellement ou éventuellement elle pourra être importée à partir d'une spécification WebML. Le module générateur de code lit la spécification et crée l'ensemble des pages Web JSP qui forment l'AW spécifiée. Le module de vérification prend en entrée la spécification de l'AW et ses propriétés temporelles. L'outil WAVE simule ensuite des exécutions (run) de l'AW pour vérifier la validité de ces propriétés. Dans le cas ou une des propriétés s'avère être fausse le module d'explication fournit un contre-exemple.

Les auteurs mentionnent dans [27, 28] s'être inspiré de la méthode WebML pour créer leur système. Les auteurs travaillent présentement avec le groupe WebML pour créer un sous-module qui lira une spécification WebML (XML) et traduira celle-ci dans le langage de spécification utilisé par WAVE.

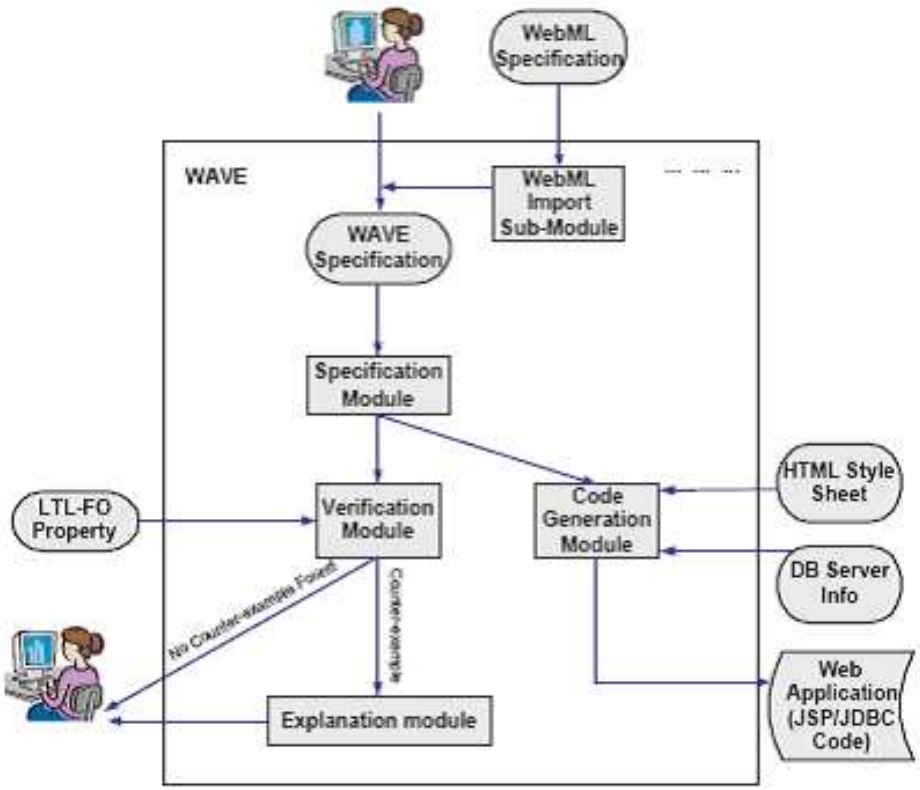

FIG. 7.1 – Modèle WAVE

Les auteurs présentent des nouvelles techniques de vérification automatiques pour services Web. Ils ont concentré leurs recherches sur les AW interactives qui génèrent dynamiquement des pages Web à partir de bases de données. L'AW prend en entrée des données fournies par des usagers et des programmes. Elle exécute certaines actions, met ses bases de données à jour et affiche la prochaine page Web définie à partir d'une requête.

Les propriétés vérifiées impliquent les entrées de données, les actions et les états pouvant découler des interactions avec des usagers. Pour exprimer ce genre de propriété les auteurs utilisent la logique temporelle. Ils utilisent deux types de propriétés. Le premier vérifie, en utilisant la logique temporelle linéaire, que toutes les exécutions possibles (runs) d'une AW satisfont une certaine condition sur la séquence d'entrée, actions et états. Le deuxième vérifie, en utilisant la logique temporelle de branchement (CTL), le bon fonctionnement d'un ensemble d'exécutions possibles effectuées simultanément. Les propriétés vérifiées vont de la logique de base (basic soundness) de la spécification (ex. La prochaine page Web affichée est toujours uniquement définie) aux propriétés sémantiques de l'AW (ex. Aucune commande n'est livrée avant que le paiement du montant exact n'est été reçu) [27]. La tâche du vérificateur WAVE est de s'assurer que toutes les exécutions d'un programme satisfont la propriété choisie. Le vérificateur cherche pour des contre-exemples.

La vérification des propriétés des AW est rendue possible par l'ajout de certaines restrictions qui limitent les entrées possibles (input boundedness) [28, 32]. En résumé, input boundedness signifie limiter la plage des entrées possibles. Par exemple pour exprimer que tout les paiements reçus sont du bon montant nous pouvons utiliser la formule suivante qui limite les entrées possibles (input-bounded)
$\forall$ x $\forall$ y [pay(x,y) $\longrightarrow$ price(x,y)], pay(x,y) est une entrée et price est une relation de base de données qui fournit le prix pour chaque produit [27].

Une spécification WAVE contient le nom de la base de données utilisée, les noms des tables, les noms des champs, l'ensemble des états possibles (State), la liste des interfaces d'entrées (Input), les constantes (Input Constant) et les schémas de page (Schema Set). Les états changent en fonction des entrées fournies par les usagers. Par exemple, le fait d'accéder la page administrative de notre AW de vente de livres fait passer l'état de celle-ci d'utilisateur à administrateur. La liste des interfaces d'entrées peut contenir des boutons, des liens et des tableaux contenant la liste des champs d'entrées. Par exemple dans la spécification WAVE de notre AW de vente de livres, dans la section In-

put, nous avons inscrit admininput(name,pwd) et clickbutton1(buttonname). Nous utilisons ces deux Inputs dans notre schéma de page Home_Page(HP) pour passer de la page Home_Page à la page Admin_Page. L'usager doit inscrire le nom d'administrateur, le mot de passe et ensuite activer un bouton pour atteindre la page Admin_Page. L'exemple complet se trouve dans l'annexe A. Dans les schémas de page Web on définit les règles d'entrée (Input Rules). Par exemple dans le schéma Home_Page nous avons mis la règle d'entrée Options@clicklink1(x) := x = "homepage". Cette règle déclare que l'activation d'un lien nommé homepage est une entrée valide sur la page Home_Page. Dans les schémas de page on trouve aussi les règles d'état (State Rules). Dans notre exemple, dans le schéma Admin_Choice_Page, nous avons définit la règle d'état user() := clicklink4("logout"). Cette règle stipule que l'activation du lien logout nous fait passer de l'état adminchoice à l'état user. Finalement dans les schémas de page nous pouvons mettre des règles d'accès (Target Rules) aux autres pages Web. Par exemple, dans notre schéma Details_Page la règle d'accès HP := clicklink2(" homepage "). Cela nous indique que l'activation du lien homepage nous transporte à la Home_Page (HP).

En résumé, une spécification WAVE contient pour chaque page Web les entrées possibles (Inputs), les règles d'entrée de données (Input Rules), les règles d'état (State Rules) et les règles pour accéder aux autres pages Web (Target Web Page Rules).

Les auteurs ont créé, en utilisant la méthode WAVE, un site Web de vente de produits informatiques. Nous présentons cet exemple pour mieux comprendre cette méthode. Sur la figure 7.2 [31] nous pouvons voir une des pages Web (Laptop Search Page) de cette AW et les pages Web auxquelles nous avons accès à partir de celle-ci.

La figure 7.3 [31] nous montre le pseudo code pour la spécification en format WAVE de la page Web Laptop Search. Les auteurs ont choisi de ne pas tenir compte des boutons "back" et "continue shopping" dans le pseudo code. Mentionnons que criteria est une base de données. Les acronymes HP, PIP et CC sont utilisés pour Home Page, Product Index Page et Cart Content Page respectivement.

En vertu de la première règle d'entrée de données, l'usager doit faire un choix entre "search", "view cart" et "logout". Nous pouvons voir que la première règle empêche un usager d'activer plus d'un bouton à la fois. La deuxième règle fait en sorte que la page Web fait une recherche dans la base de données criteria. La règle d'état (State Rule) stipule que dans le cas ou le bouton " search " est activé les choix de l'usager sont enregistrés dans la

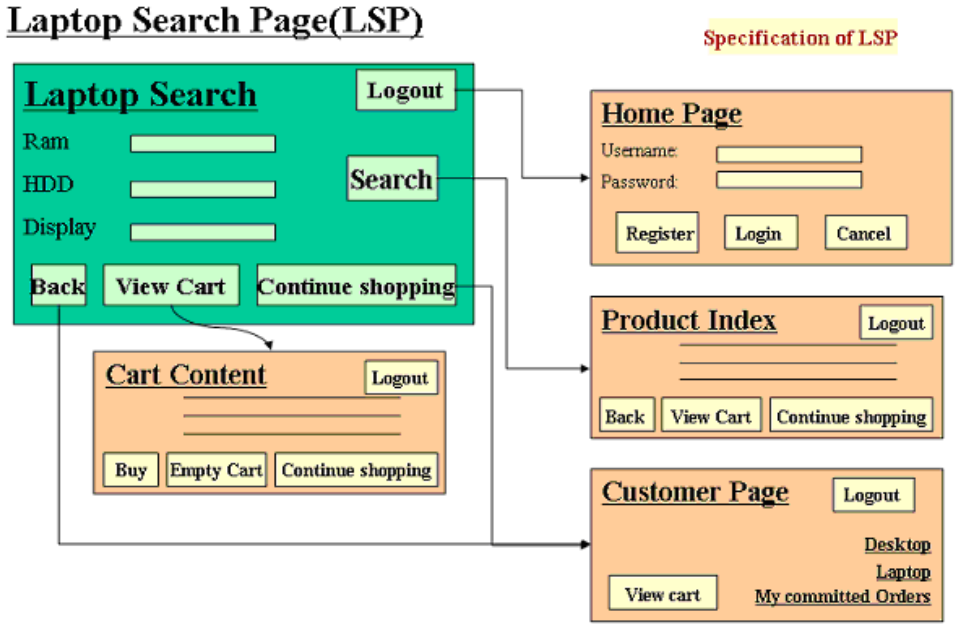

FIG. 7.2 – Spécification de la page Web LSP

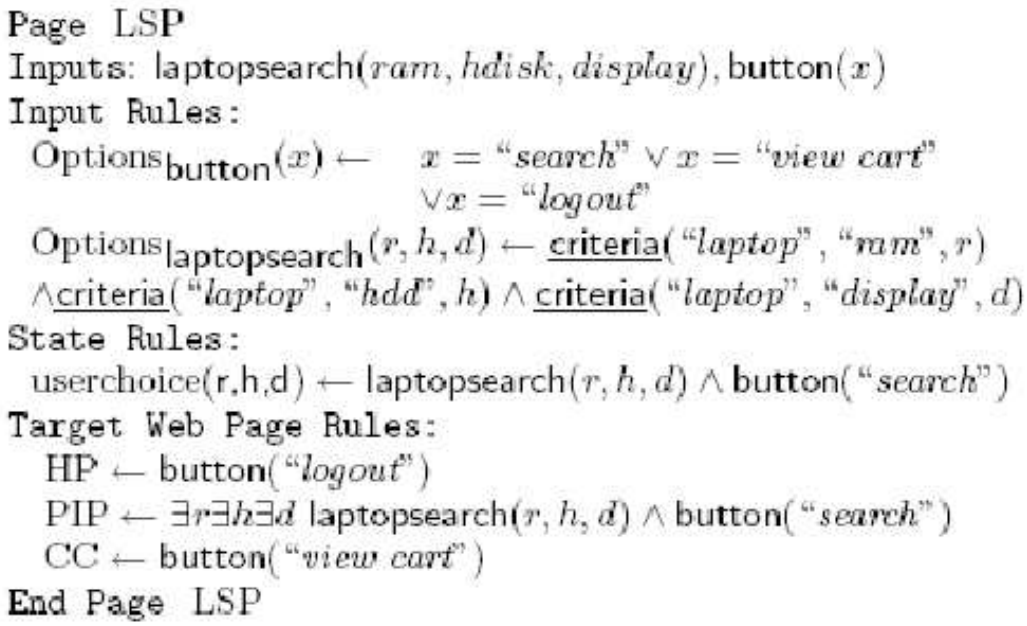

FIG. 7.3 – Pseudo code de la spécification de la page Web LSP

table userchoice. Lorsque la table userchoice contient des données dans les cases ram, hdisk et display, la deuxième règle de redirection des pages Web (Target Web Page Rule) renvoie l'AW à la page index des produits (PIP). Voyons maintenant à la figure 7.4 le code de la spécification de la page Web LSP. La spécification complète de l'AW de vente de produits informatique se trouve sur le site Web de WAVE [31].

```
Page: Laptop_Search_Page(LSP)

Input:        clickbutton4(x),  laptopsearch(r,h,d) Input Type: Menu

Input Rules:  Options@laptopsearch(r,h,d) : = criteria("laptop","ram",r)
                                              and criteria("laptop","hdd",h)
                                              and criteria("laptop","display",d);

              Options@clickbutton4(x) : = x = "back"
                                          or x = "continue_shopping"
                                          or x = "view_cart"
                                          or x = "search"

State Rules:  not category_laptop()  : = clickbutton4("back") and category_laptop() ;
              not category_laptop()  : = clickbutton4("continue_shopping") and category_laptop() ;
              not category_laptop()  : = clickbutton4("view_cart") and category_laptop()

Target Webpages: PIP, CP, CC,HP

Target Rules:  PIP : = exists r,h,d laptopsearch(r,h,d) and clickbutton4("search");
               CP : = clickbutton4("back");
               CP : = clickbutton4("continue_shopping");
               CC : = clickbutton4("view_cart")
```

FIG. 7.4 – Code de la spécification de la page Web LSP

Cette méthode est intéressante à plusieurs niveaux. Nous sommes surtout intéressés par la spécification et la méthode automatique de vérification des propriétés qui accompagne celle-ci. Les propriétés sont définies en utilisant une extension de la logique linéaire temporelle (linear-time temporal) [27]. Elles permettent de vérifier des propriétés sémantiques. Des propriétés de navigation peuvent aussi être vérifiées comme par exemple, nous pouvons nous assurer que la page d'accueil est accessible à partir de toutes les pages Web d'une AW. La robustesse d'une AW peut aussi être confirmée en s'assurant par exemple que la prochaine page Web est définie sans ambiguïté.

Des opérateurs temporels sont utilisés dans l'outil WAVE pour faire la vérification des propriétés des AW. Rappelons qu'en logique temporelle le symbole G signifie toujours et F signifie éventuellement. Continuons avec l'AW de vente de pièces informatiques. Pour vérifier que la page d'accueil est accessible à partir de toutes les pages Web de l'AW nous devons lancer cette commande dans l'outil WAVE : G (F hp()). Pour vérifier que lorsque la page d'erreur est accédée la prochaine page atteinte sera la page d'accueil ou la page d'erreur, nous utilisons cette commande :
G ! ep() || F (ep() && F (hp() || ep())). L'outil nous retourne alors succès ou un contre exemple.

Pour mieux comprendre le fonctionnement de cet outil nous avons écrit une spécification en format WAVE de notre page Web de vente de livres en ligne. Notre spécification et une représentation graphique de notre AW se trouve dans l'annexe A. Cet exercice nous a permis de mieux comprendre les bienfaits et les limites de cette méthode. Premièrement, nous avons réalisé que l'apprentissage de cette méthode est relativement difficile. Beaucoup d'effort a été requis pour écrire la spécification en format WAVE de notre AW qui est pourtant d'une complexité et taille relativement petite. Les auteurs soulignent eux même que l'écriture de la spécification des propriétés linéaires temporelles de premier ordre (LTL-FO) peut s'avérer difficile pour les non experts [27]. Ils ont par contre prouvé, à l'aide de deux exemples, que lorsqu'on comprend bien la méthode celle-ci peut être utilisée efficacement avec des AW de moyenne (Computer store) et même de grande taille (MotoGP) [31]. En comparant le graphique de notre AW sur la figure A.1 avec la spécification WAVE qui l'accompagne nous voyons que plusieurs fonctionnalités ne sont pas définies dans la spécification WAVE. Toutes les fonctions qui demandent à l'usager de fournir une entrée qui ne provient pas d'un ensemble d'entrées possibles incluses dans une table ne sont pas déclarées dans cette spécification. Les entrées possibles choisies par l'usager doivent être extraites d'une table. Par exemple, dans notre AW de vente de livres, le bouton rechercher de la Home_Page ne fait pas partie de la spécification. La raison étant que l'usager doit écrire le titre qu'il veut rechercher dans une case texte (input type="text") incluse dans un formulaire et ensuite il doit activer le bouton rechercher. Les titres ne proviennent pas d'une table donc la fonction rechercher n'est pas définie dans la spécification pour la page Home_Page.

Cet outil a été conçu pour être utilisé avec des AW dont les pages Web sont dynamiquement créées à partir de bases de données. L'ensemble des entrées possibles doit être connu au départ et accessible à partir d'une base

de données. Ces restrictions limitent le nombre d'AW pouvant bénéficier de cette méthode.

7.4 Webtest

Les deux méthodes d'écriture de spécification vues précédemment nous ont amené à réfléchir sur certains des critères intéressants qui devraient faire parti d'une spécification pour AW dont l'objectif principal est le contrôle de la qualité. Le plus grand obstacle que nous avons retenu face à leur utilisation est la complexité de celles-ci. Leur utilité demeure strictement académique. Les exemples de taille et complexité simple que nous avons faits avec ces deux outils nous ont demandé des efforts considérables pour premièrement comprendre la grammaire et ensuite la mettre en application. Il est improbable selon nous que ces outils soient utilisés dans un environnement réel de production d'AW. Ceci nous a emmené à orienter notre recherche vers un outil qui combinerait efficacité et simplicité d'utilisation. Webtest est un outil qui correspond bien à ces critères.

Dans les articles [13, 29] les auteurs présentent une méthode d'écriture de spécifications formelles de suites de tests pour AW. Ces spécifications permettent de tester différents aspects des AW incluant les fonctionnalités, le niveau de sécurité et la performance attendue des AW. Elle contient des suites de tests, des jeux d'essais et des étapes de tests. L'usager doit spécifier les entrées et sorties attendues de chacun des jeux d'essais.

Deux types de prédicats sont utilisés : simple et composé (compound). Trois prédicats de type simple sont définis : MatchPred, ContainPred et ComparisonPred. MatchPred vérifie le contenu du document de réponse. ContainPred vérifie la présence ou non de certains éléments tel que <href>, <form>, <table>, etc. MatchPred est utilisé pour vérifier que certains objects tel que string et date par exemple sont équivalents. Nous pouvons voir ci-dessous un exemple de spécification qui utilise des prédicats simples fournis en entrée à l'outil Webtest. Les spécifications sont écrites dans le format XML. Dans cet exemple on vérifie le contenu de la page http ://www.cs.depaul.edu/program pour vérifier si elle contient certains mots clefs.

```
<testcase name="Content check using predicates">
  <teststep name="Content check step one">
    <request url="http://www.cs.depaul.edu/program"/>
```

```
      <response statuscode="200">
        <and>
         <or>
          <match op="contains" regexp="false"
            select="/html/body" value="Undergraduate Degree"/>
          <match op="contains" regexp="false"
            select="/html/body" value="Bachelor Degree"/>
         </or>
         <match op="contains" regexp="true"
           select="/html/body value="[M|m]aster [D|d]egree"/>
        </and>
      </response>
  </teststep>
</testcase>
```

Les prédicats composés peuvent être créés en utilisant les connecteurs logiques non, et, ou et implique. Deux prédicats de quantification sont utilisés dans le langage de spécification soit forall et exists. Voici un exemple de spécification qui sert à vérifier que toutes les URL qui sont référencées le sont de façon absolue.

```
<testsuite>
  <testcase name="Content check using quantified predicates">
    <teststep name="content check step one" >
      <request url="http://jordan.cs.depaul.edu/webtest/testhref.htm">
        </request>
          <response>
            <forall>
              <varbale name="l" select="descendent::a/@href"/>
              <match op="startswith"
                          select="$l"
                          regexp="false"
                          value="http://"/>
            </forall>
          </response>
    </teststep>
  </testcase>
</testsuite>
```

Nous avons retenu de cet article l'idée d'utiliser httpunit pour concevoir des suites de tests.

7.5 Conclusion

L'utilisation des méthodes d'écriture de spécification Web présentées dans ce chapitre permet d'accroître la qualité des AW. Les attributs de qualité analysés par ces outils sont très variés. Aucun outil d'écriture et de vérification automatique et semi-automatique de spécification Web ne peut couvrir tous les aspects qualité d'une AW. Par contre elles apportent tous une bonification au niveau du contrôle de la qualité des AW.

L'apport positif des spécifications axées sur le contrôle de la qualité n'est pas suffisant pour justifier leur utilisation. L'effort d'apprentissage requis, leurs champs de couverture limités et le maintien de ces spécifications sont autant d'obstacles à leur utilisation. Une spécification fonctionnelle devrait par contre accompagnée toute AW.

Chapitre 8

Validation de la qualité des applications Web

8.1 Introduction

Dans la suite, nous proposons une méthodologie de validation de la qualité des AW. Un ensemble de tests et observations, passés avec succès, témoigne de l'existence d'une AW de qualité. Lorsque cela s'avère possible, nous suggérons l'utilisation d'outils automatiques et semi-automatiques. Les outils code source libre et gratuiciels sont favorisés par rapport aux produits commerciaux. Pour ce qui est des critères de qualité, les standards de qualité reconnus internationalement ont prépondérance sur les autres. Notre méthodologie a été mise à l'essai sur notre AW de vente de livres.

Définir ce qu'est une AW de qualité est une tâche relativement complexe. Dans plusieurs des articles que nous avons mentionnés jusqu'ici les auteurs ont chacun leur propre définition d'une AW de qualité. Après avoir analysé une liste significative d'articles sur ce sujet nous en sommes venus à la conclusion qu'il n'existe pas de définition unique de ce qui constitue une AW de qualité. Ceci s'explique en partie par le fait que les AW peuvent remplir une multitude de fonctions différentes les unes des autres et les critères pour mesurer la qualité de ces fonctions diffèrent beaucoup. Par exemple pour une AW de vente d'articles en ligne, la sécurité du site sera un critère très important à valider. Pour une AW de type portail, il se peut que ce soit le bon fonctionnement des liens qui revêt une plus grande importance.

L'existence ou non d'une spécification influence aussi la validation de

la qualité d'une AW. Lorsqu'une spécification existe, des critères de qualité spécifiques à l'AW en question peuvent être définis. Dans le cas contraire nous devons nous en tenir à des critères de qualités plus générales. L'accès à une spécification permet aussi de mettre sur pied des tests d'intégration plus détaillés. Il y a toujours la possibilité, si nous avons accès au code, d'analyser celui-ci pour essayer de comprendre les liens reliant les différents modules entre eux. Cette tâche peut s'avérer ardue et même irréalisable dans le cas d'AW de moyenne et grande taille.

L'utilisation d'un modèle de conception d'AW nous amènera à mettre en place une série de tests qui sera différente de celle utilisée pour la création de la même AW sans l'aide d'un tel modèle. Une fois le modèle validé, plusieurs tests n'ont pas besoin d'être faits puisque les règles de conception garantissent la conformité de l'AW ainsi produite vis à vis certains critères de qualité.

Il existe différentes façons de séparer les types de test de qualité pour AW. En voici quelques-unes :

- Ils peuvent être divisés en fonction des différentes étapes complétées pendant la durée de vie du développement des AW. Prenons par exemple le cas d'une AW construite à partir d'un cycle de vie en V. Les types de tests pourraient être scindés de cette façon et dans cet ordre : unitaires, intégrations, acceptations et tests système. Cette approche, quoi que valable pour les logiciels traditionnels, s'applique moins bien aux AW. Les AW de moyenne et grande taille sont majoritairement en constante mutation. Les changements vont des simples changements esthétiques aux ajouts et suppression de fonctionnalités. Contrairement aux logiciels traditionnels, plusieurs changements sont régulièrement faits en même temps et la documentation, lorsqu'elle existe, est souvent primaire.
- Une séparation peut aussi être faite dans la manière d'exécuter les tests : manuels, automatiques, semi-automatique, etc.
- Nous pouvons dissocier les tests fonctionnels (liens, base de données, etc.) des tests non fonctionnels (convivialité, sécurité, etc.).

Bien que toutes toutes ces approches soient valables nous avons décidé de séparer les types de tests en trois catégories : tests de type boîte noire, tests de type boîte transparente et tests de type boîte grise. Le niveau de connaissance de l'AW détermine dans quelle(s) catégorie(s) de test une personne peut s'impliquer. Les tests de type boîte noire peuvent être faits par quiconque. Ceux de type boîte blanche requièrent d'avoir accès au code interne de l'AW. Finalement les tests de type boîte grise doivent être faits par

des gens qui, sans nécessairement avoir accès au code, connaissent le fonctionnement interne de l'AW (architectures, algorithmes, etc.). Cette division nous apporte à concevoir nos tests en adoptant différents points de vue : webmestre, programmeur, simple usager, etc. Une fois la série de tests écrite et la façon de séparer les types de tests établie, nous devions ensuite choisir de quelle façon nous voulions utiliser les résultats.

Nous n'avons pas défini un système de pointage sur la qualité des AW comme certains l'ont fait [2, 22, 33]. Notre objectif n'est pas de noter le niveau de qualité des AW mais bien de déterminer si une AW rencontre tous les critères requis pour être considérée de grande qualité. Nous voulons, par une série de tests, faire ressortir les lacunes qui empêchent une AW d'atteindre un niveau de qualité supérieur. La suite logique est de remédier à ces carences en apportant les changements qui s'imposent. Il est impossible d'énumérer tous les tests possibles et imaginables pour atteindre un niveau de qualité optimale qui s'appliquerait à tous les genres d'AW. Par contre nous croyons que toute AW qui passe avec succès la totalité des tests du présent chapitre peut être considérée comme en étant une de qualité.

Notre méthodologie de validation de la qualité des AW consiste à observer et tester un ensemble de propriétés des AW qui ont un impact direct sur le niveau de qualité des AW. L'utilisateur doit premièrement parcourir l'ensemble des propriétés de notre méthodologie et déterminer celles qui s'appliquent à l'AW à être validée. Pour chacune des propriétés énoncées, une solution est proposée pour s'assurer que cette propriété respecte les standards de qualité attendus. Pour qu'une AW soit considérée de grande qualité, toutes les propriétés présentes doivent répondre adéquatement aux solutions établies pour chacune d'entres elles.

8.2 Tests de type boîte noire

8.2.1 Introduction

Tous les concepteurs d'AW font au minimum une validation de leur AW en exécutant leur AW à quelques reprises et en vérifiant s'ils obtiennent les résultats attendus basés sur la documentation qui accompagne l'AW. Pour exécuter des tests fonctionnels nous devons connaître quelles sont les entrées et sorties attendues d'un programme. Dans le meilleur des cas une spécification formelle de l'AW existe et elle est alors utilisée pour définir

l'oracle. C'est un exemple de test de type boîte noire. La majorité des articles sur la validation des AW font référence aux approches de tests de type boîte noire. Il existe des outils qui vérifient certaines propriétés des AW sans relation directe avec la documentation. Ces outils ne tiennent pas compte de la sémantique. Ils analysent des propriétés syntaxiques. L'outil browsershots par exemple nous permet de vérifier si une AW fonctionne bien dans l'ensemble des navigateurs les plus utilisés.

8.2.2 Standards d'écriture

La transmission d'information textuelle est l'activité principale de la majorité des AW. Les documents Web doivent être écrits correctement pour qu'une AW puisse être considérée de qualité. Certains standards d'écriture Web doivent aussi être respectés. Ignorer ces standards risque de rendre l'affichage imprévisible. Plus important encore, certaines personnes peuvent être privées d'accès à ces AW dû à leurs incapacités physiques. L'environnement hétérogène des AW renforce ce besoin de standardiser l'écriture.

Propriété à vérifier

Les textes affichés par une AW doivent être écrits correctement aux niveaux orthographique et grammatical.

Solution proposée

Tout texte destiné à être affiché sur une page Web doit préalablement être corrigé avec un logiciel de type correcteur orthographique et grammatical. La majorité des éditeurs de texte les plus utilisés sont fournis avec un tel correcteur. Il existe aussi des correcteurs gratuits (ex : http ://bonpatron.com/) qui peuvent être utilisés. Par contre pour faire une bonne révision orthographique et grammaticale de documents français, nous suggérons l'utilisation du logiciel antidote [34].

L'utilisation d'un logiciel correcteur ne doit pas être perçue comme un substitut à une révision humaine. Une lecture attentive par une ou des personne(s) qualifiée(s) est nécessaire surtout pour capter les erreurs grammaticales. Ceci est particulièrement vrai dans le cas d'une langue complexe comme le français. Une page parsemée de fautes aura tôt fait d'irriter plusieurs usagers. Leur niveau de confiance dans le contenu de l'AW pourrait alors être ébranlé au point de ne plus revenir sur cette AW dans le futur. Ceci démontre bien toute l'importance que nous devons lui accorder.

Lorsque cela s'applique nous devrions aussi toujours afficher les dates des dernières modifications apportées aux documents affichés sur les pages Web.

Balisage du texte Web

Au niveau des standards d'écriture, le balisage du texte Web doit respecter les standards mondialement reconnus qui sont ceux du World Wide Web Consortium [24]. Cet organisme a publié jusqu'à maintenant plus de 90 standards Web. De ceux-ci nous avons retenu trois standards de syntaxe qui nous apparaissent comme étant essentiels et qui s'appliquent à la majorité des AW.

Propriété à vérifier
Les codes HTML et XHTML doivent répondre aux exigences du standard du W3C.

Solution proposée
Utiliser l'outil source libre MarkUp Validator [35].

Propriété à vérifier
Lorsque l'AW contient des feuilles de style en cascade (CSS), elles doivent rencontrer les exigences du W3C.

Solution proposée
Utiliser l'outil CSS Validator [35].

Propriété à vérifier
Le contenu des AW doit être accessible aux personnes atteintes d'un ou plusieurs handicaps.

Solution proposée
Trois niveaux de conformité au standard Web Accessibility Initiative (WAI) peuvent être atteints. Le niveau A signifie que les exigences minimales ont été respectées pour rendre l'AW accessible à certains groupes de personnes. Par exemple toutes les images sont accompagnées d'une description. Le niveau AA englobe le niveau A. De plus, le niveau AA permet de rendre plus aisée l'accès à certaines parties d'une AW. Voici un exemple d'exigence pour atteindre le niveau AA : les liens doivent avoir un nom significatif. Il faut éviter, par exemple, de nommer un lien "cliquez ici". Finalement l'adhérence

au niveau AAA, en plus de se conformer aux critères deux premiers niveaux, signifie que le maximum a été fait pour rendre l'accès à une AW le plus simple possible pour tous. Une AW, pour être considérée de qualité, doit au minimum respecter le niveau A du standard WAI.

Déterminer le niveau du standard WAI d'une AW peut s'avérer une tâche ardue. Pour alléger cette tâche, il existe plusieurs logiciels pour analyser automatiquement une partie des critères requis pour atteindre ce niveau. Le site WAI suggère plus d'une centaine de logiciels d'évaluation des différents critères des AW qui rendent celles-ci accessibles aux personnes atteintes d'un handicap. Nous voulions des outils de type source libre, efficaces et simples à utiliser. Deux de ces logiciels répondent à ces critères : WAVE [43] et Web Accessibility Checker [44].

L'outil WAVE affiche la page Web analysée en ajoutant des commentaires aux endroits où se trouve de l'information utile aux personnes handicapées et aux endroits où il y a des carences. WAVE permet de visualiser les différentes pages Web du point de vue d'une personne aux capacités physiques limitées.

L'outil Web Accessibility Checker (WEC) vérifie un ensemble de critères statiques présents sur les pages Web. Ces critères ont un impact sur l'accès qu'offrent celles-ci aux personnes ayant un handicap. Avec WEC nous pouvons analyser les pages Web en fonction de neuf critères de qualité reconnus mondialement.

Fureteurs

Dans la validation du standard d'écriture nous incluons la vérification de la compatibilité d'AW avec différents fureteurs. Nous examinons si les mêmes résultats sont obtenus avec les différents fureteurs pour un même URI.

Propriété à vérifier
L'affichage du contenu offert par une AW doit être très semblable dans tous les fureteurs les plus utilisés.

Solution proposée
Nous suggérons l'utilisation de browsershots [45] pour vérifier l'uniformité du contenu des AW peut importe le fureteur utilisé. Ce site offre la possibilité de visualiser l'affiche d'une page Web sur près de soixante fureteurs différents. Pour qu'une AW soit considérée de qualité nous devons tester l'affichage sur cinq différents fureteurs les plus utilisés [46] : IE, Firefox (différentes

plateformes), Mozilla (différentes plateformes), Safari et Opéra.

8.2.3 Propriétés de connexion

La validation des propriétés de connexion est une partie importante du contrôle de la qualité des AW. Leur bon fonctionnement est essentiel mais d'autres aspects doivent aussi être vérifiés. Nous devons entre autre s'assurer qu'aucune page ne soit isolée des autres dans une AW. Une autre propriété de connexion importante selon nous du point de vue de la qualité est le modèle de navigation utilisé. Pour que l'expérience de navigation entre les différentes pages soit conviviale il doit y avoir une certaine uniformité dans la façon de naviguer à l'intérieur d'une AW. Dans le cas contraire un usager qui souhaite retrouver une certaine page après en avoir visité quelques unes risque de devoir procéder par essai erreur sur plusieurs liens avant de retrouver la page voulue.

Liens

Une AW contenant des liens brisés est souvent perçue comme abandonnée et désuète d'où l'importance de valider les liens. La vérification du bon fonctionnement des différents liens est simple à faire mais essentielle.

Propriété à vérifier
Tous les liens d'une AW doivent être fonctionnels.

Solution proposée
Nous proposons l'utilisation de l'outil code source libre Link Checker [47] propriété du World Wide Web Consortium [24]. Cet outil lit le code des pages Web et extrait une liste d'ancres et de liens. Il vérifie qu'aucune ancre n'est définie plus d'une fois. Ensuite il contrôle tous les liens pour s'assurer qu'ils fonctionnent. Il nous informe s'il y a des pages ou des répertoires qui nous redirigent vers un autre lien lorsqu'on fait appel à eux.

Nous n'avons aucun contrôle sur le bon fonctionnement des liens vers des AW externes. Ces AW que nous référençons peuvent cesser d'exister ou encore changer d'adresse à tout moment. C'est pour cette raison que nous suggérons de faire ce test une fois par semaine.

Link Checker peut être utilisé en ligne ou localement. Pour utiliser Link Checker localement, en plus de télécharger le logiciel Link Checker, nous de-

vons avoir le logiciel de programmation code source libre Perl d'installer. De plus un ensemble de modules Perl, variant selon la version de Perl présente, doivent être installé et configuré. La version en ligne devrait toujours être privilégiée lorsque possible. Aucune configuration n'est à faire ce qui diminue le risque d'erreurs.

Pages isolées

Normalement des changements sont apportés régulièrement sur les AW. Certaines pages Web sont modifiées, ajoutées ou supprimées de l'AW et parfois des erreurs se produisent qui font que certaines pages se retrouvent déconnectées des autres. Un exemple de situation où ce genre d'erreur pourrait ce produire est lors du changement de nom d'une page Web. Dans le cas où un seul lien rejoint cette page au reste de l'AW et que le nom du lien rejoignant ces deux pages n'est pas modifié cette page deviendra isolée donc non accessible aux clients. Une page isolée peut occasionner des conséquences fâcheuses et l'impact négatif risque d'être encore plus grand lorsqu'une page isolée se trouve être le point d'attache de plusieurs autres pages. Toutes les pages Web doivent être atteignables à partir de la ou des page(s) racine(s). Les pages racines sont les pages de départ d'une AW (ex. index.html). Les pages racines peuvent avoir un lien direct vers toutes les pages ou encore plusieurs pages peuvent être reliées les une aux autres. Un usager doit pouvoir atteindre toutes les pages Web de l'application en débutant avec la ou les pages racines en utilisant les liens qui existent entre les différentes pages. Dans le cas d'une AW de petite taille la tâche de vérifier qu'aucune page Web n'est inatteignable est simple. Il suffit de naviguer dans l'AW et de tenter d'atteindre toutes les pages Web. Nous pouvons par la suite déterminer s'il existe ou non une ou des page(s) isolée(s). Par contre lorsqu'une AW contient plusieurs pages Web le risque d'erreur devient plus grand.

Propriété à vérifier
Nous devons s'assurer qu'il n'existe pas de pages Web isolées.

Solution proposée
Pour faciliter la vérification et pour diminuer le risque d'erreurs nous avons créé le programme PagesInatteignables.java que nous avons mis dans l'annexe C. L'usager doit fournir en entrée le nom de la ou des page(s) racine(s), le nom des pages Web et les liens existant entre les pages Web.

Nous avons utilisé notre programme pour vérifier l'AW de vente de livres en ligne que nous avons créé avec WebRatio. WebRatio produit automatiquement une spécification en même temps qu'il crée les différentes pages Web qui composent une AW. Nous avons pris avantage de cette caractéristique pour automatiser davantage la vérification des pages isolées dans WebRatio. Nous avons écrit dans l'annexe E des requêtes Xquery qui servent à récupérer, à partir de la spécification produite par WebRatio, les données qui doivent être fournies en entrée dans notre programme PagesInatteignables.java. Ces requêtes récupèrent les noms de toutes les pages Web, la ou les page(s) racine(s) et la liste des pages qui sont reliées entre-elles. Ces requêtes et un exemple sont inclus dans l'annexe E. Cet exercice nous a permis de démontrer qu'il est parfois possible d'automatiser ce genre de vérification. Il faut pour cela qu'il existe une façon standard de répertorier les pages Web. C'est souvent le cas pour les modèles de conception d'AW. Une AW de qualité doit être dépourvue de page(s) isolée(s).

Modèle de navigation

Un modèle de navigation standard doit être utilisé dans chacune des sections d'une AW. Ceci afin d'éviter de confondre l'utilisateur dans les possibilités de navigation qui lui sont offertes. Prenons l'exemple simple d'une AW qui offre de l'information sur une centaine de sujets différents. La page d'accueil regroupe un ensemble de liens vers chacun de ces sujets. Chaque fois qu'un usager active un lien sur la page d'accueil, il est redirigé vers une page Web d'information sur le sujet en question. Un choix de conception pourrait être d'inclure dans chacune des pages une table des matières avec liens vers les différentes sections de cette table. Nous pourrions aussi décider d'omettre la table des matières mais de mettre à la place un bouton suivant sur chaque page. L'usager devrait alors accéder aux pages en suivant l'ordre établit par les concepteurs de l'AW. Dans cet exemple il ne serait pas souhaitable de retrouver un mélange des deux approches de navigation. Nous ne voulons pas confondre le client sur ces possibilités de navigation d'une fois à l'autre. La problématique devient plus grande au fur et à mesure où nous rajoutons d'autres façons différentes de naviguer à l'intérieur d'un même module de l'AW. Dans l'article [17] les auteurs signalent qu'il existe différents modèles de navigation entre les pages Web. Les auteurs présentent quelques exemples de modèles de navigation : arbre, hiérarchique, diamant, connections complètes et séquences indexées. Nous pouvons voir ces modèles sur la

figure 8.1.

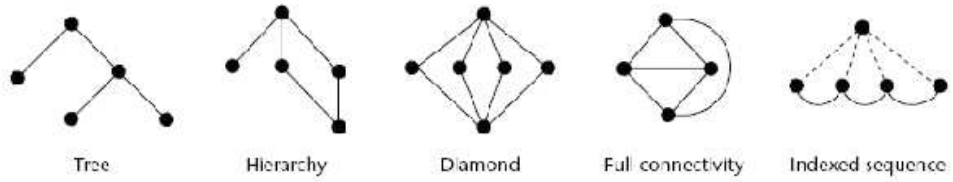

FIG. 8.1 – Exemples de modèles de navigation

Pour déterminer la forme de navigation d'une AW nous ne tenons pas compte des liens tels que les boutons home et précédent que nous pouvons trouver sur certaines pages. Bien qu'ils aient un impact réel sur les possibilités de navigation offertes, nous choisissons de les ignorer. Leur inclusion nous empêcherait de voir le modèle de navigation subjacent. Vu leur grande utilisation le fait d'inclure ces boutons nous amènerait à conclure que la forte majorité des AW utilise le modèle connections complètes. L'analyse s'arrêterait là sans apporter rien de significatif. Les noeuds représentent les pages Web et les arêtes les liens entre ces pages. Une AW utilise le modèle en forme d'arbre lorsque les liens entre les pages sont acycliques et que toutes les pages possèdent une seule page parent à l'exception de la page de départ qui n'en possède pas. Le modèle hiérarchique est lui aussi acyclique mais il contient au moins une page qui possède plus d'une page parent. Le modèle en forme de diamant est caractérisé par le fait qu'il y a une seule page d'entrée et une seule page de sortie. Le modèle de navigation d'une AW est considéré de type connections complètes lorsque toutes les pages possèdent un lien direct les unes envers les autres. Le modèle séquences indexées est utilisé lorsque chaque page nous donne accès à la prochaine page. Optionnellement chaque page peut aussi donner accès à sa page précédente. Finalement pour qu'une AW puisse prétendre utiliser le modèle séquences indexées, une page index doit être présente et celle-ci doit avoir un accès direct à toutes les pages Web qui composent l'AW. Chaque des modèle s'apparente à des styles de recherche différents. Par exemple les modèle arbre et hiérarchique correspondent à un style de recherche qui part d'une idée générale et va en se concrétisant de plus en plus à chacune des nouvelles pages Web atteintes. C'est une approche très différente du modèle connections complètes qui donne accès à l'information complète à partir de chaque page. Il n'y a pas de raffinement de la recherche

comme dans le modèle arbre et hiérarchique. Il existe d'autres modèles de navigation mais nous croyons que ceux présentés par [17] couvrent la majeur partie des modèles qui risque d'être utilisé par les concepteurs d'AW.

Propriété à vérifier
Les possibilités de navigation offertes par l'AW doivent respecter un certain standard de navigation à l'intérieur de chacune des sections de l'AW.

Solution proposée
Le fait de se conformer à un modèle facilite la navigation entre les pages pour un usager. Cette vérification peut s'avérer ardue surtout dans le cas d'AW de moyenne et grande taille. Le plus important à retenir ici est que nous devons retrouver une certaine consistance dans la façon de naviguer à l'intérieur d'une même section Web. Il faut que l'expérience de navigation soit plaisante pour l'usager. Celui-ci ne devrait pas avoir à s'ajuster à différentes façons de naviguer à l'intérieur d'un même module de l'AW. Pour faciliter la vérification à savoir si une AW peut être identifiée à un des modèles présentés ici nous avons créé trois programmes de vérification de modèle. Ces programmes se trouvent dans l'annexe B. Nous avons créé un programme pour vérifier le modèle arbre (AppWebNavigationArbre.java), hiérarchique (AppWebNavigationHierarchique.java) et connections complètes (AppWebNavigationFullConnectivity.java). Pour les autres modèles nous devons reproduire graphiquement l'ensemble des pages Web de l'AW et représenter les liens les unissant. Une analyse visuelle du graphique permettra de différencier les modèles de navigations qui respectent un modèle de navigation standard des autres.

8.2.4 Épreuve de charge

Propriété à vérifier
Il est difficile de prévoir combien d'usagers accèderont en même temps à une AW. Différents facteurs peuvent faire fluctuer grandement les demandes d'accès à une AW. Nous devons connaître les limites au delà desquelles une AW ne peut plus fournir le service attendu. Nous devons analyser l'impact qu'aura sur une AW le dépassement de ces limites. Quel genre d'erreurs risquons-nous d'avoir et comment ces erreurs sont prises en charge.

Solution proposée

Des épreuves de charge (load test) doivent être faites pour vérifier la solidité d'une AW. Nous exécutons plusieurs accès simultanés aux différentes pages et aux différents services offerts par une AW. Nous vérifions le maximum d'accès simultanés possibles, le temps de réponse des pages et la réaction de l'AW face aux différentes erreurs causées par la surcharge d'accès. Nous regardons ensuite si le nombre d'accès simultanés maximum atteint est satisfaisant pour notre AW. Ensuite nous devons déterminer si le temps de réponse de l'AW est suffisamment rapide. Un temps de réponse de plus de quelques secondes fera fuir les clients. Les traitements appliqués aux différentes erreurs causées par une surcharge de requêtes doivent aussi être étudiés. Nous analysons si le traitement des erreurs correspond bien à se qui est attendu. Dans le cas d'erreur un message doit être envoyé au client pour l'informer du problème. Plusieurs outils de type source libre permettant d'accomplir ce genre de test sont disponibles sur la page Web Software QA/Test Resource Center [30].

8.3 8.3 Tests de type boîte transparente

8.3.1 Introduction

Les tests de type boîte transparente sont moins communs dans le monde des AW en comparaison de ce que nous retrouvons dans le contrôle de la qualité des applications traditionnelles. Ces tests sont pourtant tout aussi importants que ceux de type boîte noire. Il est à noter que les méthodes utilisées dans les applications traditionnelles pour couvrir tout le code s'appliquent aussi aux AW. Malgré leur plus grande rareté, certains articles sur le sujet des tests de types boîte transparente appliqués aux AW méritent d'être mentionnés. Pour commencer dans l'article [48] les auteurs proposent une méthode de représentation de haut niveau UML pour les AW. Ce modèle peut être utilisé pour définir des critères de test de type boîte blanche et générer de façon semi-automatique les suites de tests. Les auteurs ont inventé les outils ReWeb et TestWeb pour analyser et tester les AW. ReWeb construit un modèle UML à partir d'AW en pratiquant de l'ingénierie inverse (reverse engineering) et TestWeb génère et exécute un ensemble de suites de tests à partir du modèle UML produit par ReWeb. Une approche similaire est présentée dans [36, 37].

8.3.2 Visibilité sur l'Internet

La majorité des personnes et entreprises souhaitent attirer le plus grand nombre de visiteurs vers leur AW lorsqu'ils publient celle-ci sur l'Internet. La visibilité d'une AW constitue un critère de qualité important. Une AW, disponible sur Internet, bien construite et respectant tous les standards de qualité mais qui n'est visitée par personne n'est pas très utile. Les AW doivent être facilement repérables par les différents moteurs de recherche. Elles doivent aussi posséder certains critères les rendant plus enclin d'obtenir un classement favorable dans ces mêmes moteurs de recherche. Les algorithmes utilisés pour faire des recherches diffèrent d'un moteur de recherche à l'autre. Par le fait même il n'existe pas de règles uniques assurant un classement favorable aux AW suite à une recherche dans les moteurs de recherche. Par contre il y a des règles de bases à respecter, pour augmenter les probabilités pour une AW d'obtenir un haut classement dans les moteurs de recherche les plus utilisés. Le respect de ces règles est requis pour qu'une AW soit considérée de qualité.

Propriété à vérifier
Toutes les pages Web d'une AW doivent avoir un titre.

Solution proposée
Nous devons vérifier que chaque page Web possède un titre significatif. Le titre est le texte assigné par défaut pour identifier une page ajoutée aux favoris dans les principaux fureteurs. C'est aussi un critère important utilisé pour indexer une page Web. Plus important encore, les moteurs de recherche utilisent le contenu de la balise title comme lien vers une page Web [38, 39, 40]. Un titre court et précis doit être privilégié.

Propriété à vérifier
Toutes les pages Web publics doivent posséder la balise meta avec l'attribut description.

Solution proposée
À la suite du lien formé du titre se trouvent quelques lignes de texte provenant de la page Web répertoriée. Dans le cas où aucune balise meta n'est présente le moteur de recherche affichera la plupart du temps les premières lignes de texte provenant de la page Web [40]. Pour éviter ceci toutes les pages Web, que nous voulons rendre visible, devraient contenir la balise meta avec

l'attribut description. Le contenu de la description sera alors affiché dans les moteurs de recherche. De plus cette information est une des données utilisées pour déterminer le classement d'une page Web.

Propriété à vérifier
Toutes les pages Web publiques doivent posséder la balise meta avec l'attribut keywords.

Solution proposée
Une autre balise meta doit être présente sur toutes les pages Web que nous voulons indexer par les moteurs de recherche. C'est la balise meta avec l'attribut keywords. Nous devons retrouver dans cette balise un maximum de cinq mots [40]. Ces mots clé sont utilisés par certains moteurs de recherche pour classer les pages Web. Cette balise a moins d'impact sur le classement depuis quelques années mais nous croyons cependant qu'elle doit toujours être présente car certains moteurs de recherche l'utilisent toujours.

8.3.3 Couverture du code

Si l'AW crée des pages dynamiques avec un langage de programmation impératif tel que ASP, PHP ou JSP, les critères de couverture d'une AW peuvent être établis en prenant en considération les noeuds, branches et chemins possibles du code source. Les noeuds symbolisent ici les différentes instructions. Les branches représentent les instructions suivantes. Créer une suite de tests qui couvrirait tous les chemins est impossible à réaliser exception faite de quelques rarissimes cas. Une suite de tests qui inclut tous les noeuds rajoute au niveau de confiance que nous pouvons avoir envers une AW. Une couverture de toutes les branches est une condition essentielle pour créer une suite de test de qualité d'une AW. C'est une méthode souvent utilisée dans l'univers des applications traditionnelles et elle s'applique bien aux AW.

Propriété à vérifier
Chaque instruction (noeud) du code source doit être exécutée au moins une fois.

Solution proposée
Il faut créer une suite de tests qui parcourt toutes les instruction de l'AW. Pour s'assurer que notre suite de tests soit valable, une solution serait de ra-

jouter des instructions dans les pages Web dynamiques permettant de suivre la trace des pages visitées (instrumenter le code source). Ces informations peuvent être transmisses dans un journal situé sur le serveur. Cette méthode, quoi que valable, demande beaucoup d'effort et de temps. Avant d'instrumenter le code, il faut s'interroger à savoir s'il n'existe pas une façon plus simple d'arriver au même résultat.

Par exemple, en PHP, il existe une méthode simple de vérifier quelles instructions ont été visitées. L'extension PHP code source libre Xdebug [41] permet de faire cette vérification automatiquement sans devoir modifier le code des pages Web. Un fichier texte nous enregistre les pages Web visitées et les fonctions PHP utilisées à l'intérieur de ces pages PHP. Nous avons mis dans l'annexe D les instructions pour installer et configurer Xdebug.

Propriété à vérifier
Chaque branche de l'AW doit être traversée au moins une fois.

Solution proposée
Pour confirmer que toutes les branches dynamiques d'une AW ont été visitées, nous pouvons inscrire dans un journal les instructions qui ont été exécutées. Pour ce faire, nous suggérons l'outil Xdebug.

L'ajout de deux lignes de code dans chacune des pages PHP suffit pour suivre la trace de toutes les branches visitée. Voici les deux lignes de codes à ajouter aux fichiers PHP : Xdebug_start_code_coverage() ; // Ajoutez au début de tous les fichiers PHP Var_dump(xdebug_get_code_coverage()) ; // Ajoutez à la fin de tous les fichiers PHP Ces instructions nous permettent de voir les lignes qui ont été visitées par nos tests. La suite de tests doit être modifiée jusqu'au moment où toutes les branches sont exécutées au moins une fois.

8.4 Tests de type boîte grise

8.4.1 Sécurité

La sécurité informatique est un sujet très vaste. Certains programmes scolaires y sont entièrement consacrés ce qui démontre bien la complexité et l'étendue de cette matière. Notre objectif à ce niveau n'est pas de prouver que les AW analysées sont totalements sécuritaires. L'obtention d'un niveau

de sécurité optimal des AW outrepasse les objectifs de notre mémoire. Les vérifications et tests que nous proposons de faire dans ce chapitre servent plutôt à démontrer qu'une AW atteint un certain niveau de sécurité minimal. Ce niveau de sécurité doit être atteint pour qu'une AW puisse être considérée de qualité. Notre intérêt au niveau de la sécurité des AW se limite aux données transmises entre les serveurs Web et les usagers.

Propriété à vérifier
Les AW doivent faire une utilisation minimale des boîtes de saisie de texte

Solution proposée
Une des façons d'éviter les différents problèmes que peut causer l'entrée de données mal intentionnées dans les formulaires est d'utiliser la saisie de texte seulement qu'en dernier recours. Lorsque cela est possible toujours privilégier les moyens de transmission qui offrent des choix prédéfinis aux usagers (ex : menus déroulants, boutons radio, cases à cocher, etc.).

Propriété à vérifier
Les données fournies en entrée doivent être validées avant d'être transmises au serveur Web et aux bases de donnée.

Solution proposée Des mécanismes de validation doivent être mis en place sur le serveur Web afin de contrôler l'entrée de donnée. La validation d'entrée de données fait du côté client, en utilisant javascript par exemple, n'est pas une garantie de sécurité. La validation du côté client existe pour rendre plus conviviale l'interaction du client avec l'AW. Elle ne doit pas être perçue comme étant une façon de valider l'information envoyée au serveur [42].

Au niveau de la validation des données transmises au serveur Web, un contrôle doit être exercé sur le type, la forme, la taille et les frontières de celles-ci. Une série de tests doit vérifier le bon fonctionnement de ces validations. Par exemple, nous devons prévoir des tests qui inscriront des chaînes de caractères dans les boîtes de saisie de texte où des entiers sont attendus. Voici d'autres exemples de contrôles possibles de même nature : fournir un code postal ne respectant pas le format attendu, entrer un chiffre deux fois plus grand que le maximum permis, placer une commande en ligne pour l'achat de cent téléviseurs, dépassant ainsi la frontière raisonnable attendue, etc.

Pour créer la suite de tests nous avons adopté l'idée de Tappenden [42]

d'utiliser l'outil code source libre HttPUnit. Cet outil simule le fonctionnement d'un fureteur. Par défaut HttpUnit tient compte de la validation du côté client par contre celle-ci peut être désactivée. Une fois cette validation désactivée nous pouvons ensuite lancer une série de tests qui vérifieront l'efficacité de la validation côté serveur. Nous avons programmé en utilisant HttpUnit une série de tests permettant de valider la sécurité de notre AW de vente de livres en lignes.

Propriété à vérifier
Les AW doivent empêcher le cross site scripting (XSS).

Solution proposée
L'Open Web Application Security Project (OWASP) est un organisme à but non lucratif qui travaille à l'amélioration de la sécurité des logiciels. Chaque année OWASP publie un document contenant la liste des dix plus grandes vulnérabilités de sécurité des AW. Dans le dernier document publié (2007) se trouve en tête de liste le cross site scripting (XSS). Le XSS consiste à injecter du code malicieux, habituellement du script, dans les pages Web via un forum par exemple. Les torts causés par ce genre d'attaque sont envers les clients qui accèderont ensuite ces pages Web contenant du XSS. Différentes méthodes existent pour empêcher ce genre d'attaque. En PHP, par exemple, la fonction htmlentities remplace tous les caractères qui peuvent s'écrire avec des entités en leurs entités html correspondantes. Par exemple le caractère '<' est remplacé par '&lt'.

Afin d'analyser la vulnérabilité d'une AW face au XSS nous devons, lorsque la possibilité existe, injecter un script de type JavaScript et VBScript. Ces scripts doivent être bloqués par le serveur. Pour JavaScript un test simple serait d'essayer d'inscrire le code suivant : <script language="JavaScript"> alert("test"); </script> dans un forum par exemple. Pour VBScript un test similaire serait d'essayer d'inscrire ce code au même endroit : <script language="VBScript"> MsgBox "test" </script>. Bien que ces deux exemples soient sans conséquences, ils servent à déterminer si l'injection de script est permise par le serveur Web. L'échec d'un de ces tests indiquerait une faille majeure au niveau de la sécurité. Voici un exemple de code JavaScript dangereux qui, une fois injecté dans une page Web, redirige les clients vers une page contenant un virus :
<script language="javascript">
document.location.href=http ://www.virus.ca/

</script>

Propriété à vérifier
Filtrage des caractères illégaux

Solution proposée
L'AW doit filtrer certains caractères et empêcher la transmission de ces caractères au serveur Web. Certains sont reconnus comme étant problématiques. Par exemple une AW peut décider d'interdire de transmettre dans une chaîne de caractère deux barres obliques successives. Ceci afin d'éviter une injection SQL. L'injection SQL touche les sites qui interagissent de manière non sécurisé avec une base de données. Un usager peut alors changer le but d'une requête en utilisant des variables modifiables. Par exemple supposons qu'une AW utilisant une base de données MySQL contient le code suivant : SELECT uid WHERE name= 'nom' AND password = 'mot de passe' Dans le cas où un usager entre le nom Smith' // et un mot de passe quelconque la requête devient SELECT uid WHERE name='Smith' //' AND password = 'mot de passe' En SQL tout se qui suit deux barres obliques juxtaposées devient un commentaire. Notre requête se transforme donc en SELECT uid WHERE name = 'Smith'. Dans le cas où un des usagers se nomme Smith l'utilisateur pourra se connecter au compte de Smith malgré le fait qu'il ne connaît pas son mot de passe. Nous parlons bien d'une injection ici car l'utilisateur a réussi à injecter certains caractères qui ont servi à modifier le fonctionnement de la requête.

Il existe dans certains cas des solutions préétablies pour éviter ce genre de problème. En php l'utilisation de la fonction my_sql_real_escape_string empêche les usagers de pouvoir exécuter l'injection présentée dans notre exemple. D'autres fonctions et méthodes existent pour renforcer la protection contre ces injections en php et la même chose est vraie pour d'autres scripts tels que mysql et autres.

L'ensemble des caractères potentiellements dangereux varie selon le langage de programmation, le type de base de données et le système d'exploitation du serveur Web. Les langages Web possèdent pour la plupart des fonctions pour éliminer les caractères indésirables. Par exemple, en PHP, la fonction str_replace peut être utilisée pour faire cette tâche. L'ensemble des caractères illégaux doit être affectée à une variable. Ensuite, en utilisant la fonction str_replace, nous pouvons remplacer tous les caractères transmis membres de cette liste par un espace vide. La difficulté avec cette méthode

est d'arriver à écrire un ensemble complet de tous les caractères à proscrire. C'est pour cette raison que nous recommandons de faire l'inverse c.-à-d. de ne permettre que certains caractères et de filtrer tous les autres. En PHP nous pouvons utiliser la fonction preg_match pour n'accepter que certains caractères et exclure tous les autres.

Une série de tests doit aussi être prévue pour s'assurer du bon fonctionnement du filtrage des caractères à risque. Nous suggérons, ici aussi, l'utilisation de HttpUnit, en désactivant la validation côté client, pour créer la suite de tests nécessaires.

Propriété à vérifier
Les messages d'erreur ne doivent pas dévoiler aux clients des informations pouvant compromettre la sécurité de l'AW.

Solution proposée
Lorsqu'une erreur survient un avis générique doit être envoyé au client pour l'informer de la cause de celle-ci. Aucune information sur la structure et le contenu du serveur Web ne doit être transmisse au client. L'AW, suite à une erreur, ne doit pas retourner par exemple le chemin utilisé pour se rendre à l'erreur (stack trace). Ce genre d'information ne devrait être accessible qu'en période de développement.

Il existe différentes façons, selon les langages utilisés, de limiter l'information transmise suite à une erreur. En PHP le fait de mettre le paramètre display_errors=off dans le fichier php.ini empêche toute information interne d'être transmise par mégarde dans le cas d'une erreur. Nous devons créer une suite de tests qui provoque des erreurs et ensuite observé le degré de sécurité des messages d'erreur renvoyés aux clients. Dans les pages où il faut s'authentifier nous devons faire au minimum quatre tests. Un nom d'usager existant avec un faux mot de passe et sans mot de passe. Un nom d'usager fictif avec un mot de passe quelconque et sans mot de passe. Les messages d'erreurs engendrés par ces quatre exemples ne doivent pas informer l'usager sur l'existence ou non du nom d'usager transmis. La transmission d'information partielle ou erronée au serveur Web via une boîte de recherche, un formulaire ou autres doit aussi être fait pour observer les messages d'erreur ainsi créés.

Propriété à vérifier
L'échange d'information entre le serveur Web et ses clients doit être sécuritaire.

Solution proposée
Dans un formulaire la méthode de transmission d'information au serveur doit être définie. Les deux méthodes les plus utilisées sont get et post. La méthode get transmet l'information en clair dans la barre d'adresse. La méthode post masque les données mais n'encrypte pas l'information. Entre les deux méthodes de transmission nous devons toujours choisir la méthode post lorsque de l'information personnelle est requise par un site. La méthode post doit aussi être celle utilisée lorsque des données sont transmises par le client au serveur Web.

La méthode get est risquée car l'information transmisse est visible dans la barre d'adresse. De plus, une fois connecté sur un site, un client peut enregistrer ce lien dans ces favoris. Ce lien contiendra alors le nom d'usager et le mot de passe de l'usager. Toute personne ayant accès à cet ordinateur peut alors ouvrir une session en activant le lien situé dans les favoris. Le bouton arrière dans un navigateur peut aussi être utilisé afin d'obtenir les mêmes résultats. La méthode get ne doit être utilisée que dans les cas où aucune information personnelle n'est requise. Par exemple, cette méthode est souvent utilisée dans les boîtes de recherche. Une fois l'information trouvée, il peut être utile d'enregistrer le lien dans nos favoris par exemple.

Pour toute transaction monétaire le protocole SSL doit être utilisé. Ce même protocole doit aussi être utilisé dans toute situation où des informations confidentiels sont transmises entre serveurs Web et clients.

Chapitre 9

Conclusion

Ce mémoire propose une méthodologie pour valider la qualité des applications Web (AW). Elle est profitable pour la pluralité des AW. Nous favorisons les méthodes de validation automatiques et semi-automatiques ainsi que les outils code source libre et gratuiciels. Nous avons étudié et analyser la validation de la qualité des AW avant, pendant et après leur création.

La création d'AW est un domaine récent où il n'existe pas de tradition d'assurance de la qualité comme celle qu'on retrouve dans la programmation traditionnelle de logiciels. Afin de saisir les raisons pouvant expliquer, en partie, cette disparité au niveau du contrôle de la qualité, nous avons analysé les différences qui existent entre ces deux types de programmation (documentation disponible, fréquence des modifications, architecture matériel utilisée, etc.). De façon générale, il est plus difficile de valider la qualité des AW versus les applications traditionnelles. Ceci explique en partie l'intérêt moindre observé envers l'assurance qualité des AW.

Après avoir présenté certains critères de qualité et certains types de tests utilisés par d'autres méthodologies de contrôle de la qualité des AW nous avons étudié quelques modèles de conception d'AW. Nous avons créé une AW de vente de livres en ligne avec deux de ces modèles soit IBM Rational Application Developper et WebML. Cela nous a permis de mieux se familiariser avec ces outils. Nous avons examiné, entres autres, les validations automatisées offertes par ces outils. Le modèle de conception WebML, utilisé avec le logiciel intégré WebRatio, est particulièrement intéressant. WebML rend la création d'AW de qualité, de moyenne et grande taille, accessible aux apprentis webmestres. Les modèles de conception d'AW tel que WebML représentent l'avenir du domaine de création d'AW. Au moment de concevoir

notre méthodologie de contrôle de la qualité, nous avons retenu certaines des idées provenant des modèles de conception d'AW que nous avons analysés.

Une revue de la littérature portant sur les méthodologies de validation de la qualité des AW a aussi été effectuée. Cet état de l'art nous a permis d'identifier certaines des caractéristiques entrant dans la définition du niveau de qualité des AW. Nous avons aussi noté qu'il existe peu de méthode de validation de la qualité des AW.

Nous avons étudié trois outils permettant l'écriture et la vérification automatisée de spécifications axées sur le contrôle de la qualité des AW. À l'aide de ces outils nous avons écrit une spécification avec chacun de ces outils afin de valider la qualité de notre AW de vente de livres en ligne. L'écriture de spécification, à l'aide de ces outils, requière beaucoup d'effort et de temps. De plus, ils ne permettent de vérifier qu'une partie des critères de qualité d'une AW. Malgré les limites de ces outils, certaines des approches utilisées par ces outils nous ont inspiré dans l'écriture de notre méthodologie.

Nous avons proposé une méthodologie pour valider la qualité des AW dans laquelle les types de tests sont séparés en trois catégories : tests de type boîte noire, tests de type boîte transparente et tests de type boîte grise. Le niveau de connaissance d'une AW détermine dans quelle(s) catégorie(s) de test une personne peut s'impliquer. Les tests de type boîte noire peuvent être faits par quiconque. Ceux de type boîte blanche requièrent d'avoir accès au code interne de l'AW. Finalement les tests de type boîte grise doivent être faits par des gens qui connaissent le fonctionnement interne de l'AW. Cette façon de séparer les tests à faire permet de faciliter grandement la distribution des tâches. En plus des outils proposés pour valider la qualité des AW nous avons aussi programmé quatre programmes qui permettent d'automatiser la vérification de certains des tests de validation proposés dans notre méthodologie (navigation, pages isolées). Nous avons utilisé notre méthodologie pour valider la qualité de notre AW de vente de livre.

Bien que, nous ayons conçu une méthodologie efficace et simple à utiliser qui permet de valider efficacement la qualité des AW, d'autres améliorations sont possibles. La découverte de d'autres outils de conception d'AW, d'outils d'écriture de spécification et de travaux connexes pourrait occasionner l'addition d'un ensemble de tests à notre méthodologie proposée. Certains tests manuels pourraient être automatisés. Des tests automatiques pourraient éventuellement être regroupés.

Annexe A

Spécification WAVE
Vente de livres en ligne

Sur la figure A.1 nous avons une représentation graphique de notre AW de vente de livres en ligne. Le texte en gras représente les boutons, le texte en gras et souligné les liens et finalement le texte en italique les données non-incluses dans la spécification WAVE. Les entrées qui ne sont pas délimitées (Input-bounded) ne sont pas définies dans la spécification WAVE.

Spécification WAVE de notre AW de vente de livres en ligne

```
Database:

books(oid, title, author, year, price);
administrator(username, password)

State:

error(errmsg);
admin();
user();
adminchoice()

Input:

clickbutton1(buttonname);
clicklink1(linkname);
clicklink2(linkname);
clicklink3(linkname);
```

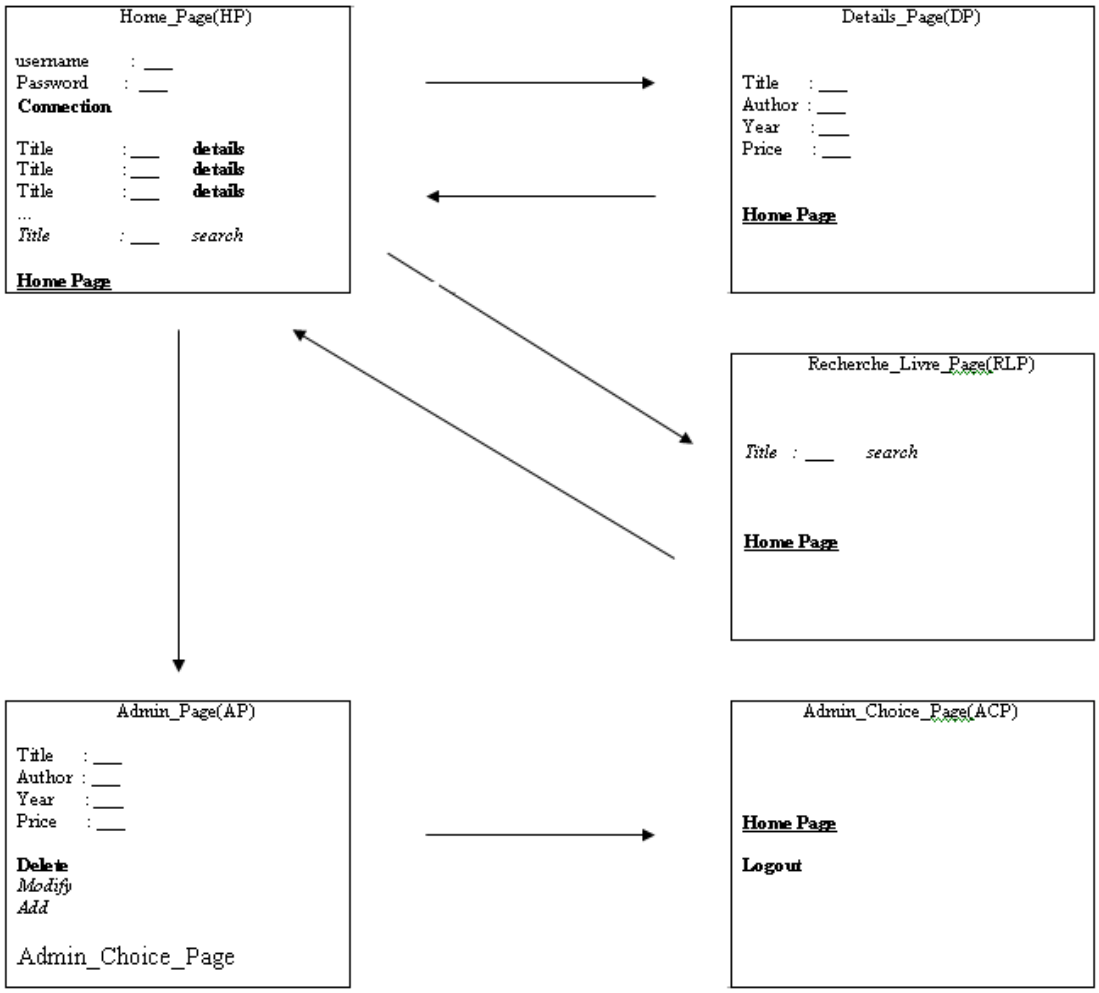

FIG. A.1 – AW de vente de livres en ligne

```
clicklink4(linkname);
clicklink5(linkname);
deletebook(title, author, year, price);
admininput(name,pwd)

Input Constant:
username, password

Web page Schema Set:
HP;DP;AP;ACP;EP

Home page: HP

Error page: EP

End of Session Rules:
not administrator(username, password) : = exists n, p admininput(n,p)
and not exists p administrator(username, p)
and clickbutton("connection")

Page: Home_Page(HP)

Input: username, password, clickbutton1(x), clicklink1(x)
Input Rules:
Options@clickbutton1(x) : = x = "connection";
Options@clickbutton1(x) : = x = "details";
Options@clicklink1(x)   : = x="homepage"

State Rules:
error("bad_user") : = not administrator(username, password)
and clickbutton1("connection");
admin() : = administrator(username, password)
and clickbutton1("connection")

Target Webpages: HP, DP, AP
Target Rules:
HP  : = clicklink1("homepage");
DP  : = clickbutton1(x) : = x = "details" and books(oid, title, author, year, price);
AP  : = administrator(username, password) and clickbutton1("connection")

Page: Details_Page(DP)

Input: clicklink2(x)
Input Rules: Options@clicklink2(x) : = x = "homepage"
```

```
Target Webpages: HP
Target Rules:
HP : = clicklink2("homepage")


Page: Admin_Page(AP)

Input: clicklink3(x)
Input Rules: Options@clicklink3(x) : = x="AdminChoicePage";
Options@deletebook(title, author, year, price) : =
exists t, a, y, p (deletebook(t, a, y, p)
and books(oid, t, a, y, p));

State Rules:
adminchoice() : = clicklink3("AdminChoicePage")

Target Webpages: ACP
Target Rules:
ACP : = clicklink3("AdminChoicePage")

Page: Admin_Choice_Page(ACP)

Input: clicklink4(x)
Input Rules: Options@clicklink4(x) : = x="homepage";
Options@clicklink4(x) : = x="Logout"

State Rules:
user() : = clicklink4("Logout")

Target Webpages: HP
Target Rules:
HP : = clicklink4("homepage")

Page: Error_Message_Page(EP)

Posted Data: error(text)
Input: clicklink5(x)
Input Rules: Options@clicklink5(x) : = x="homepage"

Target Webpages: HP
Target Rules:
HP : = clicklink5("homepage")
```

Annexe B

Modèles de navigation

Nous avons écrit trois programmes de vérification de modèle :
- modèle arbre (AppWebNavigationArbre.java)
- modèle hiérarchique(AppWebNavigationHierarchique.java)
- modèle à connections complètes (AppWebNavigationFullConnectivity.java)

```
/**
 * @(#)Noeud.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/11/26
 */

public class Noeud {
 private String from;
 private String to;

 public Noeud() {
  String from;
  String to;
 }

 public Noeud(String f, String t){
 from = f;
 to = t;
 }

 public String retournerFrom(){
  return from;
 }
 public String retournerTo(){
```

```
  return to;
 }
}

/**
 * @(#)ListeNoeuds.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/11/26
 */

import java.util.Vector;
public class ListeNoeuds{
//liste des noeuds (pages Web) du graphe (application Web)
 private Vector list;

 public ListeNoeuds(){
  list = new Vector();
 }

 public String retournerFrom(Integer i){
  return (((Noeud)list.elementAt(i)).retournerFrom());
 }

 public String retournerTo(Integer i){
  return (((Noeud)list.elementAt(i)).retournerTo());
 }

 public Object elementAt(Integer i){
  return list.elementAt(i);
 }

 public int size(){
  return list.size();
 }

 public void supprimerNoeud(Integer i){
  list.removeElementAt(i);
 }

 public boolean addNoeud(Noeud o){
  list.addElement(o);
  return true;
 }
```

```
 public String toString(){
  String temp="";
  int i;
  for(i=0; i<list.size();i++)
  {
   temp+=list.elementAt(i).toString();
   temp+="\n";
  }
  return temp;
  }
}


/**
 * @(#)AppWebNavigationArbre.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/12/16
 *
 *Ce programme utilise les deux programmes suivants :
 *Noeud.java et ListeNoeuds.java (Annexe C)
 *
 *Les entrées sont fournies dans le code du présent programme
 *
 *Entrée:   Nom(id) de toutes les pages Web de l'application
 *          Nom(id) de la ou des page(s) racine(s) (ex:index.html)
 *          Nom(from et to) des pages qui sont reliées entre elles
 *
 *Sortie:   Un affichage à l'écran nous indique si l'application Web analysée
 *          utilise une navigation de type arbre
 *
 *  Navigation en forme d'arbre:
 *  Tous les noeuds doivent être atteignables
 *  Le graphe doit être unidirectionnel
 *  Les noeuds peuvent avoir au maximum un seul noeud parent
 */

import java.awt.*;
import java.util.*;
public class AppWebNavigationArbre{
    public static void main(String args[]){
    //Ensemble des noeuds(pages Web) reliés entre eux
    ListeNoeuds ln = new ListeNoeuds();
    //Ensemble de tous les noeuds du graphe
    Vector tous = new Vector();
```

```
//Ensemble de la ou des racine(s) du graphe
Vector racine = new Vector();
//Ensemble des noeuds atteignables par au moins un autre noeud
Vector atteignables = new Vector();
//Ensemble des noeuds à vérifier
Vector a_verifier = new Vector();
//Ensemble des noeuds déjà vérifiés
Vector a_verifier_supprimer = new Vector();
//Nombre d'élément(s) dans la variable tous
Integer tous_size;
//Nombre d'élément(s) dans la variable racine
Integer racine_size;
//Nombre d'élément(s) dans la variable atteignables
Integer atteignables_size;
Integer compteur;
Integer compteur2;
String temporaire;
String from;
String to;
boolean unidirectionnel=true;
boolean multiplesParents = false;

//***Début de l'entrée de données***
//Entrez ici le nom(//PAGE/@id) de toutes les pages Web de votre application
//Exemple d'une application Web contenant 4 pages Web
tous.add("a");
tous.add("b");
tous.add("c");
tous.add("d");
tous.add("e");
tous.add("f");

//Entrez ici le nom(//PAGE/@id) de la ou des racine(s)
//Dans l'exemple suivant il n'y a qu'une seule racine soit
//la page a(//PAGE/@id="a")
 racine.add("a");

//Entrez ici les noeuds qui ont au moins un lien vers un autre noeud
//Dans l'exemple suivant a(//PAGE/@id="a") pointe vers b(//PAGE/@id="b"),
//b(//PAGE/@id="b") pointe vers a(//PAGE/@id="a") et
//a(//PAGE/@id="a") pointe vers c(//PAGE/@id="c") etc.
Noeud n1 = new Noeud("a","b");
Noeud n2 = new Noeud("a","c");
Noeud n3 = new Noeud("a","d");
Noeud n4 = new Noeud("d","e");
```

```
Noeud n5 = new Noeud("e","b");
Noeud n6 = new Noeud("b","e");

ln.addNoeud(n1);
ln.addNoeud(n2);
ln.addNoeud(n3);
ln.addNoeud(n4);
ln.addNoeud(n5);
ln.addNoeud(n6);
//***Fin de l'entrée de données***

compteur = 0;
racine_size = racine.size();
//Le contenu du vecteur racine est copié dans les vecteurs
//a_verifier et atteignables
while(compteur != racine_size){
 a_verifier.add(racine.elementAt(compteur));
 atteignables.add(racine.elementAt(compteur));
 compteur++;
}

compteur = 0;
//Création de la liste des pages atteignables à partir de la ou des racine(s)
while((a_verifier.size()!=0) && (atteignables.size()!=tous.size())){
 temporaire = ((String)a_verifier.firstElement());
 compteur = 0;
while(compteur != ln.size()){
//Vérification des noeuds de la liste des noeuds que contient la
//variable ln qui ont un attribut From équivalent à la première
//valeur du vecteur a_verifier
 if(((Noeud)ln.elementAt(compteur)).retournerFrom().equalsIgnoreCase(temporaire)){
 //Les pages atteignables à partir du vecteur a_verifier qui ne font pas déjà
 //parti du vecteur atteignables sont ajoutées à celui-ci
 if(!atteignables.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
  atteignables.add(((Noeud)ln.elementAt(compteur)).retournerTo());
  //Ajout des pages à vérifier qui n'ont pas déjà été vérifiées dans le
  //vecteur a_verifier.
 if((!a_verifier.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
 //On s'assure que la page dans le vecteur a_verifier n'a pas déjà été vérifiée
 //Ceci est fait pour éviter d'entrer dans une boucle infinie lorsque
 //deux pages pointent l'une vers l'autre.
 //Le vecteur a_verifier_supprimer contient l'ensemble des pages déjà vérifiées
 && (!a_verifier_supprimer.contains(((Noeud)ln.elementAt(compteur)).retournerTo())))
  a_verifier.add(((Noeud)ln.elementAt(compteur)).retournerTo());
  //Suppression des pages vérifiées
```

```
      if(a_verifier.removeElement(temporaire))
       a_verifier_supprimer.add(temporaire);
      }
        compteur++;
      }
       a_verifier.remove(temporaire);
     }

    compteur=0;
    //Le vecteur tous contient au départ la liste de toutes
    //les pages Web de l'application Web
    while(compteur!=atteignables.size()){
        temporaire = ((String)atteignables.elementAt(compteur));
        if(tous.contains(temporaire))
        //Suppression des pages atteignables du vecteur tous
        tous.remove(temporaire);
        compteur++;
    }

//Nous vérifions si le graphe est unidirectionnel ou non
//Ex: si A pointe vers B, B ne doit pas pointer vers A
compteur=0;
while(compteur!=ln.size()){
 from = ((Noeud)ln.elementAt(compteur)).retournerFrom();
 to = ((Noeud)ln.elementAt(compteur)).retournerTo();
 compteur++;
 compteur2=0;
  while(compteur2!=ln.size()){
   if((((Noeud)ln.elementAt(compteur2)).retournerFrom().equalsIgnoreCase(to))
   && (((Noeud)ln.elementAt(compteur2)).retournerTo().equalsIgnoreCase(from)))
    unidirectionnel=false;
    compteur2++;
 }
}

//Nous vérifions qu'aucun noeud ne possède plus d'un parent
//Dans une application au format arbre aucun noeud ne doit posséder plus
//d'un parent
a_verifier.clear();
compteur=0;
while(compteur!=ln.size()){
 to = ((Noeud)ln.elementAt(compteur)).retournerTo();
 if(a_verifier.contains(to))
 multiplesParents=true;
 a_verifier.add(to);
```

```
 compteur++;
}

    //tous.size()==0 lorsque toutes les pages sont atteignables
    if(unidirectionnel==false || tous.size()!=0 || multiplesParents == true){
        System.out.println("Cette application Web n'est pas en forme d'arbre.");
    if(unidirectionnel==false)
        System.out.println("Elle n'est pas unidirectionnel.");
    if(tous.size()!=0)
        System.out.println("Une ou plusieurs pages est inatteignable(s).");
    if(multiplesParents==true)
        System.out.println("Au moins une page possède plus d'un parent.");
    }
    else{
        System.out.println("Cette application Web utilise une navigation ");
        System.out.print("en forme d'arbre.");
    }
  }
}


/**
 * @(#)AppWebNavigationHierarchique.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/12/17
 *
 *Les entrées sont fournies dans le code du présent programme
 *
 *Entrée:   Nom(id) de toutes les pages Web de l'application
 *          Nom(id) de la ou des page(s) racine(s) (ex:index.html)
 *          Nom(from et to) des pages qui sont reliées entre elles
 *
 *Sortie:   Un affichage à l'écran nous indique si l'application Web analysée
 *          utilise une navigation de type hierarchique
 *
 *Navigation hiérarchique:
 *  Tous les noeuds doivent être atteignables
 *  Le graphe doit être unidirectionnel
 *  Au moins un noeud doit avoir plus d'un noeud parent
 */

import java.awt.*;
import java.util.*;
public class AppWebNavigationHierarchique{
```

```
public static void main(String args[]){
//Ensemble des noeuds(pages Web) reliés entre eux
ListeNoeuds ln = new ListeNoeuds();
//Ensemble de tous les noeuds du graphe
Vector tous = new Vector();
//Ensemble de la ou des racine(s) du graphe
Vector racine = new Vector();
//Ensemble des noeuds atteignables par au moins un autre noeud
Vector atteignables = new Vector();
//Ensemble des noeuds à vérifier
Vector a_verifier = new Vector();
//Ensemble des noeuds déjà vérifiés
Vector a_verifier_supprimer = new Vector();
//Nombre d'élément(s) dans la variable tous
Integer tous_size;
//Nombre d'élément(s) dans la variable racine
Integer racine_size;
//Nombre d'élément(s) dans la variable atteignables
Integer atteignables_size;
Integer compteur;
Integer compteur2;
String temporaire;
String from;
String to;
boolean unidirectionnel=true;
boolean multiplesParents = false;

//***Début de l'entrée de données***
//Entrez ici le nom(//PAGE/@id) de toutes les pages Web de votre application
//Exemple d'une application Web contenant 4 pages Web
tous.add("a");
tous.add("b");
tous.add("c");
tous.add("d");
tous.add("e");
tous.add("f");

//Entrez ici le nom(//PAGE/@id) de la ou des racine(s)
//Dans l'exemple suivant il n'y a qu'une seule racine
//soit la page a(//PAGE/@id="a")
 racine.add("a");

//Entrez ici les noeuds qui ont au moins un lien vers un autre noeud
//Dans l'exemple suivant a(//PAGE/@id="a") pointe vers b(//PAGE/@id="b"),
//b(//PAGE/@id="b") pointe vers a(//PAGE/@id="a") et
```

```
    //a(//PAGE/@id="a") pointe vers c(//PAGE/@id="c") etc.
    Noeud n1 = new Noeud("a","b");
    Noeud n2 = new Noeud("a","c");
    Noeud n3 = new Noeud("a","d");
    Noeud n4 = new Noeud("d","e");
    Noeud n5 = new Noeud("e","d");

    ln.addNoeud(n1);
    ln.addNoeud(n2);
    ln.addNoeud(n3);
    ln.addNoeud(n4);
    ln.addNoeud(n5);
    //***Fin de l'entrée de données***

    compteur = 0;
    racine_size = racine.size();
    //Le contenu du vecteur racine est copié dans les vecteurs
    //a_verifier et atteignables
    while(compteur != racine_size){
        a_verifier.add(racine.elementAt(compteur));
      atteignables.add(racine.elementAt(compteur));
      compteur++;
  }

compteur = 0;
//Création de la liste des pages atteignables à partir de la ou des racine(s)
while((a_verifier.size()!=0) && (atteignables.size()!=tous.size())){
 temporaire = ((String)a_verifier.firstElement());
 compteur = 0;
 while(compteur != ln.size()){
 //Vérification des noeuds de la liste des noeuds que contient la
 //variable ln qui ont un attribut From équivalent à la première
 //valeur du vecteur a_verifier
  if(((Noeud)ln.elementAt(compteur)).retournerFrom().equalsIgnoreCase(temporaire)){
  //Les pages atteignables à partir du vecteur a_verifier qui ne font pas déjà
  //parti du vecteur atteignables sont ajoutées à celui-ci
   if(!atteignables.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
   atteignables.add(((Noeud)ln.elementAt(compteur)).retournerTo());
   //Ajout des pages à vérifier qui n'ont pas déjà été vérifiées dans le
   //vecteur a_verifier.
   if((!a_verifier.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
   //On s'assure que la page dans le vecteur a_verifier n'a pas déjà été vérifiée
   //Ceci est fait pour éviter d'entrer dans une boucle infinie lorsque
   //deux pages pointent l'une vers l'autre.
   //Le vecteur a_verifier_supprimer contient l'ensemble des pages déjà vérifiées
```

```
   && (!a_verifier_supprimer.contains(((Noeud)ln.elementAt(compteur)).retournerTo())))
   a_verifier.add(((Noeud)ln.elementAt(compteur)).retournerTo());
   //Suppression des pages vérifiées
   if(a_verifier.removeElement(temporaire))
   a_verifier_supprimer.add(temporaire);
   }
   compteur++;
 }
a_verifier.remove(temporaire);
}

compteur=0;
//Le vecteur tous contient au départ la liste de toutes les pages Web de l'application Web
while(compteur!=atteignables.size()){
 temporaire = ((String)atteignables.elementAt(compteur));
 if(tous.contains(temporaire))
 //Suppression des pages atteignables du vecteur tous
 tous.remove(temporaire);
 compteur++;
}

//Nous vérifions si le graphe est unidirectionnel ou non
//Ex: si A pointe vers B, B ne doit pas pointer vers A
compteur=0;
while(compteur!=ln.size()){
 from = ((Noeud)ln.elementAt(compteur)).retournerFrom();
 to = ((Noeud)ln.elementAt(compteur)).retournerTo();
 compteur++;
 compteur2=0;
 while(compteur2!=ln.size()){
  if((((Noeud)ln.elementAt(compteur2)).retournerFrom().equalsIgnoreCase(to))
  && (((Noeud)ln.elementAt(compteur2)).retournerTo().equalsIgnoreCase(from)))
unidirectionnel=false;
compteur2++;
 }
}

//Nous vérifions si au moins un noeud a plus d'un parent
a_verifier.clear();
compteur=0;
while(compteur!=ln.size()){
 to = ((Noeud)ln.elementAt(compteur)).retournerTo();
 if(a_verifier.contains(to))
 multiplesParents=true;
 a_verifier.add(to);
```

```
 compteur++;
}

//tous.size()==0 lorsque toutes les pages sont atteignables
if(unidirectionnel==false || tous.size()!=0 || multiplesParents == false){
 System.out.println("Cette application Web n'est pas en forme hiérarchique.");
 if(unidirectionnel==false)
 System.out.println("Elle n'est pas unidirectionnel.");
 if(tous.size()!=0)
 System.out.println("Une ou plusieurs pages est inatteignable(s).");
 if(multiplesParents==false)
 System.out.println("Aucune page ne possède plus d'un parent.");
}
else{
 System.out.println("Cette application Web utilise une navigation ");
 System.out.print("en forme hiérarchique.");
    }
  }
}


/**
 * @(#)AppWebNavigationFullConnectivity.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/12/18
 *
 *Les entrées sont fournies dans le code du présent programme
 *
 *Entrée:   Nom(id) de toutes les pages Web de l'application
 *          Nom(id) de la ou des page(s) racine(s) (ex:index.html)
 *          Nom(from et to) des pages qui sont reliées entre elles
 *
 *Sortie:   Un affichage à l'écran nous indique si l'application Web analysée
 *          utilise une navigation de type hierarchique
 *
 *Navigation full connectivity:
 *  Tous les noeuds doivent être atteignables
 *  Tous les neouds doivent être connecter directement à tous les autres noeuds
 */

import java.awt.*;
import java.util.*;
public class AppWebNavigationFullConnectivity{
    public static void main(String args[]){
```

```
//Ensemble des noeuds(pages Web) reliés entre eux
ListeNoeuds ln = new ListeNoeuds();
//Ensemble de tous les noeuds du graphe
Vector tous = new Vector();
//Ensemble de la ou des racine(s) du graphe
Vector racine = new Vector();
//Ensemble des noeuds atteignables par au moins un autre noeud
Vector atteignables = new Vector();
//Ensemble des noeuds à vérifier
Vector a_verifier = new Vector();
//Ensemble des noeuds déjà vérifiés
Vector a_verifier_supprimer = new Vector();
//Ensemble de toutes les pages (noeuds du graphe)
Vector pages = new Vector();
//Ensemble des pages directement atteignables à partir d'une page
Integer pagesVoisines;
//Nombre d'élément(s) dans la variable tous
Integer tous_size;
//Nombre d'élément(s) dans la variable racine
Integer racine_size;
//Nombre d'élément(s) dans la variable atteignables
Integer atteignables_size;
Integer compteur;
Integer compteur2;
String temporaire;
String from;
String to;

//***Début de l'entrée de données***
//Entrez ici le nom(//PAGE/@id) de toutes les pages Web de votre application
//Exemple d'une application Web contenant 4 pages Web
tous.add("a");
tous.add("b");
tous.add("c");

//Entrez ici le nom(//PAGE/@id) de la ou des racine(s)
//Dans l'exemple suivant il n'y a qu'une seule racine soit la page a(//PAGE/@id="a")
 racine.add("a");

//Entrez ici les noeuds qui ont au moins un lien vers un autre noeud
//Dans l'exemple suivant a(//PAGE/@id="a") pointe vers b(//PAGE/@id="b"),
//b(//PAGE/@id="b") pointe vers a(//PAGE/@id="a") et
//a(//PAGE/@id="a") pointe vers c(//PAGE/@id="c") etc.
Noeud n1 = new Noeud("a","b");
Noeud n2 = new Noeud("a","c");
```

```
    Noeud n3 = new Noeud("b","a");
    Noeud n4 = new Noeud("b","c");
    Noeud n5 = new Noeud("c","a");
    Noeud n6 = new Noeud("c","b");

    ln.addNoeud(n1);
    ln.addNoeud(n2);
    ln.addNoeud(n3);
    ln.addNoeud(n4);
    ln.addNoeud(n5);
    ln.addNoeud(n6);
    //***Fin de l'entrée de données***

    compteur = 0;
    racine_size = racine.size();
    //Le contenu du vecteur racine est copié dans les vecteurs
    //a_verifier et atteignables
    while(compteur != racine_size){
        a_verifier.add(racine.elementAt(compteur));
atteignables.add(racine.elementAt(compteur));
compteur++;
}

compteur = 0;
//Création de la liste des pages atteignables à partir de la ou des racine(s)
while((a_verifier.size()!=0) && (atteignables.size()!=tous.size())){
 temporaire = ((String)a_verifier.firstElement());
 compteur = 0;
 while(compteur != ln.size()){
  //Vérification des noeuds de la liste des noeuds que contient la
  //variable ln qui ont un attribut From équivalent à la première
  //valeur du vecteur a_verifier
  if(((Noeud)ln.elementAt(compteur)).retournerFrom().equalsIgnoreCase(temporaire)){
   //Les pages atteignables à partir du vecteur a_verifier qui ne font pas déjà
   //parti du vecteur atteignables sont ajoutées à celui-ci
   if(!atteignables.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
   atteignables.add(((Noeud)ln.elementAt(compteur)).retournerTo());
   //Ajout des pages à vérifier qui n'ont pas déjà été vérifiées dans le
   //vecteur a_verifier.
   if((!a_verifier.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
   //On s'assure que la page dans le vecteur a_verifier n'a pas déjà été vérifiée
   //Ceci est fait pour éviter d'entrer dans une boucle infinie lorsque
   //deux pages pointent l'une vers l'autre.
   //Le vecteur a_verifier_supprimer contient l'ensemble des pages déjà vérifiées
   && (!a_verifier_supprimer.contains(((Noeud)ln.elementAt(compteur)).retournerTo())))
```

```
   a_verifier.add(((Noeud)ln.elementAt(compteur)).retournerTo());
   //Suppression des pages vérifiées
   if(a_verifier.removeElement(temporaire))
   a_verifier_supprimer.add(temporaire);
   }
   compteur++;
}
a_verifier.remove(temporaire);
}

compteur = 0;
//Le vecteur pages va contenir l'ensemble de toutes les pages
//Le contenu du vecteur tous est copié dans le vecteur pages
while(compteur!=tous.size()){
 temporaire = ((String)tous.elementAt(compteur));
 pages.add(temporaire);
 compteur++;
}

compteur=0;
//Le vecteur tous contient au départ la liste de toutes les pages Web de l'application Web
while(compteur!=atteignables.size()){
 temporaire = ((String)atteignables.elementAt(compteur));
 if(tous.contains(temporaire))
 //Suppression des pages atteignables du vecteur tous
 tous.remove(temporaire);
 compteur++;
}

//Nous vérifions si toutes les pages ont un accès direct
//à toutes les autres pages dans l'application Web
compteur=0;
boolean fullConnectivity = true;
while(compteur!=pages.size()){
 temporaire = ((String)pages.elementAt(compteur));
 compteur++;
 compteur2=0;
 pagesVoisines=0;
 while(compteur2!=ln.size()){
  if((((Noeud)ln.elementAt(compteur2)).retournerFrom().equalsIgnoreCase(temporaire)))
  pagesVoisines++;
  compteur2++;
 }
 if (pagesVoisines != pages.size()-1)
 fullConnectivity = false;
```

```
    }
    System.out.println(fullConnectivity);

    //tous.size()==0 lorsque toutes les pages sont atteignables
    if(tous.size()!=0 || fullConnectivity == false){
        System.out.println("Cette application Web n'est pas en forme full connectivity.");
    if(fullConnectivity==false){
        System.out.print("Il existe au moins un noeud à partir duquel nous ");
        System.out.println("ne pouvons pas rejoindre tous les autres noeuds.");
    }
    if(tous.size()!=0)
        System.out.println("Une ou plusieurs pages est inatteignable(s).");
    }
    else{
        System.out.println("Cette application Web utilise une navigation ");
        System.out.print("en forme hiérarchique.");
    }
  }
}
```

Annexe C

Pages isolées

Pour automatiser la vérification à savoir si une page web isolée existe ou non, nous avons créé le programme PagesInatteignables.java.

```
/**
 * @(#)PagesInatteignables.java
 *
 * @Auteur Eric Lafleur
 * @version 1.00 2006/11/26
 *
 *Ce programme utilise les deux programmes suivants :
 *Noeud.java et ListeNoeuds.java (Annexe C)
 *
 *Les entrées sont fournies dans le code du présent programme
 *
 *Entrée:   Nom(id) de toutes les pages Web de l'application
 *              Nom(id) de la ou des page(s) racine(s) (ex:index.html)
 *              Nom(from et to) des pages qui sont reliées entre elles
 *
 *Sortie:   Liste de la ou des page(s) qui sont inatteignable(s) à
 *             partir de la ou des page(s) racine(s)
 */

import java.awt.*;
import java.util.*;
public class PagesInatteignables{
 public static void main(String args[]){
 //Ensemble des noeuds(pages Web) reliés entre eux
 ListeNoeuds ln = new ListeNoeuds();
 //Ensemble de tous les noeuds du graphe
```

```
Vector tous = new Vector();
//Ensemble de la ou des racine(s) du graphe
Vector racine = new Vector();
//Ensemble des noeuds atteignables par au moins un autre noeud
Vector atteignables = new Vector();
//Ensemble des noeuds à vérifier
Vector a_verifier = new Vector();
//Ensemble des noeuds déjà vérifiés
Vector a_verifier_supprimer = new Vector();
String temporaire;
Integer compteur;
//Nombre d'élément(s) dans la variable tous
Integer tous_size;
//Nombre d'élément(s) dans la variable racine
Integer racine_size;
//Nombre d'élément(s) dans la variable atteignables
Integer atteignables_size;

//***Début de l'entrée de données***
//Entrez ici le nom(//PAGE/@id) de toutes les pages Web de votre application
//Exemple d'une application Web contenant 4 pages Web
tous.add("a");
tous.add("b");
tous.add("c");
tous.add("d");

//Entrez ici le nom(//PAGE/@id) de la ou des racine(s)
//Dans l'exemple suivant il n'y a qu'une seule racine soit la page a(//PAGE/@id="a")
racine.add("a");

//Entrez ici les noeuds qui ont au moins un lien vers un autre noeud
//Dans l'exemple suivant a(//PAGE/@id="a") pointe vers b(//PAGE/@id="b"),
//b(//PAGE/@id="b") pointe vers a(//PAGE/@id="a") et
//a(//PAGE/@id="a") pointe vers c(//PAGE/@id="c")
Noeud n1 = new Noeud("a","b");
Noeud n2 = new Noeud("b","a");
Noeud n3 = new Noeud("a","c");
ln.addNoeud(n1);
ln.addNoeud(n2);
ln.addNoeud(n3);
//***Fin de l'entrée de données***

compteur = 0;
racine_size = racine.size();
//Le contenu du vecteur racine est copié dans les vecteurs
```

```
 //a_verifier et atteignables
 while(compteur != racine_size){
  a_verifier.add(racine.elementAt(compteur));
atteignables.add(racine.elementAt(compteur));
compteur++;
 }

 compteur = 0;
 //Création de la liste des pages atteignables à partir de la ou des racine(s)
 while((a_verifier.size()!=0) && (atteignables.size()!=tous.size())){
  temporaire = ((String)a_verifier.firstElement());
compteur = 0;
  while(compteur != ln.size()){
   //Vérification des noeuds de la liste des noeuds que contient la
   //variable ln qui ont un attribut From équivalent à la première
   //valeur du vecteur a_verifier
   if(((Noeud)ln.elementAt(compteur)).retournerFrom().equalsIgnoreCase(temporaire)){
    //Les pages atteignables à partir du vecteur a_verifier qui ne font pas déjà
    //parti du vecteur atteignables sont ajoutées à celui-ci
    if(!atteignables.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
    atteignables.add(((Noeud)ln.elementAt(compteur)).retournerTo());
    //Ajout des pages à vérifier qui n'ont pas déjà été vérifiées dans le
    //vecteur a_verifier.
    if((!a_verifier.contains(((Noeud)ln.elementAt(compteur)).retournerTo()))
    //On s'assure que la page dans le vecteur a_verifier n'a pas déjà été vérifiée
    //Ceci est fait pour éviter d'entrer dans une boucle infinie lorsque
    //deux pages pointent l'une vers l'autre.
    //Le vecteur a_verifier_supprimer contient l'ensemble des pages déjà vérifiées
    && (!a_verifier_supprimer.contains(((Noeud)ln.elementAt(compteur)).retournerTo())))
    a_verifier.add(((Noeud)ln.elementAt(compteur)).retournerTo());
    //Suppression des pages vérifiées
    if(a_verifier.removeElement(temporaire))
    a_verifier_supprimer.add(temporaire);
   }
  compteur++;
  }
  a_verifier.remove(temporaire);
 }

 compteur=0;
 //Le vecteur tous contient au départ la liste de toutes les pages Web de l'AW
 while(compteur!=atteignables.size()){
  temporaire = ((String)atteignables.elementAt(compteur));
  if(tous.contains(temporaire))
  //Suppression des pages atteignables du vecteur tous
```

```
  tous.remove(temporaire);
  compteur++;
 }

 if(tous.size()!=0)
 System.out.println("Page(s) inatteignable(s) = " + tous);
 else
 System.out.println("Il n'a a aucune page inatteignable.");
 }
}
```

Annexe D

Configuration de Xdebug

Il existe une extention pour PHP qui permet, entres autres, de faciliter la couverture du code. Cet extention se nomme Xdebug. Nous expliquons ici quelles sont les étapes à suivre pour activer Xdebug sur un serveur Web.

Premièrement il faut télécharger le fichier dll Xdebug sur le serveur Web à partir du site Web http ://xdebug.org/. Nous devons choisir le module Xdebug en tenant compte de la version PHP installée sur le serveur Web.

Il faut rajouter les lignes suivantes à la fin du fichier php.ini :

;Modifiez le chemin d'accès et la version de xdebug au besoin
zend_extension_ts="C :\wamp\bin \php \php5.2.5 \ext \php_xdebug-2.0.3-5.2.5.dll"
[xdebug]
xdebug.remote_enable=1
xdebug.remote_host="localhost"
xdebug.remote_port=9000
xdebug.remote_handler="dbgp"
xdebug.show_local_vars=1
xdebug.profiler_enable_trigger=1 ;
;Modifiez le chemin d'accès au besoin
xdebug.profiler_output_dir = "c :\wamp \journal"
;Permet le tracage automatique des instructions visitées dans le fichier trace-.xxxx.txt
xdebug.auto_trace=on
;Modifiez le chemin d'accès au besoin

xdebug.trace_output_dir=c :\wamp\journal
;Rajoute l'information à la fin du fichier trace.xxxx.txt.
xdebug.trace_options=On

Redémarrer le serveur Web. Lorsque le paramètre xdebug.auto_trace est activé le fichier trace.xxxx.txt nous informe, entres autres, sur les pages Web dynamiques visitées et les fonctions PHP utilisées à l'intérieur de ces pages PHP.

Annexe E

Requêtes XQuery pour WebML

```
( : EnsembleDesPages.xquery :)
( : Retourne l'id des de toutes les pages de l'application Web :)
for $EnsPages in doc("SDSProject.xml")//PAGE/@id
return data($EnsPages)

( : LiensDePageAPage.xquery :)
( : Retourne les pages qui ont un lien vers une autre page :)
for $a in doc("SDSProject.xml")//PAGE
for $b in doc("SDSProject.xml")//PAGE//LINK
where data($b/@to) = data($a/@id)
return
<Page>
{
'La page Web ',
data($b/../@id),' pointe vers la page Web ',
data($a/@id)
}
</Page>
```

Lorsque nous compilons une AW construite à l'aide de l'outil WebRatio celui-ci écrit la spécification de l'application dans un fichier XML. Dans la présente version de WebRatio (version 4.3) ce fichier se nomme SDSProject.xml. Pour vérifier qu'une page Web n'est pas isolée nous analysons la spécification écrite dans le fichier SDSProject.xml. Des requêtes XQuery sont utilisées pour analyser ce fichier XML et récupérer certaines données. Nous

insérons ensuite ces données dans le programme PagesInatteignables.java que nous avons réalisé dans le cadre de ce mémoire. Ce programme nous informe s'il y a ou non des pages inatteignables dans l'application concernée et si c'est le cas la liste des pages Web isolées est affichée.

Pour aider le lecteur à mieux saisir le fonctionnement de nos requêtes nous avons utilisé notre approche sur l'AW de vente de livres que nous avons construite à l'aide de WebRatio. Au départ nous avons exécuté la requête "EnsembleDesPages.xquery" sur le fichier SDSProject.xml. L'outil WebRatio assigne automatiquement un id à chacune des pages Web créées par un usager. Cette requête retourne l'ensemble des id. Voici le résultat de cette requête :
page8 page16 page6 page10 page14 page9

Ensuite nous exécutons la requête LiensDePageAPage.xquery. Comme son nom l'indique cette requête permet à l'usager de faire ressortir tous les liens qui existent entre les pages Web d'une AW construite à l'aide de l'outil WebRatio. Elle retourne le nom des pages reliées entre elles et la direction de ces liens. Voici la réponse obtenue pour notre exemple :
<Page>La page Web page6 pointe vers la page Web page16</Page>

Les données recueillies à l'aide des requêtes sont ensuite transmises dans notre programme PagesInatteignables.java. Voici le résultat que nous avons obtenu pour notre exemple :
Il n'a a aucune page inatteignable.

Bibliographie

[1] Oresto Signore. *Towards a Quality Model for Web Sites*, Proc. of CMG Poland Annual Conference, Warsaw, Polan, May 2005.

[2] L. Mich, M. Franch et L. Gaio. *Evaluating and Designing the Quality of Web Sites*, IEEE Multimedia, Jan-Mar, 2003, pp.34-43.

[3] Winston W. Royce. *Managing the Development of Large Software Systems*, Proceedings of IEEE Wescon, pages 1-9, Août 1970.

[4] B.W. Boehm. *A spiral model of software development and enhancement*, IEEE Computer, IEEE publisher, Mai 1988.

[5] M.C. Gaudel et al. *Précis de génie logiciel*, Masson, 1996.

[6] S. Elbaum, S. Karre et G. Rothermel. *Improving Web Application Testing with User Session Data*, In Proceedings of the 25th International Conference on Software Engineering, pages 49-59, Mai 2003.

[7] S. Sampath, E. Gibson, S. Sprenkle et L. Pollock. *Coverage Criteria for Testing Web Applications*, 2002.

[8] C. Kallepalli et J. Tian. *Measuring and Modeling Usage and Reliability for Statistical Web Testing*, IEEE transactions on software engineering, vol. 27, no. 11, November 2001.

[9] Nguyen, Q. Hung. *Testing Applications on the Web*, John Wiley & Sons, 2ième édition, 2002.

[10] P. Fraternali, P.L. Lanzi, M. Matera et A. Maurino. *Techniques and Tools for Exploiting Conceptual Modeling in the Evaluation of Web Application Quality*, IEEE, 2004.

[11] P. Fraternali, M. Matera et A. Maurino. *WQA : an XSL Framework for Analysing the Quality of Web Applications*, Proc. of IWWOST'02 - ECCOP'02 Workshop, Malaga, Espagne, Juin 2002.

[12] Luis Olsina et Gustavo Rossi. *Measuring Web Application Quality with WebQEM*, IEEE Transactions on Software Engineering, Vol. 27, No. 11, Novembre 2002.

[13] Xiaoping Jia and Hongming Liu. *Rigorous and Automatic Testing of Web Applications*, Proceedings of the 6th IASTED International Conference on Software Engineering and Applications (SEA 2002), Cambridge, MA, USA. pp.280-285, Novembre 2002.

[14] J. Sant, A. Souter et L. Greenwald. *An exploration of statistical models for automated test case generation*, ACM, 2005.

[15] Angelo Gargantini et Constance Heitmeyer. *Using Model Checking to Generate Tests from Requirements Specifications Software Engineering*, ESEC/FSE'99, 7th European Software Engineering Conference, Held Jointly with the 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering, Toulouse, France, September 1999, Proceedings. Lecture Notes in Computer Science, Vol. 1687, Springer, 1999.

[16] Ian Sommerville. *Software Engineering*, Addison-Wesley Publishers, 6ième édition, 2001.

[17] Filippo Ricca et Paolo Tonella. *Understanding and Restructuring Web Sites with ReWeb*, IEEE MultiMedia, Vol. 8, No. 2, pp. 40-51, 2001.

[18] WebRatio. http ://www.webratio.com/sv1.do, 2008

[19] IBM RAD. http ://www.ibm.com/developerworks/downloads/r/rad/, 2008

[20] Tue Becher Jensen, Terkel K. Tolstrup et Michael R. Hansen. *Generating Web-Based Systems from Specifications*, ACM Symposium on Applied Computing, Mars 2004.

[21] O. Stutz, I. Schicktanz et M. Wind. *Quality Criteria for a Public-User-Friendly and Secure Web Site*, University of Bremen, Bremen, Allemagne.

[22] Nicoletta Di Blas and al. *Evaluating the Features of Museum Web Sites : (The Bologna Report)*, Museums and the Web, Avril 2002.

[23] L. Olsina. *Specifying Quality Characteristics and Attributes for Web Sites*, Proc. Web Engineering : Managing Diversity and Complexity of Web Application Development, Juin 2001.

[24] W3C World Wide Web Consortium. http ://www.w3.org/, 2008.

[25] D. Balis et J. Garcia-Vivo. *A Rule-based System for Web site Verification*, In Proc. of 1st International Workshop on Automated Specification and Verification of Web Sites (WWV'05), Valencia (Espagne), 2005.

[26] Alin Deutsch and al. *A System for Specification and Verification of Interactive, Data-Driven Web Applications*, SIGMOD Conference 2006.

[27] Alin Deutsch et al. *A Verifier for Interactive, Data-Driven Web Applications*, SIGMOD Conference, ACM, 2005.

[28] Alin Deutsch, Liying Sui et Victor Vianu. *Specification and Verification of Data-driven Web Services*, PODS 2004.

[29] Xiaoping Jia and Hongming Liu. *Formal Structured Specification for Web Applications Testing*, 2003 Midwest Software Engineering Conference (MSEC 2003), Chicago, USA, June 2003.

[30] Software QA Test Resource Center. http ://www.softwareqatest.com/qatweb1.html, 2008

[31] Page demo Internet de WAVE. http ://www.cs.ucsd.edu/ lsui/project/bottom.html, 2008.

[32] M. Spielmann. *Verification of relational transducers for electronic commerce*, JCSS, Extended abstract in PODS 2000.

[33] G. Malak. *Évaluation de la qualité des applications Web*, Mémoire présenté au département d'informatique faculté des sciences et de génie Université Laval, Avril 2002.

[34] Antidote. http ://www.druide.com/antidote.html, 2008

[35] W3C Quality Assurance Tools. http ://www.w3.org/QA/Tools/, 2008

[36] C. Bellettini, A. Marchetto et A. Trentini. *WebUml : Reverse Engineering of Web Applications*, 2004 ACM Symposium on Applied Computing, Italie, 2004.

[37] C. Bellettini, A. Marchetto et A. Trentini. *TestUml : user-metrics driven Web Applications testing*, 2005 ACM Symposium on Applied Computing, Italie, 2005.

[38] Making your Website visible. http ://www.nsrc.org/helpdesk/visibility.html, 2008

[39] Meta-Tag Optimization Tips. http ://www.clickz.com/3496146/print, 2008

[40] Understanding Title, Meta tags, Text links and other elements. http ://www.arun.info/meta-data-elements, 2008

[41] Xdebug. http ://www.xdebug.org/, 2008

[42] A. Tappenden and al.. *Agile Security Testing of Web-Based Systems via HTTPUnit*, IEEE Proceedings of the Agile Development Conference, 2005.

[43] wave 4.0. http ://wave.webaim.org, 2008

[44] Web Accessibility Checker. http ://checker.atrc.utoronto.ca/index.html, 2008

[45] Browsershots. http ://browsershots.org/, 2008

[46] W3 Schools Browser Statistics. http ://www.w3schools.com/browsers/browsers_stats.asp, 2008

[47] W3C Link Checker. http ://validator.w3.org/checklink, 2008.

[48] F. Ricca et P. Tonella. *Analysis and Testing of Web Applications*, In Int Conf of Soft Eng, 2001.

[49] S. Sampath, V. Mihaylov, A. Souter et L. Pollock. *A Scalable Approach to User-session based Testing of Web Applications through Concept Analysis*, In Auto Soft Eng Conf, Septembre 2004.

[50] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock et A. Souter. *An Empirical Comparaison of Test Suite Reduction Techniques for User-sesssion-based Testing of Web Applications*, Proceedings of the 21th IEEE International Conference on Software Maintenance, pages 587-596, Septembre 2005.

[51] Parasoft WebKing. http ://www.parasoft.com, 2004.

[52] Rational corporation Rational testing robot. http ://www.rational.com/products/robot/, 2008.

[53] S. Comai, M. Matera et A. Maurino. *A Model and an XSL Framework for Analysing the Quality of WebML Conceptual Schemas*, Proc. of the IWCMQ'02 ER'02 Workshop, Tampere, Finlande, Octobre 2002.

[54] Sahi. http ://sahi.co.in/w/, 2008.

[55] Jeff Offutt et al. *Bypass Testing of Web Applications*, Proceedings of 28th Annual International Computer Software and Applications Conference, IEEE, 2004.

[56] Paolo Tonella et Filippo Ricca. *A 2-Layer Model for the White-Box testing of Web Applications*, Proc. of the sixth IEEE International Workshop on Web Site Evolution, 2004.

[57] W3C Web Accessibility Initiative. http ://www.w3.org/WAI/, 2008

[58] D. A. Wheeler. *Secure Programming for Linux and Unix HOWTO*, http ://www.dwheeler.com/secure-programs/,Mars 2003.