UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME ÉXIGENCE PARTIELLE DU
PROGRAMME DE MAÎTRISE EN
MATHÉMATIQUES ET INFORMATIQUE
APPLIQUÉES

PAR
GHANNEM ADNANE

TEST DES APPLICATIONS WEB : MODÉLISATION ET GÉNÉRATION DE SÉQUENCES DE TEST BASÉES SUR LE CONTRÔLE

JUIN 2006

Université du Québec à Trois-Rivières

Service de la bibliothèque

Avertissement

L'auteur de ce mémoire ou de cette thèse a autorisé l'Université du Québec à Trois-Rivières à diffuser, à des fins non lucratives, une copie de son mémoire ou de sa thèse.

Cette diffusion n'entraîne pas une renonciation de la part de l'auteur à ses droits de propriété intellectuelle, incluant le droit d'auteur, sur ce mémoire ou cette thèse. Notamment, la reproduction ou la publication de la totalité ou d'une partie importante de ce mémoire ou de cette thèse requiert son autorisation.

TEST DES APPLICATIONS WEB :

MODÉLISATION ET GÉNÉRATION DES SÉQUENCES DE TEST

BASÉES SUR LE CONTRÔLE

SOMMAIRE

Le Web a évolué très rapidement et la technologie Internet a été adoptée par des utilisateurs appartenant à des domaines de plus en plus diversifiés. Comparées aux applications traditionnelles, les applications Web présentent plusieurs caractéristiques propres. Le test constitue une action cruciale dans l'assurance de leur qualité.

Nous présentons, dans ce mémoire, une méthodologie de test (génération et vérification) pour les applications Web basée sur les interactions dynamiques entre leurs différents composants. L'approche adoptée tient compte de plusieurs aspects reliés à leur contrôle. Le contrôle est utilisé pour la génération des séquences de test. Le processus de génération des séquences se base sur un arbre, construit par analyse statique du code source de l'application, représentant les différents chemins de contrôle dans une navigation. Les séquences générées, sous forme compactées, correspondent, en fait, aux différents scénarios de navigation dans une application web. Elles sont utilisées pour vérifier l'exécution de chaque scénario. Nous présentons également l'architecture ainsi que les composantes de l'outil développé pour supporter notre approche.

WEB-BASED APPLICATIONS TESTING:

MODELING AND GENERATING TESTING SEQUENCES

USING CONTROL

ABSTRACT

Web-based applications evolved very quickly and the Internet technology has been adopted by users belonging to increasingly diversified fields. Compared to traditional applications, Web-based applications show several particular characteristics. Their test constitutes a crucial task in the insurance of their quality.

The current work presents a methodology (generation and verification) supporting Web-based applications testing. The approach is based on the dynamic interactions between their various components. It takes into account several aspects related to their control. Control is used for the generation of the testing sequences. The generation process of the sequences is based on a tree, built by an automatic static analysis of the source code of the application, representing the various ways of control in a navigation. The generated sequences, in compacted form, correspond in fact to the various scenarios of navigation in a Web-based application. They are used to check the execution of each scenario. We also present the architecture as well as the components of the tool we developed to support our approach. The tool is illustrated on a real case study.

DÉDICACES

Je dédie ce mémoire

À mes chers parents Youssef Ghannem et Henia Ben-Belgacem pour tous leurs sacrifices et leur amour constants.

À mon cher frère Ramzi Ghannem, à mes chères sœurs Ines Ghannem et Asma Ghannem pour tout leur amour.

À ma chère fiancée Sonia Chaabane pour son soutien indéfectible.

À mes directeurs de recherche Linda Badri et Mourad Badri qui ont été plus que des directeurs de recherche pour moi.

REMERCIEMENTS

L'auteur voudrait avant tout remercier l'Unique pour tous Ses bienfaits trop souvent négligés.

Il m'importe ensuite de remercier ardemment toutes les personnes ayant contribué à la réalisation de ce mémoire. Mes premières pensées se dirigent vers mes directeurs de recherche, les professeurs Linda Badri et Mourad Badri. Merci pour votre patience et vos encouragements indéfectibles.

Je remercie, aussi, les professeurs Fathallah Nouboud et Hélène Desaulniers qui ont bien voulu donner de leur temps pour lire et suggérer les correctifs nécessaires à l'approbation de mon mémoire comme document scientifique.

Je remercie chaleureusement Chantal Lessard qui a su mettre de la vie dans la réalisation de ce mémoire ainsi que tous les professeurs et les étudiants de la maîtrise en mathématiques et informatique appliqués pour leur amitié désintéressée et sincère.

Je ne peux remercier toutes ces personnes sans mentionner mes parents, mes frères et sœurs, ces personnes chères à mes yeux qui m'ont apporté tout ce dont j'avais besoin pour une bonne réussite : confiance, amour et amitié.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

CHAPITRE

1

INTRODUCTION

1 PRÉSENTATION GÉNÉRALE

Les applications Web ont été intégrées dans des domaines diversifiés (commerce électronique, éducation, loisir, etc.) [MAL 02]. Elles sont devenues des programmes complexes, sophistiqués qui intègrent de plus en plus les nouvelles technologies du monde Web. Ces technologies apportent de nouveaux défis aux développeurs et aux testeurs. Les exigences du marché, la compétition, et les transactions monétaires via Internet font surgir l'épineux problème de leur fiabilité et sécurité. De plus, ces applications évoluent très rapidement et plusieurs d'entre elles sont pauvrement conçues et présentent des lacunes sur les plans de la navigabilité, de la fonctionnalité et de la traçabilité [FLE 99].

Les applications Web possèdent leurs propres caractéristiques qui les différencient des applications traditionnelles. Plusieurs chercheurs [WU 02] ont discuté les différences essentielles qui existent entre les applications web et les applications traditionnelles. Ils ont surtout mis l'accent sur des caractéristiques telles que l'architecture, les facteurs de qualité, le contrôle ainsi que la maintenance. L'architecture des applications Web est similaire à celle des systèmes client-serveur sur plusieurs aspects avec néanmoins quelques différences. Dans les systèmes client-serveur traditionnels, les rôles respectifs

des clients et des serveurs ainsi que leurs interactions sont prédéfinis (statiques). Cependant, dans les applications Web les programmes clients et leurs contenus peuvent être générés dynamiquement. Par ailleurs, les facteurs de qualité, selon plusieurs auteurs, ne semblent pas avoir la même importance dans le cas des applications Web et des applications traditionnelles. Pour ces dernières, l'exactitude et l'efficacité sont parmi les facteurs de qualité les plus importants. Des caractéristiques telles que la compatibilité et l'interopérabilité sont plus importantes dans le contexte des applications Web [WU 02]. Le flux de contrôle est entièrement géré par le programme dans le cas des applications traditionnelles. Dans le cas des applications Web, l'utilisateur peut interrompre le flux de contrôle, lors de la navigation, sans aviser le contrôleur de l'application. L'activité de maintenance est, par ailleurs, très importante et très fréquente dans le cas des applications Web contrairement aux applications traditionnelles. Ceci est dû essentiellement à l'évolution rapide des technologies Web. La maintenance doit non seulement être fréquente, mais en plus efficace en raison de la pression du *time-to-market* dans les applications Web [WU 02]. Les applications Web possèdent également des caractéristiques qui ne sont pas présentes dans les systèmes client-serveur et les systèmes distribués, en particulier, le contrôle de session, les cookies, l'aspect apatride de http et les nouvelles exigences de sécurité.

Il y a deux aspects pertinents au sein d'une application Web [YAN 98]. Le premier est relié aux aspects statiques, qui peuvent être gérés relativement facilement. Le deuxième, le plus complexe, est relié aux aspects dynamiques et comprend plusieurs éléments tels que les serveurs Web et les bases de données serveurs ODBC (*Open DataBase Connection*) ou JDBC (*Java DataBase Connection*) [YAN 98].

Le besoin de mettre au point des principes et méthodes spécifiques au développement des applications Web, d'une manière générale, et à l'assurance de leur qualité, en particulier, se fait donc de plus en plus ressentir dans l'industrie du logiciel. L'évaluation de leur qualité ainsi que leur test sont des sujets très importants qui

suscitent de plus en plus l'intérêt des chercheurs [YAN 98, YAN 99a, YAN 99b, YAN 02, WU 02]. C'est dans ce contexte, dans un objectif d'assurance qualité, que s'insère notre travail. La méthodologie de test (génération et vérification) que nous proposons pour les applications Web, est basée sur les interactions dynamiques entre leurs différents composants. Elle tient compte de plusieurs aspects reliés à leur contrôle. Dans le cadre de notre approche, une application Web est modélisée en tenant compte de plusieurs niveaux de contrôle : le niveau application, le niveau module, le niveau page et finalement le niveau script. Ces niveaux suivent un classement hiérarchique permettant ainsi la construction d'un graphe de contrôle complet. Le graphe de contrôle complet est transformé en un arbre à partir duquel sont générés les différents chemins de contrôle correspondant aux chemins de navigation potentiels dans l'application. Les chemins générés, sous forme compactée, sont transformés en séquences de test. Les séquences générées sont utilisées pour vérifier l'exécution de chaque scénario de navigation. Nous présentons également l'outil que nous avons développé pour supporter notre approche et illustrons son utilisation sur une étude de cas.

1.1 Définition d'une application Web

Selon Jim Conallen [CON 00] : « *L'application Web est un ensemble de formulaires et de pages HTML générées dynamiquement (au moins en partie), et qui constitue une application métier. L'exemple le plus simple est celui d'une horloge : en cliquant sur un lien ou un bouton on obtient en retour l'heure courante. Cette dernière bénéficie d'avantages indéniables tels qu'un déploiement beaucoup plus aisé qu'en mode client-serveur et un accès universel à moindre coût. L'interface privilégiée est celle du client léger, mais on pourra lui associer d'autres technologies si des interfaces encore plus riches sont nécessaires. La pierre angulaire d'une application Web est le serveur d'applications.* »

Les applications Web possèdent leurs propres caractéristiques qui les différencient des applications traditionnelles. Dans un cadre comparatif, Ye Wu et al. [WU 02] ont présenté une brève comparaison entre les applications traditionnelles et les applications Web (Tableau 1) mettant en évidence les caractéristiques les plus importantes telles que l'architecture, les facteurs de qualité, les aspects importants reliés au contrôle ainsi que la maintenance.

Applications / Caractéristiques	**Client Serveur traditionnelles**	**Web**
Architecture	Les rôles respectifs des clients et des serveurs sont prédéfinis (statique).	Les programmes clients et leurs contenus peuvent être générés dynamiquement
Qualité	Exactitude, efficacité.	Exactitude, efficacité compatibilité, interopérabilité
Flux de contrôle	Géré par le programme	L'utilisateur peut interrompre ces flux sans même avertir le contrôleur de l'application.
Maintenance	Moins prioritaire	Prioritaire, fréquente et doit être efficace. => importance de proposer de nouvelles techniques de test.

Tableau 1 : Comparaison entre les applications Web et les applications traditionnelles [WU 02].

1.2 Spécificités des applications Web

Selon [HEN 01], les applications Web se différencient des application classiques par :

- Une uniformisation de l'interface client basée sur le langage HTML et les navigateurs ;

- La nécessité d'utiliser des passerelles pour accéder aux données ;
- La protection par des coupe-feu (*firewall*) ;
- L'utilisation (simultanée) de plusieurs langages de programmation (HTML, Protocoles http, Java, Pages ASP, VBScript, CGI, PERL, etc);
- L'intégration massive des technologies multimédias ;
- L'accès illimité aux applications.

La validation des applications Web est donc une entreprise d'envergure, et les techniques de test ne sont pas encore complètement finalisées. Le domaine du test des applications Web est relativement jeune.

Contrairement aux applications traditionnelles où les méthodes de test sont plutôt statiques, les applications Web nécessitent l'utilisation d'outils spécialisés (la quantité d'information à manipuler est trop importante) ainsi que la prise en compte de l'aspect dynamique qui domine.

2 CATÉGORIES DE TESTS

2.1 Qu'est ce que le test ?

Définition : Le test est l'exécution ou l'évaluation d'un système ou des composants d'un système par des moyens manuels ou automatisés afin de vérifier qu'il répond à ses spécifications ou pour identifier les différences entre les résultats obtenus et les résultats attendus [HAY 94] [IEEE 82].

Le test est une activité importante du processus de Vérification & Validation (V&V) [HAY 94] [PRE 04]. Le processus de *V&V* permet à la fois de découvrir les défauts et de garantir que le logiciel correspond bien aux attentes du client. Il existe de

nombreuses définitions du processus de *Vérification & Validation.* Boehm [BOE 79], en exposant la différence entre la vérification et la validation, a implicitement donné une définition au processus de Vérification et Validation :

- *Validation : Avons-nous livré le bon produit ?*
- *Vérification : Avons-nous livré le produit correctement ?*

Les définitions les plus utilisées et acceptées du processus de Vérification et Validation proviennent de ANSI/IEEE Standard 729- 1983.

- *La Vérification est le processus consistant à déterminer si les produits d'une phase donnée du cycle de développement remplissent les spécifications établies durant la phase précédente.*

- *La Validation, quant à elle, est le processus d'évaluation du logiciel à la fin du développement pour garantir le respect des exigences du logiciel.*

Cependant, les approches standard (traditionnelles) de test ne couvrent pas toutes les questions soulevées par la validation des applications Web mentionnées par plusieurs auteurs [YAN 98], [HUA 98], [RIC 00] et [NIE 00]. Avant de présenter les concepts de base relatifs au test des applications Web, nous allons tout d'abord exposer les approches classiques de test de logiciels.

2.2 Test conventionnel

Le test conventionnel est souvent divisé en deux catégories à savoir : le test statique et le test dynamique [HAY 94]. Ces deux catégories de test sont présentes dans le processus

de Vérification & Validation. À l'intérieur de ce processus, il existe deux approches fondamentales (figure 1) pour la vérification et l'analyse d'un système [SOM 04]:

- *Inspection du Logiciel (test statique) : Analyse et vérification des représentations du système telles que les documents d'exigences, les modèles de conception et le code source des programmes.*

- *Test de logiciel (test dynamique) : implique l'exécution d'une implémentation du logiciel avec des données de test.*

Ces deux approches jouent des rôles complémentaires. Elles peuvent être reliées à plusieurs approches présentes dans la littérature à savoir les approches statiques, structurelles et fonctionnelles. Cette complémentarité (inspections de logiciel et test de logiciel) est illustrée par la figure suivante :

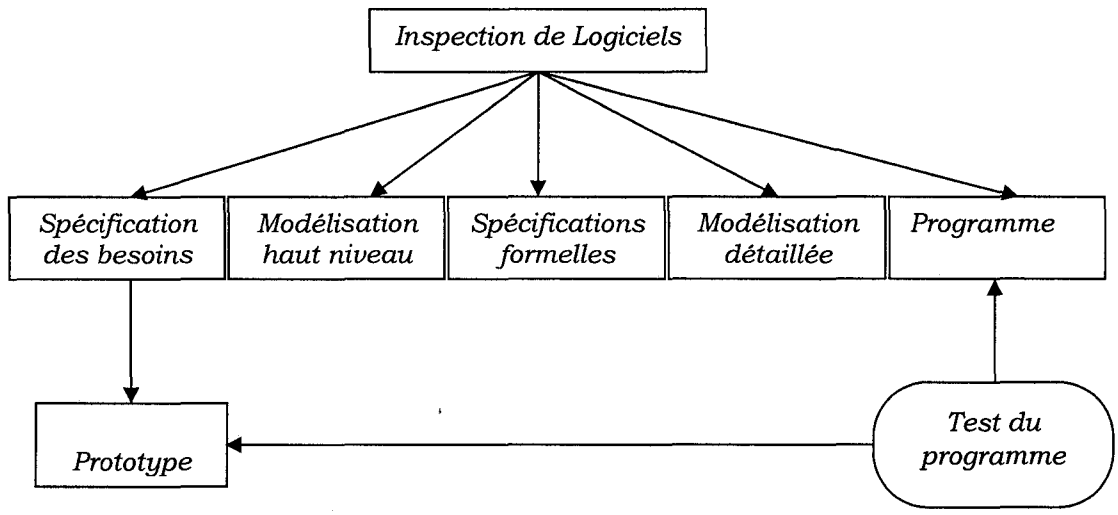


Figure 1 : Vérification et Validation statique et dynamique [SOM 04].

Les tests statiques et dynamiques tels que présentés sont supportés par différentes techniques.

2.2.1 Test statique

Le test statique se résume à une analyse, sans exécution, d'une représentation (modèle, programme, etc.) [CHU 92]. Les techniques de vérification utilisées lors du test statique impliquent l'analyse de la représentation qui peut correspondre par exemple à un code source du programme pour détecter des erreurs [SOM 04]. Parmi ces techniques, nous pouvons énumérer :

- *le processus d'inspection du programme qui met l'emphase sur la détection de défauts [Fag76] [Fag86].*
- *l'analyse statique automatisée qui examine le code sans exécution [SOM 04].*

2.2.2 Test dynamique

Le test dynamique consiste à exécuter un programme (ou partie d'un programme) afin d'analyser son comportement [CHU 92]. Tout comme le test statique, le test dynamique introduit un ensemble de techniques telles que [SOM 04], [PRE 04]:

- *Le test de composants (test unitaire)*
- *Le test d'intégration*
- *Le test d'acceptation (test de validation)*

Tout d'abord, les composants (test de composants) sont testés en premier, ce qui correspond au test unitaire. Ensuite, les tests d'intégration consistent à tester les interactions entre ces composants lors de leur intégration. Et enfin, les tests de validation

portent sur le test global des modules intégrés correspondant au test du système [HAY 94], [SOM 4].

Une des principales difficultés du processus de test porte sur la détermination des cas de test. La construction d'un ensemble de données de test sur lequel une exécution correcte d'un programme permettant de confirmer que le programme est correct, et ceci quelques soient les données mises en œuvres (domaine d'entrées du programme), est généralement impossible. Cela veut dire que les cas de test doivent être choisis de manière pertinente, car compte tenu des contraintes de temps, de coût, et de moyens, il est difficile d'avoir des couvertures de test totales.

Toutefois, en raison des nouvelles caractéristiques des applications Web, le test conventionnel ne peut être appliqué directement aux applications Web. Ces caractéristiques apportent de nouvelles dimensions en termes de complexité qui ne sont pas couvertes par les méthodes de test conventionnel.

2.3 Les différentes techniques de test

Selon [SOM 04][PRE 04], le processus de test, d'une façon générale, inclus :

2.3.1 Tests unitaires

C'est un processus ayant pour but de tester chaque composant du logiciel pris isolément (sous système, module, etc.). Cette étape de test présente plusieurs avantages :

- Concentration de l'effort de test sur chaque composant du logiciel pris isolément, ce qui représente une tache relativement simple, sauf si ce composant est fortement connecté aux autres (on parle de couplage).

- En cas de comportement anormal du composant, il est relativement plus facile de découvrir l'erreur, de la localiser, et enfin de la corriger.

- Enfin, les tests unitaires introduisent la notion de parallélisme durant le processus de test ; plusieurs composants du logiciel peuvent être testés indépendamment des autres et surtout simultanément, ce qui permet de gagner du temps (temps, coût, efficacité, etc.).

2.3.2 Tests d'intégration

Les différents composants qui ont été testés individuellement vont être progressivement assemblés pour construire un programme exécutable qui va être testé. Ce type de test permet surtout de vérifier la bonne utilisation des interfaces entre composants. Ils permettent également de vérifier que le système, dans son ensemble, réalise bien les fonctions spécifiées.

2.3.3 Tests de validation

Ce type de test intervient à la fin du processus d'intégration. Il permet de tester le système dans son ensemble avec des données réelles

Rosenberg [ROZ 98] a présenté, de façon structurée, un tableau (voir tableau 2) qui met en relation les catégories de test par rapport aux phases de développement du logiciel.

	Développement composant	Intégration	Validation
Boîte blanche	Tests de chemin Test de fuite mémoire Tests aux limites	Tests de chemin.	
Boîte noire	Tests unitaires		Tests de régression Tests de performance
Boîte grise		Tests an 2000 Tests euro	Test de régression

Tableau 2 : Phases de développement [ROZ 98].

3 PROBLÉMATIQUE

Le test a pour but, tel que mentionné précédemment, de vérifier si le système répond à ses spécifications et de mettre en évidence les défauts. Le test des applications Web, ainsi que de leurs différentes caractéristiques, ne peut se faire en se basant uniquement sur les techniques de test conventionnel [ROZ 98]. Il est nécessaire, donc, d'utiliser (développer) de nouvelles techniques et approches [WU 02]. Deux principales classes d'approches émergent dans la littérature : les approches structurelles et les approches fonctionnelles. Ces approches ont pour objectif d'assurer l'intégrité des composants système (testées individuellement), d'une part, et d'assurer une intégrité du système dans sa globalité, d'autre part.

Le besoin de mettre au point des principes et méthodes spécifiques au développement des applications Web, d'une manière générale, et à l'assurance de leur qualité, en particulier, se fait en effet de plus en plus ressentir dans l'industrie du logiciel. L'évaluation de leur qualité ainsi que leur test sont des sujets très importants qui suscitent de plus en plus l'intérêt des chercheurs [YAN 98], [WU 02], [RIC 00],...etc.

Le reste du mémoire est organisé comme suit : le chapitre 2 résume les principaux travaux effectués dans le domaine du test des applications Web. Le chapitre 3 présente la méthodologie de l'approche que nous proposons. L'outil associé, son implémentation ainsi que son illustration sur une étude de cas réelle seront présentés dans le chapitre 4. Enfin, le chapitre 5 présente une conclusion et quelques travaux futurs.

CHAPITRE

2

ÉTAT DE L'ART

1 INTRODUCTION

Les travaux sur le test des applications Web se sont succédés durant les dernières années [ELB 03, ELB 05], [YAN 98], [KUN 00], [RIC 00], ...etc. Ces travaux, spécifiques, visent à fournir des approches efficaces permettant de tester, d'une façon plus objective, les applications Web. Le besoin de mettre au point des principes et méthodes spécifiques à l'assurance qualité des applications Web est urgent. L'évaluation de leur qualité ainsi que leur test sont des sujets très importants qui suscitent de plus en plus l'intérêt des chercheurs [YAN 98, 99a, 99b, 02], [WU 02]. Certains travaux, comme [RIC 00], [YAN 98], ont abouti au développement d'outils automatisant, en partie, le processus d'évaluation de la fiabilité et de la maintenabilité des applications Web [MAL 02].

Cependant, des efforts importants en termes de recherche demeurent nécessaires, d'une part, pour valider et compléter ces travaux et, d'autre part, pour développer des techniques de test permettant de répondre au mieux à l'assurance qualité des applications Web. Par ailleurs, le développement des approches basées sur le contrôle et permettant de tenir compte des aspects dynamiques de ces applications s'avère nécessaire.

Plusieurs travaux sur le test des applications Web sont menés par plusieurs équipes de chercheurs un peu partout dans le monde. Dans la section suivante, nous dressons un état de l'art des principaux travaux publiés dans le domaine du test des applications Web. Nous concluons, enfin, dans la section 3 par les principaux enseignements que nous avons tirés de ce survol bibliographique.

2 ÉTAT DE L'ART

2.1 Les travaux de JI-Tzay Yang

Ji-Tzay Yang et al. [YAN 98] ont été parmi les premiers à s'être intéressés au domaine du test des applications Web. Ils ont mentionné l'existence de plusieurs techniques qui facilitent le développement des applications Web et dans le même temps souligné le manque d'outils qui supportent le test de ce type d'applications. Ils ont proposé une méthode permettant de modéliser les constituants d'une application Web et d'appliquer les stratégies traditionnelles de génération des cas de test. Le modèle proposé, décrivant le comportement, est transformé en un graphe de contrôle ou en un graphe des flux de données. La construction du modèle est basée sur deux types d'analyse : statique et dynamique.

Ils ont illustré leur approche sur une application qui a pour rôle de saisir des commandes et offrir des services en ligne sur plusieurs produits (Pizza, boissons…etc.). L'exemple illustré (voir annexe : figure 16) contient plusieurs modules et pages Web (*welcome.html*, *memberInfo.html*, *orderStart.asp, incomplete.html, availOffer.asp, availDrinks.asp etc.).*

Ils ont, par ailleurs, proposé un environnement de test supportant leur méthode. Cet environnement englobe trois outils intégrés de test : *test case composition*, *test case generator* et *test case execution*. Les techniques de génération de cas de test basées sur

les graphes de flux de contrôle peuvent être adaptées aux applications Web. Selon **JI-Tzay Yang et al.** [YAN 98], il y a deux stratégies de chemin de test :

- La stratégie basée sur l'exécution des instructions (couverture d'instructions) qui exige que les cas de test générés permettent l'exécution de chaque instruction dans le programme au moins une fois.

- La stratégie basée sur l'exécution des branches (couverture des branches) qui exige que les cas de test générés permettent l'exécution de chaque alternative de branche au moins une fois.

Dans [YAN 99a, YAN 99b], **JI-Tzay Yang et al.** ont mis l'emphase sur l'architecture de l'environnement de test supportant leur méthodologie. Cet environnement consiste en cinq sous systèmes : *Source Document Analysis*, *Test Management*, *Test Development*, *Test Failure Analysis*, *Test Execution*, et *Test Measurement*. Chaque sous-système coopère avec chacun des autres pour accomplir les activités de test.

Selon **JI-Tzay Yang et al.** [YAN 99a], les activités de test sont groupées dans les quatre étapes majeures suivantes :

1. La construction du modèle d'application (Modèle de graphe de flux de contrôle ou de graphe de flux de données) ;

2. La construction et la composition des cas de test ;

3. L'exécution des cas de test ;

4. La validation de la mesure des résultats de test.

La construction du modèle d'application est de la responsabilité du sous système d'analyse de document (*Source Document Analysis*). L'analyseur HTML (*HTML Analyser*) cherche les modules de programmation (pages Web) de l'application Web et lance une analyse statique des liens du code source de chaque module. Le module *Server Side Analyser* exécute l'analyse dynamique des liens sur les sorties de HTML par l'interprétation du code source avec le serveur WWW (*World Wide Web*). Le module *Client Script Analyser* exécute l'analyse dynamique des liens du côté client par la conduite du navigateur WWW pour chercher les modules de programmation et analyse les scripts du côté client. Les modèles d'application sont construits par ces analyseurs.

Cette architecture a été implémentée grâce à un prototype. Les Scripts client ont été codés en JavaScript et tournent sous un navigateur Web. Les scripts serveur sont implémentés comme une application Java. Les utilisateurs de l'environnement de test peuvent manipuler les outils depuis les Applets java incorporées dans les pages Web.

Dans [YAN 02], **JI-Tzay Yang et al.** ont proposé une architecture permettant de réutiliser plusieurs modèles et architectures de logiciel des environnements de test traditionnels. Cette architecture (voir annexe : figure 17, 18) assiste également la construction des environnements de test des applications Web par l'extension d'autres architectures bien évaluées et par l'application de certains modèles de conception. Elle se compose de cinq sous-systèmes :

- TMS (*Test Management Subsystem*) et TDS (*Test Development Subsystem*) guident les testeurs dans le développement des cas de test. L'effort exigé pour construire les cas de test pour les applications Web est réduit par la capacité du prototype à enregistrer les modes opératoires d'application et les entrées de données sous forme HTML via un navigateur Web.

- TES (*Test Execution Subsystem*) et TFAS (*Test Failure Analysis Subsystem*) aident le personnel de test à exécuter les cas de test et à vérifier le résultat d'exécution. Ils peuvent aussi faciliter l'exécution de test et réduire l'effort nécessaire pour automatiser l'entrée de données d'application ainsi que la vérification du résultat des procédures dans les environnements Web.

- TMES (*Test Measurement Subsystem*) produit un sommaire de test.

2.2 Les travaux de Suet Chun Lee et Jeff Offutt

Dans [CHU 01], **Suet Chun Lee et al.** ont présenté une technique basée sur l'utilisation de l'analyse de mutation pour tester l'exactitude sémantique des interactions des composants basés XML (*eXtended Markup Language*). Ils ont utilisé un modèle de spécification d'interaction : ISM (Interaction Spécification Model) qui se compose des définitions de type de documents DTD (*Definition Type Document*), des caractéristiques de transmission de messages, et d'un ensemble de contraintes. Pour la construction des cas de test, les auteurs se basent sur les messages XML circulant entre les composants logiciel de l'application Web (voir annexe : figures 19, 20). L'objectif visé à travers ce modèle (ISM) consiste à assurer la fiabilité des interactions entre les composants dans un système basé Web. Ils ont proposé un prototype de test appelé *Web_Test* tel qu'illustré par la figure (21) (voir annexe).

2.3 Les travaux de Xiaoping Jia

Xiaoping Jia et al. [JIA 02] ont insisté sur l'importance de la qualité dans les applications Web. Dans leurs recherches, ils ont proposé une approche pour un test rigoureux et automatique des applications Web. L'approche est basée sur les spécifications formelles. Elle est par ailleurs, supportée par un outil permettant de charger le fichier de spécification de test, de générer les cas de test et enfin d'exécuter

les cas de test. L'outil développé est composé de plusieurs composants (*Test Generator*, *Test Runner*, *Test Report*).

2.4 Les travaux d'Anneliese A. Andrews et al.

Anneliese A. Andrews et al. [AND 04] ont proposé une approche pour la génération des cas de test utilisant la technique des machines à états finis ou FSM's (*Finite State Machine*) combinée à des contraintes. L'approche présentée par **Anneliese A. Andrews et al.** [AND 04], consiste d'abord à partitionner l'application Web en sous-systèmes ou composants, construire une agrégation FSM pour l'application et générer enfin à partir du modèle obtenu les cas de test qui seront sous forme de séquences de pages Web.

2.5 Les travaux de Ye Wu et Jeff Offutt

Ye Wu et al. [WU 02] ont proposé un modèle d'analyse qui capture et modélise les concepts dynamiques des applications Web indépendamment de la technologie utilisée. Ils ont basé leur recherche sur l'aspect dynamique de l'application Web et la façon de mieux gérer cet aspect. Dans leur approche, **Ye Wu et al.** [WU 02] ont essayé de construire un modèle d'analyse permettant de générer tous les cas de test possibles à partir du code source écrit en Java. L'hypothèse de départ, considérée par **Ye Wu et al**, porte sur le fait que les clients et les serveurs communiquent uniquement via des pages HTML. Ils ont introduit, dans ce contexte, la notion de section atomique qui correspond à la plus petite unité physique élémentaire dans le code (par exemple : une instruction java qui permet d'afficher un message). Par la suite, et grâce aux règles de composition (la sélection, la séquence et l'agrégation), ils forment des expressions donnant chacune naissance à un cas de test.

2.6 Les travaux de Filippo Ricca et Paolo Tonella

Filippo Ricca et al. [RIC 00, RIC 01, RIC 02] ont présenté deux outils pour supporter le test des applications Web : *ReWeb* et *Test Web* (voir annexe : figure 21). Le modèle conceptuel des deux outils a été conçu en UML. Quand à *ReWeb*, il permet de générer un modèle UML qui sera utilisé par *TestWeb*. Dans ce modèle, la page Web correspond au noyau central qui contient l'information qui sera affichée à l'utilisateur ainsi que les liens de navigation vers les autres pages. *TestWeb* est un outil de test qui considère, en particulier, le test en boite blanche des applications Web. Les critères de test considérés dans ce cas sont : Page testing, Hyperlink testing, Definition-use testing, All-uses testing, All-paths testing. Le cas de test pour l'application Web est composé du triplet suivant : URL, input (la séquence des valeurs des variables séparées par le caractère « & »), type des paramètres passés (GET ou POST). L'exécution consiste à lancer des requêtes au serveur Web via des URL avec les inputs associés et les pages stockées en output. L'outil *TestWeb* contient le *Test Generator*, capable de générer les cas de test depuis le modèle UML de l'application Web. L'utilisateur peut intervenir par exemple pour choisir le critère de test.

2.7 Les travaux de Jens Uwe Pipka

Jens Uwe Pipka [UWE 03] a présenté deux types d'architecture de l'application Web : la première est basée sur l'aspect JSP (*Page Centric Approach*), la deuxième, est basée sur le modèle MVC (*Model-View-Controller*). Ce modèle est basé sur la technologie JSP (*Java Server Pages*) combinée avec la technologie des servlets. Dans ce modèle, l'auteur fait la différence entre l'utilisation d'une servlet simple et l'utilisation des servlets individuelles pour les requêtes clients. Partant de cela, l'auteur présente son approche pour la conduite du test dans le développement des applications Web.

Selon l'auteur, le MVC (*Model-View-Controller*) offre la possibilité de diviser les activités de test en trois scénarios de test : test de modèle, test de vue et test de contrôleur.

1. test de modèle : implémenté en Java Beans. L'implémentation peut être testée en utilisant les tests unitaires traditionnels. Ces tests peuvent s'exécuter indépendamment du reste de l'application Web.

2. test de vue : l'output du code source JSP et HTML peut être vérifié au complet. À cet effet, il est nécessaire d'utiliser les tests fonctionnels du côté client qui simulent la requête client et contrôlent la réponse serveur.

3. test de contrôleur : dans la servlet, le processus de requête incluant les flux de contrôle et l'initialisation du contexte sont testés. Il est nécessaire donc de simuler le comportement de la requête de la servlet.

2.8 Les travaux de Jiun-Long Huang

Jiun-Long Huang [HUA 98] a proposé, dans sa thèse de doctorat, un modèle d'application Web basé sur l'architecture trois tiers pour extraire le comportement. Il a aussi étendu plusieurs techniques conventionnelles de test et les a appliquées sur son modèle. En plus, il a proposé une architecture trois tiers pour un environnement de test. Cette architecture est décrite en détail dans [YAN 99a] [YAN 99b] [YAN 02].

2.9 Les travaux de Oliver Niese

Durant l'année 2000, **Oliver Niese et al.** [NIE 00] s'intéressent au domaine de CTI (*Computer Telephony Integrated*) en utilisant les concepts de spécification et de vérification des déroulements des opérations (*Workflows*). Les auteurs ont présenté un

environnement de test de niveau pour les applications CTI. Cet environnement contrôle non seulement les outils de test individuels mais tout le cycle de vie du système au niveau du test fonctionnel tout en incluant la conception de test, la génération de test, l'exécution de test, l'évaluation et le rapport de test.

Par ailleurs, **Oliver Niese et al.** [NIE 01a, 01b, 02] ont présenté un environnement de test d'intégration capable de faire l'adressage des applications réparties en totalité. Cet environnement couvre le maximum des aspects fonctionnels. En effet, leur environnement est un ensemble d'outils dont le plus important est *Test Coordinator*, construit par « *Agent Building Center* ». Le système ITE peut être utilisé pour tester les systèmes complexes tels que : les applications Web ou les solutions CTI.

2.10 Les travaux de David C. kung

David C. kung et al. [KUN 00] ont proposé une méthodologie de test utilisant un modèle WTM (*Web Test Model*) capturant les artefacts de test structurel et comportemental. Le modèle utilisé est basé sur trois perspectives : l'objet, le comportement et la structure. Le perspectif objet est décrit en utilisant l'ORD (*Java DataBase Connection*). La perspective comportementale, décrite dans le PND (*Page Navigation Diagram*), est une machine à états finis décrivant le comportement de la navigation dans une application Web. Par ailleurs, un ensemble d'OSD (*Object State Diagram*) est utilisé pour décrire les états des objets. La perspective structure, décrite dans les modèles BBD (*Block Branch Diagram*) et FCD (*Function Cluster Diagram*), capture les informations reliées aux flux de contrôle et de données dans les scripts et les fonctions continues dans une application Web. Leur modèle n'est supporté par aucun outil.

2.11 Les travaux de Barbara J. Gabrys

En s'intéressant au domaine du commerce sur le Web, **Barbara J. Gabrys et al.** [GAB 01] affirment que les affaires du commerce via le Web présentent de nouveaux défis pour les stratégies du test traditionnel. Dans ce contexte, ils ont proposé une approche de test conduite par les risques associés aux objectifs à atteindre dans des affaires de commerce électronique. Selon **Barbara J. Gabrys** [GAB 01] : « on peut adapter les méthodes de test traditionnelles en prenant en compte les différences qui existent entre les deux types d'application aussi bien traditionnelles que Web ».

2.12 Les travaux de Achim D. Brucker

S'intéressant plus particulièrement aux composants des systèmes distribués, **Achim D. Brucker** [BRU 01] a présenté une approche pragmatique, basée sur les diagrammes d'UML annotés avec des contraintes OCL (*Object Constraint Language*), qui utilise les méthodes formelles.

2.13 Les travaux de Peter Provost

Peter Provost [PRO 02] a proposé un outil de test appelé TDD (*Test-Driven Development*) qui permet de tester le code source de l'application Web. L'auteur a présenté, dans son travail, un guide d'utilisation du TDD, dans lequel il décrit les différentes fonctionnalités.

2.14 Les travaux de Tiziana Margaria

Tiziana Margaria et al. [MAR 02a, 02b, 02c] ont proposé une méthodologie pour le test fonctionnel des applications Web en se basant sur l'expérience acquise au niveau du test des applications CTI (*Computer Telephony Integrated*) [NIE 00, 01a, 01b], là où son

approche proposée a été appliquée avec succès. Les propriétés principales de l'environnement de test, qui supportent cette méthodologie, sont premièrement la facilité de manipuler le test des applications Web et la possibilité de fournir des propriétés concernant les cas de test à travers un modèle de contrôle qui assure les conditions d'exécution et le but de test.

2.15 Les travaux de Michael Benedikt

Michael Benedikt et al. [BEN 02] ont présenté une vue générale de l'outil « *VeriWeb* » qui a pour fonction la découverte et l'exploration systématique des chemins d'exécution d'une application Web.

2.16 Les travaux de Eugenio Di Sciascio et al

Eugenio Di Sciascio et al. [SCI 02] ont proposé une méthode formelle pour la vérification des applications Web, supportée par un outil appelé *AnWeb*. L'outil exécute le code source HTML des pages Web, incluant le code des pages dynamiques, construit le modèle SMV (*System Model View*) et fournit ses propres spécifications CTL (*Computer Tree Logic*).

2.17 Les travaux de Parasoft

L'organisme **Parasoft** [PAR 03] a présenté un outil appelé *WebKing* qui permet d'automatiser les pratiques de la vérification dans les applications Web : les analyses statiques, le test fonctionnel et le test de téléchargement. Cet outil aide aussi à vérifier la fonctionnalité, l'accessibilité et la performance de l'application Web.

2.18 Les travaux de Sebastien Elbaum

Elbaum et al. [ELB 03, ELB 05] ont examiné les applications Web d'un point de vue fonctionnel. Ils mentionnent le fait que l'approche qu'ils proposent, basée sur les sessions de données utilisateur «*user data session*», diffère des approches existantes dans le sens où elle capture le comportement des utilisateurs pour produire des cas de test. Selon **Elbaum et al.** [ELB 03, ELB 05], les résultats obtenus suggèrent plusieurs directions pour les travaux futurs : d'une part, la combinaison des méthodes de test traditionnelles semblerait posséder un potentiel non encore exploité entièrement, d'autre part, de nouveaux moyens doivent être utilisés pour intégrer avec succès ces approches. Leur approche n'a été supportée par aucun outil.

3 CONCLUSION

Les méthodes de test des applications Web se divisent principalement en deux catégories : les méthodes qui focalisent sur l'aspect structurel, en utilisant plusieurs stratégies telles que la spécification formelle pour modéliser l'application Web et générer les cas de test possibles, et d'autres méthodes qui considèrent l'aspect fonctionnel. Le tableau 3 donne une synthèse des principaux travaux cités dans ce mémoire. Cette synthèse donne une classification chronologique des travaux, d'une part, et une classification selon les aspects considérés par les chercheurs, d'autre part.

Nous avons considéré, dans le cadre de notre approche, les aspects structurels (et le contrôle qui leur est relié) des applications Web. Les travaux basés sur les aspects fonctionnels ont été moins abordés dans la partie bibliographique. Nous pouvons citer dans ce contexte les travaux de Barbara Gabrys [GAB 01], de Tiziana Margaria [MAR 02a, 02b, 02c], d'Oliver Niese [NIE 02] et de l'organisme Parasoft [PAR 03]. La première vue du tableau 3 montre que l'aspect structurel a été le centre d'intérêt de beaucoup de

chercheurs comme Ji-Tzay Yang et al. [YAN 98, 99a, 99b, 02], Filippo Ricca et al. [RIC 00, 01, 02],...etc.

L'approche adoptée, dans ce travail, se situe dans le côté droit du tableau (aspect structurel : composants et interactions entre composants). Notre travail commence d'abord par le niveau module (ensemble de pages Web) et leurs interconnexions. Nous considérons par la suite le niveau suivant, représentant les pages Web, pour atteindre finalement le niveau script. Le modèle adopté, basé sur les graphes de contrôle, couvre l'ensemble des quatre niveaux constituant l'application Web. Contrairement aux travaux directement reliés, effectués par Ye Wu et al. [WU 02] (Niveau script), Ji-Tzay Yang [YAN 98, 99a, 99b, 02] (niveau module sans toucher au niveau script) et Sebastien Elbaum [ELB 03, 05] (niveau module en évoquant l'importance du contrôle décrit dans les scripts), nous considérons dans le cadre de notre approche plusieurs niveaux.

Notre approche est supportée par un prototype, que nous avons développé, permettant de supporter la génération automatique des séquences de test basées sur le contrôle. Ceci constitue une contribution importante relativement aux travaux de Anneliese A. Andrews [AND 04], de Ji-Tzay Yang [YAN 98, 99a, 99b, 02], de Sebastien Elbaum [ELB 03, 05] et de Jiun Long Huang [HUA 98].

Aspects / Année	Aspects Structurels		Aspects Fonctionnels
	Composants	Interactions entre composants	
1998	[YAN 98], [HUA 98]	[YAN 98], [HUA 98]	
1999	[YAN 99a], [YAN 99b]	[YAN 99a], [YAN 99b]	
2000	[NIE 00], [RIC 00], [KUN 00]		
2001	[CHU 01], [OLI 01a], [OLI 01b], [AND 01], [BRU 01], [RIC 01]		[GAB 01]
2002	[YAN 02], [WU 02], [JIA 02], [RIC 02], [NIE 02], [PRO 02], [SCI 02], [BEN 02]	[YAN 02]	[MAR 02a], [MAR 02b], [MAR 02c], [NIE 02]
2003	[ELB 03], [UWE 03]	[ELB 03]	[PAR 03]
2004	[AND 04]	[AND 04]	
2005	[ELB 05]	[ELB 05]	

Tableau 3 : Synthèse des principaux travaux en termes d'approche.

CHAPITRE

3

APPROCHE PROPOSÉE

1 INTRODUCTION

Le besoin de mettre au point des principes et méthodes spécifiques au développement de ce type d'applications, d'une manière générale, et à l'assurance de leur qualité, en particulier, nous a poussés à nous pencher sur la question du test des applications Web. L'étude bibliographique entreprise, la synthèse des principaux travaux effectués ainsi que leur analyse nous a conduits à développer une approche originale qui tient compte de plusieurs paramètres importants (littérature) et qui complète partiellement les travaux effectués dans ce domaine. C'est dans un objectif d'assurance qualité que s'insère notre travail.

2 OBJECTIFS

L'objectif de cette approche est de supporter le test d'une application Web en tenant compte du contrôle. Il s'agit de couvrir, à l'aide d'une approche formelle de description du contrôle (grammaire et expression régulières), permettant de générer les séquences de test,

le plus grand nombre de chemin d'exécution des composants d'une application. La stratégie de test proposée est fondée sur les interactions dynamiques entre les différents composants d'une application Web.

3 PRINCIPES

La méthodologie de test (génération et vérification) que nous proposons est basée sur les interactions dynamiques entre les différents composants d'une application et tient compte de plusieurs aspects reliés à leur contrôle. Nous avons adopté, pour la modélisation, une approche qui subdivise le contrôle en quatre niveaux : le niveau **application**, le niveau **module**, le niveau **page Web** et finalement le niveau **script** (voir tableau 4). Ces niveaux suivent un classement hiérarchique permettant la construction d'un graphe de contrôle complet. Le concept de contrôle s'avère, grâce à notre approche, un moyen efficace pour générer tous les chemins (séquences de test) réels dans une application Web. Ces dernières correspondent, en fait, aux différents scénarios de navigation, spécifiés durant les phases d'analyse et de conception. Elles permettent de vérifier, pour chaque scénario, la conformité de son implémentation à sa spécification. Nous présentons avec cette approche le prototype que nous avons développé permettant la génération de l'arbre complet des chemins de navigation (basée sur le contrôle) à partir duquel, nous générons d'une façon automatique tous les chemins (séquences de test) pouvant exister lors d'une navigation dans une application Web.

Niveaux	Composants
Architecturale	Navigateur (client) / Serveur / BD
Application	Modules M_1, M_2, M_3, ...M_i
Module	Pages p_1, p_2, p_3, ...p_j
Page Web	Scripts, HTML
Script	Graphe de contrôle complet intégrant toutes les structures de contrôle.

Tableau 4 : Les différents niveaux de contrôle dans une application Web.

4 MÉTHODOLOGIE DE L'APPROCHE PROPOSÉE

4.1 Approche proposée

La méthodologie que nous proposons (figure 2) est organisée en deux étapes principales. La première étape porte sur la génération des séquences de test. Chaque séquence générée correspond à un scénario particulier de navigation dans l'application Web. La technique adoptée utilise un modèle intégrant quatre niveaux de contrôle (tableau 4): application, module, page Web, et script. Nous avons utilisé le concept de graphe de contrôle pour modéliser le comportement et le transfert de contrôle entre les différents composants de chaque niveau. Cette structuration nous permet de mieux mettre en évidence le contrôle dans une application Web : inter-niveau et intra-niveau. Par ailleurs, la technique de génération de séquences de test s'appuie sur une description formelle du contrôle relatif aux différentes interactions entre les composants. La seconde étape consiste, quant à elle, à supporter automatiquement le processus de vérification de l'exécution des séquences de test retenues. Cette étape s'appuie sur une instrumentation automatique du code source de l'application à tester. Les séquences générées, sous forme compactées, tiennent compte à la fois de l'interaction entre les différents composants et du contrôle relié à ces interactions. Cette étape nous permet également de calculer différentes couvertures de test.

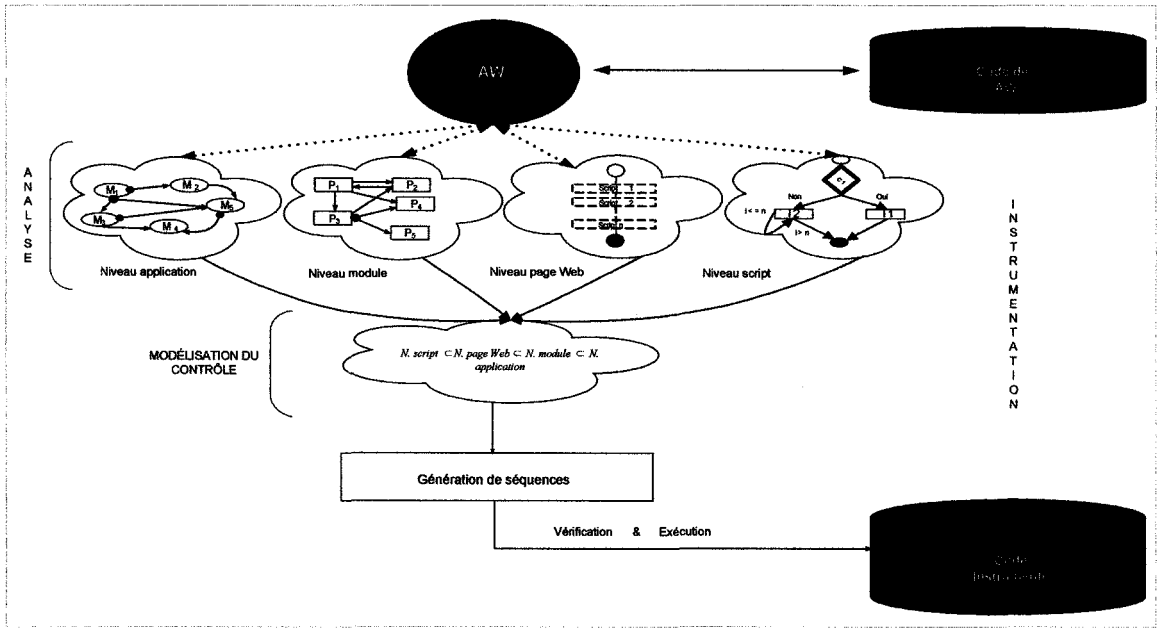


Figure 2 : Méthodologie du processus de test.

4.2 Modélisation du contrôle

La décomposition adoptée pour la modélisation du contrôle dans une application Web permet une structuration en plusieurs niveaux : application, module, page Web, et script. Le noyau du contrôle dans l'application est surtout décrit dans les scripts. La description formelle du contrôle focalise essentiellement sur ce niveau. Nous utilisons le concept d'expression régulière pour décrire les structures suivantes : la séquence, la sélection, l'itération et l'agrégation. Ce principe a déjà été utilisé par Wu et al. dans [WU 02]. Considérons *p* une composante élémentaire constituée d'une ou de plusieurs instructions s'exécutant dans le même bloc de façon séquentielle (sans possibilité de rupture du contrôle). Les différentes structures de contrôle retenues sont décrites de la façon suivante :

Séquence : $(p \rightarrow p_1, p_2, ..., p_i \ ; \ i = 1..n)$: elle exprime la succession dans l'exécution des p_i.

Sélection : $(p \rightarrow p_1 \mid p_2 \mid ... \mid p_i \ ; \ i = 1..n)$: elle exprime l'exclusion dans l'exécution des p_i.

Itération : $(p \rightarrow q^{*/+})$: elle exprime l'itération dans l'exécution de q ($0..n$ fois quand il s'agit du symbole *, et $1..n$ fois quand il s'agit du symbole +).

Agrégation : $(p \rightarrow p_1 \{p_2\})$: p_2 est inclus dans p_1.

Les différents cas de figure que nous pouvons avoir au niveau des scripts et leurs représentations formelles sont décrits par le tableau 5.

Script	Expression régulière correspondante

If (C1) Then {
I1;
Else
I2;
End If ;
	Exp = I1 | I2

Cpt = 0;
While (cpt<n)
I1;
Cp t = Cpt + 1
En while;
⇔
For Cpt = 0 to n
I1;
End for;
	Exp = I1*

Case (X) Case 1 ; I1 ; Case 2 ; I2 ; Case 3 ; I3 ; Break; End case;	Exp = I1 | I2 | I3

Tableau 5 : Les expressions régulières associées aux différentes structures de contrôle.

5 GÉNÉRATION DES SÉQUENCES DE TEST

La technique de génération des séquences de test proposée prend en compte les différents aspects du contrôle (script, interactions entre les pages Web d'un même module).

L'objectif principal étant la génération d'un ensemble de chemins théoriques couvrant l'ensemble des scénarios d'exécution et tenant compte de la nature des interactions entre pages Web (conditionnelle, inconditionnelle, itérative,...etc.). Chaque chemin correspond à une exécution particulière de l'application Web et fera l'objet d'une séquence de test particulière. Les cas de test sont sélectionnés sur la base de critères de traversée des chemins. La génération des séquences de test est organisée en trois étapes : (1) Analyse du code et construction du graphe de contrôle, (2) Construction de l'arbre des pages et (3) Génération des séquences de test.

5.1 Construction du graphe de contrôle

L'objectif à cette étape consiste à effectuer une analyse statique automatique du code permettant la construction des graphes de contrôle. Les graphes de contrôle ont été largement utilisés dans plusieurs techniques classiques de test structurel. Par ailleurs, ils procurent une vue globale et synthétique du contrôle dans une application Web. La construction du graphe de contrôle débute réellement par le plus bas niveau (niveau script). La figure 3 présente un exemple de graphe de contrôle correspondant à une application réelle. Cette étude de cas sera illustrée ultérieurement.

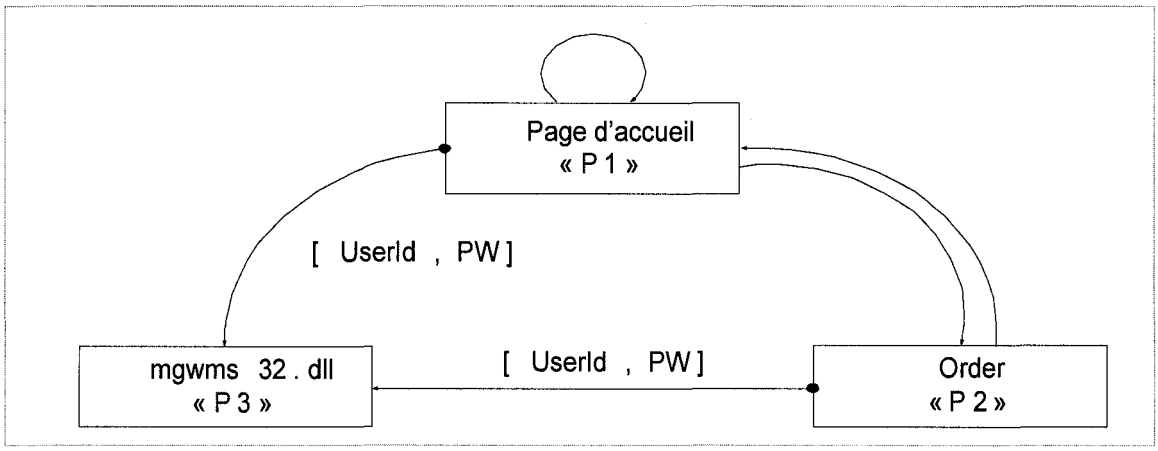


Figure 3 : Graphe de contrôle de l'application *Pizzahut.*

5.1.1 Niveau module

Au niveau module, la composante principale est la page Web. Il s'agit d'un graphe de contrôle inter-pages. Les nœuds de ce graphe représentent les noms des pages Web. Les arcs représentent le transfert de contrôle entre les pages (figure 4). Les conditions de transfert de contrôle ne sont pas indiquées, lors de cette étape, de façon explicite. Elles seront reportées ultérieurement (suite du processus d'analyse statique).

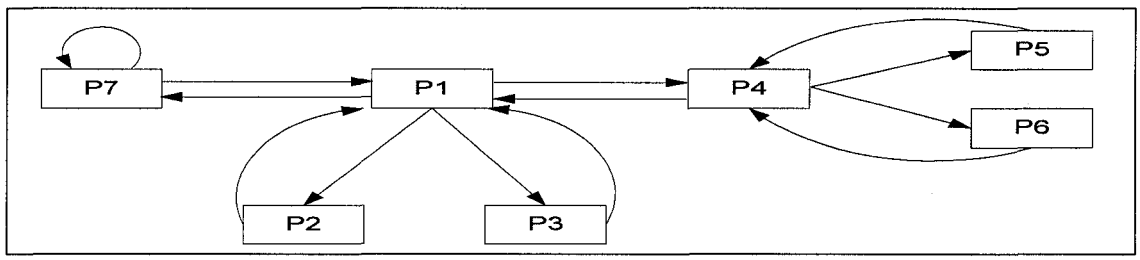
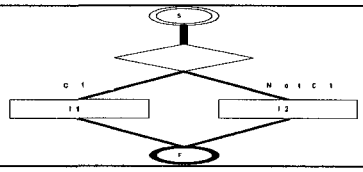


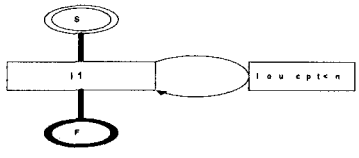
Figure 4 : Exemple de graphe inter-pages dans un module.

5.1.2 Niveau script

Durant cette étape, l'analyse s'effectue au niveau du script. Nous associons à chaque script son propre graphe de contrôle illustrant la (ou les) structure (s) de contrôle qu'il contient. Le concept est illustré par le tableau suivant (Tableau 6).

Script	Graphe de contrôle correspondant

If (C1) Then {
I1;
Else
I2;
End If ;
	

Cpt = 0;
While (Cpt < n)
I1;
Cpt = Cpt + 1
En while;
⇔
For Cpt = 0 to n
I1;
End for;
	

Case (X)
Case 1 ;
I1 ;
Case 2 ;
I2 ;
Case 3 ;
I3 ;
Break;
End case;

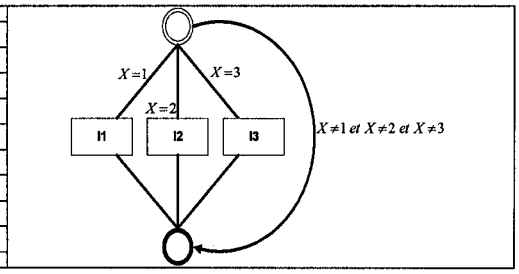


Tableau 6 : Exemple de script et du graphe de contrôle correspondant.

Après avoir construit les graphes de contrôle au niveau script, nous étendons les graphes de contrôle inter-pages en y intégrant les conditions extraites des scripts. L'intégration est réalisée d'une façon automatique. La figure 5 précédente donne un graphe de contrôle inter-pages complété par les conditions de transfert de contrôle. Les conditions de transfert de contrôle inter-pages sont indiquées sur les arcs correspondants. Ce type d'information permet de renseigner sur la nature des différents chemins de contrôle entre les pages d'un module.

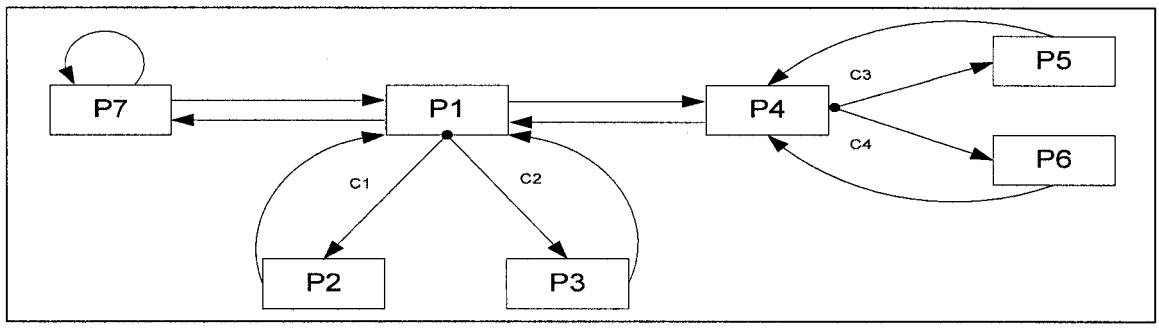


Figure 5 : Graphe de contrôle inter-pages étendu aux conditions de transfert de contrôle.

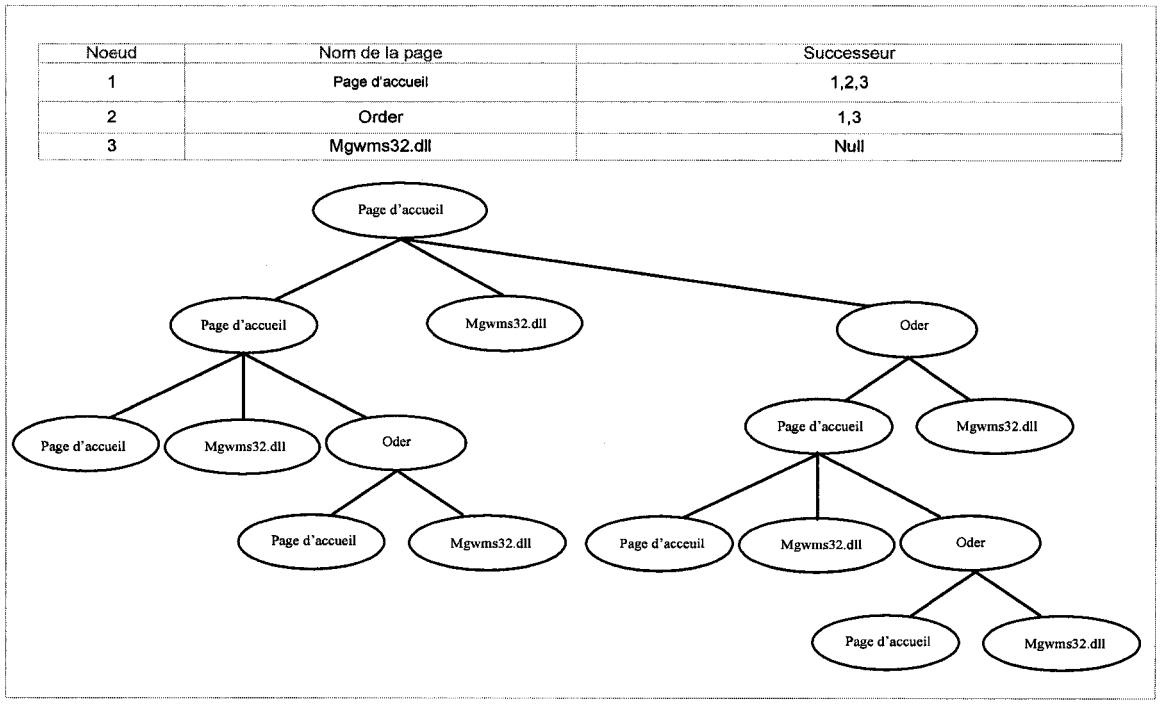
5.2 Arbre des pages

L'objectif à ce stade consiste à effectuer, dans un premier temps, une analyse du graphe de contrôle des pages constituant l'application Web pour extraire une synthèse du

comportement (contrôle) et la décrire de façon formelle. La description formelle des différentes sections de contrôle est réalisée à l'aide des expressions régulières. La génération des expressions régulières correspondantes est effectuée de façon automatique. La figure 6 donne les expressions régulières correspondant au graphe de contrôle de la figure 4. Par la suite, nous procédons à la construction de l'arbre complet des pages (figure 7). Le point d'entrée de l'arbre sera représenté par le nom de la page de démarrage déclenchant la première exécution de l'application Web. Le processus de construction est incrémental. L'objectif étant d'intégrer tous les niveaux modélisés de façon à générer automatiquement l'arbre complet de l'application. Ce dernier est décrit de façon formelle.

$$G_m = \left\{ \begin{array}{lll} P_1 & \rightarrow & P_1 \mid P_2 \mid P_3 \mid \bullet \\ P_2 & \rightarrow & P_1 \mid P_3 \mid \bullet \\ P_3 & \rightarrow & \bullet \end{array} \right\}$$

Figure 6 : Grammaire décrivant le graphe de contrôle de la figure 2.



Noeud	Nom de la page	Successeur
1	Page d'accueil	1,2,3
2	Order	1,3
3	Mgwms32.dll	Null

Figure 7 : Arbre des pages.

5.3 Séquences de test

L'objectif à cette étape consiste à générer, à partir de l'arbre des pages, l'ensemble des chemins théoriques (potentiels de navigation) en tenant compte des conditions relatives aux interactions (conditionnelles, inconditionnelles, itérations…etc.). Une séquence de test correspondra à chaque chemin généré. Les séquences de test générées (figure 7) à partir de l'arbre des pages couvrent l'ensemble des chemins théoriques, de la racine jusqu'aux feuilles. À ce stade, l'objectif est d'identifier parmi l'ensemble des chemins théoriques possibles T, correspondant à un module de l'application Web, un ensemble de chemins exécutables T_R , par application de règles de réduction. L'ensemble exécutable T_R des chemins obtenus après réduction sera utilisé pour la génération de l'ensemble S des séquences de test. À chaque chemin de l'ensemble T_R correspondra une séquence de test

de l'ensemble S, représentant en fait un scénario particulier de navigation dans l'application Web. Les règles de réduction retenues sont :

- ***Chemins infaisables*** : l'analyse des prédicats de parcours des chemins théoriques permet de déterminer les chemins infaisables (prédicats contradictoires). Ces chemins sont réduits de l'ensemble T.

- ***Réduction des cycles*** : les graphes de contrôle présentent souvent des parties itératives dans les interactions entre les pages Web pouvant engendrer un nombre important (voir infini) de chemins (théoriques). Pour des raisons évidentes, nous ne pouvons tester tous ces chemins. Nous faisons l'hypothèse que ces chemins constituent une famille de chemins similaires (du moins la partie itérative) et tester un seul chemin (une seule itération) devrait suffire. Cette hypothèse nous permet de réduire considérablement le nombre de chemins potentiels.

Page d'accueil, Page d'accueil, Page d'accueil
Page d'accueil, Page d'accueil, Mgwms32.dll
Page d'accueil, Page d'accueil, Order, Page d'accueil
Page d'accueil, Page d'accueil, Order, Mgwms32.dll
Page d'accueil, Mgwms32.dll
Page d'accueil, Order, Page d'accueil, Page d'accueil
Page d'accueil, Order, Page d'accueil, Mgwms32.dll
Page d'accueil, Order, Page d'accueil, Order, Page d'accueil,
Page d'accueil, Order, Page d'accueil, Order, Mgwms32.dll
Page d'accueil, Order, Mgwms32.dll

Figure 8 : Séquences de test pour l'application *Pizzahut.*

6 PROCESSUS DE VÉRIFICATION

6.1 Test d'une page Web

La procédure de test d'une page Web est organisée en plusieurs étapes (figure 9). L'objectif est d'exécuter, pour chaque page Web, au moins une fois chaque séquence

retenue. Il s'agit également de vérifier que la séquence exécutée correspond à celle prévue, selon les données considérées.

```
Pour chaque page Web décrite par le graphe de contrôle
  {
     Génération des séquences de test correspondantes
     Pour chaque séquence S,
      {
         Exécution du programme
         Vérification du chemin (séquence de liens) exécuté
      }
  }
```

Figure 9 : Test d'une page Web.

Le processus de vérification repose sur une instrumentation automatique du programme. Cette dernière consiste à ajouter au programme sous test des instructions particulières permettant de récupérer plusieurs informations lors de son exécution (support à l'analyse dynamique). L'analyse des traces d'exécution permet de vérifier si la séquence de test a été exécutée conformément à ce qui a été spécifié. Toute déviation des résultats obtenus par rapport à ceux attendus sera considérée comme défaillance. L'utilisateur intervient, bien entendu, durant ce processus pour vérifier les résultats de l'exécution. Les principales phases du processus de vérification sont : (1) Instrumenter le programme sous test, (2) Exécuter le programme sous test et (3) Analyser les résultats obtenus.

6.2 Critères et couverture de test

Les séquences de test sont exécutées de façon incrémentale. Ce processus permet, par analyse des différentes traces d'exécution, de déterminer les séquences exécutées (couverture des séquences : scénario principal et ses alternatives) et, par conséquent, les séquences qui restent à exécuter. Nous avons défini deux types de critères de test ainsi que les couvertures correspondantes.

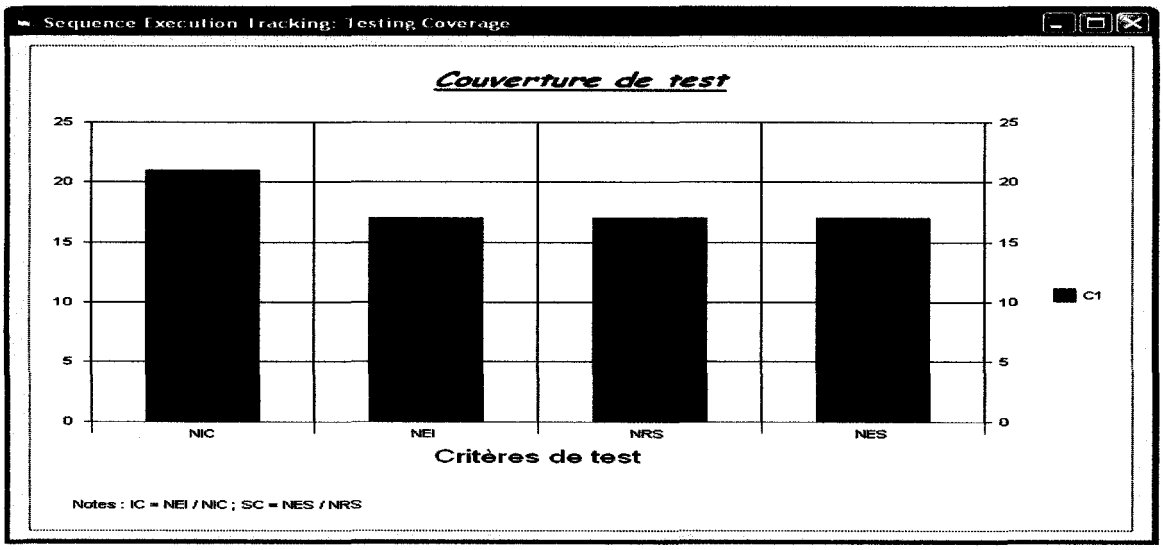


Figure 10 : Analyse des traces d'exécution : Couverture de test

- ***Interaction entre les pages*** : Chaque interaction dans le graphe de contrôle doit être exécutée au moins une fois. La couverture des interactions (IC) est définie comme le rapport du nombre des interactions exécutées (NEI) au nombre total des interactions dans le graphe (NIC). Soit IC=NEI/NIC.

- ***Séquences de test*** : Chaque séquence retenue doit être exécutée au moins une fois. Chaque séquence correspond à un scénario implémenté. La couverture des séquences (SC) est définie comme le rapport du nombre de séquences exécutées (NEC) au nombre de séquences retenues (NRS). Soit SC= NEC/NRS.

7 CONCLUSION

La technique de génération de séquences de test proposée dans ce chapitre (Figure 2) est basée sur une description formelle du comportement collectif des groupes de composants. Ceci donne une base solide au processus de vérification. Cette technique permet, pour chaque composant, à partir d'une analyse statique de la description formelle du graphe de contrôle correspondant, la génération d'un ensemble de séquences

de tests appropriés. Ces dernières prennent en compte non seulement les interactions entre les composants mais aussi les différents aspects relatifs à leurs contrôles. Chaque séquence correspond à un scénario particulier d'un composant.

CHAPITRE

4

PRÉSENTATION DE L'OUTIL ET ÉVALUATION DE L'APPROCHE

1 ENVIRONNEMENT DE TEST

La technique proposée dans ce mémoire est supportée par un environnement composé de plusieurs outils (Figure 11). Le processus de génération des séquences de test est basé essentiellement sur un analyseur, un traducteur et un générateur. L'objectif de l'analyseur consiste à analyser le code HTML et construire le graphe de contrôle. Le traducteur, quant à lui, permet de décrire de façon formelle le graphe de contrôle et construire l'arbre des pages correspondant. Enfin, le générateur, grâce à l'arbre des pages, produit toutes les séquences de test possibles. Ces séquences correspondent aux principaux scénarios de navigation dans l'application.

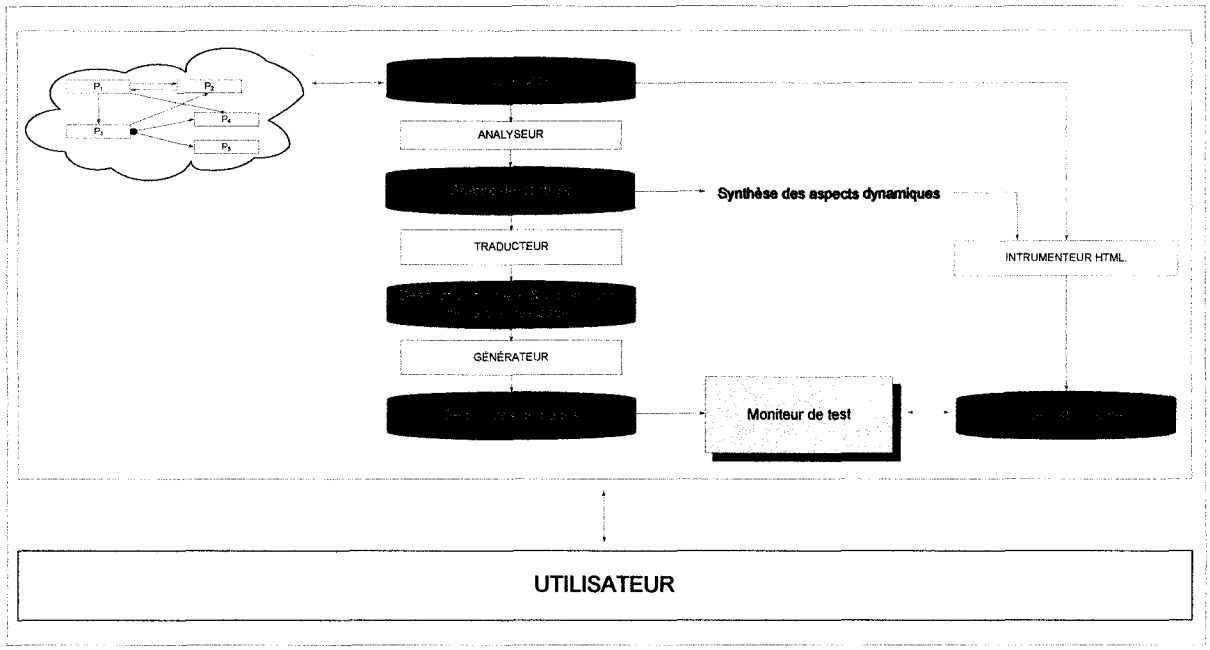


Figure 11 : Architecture de l'environnement.

2 PRÉSENTATION DE L'OUTIL TEST _AW

2.1 Introduction

L'objectif dans ce contexte est de mettre à la disposition des développeurs des applications Web un outil logiciel homogène qui assiste les développeurs à mieux tester leurs produits. La stratégie de test sous-jacente est organisée en trois phases distinctes que nous décrivons ci-dessous.

2.1.1 Première phase : génération des composants HTML

Une requête ou une adresse URL initiale est soumise à notre outil. Cette requête devrait designer une application Web que l'utilisateur veut tester. Par la suite, l'utilisateur aura

l'opportunité de reconsidérer sa requête (nouvelle adresse URL) à la lumière des résultats de recherche obtenus grâce à la requête initiale (voir figure 12).

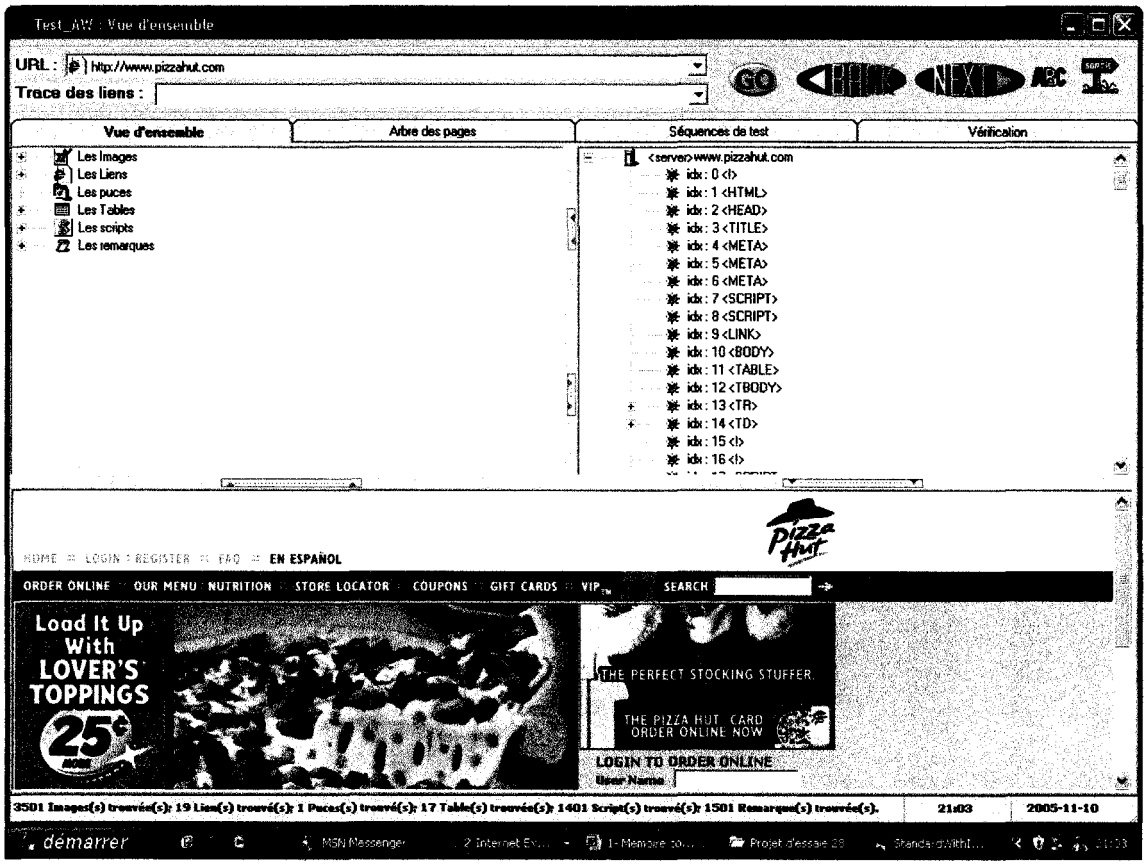


Figure 12 : Test_AW : génération des composants HTML.

2.1.2 Seconde phase : génération de l'arbre des pages

Avant de procéder à la génération de l'arbre des pages, l'utilisateur est invité à introduire le niveau de profondeur de l'arbre. La racine de cet arbre correspond à la page d'accueil ou l'adresse URL de l'application. Les nœuds qui suivent la racine sont les pages qu'un utilisateur peut visiter.

Par la suite, l'utilisateur aura l'opportunité de développer encore plus les nœuds et les sous nœuds dans l'arbre grâce au contrôle Activex utilisé avec le contrôle *TreeView* (voir figure 13).

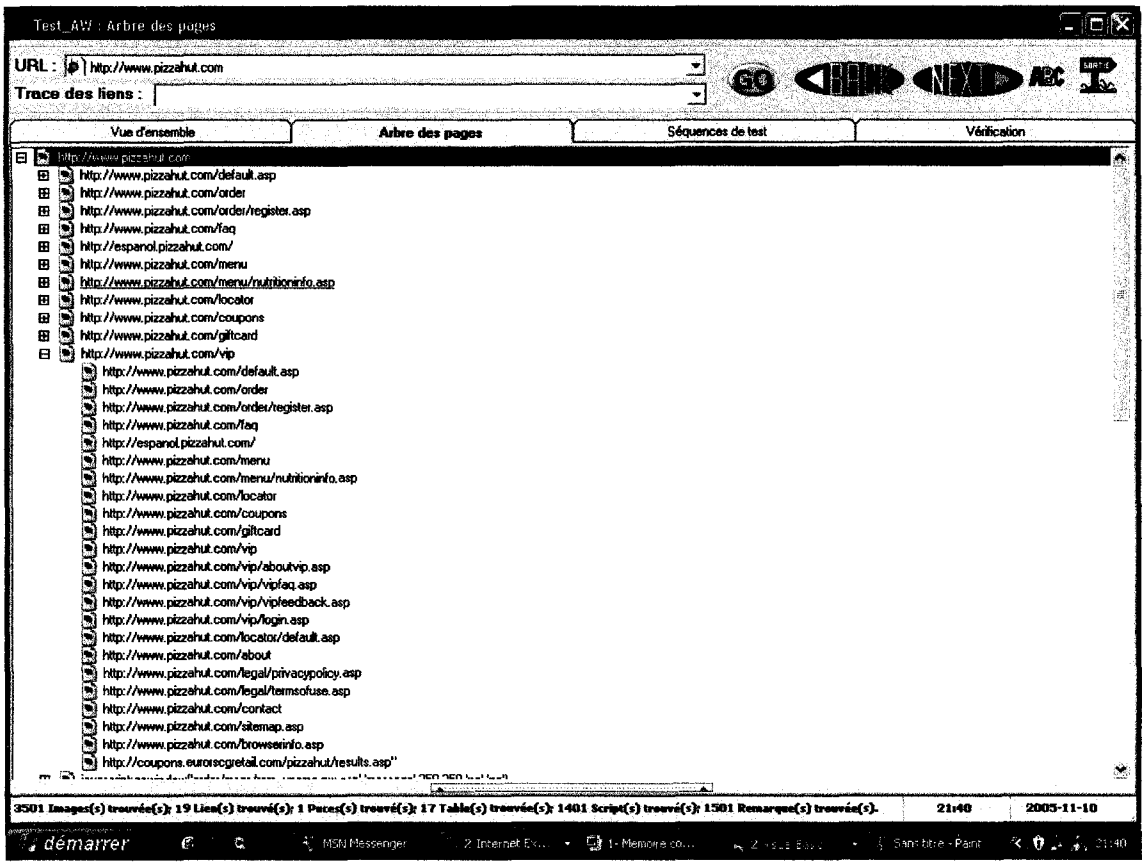


Figure 13 : Test_AW : génération de l'arbre des pages

2.1.3 Troisième phase : Génération des séquences de test

Durant cette phase, l'utilisateur peut générer les séquences de test permettant de supporter les différents scénarios de navigation. Le processus de génération des séquences est basé, entre autres, sur un algorithme de parcourt de l'arbre en profondeur (Figure 14). Chaque séquence générée correspond à un chemin dans l'arbre.

Figure 14 : Test_AW : génération des séquences de test.

2.2 Évaluation

Nous présentons dans cette section la logique de conception de l'outil *TEST_AW*. Cet outil est conçu autour du concept d'expression régulière. Rappelons que la définition d'une expression régulière, utilisée dans notre travail, est la suivante : les **expressions régulières**, aussi appelées **expressions rationnelles**, traductions controversées de l'anglais « *regular expression* » (abrégé *regexp*), sont une famille de notations compactes et puissantes pour décrire certains ensembles de chaînes de caractères. Ces notations sont utilisées par plusieurs éditeurs de texte et utilitaires (particulièrement sous Unix), par exemple vim, Emacs, sed et awk, pour parcourir de façon automatique des

textes à la recherche de morceaux ayant certaines formes, et éventuellement remplacer ces morceaux de texte par d'autres.

L'aspect le plus important de cette partie de notre travail réside dans le support automatique qu'offre notre outil. L'usager peut très facilement utiliser toutes les fonctionnalités offertes par *TEST_AW*. L'utilisateur n'a qu'à saisir l'adresse URL qui désigne l'application Web, et *TEST_AW* s'occupe du reste du travail (support des différentes phases spécifiées dans les sections précédentes). En réalité, *TEST_AW* intègre un gestionnaire d'expressions régulières. Il permet de coordonner l'utilisation des différentes expressions (spécifiant le contrôle entre pages) prédéfinies statiquement dans le code source de l'outil.

L'outil supporte également le processus de vérification des séquences exécutées. Cette étape est réalisée grâce à une instrumentation automatique du code de l'application. Elle a pour but d'insérer des instructions particulières permettant de capturer, lors de l'exécution de l'application, des informations permettant d'effectuer une analyse dynamique de l'application. L'emphase est mise, à ce stade, sur l'aide apportée à l'utilisateur lors de la phase de vérification de l'exécution des séquences. *TEST_AW* se charge de calculer, par analyse dynamique, les différentes couvertures de test retenues dans notre approche.

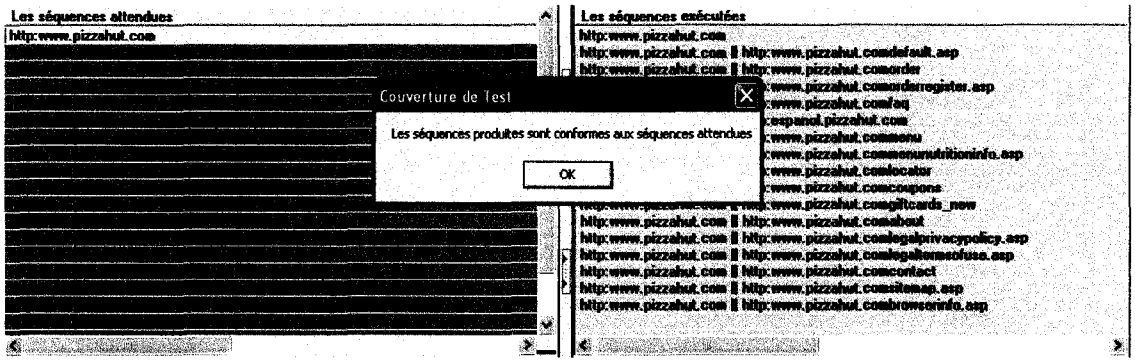
2.2.1 Test d'une page Web

La procédure de test d'une page Web est organisée selon les étapes indiquées dans la figure (9) du chapitre précédent. Nous supposons que chaque page Web est décrite par un graphe de contrôle (extrait automatiquement par analyse du code). L'objectif est d'exécuter, pour chaque page Web, au moins une fois les séquences retenues et de vérifier leur conformité à la spécification. Il s'agit également de vérifier que la séquence de test exécutée durant le test correspond à celle attendue, selon les données considérées.

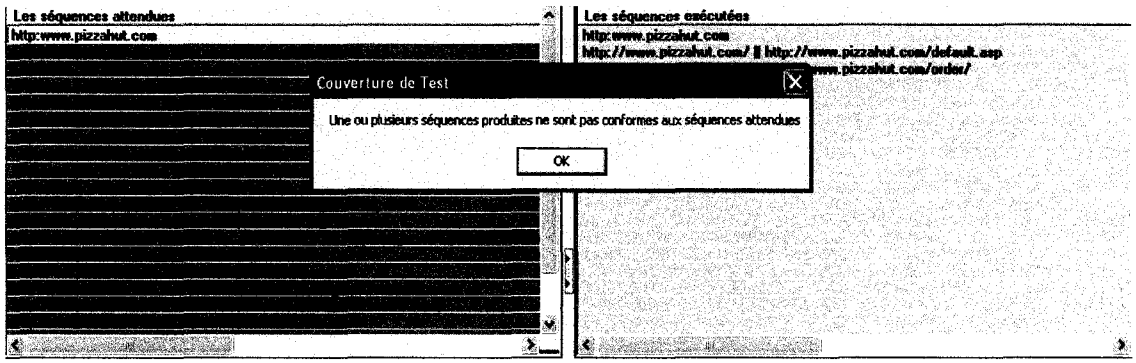
Ce processus permet essentiellement de vérifier si les séquences exécutées sont conformes à celles générées, et si les résultats obtenus sont conformes aux résultats attendus. Ce processus de vérification repose, comme mentionné précédemment, sur une instrumentation du programme qui consiste à ajouter au programme sous test des instructions particulières permettant de récupérer plusieurs informations lors de son exécution. L'analyse des traces d'exécution permet de vérifier si la séquence de test a été exécutée conformément à ce qui a été spécifié. Toute déviation des résultats obtenus par rapport à ceux attendus (en termes de séquence exécutée en particulier) sera considérée comme une défaillance. L'utilisateur intervient, bien entendu, durant ce processus pour vérifier les résultats spécifiques de l'exécution. Les principales phases du processus de vérification sont : (1) Instrumenter le programme sous test, (2) Exécuter le programme sous test et (3) Analyser les résultats obtenus.

2.2.2 Analyse de l'exécution des séquences

La figure 15 illustre le résultat du processus de vérification durant l'exécution des séquences. La figure (15 – Cas 1) présente le cas où l'environnement indique au testeur que l'ensemble des séquences exécutées est conforme à celui des séquences attendues. La figure (15 - Cas 2) présente un cas particulier où la séquence de test est non valide. Le testeur, dans de tels cas, sera automatiquement informé au sujet de la défaillance détectée. Par ailleurs, sachant qu'il y aura une exception de levée, le testeur saura exactement la partie de la séquence qui a provoqué la défaillance (écart). L'environnement détecte également le cas où la séquence exécutée n'est pas conforme à la séquence attendue. L'environnement arrêtera l'exécution dès la détection d'une déviation de la séquence exécutée relativement à celle attendue. Ceci sera intéressant pour le processus de mise au point. Les séquences exécutées durant le test sont comparées à celles attendues conformément à la spécification décrite soit au niveau du graphe de contrôle ou bien au niveau de l'arbre complet des pages correspondant.



Cas 1



Cas 2

Figure 15 : Processus de vérification

3 CONCLUSION

Il est clair que les développeurs apprécient l'existence d'une aide durant les principales phases de test des produits qu'ils développent. Les applications Web n'échappent pas à cette règle. L'outil développé va dans ce sens. Par ailleurs, l'utilisation de l'arbre dynamique pour formuler les liens selon une structuration bien adéquate permet d'inspirer les usagers à la recherche de sous liens dans les liens pour supporter l'envergure du test de l'application Web en question. Même si la méthode est aléatoire (en termes de choix de séquence), l'exploration des liens permet d'augmenter les résultats fournis par le générateur de séquences. Pour ce qui est de la vérification, elle permet à l'utilisateur d'une part d'exécuter son application en regroupant les séquences selon le niveau choisi, d'autre

part, de lancer le vérificateur automatique qui tend à comparer ce qui a été exécuté avec ce qui a été généré automatiquement.

Les travaux effectués représentent une contribution intéressante dans le domaine du test des applications Web et apportent des éléments complémentaires aux différentes approches citées dans le chapitre 2.

CHAPITRE

5

ÉPILOGUE

1 INTRODUCTION

La croissance remarquable et rapide dans le développement des applications Web est aujourd'hui telle que seule une proportion extrêmement réduite de ces applications peut être effectivement testée [PAR 03] avant d'être mise sur le net. La mise en place de techniques de test automatique, permettant en particulier de mettre en valeur de façon plus efficace ce type d'applications, correspond donc, non seulement à un défi scientifique et technique passionnant, mais également à un véritable enjeu économique, particulièrement crucial dans des domaines comme le commerce électronique, l'éducation,...etc.

En observant le mouvement expansionniste du Web, la communauté scientifique œuvrant dans le développement d'outils de test spécifiques aux applications Web s'est tournée vers trois approches prometteuses : l'approche de test à boite noire, à boite grise et celle à boite blanche (fonctionnelle, mixte et structurelle). L'outil que nous avons présenté s'inscrit dans la troisième approche. Il permet aux développeurs de tester leurs applications avant de la mettre en marche sur le marché d'exploitation. Le système *Test_AW* fournit donc une assistance, à différents niveaux, aux développeurs durant le processus de test. Grâce à l'automatisation totale du système *Test_AW*, les développeurs ne sont pas invités à

paramétrer quoi que ce soit dans le système. Les opérations d'analyse et de génération sont automatiques.

2 AMÉLIORATION ET AVENUES FUTURES

Plusieurs aspects de *Test_AW* pourraient faire l'objet d'améliorations. L'aspect temporel est évidemment une amélioration pour en généraliser l'utilisation. Ainsi, en téléchargeant le code source de chaque page Web, dont le nombre peut aller à des dizaines de pages, il serait pertinent de se pencher sur cette question pour réduire au maximum le temps des traitements. Par ailleurs, l'espace de stockage peut être un inconvénient si le nombre de pages constituant l'application est assez important. Cet aspect doit être également amélioré.

Plusieurs aspects de la conception actuelle du système pourraient être reconsidérés dans un objectif d'extension et d'amélioration. Par exemple, le langage de conception des applications Web. Nous avons considéré, dans la version actuelle de l'outil, les applications décrites en HTML. Il serait également pertinent de prendre en considération d'autres langages tel que XML ou autre. Nous nous sommes plus concentrés, dans le cadre du présent travail, à l'élaboration d'un outil « noyau » supportant le processus (méthode) préconisé. Dans cette optique, il s'agira de considérer des facteurs importants relatifs à l'hétérogénéité de ce type d'applications (en termes technologiques).

L'amélioration de l'interface graphique générale permettra également d'ajouter un plus à la facilité d'utilisation de l'outil (utilisation et présentation des résultats).

3 CONCLUSION

La méthodologie de test des applications Web, proposée dans ce mémoire, est basée sur les interactions dynamiques entre leurs différents composants. L'approche adoptée tient compte de plusieurs aspects reliés à leur contrôle. Le contrôle est utilisé pour la génération

des séquences de test. Le processus de génération des séquences se base sur un arbre, construit par analyse statique de l'application, représentant les différents chemins de contrôle dans une navigation. Les séquences générées, sous forme compactées, correspondent aux différents scénarios de navigation dans une application Web. Elles sont utilisées pour vérifier, pour chaque scénario, la conformité de son implémentation à sa spécification. Par ailleurs, pour évaluer notre approche, nous avons développé un environnement composé de plusieurs outils. Au stade actuel de nos travaux, nous avons simplement évalué l'approche sur des applications Web simples. Nous prévoyons, dans le future, l'utiliser pour tester des applications complexes.

RÉFÉRENCES

[AND 04] Anneliese A. Andrews et autre. "Testing Web Applications by modeling with FSM's", *SOftware SYstems Modeling,* 2004.

[BEN 02] Michael Benedikt, Juliana Freire et Patrice Godefroid. "VeriWeb: Automatically Testing Dynamic Web Sites", *the 11th International World Wide Web Conference (WWW'2002), Honolulu*, 2002.

[BOE 79] Boehm, B.W., Abts, C., et autre, "Software Cost Estimation with COCOMO II ", *Upper Saddle River, NJ: Prentice Hall*, 2000.

[BRU 01] Achim D. Brucker et autre. "Testing Distributed Component Based Systems Using UML/OCL", *Integrating Diagrammatic and Formal Specification Techniques Workshop at Informatics,* 2001.

[CHU 92] Chung, Chi-Ming , Ming-Chi Lee, "Object Orient Programming Testing Methodology", *Proceeding of The Fourth International Conference on Software Engineering and Knowledge Engineering, IEEE Computer Society press*, 1992

[CHU 01] Suet Chun Lee et Jeff Offutt, "Generating Test Cases for XML-based Web Component Interactions Using Mutation Analysis", *The 12th International Symposium on Software Reliability Engineering (ISSRE'01), Hong Kong, China*, 2001.

[CON 00] Jim Conallen, "Concevoir des applications Web avec UML", *Eyrolles, La Source d'Or, Paris, ISBN 2-212-09172-9,* Septembre 2000.

[ELB 03] Sebastien Elbaum, Srikanth Karre et Gregg Rothernel, "Improving Web Application Testing with User Session Data", *International Conference on Software Engineering archive, The 25th international conference on Software engineering*, 2003.

[ELB 05] Sebastian Elbaum, Gregg Rothermel, Srikanth Karre et Marc Fisher II, "Leveraging User-Session Data to Support Web Application Testing", *IEEE Transactions on Software Engineering,* March 2005.

[FAG 76] Fagan, M. E, "Design And Code Inspections to Reduce Errors in Programm Development", *IBM Systems J.*, 1976.

[FAG 86] Fagan, M. E, "Advances in Software Inspections", *IEEE Transactions On Software Engineering*, 1986.

[FLE 99] Fleming J. "WEB navigation: designing the user experience", *O'Reilly & Associates*, 1999.

[GAB 01] Barbara J Gabrys et Jeremy Dick, "An objectives-driven approach to testing Web applications", *CDS*, 2001.

[HEN 01] Michelle Hentie-Gilibersto, Corinne Rubin, Richard Dupuis et Jean_Paul Saurin, "Stratégie de test e-business", *Catalogage Electre-Bibliographie, Sys-com, Paris, Hermès Science Publications, ISBN 2-7462-0331-6,* Septembre 2001.

[HAY 94] Jane Huffman Hayes, "Testing Object-Oriented Programming Systems (OOPS): A Fault Based Approach", *The Proceeding of the International Symposium on Object Oriented Methodology and Systems (ISOOMS),* 1994.

[HUA 98] Jiun-Long Huang, "An Architecture for Web Application Testing Environment", *Thesis of doctorat, Department of computer science and Information Engineering, National Chiao Tung University, Hasinchu, Taiwan, R.O.C*, 1998.

[IEEE 82] IEEE 729-1983, *Glossary of Software Engineering Terminology*, September 23, 1982.

[JIA 02] Xiaoping Jia et Hongming Liu, "Rigorous and Automatic Testing of Web Applications", *The 6th IASTED International Conference on Software Engineering and Applications (SEA 2002), Cambridge, MA, USA*, November 2002.

[KUN 00] David C. kung, Chien-Hung LiuPei et Hsia, "An object-Oriented Web Test Model for Testing Web Applications", *Computer Software and Applications Conference (COMPSAC)*, 2000.

[MAL 02] Ghazwa Malak et autre. "Evaluation de la Qualité des Applications Web: Etat de l'Art", *XXe Congrès INFORSID, France*, 2002.

[MAR 02a] Tiziana Margaria, Oliver Niese, Bernhard Steffen et Andrei Erochok, "System level Testing of Virtual Switch (Re-) Configuration over IP", *IEEE European Test Workshop (ETW~'02), IEEE Computer Society Press*, 2002.

[MAR 02b] Tiziana margaria, Oliver Niese, et Bernhard Steffen, "A practical Approach for the regression Testing of IP-based Applications", From the book intitled

"IP Applications and Services 2003: A Comprehensive Report", *Int. Engineering Consortium (IEC) MaNS02-J, pp.* 195-208, 2002.

[MAR 02c] Tiziana margaria, Oliver Niese, et Bernhard Steffen, "Demonstration of an Automated Integrated Test Environment for Web-Based Applications", *D. Bosnacki and S. Leue (Eds.): SPIN 2002, LNCS 2318, pp. 250-253, 2002. Springer-Verglas Berlin Heidelberg, Gemany*, 2002.

[NIE 00] Oliver Niese et autre. "An Automated Testing Environment for CTI Systems Using Concepts for Specification and Verification of Workflows", *Annual Review of Communication, Int. Engineering Consortium (IEC) Chicago (USA)*, 2000.

[NIE 01a] Oliver Niese, T. Margaria et autre. "Library-Based Design and Consistency Checking of System-Level Industrial Test Cases", *FASE 2001, Int. Conf. on Fundamental Approaches to Software Engineering, Genoa, Italy. (eds.: H. Hußmann), Lecture Notes in Computer Science (LNCS), Germany*, 2001.

[NIE 01b] Oliver. Niese, T. Margaria et autre. "Automated Regression Testing of CTI-Systems". *The IEEE European Test Workshop (ETW~'01). IEEE Computer Society Press, Stockholm (Sweden)*, 2001.

[NIE 02] Oliver Niese, Tiziana margarita et Bernhard Steffen, "Automated functional testing of Web-based Applications", *The 5th International Conference On Software and Internet Quality Week Europe (QWE~'02), Brussles (Belgium)*, 2002.

[PAR 03] Parasoft Webking, "Verifying Web Application Functionality, Performance, and Accessibility", 2003.

[PRE 04] Roger S. Pressman, "Software Engineering A Practitioner's Approach", *The McGrapplications Web-hill Companies, Inc, Sixth Edition, ISBN 0-07-285318-2*, 2004.

[PRO 02] Peter Provost, "Test-Driven Development In .Net", *Bespoke Technologies, Inc., Denver, CO*, 2002.

[RIC 00] Filippo Ricca et Paolo Tonella, "Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions", *European Conferences on Theory and Practice of Software, ETAPS'2001*, 2001.

[RIC 01] Filippo Ricca, Michele Kirchner et Paolo Tonella, "Analysis and Testing of Web Applications", *International Conference on Software Engineering (ICSE)*, 2001.

[RIC 02] Filippo Ricca, Michele Kirchner et Paolo Tonella. "Dynamic Model Extraction and Statistical Analysis of Web Applications", *The International Workshop on Web Site Evolution, WSE 2002,* Montreal, Canada, 2002.

[ROZ 98] Maurice Rozenberg, "Test logiciel", *Eyrolles, La Source d'Or, ISBN 2-212-09172-9,* Octobre 1998.

[SCI 02] Eugenio Di Sciascio et autre. "AnWeb: a System for Automatic Support to Web Application Verification", *the 14th international conference on Software engineering and knowledge engineering*, 2002.

[SOM 04] Ian Sommerville, "Software Engineering, Seventh Edition", *Pearson Education Limited, ISBN 0-321-21026-3*, England, 2004.

[UWE 03] Jens Uwe Pipka, "Test-Driven Web Application Development in Java", *Lecture Notes in Computer Science (LNCS), Publisher: Springer-Verlag Heidelberg ISSN: 0302-9743, Germany*, 2003.

[WU 02] Ye Wu et Jeff Offutt, "Modeling and Testing Web-based Applications", *GMU ISE Technical ISE-TR-02-08*, 2002.

[YAN 98] Ji-Tzay Yang et autre. "A Tool Set to Support Web Application Testing", *International Computer Symposium (ICS'98)*, Taiwan, 1998.

[YAN 99a] Ji-Tzay Yang et autre. "Constructing Control-Flow Based Testing Tools for Web Application", *The 11th Software Engineering and Knowledge Engineering Conference (SEKE-99)*, 1999.

[YAN 99b] Ji-Tzay Yang et autre. "An Object-Oriented Architecture Supporting Web Application Testing", *Proc. of the 23rd Annual International Computer System and Application Conference (COMPSAC-99)*, October 1999.

[YAN 02] Ji-Tzay Yang et autre. "Constructing an Object-Oriented Architecture for Web Application Testing", *J. Information Science and Engineering,* 2002.

ANNEXE

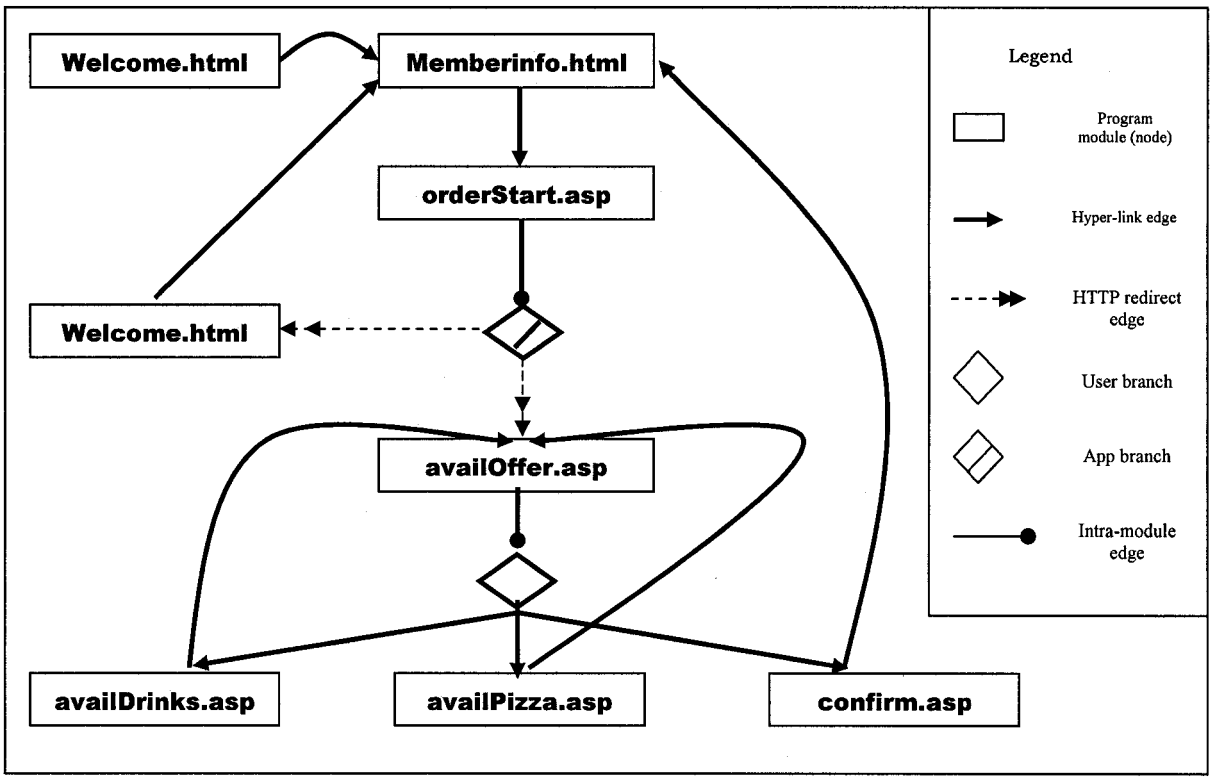


Figure 16 : Un prototype de graphe des flux de contrôle [YAN 98]

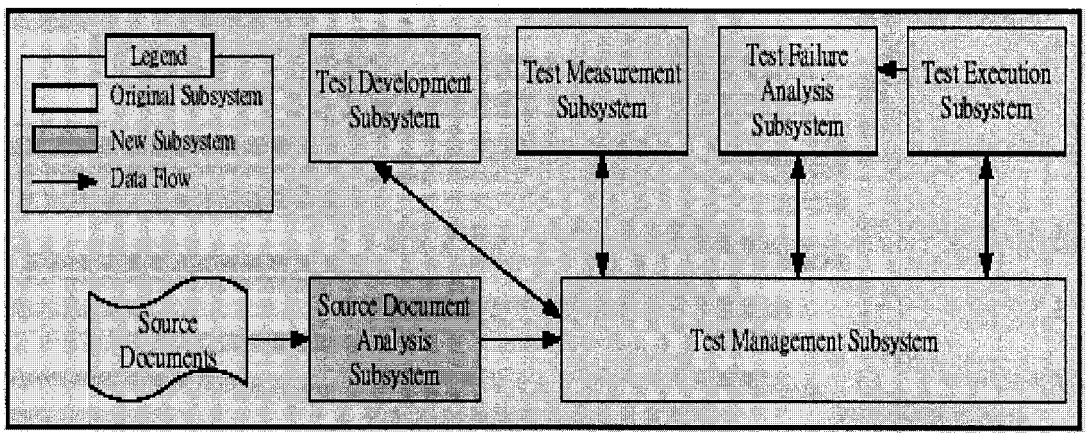


Figure 17 : L'architecture de l'environnement de test de l'application Web [YAN 02]

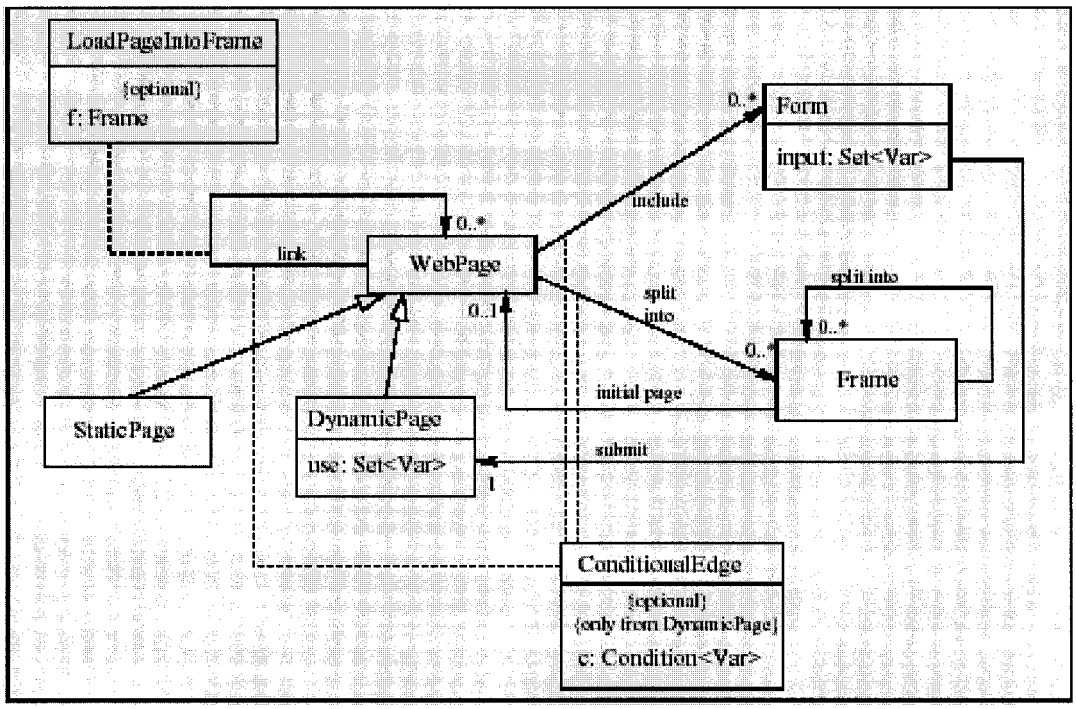


Figure 18 : Meta modèle d'une structure générique de l'application Web [RIC 02]

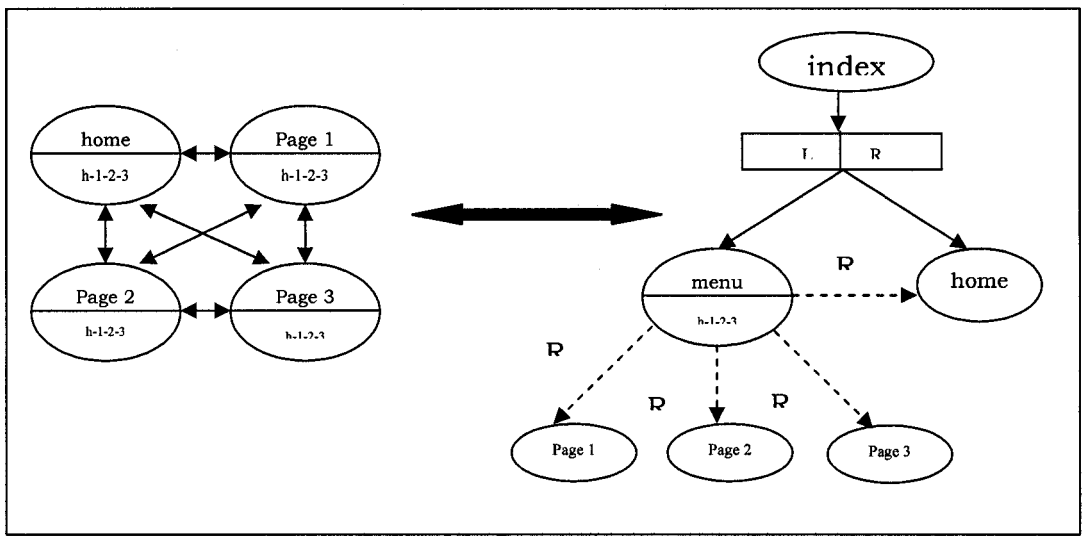


Figure 19 : Un exemple de restructuration de l'application Web [RIC 01]

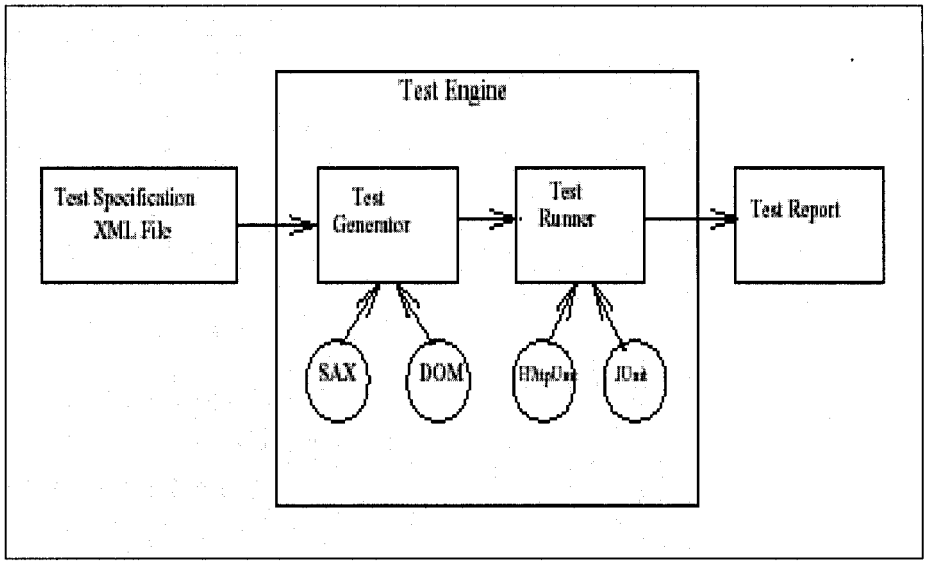


Figure 20 : Le prototype WebTest [CHU 01]

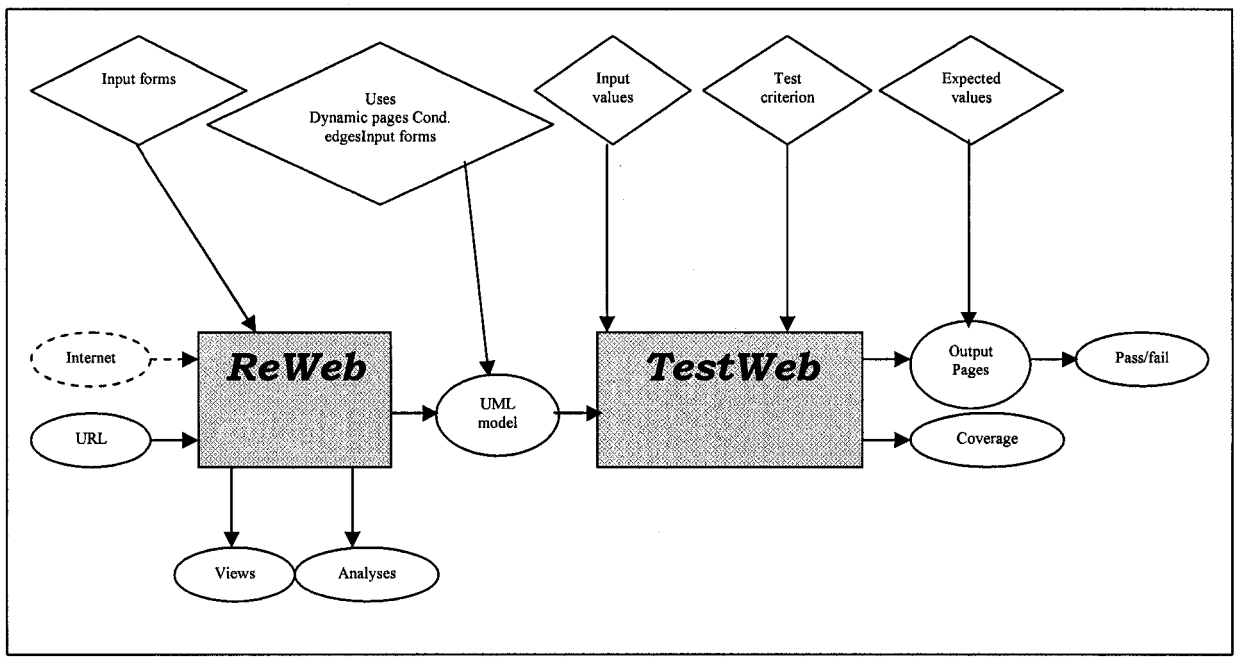


Figure 21 : Rôles de ReWeb et Test Web [RIC 02]