THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Informatique*

Présentée et soutenue par *Joseph Xiong*
Le *29 septembre 2008*

Titre : *Une méthode d'inspection automatique de recommandations ergonomiques tout au long du processus de conception des applications Web*

JURY

G. CALVARY	*Rapporteur*	*Maître de Conférences à l'Université Joseph Fourier, Grenoble 1*
C. FARENC	*Co-Directeur*	*Maître de Conférences à l'Institut Universitaire Technologique de Tarbes*
M. NOIRHOMME-FRAITURE	*Examinateur*	*Professeur aux Facultés Universitaires Notre-Dame de la Paix, Namur, BE*
P. PALANQUE	*Directeur*	*Professeur à l'Université Paul Sabatier, Toulouse*
J.P. PEREZ	*Examinateur*	*Ingénieur Conseil, Société Génigraph, Toulouse*
J. VANDERDONCKT	*Rapporteur*	*Professeur à l'Université Catholique de Louvain, Louvain-la-Neuve, BE*
M. WINCKLER	*Co-Directeur*	*Maître de Conférences à l'Institut Universitaire Technologique de Tarbes*

Ecole doctorale : *MITT Mathématiques Informatique Télécommunications de Toulouse*
Unité de recherche : *IRIT – UMR 5505*
Directeur de Thèse : *P. PALANQUE*

Remerciements

Je tiens tout d'abord à remercier les membres du jury : Jean Vanderdonckt et Gaëlle Calvary qui ont rédigé des rapports très positifs et qui ont manifesté un intérêt réel pour mes travaux : un regard extérieur est toujours très précieux. Un grand merci également à Monique Noirhomme-Fraiture qui m'avait accueilli dans son laboratoire à Namur en janvier 2007 et avec qui j'ai eu grand plaisir à discuter : les conseils qu'elle a pu me donner ont été très bénéfiques pour la fin de ma thèse.

Cette thèse doit beaucoup à l'encadrement dont j'ai pu bénéficier. Du côté de l'IRIT, je pense à Philippe Palanque qui, bien que n'étant pas directement son doctorant, a su me guider chaque fois que je lui ai posé des questions relatives à ma thèse. Je pense notamment au compte-rendu très riche qu'il m'a fourni après les rencontres doctorales à IHM'06. Un grand merci très sincère à Christelle Farenc qui a toujours été là dans les moments de doute et qui a toujours prêté une oreille attentive malgré le peu de temps libre qu'elle pouvait me consacrer. Son côté très humain a grandement participé à établir de bonnes conditions de travail. Aujourd'hui, je me retourne et me dis que je suis fier d'avoir été son étudiant. Mille mercis à Marco Winckler qui a su diriger cette thèse avec tant d'efficacité. Il a su me guider et me pousser, parfois même dans mes derniers retranchements, pour toujours améliorer la qualité de mon travail. Ses qualités professionnelles ne sont plus à prouver et j'ai vraiment été ravi d'avoir pu bénéficier de son enseignement si riche. Sans lui, cette thèse n'aurait pas été possible, et je lui en suis très reconnaissant.

Du côté de l'encadrement à Génigraph, je remercie Jean-Paul Pérez qui a toujours porté un regard très intéressé sur ma thèse. Cet intérêt s'est sans cesse manifesté par sa joie et sa bonne humeur qui rendaient les réunions très agréables. Par ailleurs, les nombreux conseils et encouragements qu'il a pu me donner ont toujours eu un effet positif sur mon travail. Je remercie aussi Olivier Nicolas qui a dirigé l'application de mes travaux sur e-Citiz. Il a su me faire confiance tout au long de cette thèse en m'accordant tout le temps nécessaire à ma recherche mais également, en me donnant assez de libertés sur l'implémentation de mes travaux. Son côté ouvert et humain fait que l'ambiance de travail a toujours été très bonne, non seulement pour moi, mais également pour l'ensemble des personnes de Génigraph Toulouse.

La plus belle des rencontres que j'ai pu faire pendant ma thèse (en réalité, depuis le DEA) a été celle de Florence Pontico, mon éternelle binôme, ma jumelle cosmique ;) Mes souvenirs les plus heureux sont les conférences auxquelles nous avons participé ensemble et où nous discutions longuement de tout et de rien, de la vie en quelque sorte. Ces moments-là restent gravés dans ma mémoire comme des instants impérissables. D'aussi loin que je me souvienne, nous nous sommes toujours aidés et écoutés l'un et l'autre tant sur le plan personnel que professionnel. Je ne pourrais malheureusement jamais exposer en quelques lignes l'aide infinie qu'elle m'a apportée et pour laquelle je lui suis profondément reconnaissant. Je tiens toutefois à la remercier sincèrement, dans le cadre précis de la thèse, pour ses nombreuses relectures et ses conseils avisés sur mon texte. Enfin, même si nous serons amenés à nous revoir souvent (je l'espère), il est vrai que parfois elle manque, comme une sœur manque à son frère.

J'ai été ravi d'avoir pu passer toutes ces années de thèse dans l'équipe LIIHS (aujourd'hui IHCS). Je garde de très bons souvenirs des gens de l'équipe. Je tiens à remercier particulièrement Amélie sans qui je n'aurai pas intégré cette équipe. Ami avec elle bien longtemps avant la thèse, elle a su me guider dans le monde de la recherche avant que je ne vole de mes propres ailes. J'ai beaucoup apprécié les rares discussions que j'ai pu avoir avec Rémi. La clarté de ses propos rend les discussions si simples sur des sujets divers et variés. Un grand merci à Syrine qui a toujours été très agréable à vivre, voire peut-être même trop. Elle a apporté de la gaieté à notre bureau. Et avant de m'éclipser de l'équipe, je lui souhaite une très bonne dernière année de thèse. Je remercie également Guillaume qui m'a aidé énormément pour mon outil. Il a toujours été disponible pour me donner un coup de main et je lui en suis très reconnaissant. Merci à Christophe pour sa joie et sa bonne humeur qui ont agrémenté mes derniers mois de thèse. Le peu de chemin que nous avons fait ensemble me laisse croire qu'il

réalisera un travail de qualité pour sa thèse : bon courage pour la suite ! Je remercie également les autres membres de l'équipe : Xavier, Eric, Sandra, Jeff, Manu et David. Enfin, les pauses café étant des moments inoubliables dans la vie d'un thésard, je tiens à dire que je n'oublierai jamais ces instants passés avec Florence, Syrine et Christophe. Je n'oublie pas non plus Mikaël avec qui j'ai partagé le bureau des DOM-TOM au 1R1 pendant un an.

J'ai également partagé mon temps de travail à Génigraph Toulouse où j'ai pu bénéficier de l'ambiance chaleureuse qui règne dans les locaux de l'agence. J'ai une pensée toute particulière envers Mouhamed Diouf, le « futur premier ministre », le « grand Sage sénégalais » avec qui j'ai eu « l'honneur » de travailler dans le cadre de nos deux thèses. Merci à lui pour ses rires quotidiens si communicatifs et son attitude toujours positive qui ont rendu les bureaux si vivants. Il m'a beaucoup aidé sur des aspects techniques liés à ma thèse et je lui en suis infiniment reconnaissant. J'espère que nos routes se croiseront à nouveau, qu'il parte s'installer à l'autre bout du monde ou non. Un grand merci à Cyril Chéné qui m'a apporté une aide gigantesque pour l'implémentation de mes travaux. Son côté logique, rigoureux et critique font de lui une personne avec qui travailler devient un enseignement quotidien. Ces mêmes qualités font également de lui une personne redoutable à la fois dans le travail mais aussi dans la vie : vous ai-je dit qu'il avait écrit toute l'encyclopédie Wikipédia et qu'il avait réponse à tout ? ;) J'ai également eu grand plaisir à partager avec lui notre amour des quiz, des pendus et de la bière. Je remercie également Richard Fagot, le Ch'ti de Génigraph. Son humour décalé a toujours agrémenté nos journées de travail. Je remercie Laurent Goncalves pour tous les conseils qu'il m'a prodigués et toutes ses méthodes de travail qu'il m'a transmises. J'ai particulièrement apprécié la bonne humeur qu'il affiche au quotidien. Merci également à Martine, Jérôme, Manu, Leonardo, Mickael(s) et Fred qui ont tous participé de près ou de loin à cette grande aventure.

Merci à tous les PEON qui sont pour la plupart des amis du lycée (et même du collège, oui, on s'fait vieux !). C'est en partie grâce à vous si j'ai toujours gardé le moral pendant ces années de thèse. Je remercie tout particulièrement Steph (et son acolyte Loïc) ainsi qu'Olivier qui m'ont soutenu et qui m'ont encouragé pendant l'écriture du manuscrit. J'ai également une pensée pour Clément qui termine également sa thèse et qui m'a toujours encouragé par MSN :)

Mes derniers remerciements vont à Christelle. Merci d'être là, merci pour tout. Je suis tellement heureux de t'avoir rencontré, tu restes la personne la plus formidable à mes yeux.

Table des matières

1. Introduction

De plus en plus d'organismes, d'entreprises, d'institutions, de particuliers, souhaitent disposer d'une vitrine sur le Web afin d'améliorer leur image, d'élargir leur cible, de dématérialiser certains de leurs services (par exemple, déclaration de revenus) et d'en proposer de nouveaux (par exemple, diffusion de conseils personnalisés sur un site d'e-commerce). Les problématiques de conception d'interface sont cruciales dans le Web, et l'utilisabilité des applications Web doit être un objectif central de conception. Une interface de qualité permet à l'utilisateur d'atteindre le but qu'il s'est fixé (par exemple, acheter un livre), avec efficacité et satisfaction. Dans le cas contraire, l'utilisateur est insatisfait, voire indisposé s'il n'a pas réussi à atteindre un but obligatoire, ou s'il a commis malgré lui une action indésirable (par exemple, achat engagé sans l'avoir souhaité). Quoiqu'il en soit, l'image de l'organisme en charge du site Web en pâtit. De nombreux pays ont pris conscience de l'importance de la qualité des applications Web et votent des lois en faveur d'une exigence minimale pour les applications Web à caractère officiel (administrations, services aux citoyens, etc.), notamment en terme d'accessibilité [Thatcher 06]. Ces lois s'accompagnent de recommandations rassemblant un ensemble de connaissances ergonomiques liées au domaine d'application, dont l'application est une obligation légale.

La conception Web est ainsi devenue une discipline avec une forte exigence de qualité, notamment pour ce qui concerne ses interfaces [Rossi 06]. Pour répondre à cette exigence, il est nécessaire de fournir des méthodes et des processus de conception adaptés aux spécificités des applications Web, notamment, du fait que ces applications évoluent rapidement dans le temps (en termes d'actualisation souvent quotidienne du contenu, évolution des services soumis à une nouvelle législation, déploiement sur de nouvelles plateformes comme par exemple les dispositifs mobiles, etc.). Compte tenu de l'évolution constante des applications web, il est nécessaire d'intégrer un suivi de la qualité ergonomique dans le processus de conception.

Pour s'assurer de la qualité ergonomique d'une application Web, de nombreuses solutions ont été développées telles que les démarches de conception centrées utilisateur, les démarches de conception participatives, l'organisation de connaissances ergonomiques, les méthodes d'évaluation [Ivory 01]. Un des avantages de l'utilisation des connaissances ergonomiques est que les règles sont applicables tout au long du processus de conception, pour guider la conception ou pour faire l'inspection des interfaces développées. Toutefois, l'application des règles ergonomiques reste une tâche complexe qui requiert une connaissance et une forte expertise, ce qui nécessite souvent un expert humain, mais qui peut être partiellement automatisé par un outil [Farenc 01]. L'automatisation procure plusieurs avantages :

- Un gain de temps considérable par rapport à une inspection manuelle, augmentant ainsi la fréquence possible des évaluations et en diminuant également le coût ;
- Une plus grande exhaustivité de la vérification des recommandations, diminuant les risques d'oublis ;
- L'accessibilité des procédures d'évaluation à des personnes non expertes en ergonomie puisque la connaissance requise est inscrite dans l'outil ;
- Certaines fonctionnalités de corrections automatiques, allégeant ainsi le travail de réparation nécessaire pour que l'application soit conforme aux recommandations.

Si les avantages de l'inspection automatique sont indéniables, elle reste néanmoins mise en œuvre uniquement en fin de processus de conception, souvent sur une application finale ou un prototype avancé. Cependant, certaines erreurs identifiées sur l'application découlent d'un mauvais choix de conception fait lors d'une des étapes initiales du processus. Une telle erreur pourrait être, par exemple, une structure de navigation (menu) trop profonde (suite à un découpage excessif de l'information en de trop nombreuses pages). Si une telle erreur est

détectée après implémentation, sa correction nécessite de revoir les spécifications et de re-concevoir l'application entière (ou plusieurs modules de l'application). En conséquence, il serait judicieux de pouvoir détecter de telles erreurs le plus tôt possible, pour réduire les coûts de re-conception et de maintenance.

D'autres types d'erreurs sont plus ponctuels mais peuvent avoir des conséquences dramatiques compte tenu de la propagation sur plusieurs pages. Par exemple, l'oubli d'un texte alternatif sur une image, comme par exemple un logo, signifierait que, sur une application déjà implémentée, l'évaluateur serait amené à corriger plusieurs occurrences d'une erreur qui aurait pu être corrigée en une seule fois si elle avait été détectée plus tôt. Si les corrections ne peuvent pas être automatisées, la correction de l'ensemble des occurrences peut s'avérer longue et fastidieuse.

Pour éviter les inconvénients liés à la détection d'erreurs sur l'application finale, il est nécessaire de pouvoir automatiser l'inspection ergonomique dans toutes les phases du processus de conception. L'inspection automatique doit alors porter sur les différents éléments, appelés artefacts, produits à chaque phase du processus de conception. Ces artefacts contribuent à définir, spécifier, concevoir, et implémenter l'application finale. Ils contiennent par conséquent des informations qui peuvent être exploitées et évaluées afin de vérifier leur conformité avec les règles ergonomiques.

L'objectif de nos travaux de thèse est justement d'étudier des méthodes d'inspection ergonomique automatique qui pourraient être appliquées tout au long du processus de conception. L'intérêt scientifique de ces travaux porte sur trois points principaux : l'identification des concepts de l'interface concernés par les règles ergonomiques tout au long du processus de conception ; la définition d'une méthode d'inspection des règles sur différents artefacts de conception ; et enfin l'automatisation de cette inspection. Ces questions ont été étudiées dans le cadre d'une convention CIFRE avec la société Génigraph et l'Université Paul Sabatier. Ce contexte industriel a guidé nos travaux vers la complète mise en œuvre de la méthode proposée et la conception d'un outil qui démontre l'opérationnalisation de notre approche sur des projets d'e-gouvernement. Une des contraintes fortes de ces projets est la conformité des applications Web développées avec les normes d'accessibilité en vigueur (WCAG 1.0 [WCAG] constituant la référence internationale ; RGAA [RGAA] et AccessiWeb [AccessiWeb] constituant les références pour la France).

L'organisation et la structuration des règles ergonomiques sont issues de corpus existants [Scapin 99] [WCAG] et ont permis de mettre en évidence les différents concepts à inspecter sur une application Web. Les concepts identifiés ont été regroupés et décrits de manière systématique dans une ontologie, ce qui a permis de les formaliser. De plus, une logique d'évaluation de chaque règle ergonomique a été dégagée, de sorte que soient identifiés les éléments conceptuels à évaluer sur les artefacts produits dans le cycle de vie. La méthode, au cœur de cette thèse, se base ainsi sur une ontologie établie à partir de connaissances ergonomiques, et dont les concepts sont mis en correspondance avec ceux manipulés par les artefacts produits au cours de la conception Web. L'application finale n'est plus le seul artefact sur lequel des évaluations peuvent être effectuées : les artefacts produits dès les premières phases de conception peuvent faire l'objet d'évaluations et de corrections éventuelles.

Le chapitre 2 présente un état de l'art portant sur les connaissances ergonomiques, les processus et méthodes de conception, et les outils de support basés sur des connaissances ergonomiques. Nous présentons les différentes dimensions concernées par la connaissance ergonomique ainsi que son expression au moyen de règles. Les processus et méthodes de conception sont également étudiés afin de mettre en évidence les différents artefacts produits dans le cycle de vie. Enfin, nous nous intéressons à l'automatisation des règles à travers les outils existants.

Le chapitre 3 propose une ontologie des éléments de l'interface utilisateur d'une application Web pour l'organisation des règles ergonomiques. Le développement de cette ontologie à partir de corpus de règles ergonomiques, ainsi que l'ontologie elle-même sont

détaillés dans ce chapitre. Pour illustrer l'utilité de l'ontologie dans le cycle de vie, une étude sur son utilisation pour l'inspection ergonomique avant l'implémentation est également présentée.

Le chapitre 4 présente une méthode pour l'inspection automatique des artefacts produits tout au long du cycle de vie des applications Web. Cette méthode se base sur l'ontologie des éléments de l'interface utilisateur présentée dans le chapitre 3 pour faire le lien entre les règles ergonomiques à inspecter et les artefacts à évaluer.

Le chapitre 5 présente l'outil que nous avons développé afin de supporter la méthode d'inspection automatique proposée au chapitre 4. Nous y détaillons les différents modules composant l'outil ainsi que l'interaction entre ces modules. Une version de cet outil, non autonome, est également présentée : cette version a été développée dans un contexte industriel pour l'atelier e-Citiz.

Le chapitre 6 présente une étude de cas sur la conception du site Web d'une mairie. Cette étude de cas illustre l'application de notre méthode pour l'inspection automatique des différents artefacts produits dans le cycle de vie de l'application. Pour chacun de ces artefacts, l'ensemble des règles vérifiables est étudié en détail.

Le chapitre 7 conclut ces travaux de thèse en présentant nos contributions et leurs limites, ainsi que les perspectives ouvertes pour des travaux futurs.

2. Etat de l'art

Résumé

L'évaluation de la qualité ergonomique d'une application Web peut être réalisée grâce aux connaissances ergonomiques établies sur ces applications. Ces connaissances peuvent être appliquées aux différents éléments produits à chaque étape du processus de conception de l'application Web, autorisant ainsi une évaluation tout au long de la conception. Actuellement, plusieurs outils d'aide à la conception et à l'évaluation basés sur des connaissances ergonomiques ont été implémentés pour assister l'évaluateur dans sa tâche, réduisant ainsi le temps et le coût de l'évaluation ergonomique grâce à une automatisation de la vérification.

Ce chapitre présente un état de l'art sur les connaissances ergonomiques, les processus, méthodes et démarches de conception Web, ainsi que les outils d'aide à la conception et à l'évaluation basés sur des recommandations ergonomiques.

Ce chapitre est organisé comme suit :

2.1 Connaissances ergonomiques

2.2 Processus et méthodes de conception Web

2.3 Outils de support basés sur des connaissances ergonomiques

2.4 Conclusion

Dans ce chapitre, nous présentons un état de l'art sur les connaissances ergonomiques pour le Web. Tout d'abord, nous nous intéressons aux différents types de connaissances, leurs sources et leur niveau d'automatisation. L'objectif de cette première section est de décrire toutes les dimensions concernées par la connaissance ergonomique et son expression au moyen des règles.

La deuxième partie de cet état de l'art concerne les processus et méthodes de conception pour le Web. Nous présentons les différentes approches utilisées actuellement, qu'elles soient issues de la recherche dans le domaine du génie logiciel pour le Web ou dans le domaine de l'IHM ou bien encore de l'observation d'approches pragmatiques. L'objectif est d'expliciter et de mettre en lumière les différents artefacts produits tout au long du processus de conception d'une application Web.

Pour compléter cette étude, la troisième section présente l'état de l'art des moyens d'automatisations des connaissances ergonomiques pour la conception et l'évaluation des applications Web. Nous nous intéressons particulièrement au niveau d'automatisation (automatique, semi-automatique, manuel) des règles dans les outils existants.

2.1. CONNAISSANCES ERGONOMIQUES

La diffusion de systèmes interactifs auprès d'un public d'utilisateurs de plus en plus large a mis les concepteurs face à des problématiques d'utilisabilité. Plusieurs corps de métier se sont attachés à l'étude de ces problématiques : ergonomes, psychologues cognitifs, développeurs d'interfaces, organismes de standardisation. Les réflexions qui ont alors été menées ont abouti à un ensemble de connaissances ergonomiques de nature différente, en accord avec les besoins et contraintes de chacun : par exemple un développeur d'interfaces aura tendance à décrire les connaissances ergonomiques d'un point de vue technique, en vue de son application à l'implémentation ; un psychologue cognitif décrira quant à lui les mécanismes d'utilisation d'une application d'un point de vue moins technique, plus centré sur l'utilisateur.

Les connaissances ergonomiques établies portent sur différentes dimensions d'une application Web telles que l'utilisabilité ou l'accessibilité. Toutes ces dimensions permettent de couvrir les différents problèmes que peut contenir une application Web. Ces connaissances vont ainsi aider lors de la conception de l'application ou lors de l'évaluation ergonomique. Une des particularités des connaissances ergonomiques est qu'elles sont très souvent énoncées en langage naturel et non directement applicables sur l'application. Par conséquent, appliquer manuellement ces connaissances sur un système en cours de conception ou lors de son

évaluation peut s'avérer difficile et coûteux. Pour diminuer ces coûts, il est pertinent d'automatiser l'application de ces connaissances ergonomiques lorsque cela est possible.

En outre, la connaissance ergonomique est aujourd'hui vaste et difficile à gérer, même pour un expert en ergonomie. Face à ce problème, des outils de gestion des connaissances ergonomiques sont apparus. Ils facilitent par exemple l'administration de plusieurs bases de connaissances ou la sélection et la recherche des connaissances selon différents critères.

Cette section est divisée en cinq parties dans lesquelles nous présentons respectivement les différents types de connaissances ergonomiques, les différentes sources dans lesquelles sont collectées ces connaissances, les dimensions traitées par les connaissances, les niveaux d'automatisation possibles pour les connaissances ergonomiques et enfin, la gestion de ces connaissances dans les outils.

2.1.1. Différents types de connaissances ergonomiques

Il existe plusieurs types de connaissances ergonomiques pouvant aller d'énoncés d'ordre général à des énoncés très spécifiques [Ohnemus 97] [Vanderdonckt 99] [Mariage 05a]. Mariage [Mariage 05a] classe les règles en trois niveaux d'abstraction dénommés par ordre décroissant : *principes*, *règles* et *recommandations* (voir Figure 1).

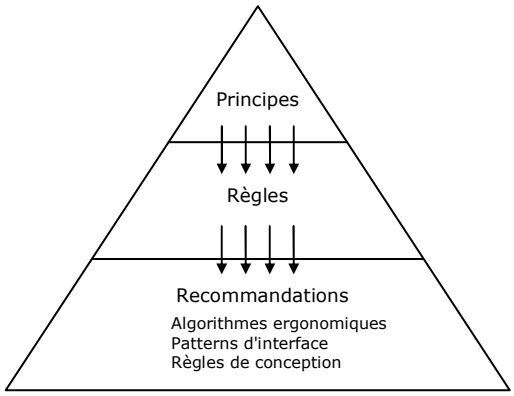


Figure 1 – Caractérisation des connaissances ergonomiques [Mariage 05a]

Elle définit les différents types de connaissances de la manière suivante :

Les **principes** sont des énoncés généraux qui spécifient les buts et objectifs de haut niveau afin de guider le développement des interfaces utilisateurs. Ils s'appuient sur la connaissance autour du comportement de l'homme en interaction avec l'ordinateur. Un exemple de principe serait « Evitez de surcharger la mémoire de l'utilisateur ». Ainsi, les principes constituent des objectifs à atteindre sans que soient précisées les façons de les satisfaire.

Les **règles** sont des énoncés qui se basent sur des principes et qui sont relatifs à un contexte particulier pour une population d'utilisateur donnée. Par exemple, dans le domaine du Web, la règle suivante est relative à la surcharge de mémoire : « Les éléments de l'écran doivent avoir du sens et doivent être consistants à travers le site de telle façon que l'utilisateur puisse reconnaître, plutôt que d'avoir à se souvenir, la signification des éléments d'une page à une autre. Les nouveaux items et nouvelles fonctions doivent se rapporter à ceux déjà connus de l'utilisateur ».

Les **recommandations** guident les décisions et choix de conception au sein d'une organisation. L'application peut être conçue de différentes manières et toutes équivalentes mais un choix doit être fait pour s'aligner avec les conventions de l'organisation. Une recommandation est par conséquent précise et non ambiguë. Dans cette dernière catégorie se trouvent les algorithmes ergonomiques, les patterns d'interface et les règles de conception.

Un **algorithme ergonomique** apparaît comme une procédure dont les étapes sont clairement définies et qui permet de systématiser la construction de l'aspect d'une interface utilisateur. La plupart du temps, il se trouve sous la forme d'un composant logiciel qui implémente un algorithme plutôt qu'une procédure papier. Par exemple, Design Advisor [Faraday 00] inclut un algorithme ergonomique qui permet de prédire l'ordre de lecture visuelle d'une page Web en fonction des animations, images, taille du texte, etc. contenues dans celle-ci.

Un **pattern d'interface** est une réponse à un problème de conception récurrent par une solution reconnue comme valide (empiriquement ou scientifiquement) par la communauté de ses utilisateurs [Alexander 77]. Un pattern d'interface organise ainsi les règles ergonomiques en une solution directement applicable. Il n'y a donc pas de problèmes d'interprétation et aucune expertise n'est requise.

Une **règle de conception** est une spécification fonctionnelle et/ou opérationnelle pour la conception d'une interface particulière et ne nécessite pas d'interprétation. Ces règles concernent l'aspect de l'interface tel que la taille de la fenêtre, l'agencement du contenu, la typographie, les couleurs, etc. Un exemple de règle pour une application Web, toujours en relation avec la surcharge de mémoire, serait : « Utiliser les mécanismes de navigation de manière cohérente, ils doivent apparaître au même endroit dans chaque page ».

2.1.2. Différentes sources

Il existe plusieurs catégories de connaissances ergonomiques qui se trouvent dans des documents de nature différente et sur différents supports (en ligne ou au format papier). Ainsi, une source peut être un document à caractère officiel (par exemple, le standard ISO 9241 [ISO9241]) ou un recueil de règles publié par un individu (par exemple, le document intitulé *Web Style Guide* de Lynch et Horton [Lynch & Horton 02]). Dans cette section, nous nous intéressons aux différentes sources que nous pouvons trouver dans la littérature et nous établirons pour chacune d'elle une liste de références.

2.1.2.1. Guide de style

Un *guide de style* est un recueil de principes, de règles et de conventions (ou recommandations) collectés dans un seul document afin de définir un aspect visuel (look and feel) unifié lors du développement d'applications [Ohnemus 97].

Pour les applications traditionnelles, les guides de style aident à définir la présentation et le dialogue d'une interface utilisateur afin de rendre consistante la conception de plusieurs modules sur une même plateforme ou un même produit. Ces recommandations proposent des conseils de nature différente, allant de la mise en page statique (agencement des composants de l'interface, couleurs, polices, etc.) à l'échange d'informations avec l'utilisateur (boîtes de message, formulaires, aide, etc.). Désireux d'unifier l'aspect visuel de leurs productions, de nombreux constructeurs publient ainsi des guides de style pour le développement d'applications sur leur plateforme. Des guides de style sont parfois publiés pour un domaine particulier ou au sein d'une entreprise. Nous pouvons citer par exemple :

- Windows XP Visual Guidelines de Microsoft [Microsoft 02] ;
- Apple Human Interface Guidelines de Apple [Apple 06] ;
- Java Look and Feel Design Guidelines de Sun [Sun 99].

Dans le domaine du Web, les guides de style se focalisent principalement sur les problèmes liés à la navigation, au look and feel commun, ainsi qu'à l'affichage de l'information et, contrairement aux guides de style traditionnels, certains points ne sont pas ou peu traités comme l'aide, les boîtes de message ou les actions de l'utilisateur (sélection, désignation, saisie de données, etc.) [Ohnemus 97]. La navigation est un aspect central des applications Web que nous ne trouvons pas ou peu dans les systèmes traditionnels, ce qui explique que la part consacrée à la navigation dans les guides de styles Web est importante. Le look and feel est

aussi une préoccupation majeure : les applications Web sont diverses et un look and feel commun permet de ne pas déstabiliser l'utilisateur lorsqu'il navigue entre plusieurs applications. Les recommandations pour l'affichage de l'information constituent aussi une grande part des guides de style Web puisque les applications Web peuvent contenir beaucoup d'informations.

En outre, contrairement aux guides de style traditionnels, les connaissances inclues dans un guide de style Web peuvent provenir de plusieurs domaines : facteurs humains, marketing, arts graphiques, relations publiques, etc. Enfin, les guides de style Web apparaissent comme moins rigoureux que les guides de style traditionnels puisque beaucoup sont élaborés par des auteurs individuels alors que les guides traditionnels sont en général établis par des organismes, sociétés, etc. Voici quelques références pour les guides de style Web :

- Style Guide for online hypertext, Tim Berners-Lee [Berners-Lee 98] ;
- IBM Style Guidelines, IBM [IBM 06] ;
- Web Style Guide, Monash University [MonashUniversity 06] (Guide de style pour la création de pages Web au sein du site Web de l'université de Monash).

2.1.2.2. Standard

Un *standard* regroupe un ensemble de spécifications fonctionnelles et/ou opérationnelles dans le but de standardiser la conception d'une interface utilisateur [Vanderdonckt 99]. Ce type de documents est promulgué par des organisations internationales ou nationales qui peuvent être militaires, gouvernementales, civiles ou industrielles. Nous pouvons citer par exemple l'AFNOR (Association Française de NORmalisation) en France, DoD (American Department of Defense) pour les Etats-Unis, ou encore l'ISO (International Standard Organization) situé en Europe. L'objectif des standards est l'homogénéité des interfaces et la généralisation des principes ergonomiques à travers les différentes plateformes et les différents environnements de développement qui ont chacune leurs propres particularités. Toutefois, certains standards peuvent entrer en conflit : dans ce cas, le concepteur devra prendre une décision. Voici des références à quelques standards :

- ISO 9241, "Ergonomic Requirements for Office Work with Visual Display Terminals" [ISO9241] ;
- HFES3/ANSI4 200, "Standard : Draft, Human Factors and Ergonomics Society" [HFES ANSI 200] ;
- MIL-STD-1472C, "Military Standard: Human Engineering Design Criteria for Military Systems" [MIL-STD-1472C].

2.1.2.3. Règles isolées

Les *règles isolées* prescrivent un énoncé à appliquer à l'interface, parfois accompagné d'exemples, avec ou sans explications ou commentaires [Mariage 05a].

Ces règles sont généralement issues d'une réflexion et ont fait l'objet d'un consensus entre plusieurs experts en facteurs humains. Certaines règles ont été validées expérimentalement, d'autres sont apparues avec l'usage. Les règles isolées se focalisent en général sur un seul aspect comme l'accessibilité [WCAG] ou la navigation [Fleming 98]. Enfin, il faut noter que les règles isolées sont souvent publiées dans les actes d'une conférence ou dans des revues scientifiques, rendant ainsi leur accès difficile. Certaines sont cependant disponibles sur le Web et par conséquent plus faciles d'accès. Voici quelques références :

- Web Content Accessibility Guidelines, W3C/WAI [WCAG] ;
- Web Navigation, Fleming, J. [Fleming 98] ;
- Useit.com, Nielsen, J. [UseIt] ;

- Web Style Guide, Lynch, P. J. & Horton, S. [Lynch & Horton 02];
- IBM Web Accessibility, IBM [IBM 04] ;
- “Increasing usability when interacting through screen readers”, Leporini, B. & Paternò, F. [Leporini 04] ;
- Usability Engineering, Nielsen, J. [Nielsen 93].

2.1.3. Dimensions traitées par les connaissances ergonomiques

Les connaissances ergonomiques portent sur différentes dimensions d’une application Web, par exemple, l’utilisabilité ou l’accessibilité. Ces dimensions ont des objectifs différents (par exemple, l’utilisabilité consiste à faciliter l’utilisation d’une application alors que l’accessibilité consiste à faciliter l’accès aux informations contenues dans l’application) mais contribuent toutes à une meilleure qualité ergonomique de l’application. Dans cette section, nous détaillons des différentes dimensions prises en compte par les connaissances ergonomiques.

2.1.3.1. Utilisabilité

D’une manière générale, l’utilisabilité vise à améliorer l’utilisation d’un système par l’utilisateur. L’ISO définit un système comme utilisable lorsqu’il permet à l’utilisateur de réaliser sa tâche avec *efficacité*, *efficience* et *satisfaction* dans le contexte d’utilisation spécifié [ISO9241]. Autrement dit, un système est utilisable lorsque l’utilisateur peut réaliser sa tâche (efficacité) en consommant un minimum de ressources (efficience) et que le système est agréable à utiliser (satisfaction). Nielsen [Nielsen 93] précise davantage la notion d’utilisabilité en la définissant comme étant une propriété multidimensionnelle d’une interface utilisateur associée à cinq facteurs et qui entre dans le cadre de l’acceptabilité du système (voir Figure 2 ci-dessous) :

- *facilité d’apprentissage* : le système doit être facile à apprendre de telle sorte que l’utilisateur puisse rapidement travailler avec ;
- *efficience* : le système doit être efficace à l’utilisation, de telle manière qu’une fois la phase d’apprentissage passée, un haut niveau de productivité soit possible ;
- *facilité à mémoriser* : le système doit être facile à mémoriser afin qu’un utilisateur occasionnel puisse revenir après une longue période et retravailler sans avoir à tout réapprendre ;
- *erreurs* : le système doit avoir un faible taux d’erreurs pour éviter que l’utilisateur n’en rencontre trop pendant l’utilisation. Si jamais des erreurs surviennent, l’utilisateur doit pouvoir récupérer de celles-ci. Les erreurs catastrophiques ne doivent pas survenir ;
- *satisfaction* : le système doit être agréable à utiliser de telle sorte que l’utilisateur en soit satisfait.

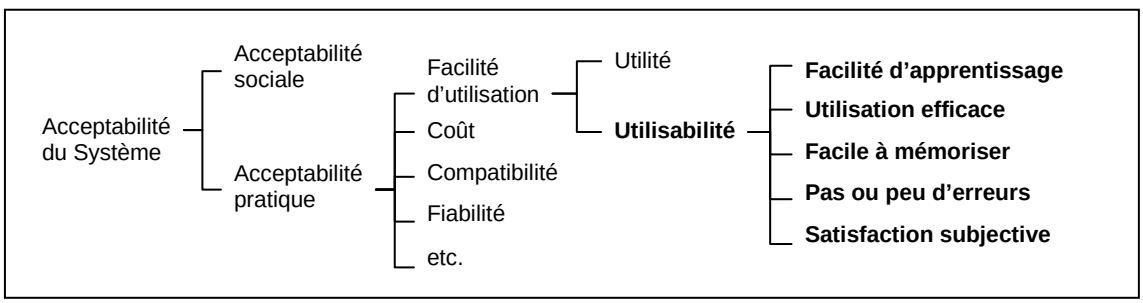


Figure 2 – Les différents facteurs de l’utilisabilité [Nielsen 93]

Cette définition a l'avantage de préciser les différents facteurs mesurables pour l'évaluation de l'utilisabilité, contrairement à la définition donnée par la norme ISO. Nous pouvons noter que, bien que la définition de Nielsen ait été donnée dans le cadre des applications classiques (c'est-à-dire non Web), cette définition est toute aussi valable et adaptable aux applications Web [Fleming 98].

Depuis les années 80, les connaissances ergonomiques relatives à l'utilisabilité des systèmes informatiques ont été nombreuses, nous pouvons citer par exemple : [Smith & Mosier 86], [Nielsen 93], [Vanderdonckt 94]. Depuis l'apparition du Web, les connaissances ergonomiques sur l'utilisabilité ont été adaptées pour ce type d'applications, par exemple : [Fleming 98], [Nielsen 00].

2.1.3.2. Accessibilité

L'accessibilité vise à rendre possible l'utilisation d'une application à un public le plus large possible en éliminant les barrières techniques potentielles qui empêcheraient ces utilisateurs d'accéder aux données et d'interagir avec l'application. Ces barrières peuvent être liées à plusieurs éléments :

- *Le contenu (information)* : des problèmes peuvent survenir si l'information contenue dans l'application Web n'est pas disponible, peu ou pas compréhensible et inefficace d'interaction (par exemple, difficulté de navigation d'une information à une autre) ;
- *Le handicap* : certaines applications mal conçues peuvent être difficiles d'accès pour certaines personnes ayant un handicap. Par exemple, un utilisateur malvoyant pourra naviguer dans un site Web grâce à un lecteur d'écran mais s'il est confronté à des images n'ayant aucune alternative textuelle, les informations véhiculées par ces images ne lui seront pas transmises. Outre le handicap visuel, les handicaps auditifs, physiques, cognitifs et/ou neurologiques nécessitent d'être aussi pris en compte. Notons que la population de personnes handicapées augmente avec l'âge, par exemple, dans la tranche des 45-54 ans, plus d'une personne sur cinq est handicapée [Thatcher 02] ;
- *Les dispositifs utilisés* : les dispositifs matériel et/ou logiciel utilisés peuvent être la source de problèmes d'accessibilité. La configuration matérielle typique d'une personne malvoyante consiste en l'unité centrale de l'ordinateur auquel est branché un clavier. L'utilisation et la navigation dans le site Web se fait alors via ce clavier, seul dispositif d'entrée. Ainsi, des problèmes d'accessibilité peuvent survenir si l'information n'est pas navigable ou difficilement navigable au moyen du clavier. D'autres dispositifs matériels comme les téléphones portables ou PDA sont aussi susceptibles d'engendrer des difficultés dues à la petite taille et la faible résolution des écrans. Les dispositifs logiciels peuvent aussi contribuer aux problèmes d'accessibilité : une application Web peut être interprétée différemment suivant le navigateur et/ou le système d'exploitation utilisés et ces différences peuvent rendre une information non accessible. C'est le cas par exemple, de l'emploi de styles CSS, bien interprétés sur certains navigateurs mais peu ou mal supportés dans d'autres, pouvant provoquer un comportement inattendu : l'information est cachée, l'information ne se trouve pas au bon endroit, etc. Les problèmes d'accessibilité liés aux dispositifs risquent de s'accroître dans les années à venir puisque leur nombre augmente considérablement : les téléphones portables, les PDA, les systèmes domotiques, etc. [Sierkowsky 02]
- *La situation d'utilisation* : l'accès au Web est de nos jours rendu possible depuis différents dispositifs comme nous venons de le voir. L'utilisation d'Internet peut ainsi se faire en situation de mobilité depuis un téléphone portable ou un PDA, limitant ainsi les possibilités d'affichage et augmentant par là même les problèmes potentiels d'accessibilité. En outre, cette utilisation peut se faire dans un environnement peu lumineux rendant difficile la perception des contrastes par exemple sur l'interface

d'une application Web. Ainsi, des informations qui étaient mises en valeur peuvent ne plus l'être dans un contexte où la luminosité n'est pas suffisante. Dans le cas d'un environnement bruyant, l'accès à des informations audio peut être difficile. Or, aujourd'hui, l'utilisation de la vidéo (dans laquelle l'information sonore est toute aussi importante que l'information visuelle) est de plus en plus courante sur Internet : des sites Web comme Youtube[1] ou Dailymotion[2] contiennent un nombre impressionnant de vidéos, allant de vidéos amateurs jusqu'à des émissions de télévision et des informations journalistiques. Plusieurs sites Web n'hésitent plus à faire référence à ces vidéos pour relater un événement ou tout simplement relayer l'information.

L'importance de l'accessibilité a été identifiée dès l'apparition du Web et dès 1999 sont apparus les WCAG 1.0 (Web Content Accessibility Guidelines 1.0) [WCAG], document publié par le W3C/WAI et établissant les recommandations pour l'accessibilité. Ce dernier est aujourd'hui encore la référence internationale en matière d'accessibilité sur le Web.

En outre, Internet (et les nouvelles technologies) a ouvert la voie à l'administration électronique : plusieurs sites Web à caractère administratif offrent désormais divers services aux citoyens tel que la demande d'extrait d'acte de naissance, la déclaration des impôts, etc. La conséquence a été une prise de conscience des différents pays du monde de l'importance de voter une loi en faveur de l'accessibilité des sites Web administratifs et gouvernementaux, ceci afin de n'écarter aucun citoyen de l'accès à ces services électroniques. Chaque loi est accompagnée par des recommandations pour l'accessibilité propres au pays en question, ou à défaut, par celles du W3C/WAI. Pour ne citer que quelques exemples, nous avons les recommandations de la Section 508[3] (Etats-Unis), les recommandations RGAA[4] (Référentiel Général d'Accessibilité des Administrations, France), AnySurfer[5] (Belgique), etc. Une étude plus détaillée des différentes lois et recommandations pour chaque pays est donnée dans [Thatcher 06].

2.1.3.3. Connaissances liées à un domaine d'application : exemple de la Mobilité

Dans un contexte de mobilité, les besoins de l'utilisateur interagissant avec un dispositif mobile (téléphone portable, PDA, etc.) sont très différents des besoins classiques avec un ordinateur de bureau ou un ordinateur portable : les dispositifs utilisés doivent être petits, compacts et facilement transportables pour pouvoir être utilisés en déplacement ; ils doivent avoir une grande autonomie pour pouvoir rester en permanence allumés ou en veille ; ils doivent démarrer rapidement (l'utilisateur ne doit pas attendre plusieurs secondes avant que son dispositif ne s'allume) ; ils doivent être fiables et ne pas comporter d'erreurs fatales [Salmre 05].

Ces besoins ont pour conséquence une limitation dans les performances (la puissance de calcul est faible par rapport aux ordinateurs de bureau et les ordinateurs portables) ainsi que dans les ressources (les dispositifs mobiles ne disposent pas de beaucoup de mémoire, par conséquent, seules les ressources peu volumineuses peuvent être manipulées par les applications installées). En outre, du fait de leur petit format, les écrans sont de dimension réduite, la taille des claviers (matériels et/ou logiciels) est restreinte et il n'y a pas de souris. Toutes ces spécificités font que les connaissances ergonomiques doivent être adaptées aux dispositifs mobiles.

D'autre part, l'utilisation du Web est de plus en plus courante sur les dispositifs mobiles, par exemple, pour vérifier ses e-mails lorsqu'on est en déplacement, pour lire l'actualité, etc. Aujourd'hui, la plupart des dispositifs mobiles offrent un navigateur Web. A la différence des navigateurs Web classiques (c'est-à-dire de bureau), ces navigateurs ont été conçus pour extraire l'information essentielle d'une page Web dans le but d'obtenir une page dite « distillée », c'est-à-dire une page n'affichant que cette information. Cependant, cette extraction

[1] http://www.youtube.com/
[2] http://www.dailymotion.com/
[3] http://www.section508.gov/
[4] http://rgaa.referentiels.modernisation.gouv.fr/
[5] http://www.anysurfer.be/

est une tâche complexe et difficile. Un moyen pour réduire cette difficulté et obtenir ainsi de meilleurs résultats est de bien concevoir l'application Web pour ce genre de dispositifs.

Des recommandations ont été publiées par le W3C/MWI[6] afin de vérifier qu'une application Web peut être supportée et utilisée depuis un dispositif mobile. Ces recommandations portent le nom de *mobileOk Basic Tests 1.0* [mobileOK] et la dernière version en date est celle du 30 novembre 2007. D'autres recommandations pour la mobilité ont été publiées telles que les patterns d'interface de Little Springs Design[7].

2.1.3.4. Bonne pratique d'un langage informatique

Certaines règles concernent la bonne pratique d'un langage informatique. Ces règles ont commencé à apparaître avec les premiers navigateurs dans le but d'assurer une compatibilité au niveau du comportement et du rendu de l'application finale quel que soit le navigateur utilisé. En effet, les navigateurs sont trop permissifs : ils acceptent du code mal écrit tout en affichant, dans certains cas, la page Web telle que voulue par le développeur. Si cet effet non voulu n'est pas gênant sur l'instant (la page est affichée comme le développeur le désirait), il n'est pas assuré, d'une part, que dans les futures versions de ce navigateur, l'affichage de la page se passe toujours comme prévu, et d'autre part, que les autres navigateurs traitent cette page de la même manière. Quelques exemples typiques pouvant provoquer ces problèmes d'utilisabilité sont des balises HTML mal fermées ou des balises entrelacées.

Pour pallier à ces défauts, des règles de validation et de correction du code source ont été publiées et directement implémentées dans des outils tels que Weblint [Bowers 96] ou Tidy [Raggett 98].

2.1.4. Niveau d'automatisation des connaissances ergonomiques

La vérification des connaissances ergonomiques sur une application Web est une tâche qui peut être totalement automatisée, partiellement automatisée, ou non automatisable. Si l'on se réfère aux différents types de connaissances (voir section 2.1.1) et à leur niveau d'abstraction (voir Figure 1), les *principes ergonomiques* ne sont pas automatisables de par leur nature puisqu'ils énoncent des objectifs de haut niveau ; les *règles ergonomiques* peuvent se prêter à une automatisation totale ou partielle (elles nécessitent parfois d'être interprétées par un évaluateur humain) ; enfin, les *recommandations ergonomiques* sont la plupart du temps totalement automatisables. Par conséquent, nous ne traiterons que les règles et les recommandations puisque les principes ne sont pas automatisables, et nous regrouperons ces deux types de connaissances sous le terme de « règle ergonomique ».

Plusieurs paramètres rendent une règle ergonomique plus ou moins automatisable :

- la nature des éléments qu'elle considère (ex. une règle ergonomique traitant du vocabulaire et/ou du style employé dans un texte sera difficile à automatiser) ;
- l'étape de conception et notamment l'état courant de l'application et des éléments établis au cours de la conception (ex. une règle traitant du poids maximal d'un objet peut être évaluée seulement lorsque l'objet est défini : ainsi, on ne pourra évaluer le poids d'une page qu'après avoir renseigné tout son contenu, c'est-à-dire à une étape avancée de la conception).

Dans ces circonstances, nous considérons donc que, selon sa nature et son contexte d'évaluation, une règle ergonomique pourra être plus ou moins automatisée, aux degrés que nous définissons ci-après :

- **Intégrée** – Une règle sera dite intégrée ou vérifiée par construction dès l'instant où un outil d'édition peut être implémenté pour contraindre le concepteur à respecter la règle : la règle est alors automatiquement vérifiée. Il est à noter que le concepteur peut

[6] W3C Mobile Web Initiative, http://www.w3.org/Mobile/
[7] http://patterns.littlespringsdesign.com/~newlsdpatterns/index.php/Main_Page, accédé le 28/02/2008

toutefois contourner ces restrictions en négligeant l'outil. Par conséquent, cette dimension n'a de sens que dans le cadre d'outils d'aide à la conception (par opposition aux outils d'évaluation dans lesquels de telles intégrations n'ont pas de sens).

- **Automatique** – Une règle sera dite automatique lorsqu'elle ne nécessitera aucune intervention humaine pour être vérifiée.
- **Semi-automatique** – Une règle sera qualifiée de semi-automatique si elle nécessite la coopération du système et de l'évaluateur pour pouvoir être vérifiée. Deux formes de collaboration peuvent se présenter :
    1. Le système peut évaluer la règle mais l'utilisateur doit lui fournir une information manquante afin qu'il puisse procéder à l'évaluation.
    2. Le système ne peut pas évaluer la règle mais il peut fournir des informations à l'utilisateur afin que celui-ci procède à une évaluation manuelle. Si l'utilisateur souhaite corriger les erreurs, le système lui proposera par exemple un ensemble d'éléments à examiner.
- **Manuelle** – Une règle sera dite manuelle ou non automatique lorsqu'elle ne pourra être évaluée que par un évaluateur humain. Les règles seront accessibles via l'outil sous la forme d'un document mentionnant les règles à évaluer pour obtenir une interface utilisable. Le concepteur aura l'entière responsabilité de l'évaluation de ces règles et des éventuelles réparations que leur violation implique.
- **Non vérifiable** – Une règle sera qualifiée de non vérifiable lorsque ni le système ni l'évaluateur humain ne peuvent évaluer la règle.

2.1.5. Gestion des connaissances ergonomiques dans les outils

Les connaissances ergonomiques ont été établies et collectées dans différentes sources (voir section 2.1.2). Ces sources apparaissent sous forme de livres, de compilations papier, ou de ressources digitales (par exemple, un site Web) et la connaissance ergonomique contenue dans celles-ci peut être vaste : une source peut parfois faire plusieurs centaines de pages. L'organisation de cette connaissance ergonomique au sein d'un tel document est donc importante et peut se faire de différentes manières : par exemple, en plusieurs principes divisés eux-mêmes en plusieurs règles et recommandations, accompagnées parfois d'exemples, de contre-exemples, d'information sur leur application dans différents contextes, etc.

Cependant, même si cette connaissance est organisée et face à la multitude de sources qu'un évaluateur peut trouver dans la littérature, un outil de gestion des connaissances ergonomiques s'avère indispensable. Ce type d'outils a pour objectif d'aider à la manipulation de la connaissance ergonomique à travers la gestion de plusieurs bases de règles et la facilitation de la recherche des règles selon plusieurs critères : nom de la source, type de source, description du contenu, origine (nom de l'organisme responsable de sa diffusion), critère ergonomique, objet à évaluer, dispositif, plateforme, aspect, domaine d'application, règles liées, etc.

La gestion des connaissances ergonomiques est proposée par des outils entièrement et exclusivement dédiés mais aussi par des outils d'aide à l'évaluation qui proposent une gestion minimale de ces connaissances. Les outils exclusivement dédiés à la gestion ont pour seul objectif de faciliter l'accès et la sélection des règles en vue d'aider à la conception ou à l'évaluation d'une application. Nous pouvons citer par exemple SIERRA [Vanderdonckt 95], HyperSAM [Iannella 95], Sherlock [Grammenos 00] ou plus récemment MetroWeb [Mariage 05b]. Nous ne détaillerons pas ces outils dans ce mémoire puisqu'ils ne supportent pas l'évaluation ergonomique d'une application. Les outils d'aide à l'évaluation proposent une gestion des règles en plus de l'évaluation de l'application conformément à ces règles (ainsi que d'autres fonctionnalités selon les outils). Cependant, la gestion des règles demeure secondaire et apparaît souvent comme minimale. Ces outils seront présentés ultérieurement dans la section 2.3.

La gestion des connaissances ergonomiques diffère selon les outils : certains intègrent directement les connaissances en dur dans le code source et empêchent par là même leur modification par l'évaluateur, alors que d'autres autorisent l'ajout, la suppression et la modification des connaissances dans la base de connaissances donnant ainsi à l'évaluateur plus de contrôle sur leur gestion. En outre, certains outils offrent la possibilité de gérer différents corpus de règles. Enfin, selon l'outil, les connaissances peuvent être sélectionnées en vue de l'évaluation d'une application Web. Nous décrivons les différentes possibilités de gestion des connaissances ergonomiques de la manière suivante :

- **Codées en dur** – Les connaissances sont codées en dur dans l'outil. Il est impossible de les extraire et de les gérer. Par conséquent, l'ajout, la suppression et la modification d'une connaissance sont impossibles sans recoder l'outil. En outre, lorsque cette gestion est destinée à l'évaluation d'une application, la logique d'évaluation est fixée une fois pour toutes et l'évaluateur ne peut la changer ;
- **Ajout, Suppression et Modification de règles** – Les règles peuvent être gérées indépendamment du moteur d'évaluation. Elles sont ainsi stockées dans une base de connaissances. Les possibilités d'ajout, de suppression et de modification de règles sont alors possibles sur cette base si de nouvelles règles doivent être intégrées ou si des règles doivent être mises à jour. Le format des règles est souvent propre à chaque outil car il n'existe pas de format standard pour la syntaxe des règles ;
- **Ajout, Suppression et Modification de corpus de règles** – Les règles peuvent être gérées et regroupées par corpus, c'est-à-dire selon leur source (WCAG 1.0, Section 508, RGAA, etc.). Les fonctionnalités d'ajout, de suppression et de modification de corpus de règles sont ainsi possibles. Les règles peuvent être gérées de la même manière au sein des différents corpus ;
- **Possibilité de sélectionner les règles à évaluer** – Lors de l'évaluation d'une application conformément à des règles, il est possible de sélectionner (ou désélectionner) une règle à vérifier. Ainsi, cette règle prendra (ou ne prendra pas) part à l'évaluation ;
- **Possibilité de sélectionner les corpus de règles à évaluer** – Lors de l'évaluation d'une application conformément à des règles, il est possible de sélectionner (ou désélectionner) un corpus de règles à vérifier. Ainsi, les règles contenues dans ce corpus seront (ou ne seront pas) évaluées.

2.2. PROCESSUS ET METHODES DE CONCEPTION WEB

L'application de connaissances ergonomiques implique la réalisation d'un produit ou d'un système interactif. La conception des systèmes interactifs est une tâche complexe qui nécessite la réalisation de plusieurs activités. L'ensemble de ces activités et de leur enchaînement forme ce que l'on appelle le *processus de conception* (si l'on ne parle que de la conception de l'application) ou de manière plus générale, le *cycle de vie de l'application* (si l'on tient compte aussi de l'utilisation et la maintenance de l'application) [Dix 03]. Les activités menées dans chaque phase du cycle de vie produisent différentes données, par exemple, le cahier des charges, des documents de spécification et de conception, différents modèles tels que des modèles de tâche, du domaine, du système, le code source de l'application, etc. Ces données, que nous dénoterons tout au long de ce mémoire par le terme générique « *artefacts* », servent également de documentation et de support aux discussions, à la prise de décisions, ou encore à la validation, depuis l'analyse des besoins jusqu'au développement, afin d'aboutir à l'application finale. Selon le cycle de vie, ces artefacts sont également modifiés et mis à jour dans les phases d'évaluation et de maintenance de l'application.

Le type d'artefacts produit à chaque étape du processus de conception dépend de la méthode de conception utilisée. Les méthodes basées sur des modèles préconisent la création et l'utilisation de leurs propres modèles au cours des différentes étapes d'un processus de

conception (ce processus est en général sous-jacent à la méthode). Les modèles utilisés, c'est-à-dire les artefacts, servent à décrire différents aspects de l'application Web comme la navigation ou la structure.

Les méthodes basées sur des langages de description d'interface permettent d'obtenir une interface abstraite de l'application finale, bénéficiant ainsi de plusieurs avantages tels que l'indépendance à une plateforme donnée, la génération semi-automatique de l'interface finale, etc. Ces méthodes ont recours à un langage permettant d'exprimer et de manipuler les différents concepts que nous pouvons trouver dans une interface utilisateur (boutons, champs de texte, menu, etc.) à plusieurs niveaux d'abstraction (interface concrète contenant des objets d'interaction concrets, ou interface abstraite ne contenant que des objets d'interaction abstraits). Ces méthodes s'inscrivent dans des processus de conception adaptés et à chaque étape, différentes interfaces utilisateurs, ainsi que différents modèles sont manipulés.

Outre ces méthodes, il est utile de mentionner qu'il existe des démarches de conception non formalisées qui décrivent la conception d'une application Web sans pour autant préciser de manière formelle le processus de conception ou la méthode adoptée. L'objectif de ces démarches, telles que celles de Nielsen [Nielsen 00], Fleming [Fleming 98] ou Lynch & Horton [Lynch & Horton 02], est de promouvoir et de mettre en lumière les différents principes d'une bonne conception à travers les différents conseils et exemples donnés, et en proposant des solutions basées sur la théorie mais aussi, qui ont été prouvées comme efficaces par l'expérience. Si des conseils sont donnés sur les différents artefacts à produire (par exemple, page Web, contenu, styles, navigation, résultats de tests), certains traits caractéristiques aux processus et méthodes formalisées ne sont pas explicités tels que l'ordre et l'enchaînement des activités, ou encore le processus d'obtention des différents artefacts. Ces démarches non formalisées ne proposant pas de méthodes de conception, nous ne nous y intéresserons pas dans l'état de l'art.

Dans cette section, nous présentons dans une première partie une discussion sur les processus de conception Web. Dans les deux sections suivantes, nous nous intéressons aux méthodes basées sur des modèles et méthodes basées sur des langages de description d'interface et nous établissons pour chacune d'elles la liste des artefacts produits au cours des différentes étapes du processus de conception.

2.2.1. Processus de conception Web

De nombreux cycles de vie ont été proposés dans la littérature depuis les années 70. Pour les systèmes interactifs classiques (par opposition aux systèmes interactifs Web), nous pouvons citer le cycle en cascade [Royce 70], le cycle en V [McDermid & Ripken 84], le cycle en spirale [Boehm 86], le cycle Star [Hix & Hartson 93], ou plus récemment RUP (Rational Unified Process) [Kruchten 00].

Cependant, contrairement aux systèmes interactifs classiques, les systèmes interactifs Web sont des systèmes ayant leurs propres particularités que ne prennent pas en compte ces processus de conception classiques :

- **La navigation** est un aspect important des applications Web [Fleming 98] que nous ne retrouvons pas dans les systèmes interactifs classiques. Dans une application Web, nous ne parlons plus d'« utiliser » l'application mais de « naviguer » dans l'application. Si la navigation est mal conçue, l'utilisateur peut se perdre et passer du temps à chercher l'information voulue. Dans le pire des cas, l'application ne lui offrira pas la possibilité de revenir sur ses pas (auquel cas, l'utilisation de la fonction « retour arrière » du navigateur sera obligatoire). Par conséquent, la navigation peut causer des problèmes d'utilisabilité importants et **nécessite d'être prise en compte** dans une étape du processus de conception ;
- Le contenu informatif est important pour les systèmes interactifs Web, contrairement aux systèmes interactifs classiques où le traitement des données est plus important.

Ainsi, les informations sont fréquemment mises à jour (d'autant plus que ces mises à jour sont rapides et peu coûteuses). Le contenu d'une application Web peut être amené, dans certains cas, à évoluer plusieurs fois en une journée ceci afin d'y ajouter de nouvelles informations, d'y apporter des modifications sur le contenu existant, etc. Par conséquent, **les itérations dans le cycle de vie peuvent être rapides et nombreuses** dans le cas des applications Web, contrairement aux applications traditionnelles où une itération peut durer jusqu'à plusieurs mois.

A l'heure actuelle, il n'existe pas de consensus, ni sur les phases de développement nécessaires, ni sur le cycle de vie qui décrirait le mieux le processus de développement des applications Web. Cependant, le cycle de vie du développement d'un site Web peut généralement être vu comme un processus itératif. Scapin et al. [Scapin 00a] proposent un processus de conception itératif pour le Web. Il est composé des six phases suivantes (voir Figure 3) : Expression des besoins, Spécification du site, Conception du site, Développement du site, Utilisation & Evaluation du site, et Maintenance du site.

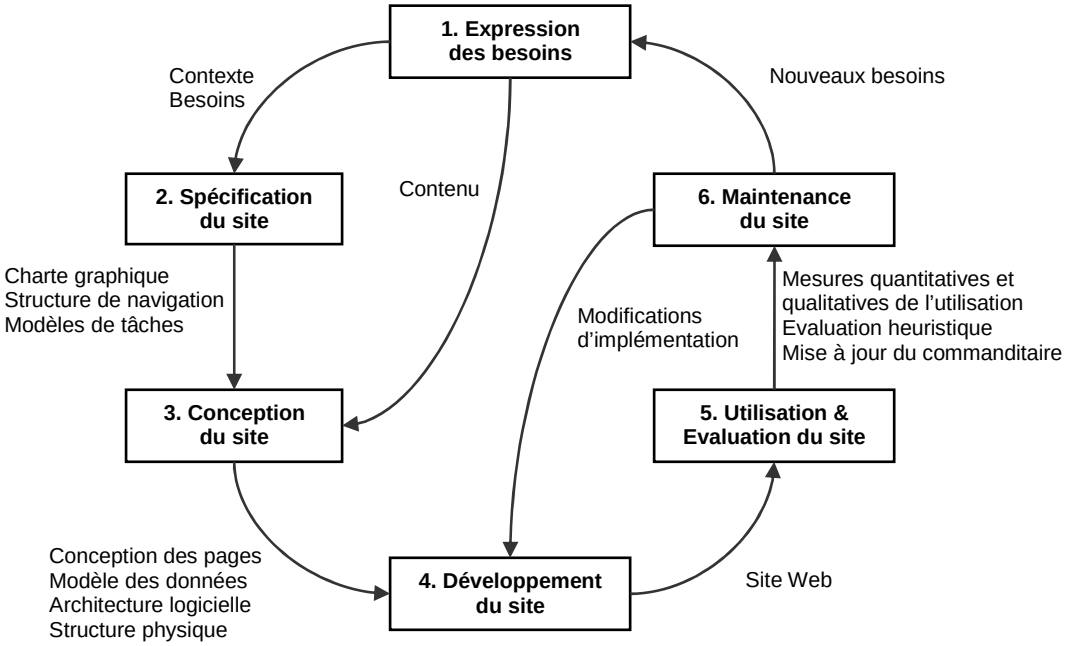


Figure 3 – Cycle de vie en O d'une application Web [Scapin 00a]

- **Expression des besoins** – La phase *Expression des besoins* (ou Analyse des besoins) identifie les principaux buts des commanditaires, le contexte d'utilisation et les besoins. Elle comprend la collecte du contenu qui sera utilisé plus tard.
- **Spécification** – La phase de *Spécification* (ou Modélisation conceptuelle) produit des spécifications à partir du contexte d'utilisation et des besoins recueillis dans la phase précédente. Des modèles détaillés sont construits pour formaliser les besoins, comme, par exemple, la structure de navigation, les tâches utilisateur réalisables avec l'architecture de l'application et du site.
- **Conception** – La phase de *Conception* consiste à affiner les spécifications d'après leur contenu. A la fin, des modèles de navigation, de page et de données sont construits. Une spécification détaillée est produite afin de guider l'implémentation de l'application Web.
- **Implémentation** – La phase d'*Implémentation* (ou Développement) correspond à la construction physique de l'application Web, à la production des pages HTML et à l'éventuelle intégration de contenus multimédias (son ou vidéo). A la fin de cette phase, toutes les pages ont un contenu, des liens et des éléments graphiques incorporés. L'application est prête à être livrée.

- **Utilisation & Evaluation du site** – La phase d'*Utilisation & Evaluation* est destinée à évaluer des prototypes avancés avec des utilisateurs finaux. Le produit issu des phases précédentes est vérifié conformément aux besoins et au contexte d'utilisation identifiés dans la première phase. L'évaluation de l'utilisabilité, par des tests utilisateurs ou par l'inspection de règles ergonomiques, est l'activité principale de cette phase.
- **Maintenance** – La phase de *Maintenance* consiste en une activité de recueil de nouvelles informations et de modifications de planning qui ont été demandées au cours de la phase d'utilisation et d'évaluation.

Ce processus cyclique ne prend pas en compte les activités de prototypage qui sont habituellement au cœur du processus de conception des applications Web. Toutefois, la flèche de gauche intitulée « Contenu » (voir Figure 3) représente un raccourci possible pour une telle activité dès la phase de spécification. En effet, dès que l'expression des besoins a été établie, le concepteur peut commencer la conception du site et discuter ainsi avec les commanditaires sur la base d'informations précises. La flèche de droite intitulée « Modifications d'implémentation » représente un raccourci possible pour que le développement soit plus rapide et pour tenir compte de l'utilisation et des évaluations de manière plus centrale. Ce raccourci symbolise que l'application Web peut être modifiée par les développeurs directement après une évaluation sans repasser par les trois premières phases du processus de conception.

Nous nous réfèrerons à ce cycle de vie pour mener les différentes discussions tout au long de ce mémoire.

2.2.2. Méthodes basées sur des modèles

Certaines méthodes de conception se basent sur l'utilisation de modèles à travers le processus de conception pour le développement d'une application Web. L'emploi de modèles comporte plusieurs avantages : ils permettent de spécifier l'application de manière formelle ; ils facilitent l'implémentation du système final, sa maintenance, et son évolution à travers une génération semi-automatique du code, des vérifications et des corrections automatiques ; ils permettent une meilleure compréhension et visualisation de l'application en cours de conception, et peuvent servir de documentation entre les différents intervenants.

Ces méthodes interviennent en général dans plusieurs étapes du processus de conception, voire dans toutes les étapes. Selon le type d'applications Web (par exemple, applications Web de type informationnel, application Web orientée données, ou applications d'e-commerce), une méthode de conception sera plus ou moins adaptée. Chacune de ces méthodes définit un processus de conception sous-jacent et utilise ses propres modèles dans chaque étape, par exemple, des modèles de structure, de comportement, de navigation, de présentation, ou de domaine.

Nous présentons dans cette section, de manière non exhaustive, différentes méthodes existantes basées sur des modèles pour les applications Web. Le lecteur pourra se référer à [Winckler 04] pour une étude exhaustive des méthodes de conception Web basées sur des modèles.

2.2.2.1. OOHDM

OOHDM (Object-Oriented Hypermedia Design Method) [Schwabe 98] est une approche basée sur des modèles pour implémenter des applications hypermédias. OOHDM comprend quatre étapes, à savoir : Modélisation conceptuelle, Conception de la navigation, Conception de l'interface abstraite et Implémentation.

Dans la phase de *Modélisation conceptuelle*, le domaine d'application est modélisé selon les principes de l'approche orientée objet avec une notation similaire à UML. L'objectif dans cette phase est de capturer la sémantique relative au domaine. Le résultat de cette étape est un

ensemble de diagrammes de classes appelé 'modèle conceptuel' et représentant les différents concepts du domaine et les liens qu'il peut y avoir entre eux.

Dans la phase de *Conception de la navigation*, les nœuds (pages) de navigation, les contextes de navigation (ensemble de nœuds de navigation) ainsi que les chemins de navigation sont identifiés à partir du modèle conceptuel. Cette identification se fait d'après le profil de l'utilisateur final et l'analyse de différents scénarios. Le produit de cette phase est un diagramme de contexte représentant les différents contextes de navigation.

Dans la phase de *Conception de l'interface abstraite*, les objets de l'interface perçus par l'utilisateur sont spécifiés au moyen de ADV (Abstract Data Views) [Cowan 95], une notation formelle pour les objets de l'interface d'une application hypermédia. Ainsi chaque objet de l'interface est décrit par sa structure, ses relations aux objets de navigation, et ses réactions aux événements extérieurs.

Dans la phase d'*Implémentation*, l'application finale est codée conformément aux spécifications ADV de la phase précédente.

Les artefacts produits et utilisés dans la méthode OOHDM sont présentés dans le Tableau 1.

Tableau 1 – Artefacts produits dans le cycle de vie selon la méthode OOHDM [Schwabe 98]

Etapes	Modélisation Conceptuelle	Conception de la navigation	Conception de l'interface abstraite	Implémentation
Artefacts produits	Classes, sous systèmes, relations, attributs, perspectives	Nœuds, liens, structures d'accès, contextes de navigation, transformations navigationnelles	Objets de l'interface abstraite, réponses aux événements externes, transformations de l'interface	Pages Web

2.2.2.2. WebML

Web Modeling Language (WebML) [Ceri 00] est une notation pour spécifier des applications Web complexes. WebML permet de modéliser plusieurs aspects d'une application Web : les données, les pages qui la composent, la navigation entre ces pages, la présentation et la personnalisation par utilisateur. WebML est une notation graphique ayant une syntaxe textuelle XML.

Un processus de conception typique utilisant WebML procède en itérant les étapes suivantes pour chaque cycle de conception : Collecte des besoins, Conception des données, Conception de l'hypertexte, Conception de la présentation, Conception des utilisateurs et groupes d'utilisateurs, Conception de la personnalisation.

Dans la phase de *Collecte des besoins*, les besoins sont identifiés. Cela inclut les principaux objectifs du site, les utilisateurs cibles, les exemples de contenu, les guides de styles, la personnalisation et les contraintes dues à la législation.

Dans la phase de *Conception des données*, un expert métier construit le modèle structurel. Ce modèle structurel définit le domaine de l'application, à savoir les données de l'application (entités, attributs et composants) et les liens qui existent entre elles. La description de ce modèle se fait en XML et peut être assimilée à un diagramme de classes UML.

Dans la phase de *Conception de l'hypertexte*, l'architecte de l'application Web construit deux modèles : le modèle de composition et le modèle de navigation. Le modèle de composition consiste à décrire les nœuds qui vont faire partie de l'application Web en termes de pages et autres unités de contenu (i.e. éléments d'information atomiques qui peuvent apparaître sur une page Web). Le modèle de navigation, quant à lui, va s'attacher à décrire comment ces pages et unités de contenu vont être liées entre elles et par conséquent, comment l'utilisateur pourra naviguer d'unité en unité. Pour cela, deux types de liens sont différenciés : les liens contextuels

et les liens non contextuels, la différence étant qu'un lien contextuel transporte de l'information (contextuelle) depuis l'unité source jusqu'à l'unité cible.

Dans la phase de *Conception de la présentation*, le look and feel final de l'application est spécifié grâce à un modèle de présentation. Ce modèle consiste en un ensemble de feuilles de styles où la mise en page (layout) et la présentation du contenu sont spécifiées. Deux types de feuilles de styles sont différenciés : les feuilles de styles non typées qui ne définissent que la mise en page indépendamment du contenu, elles peuvent donc être appliquées sur n'importe quelle page ; et les feuilles de styles typées qui sont spécifiques au contenu d'une page et ne peuvent être appliquées que sur une page donnée.

Dans la phase de *Conception des utilisateurs et groupes d'utilisateurs*, les différents groupes d'utilisateurs sont décrits. Un groupe d'utilisateur englobe des utilisateurs partageant des caractéristiques communes, caractéristiques qui sont définies grâce aux propriétés des entités du modèle structurel. Ces caractéristiques communes forment le « profil utilisateur ». Ces profils étant définis grâce au modèle structurel, ils bénéficient du pouvoir expressif d'UML et peuvent être structurés en interne (par exemple, regroupement d'attributs sous forme de composants), être classés entre eux grâce à l'héritage de profils, être reliés à d'autres entités, etc. Typiquement, un profil contient les données de l'utilisateur ou du groupe tels que les derniers objets visités (par exemple, sur un site d'e-commerce : les items consultés, la liste des derniers achats). Ce profil évolue donc au cours de la navigation.

Dans la phase de *Conception de la personnalisation*, il s'agit de définir le contenu d'une page et sa présentation en fonction des données du profil utilisateur pour une personnalisation de la page. Deux types de personnalisation peuvent être mis en place : la personnalisation déclarative et la personnalisation procédurale. La personnalisation déclarative consiste en un ensemble d'entités, d'attributs, etc. défini par l'application en fonction du comportement d'un utilisateur donné. Par exemple, un rabais sur un achat peut être fait en fonction des précédents achats de l'utilisateur. La personnalisation procédurale consiste en l'écriture de règles métier pour guider la personnalisation. Par exemple, un utilisateur peut se voir affecté au groupe « client privilégié » sur la base de son historique d'achats.

Les artefacts produits et utilisés dans la méthode WebML sont présentés dans le Tableau 2.

Tableau 2 – Artefacts produits dans le cycle de vie selon la méthode WebML [Ceri 00]

Etapes	Collecte des besoins	Conception des données	Conception de l'hypertexte	Conception de la présentation	Conception des utilisateurs et groupes d'utilisateurs	Conception de la personnalisation
Artefacts produits	Objectifs du site, utilisateurs cibles, exemples de contenu, guides de styles, personnalisation, contraintes	Entités, attributs, composants, liens entre entités	Modèle de composition (nœuds, page, unité de contenu), Modèle de navigation (liens contextuels, liens non contextuels)	Feuilles de styles typées, Feuilles de styles non typées	Profil utilisateur	Entités, Attributs, Composants (pour la personnalisation déclarative) et Règles métier (pour la personnalisation procédurale)

2.2.2.3. UWE

UWE (UML-based Web Engineering) [Koch 02] [Hennicker 00] est une méthode pour la conception d'applications Web. L'une des particularités de cette méthode est qu'elle se base sur une extension de la notation UML (une telle extension est aussi appelée « profil UML » selon la terminologie de l'OMG) pour modéliser plusieurs aspects de l'application Web tels que la navigation ou la présentation.

Cette méthode de conception définit quatre activités de modélisation qui sont l'analyse des besoins, la modélisation conceptuelle, la modélisation de la navigation et la modélisation de la présentation.

Dans la phase d'*analyse des besoins*, des cas d'utilisation (use cases) sont produits afin de déterminer quels seront les différents utilisateurs de l'application ainsi que les différentes actions qu'ils seront amenés à faire. Cette phase permet donc d'identifier les différents types d'utilisateurs de l'application ainsi que les fonctionnalités qu'il faudra fournir pour chacun d'eux.

Dans la phase de *modélisation conceptuelle*, les cas d'utilisation produits dans la phase précédente servent de base à la modélisation en UML du domaine. L'objectif de cette phase est de produire un modèle du domaine qui prend en compte les chemins de navigation, la présentation et l'interaction, aspects qui seront nécessaires dans les deux phases suivantes (modélisation de la navigation et modélisation de la présentation). Ce modèle est décrit sous forme de diagrammes de classes UML.

La phase de *modélisation de la navigation* spécifie à la fois les objets pouvant être visités par navigation (modèle de l'espace de navigation) mais aussi la manière dont ces objets peuvent être atteints (modèle de la structure de navigation). Le modèle de l'espace de navigation est un diagramme UML constitué de classes navigables chacune identifiée par le stéréotype « navigational class ». La navigation de classe en classe se fait grâce à des liens de « navigabilité directe » modélisés par des associations entre classes. Le modèle de la structure de navigation transforme le modèle de l'espace de navigation et l'enrichit avec des éléments de plus haut niveau tels que « index », « visite guidée », « requête » et « menu ». Ces éléments permettent de structurer la navigation en spécifiant comment accéder à chaque page (par un index, par une requête, etc.). Le résultat est un diagramme de classes UML enrichi.

La phase de *modélisation de la présentation* décrit comment l'information accessible à travers l'espace de navigation et ses différentes structures est présentée à l'utilisateur. Cette phase définit donc l'interface abstraite de l'application. A chaque classe navigable et à chaque structure de navigation sont associées une ou plusieurs classes de présentation. Une classe de présentation est un conteneur qui comprend des éléments basiques tels que 'image', 'ancre', 'collections d'ancres', etc. Ainsi, chaque classe ou structure de navigation se voit affecter un modèle (template) pour sa présentation.

Les artefacts produits et utilisés dans la méthode UWE sont présentés dans le Tableau 3.

Tableau 3 – Artefacts produits dans le cycle de vie selon la méthode UWE [Koch 02]

Etapes	Analyse des besoins	Modélisation Conceptuelle	Modélisation de la navigation	Modélisation de la présentation
Artefacts produits	Cas d'utilisation en UML	Diagrammes de classes UML	Diagrammes de classes UML enrichis avec des classes et des structures de navigation	Classes de présentation (ou template)

2.2.2.4. eW3DT

eW3DT (extended World Wide Web Design Technique) [Scharl 99] est un méta modèle permettant la description de Systèmes d'Informations Web (WIS) et est une extension de W3DT [Bichler 96]. La modélisation d'un WIS se fait grâce à la notation graphique associée au méta modèle eW3DT, à travers des diagrammes composés de différents types d'objet : Information, Navigation et Structure. Les objets de type Information (page, menu, index, etc. tous considérés comme unités d'information atomiques) modélisent le contenu de l'application et peuvent être de classe statique ou dynamique (excepté l'objet DBase qui est toujours considéré comme dynamique). Les objets de type Navigation regroupent différents types de lien (statique, dynamique, représentatif et horizontal). Enfin, les objets de type Structure (élément de

structuration primaire, lien externe et sources multiples) permettent de gérer la complexité en faisant le lien entre plusieurs sous modèles d'une même application. Plus de détails sur les types d'objets sont donnés dans [Scharl 99]. Cette notation permet donc de modéliser d'une manière abstraite l'application sans entrer dans les détails techniques de l'implémentation.

Les diagrammes eW3DT permettent d'établir soit un modèle de référence (spécifique à un domaine mais indépendant d'une organisation/société donnée) afin de déduire l'application finale (approche descendante), soit un modèle destiné à l'analyse de l'application existante en vue d'améliorations ou de corrections (approche ascendante). Pour ces deux types de modèles ayant des objectifs différents (conception vs évaluation), les objets modélisés comportent des informations différentes. Dans un contexte de conception (approche descendante), les objets mentionnent des informations qui leur sont propres, telles que leur niveau hiérarchique (en terme de localisation) dans l'application, les références vers les unités fonctionnelles chargées du contenu et de l'implémentation, et l'effort de maintenance. Dans un contexte d'évaluation (approche ascendante), les objets mentionnent les informations suivantes : le nombre de requêtes sur ceux-ci pendant une certaine période et le temps moyen en secondes de leur visualisation par les utilisateurs. Les diagrammes eW3DT couvrent ainsi à la fois les phases de conception et d'évaluation d'un système d'information Web.

Le processus de conception préconisé par les auteurs est composé de deux processus distincts appelés révolutionnaire et évolutionniste, schématisés sur la Figure 4 et détaillés ci-après.

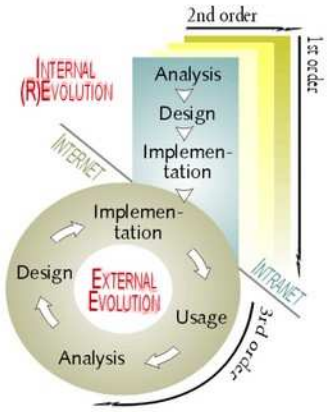


Figure 4 – Processus de développement eW3DT d'un Système d'Information Web [Scharl 99]

Le *processus révolutionnaire* (Internal (R)evolution) consiste en la création de l'application et passe par trois phases : *analyse*, *conception* et *implémentation*. Dans ce processus, deux types d'évolution apparaissent : la modification d'un produit ou service au sein de l'application (1er ordre) ; la modification du processus, de la méthodologie et de la conception générale de l'application (2nd ordre).

Le *processus évolutionniste* (External Evolution) consiste quant à lui en l'évolution même de l'application à travers l'usage qui est fait de lui. Ce processus est un enchaînement cyclique des phases suivantes : *utilisation*, *analyse*, *conception*, *implémentation*. Contrairement au processus révolutionnaire, il tient compte des facteurs extérieurs à la société pour laquelle l'application est développée, tels que les relations avec les clients, avec les autres sociétés, les institutions gouvernementales, les comités de standardisation, etc.

Les artefacts produits et utilisés dans la méthode eW3DT sont présentés dans le Tableau 4.

Tableau 4 – Artefacts produits dans le cycle de vie selon la méthode eW3DT [Scharl 99]

Etapes	Révolution			Evolution			
	Analyse	Conception	Implémentation	Utilisation	Analyse	Conception	Implémentation
Artefacts produits	–	Modèle de référence (approche descendante) : - diagramme d'objets de type Information, Navigation et Structure - chaque objet contient les informations suivantes : + niveau hiérarchique + références vers les unités fonctionnelles chargés du contenu et de l'implémentation + effort de maintenance	–	–	Modèle d'analyse (approche ascendante) : - diagramme d'objets de type Information, Navigation et Structure - chaque objet contient les informations suivantes : + le nombre de requêtes sur ceux-ci pendant une certaine période + le temps moyen en secondes de leur visualisation par les utilisateurs	Modèle de référence (approche descendante) : - diagramme d'objets de type Information, Navigation et Structure - chaque objet contient les informations suivantes : + niveau hiérarchique + références vers les unités fonctionnelles chargées du contenu et de l'implémentation + effort de maintenance	–

2.2.3. Méthodes basées sur des langages de description d'interface

Un langage de description d'interface utilisateur est un langage informatique de haut niveau permettant de décrire de manière abstraite les caractéristiques d'une interface utilisateur. La description d'une interface au moyen d'un tel langage peut être considérée comme un moyen de spécifier une interface utilisateur indépendamment d'un langage cible (de programmation ou de balisage). Plus précisément, elle permet le développement d'une interface utilisateur comme un module d'une application interactive plutôt qu'une suite de lignes de code ; cette interface abstraite peut jouer le rôle de spécifications susceptibles d'être partagées avec les personnes impliquées dans la conception de l'application ; elle est également adaptée à la génération semi-automatique de l'interface finale ; et enfin, une telle description abstraite rend l'interface utilisateur multi plateforme (l'interface peut s'exécuter sur plusieurs plateformes) sans avoir à adapter le code source pour une plateforme donnée.

Les méthodes basées sur des langages de description d'interface interviennent dans plusieurs étapes du processus de conception. Cependant, à la différence des méthodes basées sur des modèles, les méthodes utilisant un langage de description d'interface ne s'attachent qu'à l'élaboration de l'interface utilisateur. Ainsi, l'utilisation de modèles de tâche et de domaine peut servir de spécifications et de point de départ à la description de l'interface ; cette dernière est ensuite raffinée en une interface ne contenant que des objets d'interaction abstraits (l'interface est indépendante d'une modalité choisie) pour plus tard ne contenir que des objets d'interaction concrets (l'interface est indépendante d'une plateforme donnée), etc. De telles méthodes ne se préoccupent pas des autres aspects telles que la navigation ou la structure de l'application.

Il existe plusieurs langages de description d'interface dans la littérature. Dans cette section, nous ne présenterons que quelques langages représentatifs. Le lecteur pourra trouver une étude comparative sur les langages de description d'interface dans [Souchon 03].

2.2.3.1. Teresa

Teresa (Transformation Environment for inteRactivE Systems representAtions) [Paternò 03] désigne à la fois un environnement de développement supportant la conception et la génération d'une interface utilisateur multimodale, mais aussi le langage de description d'interface utilisateur qui l'accompagne.

La méthode utilisée pour le développement d'une interface multimodale consiste en plusieurs raffinements successifs d'un modèle de tâche décrit à un haut niveau d'abstraction. Cette méthode est divisée en quatre étapes : modélisation haut niveau de la tâche d'une

application multi contexte, développement du modèle de tâche système pour les différentes plateformes considérées, transformation d'un modèle de tâche système vers une interface utilisateur abstraite, et génération de l'interface utilisateur (voir Figure 5 ci-dessous).

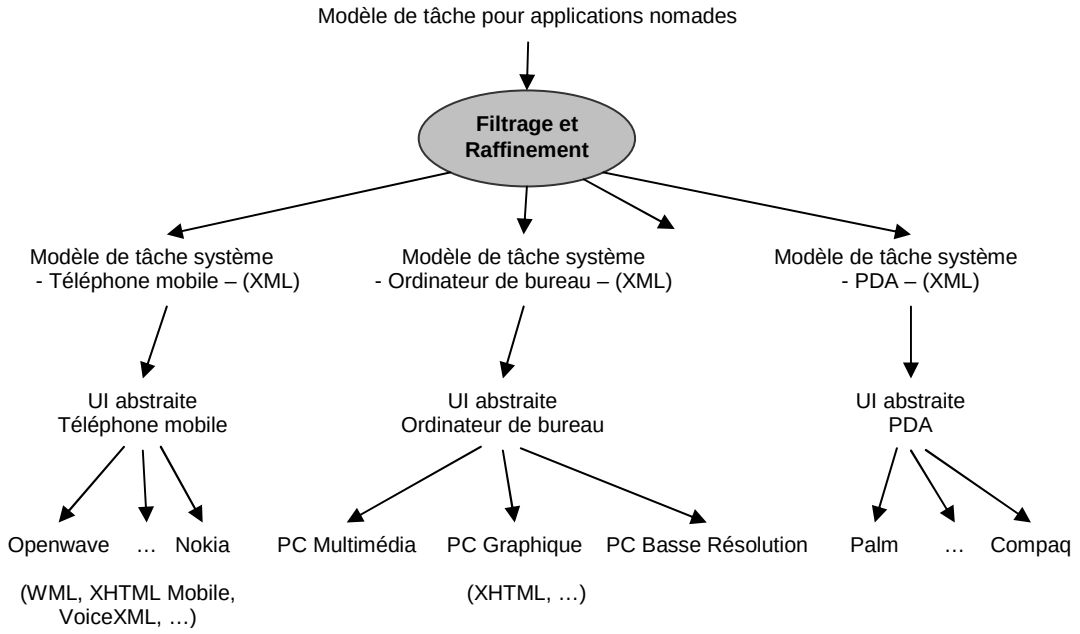


Figure 5 – Méthode pour la conception et la génération d'interfaces multimodales adoptée dans Teresa [Paternò 03]

Dans la première étape, intitulée *modélisation haut niveau de la tâche d'une application multi contexte*, les activités logiques ainsi que les liens entre ces activités sont spécifiés. Un modèle de tâche unique décrivant ces activités et ces liens, et tenant compte de tous les contextes d'utilisation possibles, est alors conçu. En outre, puisque ce modèle de tâche décrit et manipule des objets du domaine, un modèle du domaine est également établi. Ces deux modèles sont décrits au moyen de la notation *ConcurTaskTrees* (CTT) [Paternò 97].

La deuxième étape, *développement du modèle de tâche système pour les différentes plateformes considérées*, consiste en un raffinement du modèle de tâche précédent selon la plateforme cible : les tâches ne pouvant pas être supportées sur cette plateforme sont supprimées et des tâches dites navigationnelles sont ajoutées lorsque nécessaire. Chaque tâche est décomposée si besoin est. Le modèle de tâche ainsi raffiné pour la plateforme considérée est appelé modèle de tâche système.

La troisième étape, *transformation d'un modèle de tâche système vers une interface utilisateur abstraite*, consiste en l'obtention de l'interface utilisateur abstraite. Cette interface est composée d'un ensemble d'éléments de présentation identifiés à partir de l'analyse des relations entre les tâches. Chaque élément de présentation est caractérisé par une structure qui décrit l'organisation statique de l'interface utilisateur composé d'objets d'interaction abstraits (AIO [Vanderdonckt 93]) et par des connexions qui représentent les relations avec les autres éléments de présentation (c'est-à-dire les transitions vers les autres éléments de présentation). Chaque élément d'une structure peut être soit un AIO élémentaire soit une composition de ces AIOs. Chaque AIO peut être soit un AIO d'interaction soit un AIO d'application selon qu'il y a ou non une interaction entre l'utilisateur et l'application.

La quatrième et dernière étape, *génération de l'interface utilisateur*, consiste en l'obtention de l'interface utilisateur finale à partir de l'interface utilisateur abstraite précédente. Dans cette phase, chaque AIO est mis en correspondance avec une technique d'interaction supportée par la plateforme cible.

Pour supporter les différents modèles utilisés dans cette méthode, le langage XML Teresa a été développé. Ce langage est composé de deux parties : une description XML de la notation CTT [Paternò 97], notation utilisée pour décrire un modèle de tâche, et un langage pour décrire l'interface utilisateur abstraite.

Les artefacts produits et utilisés dans la méthode Teresa sont présentés dans le Tableau 5.

Tableau 5 – Artefacts produits dans le cycle de vie selon la méthode Teresa [Paternò 03]

Etapes	Modélisation haut niveau de la tâche d'une application multi contexte	Développement du modèle de tâche système pour les différentes plateformes considérées	Transformation d'un modèle de tâche système vers une interface utilisateur abstraite	Génération de l'interface utilisateur
Artefacts produits	Modèle de tâche, Modèle du domaine	Modèle(s) de tâche système	Interface Utilisateur Abstraite composée d'AIOs	Interface Utilisateur Finale

2.2.3.2. UIML

UIML (User Interface Markup Language) [Abrams 99] est un métalangage qui peut être utilisé pour spécifier indépendamment les quatre aspects d'une interface utilisateur (IU) : structure, style, contenu et comportement. Cela permet de décrire une IU indépendante d'une plateforme donnée dans l'objectif de faire correspondre l'IU à différents systèmes d'exploitation, langages et dispositifs.

Un document UIML est un document XML constitué de trois parties : une description de l'IU, une section où sont définis les mises en correspondances à partir d'un document UIML vers des entités externes (rendu sur la plateforme cible et logique d'application), et finalement une section template qui permet la réutilisation de fragments du document UIML déjà décrits. Dans le langage UIML, une IU est décrite comme un ensemble d'éléments d'interface avec lesquels l'utilisateur interagit. Pour chaque élément d'interface, un style de présentation est donné (par exemple, position, police, couleur), le contenu est renseigné (texte, images, etc.) ainsi que les événements utilisateurs et actions résultantes.

Le rendu d'une IU, c'est-à-dire l'obtention de l'IU finale sur une plateforme donnée, est obtenu grâce à un renderer. Il existe plusieurs types de renderer, un pour chaque plateforme cible : Java, HTML, WML et VoiceXML [Ali 03]. Chaque renderer contient un vocabulaire spécifique à la plateforme permettant de décrire les éléments de l'IU, leur comportement et leur mise en page. Une IU est ainsi rendue comme spécifiée dans la partie « presentation » du document UIML. La logique d'application, c'est-à-dire les actions en réponse aux interactions utilisateurs (par exemple, appel à un script ou à une méthode sur un objet), est spécifiée dans la partie appelée « logic ».

Enfin, la description UIML est soit interprétée sur le dispositif client (de la même manière qu'un navigateur Web interprète une page HTML) soit compilée vers un autre langage tel que WML ou HTML.

Conscient qu'UIML n'est pas suffisant à lui seul pour le développement d'IU multi plateformes, les auteurs proposent dans [Ali 03] un processus de conception composé de trois étapes : Modèle logique, Modèle de famille, Modèle spécifique à une plateforme (voir Figure 6).

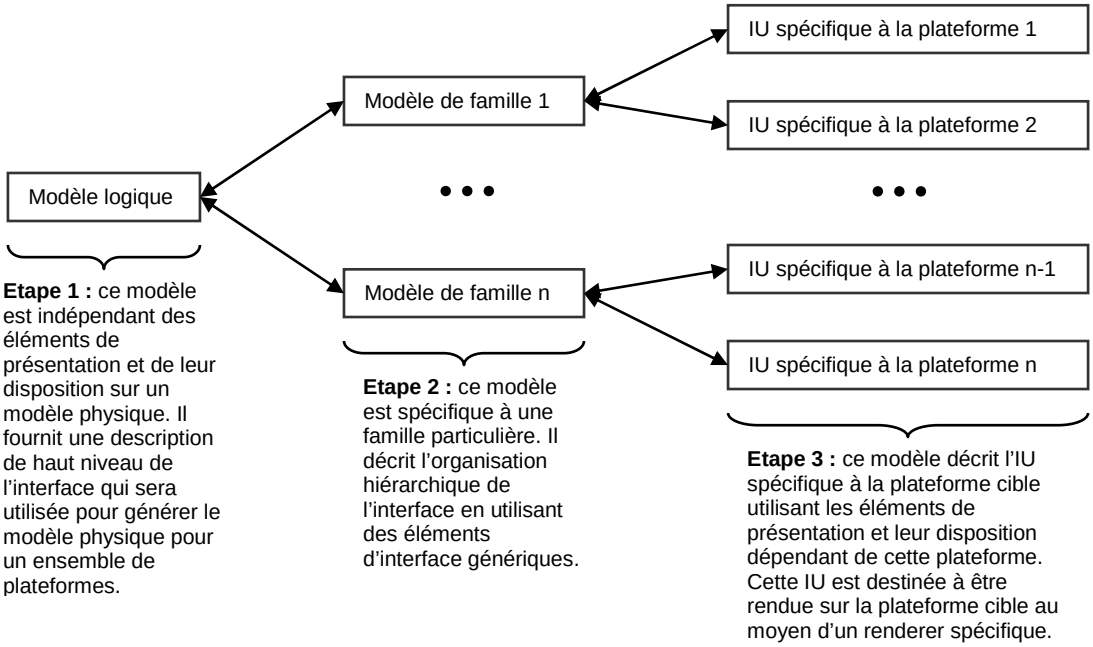


Figure 6 – Processus de conception adopté par UIML pour une IU multi plateformes [Ali 03]

Dans l'étape *Modèle logique*, l'objectif est de capturer la description de l'IU à un haut niveau d'abstraction. Cette description est donnée par un modèle hybride de tâche et de domaine.

Dans l'étape *Modèle de famille*, des modèles spécifiques à une famille sont décrits. Une famille est un ensemble de plateformes ayant un mécanisme de mise en page (layout) similaire. Dans ces modèles, les éléments d'interfaces manipulés sont génériques.

Dans l'étape *Modèle spécifique à une plateforme*, l'IU spécifique à une plateforme est décrite. Les éléments de présentation ainsi que leur disposition dépendent de la plateforme. Cette IU permettra d'obtenir l'interface utilisateur finale par rendu.

L'obtention de l'IU finale n'apparaît pas dans le cadre de référence proposé bien qu'elle soit obtenue à la suite de l'étape « Modèle spécifique à une plateforme ». Nous rajoutons, pour notre étude, une quatrième étape que nous appellerons *Interface Utilisateur finale*.

Les artefacts produits et utilisés dans la méthode UIML sont présentés dans le Tableau 6.

Tableau 6 – Artefacts produits dans le cycle de vie selon la méthode UIML [Abrams 99]

Etapes	Modèle logique	Modèle de famille	Modèle spécifique à une plateforme	Interface Utilisateur finale
Artefacts produits	Modèle hybride de tâche et du domaine	IU ne contenant que des éléments d'interfaces génériques.	IU pouvant contenir des éléments d'interfaces spécifiques à une plateforme.	Interface Utilisateur Finale

2.2.3.3. UsiXML

UsiXML (USer Interface eXtensible Markup Language) [Limbourg 04] est un langage XML de description d'une interface utilisateur pour différents contextes d'utilisation telles que les interfaces graphiques, tactiles, sonores ou multimodales. Ce langage s'inscrit dans une méthode dite de développement multidirectionnel d'interface utilisateur. Ce type de développement nécessite un cadre de référence de base, c'est-à-dire un processus de conception composé de plusieurs étapes adapté au développement d'applications interactives multi contextes.

UsiXML se base sur les travaux du projet Cameleon [Calvary 03] (voir Figure 7) dans lequel le processus de conception est constitué des quatre étapes suivantes pour deux contextes d'utilisation différents : Tâche et Concepts, UI abstraite, UI concrète et UI finale.

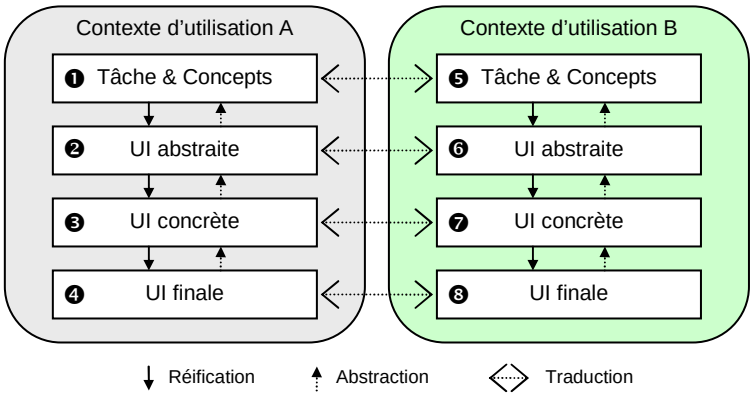


Figure 7 – Cadre de référence Cameleon [Calvary 03]

Dans l'étape *Tâche & Concepts*, les tâches diverses qui doivent être effectuées et les concepts du domaine sont décrits. Ces objets sont considérés comme des instances de classes représentant les concepts manipulés.

Dans l'étape *UI abstraite*, l'interface utilisateur abstraite (IUA) est conçue : les espaces d'interaction (ou unités de présentation) sont définis en groupant les sous-tâches selon plusieurs critères (par exemple, patrons structurels d'un modèle de tâche, analyse de la charge cognitive, identification des relations sémantiques) ; un schéma de navigation entre les espaces d'interaction est également défini ; et les objets abstraits d'interaction (AIO [Vanderdonckt 93]) sont sélectionnés pour chaque concept de manière qu'ils soient indépendants d'une modalité. Une IUA est une réification des tâches et des concepts du domaine mais également une abstraction d'une interface utilisateur concrète en rendant cette dernière indépendante d'une modalité d'interaction (par exemple, interaction graphique, interaction vocale, synthèse et reconnaissance vocale, interaction vidéo, virtuelle, augmentée ou réalité mixte).

Dans l'étape *UI concrète*, l'interface utilisateur concrète (IUC) est conçue et est constituée d'objets d'interaction concrets (CIO [Vanderdonckt 93]) pour un contexte d'utilisation donné afin de définir la disposition des widgets ainsi que la navigation dans l'interface. Une IUC peut être considérée comme une réification d'une IUA au niveau supérieur et une abstraction d'une interface utilisateur finale (IUF) selon la plateforme. L'IUF est abstraite en une définition de l'interface utilisateur qui est indépendante d'une plateforme particulière. Bien que l'IUF rende explicite le look and feel final pour une IUF, ce n'est encore qu'un prototype qui ne s'exécute que dans un environnement particulier.

Dans l'étape *UI finale*, nous trouvons l'interface utilisateur finale (IUF), à savoir une interface opérationnelle exécutable sur une plateforme particulière (navigateur Web, système d'exploitation, etc.).

Entre chaque étape de ce processus, des transformations sont effectuées. Des transformations peuvent aussi intervenir entre étapes de même niveau mais de contexte d'utilisation différent. Les trois transformations de base sont l'*abstraction*, la *réification* et la *traduction* (voir Figure 7). L'abstraction est un processus d'obtention d'artefacts plus abstraits que ceux utilisés dans la phase courante. La réification est le processus inverse, à savoir, l'obtention d'artefacts plus concrets que ceux utilisés dans la phase courante. La traduction est un processus d'obtention d'artefacts prévus pour un contexte d'utilisation particulier à partir d'artefacts d'une étape de développement similaire mais dont le contexte d'utilisation est différent.

UsiXML intervient dans les différentes étapes de ce processus en proposant une collection de modèles : un modèle de tâche (ici, CTT [Paternò 97] utilisé dans l'étape *Tâche & Concepts*),

un modèle du domaine (étape *Tâche & Concepts*), un modèle de l'interface abstraite (étape *UI abstraite*), un modèle de l'interface concrète (étape *UI concrète*), un modèle de transformation, un modèle de mapping, et un modèle du contexte (ces trois derniers modèles sont utilisés tout au long du processus). L'ensemble de ces modèles couvre ainsi toutes les étapes du processus.

Les artefacts produits et utilisés dans la méthode UsiXML sont présentés dans le Tableau 7.

Tableau 7 – Artefacts produits dans le cycle de vie selon la méthode UsiXML [Limbourg 04]

Etapes	Tâche & Concepts	UI abstraite	UI concrète	UI finale
Artefacts produits	Modèle de transformation, Modèle de mapping, Modèle de contexte			
	Modèles de tâche, Modèle du domaine	Interface Utilisateur Abstraite (IUA) composée d'AIOs	Interface Utilisateur Concrète (IUC) composée de CIOs	Interface Utilisateur Finale (IUF)

2.2.4. Synthèse

Nous avons vu dans cette section différents processus et méthodes de conception Web. Le Tableau 8 (voir page 28) rassemble les artefacts produits par chaque méthode présentée dans le processus de conception. Les différents processus de conception détaillés tout au long de cette section ont été replacés par rapport à celui de Scapin et al. [Scapin 00a] que nous avons choisi comme référence pour cette discussion (voir section 2.2.1 pour plus de détails sur ce processus de conception).

Une première observation est que certaines méthodes ne couvrent pas le cycle de vie en entier, ceci est particulièrement vrai pour les méthodes basées sur des langages de description d'interface (par exemple, UsiXML [Limbourg 04] ou Teresa [Paternò 03]) qui ne traitent pas, par exemple, des phases d'utilisation & évaluation de l'interface utilisateur finale et de maintenance.

En outre, chaque processus de conception Web est différent des autres sur plusieurs points : l'activité menée une même étape peut être différente (par exemple, dans l'étape se spécification de l'application, appelé « Modélisation conceptuelle », de la méthode OOHDM [Schwabe 98], l'activité porte sur la modélisation du domaine, alors que dans la même étape pour Teresa [Paternò 03], l'activité porte sur la modélisation de la tâche) ; certaines étapes sont divisées en sous étapes (par exemple, WebML [Ceri 00] divise l'étape de conception en quatre étapes, alors que dans OOHDM [Schwabe 98], cette même étape est divisée en deux sous étapes) ; certaines méthodes regroupent des étapes en une seule (par exemple, eW3DT [Scharl 99] regroupe les étapes de spécification et de conception en une seule appelée « conception »).

Si tous ces processus de conception sont différents, ils partagent cependant une caractéristique commune, celle de produire et d'utiliser des artefacts dans chacune des activités. Ces artefacts sont construits et élaborés dans l'objectif d'aider au développement de l'application Web finale en servant de base et de support, en permettant de diriger la conception et parfois, de générer une partie de l'application.

Ces artefacts peuvent être de plusieurs natures : des documents papiers (définition des besoins, document de spécification, etc.), des modèles (de tâche, de domaine, de navigation, de structure, etc.), des prototypes, les pages Web finales, des données d'usage (ou traces de navigation obtenues pendant l'utilisation), etc.

Tableau 8 – Artefacts produits dans les différentes étapes du processus de conception

		Etapes du cycle de vie d'une application Web [Scapin 00a]								
		Expression des besoins	**Spécification**	**Conception**				**Développement**	**Utilisation & Evaluation**	**Maintenance**
Méthodes basées sur des modèles	**OOHDM**		**Modélisation conceptuelle**	**Conception de la navigation**		**Conception de l'interface abstraite**		**Implémentation**		
			Classes, sous systèmes, relations, attributs, perspectives	Nœuds, liens, structures d'accès, contextes de navigation, transformations navigationnelles		Objets de l'interface abstraite, réponses aux événements externes, transformations de l'interface		Pages Web		
	WebML	**Collecte des besoins**	**Conception des données**	**Conception de l'hypertexte**	**Conception de la présentation**	**Conception des utilisateurs et groupes d'utilisateurs**	**Conception de la personnalisation**			
		Objectifs du site, Utilisateurs cibles, Exemples de contenu, Guides de styles, Personnalisation, Contraintes	Entités, attributs, composants, liens entre entités	Modèle de composition, Modèle de navigation	Feuilles de styles typées, Feuilles de styles non typées	Profil Utilisateur	Entités, Attributs, Composants et Règles métier			
	UWE	**Analyse des besoins**	**Modélisation conceptuelle**	**Modélisation de la navigation**		**Modélisation de la présentation**				
		Cas d'utilisation en UML	Diagrammes de classes UML	Diagrammes de classes UML enrichis avec des classes et des structures de navigation		Classes de présentation (ou template)				
	eW3DT	**Analyse**	**Conception**					**Implémentation**	**Utilisation**	
		Modèle d'analyse	Modèle de référence					Pages Web	-	
Méthodes basées sur des langages de description d'interface	**Teresa**		**Modélisation haut niveau de la tâche**	**Développement du modèle de tâche système**		**Transformation d'un modèle de tâche système vers une IU abstraite**		**Génération de l'Interface Utilisateur**		
			Modèle de tâche, Modèle du domaine	Modèle(s) de tâche système		Interface Utilisateur Abstraite composée d'AIOs		Interface Utilisateur Finale		
	UIML		**Modèle logique**	**Modèle de famille**		**Modèle spécifique à une plateforme**		**Interface Utilisateur Finale**		
			Modèle hybride de tâche et du domaine	IU ne contenant que des éléments d'interfaces génériques.		IU pouvant contenir des éléments d'interfaces spécifiques à une plateforme.		Interface Utilisateur Finale		
	UsiXML		**Tâche & Concepts**	**UI abstraite**		**UI concrète**		**UI finale**		
			Modèle de transformation, Modèle de mapping, Modèle de contexte							
			Modèles de tâche, Modèle du domaine	Interface Utilisateur Abstraite (IUA) composée d'AIOs		Interface Utilisateur Concrète (IUC) composée de CIOs		Interface Utilisateur Finale (IUF)		

Chaque artefact traite d'un aspect particulier de l'application Web, par exemple, un modèle de tâche décrit les tâches de l'utilisateur avec le système (Teresa [Paternò 03]), un modèle de navigation décrit les connexions entre les différentes pages de l'application ([Fleming 98]), une interface abstraite sert à décrire les différents éléments composant une page Web (UsiXML [Limbourg 04]), un prototype s'attache à tester certaines fonctionnalités précises de l'application ([Lynch & Horton 02]).

Ces artefacts s'avèrent donc être une représentation abstraite ou partielle de l'application.

Du point de vue de l'évaluation ergonomique, nous remarquons que ces artefacts sont déjà porteurs d'une partie des informations évaluables. Par exemple, une interface abstraite telle que proposée par UsiXML [Limbourg 04] est assez riche pour pouvoir mener une évaluation et détecter des problèmes d'utilisabilité ou d'accessibilité puisque la structure d'une page Web est pleinement renseignée via la composition des objets d'interactions abstraits (ou concrets) dans l'interface ; un modèle de navigation spécifie le comportement de l'interface, c'est-à-dire la réponse à l'utilisateur lors de l'activation des liens (page cible, paramètres envoyés, etc.) ; un prototype fonctionnel peut être évalué par un expert en ergonomie qui naviguerait dans ce prototype.

Certains artefacts, comme dit précédemment, se focalisent sur un aspect donné de l'application, par exemple, la navigation ou la structure d'une page. En outre, ces artefacts ne sont qu'une représentation partielle ou abstraite de l'application finale. Par conséquent, l'ensemble de toutes les règles ergonomiques ne peut pas être évalué, d'une part, sur un seul et même artefact et d'autre part, à chaque étape du processus de conception. Ainsi, une évaluation ergonomique couvrant au maximum les connaissances ergonomiques à notre disposition doit se faire sur l'ensemble de ces artefacts.

2.3. OUTILS DE SUPPORT BASES SUR DES CONNAISSANCES ERGONOMIQUES

Les outils de support basés sur des connaissances ergonomiques sont nombreux et peuvent se diviser en deux catégories : les outils d'aide à la conception, dans lesquels les connaissances sont appliquées pendant la conception de l'application, et les outils d'aide à l'évaluation destinés à évaluer l'application finale conformément à ces connaissances. L'utilisation de ces outils est importante pour réduire les coûts et améliorer ainsi la productivité. Cependant, un outil peut être choisi pour diverses raisons, par exemple, pour des raisons de flexibilité dans la gestion des connaissances, pour des raisons de fonctionnalités offertes, etc. Ces outils peuvent être comparés sur plusieurs points :

- **réutilisabilité :** l'outil peut-il être utilisé dans différentes étapes du processus de conception ? est-il lié ou non à un environnement de conception ? etc. ;
- **fonctionnalités :** l'outil permet-il l'évaluation de l'application conformément aux règles ergonomiques ? permet-il la correction automatique des erreurs identifiées ? permet-il une correction assistée ? etc. ;
- **automatisation :** l'outil permet-il de faire de l'évaluation automatique ? permet-il de faire de l'évaluation semi-automatique ? gère-t-il les règles dont l'évaluation est manuelle ? etc. ;
- **flexibilité :** l'outil intègre-t-il plusieurs ensembles de règles ? permet-il de les gérer? autorise-t-il la sélection des règles à évaluer ? etc. ;
- **compréhensibilité :** le rapport d'évaluation généré par l'outil est-il compréhensible ? est-il configurable ? etc.

Nous présentons tout d'abord dans cette section les dimensions considérées pour chaque outil, avant de présenter respectivement dans les sections 2.3.9 et 2.3.10 les outils d'aide à la

conception et les outils d'aide à l'évaluation. Enfin, la section 2.3.11 présente une synthèse de ces différents outils.

2.3.1. Etape du Processus de conception

Chaque étape du processus de conception résulte en la production de plusieurs artefacts (voir section 2.2.1). Ces artefacts peuvent faire l'objet de vérifications conformément à des recommandations ergonomiques. Les outils d'aide à l'évaluation sont par conséquent susceptibles d'intervenir dans les différentes phases du processus de conception. A l'heure actuelle, il n'existe pas de processus de conception standard Web comme nous avons pu le voir dans la section 2.2.1. Le cycle de vie en O de Scapin et al. [Scapin 00a] introduit lors de cette section nous servira de base à la discussion sur les outils. Ce processus de conception est composé des six phases suivantes : *expression des besoins*, *spécification du site*, *conception du site*, *développement du site*, *utilisation & évaluation du site*, et *maintenance du site*. Le lecteur pourra se référer à la section concernée pour le détail de chacune de ces phases.

2.3.2. Fonctionnalités

Les outils peuvent offrir plusieurs fonctionnalités : le guidage, la génération automatique, l'évaluation d'une ou de plusieurs pages Web, la réparation automatique des erreurs détectées, et dans certains cas, une réparation assistée quand cela est possible, et enfin, un accès à des ressources supplémentaires expliquant la cause des erreurs :

- **Guidage –** Le guidage aide le concepteur dans sa tâche afin de minimiser le risque d'erreurs ou de problèmes d'ergonomie sur l'interface finale. Le guidage consiste, par exemple, en une aide interactive ou une proposition d'options comme des options d'accessibilité, lorsque l'application est en cours de conception.
- **Génération automatique –** La génération automatique est une fonctionnalité permettant d'obtenir l'application ou une partie de l'application par génération. La génération permet d'augmenter la productivité mais également de se conformer aux règles ergonomiques en générant par exemple des structures tels que des tableaux directement accessibles (conformes aux directives d'accessibilité).
- **Evaluation** – L'évaluation automatique des règles ergonomiques permet de détecter les problèmes sur une ou plusieurs pages Web.
- **Réparation automatique** – La fonctionnalité de réparation automatique permet de corriger automatiquement certaines erreurs identifiées lors de l'évaluation. Cette correction est seulement possible sur une catégorie d'erreurs, les plus fréquemment rencontrées étant les erreurs de syntaxe dans le code source ou les attributs manquants pour une balise, par exemple dans un document HTML.
- **Réparation assistée** – La réparation assistée consiste en la réparation d'une erreur par un évaluateur humain, lui-même aidé par l'outil. Cette aide peut être fournie de plusieurs manières : navigation d'erreurs en erreurs, saisie d'un texte alternatif sur une image par l'utilisateur et insertion automatique dans le code source par l'outil, etc.
- **Accès à des ressources pour une évaluation/réparation manuelle** – L'outil aide l'évaluateur dans l'évaluation manuelle des règles en fournissant des ressources qui guideront ce dernier dans cette tâche. Ces ressources peuvent être une documentation en format texte ou hypertexte, en local ou en ligne.

2.3.3. Dimensions traitées par les règles ergonomiques

Les règles intégrées initialement aux outils peuvent traiter de plusieurs dimensions telles que l'utilisabilité ou l'accessibilité. Ainsi, selon l'outil, l'évaluateur aura la possibilité d'évaluer une ou plusieurs dimensions de l'ergonomie de l'application Web. Dans la section 2.1.3, nous nous sommes attachés à présenter les différentes dimensions traitées par les règles

ergonomiques. Nous reprenons ici les dimensions mentionnées dans cette précédente section, à savoir : l'*utilisabilité*, l'*accessibilité*, la *mobilité* et les *bonnes pratiques d'un langage informatique*.

2.3.4. Gestion des règles ergonomiques

Les règles ergonomiques d'un guide de recommandations sont nombreuses et nécessitent d'être gérées. Comme nous l'avons vu dans la section 2.1.5, parmi les outils existants, certains sont entièrement dédiés à la gestion des règles, alors que d'autres proposent en plus de les évaluer. Cette gestion peut se faire de plusieurs manières au sein des outils :

- Aucune gestion possible (les règles sont codées en dur dans l'outil) ;
- Possibilité d'ajouter, de supprimer et de modifier une règle ;
- Possibilité d'ajouter, de supprimer et de modifier un corpus de règles ;
- Possibilité de sélectionner les règles à évaluer
- Possibilité de sélectionner les corpus de règles à évaluer

Le lecteur pourra se reporter à la section 2.1.5 pour une description plus détaillée des différentes possibilités de gestion.

2.3.5. Type de vérification

Les outils d'aide à l'évaluation peuvent procéder à différents types de vérification. Ces vérifications se divisent en quatre catégories : *analyse de la structure du site et des pages Web*, *vérification des propriétés physiques*, *analyse linguistique du contenu*, et *validation syntaxique du code source*.

- **Analyse de la structure du site et des pages Web** – L'analyse de la structure du site ou d'une page Web permet de détecter des problèmes de navigation et/ou des incohérences dans la structure des pages Web. Au niveau du site Web, il s'agit de vérifier la navigation dans le site (par exemple, vérifier que toute page contient au moins un lien vers une autre page du site). En ce qui concerne la composition structurelle d'une page Web, il s'agit de déterminer si le contenu est bien structuré (par exemple, Titre/Chapitre/Section dans cet ordre).
- **Vérification de propriétés physiques** – Les propriétés mesurables et auxquelles nous pouvons attribuer une valeur numérique, un pourcentage ou une valeur booléenne, sont appelées propriétés physiques. Ces propriétés peuvent être comparées à des valeurs de référence. Par exemple, nous pouvons vérifier que le nombre de liens dans une page ne dépasse pas un certain nombre, de même pour la taille en Ko des images, le temps de téléchargement, etc.
- **Analyse linguistique du contenu** – L'analyse linguistique du contenu consiste en une aide à la détection d'erreurs lexicales et/ou syntaxiques du contenu. Cette analyse se fait au moyen d'un dictionnaire de termes : des alertes sont lancées lorsqu'un terme est mal orthographié ou n'a aucune entrée dans le dictionnaire.
- **Validateur syntaxique du code source** – Ce type de validation a pour objectif de valider que le code source est syntaxiquement correct. Nous entendons par « syntaxiquement correct » un code source qui respecte la syntaxe du langage (par exemple, la DTD[8] du langage XHTML), mais également et dans la majorité des cas, un code dont la syntaxe est conforme aux règles ergonomiques (par exemple, en HTML, une image sans attribut « alt » peut être considérée comme une erreur de syntaxe, ce point de vue est discuté dans [Centeno 07]).

[8] Acronyme de « Document Type Definition », document permettant de décrire un modèle de document SGML ou XML.

2.3.6. Technique de vérification

Les vérifications décrites précédemment peuvent être réalisées avec des techniques différentes. Par exemple, calculer le nombre de liens dans une application Web peut se faire en analysant un modèle de l'application mais également en analysant le code source de l'application réelle. Nous avons distingué quatre techniques de vérification : *l'analyse de modèle*, *l'analyse statique du code source*, *l'analyse de logs*, et la *génération automatique*.

- **Analyse de modèle** – Cette technique permet d'identifier des erreurs sur une représentation abstraite du système, c'est-à-dire sur un modèle de l'application. Ces modèles servant de base à l'implémentation et/ou à la génération partielle de l'application finale peuvent se prêter à des vérifications automatiques (par exemple certaines propriétés physiques telles que le nombre de liens contenus dans une page peuvent être évaluées sur un modèle de navigation).
- **Analyse statique du code source** – L'analyse statique du code source consiste en l'analyse des données extraites du code source. Pour les applications Web, la plupart des informations sont extraites du code source HTML. Certaines informations sont issues des feuilles de styles CSS.
- **Analyse de logs** – Cette technique consiste en l'analyse des traces générées lors de l'utilisation du système. Concrètement, l'utilisation de l'application par une personne permet de récolter des données tels que les clics souris, les entrées clavier, la date et l'heure, les pages visitées, etc. Ces données seront analysées par un outil qui est capable d'interpréter les fichiers de log.
- **Génération automatique** – La génération automatique permet de générer du code accessible lors de la construction d'une page ou d'un site à l'aide d'un outil (par exemple, l'insertion d'un tableau accessible dans une page). La règle d'accessibilité correspondante est ainsi vérifiée par construction. La génération automatique n'est toutefois possible qu'avec un outil d'aide à la conception (voir section 2.3.9).

2.3.7. Catégorie

Plusieurs catégories d'outils de conception et d'évaluation existent : ceux qui correspondant à une application autonome installée sur un monoposte, ceux qui sont intégrés à un environnement de conception, les plug-ins, ou alors les services Web qui ne nécessitent aucune installation.

- **Outil autonome** – Un outil autonome est un outil installé sur un poste (ou sur un serveur). Il est utilisable de manière indépendante, c'est-à-dire qu'il n'y a aucune nécessité de s'intégrer à un environnement de conception pour utiliser l'outil.
- **Outil intégré à un environnement de conception** – Les outils appartenant à cette catégorie sont liés à un environnement de conception et n'existent pas sans ce dernier.
- **Plug-in** – Les outils d'évaluation sous forme de plug-in se greffent à d'autres outils ou à un environnement de conception pour être fonctionnels. S'ils dépendent de ces outils ou environnement pour pouvoir fonctionner, ils ne leur sont cependant pas nécessairement dédiés.
- **Service Web** – Un outil d'évaluation peut également être un service Web, c'est-à-dire un outil accessible à partir d'un navigateur Web et ne nécessitant aucune installation. Il prend généralement la forme d'un formulaire contenant un champ de saisie dans lequel l'adresse du site à évaluer est renseignée. L'outil procède alors à l'évaluation du site en question.

2.3.8. Rapport d'évaluation

L'évaluation d'une application par un outil génère en sortie un rapport d'évaluation. Celui-ci est important car il est à la base des corrections à effectuer pour atteindre un certain niveau de conformité à des guides de recommandations. Pour l'évaluateur, il doit être clair, concis et bien organisé, afin que les modifications à apporter en soient facilitées. Si dans certains cas, le rapport d'évaluation ne peut pas être sauvegardé et ne peut pas être exploité par un évaluateur, certains outils offrent cependant la possibilité d'enregistrer les rapports. Les avantages ainsi procurés sont multiples : relecture du rapport sans relancer l'évaluation, comparaison entre rapports effectués à différentes dates, échange des rapports d'évaluation entre outils lorsqu'ils partagent un format commun (par exemple, le format EARL [EARL] proposé par le W3C). Les outils existants peuvent proposer :

- **Un paramétrage du rapport d'évaluation :** le rapport est paramétrable par l'utilisateur qui peut ainsi l'afficher selon ses préférences. Les erreurs peuvent ainsi être classées par guide de recommandation évalué, par niveau de sévérité, par objet de l'interface, etc.
- **Différents formats :** le rapport d'évaluation peut être généré dans plusieurs formats tels que XML, HTML, PDF, texte brut, etc. Selon le format utilisé, l'évaluateur peut ou non interagir avec le rapport pour identifier les erreurs et leurs emplacements et pour obtenir de l'aide supplémentaire afin de corriger celles-ci.

2.3.9. Outils d'aide à la conception

Les outils d'aide à la conception assistent l'équipe de conception et de développement en appliquant automatiquement une partie des recommandations ergonomiques pendant la conception de l'application Web. Cette automatisation a pour avantage de prévenir les éventuelles erreurs liées aux recommandations, ceci afin d'éviter au maximum les corrections dans les phases ultérieures d'évaluation. Les fonctionnalités offertes par ces outils sont :

- Ils produisent du code correct par construction (c'est-à-dire conforme aux règles ergonomiques) : par exemple, dans le cas de règles d'accessibilité, les tableaux et formulaires insérés dans une page Web seront automatiquement accessibles (les tableaux contiennent des cellules d'en-têtes, une légende, un titre et les formulaires contiennent des groupes de champs avec pour chaque champ un libellé qui lui est lié, etc.) ;
- Ils valident le code à la volée (lorsque l'outil n'a pas le contrôle sur le code écrit par l'utilisateur) : par exemple, dans le cas de l'accessibilité, certaines balises HTML ont été déclarées inaccessibles par le W3C ; si de telles balises sont utilisées, l'outil affichera des erreurs. Nous pouvons noter que ces techniques de vérification peuvent être assimilées à du model checking [Brambilla 07] dans le cas d'un outil basé sur des modèles ;
- Ils guident le concepteur : par exemple, concernant l'accessibilité, lorsqu'un lien est inséré, une fenêtre de dialogue est affichée par l'outil pour demander au concepteur de rentrer une alternative textuelle (règle d'accessibilité pour les utilisateurs malvoyants qui auraient du mal ou seraient dans l'incapacité de visualiser l'image en question) ;

Dans cette section, nous présentons les outils d'aide à la conception existants.

2.3.9.1. Adobe Dreamweaver

Adobe® Dreamweaver® [Adobe Dreamweaver] est un éditeur WYSIWYG[9] permettant de développer et de gérer une application Web. Cet outil autorise la création de pages Web sans connaissances approfondies du langage HTML (le développeur compose sa page en éditant et

[9] Acronyme de « What You See Is What You Get ». Cet acronyme désigne un type d'éditeurs permettant de composer visuellement l'interface graphique (c'est-à-dire la page Web dans le cas des applications Web) telle qu'elle le sera au final.

en ajoutant directement le contenu depuis l'outil) et réduit ainsi la complexité et le temps de développement de l'application. Dreamweaver permet aussi d'éditer les feuilles de styles CSS, de générer de manière semi-automatique du code pour des structures complexes (tableaux, formulaires, etc.), et de mettre à disposition des blocs de scripts préconçus (Javascript, Flash, etc.).

Dreamweaver propose des options d'accessibilité pour aider le développeur pendant l'implémentation des pages Web. Dans certains cas, l'aide proposée est semi-automatique : elle consiste à demander au développeur de renseigner certaines informations sur l'objet qu'il s'apprête à insérer dans la page Web (par exemple, le texte alternatif sur une image). Dans d'autres cas, cette aide ne nécessite pas d'intervention de la part du développeur (règles dites intégrées, voir section 2.1.4) : l'élément sera inséré et rendu automatiquement accessible, c'est le cas, par exemple, des tableaux qui sont insérés automatiquement avec des cellules d'en-tête (un tableau accessible doit avoir des cellules d'en-tête). L'aide fournie se limite cependant aux éléments suivants : contrôles de formulaire, cadres, objets multimédia, images et tableaux. Pour les autres éléments, Dreamweaver donne accès à des références (document hypertexte) sur les recommandations pour l'accessibilité afin que le développeur puisse s'y référer pendant le développement de l'application. Ces recommandations sont issues de UsableNet[10].

Nous pouvons noter que Dreamweaver intègre des outils d'évaluation de l'accessibilité sur les pages Web en cours d'implémentation. Cependant, ces outils sont des extensions, c'est-à-dire des programmes appelés par Dreamweaver, qui ne sont pas utilisés pendant le développement des pages, mais une fois les pages développées. Nous pouvons les assimiler par conséquent à des outils d'aide à l'évaluation (voir section 2.3.10).

2.3.9.2. Microsoft Expression Web

Microsoft® Expression® Web [Microsoft Expression Web] est un éditeur WYSIWYG pour le développement d'applications Web, issu de la suite Microsoft® Expression®. Il remplace son prédécesseur Microsoft® FrontPage® 2003[11]. Cet outil permet de développer une application Web en composant visuellement chaque page Web telle qu'elle sera visualisée par l'utilisateur. Le code HTML est généré au fur et à mesure que la page est composée par le développeur, facilitant ainsi le développement de l'application (les connaissances avancées en HTML ne sont pas requises). Expression Web gère également les templates CSS et templates HTML, l'exploitation de données XML, les flux RSS, etc.

Des options d'accessibilité sont offertes au développeur pour l'aider à produire des pages Web accessibles. Cependant, l'outil ne va pas appliquer ces aides automatiquement lors de l'insertion des différents éléments dans la page. Ceci est à la charge du développeur qui doit être conscient que ces aides pour l'accessibilité existent et qu'il doit les utiliser sur les éléments appropriés. Par exemple, lorsqu'un tableau est inséré dans la page, le développeur doit désigner quelles cellules sont les cellules d'en-tête, ceci grâce à une aide fournie par l'outil qui se chargera en retour d'adapter le code source. Les éléments sur lesquels nous trouvons des options d'accessibilité sont les suivants : images, images cliquables, tableaux, contrôles de formulaire et cadres.

Expression Web inclut aussi un vérificateur d'accessibilité (accessibility checker) qui permet d'évaluer une ou plusieurs pages Web de l'application et d'afficher le rapport d'évaluation. Cependant, ce vérificateur n'est exécuté que sur une page Web déjà développée (cet outil ne s'exécute pas pendant que le développeur compose visuellement la page, c'est-à-dire pendant qu'il insère les différents éléments dans la page Web). Nous considérons par conséquent, tout comme pour Dreamweaver, qu'un tel vérificateur est un outil d'aide à l'évaluation (voir section 2.3.10).

[10] http://www.usablenet.com/
[11] http://office.microsoft.com/frontpage/

2.3.10. Outils d'aide à l'évaluation

Les outils d'aide à l'évaluation allègent le travail de l'évaluateur en inspectant l'application de manière automatique afin de détecter les problèmes relatifs à un ensemble de connaissances ergonomiques. L'utilisation de ces outils comporte beaucoup d'avantages :

- Ils accélèrent les phases d'évaluation de la qualité ergonomique de l'application : plusieurs évaluations peuvent ainsi être réalisées tandis qu'une évaluation avec un expert peut s'avérer longue et coûteuse ;
- Ils peuvent être utilisés par des évaluateurs non expert en ergonomie : l'expertise requise est intégrée dans l'outil ;
- Ils passent systématiquement en détail l'application entière (ou une partie de l'application) alors que dans le cas d'une évaluation faite par un expert, certaines erreurs ou oublis peuvent survenir pendant l'évaluation ;
- Ils suppriment les problèmes d'interprétation et d'ambiguïté propres aux règles ergonomiques ;
- Ils produisent des rapports d'évaluation très précis qui peuvent servir de documents techniques et qui peuvent mettre en lumière les problèmes d'ergonomie importants de l'application ;
- Ils effectuent pour certains d'entre eux une correction automatique des erreurs, réduisant ainsi le temps de développement nécessaire pour la mise en conformité avec les connaissances ergonomiques.

Cependant, il faut garder à l'esprit que même si ces outils font une grande partie du travail, certaines règles ne peuvent être entièrement évaluées par un outil automatique et l'intervention humaine reste indispensable, c'est typiquement le cas des règles semi-automatiques et manuelles. Les outils d'aide à l'évaluation existants sont présentés dans cette section.

2.3.10.1. A-Prompt

A-Prompt [A-Prompt] est un outil autonome d'évaluation et de réparation automatique/assistée de pages Web conformément aux règles d'accessibilité du WCAG 1.0 et/ou de la Section 508. La version en ligne, sous forme de service Web [ATRC], est plus récente et intègre les règles du Italian Stanca Act et les règles préliminaires du WCAG 2.

Cet outil analyse des documents HTML ou XHTML afin de détecter des violations dans les règles d'accessibilité. Par conséquent, l'application Web doit être développée avec le contenu pleinement renseigné. Cet outil ne peut donc être utilisé qu'en phase d'évaluation et d'utilisation du site.

L'évaluation de l'accessibilité peut se faire en sélectionnant un ou plusieurs corpus de règles (WCAG 1.0, WCAG 2, Section 508 ou Italian Stanca Act) mais la sélection individuelle des règles n'est pas possible. Ces règles sont toutes implémentées en dur dans l'outil.

A-Prompt permet l'évaluation automatique sur une partie des règles d'accessibilité mais aussi l'évaluation semi-automatique en aidant l'utilisateur lorsqu'une règle est violée. Par exemple, lorsqu'une image n'a pas d'alternative textuelle, l'outil affiche une fenêtre dans laquelle se trouve un aperçu de l'image et un champ que l'utilisateur devra remplir pour renseigner l'alternative textuelle. Enfin, A-Prompt affiche la liste des règles manuelles à vérifier par l'évaluateur.

Des fonctionnalités de correction automatique sont également offertes à l'évaluateur : par exemple, le concepteur indique la langue principale du document et l'outil active alors la correction automatique de la règle suivante : « Identifier le langage naturel principal du

document ». Le document sera corrigé automatiquement si la langue principale du document n'y est pas renseignée.

A la fin de l'évaluation, A-Prompt génère un rapport d'évaluation au format hypertexte dans sa version autonome et au format EARL dans sa version en ligne.

2.3.10.2. AccessEnable

AccessEnable [Brinck 02] est un outil d'évaluation de l'accessibilité d'une application Web se présentant sous la forme d'un service Web.

Cet outil intègre les règles d'accessibilité de la Section 508 et les WCAG 1.0 publiées par le W3C/WAI. L'ensemble des règles au sein de l'outil peut être facilement et rapidement enrichi pour pouvoir intégrer de nouveaux corpus ou intégrer, par exemple, un guide de style dédié à une entreprise. Cependant, les auteurs ne donnent pas d'informations supplémentaires sur la gestion des règles et cette fonctionnalité n'est pas offerte depuis l'interface Web.

L'analyse d'un site Web est effectuée en collectant toutes les pages Web grâce à un robot (en anglais, *Spider*[12]). Avec toutes les pages Web de l'application à sa disposition, AccessEnable est capable de détecter des problèmes liés à la structure du site (problèmes de navigation). Cet outil procède également à une analyse individuelle des pages. Il permet de détecter des problèmes de syntaxe du langage HTML (par exemple, détection du caractère '<' dans un texte, mauvaise utilisation des guillemets) ainsi que les règles d'accessibilité (Section 508 et WCAG 1.0).

L'analyse se fait côté serveur sur les copies téléchargées des pages Web. Lorsqu'une règle est violée, un tag est rajouté dans le code source à l'endroit précis où l'erreur a été identifiée. Ainsi, une fois l'évaluation terminée, un rapport d'évaluation est généré au format HTML avec la liste des problèmes identifiés et leur localisation (grâce à l'utilisation des tags).

La correction des erreurs identifiées peut être faite de manière automatique ou assistée. La correction automatique peut être paramétrée par l'évaluateur : pour une erreur donnée, celui-ci a le choix de lancer la correction automatique soit pour la page actuelle soit pour le site Web entier. La correction assistée permet à l'évaluateur de se faire aider par l'outil pour corriger les erreurs, par exemple, rajouter le texte alternatif sur une image. Cette aide lui donne ainsi la possibilité de naviguer d'erreur en erreur pour les corriger. Cependant, ce dernier peut choisir de ne pas corriger l'erreur et d'ajouter un commentaire depuis l'outil pour indiquer la raison de son choix. Ces commentaires seront alors enregistrés dans le rapport d'évaluation. Enfin, toutes les modifications non désirées peuvent être abandonnées si l'évaluateur souhaite revenir à la version précédente.

AccessEnable facilite l'évaluation manuelle, d'une part en affichant des alertes sur les règles qui ne peuvent pas être automatisées mais aussi en fournissant de la documentation pour aider l'évaluateur dans sa tâche.

2.3.10.3. DESTINE

DESTINE [Beirekdar 04] est un outil autonome d'aide à l'évaluation automatique de l'utilisabilité d'une application Web, l'évaluation consistant à une analyse statique du code HTML.

Cet outil intègre des recommandations d'accessibilité et d'utilisabilité dans une base de règles. Chaque règle est décrite au moyen d'un langage de haut niveau, le *Guideline Definition Language* (GDL) [Beirekdar 02], un langage XML permettant de modeler les aspects évaluables du code HTML. Les règles écrites avec ce langage dédié permettent la séparation de la base de règles du moteur d'évaluation : les règles ne sont donc pas inscrites en dur dans l'outil et il est donc possible d'ajouter, de supprimer et de modifier des règles.

[12] Un *Spider* est un robot parcourant toutes les pages d'une application Web et archivant ces pages dans une base de données.

Une fois les règles et les différents corpus de règles sélectionnés, le moteur d'évaluation évalue de manière automatique l'application conformément à ces règles. Lorsque des erreurs ont été détectées et quand cela est possible, une correction automatique, ou assistée, est proposée. Lorsqu'une évaluation automatique n'est pas possible, l'évaluateur peut accéder à des ressources pour une évaluation manuelle de l'application Web à partir du module de gestion des connaissances ergonomiques.

Le rapport d'évaluation est un ensemble de pages Web dans lequel l'évaluateur navigue pour visualiser les erreurs (l'évaluateur a aussi le choix de générer le rapport au format PDF). Ce rapport est configurable par l'évaluateur [Beirekdar 05] : les erreurs peuvent être affichées par source (par exemple, W3C/WAI, ou Section 508), par aspect ergonomique (utilisabilité ou accessibilité) ou par objet HTML (images, tables, etc.).

2.3.10.4. Doctor Watson

Doctor Watson [Doctor Watson] est un outil de diagnostic de pages Web disponible en ligne sous forme de service Web. Cet outil procède à des vérifications dans le but d'améliorer l'utilisabilité d'une page Web.

Ces vérifications consistent en une analyse syntaxique du code HTML (détection des éléments/attributs inconnus, détection des attributs obligatoires, etc.), une vérification des liens brisés (analyse de la structure du site), une analyse linguistique du texte contenu dans la page, une estimation du temps de téléchargement de la page (propriétés physiques), une analyse du nombre de mots contenu dans la page, une analyse de la visibilité par rapport aux moteurs de recherche, et une vérification de la popularité de la page (nombre de pages ayant un lien vers la page évaluée).

Bien que toutes ces vérifications soient codées en dur dans l'outil, l'évaluateur peut choisir d'effectuer les vérifications désirées en les sélectionnant avant de lancer l'évaluation. Lorsque l'évaluation est terminée, un rapport d'évaluation est affiché sous forme d'une page HTML.

2.3.10.5. EvalAccess

EvalAccess [Abascal 04] [Abascal 06] est un outil d'analyse statique du code HTML, sous forme de service Web, permettant d'évaluer l'accessibilité d'une application Web. Cet outil est une évolution de EvalIris [Abascal 04] développé par les mêmes auteurs.

Les règles d'accessibilité inclues dans EvalAccess sont celles des WCAG 1.0 [WCAG] éditées par le W3C/WAI. Ces règles ont été reformulées dans le langage appelé *Unified Guidelines Language* (UGL) [Arrue 07], un langage XML permettant de décrire les règles à un haut niveau d'abstraction dans l'objectif de les séparer du moteur d'évaluation. Ces règles sont ainsi stockées à part, dans une base de données XML.

EvalAccess permet de gérer plusieurs corpus de règles (fonctionnalités d'ajout, d'édition et de suppression de règles et de corpus de règles) mais ne donne pas la possibilité de sélectionner les règles désirées pour l'évaluation (pour ce dernier point, nous nous basons sur la version en ligne[13]).

Enfin, l'évaluation d'une application Web produit un rapport d'évaluation structuré lui aussi au format XML, dont le schéma est donné dans [Abascal 04].

2.3.10.6. HTML Toolbox

HTML Toolbox [NetMechanic], dans sa version gratuite, est un service Web qui permet l'évaluation de l'utilisabilité d'une application Web à partir d'une analyse statique du code HTML.

[13] La version en ligne de EvalAccess est disponible à l'adresse suivante : http://sipt07.si.ehu.es/evalaccess2/

Cet outil permet de détecter les balises dépréciées et les erreurs de syntaxe dans le code source (validation syntaxique), il effectue une analyse lexicale automatique dans onze langues prédéfinies avec possibilité pour l'évaluateur de personnaliser son propre dictionnaire (analyse linguistique), il estime le temps de chargement des différents éléments de la page (vérification de propriétés physiques), et vérifie les liens brisés (analyse de la structure). L'ensemble de ces vérifications est codé en dur dans l'outil.

Après une évaluation, certaines erreurs peuvent être réparées automatiquement grâce à l'outil appelé HTML Check, mais cette fonctionnalité n'est offerte que dans la version payante de HTML Toolbox. Enfin, le rapport d'évaluation généré est au format HTML.

2.3.10.7. HyperAT

HyperAT (Hypertext Authoring Tool) [Theng 98] est un outil autonome d'évaluation d'une application Web permettant d'effectuer une évaluation de l'utilisabilité selon 2 modes : une *analyse structurelle* et une *évaluation avec des utilisateurs réels*.

L'*analyse structurelle* calcule le nombre de pages, le nombre de liens par page, le nombre de chemins possibles d'une page vers une autre page, la profondeur maximale du site Web, et le nombre de successeurs d'une page. L'*évaluation avec des utilisateurs réels* consiste en l'analyse des données d'usage collectées (vérification de propriétés physiques) grâce à une technique d'analyse de logs : fréquence de visite d'une page, informations sur la navigation d'un utilisateur (les chemins empruntés), dates et heures de visite d'une page donnée. Toutes ces vérifications sont implémentées en dur dans l'outil et il est impossible de les paramétrer.

A la fin de l'évaluation, un rapport textuel est généré comparant les performances d'un utilisateur par rapport à l'objectif donné (c'est-à-dire la tâche).

2.3.10.8. LIFT Machine

LIFT Machine [LIFT Machine] est un outil d'évaluation de l'accessibilité et de l'utilisabilité d'une application Web. LIFT Machine fait partie d'une famille de six outils : LIFT Mobile, LIFT Text Transcoder, LIFT Machine, LIFT for Macromedia Dreamweaver, LIFT for Microsoft FrontPage et LIFT Online.

Sans entrer dans les détails, *LIFT Mobile* est un outil destiné aux applications Web sur PDA de type Blackberry, Treo et autre. *LIFT Text Transcoder* est un outil de transformation à la volée d'une page HTML en une version entièrement textuelle. *LIFT for Macromedia Dreamweaver* et *LIFT for Microsoft FrontPage* sont des versions de LIFT Machine (détaillé plus loin) qui s'intègrent dans ces deux environnements de conception. Enfin, *LIFT Online* est un service de souscription en ligne qui permet à l'évaluateur de bénéficier des mêmes services que LIFT Machine sans avoir besoin de posséder une machine serveur hébergeant ce dernier (l'évaluateur utilisera alors les services de UsableNet).

LIFT Machine est le seul outil, parmi ses six outils de cette famille, capable d'évaluer l'accessibilité et l'utilisabilité d'une application. Cet outil est une application qui s'exécute côté serveur et intègre environ 140 règles (en dur dans l'outil) : les règles W3C/WAI WCAG 1.0, les règles de la Section 508 et des règles supplémentaires d'utilisabilité concernant les problèmes de recherche, de navigation et d'utilisabilité sur le site.

Avant de procéder à l'évaluation, il est possible de sélectionner les règles désirées. Cette sélection peut être sauvegardée afin de pouvoir aisément exécuter plusieurs configurations de tests par la suite. L'évaluation d'une application Web est automatique et génère un rapport d'erreurs sous forme hypertexte (ce rapport est paramétrable par l'utilisateur). Enfin, une fois l'évaluation terminée, une correction automatique des erreurs détectées peut être lancée.

2.3.10.9. MAGENTA

MAGENTA[14] (Multi-Analysis of Guidelines by an ENhanced Tool for Accessibility) [Leporini 06] est un outil autonome d'évaluation de l'accessibilité et de l'utilisabilité d'une application Web. Cet outil intervient dans la phase d'évaluation et d'utilisation du site en analysant le code HTML et CSS des pages Web.

Les règles d'accessibilité et d'utilisabilité évaluées ne sont pas directement codées dans l'outil mais sont indépendantes et décrites dans le langage appelé *Guideline Abstraction Language* (GAL) défini par les auteurs. Ce langage XML permet de décrire les règles de manière abstraite et de les dissocier ainsi de l'outil. L'outil n'est alors qu'un support à l'évaluation des règles décrites en GAL. Détacher les règles de l'outil donne aussi à MAGENTA la possibilité de gérer plusieurs ensembles de règles. A l'heure actuelle, trois ensembles de règles sont implémentés par cet outil, à savoir, les règles d'accessibilité WCAG 1.0 du W3C/WAI, les règles d'accessibilité et d'utilisabilité pour les utilisateurs malvoyants [Leporini 04] et les règles recommandées par la loi pour les sites Web des services publics en Italie [LawStanca 04].

MAGENTA offre deux types d'évaluation : automatique et manuelle. L'évaluation automatique détecte automatiquement les erreurs d'accessibilité et d'utilisabilité dans les pages évaluées. L'outil effectue aussi une analyse lexicale du texte grâce à un dictionnaire des termes inappropriés tels que « Cliquez ici ». Lorsque des termes de ce dictionnaire sont identifiés dans la page, une erreur est enregistrée dans le rapport d'évaluation. Ce dictionnaire peut être personnalisé par l'évaluateur. Il est à noter qu'il y a un dictionnaire par langue. En ce qui concerne l'évaluation manuelle, elle est possible en intervenant directement sur le rapport d'évaluation : l'évaluateur peut annoter ce rapport avec des commentaires pour les règles manuelles.

Le rapport d'évaluation est décrit en langage RDL (Report Definition Language), un langage XML proposé par les auteurs. Ce format permet par ailleurs de contenir les mêmes informations que le format EARL [EARL] proposé par le groupe de travail W3C ERT (Evaluation and Repair Tools), groupe œuvrant pour un format unique pour les résultats d'évaluation automatique des outils pour l'accessibilité. En outre, le rapport d'évaluation est configurable par l'utilisateur.

Lorsque des erreurs sont détectées, une correction assistée peut être proposée par l'outil : par exemple, lorsqu'il faut vérifier l'intitulé d'un lien, l'outil amène directement l'évaluateur sur l'erreur (la balise HTML en question) et ce dernier a la charge de remplir correctement l'intitulé. Enfin, lorsqu'une correction assistée n'est pas possible, l'outil donne accès à des ressources supplémentaires directement depuis l'application pour aider à la correction manuelle des erreurs.

2.3.10.10. Ocawa

Ocawa (Outil de Contrôle et d'Analyse pour le Web Accessible) [Ocawa] est un outil d'aide à l'évaluation de l'accessibilité d'un site Web. Cet outil se présente sous la forme d'un service Web mais également sous la forme d'une barre d'outil (extension pour le navigateur Firefox 1.5 et plus). Cet outil existe en version gratuite (audit de cinq pages journalières), en version dite « web-service » (audits illimités) et en version « serveur » (un serveur Ocawa est installé chez le client, en général, une entreprise ayant des besoins importants). Les deux dernières versions citées ne sont pas gratuites.

Ocawa se présente sous la forme d'un système expert et utilise une base de connaissances où sont stockées les règles (voir Figure 8). Les règles sont donc séparées du moteur d'évaluation pour une meilleure gestion.

[14] Une première version de l'outil est sortie sous le nom de NAUTICUS [Correani 06]. Cet outil était destiné à améliorer l'accessibilité et l'utilisabilité des applications Web pour les utilisateurs ayant une déficience visuelle. Les recommandations intégrées dans cet outil sont un sous-ensemble de celles contenues dans MAGENTA [Leporini 06] à la différence qu'elles étaient codées en dur dans l'outil.

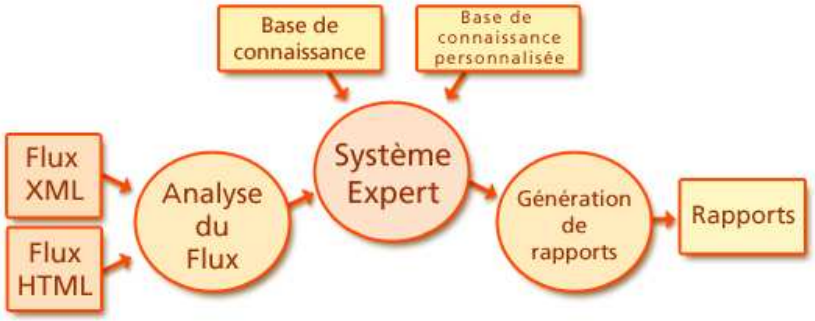


Figure 8 – Architecture de OCAWA [Ocawa]

Les règles intégrées dans Ocawa sont écrites dans un langage XML permettant une description de haut niveau, le OKWML (le format de ce langage n'est cependant pas public). L'ajout, la suppression et la modification des règles dans la base de connaissances d'Ocawa n'est possible que dans sa version « serveur ».

Cet outil procède à une analyse automatique du code HTML pour évaluer l'accessibilité conformément à plusieurs ensembles de recommandations : W3C/WAI WCAG 1.0, ADAE/DGME (Agence pour le Développement de l'Administration Electronique[15]), et France-Télécom (ces règles sont appelées *règles Ocawa* et ont été définies par un groupe de travail de France Télécom R&D. Ces règles sont basées sur les directives internationales du WAI, en y apportant une composante d'ergonomie). Chacun de ces corpus de règles peut être sélectionné individuellement avant l'évaluation.

Le rapport d'évaluation est généré au format HTML pour le service Web Ocawa. En ce qui concerne la barre d'outil Ocawa, l'audit d'une page est lancée directement sur la page visualisée dans le navigateur et le rapport d'évaluation est généré dans un panneau latéral à la page. Le rapport d'évaluation (toujours dans la version « barre d'outil ») permet de localiser l'élément sur lequel l'erreur a été détectée : cet élément HTML est mis en évidence lorsque l'évaluateur clique sur l'erreur concernée dans le rapport d'évaluation. En outre, pour faciliter la lecture du rapport d'évaluation, un filtre est mis à disposition afin de n'afficher qu'un sous-ensemble des erreurs détectées. Ce filtre consiste en un motif qui, une fois donné par l'évaluateur, permettra de sélectionner les erreurs respectant ce motif (par exemple, le motif 'font' sélectionnera les erreurs liées à la balise HTML 'font').

Enfin, nous pouvons noter que Ocawa ne propose pas de corrections automatiques.

2.3.10.11. ReWeb & TestWeb

ReWeb et TestWeb [Ricca & Tonella 01a] [Ricca & Tonella 01b] sont deux outils autonomes destinés à évaluer l'utilisabilité d'une application Web à travers la qualité fonctionnelle (détection des anomalies telles que des liens vers des pages inexistantes) et la qualité structurelle (analyse de la structure du site Web, par exemple, analyse des chemins possibles à travers le système).

ReWeb télécharge les pages d'une application Web et en construit un modèle UML (Reverse engineering). Ce modèle respecte un méta modèle défini par les auteurs. Une analyse statique est ensuite effectuée par cet outil. TestWeb, quant à lui, génère et exécute un ensemble de cas de test à partir du modèle construit par ReWeb.

Le processus d'évaluation est schématisé dans la Figure 9. Celui-ci est semi-automatique car l'intervention d'un humain est nécessaire (ces interventions sont modélisées par des losanges en haut de la figure).

[15] La DGME, acronyme de « Direction Générale de la Modernisation de l'Etat », est la structure ayant remplacé l'ADAE suite au décret n°2005-1792 datant du 30 décembre 2005. Cependant, même si l'ADAE n'existe plus, la désignation par « règles d'accessibilité de l'ADAE » reste encore très utilisée.

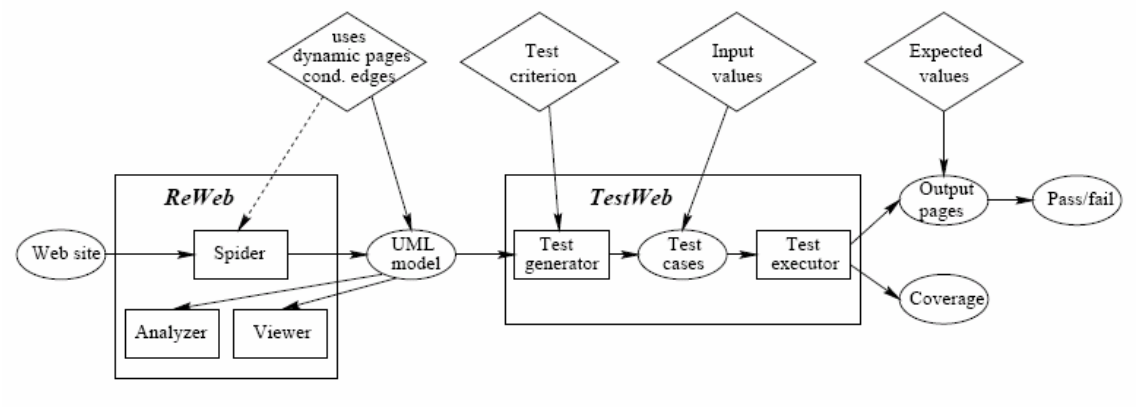


Figure 9 – Rôles de ReWeb et TestWeb dans le processus d'analyse et de test
[Ricca & Tonella 01a]

L'analyse statique effectuée par ReWeb consiste en la vérification des pages non atteignables, des pages fantômes (pages qui référencent des pages inexistantes), du nombre de frames dans lequel une page peut apparaître, de la dépendance des données (ou variables) entre pages, des dominateurs d'une page (ensemble des pages à franchir avant d'atteindre celle-ci), et du chemin le plus court pour atteindre une page.

La validation dynamique effectuée par TestWeb consiste en une technique de White-Box testing, à savoir, jouer plusieurs jeux de tests avec des données de départ sur l'application et comparer les résultats obtenus avec ceux attendus. Avec cette technique, TestWeb procède à plusieurs tests : test sur les pages (chaque page est traversée au moins une fois), test sur les liens (tous les liens de toutes les pages sont traversés au moins une fois), test sur la définition et l'utilisation d'une variable (tous les chemins de navigation engendrés par chaque variable sont empruntés), test sur l'utilisation complète d'une variable (au moins un chemin de navigation utilisant une variable et son utilisation est emprunté), et test sur tous les chemins (tous les chemins sont empruntés).

Toutes les vérifications effectuées par ReWeb et TestWeb sont codées en dur dans l'outil et il n'est pas possible ni de gérer ces règles de vérification ni de sélectionner les vérifications désirées.

Enfin, aucune information n'est donnée sur le format du rapport affichant les résultats de ces vérifications.

2.3.10.12. TAW

TAW (Test de Accesibilidad Web, ou Test d'Accessibilité Web) [TAW] est un outil d'évaluation automatique de l'accessibilité d'une application Web. Cet outil existe sous plusieurs formes : *TAW3 standalone* (outil autonome), *TAW3 Web Start* (outil lancé depuis un navigateur Web en utilisant la technologie Java Web Start[16]), *TAW3 online* (service Web) et *TAW3 with a click* (barre d'outil, extension à installer sur le navigateur Firefox). Nous ne présentons ici que la version autonome, TAW3 standalone (que nous appellerons TAW), puisqu'elle est la plus complète en termes de fonctionnalités et qu'elle est librement téléchargeable.

TAW intègre les règles d'accessibilité WCAG 1.0 du W3C/WAI. Cependant, l'évaluateur peut rajouter des règles dans l'outil s'il le désire. L'ajout de nouvelles règles se fait via une interface graphique de saisie, et l'écriture d'une règle se limite au langage HTML (vérification de la présence/absence d'un élément/attribut, vérification du contenu d'un attribut, etc.). Chaque règle est écrite dans un format propriétaire.

[16] Les détails sur la technologie *Java Web Start* sont disponibles à l'adresse suivante : http://java.sun.com/products/javawebstart/

Avant de lancer l'évaluation, l'évaluateur peut sélectionner les règles qu'il désire évaluer : par niveau de conformité A, AA ou AAA, ou selon une sélection personnalisée. Cette sélection peut être sauvegardée afin de permettre de créer plusieurs configurations de test et de lancer ces configurations de manière périodique.

L'évaluation peut être paramétrée pour être effectuée soit sur l'application Web entière, soit sur une page Web individuelle. L'évaluation est faite sur le code HTML des pages Web.

Une fois l'évaluation terminée, l'évaluateur a accès à trois types de rapport tous configurables : un rapport au format HTML, un rapport au format EARL [EARL] et un rapport succinct au format HTML. Ces rapports étant identiques (le rapport succinct est un résumé des informations contenues dans les deux autres rapports), nous ne présentons que le rapport d'évaluation au format HTML. Ce rapport regroupe les résultats de tests par niveau de conformité. Pour chaque règle d'accessibilité sont notés le nombre de problèmes identifiés automatiquement ainsi que le nombre de vérifications qui doivent être manuellement effectuées.

L'évaluation manuelle est facilitée par TAW : l'outil crée une liste des vérifications manuelles qu'il reste à effectuer. Une fois que l'évaluateur a procédé à une évaluation manuelle, la règle concernée peut être marquée dans cette liste avec les informations suivantes : succès, échec, non testé, non vérifiable, non applicable. Enfin, pour aider l'évaluateur à vérifier manuellement l'application conformément aux règles manuelles, celui-ci a accès, depuis l'outil, à des ressources supplémentaires.

TAW n'offre pas de fonctionnalités de corrections automatiques ou assistées.

2.3.10.13. WAEX

WAEX (Web Accessibility Evaluator in a single XSLT file) [Centeno 07] est un outil d'évaluation de l'accessibilité d'une application Web dont la particularité est de n'être qu'un simple fichier XSLT[17] dans lequel se trouvent toutes les règles d'accessibilité. L'évaluation de l'accessibilité consiste en l'application de ce fichier XSLT à un fichier XHTML donné en entrée. Ainsi, ce dernier est transformé (la transformation ne détruit pas le fichier XHTML source mais produit un autre fichier) conformément au fichier XSLT pour produire le rapport d'évaluation (en XHTML). Les règles d'accessibilité apparaissent sous la forme de templates XSLT[18] qui, lors de la phase de transformation, contribuent au rapport d'évaluation (ajout d'erreurs lorsqu'elles sont détectées).

Les avantages de l'utilisation du langage XSLT sont multiples :

- Les règles d'accessibilité sont exprimées de manière déclarative, elles ne sont donc pas codées en dur dans l'outil, un évaluateur peut librement changer l'écriture d'une règle s'il le désire ;
- Les règles sont plus simples à comprendre (une connaissance même minimale du langage XSLT est suffisante) que lorsqu'elles sont écrites dans un langage de haut niveau tel que le GDL [Beirekdar 02] ou le GAL [Leporini 06] ;
- L'outil est réutilisable ; si de nouvelles règles sont publiées, un évaluateur peut réutiliser l'outil pour intégrer celles-ci en modifiant le fichier XSLT ;
- L'outil est léger et peu coûteux comparé aux outils d'évaluation entièrement développés dans le même but.

WAEX intègre non seulement les règles d'accessibilité du W3C/WAI [WCAG] mais aussi celles du W3C's MobileOK Basic [mobileOK] liées à l'aspect mobilité des applications Web. Enfin, il faut toutefois noter que si WAEX est capable d'analyser des documents

[17] XSLT (eXtensible Stylesheet Language Transformations) est un langage de transformation XML de type fonctionnel. Les spécifications de ce langage (XSLT version 2.0, en date du 23 janvier 2007) sont disponibles à l'adresse suivante : http://www.w3.org/TR/xslt20/

[18] Un template est une règle de transformation XML dans le langage XSLT.

XHTML, il n'est pas capable par exemple d'évaluer une feuille de styles CSS (le langage XSLT n'est capable de traiter que des fichiers XML). Dans de tels cas, les règles d'accessibilité concernées sont vérifiées par appel à des routines (par exemple, le W3C CSS Validator).

2.3.10.14. Weblint

Weblint [Bowers 96] est un outil de validation du code source HTML. Il peut être utilisé comme un outil autonome ou comme un service Web.

L'analyse automatique d'une page Web permet de détecter des erreurs de type lexical (par exemple, utilisation d'une balise au mauvais endroit), de type syntaxique (par exemple, balise ou attribut d'une balise inconnu), d'utilisation (usage) du langage HTML (par exemple, utilisation d'une balise obsolète), d'intégrité structurelle (par exemple, détection des liens brisés), de portabilité du code HTML (par exemple, utilisation d'une balise spécifique à un navigateur), et stylistique (par exemple, étiquette des liens non explicite, fautes d'orthographe et de grammaire).

Ces vérifications sont codées en dur dans l'outil, cependant, l'évaluateur peut sélectionner les règles qu'il veut évaluer en éditant le fichier de configuration de Weblint : par défaut, toutes les vérifications sont activées mais l'évaluateur peut faire le choix d'en désactiver certaines.

Le rapport généré est au format textuel et indique les lignes où ont été détectées les différentes erreurs, ainsi qu'un texte expliquant ces erreurs. La correction manuelle de ces erreurs reste à la charge de l'évaluateur.

2.3.10.15. WebSAT

WebSAT (the Web Static Analyzer Tool) [NIST Web Metrics] est un outil autonome d'évaluation automatique de l'utilisabilité et de l'accessibilité d'une application Web à travers une analyse du code source HTML des pages Web.

Les règles automatiquement vérifiées par WebSAT sont codées en dur dans l'outil et proviennent de deux corpus de règles, à savoir un ensemble de règles d'utilisabilité et d'accessibilité propre à WebSAT et les règles issues du standard ISO Std 2001-1999.

Les règles ergonomiques propres à WebSAT se divisent en six catégories[19] : accessibilité (par exemple, vérification de l'alternative textuelle pour une image), utilisation de formulaire (par exemple, vérification de la présence d'un bouton pour soumettre le formulaire), performance (par exemple, vérification de la taille en Ko de la page), maintenabilité (par exemple, vérification de l'adresse relative des liens), navigation (par exemple, vérification de la présence d'au moins un lien dans la page), et lisibilité (par exemple, vérification de l'absence de lignes horizontales dans la page).

Les règles du standard ISO Std 2001-1999 sont divisées en deux catégories[20] : les règles concernant les informations sur l'en-tête d'une page HTML (par exemple, vérification de la présence de la balise *title*) et celles sur le corps de la page (par exemple, vérification de la présence d'aides à la navigation).

Le rapport généré suite à l'évaluation est au format HTML. Il affiche, pour chaque règle violée, un lien vers l'erreur dans le code source. La correction de ces erreurs reste manuelle.

2.3.10.16. WebXACT

WebXACT [WebXACT] est un outil d'évaluation de l'accessibilité d'une application Web à partir d'une analyse statique du code HTML et se présente sous la forme d'un service

[19] Une description des règles propres à WebSAT est donnée à l'adresse suivante : http://zing.ncsl.nist.gov/WebTools/WebSAT/websat_rules.html
[20] Une description de ces règles est donnée à l'adresse suivante : http://zing.ncsl.nist.gov/WebTools/WebSAT/ieee_guide.html

Web[21]. Cet outil supporte un sous-ensemble des fonctionnalités de deux outils nommés WebXM et WebQA.

Les règles évaluées automatiquement par cet outil sont les WCAG 1.0 publiées par le W3C/WAI, mais WebXACT intègre également des règles sur la qualité (contenu défectueux, informations sur la navigation, temps de chargement de la page, etc.) et la vie privée (informations sur la collecte des données, cookies, etc.). Toutes ces règles sont codées en dur et l'outil ne permet pas de rajouter ses propres règles ni même de choisir les règles qui seront évaluées.

L'évaluation génère un rapport au format HTML dans lequel l'utilisateur a accès à des précisions et des ressources supplémentaires sur la raison des erreurs détectées. Ces ressources vont également servir pour une évaluation manuelle de l'application pour les règles non automatisées. Aucune correction automatique n'est proposée par WebXACT.

2.3.10.17. L'outil de Takata et al. [Takata 04]

Takata et al. ont proposé dans [Takata 04] un outil autonome d'évaluation automatique de l'accessibilité d'une application Web.

Cet outil intègre les règles WCAG 1.0 du W3C/WAI. Toutes ces règles sont écrites dans un langage XML de haut niveau, le *Simple Guideline Specification Language* (SGSL), qui permet ainsi de séparer les règles du moteur d'évaluation et de les gérer (ajout, suppression, modification). Ce langage étend le langage XPath[22] en y introduisant les quantificateurs de la logique du premier ordre (c'est-à-dire 'pour tout' et 'il existe'). Les auteurs conjecturent que le langage SGSL ainsi défini a un pouvoir expressif plus grand que celui de XPath.

L'évaluation d'une application Web conformément aux règles d'accessibilité écrites en SGSL consiste en un processus de transformations (voir Figure 10). Le document *d* (page HTML) à évaluer est donné en entrée de l'outil. Parallèlement, les règles SGSL existantes sont compilées et transformées en une feuille de style XSLT. Les règles exprimées jusqu'à présent en SGSL sont désormais exprimées en templates XSLT (l'outil WAEX [Centeno 07] vu précédemment utilise le même principe). Ces règles XSLT sont alors exécutées sur le document *d* donné en entrée et transforme celui-ci en un autre document, le rapport d'évaluation (appelé dans la figure « Résultat de la validation »).

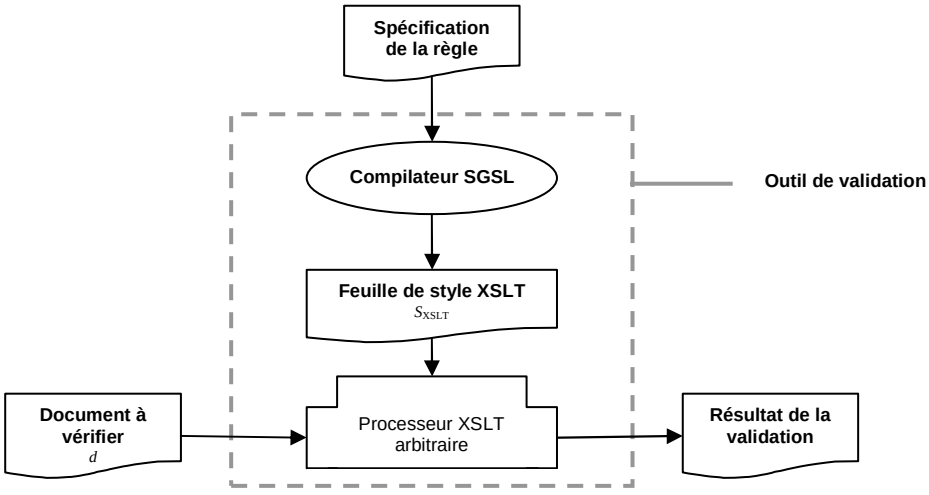


Figure 10 – Processus d'évaluation automatique dans l'outil de Takata et al. [Takata 04]

[21] WebXACT était anciennement connu sous le nom de *Bobby*. Depuis sa création, cet outil était gratuit mais il est devenu payant au 1er février 2008.

[22] XPath est un langage permettant de désigner une portion d'un document XML. Actuellement, ce langage en est à la version 1.0 et a fait l'objet d'une recommandation par le W3C. La spécification de XPath se trouve à l'adresse suivante : http://www.w3.org/TR/xpath

Le rapport d'évaluation est au format XML et contient la liste des règles violées ainsi qu'un message descriptif. Cependant, l'approche de Takata et al. ne permet pas un rapport détaillé : cette approche peut être assimilée à une approche ensembliste, c'est-à-dire que, pour une règle donnée, l'outil va constituer l'ensemble des éléments qui violent cette règle. Si cet ensemble n'est pas vide à la fin de la transformation, alors la règle est violée : cette information est la seule que nous avons lorsque la transformation est terminée. Par conséquent, le rapport d'évaluation n'est capable de mentionner ni le nombre d'erreurs trouvées pour une règle donnée ni les endroits (dans le code source de la page) où cette règle a été violée.

2.3.11. Synthèse sur les outils

Le Tableau 9 présente la liste des outils discutés dans cette section. Ce tableau montre tout d'abord qu'il y a une disproportion entre le nombre d'outils d'aide à la conception et le nombre d'outils d'aide à l'évaluation. Pourtant, intégrer des règles ergonomiques dans un outil de conception permettrait, dès l'implémentation des différentes pages, d'anticiper et d'empêcher les problèmes liés à ces règles. Par exemple, Dreamweaver [Adobe Dreamweaver] oblige le développeur à renseigner l'alternative textuelle à chaque fois qu'une image est insérée dans la page. Investir des efforts dans l'implémentation d'outils de conception permettrait de diminuer le nombre d'erreurs identifiées à la fin du processus de conception.

Les outils d'évaluation sont en général très performants pour la détection automatique des problèmes liés aux règles ergonomiques. Ils permettent d'inspecter une page ou une application Web entière et de pointer précisément où ces erreurs ont été identifiées dans le code source. Cependant, peu d'outils sont capables de faire de l'évaluation semi-automatique (seuls les outils A-Prompt [A-Prompt] et ReWeb & TestWeb [Ricca & Tonella 01a] font de l'évaluation semi-automatique). Quant à l'évaluation manuelle, il reste encore une grande majorité d'outils qui ne fournissent aucune aide à l'évaluateur pour procéder aux vérifications manuelles restantes. Tous ces outils évaluent les règles ergonomiques sur le code source HTML de l'application, exceptés ReWeb & TestWeb [Ricca & Tonella 01a]. Ces derniers se basent sur des modèles qui leurs sont dédiés.

Nous voyons apparaître de plus en plus d'outils qui séparent les règles ergonomiques du moteur d'évaluation. La principale motivation à cette séparation est de ne pas avoir à recoder l'outil si de nouvelles règles ergonomiques doivent être intégrées à l'outil (le moteur d'évaluation accepte d'évaluer une règle du moment qu'elle respecte une syntaxe précise ; si cette condition est respectée, l'évaluateur peut alors ajouter autant de règles qu'il le désire). Par exemple, DESTINE [Beirekdar 04], MAGENTA [Leporini 06], EvalAccess [Abascal 04], ou encore [Takata 04], ont proposé un format XML d'écriture des règles ergonomiques. Cependant, chacun de ces langages exprime les règles relativement au code HTML, c'est-à-dire qu'une règle est vue comme une condition exprimée sur le langage HTML. Par exemple, la règle « Fournir une alternative textuelle à une image » sera traduite par « Si l'élément analysé est `<img>`, alors si cet élément n'a pas d'attribut `alt`, signaler une erreur ». Ainsi, bien que ces langages de haut niveau permettent de séparer les règles du moteur d'évaluation, ils ne permettent pas de se détacher du langage HTML. Or, nous avons vu dans la section 2.2 consacrée aux processus et méthodes de conception Web, que les différents artefacts produits tout au long du cycle de vie pouvaient être évalués conformément aux connaissances ergonomiques. Les outils existants ne pouvant évaluer que du code HTML, ils sont dans l'incapacité d'évaluer ces artefacts.

Les rapports d'évaluation générés par ces outils sont la plupart du temps assez détaillés et parfois même navigables : certains outils permettent en cliquant sur un lien d'accéder directement à l'erreur dans le code source (par exemple, Ocawa [Ocawa]). Beirekdar et al. [Beirekdar 05] ont souligné l'importance de la qualité du rapport d'évaluation final mais aussi l'utilité de donner à l'évaluateur la possibilité de configurer ces rapports. Actuellement, seuls quelques outils offrent des rapports paramétrables (par exemple, DESTINE [Beirekdar 04], MAGENTA [Leporini 06], TAW [TAW]). Dans certains cas, le rapport d'évaluation ne décrit que le strict minimum, à savoir, si une règle est violée ou non ([Takata 04]).

Tableau 9 – Classification des outils basés sur des connaissances ergonomiques

Outil	Etape du processus de conception						Fonctionnalités						Niveau d'automatisation					Dimension traitée				Gestion des règles					Type de vérification				Technique de vérification					Catégorie				Rapport d'évaluation			
	Expression des besoins	Spécification	Conception	Implémentation	Evaluation & Utilisation du site	Maintenance du site	Guidage	Génération automatique	Evaluation	Réparation automatique	Réparation assistée	Accès à des ressources pour une évaluation/réparation manuelle	Intégré	Automatique	Semi-automatique	Manuel	Non Vérifiable	Utilisabilité	Accessibilité	Mobilité	Bonne pratique d'un langage informatique	Codé en dur	Ajout, Suppression, Modification de corpus de règles	Ajout, Suppression, Modification de règles	Possibilité de sélectionner les corpus de règles à évaluer	Possibilité de sélectionner les règles à évaluer	Analyse de la structure du site et des pages Web	Analyse linguistique du contenu	Vérification de propriétés physiques	Validateur syntaxique du code source	Analyse de modèle(s)	Analyse du code source		Analyse de logs	Génération automatique	Outil autonome	Service Web	Outil intégré à un environnement de conception	Plug-in	Configurable (paramétrable par l'utilisateur)	Format texte	Format hypertexte	Format XML
																																Analyse du code HTML	Analyse de feuilles de style CSS											
Conception																																											
Adobe Dreamweaver				X			X	X	X			X	X		X				X			X							X				X	X								
Microsoft Expression Web				X			X	X	X						X				X			X							X				X	X								
Evaluation																																											
A-Prompt					X				X	X	X	X		X	X	X			X			X			X					X				X	X					X	X
AccessEnable					X				X	X	X	X		X		X		X	X					X			X			X		X				X				X	
DESTINE					X				X	X	X	X		X		X		X	X				X	X	X	X				X		X			X			X	X	X	
Doctor Watson					X				X					X				X			X	X				X	X	X	X	X		X				X				X	
EvalAccess					X				X			X		X		X			X			X	X							X		X				X					X
HTML Toolbox					X				X	X				X				X			X	X					X	X	X	X		X				X			X		
HyperAT					X	X			X					X				X				X					X		X			X		X			X			X		
LIFT Machine					X				X	X				X				X	X	X*		X				X	X			X		X			X	X	X	X	X		X	
MAGENTA					X				X		X	X		X		X		X	X				X	X	X	X		X		X		X	X		X			X	X	X	X
Ocawa					X				X					X					X				X	X	X					X		X				X				X	
ReWeb & TestWeb					X				X						X			X				X					X				X				X						
[Takata 04]					X				X					X					X					X						X		X				X					X
TAW					X				X			X		X		X			X					X		X				X		X				X	X		X	X	X
WAEX					X				X					X					X	X				X						X		X				X				X	
Weblint					X				X					X				X			X	X				X	X	X	X	X		X				X			X		
WebSAT					X				X					X				X	X			X					X	X	X	X		X				X				X	
WebXACT					X				X			X		X		X		X	X			X					X		X	X		X					X			X	

* La version LIFT Mobile intègre des règles sur la mobilité

En ce qui concerne la correction automatique ou assistée, nous pouvons regretter que ces fonctionnalités ne soient pas offertes de manière systématique (parmi les outils offrant des options de correction, nous pouvons citer A-Prompt [A-Prompt] ou AccessEnable [Brinck 02]). Ces fonctionnalités sont pourtant indispensables quand on sait que sur des applications Web de grande taille, la correction manuelle des erreurs liées à une seule règle peut être longue et fastidieuse.

Enfin, la plupart des outils sont libres d'accès sur le Web, qu'ils soient disponibles sous la forme d'un service Web (par exemple, EvalAccess [Abascal 04]) ou d'un outil autonome (ces derniers peuvent être téléchargés gratuitement comme TAW [TAW]). La mise à disposition gratuite de ces outils est bénéfique pour tous et permet de vérifier la qualité de ses pages Web avant de les publier en ligne.

2.4. CONCLUSION

Dans ce chapitre, nous avons vu que la connaissance ergonomique existe sous plusieurs formes (principe, règle, recommandation, pattern d'interface, etc.), qu'elle traite de plusieurs dimensions d'une application (utilisabilité, accessibilité, etc.), qu'elle peut se prêter à une automatisation (automatique, semi-automatique, manuelle, etc.), et qu'elle peut être collectée dans différentes sources (guides de style, standard et règles isolées). Cette connaissance est cependant vaste et réunir ce savoir est difficile. L'organisation de cette connaissance ergonomique s'avère donc essentielle à l'application et à l'évaluation ergonomique d'une application. Cette organisation est facilitée par des outils de gestion des connaissances ergonomiques, outils nécessaires pour diminuer la complexité de gestion des règles. Ils facilitent ainsi la recherche et la sélection des règles selon certains critères : objet à évaluer, critère ergonomique, dispositif, aspect, domaine d'application, source, etc. Ils fournissent aussi de l'aide pour chaque règle ergonomique : liens vers d'autres ressources supplémentaires, règles liées, etc. En outre, ils intègrent en général une base de connaissances qui peut être alimentée facilitant ainsi l'ajout, la suppression et la modification de corpus de règles et/ou de règles.

L'étude des processus de conception Web existants montre que des artefacts sont produits dans chaque étape : modèles, descriptions d'interface, code de l'application, etc. Ces artefacts intègrent des options de conception et modélisent certains éléments de l'interface utilisateur de l'application finale qui pourraient être guidés par des connaissances ergonomiques. Ces connaissances peuvent donc être organisées autour des éléments de l'interface utilisateur en vue d'une inspection automatique sur les artefacts produits dans le processus de conception. L'organisation des connaissances ergonomiques est traitée dans le chapitre 3.

L'inspection ergonomique d'une application Web est coûteuse : il faut souvent interpréter les règles ergonomiques pour enlever les ambiguïtés, appliquer les connaissances selon le contexte, et inspecter l'application entière. L'automatisation de l'évaluation est donc nécessaire pour réduire les coûts, les erreurs humaines (ou oublis), ainsi que les problèmes d'interprétation/d'ambiguïté des connaissances. La connaissance ergonomique est alors intégrée dans l'outil d'évaluation et l'évaluateur n'a plus besoin d'être expert en facteurs humains. En outre, rendre les règles ergonomiques exécutables autorise la correction automatique allégeant ainsi l'effort de modification. Actuellement, les outils existants montrent que cette évaluation automatique n'est possible que sur le code HTML/CSS de l'application Web. L'automatisation de cette connaissance n'est donc pas réutilisable sur des données autres que le code. Autrement dit, les outils existants de support à l'évaluation ergonomique ne sont pas capables d'évaluer autre chose que l'application Web finale, et ne peuvent donc pas intervenir dans les différentes étapes du processus de conception. Pour pallier ces défauts, l'inspection des artefacts produits pendant la conception est nécessaire : cette problématique est au cœur de cette thèse et sera étudiée dans le chapitre 4.

3. Une ontologie pour l'organisation des règles ergonomiques

Résumé

Les règles ergonomiques étant énoncées en langage naturel, il est nécessaire de les organiser de manière à savoir précisément sur quels éléments de l'interface elles portent. Ces éléments, une fois formalisés, constituent alors un vocabulaire standard sur lequel s'appuyer pour l'organisation des règles ergonomiques.

Nous avons établi ce vocabulaire en développant une ontologie où les concepts relatifs aux règles ergonomiques ont été décrits formellement. Cette ontologie a été établie à partir de deux corpus de règles : les règles de navigation issues du projet EvalWeb et les règles d'accessibilité WCAG 1.0 issues du W3C/WAI.

Sur la base de cette ontologie, nous avons réalisé une étude sur son utilisation pour l'inspection ergonomique avant l'implémentation de l'application.

Ce chapitre est organisé de la manière suivante :

3.1 Méthode pour le développement de l'ontologie

3.2 Description de l'ontologie

3.3 Association ontologie / règles ergonomiques

3.4 Couverture des concepts dans les règles WCAG 1.0 et EvalWeb

3.5 Utilisation de l'ontologie pour l'inspection ergonomique avant implémentation

3.6 Discussion et conclusion

Les connaissances ergonomiques sont un moyen de guider la conception et l'évaluation, et leur utilisation lors de la conception d'un système a montré qu'elles ont un effet positif sur la qualité ergonomique [Borges 96] [Nielsen 94]. Toutefois, ces connaissances, puisqu'étant énoncées la plupart du temps en langage naturel (voir section 2.1), sont difficiles à appliquer sur l'interface utilisateur. Ces principales difficultés sont : l'ambiguïté due au langage naturel, l'incompréhension d'un énoncé trop général, ou encore, l'utilisation d'un jargon difficile à comprendre qui fait qu'il est difficile de savoir sur quels éléments de l'interface ces connaissances portent, mais également quelles vérifications doivent être menées. Les difficultés inhérentes aux interprétations des connaissances ergonomiques sont exposées plus exhaustivement dans [Nielsen 94] [Vanderdonckt 99] [Mariage 05b].

Selon le type de connaissances ergonomiques, ces difficultés sont plus ou moins importantes. Mariage [Mariage 05a] distingue trois catégories de connaissances ergonomiques (voir section 2.1.1) : les principes, les règles ergonomiques, et les recommandations. Sur ces trois types de connaissance ergonomique, seules les règles ergonomiques et les recommandations sont assez concrètes pour minimiser les difficultés d'utilisation de ces connaissances. Dans nos travaux, et plus particulièrement dans ce chapitre, nous nous sommes limités à ces deux types de connaissances que nous avons regroupés sous le terme de « règles ergonomiques ».

Dans ce chapitre, nous proposons de formaliser ces éléments conceptuels qui permettront l'organisation des règles. Ces éléments sont formalisés à l'aide d'une ontologie avec une définition formelle pour chaque élément, et une description de leurs propriétés ainsi que des relations qui existent entre eux.

Une ontologie est définie comme une spécification d'une conceptualisation [Gruber 93]. C'est une représentation formelle et déclarative d'une vue abstraite et simplifiée du monde (du domaine). Toute conceptualisation se base sur des concepts, objets et autres entités qui sont supposés exister dans un domaine d'intérêt et sur les relations qui existent entre eux. Les concepts et les contraintes sur l'utilisation de ces concepts sont exprimés dans une ontologie de manière déclarative et explicite en utilisant un langage formel. Une ontologie n'est donc pas exécutable, elle ne représente qu'une connaissance de manière déclarative visant à être utilisée par d'autres programmes.

L'objectif de l'ontologie est de formaliser la sémantique des concepts liés à un domaine et de leurs relations. En outre, l'ontologie aidera à uniformiser les énoncés des règles ergonomiques. Par exemple, les règles « Intégrez des pointeurs vers les principales sections dans l'aide à la navigation » et « Essayez de fournir constamment des liens vers la page d'accueil et vers les catégories importantes du site » mentionnent les termes « pointeurs » et « liens » qui se réfèrent au même concept. Une ontologie appropriée permet d'unifier ces deux termes en un seul, par exemple, le terme « lien ». En outre, l'ontologie établit une sémantique non ambiguë pour la signification du terme « lien ».

Nous présentons dans la section 3.1 la méthode que nous avons adoptée pour le développement de l'ontologie. Les deux corpus de règles qui sont à la base du développement de l'ontologie y seront détaillés : les règles d'accessibilité WCAG 1.0 et les règles de navigation EvalWeb. La section 3.2 donne ensuite une description de l'ontologie. La section 3.3 s'attache à décrire comment les règles ergonomiques peuvent être rattachées aux concepts de l'ontologie. La section 3.4 discute de la couverture des concepts de l'ontologie par les deux corpus de règles précédemment considérés. Dans la section 3.5, nous menons une étude sur l'utilisation de cette ontologie pour l'inspection ergonomique avant l'implémentation de l'application. Enfin, nous présentons une discussion et une conclusion dans la section 3.6.

3.1. METHODE POUR LE DEVELOPPEMENT DE L'ONTOLOGIE

De nombreuses méthodes ont été proposées pour le développement d'une ontologie. L'ensemble de ces méthodes couvre la construction, la fusion, l'adaptation, la reconception, la maintenance et l'évolution des ontologies [Corcho 02].

Dans l'étude comparative des différentes méthodes de développement d'ontologie de Corcho et al. [Corcho 02], il apparaît que les méthodes actuelles ne sont pas unifiées (il n'y a pas de correspondances entre les différentes méthodes proposées) et elles ne couvrent pas l'ensemble des étapes du cycle de vie d'une ontologie : la plupart des méthodes couvrent les phases de conception et d'implémentation mais d'autres phases telles que l'exploration des concepts, la maintenance ou la vérification et la validation ne sont prises en compte que par peu de méthodes.

Nous nous intéressons dans cette section à la construction de l'ontologie. Une ontologie modélise des concepts, leurs propriétés, la valeur de ces propriétés, les événements et leurs causes et effets, les processus et le temps [Chandrasekaran 99]. Par ailleurs, les concepts d'une ontologie peuvent être liés entre eux par des relations d'héritage (est_un), des relations d'agrégation (est_composé_de), et des instances peuvent être créées pour chaque concept. Une méthode de développement d'une ontologie doit par conséquent permettre la création de tous ces éléments. Nous pouvons noter qu'au vu de la méthode de construction d'une ontologie, la création d'une ontologie est analogue à une démarche de conception orientée objet [Devedžic 02].

La méthode que nous avons suivie pour la création de l'ontologie est fortement inspirée de celle proposée par Noy & McGuinness [Noy & McGuinness 01] et est composée des cinq étapes suivantes :

- **Sélection des règles ergonomiques :** dans cette première étape, l'ensemble des règles ergonomiques est sélectionné afin de constituer les données pour établir l'ontologie ;
- **Enumération des termes importants de l'ontologie :** dans cette étape, la liste des termes importants doit être établie sur la base des corpus sélectionnés à l'étape précédente ; nous ne nous soucions pas de savoir si les termes seront considérés au final comme des concepts ou comme des propriétés de concepts, ou si des termes se chevauchent et si des synonymes sont introduits ;
- **Définition des concepts :** cette étape consiste en l'identification des concepts. La définition des concepts se fait sur la base de la liste des termes établie dans l'étape

précédente. Ces concepts doivent être identifiés de manière unique. Autrement dit, les synonymes et les termes qui se chevauchent doivent être supprimés ;

- **Définition des hiérarchies de concepts et des relations d'agrégation entre concepts :** dans cette étape, les relations entre concepts sont modélisés. Les relations d'héritage ou de spécialisation (est_un) entre concepts forment ainsi plusieurs hiérarchies de concepts. En outre, les relations d'agrégation (est_composé_de) permettent de modéliser les composants de chaque concept ;
- **Définition des propriétés des concepts :** dans cette étape sont définis les propriétés des concepts (par exemple, nom, utilisation, etc.). Pour chacune de ces propriétés, le type, la cardinalité et les valeurs autorisées sont définis.

Les sections suivantes détaillent chacune de ces étapes.

3.1.1. Sélection des corpus de règles

Dans le cadre de nos travaux, nous nous sommes basés sur deux corpus de règles ergonomiques : les règles de navigation issues du projet EvalWeb et les règles d'accessibilité du W3C/WAI.

La navigation est un aspect important pour les applications Web et peut engendrer des problèmes importants d'utilisabilité si elle est mal conçue [Fleming 98]. Un ensemble de règles de navigation a été choisi pour établir notre ontologie. Ces règles sont issues du projet EvalWeb [Scapin 99] qui s'est déroulé de 1998 à 2002. Dans le cadre de ce projet dont l'objectif général était de participer à l'optimisation de la qualité ergonomique des Systèmes Interactifs Web, un travail de recueil, de classification et de réorganisation des recommandations ergonomiques pour le Web a été effectué. Les règles ergonomiques collectées dans ce projet sont issues de plusieurs sources et traitent de tous les aspects d'une application Web. Pour nos travaux, nous avons extrait de ces règles celles liées à la navigation. Ces règles de navigation constituent un total de 48 règles et sont présentées en Annexe D.

En matière d'accessibilité pour le Web, les recommandations WCAG 1.0 [WCAG] publiées par le W3C/WAI constituent la référence internationale et sont reconnues officiellement par de nombreux pays [Winckler 07]. Ces recommandations sont constituées de 14 directives, chacune divisées en plusieurs points de contrôle dont l'objectif est de décrire de manière plus précise les vérifications à mener pour chaque directive. Ces points de contrôle, que nous appellerons « règles », sont au nombre de 65. A chaque règle est associé un niveau de priorité allant de 1 (priorité haute) à 3 (priorité basse). Sur la base de ces niveaux de priorité, une application Web est dite conforme de niveau « A » si toutes les règles de priorité 1 sont satisfaites, de niveau « AA » si toutes les règles de priorité 1 et 2 sont satisfaites, et de niveau « AAA » si toutes les règles sont satisfaites. L'Annexe C illustre cette organisation.

Ces règles sont aujourd'hui reconnues comme permettant d'améliorer l'accessibilité des applications Web. Actuellement, la plupart des outils d'évaluation existants intègrent les règles WCAG 1.0 pour la vérification de l'accessibilité sur les applications Web. Toutefois, ces règles ne sont qu'un support à la suppression des barrières techniques de l'accessibilité et, même si ces règles sont respectées, des problèmes d'accessibilité peuvent subsister lors de l'utilisation en situation réelle avec des personnes handicapées.

Nous pouvons noter enfin que ces règles d'accessibilité sont accompagnées d'une documentation très détaillée. A chacune des règles sont associées des techniques HTML, CSS, et théoriques sur lesquelles l'évaluateur et le concepteur peuvent s'appuyer, par exemple, pour lever des ambiguïtés sur l'énoncé de la règle ou pour aider à une meilleure conception.

3.1.2. Enumération des termes importants de l'ontologie

Une liste des termes relatifs à la connaissance ergonomique des applications Web a été établie manuellement à partir des corpus de règles considérés. Chaque terme a été introduit tel

qu'il apparaissait dans l'énoncé de la règle. Les synonymes et les termes se chevauchant ont par conséquent été intégrés sans distinction. En outre, cette liste a été constituée sans se soucier de savoir d'une part, quelles étaient les relations entre ces termes, et d'autre part, si ces termes seraient considérés au final comme un concept ou une propriété d'un concept.

3.1.3. Définition des concepts

L'identification des concepts doit se faire en éliminant les synonymes de la liste des termes établis dans l'étape précédente, et en traitant au cas par cas les termes qui se chevauchent.

Les termes synonymes doivent être unifiés en un seul terme. Par exemple, les termes « pointeur » et « lien » font, tous les deux, référence à un lien hypertexte dans une page Web. Il n'y a pas de règles pour décider lequel de ces deux termes doit être choisi, ni si un troisième terme doit être introduit pour unifier ces deux derniers. Cette décision reste par conséquent subjective. Dans notre exemple, nous avons choisi arbitrairement d'unifier ces deux termes par « lien » puisque ce terme est plus répandu que « pointeur » dans l'ensemble des règles étudiées.

Le recouvrement de deux termes révèle en général qu'un des termes est une spécialisation du deuxième, ou bien qu'une propriété diffère entre ces deux termes. Par exemple, un « lien textuel » et un « lien image » sont des termes qui se chevauchent. Ces deux termes peuvent être vus comme une spécialisation du terme générique « lien », ou alors, comme étant tous les deux des liens mais avec, par exemple, la propriété « type = 'text' » (respectivement « type = 'image' »). Là encore, le choix reste subjectif. Toutefois, lorsque l'un des deux termes définit des propriétés qui lui sont propres, la spécialisation est une solution à privilégier afin de ne pas faire intervenir des propriétés inutiles dans le terme générique (dans notre exemple, le terme « lien »). Par exemple, un lien textuel est un lien auquel la propriété 'couleur' est ajoutée pour définir la couleur du texte du lien. Une telle propriété n'a pas de sens pour un lien image. Par conséquent, nous avons choisi de représenter un lien textuel et un lien image comme des spécialisations du terme « lien » avec, pour un lien textuel, une propriété 'couleur' supplémentaire (propriété qui n'existe pas pour un lien image).

3.1.4. Définition des hiérarchies de concepts et des relations d'agrégation entre concepts

Seules certaines hiérarchies de concepts ont été définies lors de l'identification des concepts dans l'étape précédente pour régler les problèmes liés au chevauchement de termes (voir section 3.1.3). En conséquence, un travail systématique sur tous les concepts a été mené pour déterminer l'ensemble de ces hiérarchies.

Une hiérarchie de concepts permet de modéliser des relations « est_un » entre concepts. Un concept est représenté formellement par une classe, par conséquent, une hiérarchie de concepts définit une hiérarchie de classes. Il existe plusieurs stratégies pour établir une hiérarchie de classes [Uschold & Gruninger 96] :

- Une approche *top-down* qui consiste à définir les classes les plus générales en premier, puis à spécialiser ces classes. Par exemple, un contenu multimédia peut se définir par la classe MultimediaContent. Cette classe est ensuite raffinée en ImageContent, AudioContent, VideoContent, et ProgrammaticObject. Chacune de ces classes peut à son tour être raffinée, par exemple, ImageContent est raffinée en ImageLink, Image, ImageMap, ASCIIArt, et ainsi de suite.
- Une approche *bottom-up* qui consiste à partir des classes les plus spécifiques, autrement dit les classes se trouvant au plus bas dans la hiérarchie, et de procéder à des regroupements successifs en des classes plus générales. Par exemple, les classes Image et ImageLink peuvent être regroupées en une classe plus générale ImageContent qui elle-même est un MultimediaContent, et ainsi de suite.

- Une approche *middle-out* qui est une combinaison des approches top-down et bottom-up. Les classes les plus importantes sont identifiées en premier lieu. Chacune de ces classes est ensuite spécialisée et/ou généralisée en fonction. Par exemple, parmi les classes importantes, nous trouvons la classe Image. A partir de cette classe, nous pouvons la généraliser en ImageContent, mais également la spécialiser, par exemple, en GraphicalRepresentationOfText.

Les concepts identifiés sur les corpus de règles à disposition pouvant être soit généralisés soit spécialisés, nous avons adopté une approche middle-out pour définir les différentes hiérarchies de classes (voir Figure 11). Lors de la définition des hiérarchies, certains concepts (ou classes) ont été introduits, c'est le cas, par exemple, du concept ImageContent. L'ontologie que nous avons établie contient par conséquent des concepts qui ne figurent pas dans les corpus de règles ergonomiques étudiés.

Une fois les hiérarchies de classes établies, les relations d'agrégation entre classes sont ajoutées. Par exemple, un lien image (concept ImageLink) doit contenir une référence vers son image (concept Image), et en conséquence, une relation d'agrégation est ajoutée entre les classes ImageLink et Image (voir Figure 11).

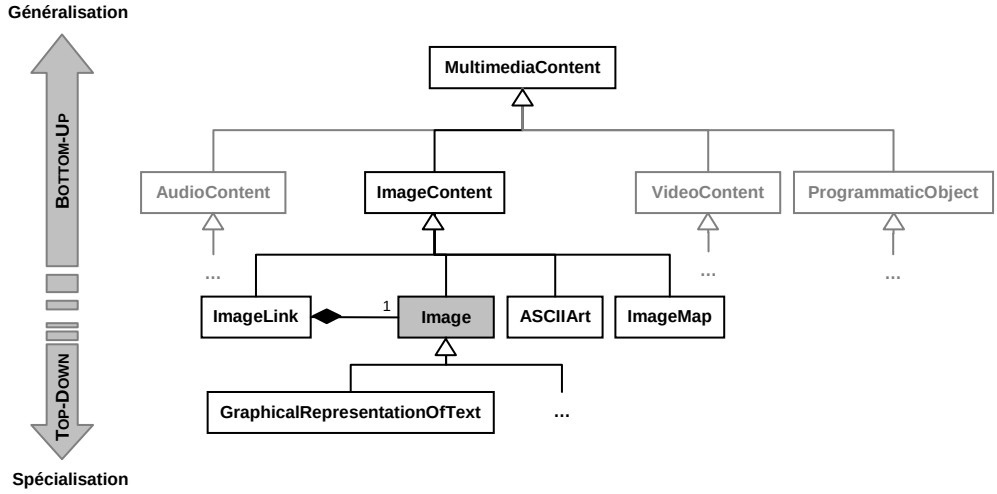


Figure 11 – Définition d'une hiérarchie de classes à partir du concept *Image*

3.1.5. Définition des propriétés des concepts

A partir des règles ergonomiques, nous avons défini les propriétés des différents concepts identifiés dans la phase précédente. Généralement, ce sont sur ces propriétés que portent les règles ergonomiques. Par exemple, un contenu multimédia possède une alternative textuelle et une description, un contenu image possède une largeur et une hauteur, et une image contient une référence vers son fichier physique (voir Figure 12). L'alternative textuelle d'un contenu multimédia étant un texte facultatif, son type est une chaîne de caractères (String), la cardinalité est de 0..1, et toutes valeurs sont autorisées (tout texte est valide).

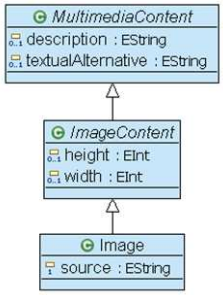


Figure 12 – Définition des propriétés des concepts pour les concepts *MultimediaContent*, *ImageContent*, et *Image*

3.2. DESCRIPTION DE L'ONTOLOGIE

La section 3.1 a présenté la méthode adoptée pour le développement de l'ontologie. Une vue d'ensemble de l'ontologie est présentée en Figure 15. Cette ontologie est décrite formellement en OWL (Ontology Web Language) [OWL 04] (voir Annexe B) et contient 98 concepts tels que « Page », « Link », « Table », « Frame », « NavigationalAid », ou « SiteMap ».

Les concepts de l'ontologie sont organisés autour de cinq catégories : Site, Metadata, Container, Page, et Content. La Figure 13 présente ces catégories sous la forme d'un diagramme de classes UML, pour mettre en valeur les relations qui lient ces catégories. Le concept Site représente une application Web auquel sont rattachés des conteneurs (concept Container). Généralement, ces conteneurs sont des pages Web (concept Page) et chacune de ces pages possède du contenu (concept Content). Un site Web, ainsi qu'une page Web peuvent contenir des méta données (concept MetaData). Chacune de ces catégories définit une hiérarchie de concepts, autrement dit, tout autre concept de l'ontologie appartient à une de ces catégories.

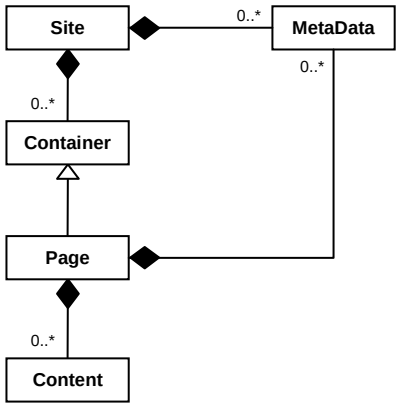


Figure 13 – Concepts piliers de l'ontologie : Site, MetaData, Container, Page, et Content

L'ontologie ayant été construite sur la base de règles ergonomiques relatives aux applications Web, la plupart des concepts de l'ontologie sont relatifs aux éléments de l'interface utilisateur finale tels que « Link », « Image », « CheckBox », ou « SubmitButton ».

Certains concepts identifiés sont toutefois d'un niveau d'abstraction plus élevé, comme par exemple, le concept d'aide à la navigation (NavigationalAid). Une aide à la navigation représente une structure contenue dans une page Web destinée à faciliter la navigation dans l'application Web. Il existe plusieurs types d'aide à la navigation (voir Figure 14) : fonction de recherche (SearchFacility), index alphabétique (AlphabeticalIndex), en-tête de localisation (LocationHeader), barre de navigation (NavigationBar), plan du site (SiteMap), et table des matières (TableOfContents).

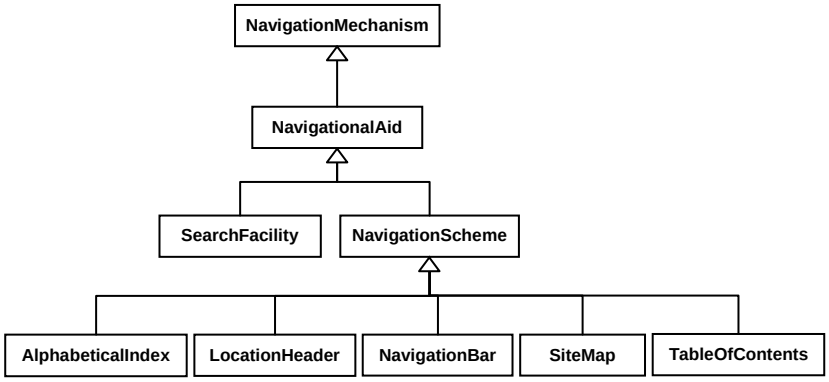


Figure 14 – Diagramme de classes pour le concept *NavigationalAid*

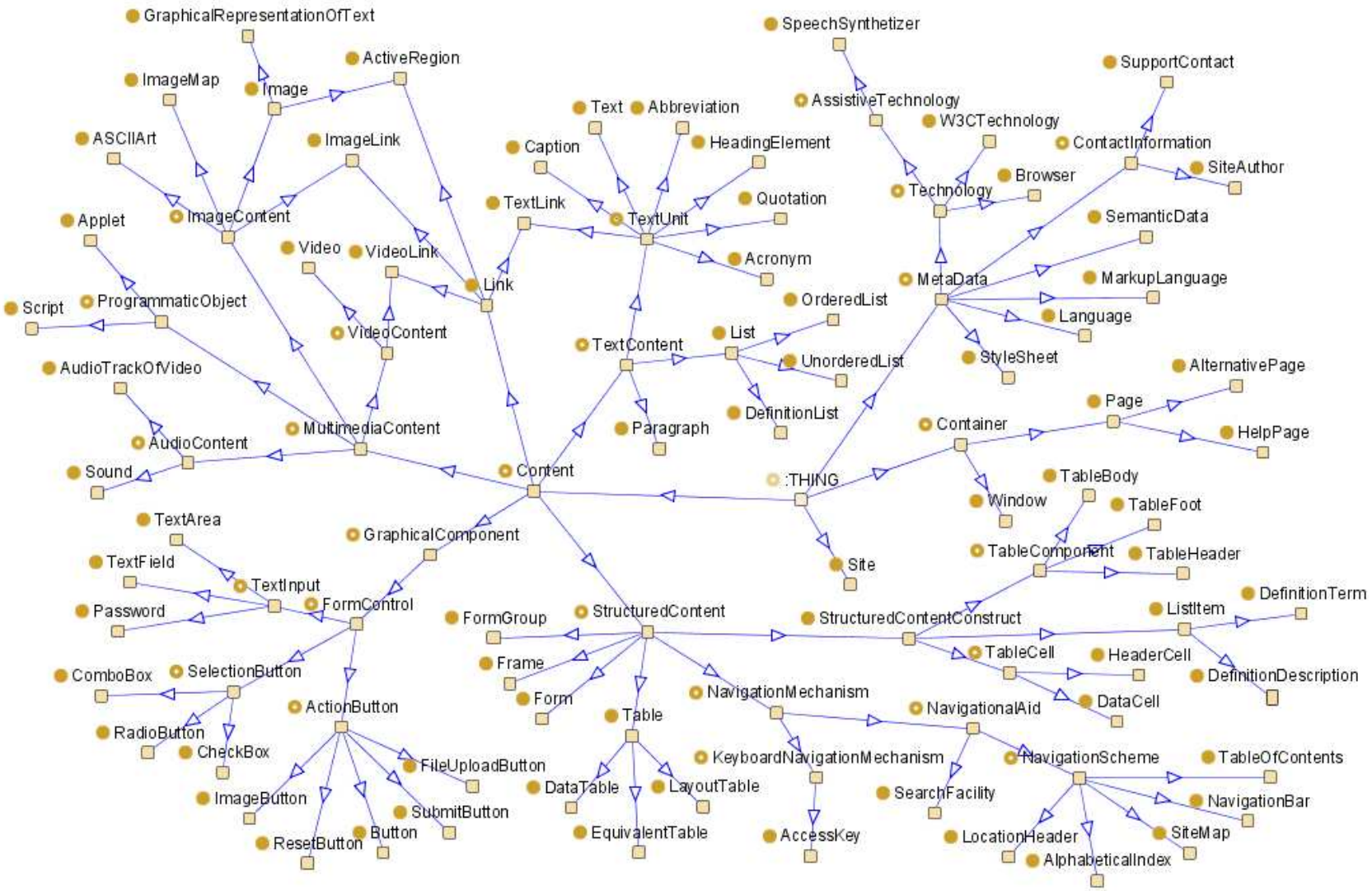


Figure 15 – Vue d’ensemble de l’ontologie

Toutes ces aides à la navigation ont des fonctionnalités très différentes. Par exemple, une fonction de recherche permet à l'utilisateur de trouver son information via une recherche par mots clé, alors qu'un plan du site permet à celui-ci de trouver l'information en consultant le schéma global du site. La description de ces aides à la navigation est d'un niveau d'abstraction élevé et ces éléments ne correspondent pas à un élément de l'interface utilisateur particulier.

3.3. ASSOCIATION ONTOLOGIE / REGLES ERGONOMIQUES

En créant l'ontologie, notre objectif est d'organiser les règles par rapport aux concepts de cette ontologie, autrement dit, de rattacher les règles aux concepts de l'ontologie. L'association entre une règle ergonomique et les concepts de l'ontologie est une tâche nécessitant une bonne compréhension du sens porté par la règle ergonomique afin d'identifier clairement les concepts qui doivent être rattachés à cette règle.

Certaines règles, dans leur énoncé original, sont trop abstraites pour que l'identification des concepts puisse être directement effectuée. C'est le cas par exemple de la règle « Testez la navigation » (règle EvalWeb n°2, voir Annexe D) qui ne renseigne ni sur les éléments à évaluer, ni sur ce qu'il faut faire pour les évaluer. Pour obtenir ces informations, une interprétation de cette règle en une ou plusieurs règles plus concrètes est alors nécessaire. La phase d'interprétation des règles sera abordée en totalité dans la section 4.3.1, illustrée par l'exemple « Testez la navigation ». Nous ne nous intéressons dans cette section qu'à une partie de la phase d'interprétation : celle concernant l'identification des éléments à évaluer, autrement dit, l'identification des concepts de l'ontologie à rattacher à la règle.

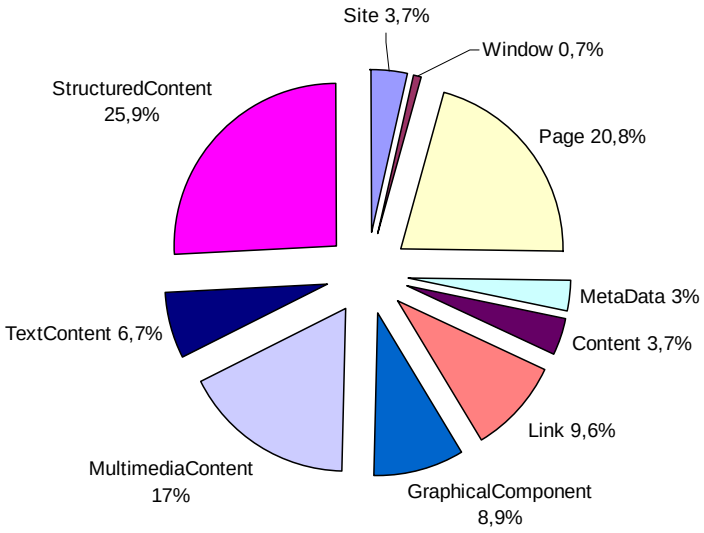
L'identification des concepts de l'ontologie sur lesquels s'appliqueront les règles plus concrètes dérivées de la règle originale nécessite d'une part, de bien connaître les concepts de l'ontologie, et d'autre part, de tenir compte des relations d'héritage entre concepts de l'ontologie. Ces relations font que les règles qui s'appliquent à un concept s'appliquent également à ses concepts fils (c'est-à-dire les concepts qui en héritent). Par exemple, la règle suivante « Vérifier que les liens connectent vers des pages qui existent » (règle EvalWeb n°1, voir Annexe D) est rattachée au concept *Link* de l'ontologie. De ce fait, elle sera automatiquement rattachée aux concepts qui héritent de *Link*, par exemple, *TextLink* et *ImageLink*. Cette propriété de l'ontologie a pour avantage le rattachement automatique à plusieurs concepts dès lors que nous rattachons une règle à un concept d'une hiérarchie. De ce fait, il est nécessaire de faire attention à ce que la règle à laquelle nous rattachons un concept soit valable également pour ses concepts fils dans la hiérarchie.

Le travail d'identification des concepts a été effectué pour chacune des règles des corpus WCAG 1.0 et EvalWeb que nous avons considérés. Le détail est donné pour chaque règle en Annexe C et Annexe D.

3.4. COUVERTURE DES CONCEPTS DANS LES REGLES WCAG 1.0 ET EVALWEB

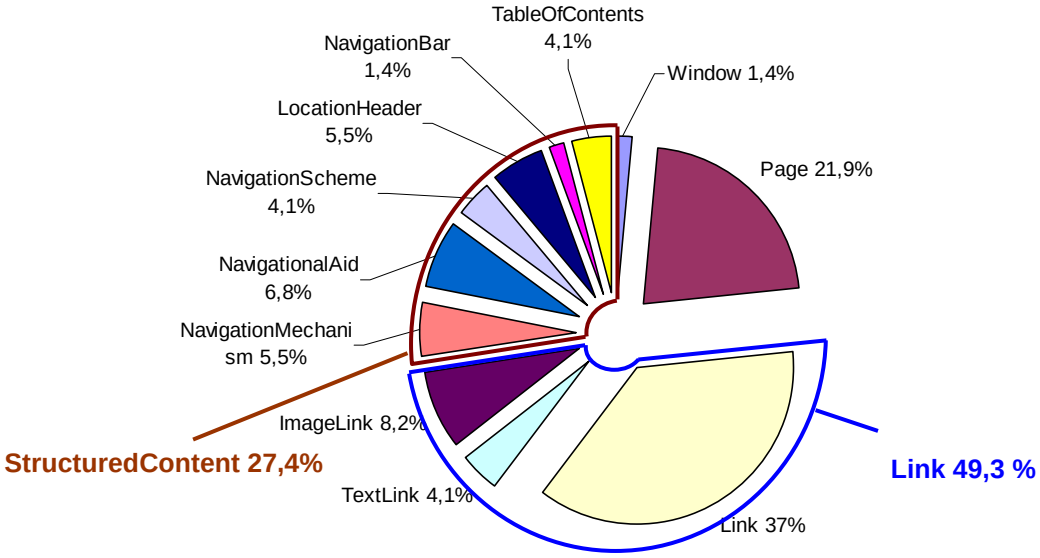
Cette section discute de la couverture des concepts de l'ontologie par les corpus de règles pris en compte. Les concepts décrits par l'ontologie couvrent principalement les éléments adressés par un corpus de connaissances ergonomiques donné. En outre, comme nous l'avons vu dans la section 3.1.4, certains concepts, ne faisant pas partie de ce corpus, peuvent être ajoutés pour établir les relations de hiérarchie entre ces concepts. Ceci a pour conséquence que la liste des concepts de l'ontologie ne contient pas nécessairement tous les éléments de l'interface utilisateur d'une application Web puisque certains éléments ne sont concernés par aucunes règles. La liste des concepts de l'ontologie, leurs propriétés ainsi que les relations qui existent entre eux sont présentés de manière détaillée en Annexe A.

La Figure 16 et la Figure 17 présentent la proportion des concepts tels qu'ils sont adressés respectivement par les règles d'accessibilité WCAG 1.0 et les règles de navigation EvalWeb.



	Site	Window	Page	MetaData	Content	Link	Graphical-Component	Multimedia-Content	Text-Content	Structured-Content	**Total**
Nbre de règles	5	1	28	4	5	13	12	23	9	35	135
%	3,7	0,7	20,8	3	3,7	9,6	8,9	17	6,7	25,9	100

Figure 16 – Distribution des concepts de l'ontologie selon les règles du W3C/WAI



	Window	Page	Link	TextLink	ImageLink	Navigation-Mechanism	Navigational-Aid	Navigation-Scheme	Location-Header	Navigation-Bar	TableOf-Contents	**Total**
Nbre de règles	1	16	27	3	6	4	5	3	4	1	3	73
%	1,4	21,9	37	4,1	8,2	5,5	6,8	4,1	5,5	1,4	4,1	100

Figure 17 – Distribution des concepts de l'ontologie selon les règles du projet EvalWeb

Les règles WCAG 1.0 (voir Figure 16) portant sur l'accessibilité du contenu de l'application Web, il n'est pas surprenant que 72,1% des règles portent sur des concepts issus de la catégorie Content (StructuredContent, TextContent, MultimediaContent, GraphicalComponent, Link, et Content). Parmi ces règles régissant le contenu, celles consacrées aux contenus structurés (tables, listes, etc.) et aux contenus multimédias (contenu audio, vidéo, script, etc.) représentent une large majorité (respectivement un tiers et un quart des règles liées au contenu) puisque ces deux types de contenu peuvent être complexes et poser des problèmes importants d'accessibilité. Nous remarquons également que plus de 20% des règles sont

relatives aux pages Web (concept Page) : conformité avec la syntaxe XHTML, association d'une feuille de styles pour chaque page, utilisation de pages alternatives, etc.

Les règles de navigation EvalWeb (voir Figure 17) ne portant exclusivement que sur les éléments de navigation, nous ne retrouvons que des concepts liés à la navigation, ainsi que les containers (Page ou Window) susceptibles soit de contenir ces éléments, soit d'être la cible de ses éléments (par exemple, une page peut être considérée comme la cible d'un lien).

La moitié des règles (49,3%) concernent les liens (TextLink, ImageLink et Link) puisqu'un lien est l'élément de base pour naviguer. Plus du quart des règles (27,4%) est dédié aux mécanismes de navigation (NavigationMechanism, NavigationalAid, NavigationScheme, LocationHeader, NavigationBar, et TableOfContents). Un mécanisme de navigation désigne tout élément logiciel (par exemple, barre de navigation) ou matériel (par exemple, navigation par clavier) permettant de naviguer dans l'application Web. Enfin, 23,3% des règles concernent les containers cibles ou sources de ces éléments de navigation (Window et Page). Le tableau complet des règles de navigation ainsi que des concepts concernés par chacune de ces règles est donné en Annexe D.

3.5. UTILISATION DE L'ONTOLOGIE POUR L'INSPECTION ERGONOMIQUE AVANT IMPLEMENTATION

Nous considérons que les concepts de l'ontologie sont suffisamment génériques pour que l'on puisse les identifier sur différents artefacts utilisés pour la conception des applications web. Dans cette section, nous montrons tout d'abord qu'il est possible d'identifier les concepts de l'ontologie sur des modèles utilisés pour la conception des applications web, et qu'il est ainsi possible d'estimer le nombre de règles ergonomiques (attachées à ces concepts) potentiellement vérifiables. Pour cela, nous avons étudié les modèles utilisés dans trois méthodes de conception, mais également les outils supportant chacune de ces méthodes de manière à pouvoir estimer le niveau d'automatisation de ces règles ergonomiques. L'étude prend en compte :

- trois ensembles de modèles issus des méthodes WebML, OO-H et UsiXML ;
- trois outils (WebRatio, VisualWade, SketchiXML) associés à ces méthodes ;
- les règles d'accessibilité WCAG 1.0 de priorités 1 et 2, soit un total de 49 règles (voir détail en Annexe C) ;

La section 3.5.1 présente tout d'abord la démarche méthodologique suivie pour mener cette étude. Les méthodes et outils de conception Web qui ont servi à cette étude sont présentés dans la section 3.5.2. Les niveaux d'automatisation pris en compte sont présentés ensuite dans la section 3.5.3. Enfin, dans la section 3.5.4, nous présentons et analysons les résultats de cette étude.

Le travail présenté ici est issu en partie des travaux publiés dans [Xiong 07] et [Xiong 08b]. Le lecteur pourra s'y référer pour une étude plus détaillée portant sur un nombre plus large d'outils.

3.5.1. Protocole suivi pour l'estimation des règles vérifiables

Tous les modèles produits avec les outils étudiés (voir section 3.5.2) ont été analysés afin de déterminer si l'information qu'ils contenaient pouvait être utilisée (ou non) pour supporter l'inspection des règles d'accessibilité. Autrement dit, nous avons analysé les modèles utilisés afin de déterminer s'ils étaient capables de représenter les concepts identifiés dans l'ontologie dans le but final de l'inspection automatique.

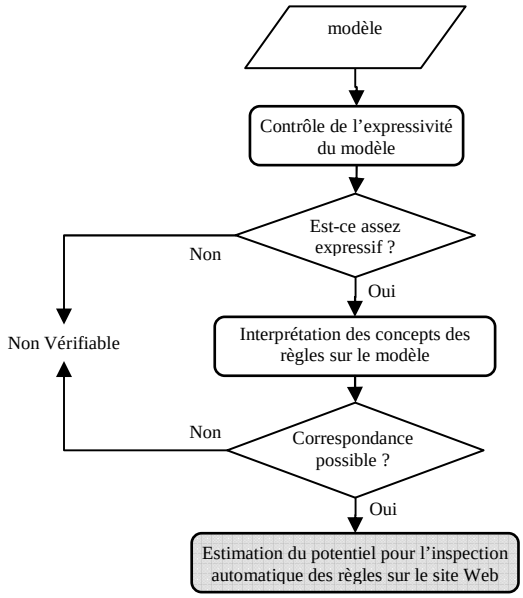


Figure 18 – Méthode utilisée pour l'étude du nombre de règles vérifiables par outil

La méthode que nous avons suivie pour cette analyse est présentée en Figure 18. L'artefact étudié est tout d'abord contrôlé afin de vérifier son expressivité. Cette phase consiste à vérifier que l'artefact peut être soumis à des vérifications automatiques. S'il est jugé peu expressif, l'inspection ergonomique est considérée comme impossible. Par exemple, les cas d'utilisation UML sont trop abstraits pour qu'une inspection ergonomique puisse être menée. Dans le cas où l'artefact est jugé assez expressif, une phase d'interprétation des concepts des règles sur l'artefact a lieu. Elle consiste à mettre en correspondance les concepts de l'ontologie avec les concepts de l'artefact. Par exemple, dans une notation basée sur des diagrammes d'états, un état peut être interprété comme une page Web et les transitions entre états comme des liens entre ces pages, et ainsi de suite. Si un artefact supporte une telle sémantique, nous interprétons et appliquons les règles appropriées pour inspecter les états et les transitions comme s'ils étaient des pages et des liens. A partir de ces mises en correspondance, nous établissons une estimation des règles d'accessibilité pouvant être inspectées automatiquement.

3.5.2. Méthodes de conception pour le Web étudiées

Dans cette étude, nous avons considéré les trois méthodes suivantes : WebML, OO-H et UsiXML. Les méthodes WebML et UsiXML ont été abordés précédemment dans l'état de l'art, respectivement dans les sections 2.2.2.2 et 2.2.3.3 ; leurs caractéristiques principales sont brièvement rappelées ici. La méthode OO-H n'a pas été décrite dans l'état de l'art car elle présentait un profil méthodologique voisin de celui de WebML. Ces deux méthodes ne produisent toutefois pas les mêmes artefacts, c'est pourquoi elles sont toutes deux évaluées dans cette étude. Pour chaque méthode, nous ne présentons que les modèles exploitables pour une inspection automatique. Ceci explique que les documents papiers ne sont pas considérés comme des modèles et par conséquent, ne sont pas recensés ici.

Les méthodes présentées sont supportées par des outils qui prennent en charge la création, l'édition et la sauvegarde des modèles utilisés. En outre, tous ces outils permettent la génération de l'interface utilisateur finale. Ils sont présentés ici à la suite de la méthode qu'ils supportent.

WebML [Ceri 00] est une notation destinée à la spécification d'applications Web. La modélisation couvre les données, les pages Web, la navigation entre ces pages, la présentation et la personnalisation par utilisateur. Les modèles produits sont : un modèle structurel, un modèle de composition, un modèle de navigation, des feuilles de styles typées et non typées, et des profils utilisateurs.

L'outil associé à la méthode WebML est WebRatio [Acerbis 04] [Acerbis 08] [WebRatio]. Outre les fonctionnalités citées précédemment, une des caractéristiques de cet outil est que les modèles produits avec l'outil sont en constante relation avec l'application finale générée. Cette caractéristique a son importance car elle facilite la maintenance d'une application déployée et les erreurs détectées lors des inspections menées dans les phases d'utilisation de l'application (post-implémentation) peuvent alors être prises en compte et gérées par l'outil.

UsiXML [Limbourg 04] est un langage XML de description d'une interface utilisateur destinée à être utilisée dans différents contextes d'utilisation tels que les interfaces graphiques, tactiles, sonores ou multimodales. L'ensemble des modèles utilisés permet ainsi de représenter l'interface utilisateur tout au long du processus de conception. Ces modèles sont : le modèle de transformation, le modèle de mapping, le modèle de contexte, le modèle de tâche, le modèle du domaine, l'interface utilisateur abstraite, l'interface utilisateur concrète, et l'interface utilisateur finale.

UsiXML est un langage accompagné de plusieurs outils dont la liste peut être trouvée à cette adresse : http://www.usixml.org/. Parmi ces outils, nous avons choisi de nous intéresser à SketchiXML. SketchiXML [Coyette 07] est un outil offrant la possibilité à un concepteur de dessiner rapidement une interface utilisateur de la même manière qu'avec un logiciel de dessin classique. Cet outil est équipé d'un moteur de reconnaissance de textes et de formes permettant l'identification automatique de croquis et la reconstruction automatique des composants de l'interface utilisateur en conséquence. L'interface ainsi obtenue est spécifiée en langage UsiXML. L'utilisation d'un tel outil permet d'obtenir facilement et rapidement une maquette reflétant l'interface utilisateur d'une page Web. Bien que SketchiXML ne soit pas doté de fonctionnalités dédiées à la conformité avec l'accessibilité, des évaluations peuvent être réalisées sur la base du code source UsiXML de la page grâce à un module appelé UsabilityAdviser. Les résultats présentés dans cette étude ne tiennent pas compte de ce plug-in.

OO-H [Gómez 03] est une méthode de conception d'applications Web dont les activités de modélisation permettent de couvrir les données, la navigation, et la présentation. La modélisation des données se fait au moyen de cas d'utilisation et de diagrammes de classes UML. La navigation est spécifiée au moyen de diagrammes NAD (Navigational Access Diagram). Ces diagrammes s'articulent autour de quatre concepts : les classes de navigation, les cibles de navigation, les liens de navigation et les collections. Il est important de noter d'une part, que les liens de navigation sont typés, et d'autre part, que des patrons de navigation associés aux liens de navigation et/ou aux collections existent (index, visite guidée, visite guidée indexée, et affichage des résultats (d'une requête)). Ces éléments apportent ainsi de la sémantique supplémentaire susceptible d'être pertinente pour l'inspection ergonomique. Concernant la présentation, cinq gabarits, décrits en XML, sont utilisés pour spécifier l'apparence visuelle. Ces gabarits définissent la disposition de l'information à présenter, l'aspect visuel de la page, la présentation des formulaires, les fonctionnalités client, et l'affichage de plusieurs vues simultanées.

VisualWade [Gómez 04] est l'outil développé pour supporter la méthode OO-H. Contrairement à WebRatio, les modèles développés au moyen de VisualWade ne permettent qu'une génération de l'application finale et ne sont plus synchronisés par la suite avec cette dernière. Par conséquent, si des inspections ergonomiques effectuées sur l'application finale révèlent des erreurs, les corrections ne pourront se faire automatiquement que sur l'application finale. La maintenance de l'application est donc impossible sur les modèles édités par l'outil.

3.5.3. Niveaux d'automatisation des règles selon l'outil

L'automatisation des règles diffère selon les outils : si en théorie, une règle est caractérisée comme automatisable selon les niveaux d'automatisation présentés en section 2.1.4, en pratique, il est possible que l'outil se limite à une vérification manuelle de cette règle. Il est également possible que l'outil effectue une analyse automatique afin de vérifier si cette règle est respectée ou non. Dans le meilleur des cas, l'outil peut proposer des améliorations dans le cas

où la règle n'est pas respectée. Il est par conséquent important de pouvoir caractériser les niveaux d'automatisation fournis par les différents outils. Pour cela, nous avons étendu la taxonomie proposée par Ivory [Ivory 03] qui se compose des cinq niveaux d'automatisation suivants :

- **Aucun (None) –** La règle ergonomique est supportée par l'outil mais aucune analyse automatique n'est effectuée. C'est le cas typique des règles qui, lors de l'évaluation, génèrent systématiquement une alerte, telle que la règle « Utiliser le langage le plus clair et le plus simple possible adapté au contenu de votre site » (source : WCAG 1.0, règle 14.1 [Priorité 1]) ;
- **Capture –** L'outil enregistre automatiquement des données (par exemple, le poids des pages Web) qui peuvent être utilisées ultérieurement pour la vérification des règles telles que « une page Web ne doit pas dépasser 70Ko » (source : AccessiWeb 1.0, règle 12.8 [Or]) ;
- **Analyse –** L'outil détecte automatiquement les violations sur la règle mais ne propose pas d'améliorations. Par exemple, le non respect de la règle « Fournir un équivalent textuel à chaque élément non textuel » (source : WCAG 1.0, règle 1.1 [Priorité 1]) pourrait être reporté comme « L'attribut alt de l'élément img n'est pas spécifié, ligne 21 » ;
- **Critique –** L'outil identifie automatiquement les violations sur la règle et décrit le processus pour se conformer à la règle ; par exemple, la règle « Identifier le langage naturel principal du document » (source : WCAG 1.0, règle 4.3 [Priorité 3]), les indications fournies par l'outil pourraient être : « Si le contenu est du HTML, vérifier l'attribut lang de l'élément html. Si le contenu est du XHTML 1.0 ou toute autre version de XHTML référencée comme "text/html", vérifier à la fois les attributs lang et xml:lang de l'élément html. Si le contenu est du XHTML 1.1 ou plus et référencée comme "application/xhtml+xml", vérifier l'attribut xml:lang de l'élément html. » ;
- **Suggestion**[23] **–** L'outil propose des options dans les phases de conception, afin de se conformer à la règle par construction. Par exemple, une option d'accessibilité pourrait être activée afin que soit proposée automatiquement au développeur de l'application une alternative textuelle chaque fois que ce dernier insère une image dans une page. Un autre exemple est d'activer une option afin que toutes les tables créées soient automatiquement accessibles.

Les niveaux d'automatisation présentés ici ne sont pas en contradiction avec les niveaux d'automatisation des règles présentés dans la section 2.1.4 où les niveaux sont indépendants d'un quelconque outil. Au contraire, les niveaux d'automatisation décrits dans cette section permettent de refléter au mieux l'automatisation des règles telles qu'elles ont été implémentées dans chaque outil.

3.5.4. Résultats de l'étude

Nous présentons dans cette section les résultats. Le Tableau 10 présente les données recueillies par outil.

[23] La catégorie *Suggestion* a été ajoutée à celles proposées initialement par Ivory [Ivory 03].

Tableau 10 – Nombre de règles WCAG 1.0 supportées

Priorité	N. règle	WebRatio	VisualWade	SketchiXML
1	1.1		Su	Su
	1.2			
	1.3			
	1.4			
	2.1			
	4.1			
	5.1			Su
	5.2			Su
	6.1			
	6.2			
	6.3			
	7.1			
	8.1			
	9.1			
	11.4			
	12.1	Su	Su	Su
	12.4		Su	
	14.1			
2	2.2	No		An
	3.1			
	3.2			
	3.3	Su	Su	
	3.4	An		
	3.5			
	3.6			
	3.7			
	5.3			
	5.4			
	5.5	An		Su
	6.4			
	6.5			
	7.2			
	7.3			
	7.4			
	7.5			
	8.1			
	9.2			
	9.3			
	10.1			
	10.2		Su	No
	11.1			
	11.2			
	12.2	Su	Su	
	12.3	No		No
	12.4		Su	
	13.1			No
	13.2	Su	Su	
	13.3	Su	Su	
	13.4	Su	Su	
Nombre de règles		10	10	9
% règles supportées		20,41%	20,41%	18,37%

Légende

No	None
Ca	Capture
An	Analyse
Cr	Critique
Su	Suggestion

	Non vérifiable
	Supporté

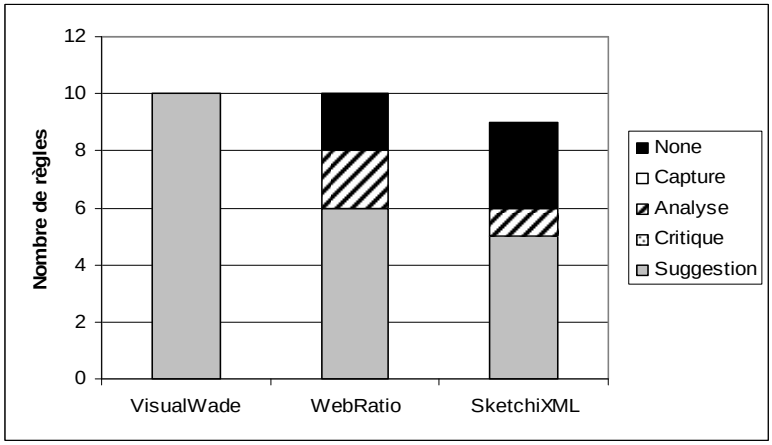
La Figure 19 présente graphiquement le nombre de règles d'accessibilité qui pourraient être théoriquement inspectées par outil. VisualWade et WebRatio peuvent inspecter 10 règles sur les 49 de ce corpus, alors que SketchiXML peut en inspecter 9.

L'estimation que nous avons fourni sur le niveau d'automatisation de chaque règle supportée est optimiste : par exemple, une règle pouvant avoir les niveaux d'automatisation *analyse* (l'inspection de la règle peut être réalisée automatiquement par l'outil) et *suggestion* (l'outil a un rôle proactif en contraignant l'utilisateur à respecter la règle) sera considérée comme ayant le niveau *suggestion* car nous considérons le cas le plus favorable.

Comme le montre la Figure 19, la majorité des règles supportées est de niveau *suggestion*. Certaines règles ne peuvent cependant pas être inspectées de manière proactive et n'ont alors que le niveau *analyse*. C'est le cas par exemple de la règle 2.2 pour SketchiXML : « S'assurer que la combinaison de couleurs entre le premier plan et l'arrière-plan utilise suffisamment de contraste […] ».

Certaines règles ne peuvent pas être automatisées car la vérification à effectuer ne peut être faite que manuellement. C'est le cas, par exemple, de la règle 12.3 : « Lorsque c'est approprié, diviser les grands blocs d'information en groupes plus petits et plus facilement

manipulables ». Cette vérification manuelle se fait par l'évaluateur en analysant le ou les concepts mentionnés dans la règle ergonomique directement sur l'artefact.



	VisualWade	WebRatio	SketchiXML
Suggestion	10	6	5
Critique	0	0	0
Analyse	0	2	1
Capture	0	0	0
None	0	2	3
Total	**10**	**10**	**9**

Figure 19 – Nombre de règles supportées par niveau d'automatisation

L'inspection automatique d'une application permet de systématiser l'évaluation d'une application conformément à un ensemble de règles ergonomiques [Nielsen 94]. Cependant, la nature de ces règles fait qu'il n'est pas possible de toutes les automatiser, et certaines requièrent encore une interprétation et une vérification manuelle. La limite de l'automatisation a été estimée à 44% des règles ergonomiques [Farenc 96] pour les interfaces graphiques classiques. En ce qui concerne l'accessibilité des applications Web, cette limite a été estimée à 15% des règles pour l'inspection totale, 35% pour l'inspection partielle et 50% pour l'inspection manuelle [Cooper 99].

Cette étude montre que des règles peuvent être inspectées dès les premières phases de la conception et peuvent également être intégrées dans les outils de développement, évitant ainsi aux concepteurs d'introduire des problèmes d'accessibilité par erreur. Les résultats montrent que 20,41% des règles (voir Tableau 10) peuvent être supportées, tous niveaux d'automatisation confondus, par WebRatio et VisualWade dès les phases de spécification.

La règle suivante « Associez le champ d'un formulaire avec son étiquette » (WCAG 1.0 n° 12.4) est un exemple typique de règles pouvant être inspectées automatiquement mais aussi être intégrées de manière proactive afin de créer du code directement accessible.

3.6. DISCUSSION ET CONCLUSION

De manière à gérer la multitude de règles ergonomiques existantes, une organisation de ces règles est indispensable et de nombreux travaux visant cet objectif ont été exposés dans la section 2.1.5. Ces travaux proposent une organisation des règles selon plusieurs dimensions telles que le critère ergonomique associé, le nom de la source, la plateforme cible, etc. Afin de pouvoir appliquer les règles ergonomiques, leur organisation doit permettre d'identifier quand et comment elles peuvent être utilisées pendant la conception et l'évaluation d'un système [Scapin 00b]. Un moyen de répondre à ce besoin est d'organiser les règles par objets de l'interface utilisateur, c'est-à-dire un ensemble d'éléments conceptuels qui représentent, de

manière abstraite, les éléments de l'interface utilisateur, puisque ces éléments peuvent être identifiés dans les différents artefacts produits tout au long du processus de conception.

L'organisation des règles ergonomiques par objets de l'interface utilisateur a été proposée dans des travaux antérieurs [Mariage 05b] [Scapin 00b]. Toutefois, la sémantique de ces objets n'a jamais été formellement définie dans l'objectif de classification des règles ergonomiques. En conséquence, lors de l'application de ces règles sur un système en cours de conception (d'évaluation), l'identification des objets de l'interface ne peut se faire que manuellement par le concepteur (l'évaluateur). De plus, les règles ergonomiques sont souvent interprétées directement sur les éléments de l'interface exprimés dans une technologie cible. Par exemple, en HTML, les règles relatives aux images seront interprétées directement comme étant relatives à la balise <img>. Si dans le cas des images, la sémantique de la règle originale n'est pas perdue (la traduction de l'élément conceptuel *image* vers la balise <img> conserve la sémantique), dans des cas plus complexes, cette sémantique n'est pas conservée. Par exemple, la règle suivante : « L'utilisation des index […] aide à l'orientation des utilisateurs. » (Source : EvalWeb, règle n°39, voir Annexe D) est relative au concept de *menu*. Sur une page Web HTML, un menu peut apparaître sous la forme d'une liste de liens, d'une suite de liens adjacents, d'une image cliquable, etc. Un menu n'est donc pas aisément identifiable et une tentative d'interprétation de la règle ne conserverait pas la sémantique. Par exemple, décliner cette règle en plusieurs sous-règles relatives aux listes de liens, aux suites de liens adjacents, et aux images cliquables ne permet pas de conserver la sémantique de l'élément original menu.

Pour être comprises, appliquées à bon escient et vérifiées sur les artefacts produits dans le cycle de vie de l'application Web, les règles ergonomiques doivent être interprétées et décrites de manière formelle. Cette formalisation nécessite de caractériser, de manière précise et non ambiguë, les **différents objets de l'interface** sur lesquels portent les règles ergonomiques afin de permettre leur identification sur les artefacts évalués. Ces objets de l'interface ont été formellement décrits dans notre ontologie.

Une ontologie formalise généralement non seulement la connaissance relative à un domaine d'application (concepts, propriétés des concepts, relations entre concepts) mais également les axiomes et règles d'inférence sur cette connaissance dans l'objectif de faire du raisonnement automatique : on parle alors d'ontologie « heavyweight ». Lorsque les axiomes et les règles d'inférences ne sont pas décrits, l'ontologie est désignée comme « lightweight » [Corcho 02]. L'ontologie que nous avons présentée a été établie à l'aide de deux corpus de règles : les règles de navigation issues du projet EvalWeb et les règles d'accessibilité du W3C/WAI. Puisque nous n'avons défini ni axiomes ni règles d'inférences sur les concepts établis à partir de ces corpus, notre ontologie est considérée comme « lightweight ». Cette ontologie forme un vocabulaire de termes qui permet de formaliser l'ensemble des concepts relatifs à l'interface mentionnés dans les règles ergonomiques.

Dans la littérature, des **ontologies pour le développement d'une application Web** ont été proposées [Plessers 05] [Rossi 06]. L'ontologie WAfA (Web Authoring for Accessibility) [Plessers 05] est utilisée pour annoter manuellement les pages Web en cours de conception, l'objectif étant d'apporter de la sémantique et d'indiquer le rôle des éléments contenus dans une page Web afin de faciliter la lecture d'une page par un lecteur d'écrans. L'ontologie proposée par Rossi et al. [Rossi 06] formalise les éléments abstraits de l'interface dans l'objectif de proposer un vocabulaire sur lequel s'appuyer pour concevoir l'interface abstraite de l'application.

Nous avons choisi de formaliser les éléments de l'interface avec une ontologie, d'autres travaux suivent quant à eux une approche dirigée par les modèles pour formaliser ces mêmes éléments [Moreno 08], [Baresi 06], [Pastor 06] ou [Ceri 00]. Ces éléments d'interface sont utilisés pendant l'édition des modèles conceptuels. Nous pouvons considérer que l'ensemble de ces éléments conceptuels constitue un vocabulaire et par conséquent une ontologie, même si toutefois, les relations (d'agrégation ou hiérarchiques) entre chacun des éléments ainsi que leurs propriétés ne sont pas clairement définies.

Le point commun des ontologies référencées ci-dessus est que la connaissance exprimée est relative à la conception des applications Web. Ces ontologies ne permettent pas de décrire les connaissances qui peuvent être énoncées dans les règles ergonomiques. Par exemple, certains concepts ne sont pas formalisés tels que le concept d'aide à la navigation et il est par conséquent impossible de décrire toutes règles ergonomiques relatives aux aides à la navigation avec ces ontologies.

Plusieurs propriétés permettent d'**évaluer la qualité d'une ontologie** [Gómez-Pérez 01] [Gómez-Pérez 95] [Gruninger 95] : la *vérification syntaxique* des définitions (erreurs de mots-clés dans les définitions, présence/absence de documentation, absence de cycles entre plusieurs définitions, etc.), la *consistance* (les définitions des concepts ne se contredisent pas), la *complétude* (la définition de chaque concept doit être complète et l'ensemble des concepts doit couvrir le domaine désiré), la *concision* (toute information doit être utile et précise). Enfin, pour que l'ontologie puisse être diffusée le plus largement possible, la documentation l'accompagnant est indispensable et doit également être évaluée : elle doit renseigner notamment sur les informations générales de l'ontologie, donner la définition informelle de chaque concept, fournir des exemples, donner accès à une foire aux questions (FAQ), etc.

L'ontologie que nous avons développée n'est actuellement pas encore validée. Une ontologie, par définition, doit faire l'objet d'un consensus entre plusieurs personnes puisqu'elle formalise un vocabulaire qui doit être compris par une communauté, par exemple, dans notre cas, les experts en utilisabilité Web. Bien que notre ontologie ait été développée sur la base de deux corpus et de plusieurs sources, nous sommes conscients que les termes utilisés pour désigner les concepts doivent être validés par plusieurs personnes. La validation de l'ontologie est une perspective de nos travaux de thèse.

Notre ontologie peut être vue comme un moyen de **formaliser et d'organiser** les règles ergonomiques par rapport aux éléments de l'interface utilisateur. Elle forme ainsi un cadre auquel les règles ergonomiques sont rattachées et où la prise en compte de nouvelles règles ergonomiques est facilitée : pour ajouter une règle à ce cadre, il suffit de l'associer aux concepts de l'ontologie concernés. En outre, l'ontologie peut être enrichie avec de nouveaux concepts, par exemple, pour prendre en compte de nouveaux corpus de règles. Grâce à la nature de l'ontologie, l'introduction d'un nouveau concept dans une hiérarchie déjà existante permet de le rattacher automatiquement à un ensemble de règles ergonomiques, ces mêmes règles auxquelles sont rattachés les concepts parents (c'est-à-dire les concepts de plus haut niveau dans la hiérarchie). Par exemple, si un nouveau type de liens doit être inclus dans l'ontologie, nous le ferions naturellement hériter du concept *Link*. La conséquence directe est que les règles rattachées au concept *Link* seront automatiquement valables pour ce nouveau concept. Ce mécanisme facilite et allège ainsi la maintenance de l'ontologie en assurant que les associations existant entre les règles restent intactes malgré l'introduction de nouveaux concepts.

Sur la base de cette ontologie, nous avons pu donner une **estimation des règles potentiellement vérifiables** par des outils de conception existants manipulant des modèles. Ces outils ne réalisent pas d'inspection automatique sur les artefacts produits en cours de conception. Toutefois, nous avons estimé que plus de 20% des règles d'accessibilité du WCAG 1.0 pourraient être supportées (tous niveaux d'automatisation confondus) par WebRatio et VisualWade dès les phases de spécification, sur la base des artefacts manipulés par ces outils (par exemple, modèle de navigation). Ces résultats ont été obtenus sur la base de l'ontologie que nous avons établie, en mettant en correspondance les concepts de cette ontologie avec les concepts des artefacts manipulés par les outils de conception. Cette étude a ainsi démontré que les outils de conception actuels pourraient proposer une inspection automatique de certaines règles sur des artefacts en cours de conception.

4. Méthode d'évaluation ergonomique tout au long du cycle de vie de l'application

Résumé

Dans le chapitre précédent, nous avons présenté une ontologie pour l'organisation des règles ergonomiques autour des éléments de l'interface d'une application Web.

Nous présentons dans ce chapitre une méthode permettant l'évaluation ergonomique des artefacts produits tout au long du cycle de vie de l'application Web. Cette méthode se base sur l'ontologie qui permet de faire le lien entre les règles ergonomiques et les artefacts à évaluer. L'utilisation de cette méthode permet d'inspecter de manière systématique les artefacts produits et de déceler les éventuels problèmes ergonomiques.

Ce chapitre est organisé comme suit :

4.1 Introduction

4.2 Vue générale de la méthode

4.3 Collecte et articulation des données pour l'évaluation

4.4 Paramétrage de l'évaluation

4.5 Evaluation

4.6 Discussion : l'inspection combinée d'artefacts

4.7 Conclusion

4.1. INTRODUCTION

Dans le chapitre précédent, nous avons vu que les règles ergonomiques pouvaient être organisées et décrites autour des éléments de l'interface d'une application Web. Leur vérification consiste alors en une identification et une analyse des propriétés de ces éléments sur l'application courante pour s'assurer que les règles sont bien respectées.

Les artefacts produits dans le cycle de vie sont construits et utilisés afin d'aboutir à l'application Web finale et reflètent par conséquent, à des degrés différents de modélisation, l'interface utilisateur finale de l'application. Par conséquent, une inspection ergonomique sur chacun de ces artefacts peut être envisagée pour s'assurer de la qualité ergonomique tout au long du cycle de vie de l'application. Toutefois, les artefacts produits sont de différente nature (documents papier, modèles, code de l'application, etc.) et utilisent une syntaxe qui leur est propre (grammaires différentes, utilisation du langage naturel, etc.) faisant de l'inspection ergonomique une tâche complexe. Un travail d'adaptation des artefacts doit donc être mené afin de rendre possible cette inspection ergonomique.

En outre, pour pouvoir appliquer les règles ergonomiques de manière systématique, un travail de « formalisation » des règles est indispensable pour identifier d'une part les éléments de l'interface utilisateur à vérifier et d'autre part, la logique d'évaluation.

Dans ce chapitre, nous présentons une méthode supportant ce travail de formalisation des règles et des artefacts pour la vérification automatique des règles ergonomiques.

Dans la section 4.2, nous présentons une vue générale de la méthode. Nous présentons ensuite en détail chaque phase de cette méthode dans les sections 4.3, 4.4, et 4.5. Dans la section 4.6, nous discutons de l'application de cette méthode à plusieurs artefacts dans le cycle de vie, et enfin, nous concluons dans la section 4.7.

4.2. VUE GENERALE DE LA METHODE

Notre méthode propose de réaliser l'inspection ergonomique des artefacts produits dans le cycle de vie d'une application Web. L'utilisation de notre méthode n'est pas conditionnée par le

choix d'un cycle de vie particulier ; elle peut être appliquée quel que soit le cycle de vie et les méthodes de conception choisis par l'équipe de développement au sein du projet.

Les artefacts employés au cours d'un cycle de vie, au sein d'un projet donné, doivent toutefois être déclarés explicitement pour que notre méthode puisse être déployée. Pour cela, une phase préliminaire de traitement des artefacts choisis par l'équipe de conception (par exemple, modèle de tâche CTT, modèle de navigation StateWebCharts, prototype basse fidélité, fichier CSS) doit avoir lieu pour les rendre exploitables par notre méthode (voir **Phase A** de la Figure 20). Il s'agit de dégager les concepts manipulés par chacun des artefacts du cycle de vie, pour pouvoir associer chaque concept à un concept de l'ontologie. Par exemple, pour un artefact « modèle de navigation StateWebCharts », le concept « transition » est associé au concept « Link » de l'ontologie (voir Annexe A 5.1). Dans un contexte industriel, cette phase est réalisée lors de la première utilisation de notre méthode dans un projet. Puis, elle est réalisée à chaque fois qu'un nouvel artefact est intégré au cycle de vie, ou que les caractéristiques d'un artefact sont modifiées. Dans le cas du choix d'un cycle de vie en O, cette phase est réalisée avant d'entrer dans le cycle de vie (voir Phase 0. de la Figure 21).

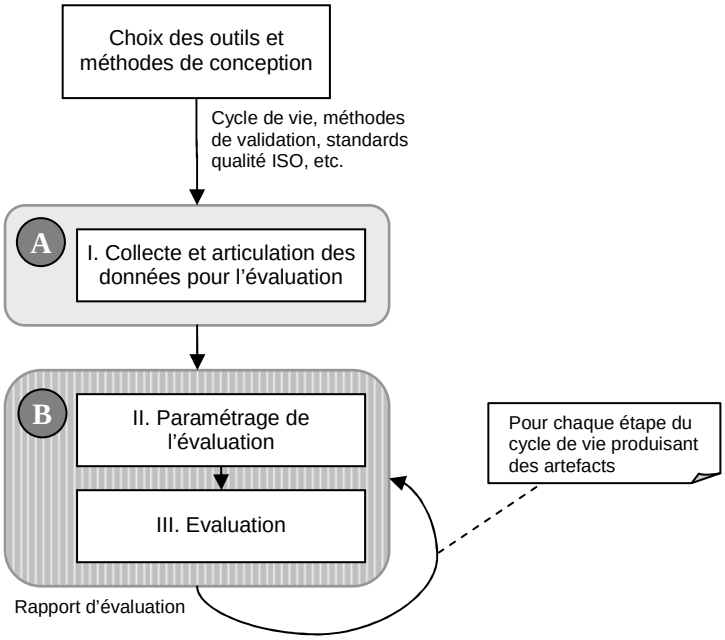


Figure 20 – Incorporation de la méthode à un cycle de vie

Une fois cette mise en place de notre méthode établie, la **phase B** (voir Figure 20) d'inspection peut être menée sur chacune des phases de conception productrices d'artefacts exploitables. Le paramétrage de l'évaluation consiste en la sélection des artefacts concernés par l'évaluation et la sélection des règles à évaluer. La phase d'évaluation est l'évaluation proprement dite, à l'issue de laquelle un rapport d'évaluation est produit automatiquement et est analysé par l'évaluateur. Dans le cas d'un cycle de vie en O, la phase B peut se dérouler dans toutes les phases de conception à l'exception de la phase 1 d'Expression des besoins (voir Figure 21) qui ne produit pas d'artefacts exploitables. En effet, la phase 1 consiste en la rédaction d'un cahier des charges : aucune règle ergonomique ne régit les informations qu'il contient car l'application Web n'y est décrite que trop abstraitement (par exemple « L'application doit être accessible 24h/24 »). Il est recommandé de répéter la phase B d'inspection jusqu'à ce que le rapport d'évaluation ne présente plus d'erreurs, c'est-à-dire que toutes les règles sélectionnées soient respectées pour les artefacts choisis.

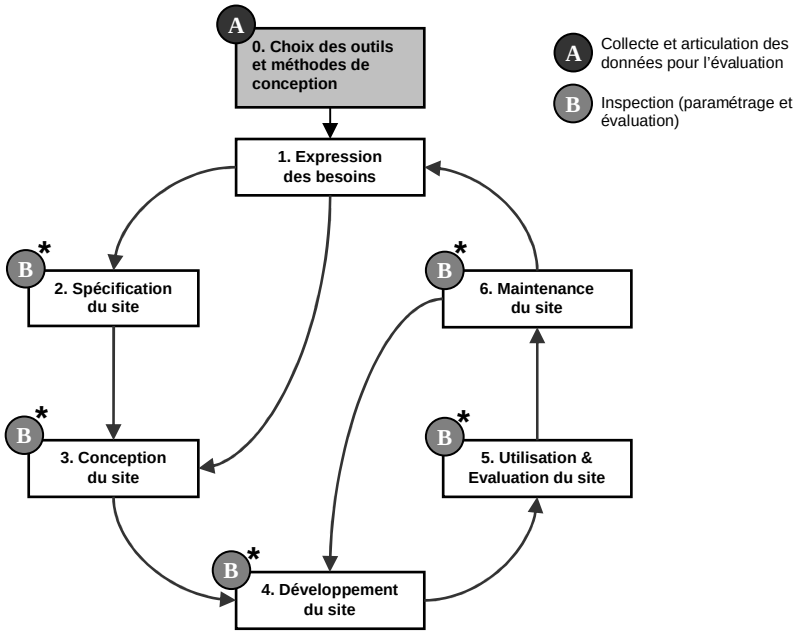


Figure 21 – Exemple d'application de notre méthode dans le cadre d'un cycle de vie en O [Scapin 00a]

Pour que les concepteurs Web puissent profiter des atouts de notre méthode, un mode opératoire particulier doit être suivi. Cette section présente d'abord le rôle et la responsabilité que les intervenants dans le processus de conception endossent dans le cadre de notre méthode (voir section 4.2.1). Elle présente ensuite les informations requises pour la réalisation des activités de mise en place (voir section 4.2.2) et d'utilisation de notre méthode (voir sections 4.2.3 et 4.2.4).

4.2.1. Intervenants

La méthode d'évaluation que nous proposons fait intervenir plusieurs compétences avec les responsabilités et les tâches qui leur sont associées : compétences en ergonomie, en informatique, et en évaluation (voir Tableau 11). L'ergonome trace les lignes directrices en termes de qualité ergonomique, l'informaticien connaît les artefacts manipulés sous un angle technique, et l'évaluateur réalise et analyse les inspections. Leurs activités sont coordonnées pour la mise en œuvre de notre méthode comme il est indiqué dans la Figure 22. Les sections suivantes entrent dans le détail de ces activités.

Tableau 11 – Les différentes compétences requises par la méthode d'évaluation

Compétence	Responsabilités	Tâches
Ergonomie	• Couverture des règles • Utilité des règles • Utilisabilité des règles	• Collecter les règles • Interpréter les règles • Récrire les règles
Informatique	• Opérationnalité des artefacts	• Dégager les concepts des artefacts • Lier les artefacts et l'ontologie
Evaluation	• Déroulement des évaluations • Respect des règles	• Planifier les évaluations • Réaliser les évaluations • Analyser les rapports • Remonter les erreurs pour leur correction

L'*expert en ergonomie* choisit l'ensemble des règles à évaluer, en fonction des exigences définies pour la qualité ergonomique de l'application. Ce choix est fait parmi l'ensemble des corpus de règles qu'il connaît et estime pertinents pour le domaine d'application. Il doit également faire en sorte que les règles choisies soient réécrites sous une forme exploitable, c'est-à-dire utile et utilisable : elles doivent être un support concret à la conception et à

l'évaluation (voir Phases I. ❶, I. ❷ et I. ❹ de la Figure 22). L'*expert en informatique* dispose des connaissances techniques nécessaires pour dégager les concepts fondateurs des artefacts de la méthodologie de conception choisie pour le projet. Il est également disposé à mettre ces concepts en parallèle de ceux qui figurent dans l'ontologie, pour que les artefacts puissent être exploités dans la méthode (voir Phases I. ❸ et I. ❺ de la Figure 22). L'*évaluateur* planifie, organise et effectue les évaluations ergonomiques à chaque étape du cycle de vie. Il y a une forte collaboration entre l'ergonome et l'évaluateur. En effet, l'ergonome est chargé de définir l'ensemble des règles ergonomiques de telle manière que ces règles couvrent l'ensemble des évaluations que l'évaluateur sera amené à réaliser. En fonction des exigences de qualité ergonomique définies par l'ergonome et le client dans le cahier des charges, l'évaluateur sélectionne l'ensemble des règles devant être respectées par l'artefact en cours d'évaluation. Il est enfin chargé de l'analyse des rapports d'évaluation : il traduit les erreurs constatées en suggestions de correction à destination de l'équipe de développement (voir Phases I. ❻ à I. ❾ de la Figure 22).

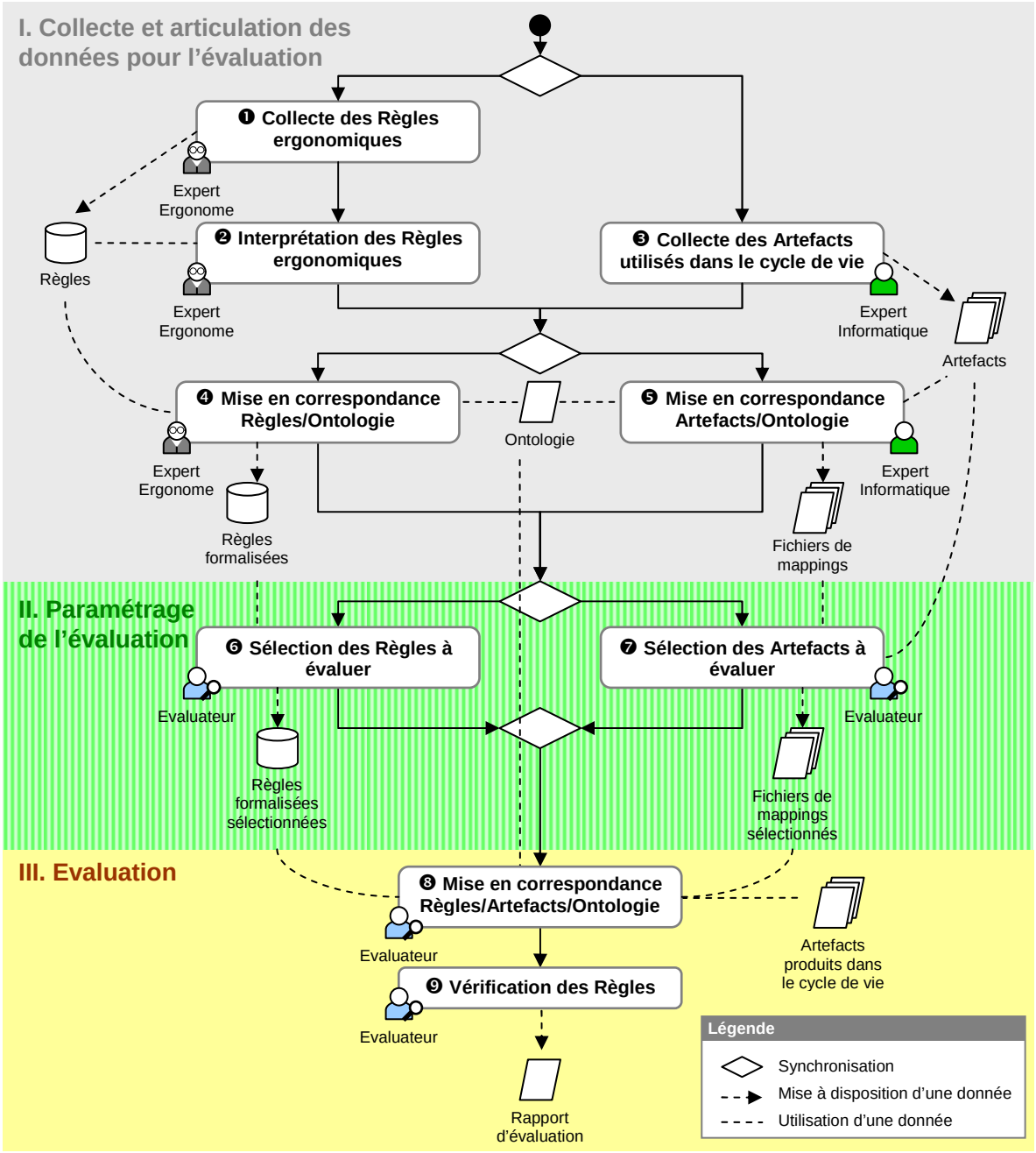


Figure 22 – Procédure d'inspection ergonomique basée sur des artefacts

4.2.2. Collecte et articulation des données pour l'évaluation

La première phase de notre méthode, appelée *Collecte et articulation des données pour l'évaluation* (voir Figure 22 I.), consiste à identifier les données nécessaires à l'évaluation et à les configurer afin de les rendre exploitables pour l'évaluation. L'évaluation consistant à inspecter les règles ergonomiques sélectionnées sur les artefacts produits dans le cycle de vie, les données que nous devons identifier et configurer en premier lieu sont les règles et les artefacts.

Les règles ergonomiques sont sélectionnées à partir de corpus existants ou de règles isolées. Ce travail de collecte nécessite une bonne connaissance des règles ergonomiques du domaine d'application et requiert par conséquent des compétences en ergonomie. Les artefacts quant à eux sont recensés et sélectionnés en fonction du cycle de vie adopté. Une vue d'ensemble sur le cycle de vie et une connaissance technique des artefacts utilisés dans ce cycle sont nécessaires afin de déterminer lesquels sont assez expressifs pour pouvoir être inspectés. Pour réaliser cette tâche, des compétences en informatique sont par conséquent requises.

Outre la collecte des règles et des artefacts, un travail de préparation doit être effectué dans cette phase pour les rendre automatiquement exploitables par la suite. Les règles ergonomiques énoncées en langage naturel dans leur forme originale doivent être interprétées afin d'en donner une description formelle. Une règle sera ainsi décrite par rapport aux éléments de l'interface sur lesquels elle porte et par rapport à sa logique d'évaluation sur ces éléments. L'identification des éléments de l'interface se fait sur la base de l'ontologie que nous avons développée et détaillée dans le chapitre 3.

De même que pour les règles, les artefacts nécessitent d'être rendus inspectables. Notre méthode exige à ce stade une mise en correspondance entre les concepts utilisés pour la conception des artefacts et les concepts de l'ontologie. Même si les artefacts sont de différentes natures, ils doivent être évalués de manière systématique, c'est-à-dire que les règles doivent être vérifiées automatiquement sur ces artefacts, quelle que soit leur nature. Les règles étant décrites avec les concepts de l'ontologie, leur inspection nécessite de pouvoir retrouver ces concepts sur les artefacts, c'est-à-dire d'identifier à quels concepts de l'ontologie correspondent les concepts de chaque artefact. Aucune adaptation des artefacts conçus n'est ainsi requise pour utiliser notre méthode.

En résumé, voici les activités à mener dans cette phase. Les activités de *Collecte des Règles ergonomiques* (Figure 22 ❶) et d'*Interprétation des Règles ergonomiques* (Figure 22 ❷) consistent à collecter les règles ergonomiques du domaine d'application souhaité et à faire un travail préliminaire sur celles-ci pour les rendre compréhensibles et directement applicables. En parallèle, l'activité de *Collecte des Artefacts utilisés dans le cycle de vie* (Figure 22 ❸) consiste à recenser l'ensemble des artefacts qui vont être évalués. Une fois ces activités effectuées, les activités de *Mise en correspondance Règles/Ontologie* (Figure 22 ❹) et de *Mise en correspondance Artefacts/Ontologie* (Figure 22 ❺) peuvent s'effectuer en parallèle pour, respectivement, spécifier la logique d'évaluation des règles par rapport aux concepts de l'ontologie et obtenir ainsi des règles formalisées, et mettre en relation les concepts de chaque artefact avec ceux de l'ontologie dans un fichier dit « de mappings[24] ».

4.2.3. Paramétrage de l'évaluation

La deuxième phase, appelée *Paramétrage de l'évaluation* (Figure 22 II.), consiste en la configuration de l'évaluation. En fonction des besoins de l'évaluation, seul un sous-ensemble des règles et des artefacts sera pris en compte. Par exemple, si nous désirons évaluer la navigation à travers l'application, seules les règles de navigation ainsi que les artefacts dédiés à la navigation sont susceptibles d'être sélectionnés. Cette phase est constituée de deux activités qui peuvent être menées en parallèle : la *Sélection des Règles à évaluer* (Figure 22 ❻) ; et la *Sélection des Artefacts à évaluer* (Figure 22 ❼).

[24] En français : mises en correspondance

4.2.4. Evaluation

La troisième et dernière phase, appelée *Evaluation* (Figure 22 III.), consiste en l'évaluation proprement dite des artefacts. A cette étape de la méthode, les règles ainsi que les artefacts sélectionnés sont tous mis en relation avec l'ontologie. La logique d'évaluation d'une règle étant décrite en termes de conditions à satisfaire sur les concepts de l'ontologie (cette description formelle a été réalisée dans l'activité ❹ de *Mise en correspondance Règles/Ontologie*), nous devons, pour pouvoir évaluer un artefact, identifier les éléments de l'artefact correspondant à ces concepts, et les extraire de l'artefact pour leur évaluation. L'identification des éléments de l'artefact par rapport aux concepts de l'ontologie est rendue possible grâce aux mises en correspondance établies précédemment (dans l'activité ❺ de *Mise en correspondance Artefacts/Ontologie*). Cette activité, qui grâce à l'utilisation de l'ontologie, permet de faire le lien entre les règles à respecter et les artefacts à évaluer, est appelée *Mise en correspondance Règles/Artefacts/Ontologie* (Figure 22 ❽).

Une fois les éléments de l'artefact identifiés, ils sont évalués conformément à la logique d'évaluation de la règle, autrement dit, les conditions à satisfaire sont vérifiées. Cette évaluation est réalisée dans l'activité suivante de *Vérification des Règles* (Figure 22 ❾).

Chaque activité de cette méthode est détaillée dans cette section. Pour comprendre ces différentes activités, nous illustrerons cette méthode avec la règle de navigation suivante « Tester la navigation » issue du corpus de règles EvalWeb (voir Annexe D) sur un artefact modèle de navigation StateWebCharts [Winckler 03]. Cette règle a été choisie pour la raison suivante : elle est abstraite et par conséquent sujette à interprétation.

4.3. COLLECTE ET ARTICULATION DES DONNEES POUR L'EVALUATION

La phase de collecte et articulation des données pour l'évaluation est un travail préalable de recueil et de configuration des données afin de rendre ces dernières exploitables pour les évaluations à mener dans le cycle de vie. Compte tenu du fait qu'une fois les données configurées pour l'évaluation, ces dernières pourront être exploitées autant de fois que nécessaire dans la vie de l'application concernée, cette phase n'est réalisée qu'une fois pour chaque donnée, avant toutes évaluations. Toutefois, s'il s'avère que, pendant les phases d'évaluation, une donnée n'a pas été prise en compte (par exemple, un artefact), le recours à cette phase sera nécessaire pour l'intégrer.

La configuration des données se fait par rapport à l'ontologie développée et présentée au chapitre 3. Nous supposerons tout au long de ce chapitre que l'ontologie est fixe et n'évolue pas. Si toutefois, elle est amenée à évoluer (par exemple si un nouveau corpus de règles est publié ou si un nouveau type d'interface Web, comme les interfaces multimodales, doit être pris en compte), l'ontologie devra être complétée. Pour cela, la méthode que nous avons présentée dans la section 3.1 pour le développement de l'ontologie peut être appliquée.

Nous présentons dans cette section les activités qui composent cette première phase, à savoir l'interprétation des règles ergonomiques en section 4.3.1, la mise en correspondance règles/ontologie en section 4.3.2, et la mise en correspondance artefacts/ontologie en section 4.3.3.

4.3.1. Interprétation des règles ergonomiques

Dans certains cas, une règle ergonomique ne se prête pas directement à une automatisation. Par exemple, la règle de navigation choisie « Tester la navigation » est trop imprécise pour pouvoir être évaluée de manière automatique. Elle doit être interprétée afin que sa signification soit concrète, c'est-à-dire qu'elle ne doit pas être trop générale (l'évaluation doit explicitement cibler les éléments de l'interface à évaluer) et non ambiguë (l'évaluation à réaliser sur ces éléments ne prête pas à confusion).

Une interprétation possible de cette règle pourrait être par exemple : « Chaque page doit contenir au moins un lien ». Cette interprétation peut être considérée comme valide dans le sens où une page ne contenant pas de liens ne permet pas de poursuivre la navigation et notamment, de revenir sur ses pas (si toutefois l'utilisation du bouton « back » du navigateur est toujours possible, le recours à cette fonction peut être considéré comme un problème de conception). Cette interprétation n'est qu'une interprétation parmi d'autres mais dans la mesure du possible, nous devons traiter tous les cas. Le Tableau 12 montre les interprétations que nous avons données pour cette règle.

Tableau 12 – Interprétation de la règle « Tester la navigation »

N° de la règle	Intitulé
1.	Chaque page doit contenir au moins un lien
2.	Chaque lien doit mener vers une page existante
3.	Le libellé d'un lien doit être explicite pour que l'utilisateur ait une idée de l'information qui va lui être affichée
4.	Toute page de l'application doit pouvoir être atteinte

La règle concrète 1. consiste en la vérification de la présence d'un lien dans une page. Cette vérification peut être considérée comme une vérification syntaxique et peut être traitée de manière automatique. Ce n'est pas le cas de la règle 3. qui consiste à vérifier que le libellé d'un lien est sémantiquement correct, à savoir, qu'il renseigne suffisamment l'utilisateur pour que ce dernier puisse savoir si l'activation de ce lien lui affichera l'information désirée. Cette vérification, bien que possible, ne peut qu'être réalisée par un évaluateur humain.

L'interprétation est un processus subjectif (deux évaluateurs pourraient avoir différentes interprétations pour une même règle) et peut créer une distance sémantique entre la règle ergonomique originale et la méthode à utiliser pour vérifier qu'elle est respectée.

4.3.2. Mise en correspondance Règles/Ontologie

La logique d'évaluation d'une règle est spécifiée en termes de conditions à vérifier sur les éléments de l'interface utilisateur. Par conséquent, une première étape est d'identifier les éléments de l'interface utilisateur concernés par chacune des règles concrètes. Cette identification se fait par rapport à l'ontologie que nous avons établie dans le chapitre 3 précédent et a été discutée dans la section 3.3. Par exemple, la règle 1. (cf. Tableau 12) concerne les concepts *Page* et *Link* de l'ontologie. Dans la dernière colonne du Tableau 13, nous avons identifié, pour chacune des règles concrètes, les concepts de l'ontologie mis en jeu.

Tableau 13 – Ecriture des règles concrètes pour la règle « Tester la navigation »

N° de la règle	Ecriture	Concepts
1.	Pour tout *p* de type *Page*, si *p.contientAuMoinsUnLien() = faux* alors la règle n'est pas respectée	Page, Link
2.	Pour tout *l* de type *Link*, si *l.target = null* alors la règle n'est pas respectée	Link
3.	Non vérifiable automatiquement	Link
4.	Pour tout *p* de type *Page*, si il n'existe pas de lien *l* tel que *l.target = p* alors la règle n'est pas respectée	Page, Link

Une fois que l'identification est faite, la logique d'évaluation peut être spécifiée. Par exemple, dans la règle 1., la condition à évaluer porte sur la présence d'au moins un lien dans chaque page de l'application. Cette règle peut s'écrire comme suit : « Pour tout *p* de type *Page*, si *p.contientAuMoinsUnLien() = faux* alors la règle n'est pas respectée ». Le Tableau 13 donne une écriture des règles concrètes présentées dans la section précédente.

L'écriture formelle des règles peut être réalisée en général via l'utilisation de langages déclaratifs tels que la logique temporelle [Farenc 99], un langage XML dédié [Beirekdar 04] [Leporini 06] [Abascal 04], le langage XSLT [Centeno 07], ou encore des langages logiques tels que Prolog. Afin de se détacher volontairement d'un langage de description de règles, nous avons utilisé dans le Tableau 13 un pseudo langage proche d'un langage de programmation

déclaratif pour spécifier la logique d'évaluation des règles ergonomiques. Ces règles ainsi décrites formalisent celles énoncées en langage naturel et présentées dans le Tableau 12.

Dans certains cas, les vérifications ne peuvent être réalisées que par un évaluateur humain et il n'est pas possible de spécifier la logique d'évaluation de manière formelle. C'est le cas ici de la règle 3. dont l'objet est de vérifier que le libellé du lien est sémantiquement correct. Puisque cette vérification ne peut pas être automatisée, une stratégie possible pour l'écriture de telles règles est d'identifier les concepts à vérifier (ici, la règle 3. porte sur des concepts de type Link) et de lancer une alerte pour chaque lien détecté dans un artefact en indiquant précisément que la vérification doit être faite manuellement par l'évaluateur (l'alerte pourrait être ici : « Vérifiez que le libellé du lien est explicite pour l'utilisateur »).

Pour aider à l'écriture des règles, il est possible de mettre à disposition des ergonomes une bibliothèque de méthodes utilitaires. Lorsque la logique d'évaluation d'une règle ne peut pas être explicitée en une ligne, une de ces méthodes utilitaires pourra ainsi aider à son écriture. Par exemple, la règle 1. utilise la méthode utilitaire « contientAuMoinsUnLien() » qui récupère le contenu d'une page et itère jusqu'à trouver un concept de type Link. Si cette page n'en contient pas, cette méthode retourne *faux*. L'implémentation de cette méthode pourrait être la suivante :

```
contents <- p.contents;  // p est la page courante
Pour i de 0 à taille(contents) faire
   Si contents[i] est de type Link
   alors retourner vrai;
   i <- i + 1;
fin pour;
retourner faux;
```

4.3.3. Mise en correspondance Artefacts/Ontologie

Dans cette activité, il s'agit de mettre en correspondance les concepts d'un artefact avec ceux de l'ontologie. Nous illustrons un exemple de mises en correspondance dans le Tableau 14 avec trois artefacts : un modèle de navigation StateWebCharts (formalisme abordé en détail dans la section 4.3.3.2), un fichier CSS et un fichier HTML. Les concepts de ces artefacts sont mis en correspondance avec les concepts *Page* et *Link* de l'ontologie.

Tableau 14 – Exemple de mises en correspondance avec les concepts *Page* et *Link*

Concept de l'ontologie	StateWebCharts	CSS	HTML
Page	Etat statique *ou* dynamique	–	<html>
Link	Transition Utilisateur	a:link	<a>

Les mises en correspondances les plus simples sont celles qui font correspondre exactement un concept de l'artefact considéré à un concept de l'ontologie (correspondance 1-1). C'est le cas par exemple de la mise en correspondance {<*html*>, *Page*} ou encore {*Transition utilisateur*, *Link*}. Nous trouvons également des mises en correspondances plus complexes telles que {*Etat statique*, *Page*} et {*Etat dynamique*, *Page*} : à deux concepts de la notation StateWebCharts correspond un unique concept de l'ontologie, celui de *Page* (correspondance 2-1). Enfin, nous trouvons aussi dans cet exemple, un cas où il n'y a pas de correspondances possibles : le concept de *Page* ne correspond à aucun élément du langage CSS.

Le problème de la mise en correspondance a été largement traité, dans le domaine de l'*Ingénierie Dirigée par les Modèles* (IDM) [Schmidt 06], dans l'objectif de réaliser des transformations de modèles. Gardner et al. [Gardner 03] mettent en évidence les différents cas de mise en correspondance (ces auteurs utilisent toutefois le terme « transformation » puisque c'est leur objectif final) :

- **Correspondance 1-1 :** un élément du modèle source correspond à un et un seul élément du modèle cible ; ce type de correspondances est la plus simple ;
- **Correspondance 1-m :** un élément du modèle source correspond à plusieurs éléments du modèle cible ; ce cas se présente lorsque, par exemple, un élément du modèle source possède plusieurs correspondants clones ;

- **Correspondance m-1 :** plusieurs éléments du modèle source correspondent à un seul élément du modèle cible ; c'est le cas par exemple lorsqu'un élément du modèle cible correspond à la composition de plusieurs éléments du modèle source ;
- **Correspondance m-n :** plusieurs éléments du modèle source correspondent à plusieurs éléments du modèle cible ; ce cas est la généralisation des deux cas précédents.

Bien que ces mises en correspondance puissent être établies de manière simple et de manière déclarative dans la plupart des cas, elles peuvent cependant être complexes selon la transformation à effectuer. Par exemple, il arrive qu'un élément du modèle source corresponde à un élément du modèle cible ayant une structure complexe, structure qu'il faudra construire pas à pas, de manière impérative. Ces problèmes ayant été détaillés dans plusieurs travaux (par exemple, [Gardner 03] [Jouault 05] [Tratt 05]), nous n'en discuterons pas ici. Toutefois, nous présenterons un exemple de mise en correspondance complexe lors de la construction du modèle de l'application abstraite à partir d'un modèle StateWebCharts en section 4.5.1.1.

Le problème de la mise en correspondance tel que défini dans le cadre de la transformation de modèles est équivalent au problème de la mise en correspondance entre un artefact et l'ontologie si l'on considère ces deux derniers comme des modèles. Nous nous appuyons ainsi sur les travaux existants dans le domaine de la transformation de modèles pour supporter la mise en correspondance entre les artefacts et l'ontologie.

Pour cela, nous présentons tout d'abord en quoi consiste la transformation de modèles, comment elle est réalisée et comment nous pouvons utiliser ces techniques de transformation de modèles pour faire la mise en correspondance. Nous présentons ensuite un exemple de mise en correspondance avec un artefact modèle de navigation StateWebCharts [Winckler 03].

4.3.3.1. Transformation de modèles

L'*Ingénierie Dirigée par les Modèles* (IDM) est un paradigme de développement qui place les modèles au centre des activités. Cette approche à base de modèles est née du besoin constant d'abstraction afin de pouvoir maîtriser et gérer la complexité des systèmes développés, à travers la modélisation. Une telle approche doit toutefois s'inscrire dans un cadre permettant l'utilisation effective des différents modèles ainsi que leur manipulation, et où la transformation de modèles joue un rôle clé [Schmidt 06].

L'initiative la plus complète à l'heure actuelle, pour fournir un cadre à l'IDM, est celle proposée par l'OMG[25] et appelée *Model-Driven Architecture* (MDA). Elle fournit une architecture dans laquelle les différents modèles sont organisés par niveaux : M0, M1, M2 et M3 (voir Figure 23).

Au niveau *M0*, nous trouvons le monde réel, c'est-à-dire les données comme par exemple une application informatique qui s'exécute (voir Figure 23 colonnes « Exemples »). Au niveau *M1*, nous avons les modèles qui décrivent les données du monde réel. Ces modèles peuvent correspondre, comme dans notre exemple, à des programmes Java, un modèle StateWebCharts, ou une page HTML. Au niveau *M2*, nous trouvons les méta modèles qui permettent de définir la structure des modèles de niveau M1. Chaque modèle est ainsi dit conforme à son méta modèle. Dans notre exemple, cela revient à dire par exemple qu'un programme Java est conforme à sa grammaire, autrement dit, qu'il est syntaxiquement correct. Au niveau M3, nous trouvons le méta méta modèle qui est le langage de définition des méta modèles : il définit la structure des méta modèles. La particularité d'un méta méta modèle est qu'il est conforme à lui-même, c'est-à-dire que sa description peut être faite par lui-même. Dans notre exemple, la grammaire du langage Java est décrite au moyen de la notation EBNF (Extended Backus-Naur Form). Les autres exemples donnés (Ontologie, StateWebCharts, CSS et HTML) suivent cette même organisation.

[25] Object Management Group, http://www.omg.org/

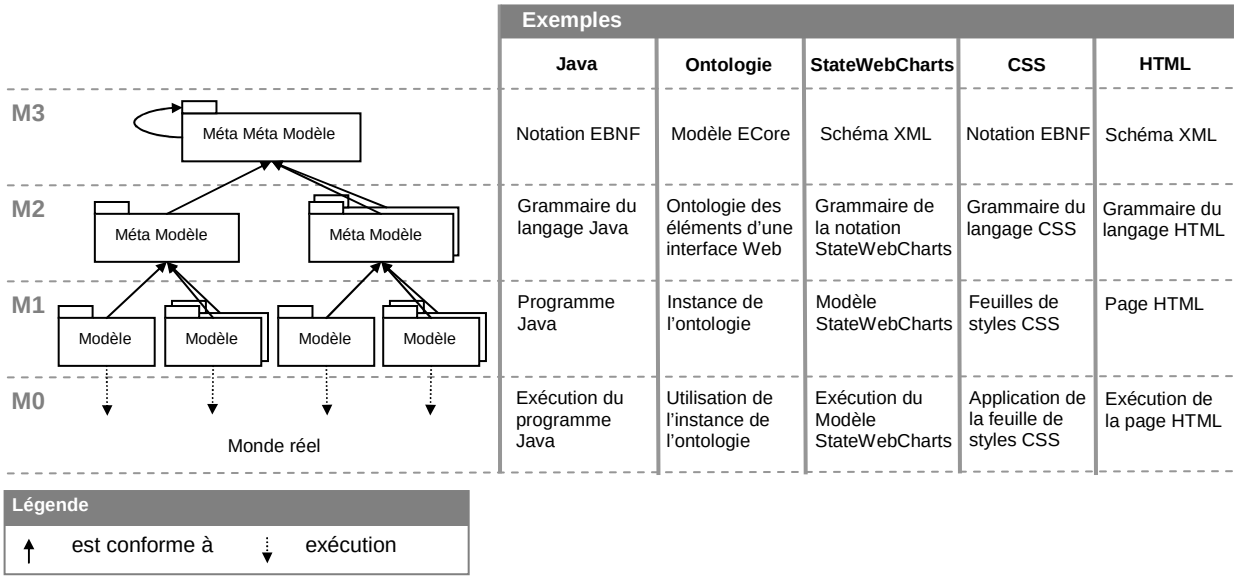


Figure 23 – Architecture à quatre niveaux proposée dans l'initiative MDA

Dans cette architecture, des transformations peuvent s'opérer. Il existe plusieurs approches pour la transformation [Czarnecki 03] ayant conduit à l'apparition de plusieurs langages de transformation [Tratt 05], mais à l'heure actuelle, il n'y a pas de consensus ni sur l'approche ni sur le langage à adopter pour réaliser la transformation de modèles. Par conséquent, nous avons choisi de présenter ici la transformation de modèles telle qu'elle est réalisée dans le langage ATL (Atlas Transformation Language) [Jouault 05] puisque ce langage a l'avantage de se positionner par rapport à l'architecture MDA. En outre, la contrainte forte imposée par la plateforme de notre outil a guidé notre choix sur ce langage (voir chapitre 5).

La transformation de modèles ATL est illustrée en Figure 24.

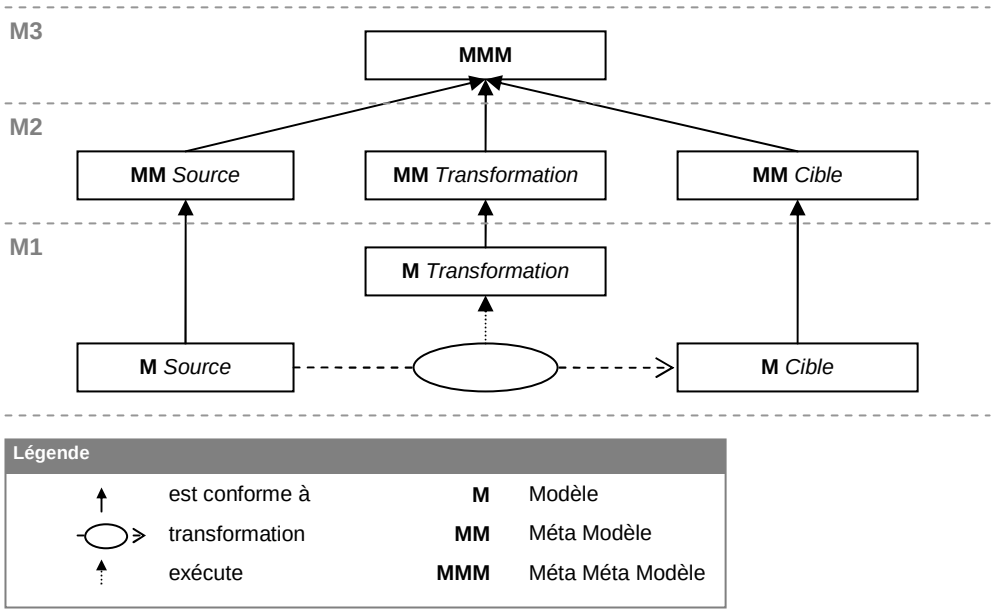


Figure 24 – Transformation de modèles (ATL)

Le modèle source **M** *Source* est transformé en un modèle cible **M** *Cible* selon les mises en correspondances définies dans le fichier de transformation **M** *Transformation*. Puisque dans le paradigme de l'IDM tout est modèle, le fichier de transformation est également un modèle. Ces trois modèles appartiennent au niveau M1 du MDA et sont conformes à leur méta modèle respectif (de niveau M2) : **MM** *Source*, **MM** *Cible* et **MM** *Transformation*. Enfin, ces trois

méta modèles doivent être conformes à un méta méta modèle **MMM** (de niveau M3) qui, en ATL, est le MOF[26].

Le point clé dans la transformation de modèles est le fichier de transformation qui contient toutes les informations utiles pour faire correspondre les éléments du modèle source aux éléments du modèle cible.

En considérant le méta modèle source comme l'artefact source et le méta modèle cible comme l'ontologie, nous pouvons nous placer dans le cadre de la transformation de modèles pour réaliser la mise en correspondance entre concepts de l'artefact et concepts de l'ontologie. Cette mise en correspondance est effectuée via le fichier de transformation que nous appellerons *fichier de mappings* (nous évitons le terme « transformation » puisque la transformation de modèles n'est qu'un moyen pour réaliser la mise en correspondance).

Un exemple de fichier de mappings est donné dans l'Annexe E. Ce fichier correspond aux mises en correspondance sur la notation StateWebCharts que nous illustrons dans la section suivante.

4.3.3.2. Application sur StateWebCharts

StateWebCharts (SWC) [Winckler 03] est une notation formelle basée sur la notation Statecharts [Harel 87] et dédiée à la modélisation de la navigation dans les applications Web. Un diagramme SWC est représenté sous la forme d'un diagramme hiérarchique d'état où, globalement, un état modélise une page Web et une transition entre deux états modélise un lien hypertexte entre deux pages. Il existe plusieurs types d'états et de transitions. La représentation visuelle des éléments de la notation SWC est montrée en Figure 25. Dans cette section, nous n'expliquons que succinctement les différents éléments de SWC (une description plus détaillée et formelle de SWC est donnée dans [Winckler 03]) et nous montrons à quels concepts de l'ontologie ils correspondent.

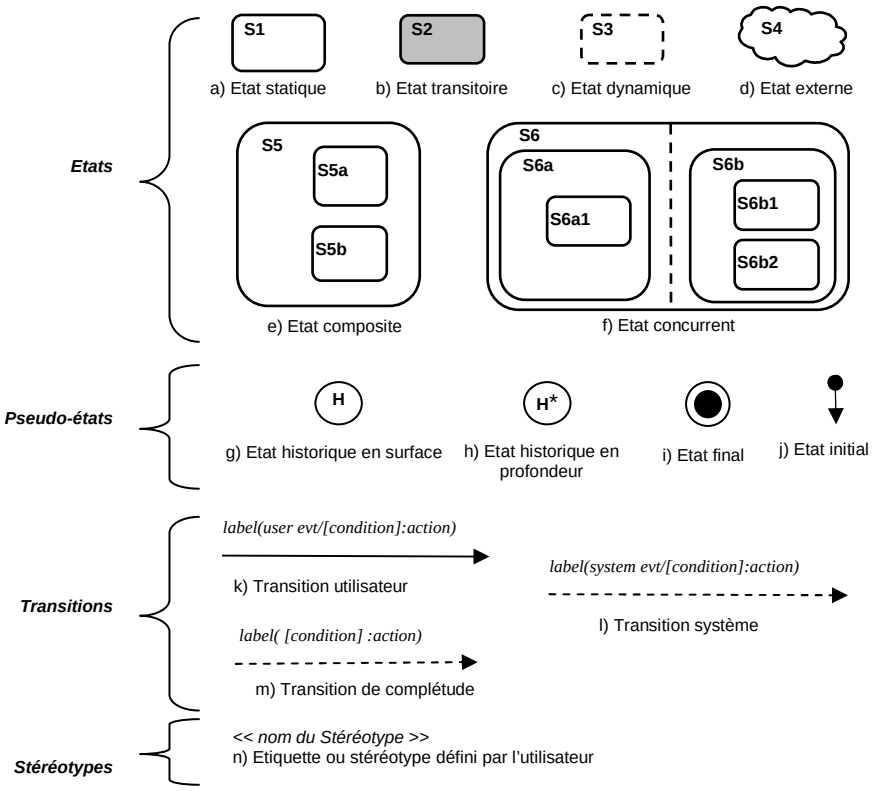


Figure 25 – Concepts de la notation StateWebCharts [Winckler 03]

[26] Le MOF, acronyme de « Meta-Object Facility », désigne le méta méta modèle utilisé dans MDA. Il est notamment connu pour être le méta méta modèle de la notation UML. La spécification du MOF peut être trouvée à l'adresse suivante : http://www.omg.org/mof/

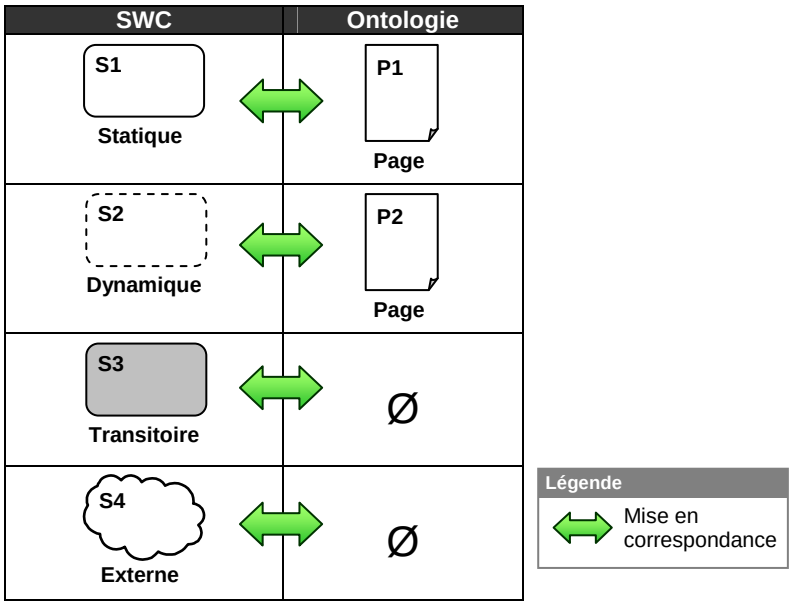
4.3.3.2.1. Etats

Dans le formalisme SWC, les états représentent des conteneurs pour objets (graphiques ou exécutables). Ces conteneurs sont habituellement des pages HTML pour les applications Web. Pour une meilleure compréhension de leur mise en correspondance, nous distinguons les états *basiques* (statique, dynamique, transitoire, externe), *composites*, et *concurrents*.

A) Etats basiques

La sémantique associée aux états basiques est la suivante : un état *statique* représente une page Web ayant un nombre fixe d'objets ; un état *dynamique* est une page Web dont le contenu a été généré dynamiquement ; un état *transitoire* représente un traitement côté serveur ; et un état *externe* représente une application Web externe (c'est-à-dire indépendante de l'application Web que nous modélisons). Cette sémantique fait que la mise en correspondance avec les concepts de l'ontologie est triviale (voir Tableau 15).

Tableau 15 – Mise en correspondance des états SWC

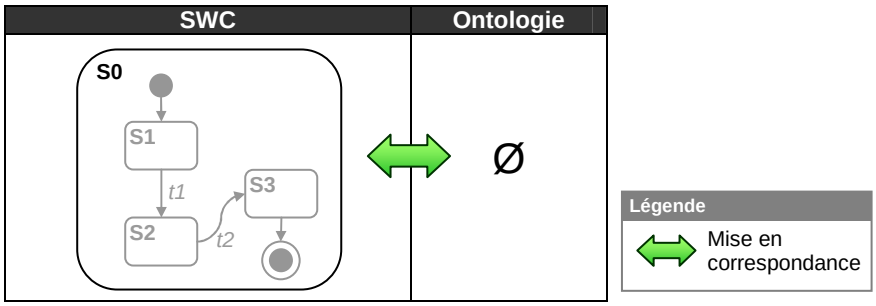


Un état de type statique (S1) ou dynamique (S2) correspond à une page selon notre ontologie (P1 et P2 respectivement). Un état transitoire (S3) n'a aucun correspondant puisque cet état représente une activité menée côté serveur n'ayant aucune représentation visuelle pour l'utilisateur. Un état externe (S4), puisque représentant une application externe indépendante de celle modélisée, n'a, lui non plus, aucun correspondant.

B) Etats composites

Un état *composite* est un état pouvant contenir des états fils et impose une hiérarchie d'états (l'état composite est appelé « super état » relativement à ses états fils) avec une sémantique formelle associée aux états fils et aux transitions entrantes et sortantes : à l'intérieur d'un état composite, un seul état est actif à la fois ; les transitions entrantes d'un état composite sont toutes dirigées vers l'état initial ; et les transitions sortantes d'un état composite sont toutes héritées par les états fils.

Tableau 16 – Mise en correspondance d'un état composite SWC



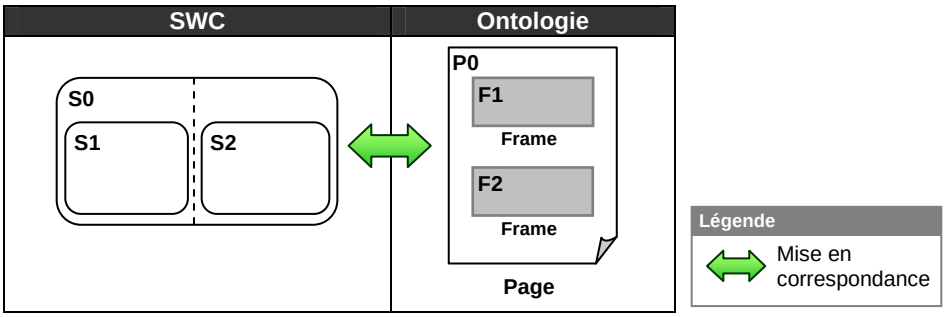
Conceptuellement, un état composite correspondrait à une super page, c'est-à-dire une page contenant d'autres pages. Un tel concept n'existe pas dans l'ontologie et par conséquent, un état composite (ici, S0 dans le Tableau 16) n'a pas d'équivalent dans l'ontologie. Cependant, nous verrons dans la section 4.5.1.1 que, par la sémantique associée à un état composite, les états fils doivent prendre en compte les transitions de leur état composite parent pour l'évaluation.

Nous pouvons noter que, bien que nous ayons représenté dans le Tableau 16 des états fils S1, S2 et S3, ainsi que des transitions t1 et t2 entre ces états, leur mise en correspondance n'est pas traitée ici (et n'est donc pas représentée dans la figure), mais dans les sections qui leur sont consacrées, à savoir « Etats basiques » (vus dans la section précédente) et « Transitions » (qui seront traitées plus loin).

C) Etats concurrents

Un état *concurrent* est un état contenant deux ou plusieurs régions concurrentes, chacune modélisant la navigation d'un module de l'application. Formellement, un seul état est actif dans chaque région à un instant donné.

Tableau 17 – Mise en correspondance d'un état concurrent SWC



La sémantique que nous venons de donner fait qu'un état concurrent correspond au concept d'une page Web dans l'ontologie, contenant des régions s'exécutant en parallèle. Ces régions correspondent au concept de frame. Dans le Tableau 17, l'état concurrent S0 contient deux régions concurrentes S1 et S2. Le détail de ces deux états n'est pas important et n'a pas été représenté dans cet exemple. Le concept correspondant à l'état concurrent S0 est par conséquent une page Web (ici P0) et les correspondants des régions S1 et S2 sont des frames (ici F1 et F2) contenus dans P0.

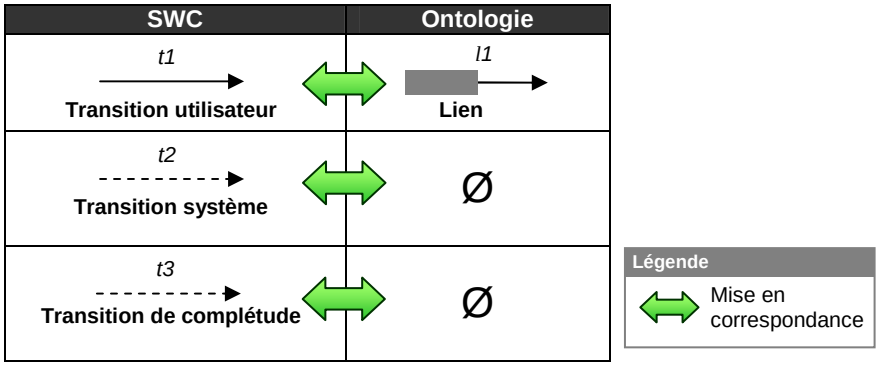
D) Pseudo-états

Les *pseudo-états* sont utiles pour l'exécution de la machine à états mais n'ont pas de représentation visuelle pour l'utilisateur : ils n'adressent donc aucun élément de l'interface utilisateur. Par conséquent, il n'existe pas de correspondants aux pseudo-états dans l'ontologie.

4.3.3.2.2. Transitions

L'utilisation de transitions permet de modéliser la navigation entre états. Lorsque l'utilisateur est à l'origine de l'activation d'une transition, une transition utilisateur (voir t1 dans le Tableau 18) est utilisée dans le diagramme ; lorsque c'est le système, une transition système ou de complétude (voir respectivement t2 et t3 dans le Tableau 18) est utilisée. Le concept de transition correspond naturellement au concept de lien hypertexte et la mise en correspondance est immédiate (voir Tableau 18).

Tableau 18 – Mise en correspondance des transitions SWC

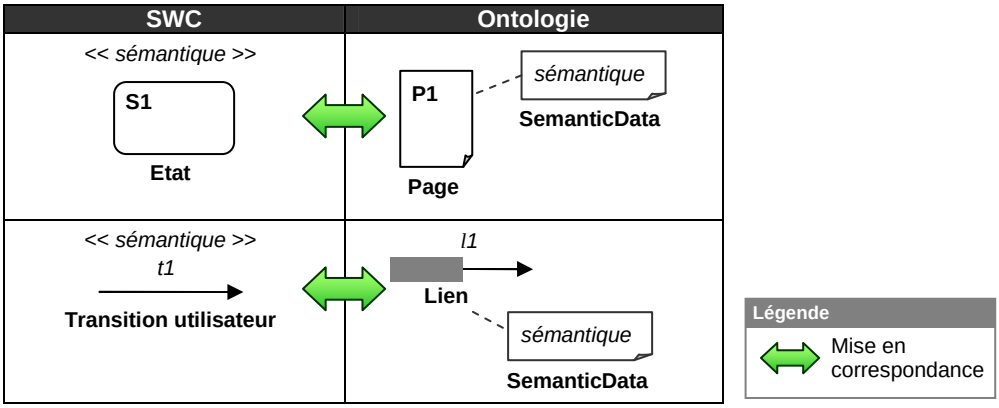


Une transition utilisateur (t1) correspond à un lien (le lien l1 est représenté dans le Tableau 18 par une ancre, c'est-à-dire la zone cliquable dans la page, représentée par un rectangle grisé, accompagnée d'une flèche supposée pointer vers la cible du lien). Une transition système (t2), ainsi que de complétude (t3), n'ont pas de correspondant dans l'ontologie puisqu'ils modélisent des actions déclenchées par le système.

4.3.3.2.3. Stéréotypes

Un stéréotype affecte une information supplémentaire à un concept donné, par exemple à un état ou une transition. La représentation graphique d'un stéréotype est un texte entre accolades situé au-dessus de l'élément (voir Tableau 19).

Tableau 19 – Mise en correspondance des stéréotypes SWC



Le concept de stéréotype correspond dans notre ontologie à une donnée sémantique (concept *SemanticData*) attachée à un autre concept. Dans le Tableau 19, l'état S1 ayant le stéréotype « sémantique » correspond à la page P1 à laquelle une donnée sémantique est attachée. De même, la transition t1 correspond au lien l1 auquel est également attachée une donnée sémantique.

4.4. PARAMETRAGE DE L'EVALUATION

Avant de pouvoir effectuer des évaluations, l'évaluateur doit paramétrer l'évaluation, d'une part en sélectionnant les règles ergonomiques, et d'autre part, en sélectionnant les artefacts qui vont être évalués. Ces paramètres (ensemble de règles et ensemble d'artefacts) peuvent être définis une fois pour toute avant l'évaluation. Dans ce cas, chaque évaluation contiendra les mêmes paramètres.

Nous présentons dans cette section les activités de sélection des règles ergonomiques et de sélection des artefacts.

4.4.1. Sélection des règles ergonomiques

Dans cette étape, les règles ergonomiques sont sélectionnées pour l'évaluation des artefacts. Un évaluateur peut choisir de n'évaluer qu'une partie des règles existantes selon ses besoins. Par exemple, seules les règles d'accessibilité sont sélectionnées si l'accessibilité de l'application doit être évaluée ; si certains critères ergonomiques sont importants, seules les règles relatives à ces critères sont sélectionnées.

4.4.2. Sélection des artefacts

Les artefacts qui doivent être évalués sont sélectionnés dans cette étape. Pour qu'un artefact puisse être évalué, il est nécessaire de fournir le fichier de mappings (voir section 4.3.3) qui contient les mises en correspondance avec les concepts de l'ontologie.

4.5. EVALUATION

La phase d'évaluation est constituée de deux activités : la première consiste en la mise en correspondance entre les règles, l'artefact à évaluer et l'ontologie, et la deuxième, en l'évaluation de cet artefact afin de détecter les règles qui ne sont pas respectées. Nous présentons ces deux activités dans cette section.

4.5.1. Mise en correspondance Règles/Artefacts/Ontologie

A partir des mises en correspondance règles/ontologie et artefact/ontologie, il est possible de déterminer et d'extraire les éléments de l'artefact à évaluer pour chaque règle. Puisque les règles sont exprimées avec les concepts de l'ontologie, chaque élément ainsi extrait de l'artefact est traduit en son concept correspondant (les attributs de chaque concept créé prennent les valeurs extraites de l'élément source de l'artefact).

Pour éviter cependant d'avoir à créer pour chaque règle les concepts de l'ontologie correspondant aux éléments de l'artefact, puis de les évaluer, nous avons choisi de mettre en correspondance tous les éléments de l'artefact pour ne plus raisonner que sur ce dernier. Nous obtenons ainsi, à partir de l'artefact source, un modèle équivalent (dans le sens où la connaissance ergonomique contenue dans l'artefact est la même que dans ce modèle) mais exprimé avec les concepts de l'ontologie. Nous avons appelé ce modèle par « modèle de l'application abstraite ».

La construction du modèle de l'application abstraite est réalisée à partir du fichier de mappings établi dans la phase de mise en correspondance Artefacts/Ontologie. Ce modèle de l'application abstraite est construit par transformation de modèles (voir Figure 24 page 76) où : le modèle source est l'artefact à évaluer, le fichier de transformation est le fichier de mappings, le modèle cible est le modèle de l'application abstraite, et le méta modèle cible est l'ontologie (le modèle de l'application abstraite étant une instance de l'ontologie, il est considéré comme un modèle de niveau M1, au sens de MDA ; l'ontologie, quant à elle, est un méta modèle de niveau M2).

Dans la section suivante, nous montrons la construction d'un modèle de l'application abstraite à partir d'un modèle SWC et de la mise en correspondance avec les concepts de l'ontologie (vue dans la section 4.3.3.2).

4.5.1.1. Application sur un modèle SWC

Le modèle SWC que nous prenons en exemple est illustré dans la partie gauche de la Figure 26. Il contient des états statiques (étiquetés de S1 à S6), un état composite (S0), un pseudo-état initial (lié à S1), un pseudo-état final (lié à S3), ainsi que des transitions utilisateur (étiquetées de t0 à t4).

Les mises en correspondance définies pour SWC dans la phase de Préparation de l'évaluation (voir section 4.3.3.2) établissent les relations suivantes (pour des raisons de lisibilité, nous choisissons de garder les mêmes indices pour les concepts de l'ontologie correspondants aux concepts SWC) :

a. Un état statique S_i correspond à une page P_i ;

b. Une transition utilisateur t_j correspond à un lien l_j ;

c. Un pseudo-état n'a pas de correspondant ;

d. Un état composite n'a pas de correspondant.

Cependant, même si un état composite n'a pas de concept correspondant dans l'ontologie (voir la relation d.), la sémantique associée à un tel état dans la notation SWC fait que ses états fils sont impactés par ce dernier (voir section 4.3.3.2) : les liens entrants (ici t0) d'un état composite (ici S0) sont tous redirigés vers l'état fils initial (ici S1) et les liens sortants (ici t3 et t4) sont tous hérités par les états fils (autrement dit, les états S1, S2 et S3 ont tous un lien vers S5 et un autre vers S6).

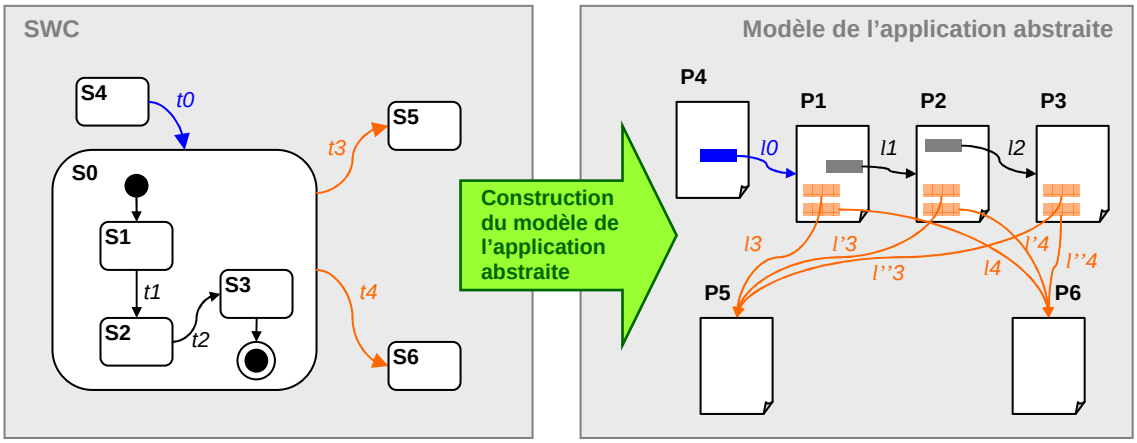


Figure 26 – Construction du modèle de l'application abstraite à partir d'un modèle SWC

Sur la base de ces mises en correspondance, nous obtenons le modèle de l'application abstraite (voir la partie droite de la Figure 26), où :

a. les pages P1 à P6 ont été construites à partir des états S1 à S6 ;

b. le lien l0 est construit à partir de la transition t0 et lie la page P4 à P1 (P4 est le correspondant de l'état S4, et P1 le correspondant de l'état initial S1) ;

c. les liens l1 et l2 sont construits à partir de t1 et t2 et lient respectivement les pages P1 à P2 et P2 à P3 (P1, P2 et P3 sont les correspondants respectifs de S1, S2 et S3) ;

d. les liens l3, l'3, et l''3, ainsi que l4, l'4, et l''4 sont construits à partir des transitions t3 et t4 ; ces transitions menant respectivement à S5 et S6 et étant héritées par S1, S2, et S3, les pages P1, P2 et P3 correspondantes ont tous des liens vers les pages P5 et P6 (correspondant respectivement aux états S5 et S6).

Nous pouvons noter que les liens l3, l'3 et l''3 (respectivement l4, l'4 et l''4) ont été créés à partir de la transition t3 (respectivement t4). Bien qu'une transition ne corresponde qu'à un lien (relation 1-1), nous avons un exemple ici d'une correspondance entre une transition et plusieurs liens clones (relation 1-n). Cette relation complexe est due à la sémantique d'un état composite qui oblige à construire, de manière impérative, un lien clone représentant la transition source. Cet exemple illustre ce que nous avons appelé « mise en correspondance complexe » dans la section 4.3.3.

4.5.2. Vérification des règles

La logique d'évaluation d'une règle ergonomique est spécifiée en termes de conditions sur un ou plusieurs concepts de l'ontologie. Par conséquent, pour vérifier qu'une règle est respectée, il faut évaluer les conditions portant sur ces concepts à partir du modèle de l'application abstraite construit dans l'étape précédente.

Si certains concepts ne peuvent pas être identifiés, les conditions relatives à ces concepts ne peuvent pas être évaluées. Dans ce cas-là, elles sont considérées comme étant satisfaites. Par exemple, si des règles concernent des objets de type vidéo mais que l'application Web modélisée n'en contient pas, ces règles sont supposées être respectées.

Lorsqu'une règle n'est pas respectée, une erreur est ajoutée dans le rapport d'évaluation avec une référence vers la règle originale (seule la règle originale a un sens pour l'évaluateur puisque la règle concrète n'est qu'une interprétation de la règle originale) et la cause de l'erreur. Nous verrons dans le chapitre 5 comment un tel rapport d'évaluation peut être construit.

Sur la règle « Tester la navigation » que nous avons prise pour exemple tout au long de cette section, l'évaluation porte sur les quatre règles concrètes issues de la phase d'interprétation (voir Tableau 12 page 73). Dans cette section, nous illustrons sur le modèle de l'application abstraite présenté précédemment en Figure 26 comment les conditions de ces règles sont évaluées.

La règle 1. « Chaque page doit contenir au moins un lien » fait intervenir les concepts *Page* et *Link*. La condition d'évaluation portant sur la présence d'au moins un lien dans le contenu d'une page, chaque page (de P1 à P6) est analysée. Puisqu'elles ne contiennent pas de liens, les pages P5 et P6 ne respectent pas la règle.

La règle 2. « Chaque lien doit mener vers une page existante » consiste à vérifier que, pour chaque lien, la cible est une page existante. Il se peut que certains liens référencent une page externe, c'est-à-dire une page faisant partie d'une autre application Web, indépendante de celle que nous concevons. Puisque nous ne maîtrisons pas cette page cible, nous supposerons que les liens faisant référence à des pages externes respectent la règle. Dans notre exemple, tous les liens remplissent la condition imposée par la règle 2. puisque chaque lien référence une page de l'application.

La règle 3. « Le libellé d'un lien doit être explicite pour que l'utilisateur ait une idée de l'information qui va lui être affichée » n'est pas vérifiable automatiquement. Dans ce cas, comme discuté dans la section 4.3.1, nous pouvons afficher une alerte dans le rapport d'évaluation sur chaque lien détecté (ou, pour éviter de rendre ce rapport illisible, choisir de ne lancer qu'une seule alerte pour l'ensemble des liens).

La règle 4. « Toute page de l'application doit pouvoir être atteinte » consiste à s'assurer que pour toute page, il existe un lien ayant pour cible cette page. Toutes les pages du modèle de l'application abstraite remplissent ici cette condition sauf la page P4 qui n'a aucun lien entrant. Cette règle n'est donc pas respectée.

En conséquence, nous voyons sur cet exemple que la règle « Tester la navigation » n'est pas respectée pour le modèle SWC conçu (voir Figure 26) si l'on considère l'inspection de toutes les règles concrètes produites (seule la règle 2. est respectée).

4.6. DISCUSSION : L'INSPECTION COMBINEE D'ARTEFACTS

Pour chaque étape du cycle de vie, un ou des artefacts sont produits pour satisfaire, à eux tous, l'objectif global de l'étape. Par exemple, en phase de spécification, l'objectif est de formaliser les besoins à travers la production de différents modèles (utilisateur, tâches, navigation, etc.) ; en phase de développement, des pages HTML et des fichiers CSS sont produits pour construire l'application finale ; et ainsi de suite. Dans une même étape du cycle de vie, nous pouvons être amenés à évaluer plusieurs artefacts de nature différente.

Deux artefacts peuvent modéliser un même élément de l'interface utilisateur finale selon un point de vue différent (par exemple, un lien est modélisé par une transition dans un modèle de navigation alors qu'il est modélisé par une balise <a> en HTML). Une cohérence est nécessaire entre les différentes représentations de ce même élément au sein de ces artefacts.

Deux artefacts A1 et A2 sont dits cohérents, si pour tout élément e1 ∈ A1 et e2 ∈ A2 (où e1 et e2 représentent le même élément de l'application), et pour toute règle r, nous avons :

Respecte(r, e1) == Respecte(r, e2)
où Respecte (r, e) == vrai si e respecte la règle r
et Respecte (r, e) == faux si e ne respecte pas la règle r

Pour l'heure, notre méthode ne supporte pas l'inspection simultanée de plusieurs artefacts, et par conséquent, elle ne vérifie pas la cohérence entre artefacts. L'évaluation de plusieurs artefacts dans une même étape garantit que les règles sont respectées sur chacun de ces artefacts indépendamment. Il est du ressort de l'évaluateur de comparer les rapports d'évaluation pour relever manuellement d'éventuelles incohérences. Au cours de cette section, nous étudions toutefois, par le biais d'exemples, les pistes de travail suivantes :

- Comment s'assurer de la cohérence entre différents artefacts produits au sein d'une même étape ?
- Comment s'assurer de la cohérence entre différents artefacts produits dans des étapes différentes ?
- Comment vérifier que des règles qui nécessitent l'inspection de plusieurs artefacts sont respectées ?

Pour répondre à ces questions, nous menons une discussion dans la section suivante sur un exemple où le cycle de vie choisi est celui de Scapin et al. [Scapin 00a].

Dans cette section, nous présentons un exemple sur lequel nous discutons des problèmes identifiés précédemment, relatifs à la cohérence des artefacts évalués dans le cycle de vie, mais également à la vérification de règles qui nécessitent l'inspection de plusieurs artefacts. Pour cela, nous supposerons que les deux premières phases de notre méthode sont réalisées une fois pour toute avant la phase d'évaluation. Autrement dit, les données sont toutes configurées pour l'évaluation, et les paramètres de l'évaluation ne changent pas dans le cycle de vie. Ainsi, l'exemple présenté ne tient compte que de la phase d'évaluation représenté ici dans le cycle de vie avec les deux activités constituant cette phase : la Mise en correspondance Règles/Artefacts/Ontologie et la Vérification des règles (voir Figure 27).

Nous avons choisi un exemple d'artefacts produits pour chacune des étapes du cycle de vie en O de Scapin et al [Scapin 00a] (voir Figure 27). Ainsi, dans l'étape d'expression des besoins, le cahier des charges est défini ; dans l'étape de spécification, des modèles de navigation sont produits ; dans l'étape de conception, des templates (ou patrons de page) pour la mise en page sont conçus ; dans l'étape de développement, les pages Web finales ainsi que leurs fichiers CSS de mise en page sont développées ; dans l'étape d'évaluation, les données d'usage de l'application sont recueillies ; et dans l'étape d'utilisation et d'évaluation, aucun artefact

n'est produit (cette étape représente les modifications d'implémentation à travers une redéfinition des besoins ou une nouvelle phase de développement).

L'évaluation de documents papiers tels que le cahier des charges dans l'étape d'expression des besoins n'est pas supportée par notre méthode. Par conséquent, nous ne tenons volontairement pas compte de cet artefact pour l'illustration de l'intégration de notre méthode dans le cycle de vie.

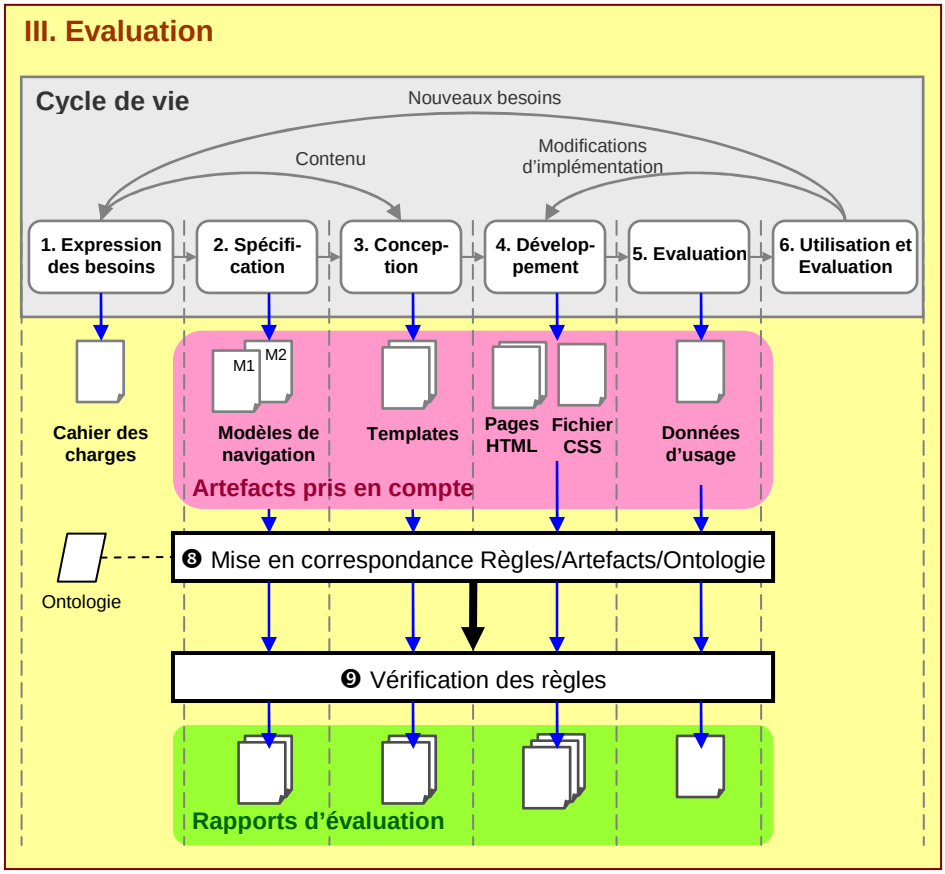


Figure 27 – Exemple d'inspection d'artefacts produits dans un cycle de vie en O

4.6.1.1. Cohérence des artefacts produits dans une même étape

Considérons la règle 1. « Chaque page doit contenir au moins un lien » vue dans la section 4.3.1. La détection d'une incohérence vis-à-vis de cette règle pour une même page P peut se faire sur des artefacts de même niveau tels que deux modèles de navigation M1 et M2 produits dans l'étape de spécification (voir Figure 27). Par exemple, M1 modélise P comme étant une page ayant au moins un lien, alors que M2 modélise P comme une page ne contenant pas de liens. L'évaluation de ces deux modèles identifiera une erreur pour le modèle M2 sur la page P. Cette erreur doit alerter l'évaluateur sur l'incohérence entre ces deux modèles, et ce dernier devra alors prendre les mesures pour identifier quel modèle, de M1 ou de M2, est incorrect et le corriger en conséquence.

Toutefois, la vérification de ce type de cohérence est limitée : la même page P modélisée à la fois dans M1 et M2 peut contenir un nombre de liens non nul mais différent. Dans ce cas, l'incohérence ne peut pas être identifiée par la seule inspection de ces deux modèles, et par conséquent par l'évaluateur. Ce problème de cohérence ne peut pas être détecté via notre méthode.

4.6.1.2. Cohérence des artefacts produits dans des étapes différentes

Certains artefacts peuvent être obtenus par réification d'autres artefacts produits dans une phase précédente du cycle de vie. Par exemple, dans la méthode Teresa [Paternò 03], les pages d'une application Web sont obtenues à partir d'interfaces utilisateur abstraites, elles-mêmes obtenues à partir d'un modèle de tâche système.

Si des problèmes sont détectés et corrigés sur des artefacts produits à une certaine étape du cycle de vie, une vérification de la cohérence peut être effectuée en vérifiant que ces problèmes n'apparaissent plus dans les étapes ultérieures. En effet, des erreurs identifiées dans une étape, alors qu'elles l'ont déjà été et corrigées dans les étapes précédentes, doivent mettre en garde l'évaluateur.

Toutefois, les activités menées dans chaque étape ont des objectifs différents et par conséquent, les artefacts de différentes étapes ne modélisent pas nécessairement un même élément du même point de vue. Par exemple, dans l'étape de conception, l'activité est focalisée sur la description de la structure d'une application Web (structure du contenu et navigation entre pages) et le contenu n'est pas renseigné, alors que dans l'étape de développement, l'objectif est de produire les pages Web finales avec le contenu renseigné. Ce contenu peut ainsi introduire des erreurs sans toutefois qu'il y ait incohérence entre les artefacts produits dans l'étape de conception et ceux réifiés dans l'étape de développement. Il est à la charge de l'évaluateur de faire la distinction entre ces erreurs et celles dues à un problème de cohérence entre artefacts produits dans différentes étapes.

4.6.1.3. Vérification de règles nécessitant l'inspection de plusieurs artefacts

Certaines règles nécessitent l'inspection de plusieurs artefacts afin de déterminer si une règle est respectée ou non. C'est le cas par exemple de la règle 6.1 des WCAG 1.0 « Pour tout contenu généré avec des feuilles de styles CSS, fournir un équivalent textuel dans le code source (HTML) ». Le contenu généré par une feuille de styles n'apparaissant pas dans le code source de la page HTML, il est impossible pour un non voyant d'avoir accès à cette information via l'utilisation d'un lecteur d'écran. Par conséquent, pour vérifier que cette règle est respectée, nous devons nous assurer qu'une alternative textuelle est donnée dans la page HTML lorsque du texte est généré depuis une feuille de styles CSS.

Si l'évaluation d'une seule page HTML est effectuée, il est impossible de savoir si du contenu a été généré avec une feuille de style CSS ou non. Réciproquement, si nous n'évaluons qu'une page CSS générant du contenu, il est impossible de savoir si la page HTML à laquelle la page CSS est associée contient une alternative textuelle. Cette règle nécessite par conséquent l'évaluation combinée d'une feuille de style CSS et d'une page Web HTML.

Ces règles, bien que peu nombreuses (nous n'avons identifié qu'une seule règle de ce type, la règle 6.1, dans les règles d'accessibilité du W3C/WAI), existent et sont à l'heure actuelle très difficiles à automatiser. Le problème des règles nécessitant l'inspection de plusieurs artefacts pour leur vérification, est très complexe. Il est nécessaire d'établir un pont entre ces artefacts. Dans notre exemple, il est nécessaire qu'un lien soit fait entre le contenu généré (CSS) et son alternative textuelle (HTML) pour que la règle puisse être vérifiée. Des travaux plus poussés doivent être menés dans cette direction pour que ce genre de règles puisse être pris en compte.

4.7. CONCLUSION

Nous avons présenté dans ce chapitre une méthode pour l'inspection ergonomique d'une application Web tout au long du cycle de vie de l'application. Cette méthode est constituée de trois phases : la *collecte et articulation des données pour l'évaluation* qui consiste à sélectionner et à configurer les règles ergonomiques et artefacts pour l'inspection ; le *paramétrage de l'évaluation* où l'évaluateur sélectionne les règles et artefacts à inspecter ; et l'*évaluation* qui consiste en l'inspection proprement dite des artefacts sélectionnés.

Cette méthode s'appuie sur l'ontologie développée et détaillée dans le chapitre 3 pour l'inspection des artefacts produits dans le cycle de vie. Cette ontologie établit un vocabulaire de concepts formalisés servant à la fois à décrire de manière formelle les règles ergonomiques (concepts à évaluer et logique d'évaluation de la règle), mais également à décrire l'ensemble des concepts manipulables par chaque artefact. Ces étapes dites de mises en correspondance règles/ontologie et artefacts/ontologie sont établies avant toutes inspections, dans la phase de collecte et articulation des données. Une fois ces mises en correspondances effectuées, l'inspection ergonomique consiste en l'évaluation des concepts mentionnés dans la règle ergonomique, en les récupérant directement sur l'artefact par une étape de mise en correspondance règle/artefact/ontologie dans la phase d'évaluation. L'ontologie forme ainsi le pont entre les règles ergonomiques et les artefacts à évaluer.

La mise en correspondance règles/ontologie nécessite une bonne connaissance des règles ergonomiques pour comprendre quels concepts de l'ontologie doivent être évalués et ainsi pouvoir décrire formellement chaque règle ergonomique. De même, la mise en correspondance artefacts/ontologie requiert une bonne connaissance technique des artefacts à inspecter, notamment, de la sémantique de chaque concept manipulable par ces derniers. Cette compétence technique limite ainsi la réalisation de la phase de mise en correspondance artefacts/ontologie à un expert informatique.

Les étapes de mise en correspondance règle/ontologie et artefacts/ontologie peuvent être longues et fastidieuses, mais elles ne sont réalisées qu'une fois par règle et par artefact, dans la phase de Collecte et articulation des données. Une fois l'artefact mis en correspondance avec l'ontologie, tout artefact de même nature peut être inspecté. Par exemple, une fois les concepts de la notation SWC mis en correspondance avec l'ontologie, tout modèle SWC produit pourra être inspecté.

A l'heure actuelle, la méthode que nous proposons ne gère ni les incohérences entre inspections de plusieurs artefacts dans le cycle de vie, ni l'inspection combinée d'artefacts. Nous avons vu dans les sections 4.6.1.1 et 4.6.1.2 que des incohérences pouvaient être détectées, par notre méthode, entre artefacts produits dans une même étape ou dans des étapes différentes. Si à l'heure actuelle, ces incohérences ne sont pas prises en compte explicitement par notre méthode, elles doivent cependant alerter l'évaluateur qui devra prendre des mesures pour corriger ces erreurs. L'inspection combinée d'artefacts, quant à elle, consiste en l'évaluation simultanée de plusieurs artefacts afin de vérifier qu'une règle est respectée, autrement dit, les informations nécessaires à la vérification de la règle se trouvent répartis dans chaque artefact. Notre méthode ne permettant d'évaluer qu'un seul artefact à la fois, ce type de règles ne peut pas être vérifié. La gestion des incohérences permettrait d'améliorer la couverture mais également le guidage des évaluations en pointant explicitement l'évaluateur sur les incohérences que notre méthode ne peut pas à l'heure actuelle détecter automatiquement. L'inspection combinée d'artefacts reste également une perspective toute aussi intéressante pour augmenter la couverture des règles automatiquement vérifiables.

5. Outil de support

Résumé

Dans les chapitres précédents, nous avons présenté une méthode supportant l'inspection des règles ergonomiques tout au long du processus de développement.

Nous avons vu d'une part que les règles ergonomiques devaient être structurées et interprétées avant de pouvoir être appliquées sur un artefact. D'autre part, nous avons identifié que ces règles manipulaient différents concepts liés aux éléments de l'interface d'une application Web. Ces concepts, unifiés dans une ontologie, permettent de décrire les règles ergonomiques à un niveau d'abstraction élevé (Chapitre 3).

L'utilisation de cette ontologie permet d'appliquer les règles ergonomiques sur les artefacts produits pendant le processus de conception en mettant en correspondance les concepts manipulés par chaque artefact avec les concepts de l'ontologie. De cette manière, les règles exprimées avec les concepts de l'ontologie sont immédiatement traduites sur un artefact donné, et par conséquent, évaluables sur cet artefact (Chapitre 4).

Nous présentons dans ce chapitre un outil supportant cette méthode. Cet outil est capable d'évaluer un artefact quelconque en entrée conformément à des règles ergonomiques et de corriger les erreurs identifiées. Pour pouvoir supporter plusieurs corpus de règles, les règles ergonomiques sont dissociées du moteur d'évaluation et sont contenues dans une base de règles. Un éditeur de règles permet de gérer (ajout, suppression, modification) ces règles ergonomiques.

L'organisation de ce chapitre est la suivante :

5.1 Introduction

5.2 Architecture

5.3 Intégration des différents modules

5.4 Discussion

5.5 Conclusion

5.1. INTRODUCTION

Dans le chapitre 4 précédent, nous avons présenté en détail une méthode pour l'inspection ergonomique tout au long du cycle de vie. Cette méthode s'appuie sur l'ontologie que nous avons décrite au chapitre 3 pour établir la correspondance entre les règles ergonomiques à vérifier et les artefacts à inspecter. Afin d'alléger le travail des concepteurs et des évaluateurs dans le cycle de vie de l'application, la méthode d'inspection proposée doit pouvoir être supportée par des outils informatiques capables de soutenir de manière automatique les différentes phases de notre méthode. Nous décrivons dans ce chapitre l'outil que nous avons développé pour supporter notre méthode et dont les besoins ont été identifiés comme suit :

- les règles ergonomiques à appliquer sur les artefacts doivent :
    - être gérables de manière flexible : un expert en ergonomie doit pouvoir écrire de nouvelles règles et les ajouter dans la base de règles, en supprimer, et les mettre à jour ;
    - être écrites de manière simple (même si toutefois, l'apprentissage d'un langage est nécessaire).
- l'inspection ergonomique doit :
    - supporter un grand ensemble de règles et d'artefacts éventuellement de taille importante (contenu riche) ;
    - prendre en compte la parution de nouveaux artefacts : autrement dit, elle ne doit pas dépendre d'un ensemble d'artefacts particuliers ;
    - être réalisable dans n'importe quelle étape du cycle de vie de l'application.
- un rapport d'évaluation doit être généré et doit :
    - être détaillé et expliquer de manière compréhensible les erreurs détectées ;

- être configurable pour s'adapter aux préférences de chaque évaluateur/concepteur (par exemple, organisation du rapport par type d'erreurs ou par objets de l'interface, choix des couleurs pour les différents type d'erreurs, ou encore, choix du format du rapport généré) ;
- pouvoir être sauvegardé, et si possible versionné, pour servir de base à de la documentation (par exemple, destiné à l'équipe chargée de la qualité du produit).

- la correction des erreurs identifiées lors de l'évaluation doit :
  - être automatique lorsque cela est possible afin d'alléger le travail de correction qui peut s'avérer pénible sur une application (ou artefact) de grande taille ;
  - être guidée, de manière semi-automatique ou manuelle, lorsqu'une correction automatique n'est pas possible.
- l'outil doit :
  - être intégrable à n'importe quel environnement de développement ;
  - être modulaire et chaque module doit pouvoir être remplacé selon les besoins et contraintes de chaque projet : par exemple, une équipe de développement doit pouvoir choisir son moteur d'évaluation pour des raisons de performance, de coûts, de compatibilité technologique, etc.
  - être relativement facile à prendre en main, sans trop de complexité, pour favoriser son acceptation dans le cadre d'un projet où plusieurs intervenants sont susceptibles de l'utiliser.

Outre ces besoins, il est nécessaire que l'outil puisse permettre d'effectuer cinq activités essentielles pour supporter pleinement notre méthode : l'*édition des règles* à inspecter sur les artefacts, activité supportant l'étape de mise en correspondance règles/ontologie (voir Figure 28 A) ; la *mise en correspondance des artefacts avec l'ontologie* pour évaluer un artefact donné, activité supportant l'étape de même nom (voir Figure 28 B) ; une *transformation* permettant de prendre un artefact en entrée et de le transformer en un modèle de l'application abstraite, activité supportant l'étape de mise en correspondance règles/artefacts/ontologie (voir Figure 28 C) ; une *évaluation* de ce modèle conformément aux règles ergonomiques, activité supportant l'étape d'évaluation (voir Figure 28 D) ; et enfin, une *correction* des erreurs identifiées sur ce modèle, activité classique d'un outil d'inspection et indépendante de notre méthode (voir Figure 28 E).

Ces activités devant être supportées par l'outil, ce dernier doit être utilisé dans tout le cycle de vie aux endroits où la méthode doit s'appliquer. Par exemple, l'édition des règles et la mise en correspondance artefacts/ontologie ont lieu dans la phase *I. Collecte et articulation des données* avant toute inspection dans le cycle de vie, conformément à notre méthode (voir section 4.2). Nous avons représenté, dans la Figure 28, l'ensemble des activités supportées par l'outil par rapport aux différentes phases de notre méthode.

Tout comme notre méthode, l'utilisation de cet outil fait intervenir plusieurs acteurs dont les compétences, responsabilités et tâches ont été détaillées dans la section 4.2.1. Toutefois, nous avons introduit ici un nouvel intervenant, le *développeur*, ou plus généralement, l'équipe de développement (voir en bas à droite de la Figure 28). Son rôle est de spécifier, concevoir et développer les différents artefacts nécessaires à la construction de l'application, il s'occupe par conséquent de l'édition des artefacts ; cependant, cette phase d'édition ne dépend pas de notre outil mais d'outils d'édition spécifiques aux artefacts et par conséquent, son activité n'a aucun impact sur notre outil. Nous avons choisi de représenter cet intervenant parce qu'il est la personne qui produit les artefacts à inspecter.

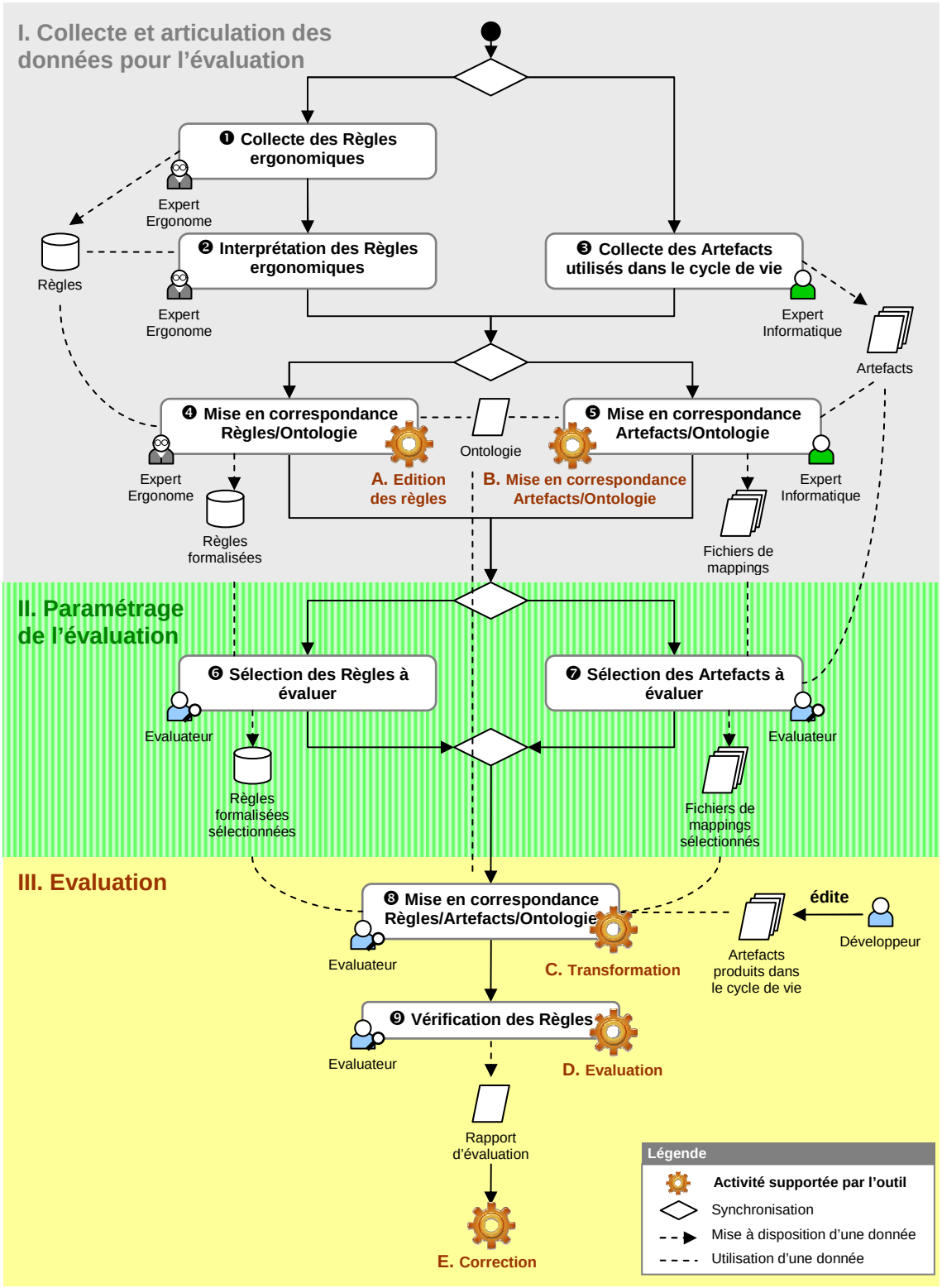


Figure 28 – Activités supportées par l'outil dans les différentes phases de notre méthode

Les différentes sections de ce chapitre décrivent en détails les modules qui ont été implémentés pour soutenir chacune de ces activités. Chacun de ces modules peut être utilisé de façon indépendante ou intégrée. Ces modules sont l'*Editeur de mappings*, l'*Editeur de règles*, le *Moteur de transformation* et l'*Extracteur de faits*, le *Moteur de règles* et le *Correcteur*. Ces différents modules s'appuient sur la plateforme Eclipse[27]. Le choix de cette plateforme de

[27] Le lecteur pourra trouver plus d'informations sur la plateforme Eclipse à l'adresse suivante : http://www.eclipse.org

développement était une contrainte imposée pour l'exploitation de ces outils dans l'atelier e-Citiz [eCitiz] développé par la société Genigraph où une partie de la thèse a été réalisé dans le cadre d'une convention CIFRE. Ce choix n'a pas d'incidence majeure puisque les différents modules implémentés ont été articulés de façon à ce qu'ils puissent être également disponibles et utilisables dans d'autres projets.

Ce chapitre est organisé de la manière suivante : nous présentons d'abord l'architecture détaillée de l'outil proposé et les modules de transformation, d'édition, d'évaluation et de correction (voir section 5.2) ; une section est ensuite consacrée à l'intégration des différents modules sur la plateforme e-Citiz (voir section 5.3) ; puis, nous discuterons des améliorations à apporter à notre outil (voir section 5.4) avant de conclure ce chapitre (voir section 5.5).

5.2. ARCHITECTURE

Parmi les besoins exprimés, nous avions identifié que les modules constituant l'outil devaient être indépendants pour pouvoir être remplacés selon les besoins et contraintes d'un projet mais aussi pour pouvoir être intégrés dans différents environnements de développement.

Pour répondre à ces besoins, nous proposons une architecture souple et robuste, capable de supporter toutes les activités identifiées dans la section précédente, et qui est constituée de modules indépendants communiquant entre eux par l'intermédiaire de données/modèles (voir Figure 29). Chaque règle est écrite par un expert ergonome à l'aide de l'*Editeur de règles* (présenté dans la section 5.2.3) qui donne la possibilité d'exprimer des règles ergonomiques à partir des concepts de l'ontologie. Chaque règle ainsi créée alimente la base de règles.

En parallèle, les artefacts susceptibles d'être inspectés doivent être mis en correspondance avec les concepts de l'ontologie. Cette activité est supportée par l'*Editeur de mappings* (présenté dans la section 5.2.2) qui permet de créer et d'éditer les mises en correspondances. Cet éditeur prend en entrée le méta modèle de l'artefact ainsi que l'ontologie et produit en sortie un fichier de mappings (voir section 4.3.3.1) par artefact.

Ces fichiers de mappings sont nécessaires pour la transformation des artefacts en un modèle de l'application abstraite (cette transformation est expliquée en détail dans la phase de mise en correspondance règles/artefacts/ontologie en section 4.5.1). La transformation d'un artefact nécessite en outre que ce dernier soit syntaxiquement correct, autrement dit, qu'il soit conforme à son méta modèle (un méta modèle définit la syntaxe d'un modèle, voir chapitre 4). Nous supposerons que les artefacts à évaluer sont syntaxiquement corrects puisque l'édition des artefacts (activité réalisée par l'équipe de développement, voir Figure 29 en haut à gauche) est réalisé à l'aide d'outils indépendants du notre. Une fois l'artefact, son méta modèle, l'ontologie et le fichier de mappings disponibles, la transformation peut s'effectuer. Cette transformation est réalisée par le *Moteur de transformation* (présenté dans la section 5.2.1) qui produit en sortie le modèle de l'application abstraite attendu.

Le modèle de l'application abstraite est ensuite évalué : les règles ergonomiques sélectionnées sont inspectées sur ce modèle. Cette activité doit être supportée par un module d'évaluation mais n'impose pas une architecture spécifique sur ce module. Toutefois, compte tenu des choix d'implémentation que nous avons faits, nous présentons ici une architecture particulière pour ce module, imposée par le moteur d'évaluation que nous avons utilisé. L'évaluation du modèle de l'application abstraite est ainsi réalisée en deux étapes. La première consiste à extraire les différents éléments (concepts de l'ontologie) à évaluer. Cette tâche est réalisée par l'*Extracteur de faits* (présenté dans la section 5.2.4) qui prend le modèle de l'application abstraite en entrée et insère ces éléments (appelés faits) dans la base de faits (la base de faits est par conséquent la liste des concepts qui ont été extraits). Une fois la base de faits constituée, ces faits sont évalués conformément aux règles contenues dans la base de règles (le couple base de faits/base de règles est appelée base de connaissances) par le *Moteur d'évaluation* (présenté dans la section 5.2.4) qui produit en résultat le rapport d'évaluation contenant les erreurs détectées lors de l'évaluation.

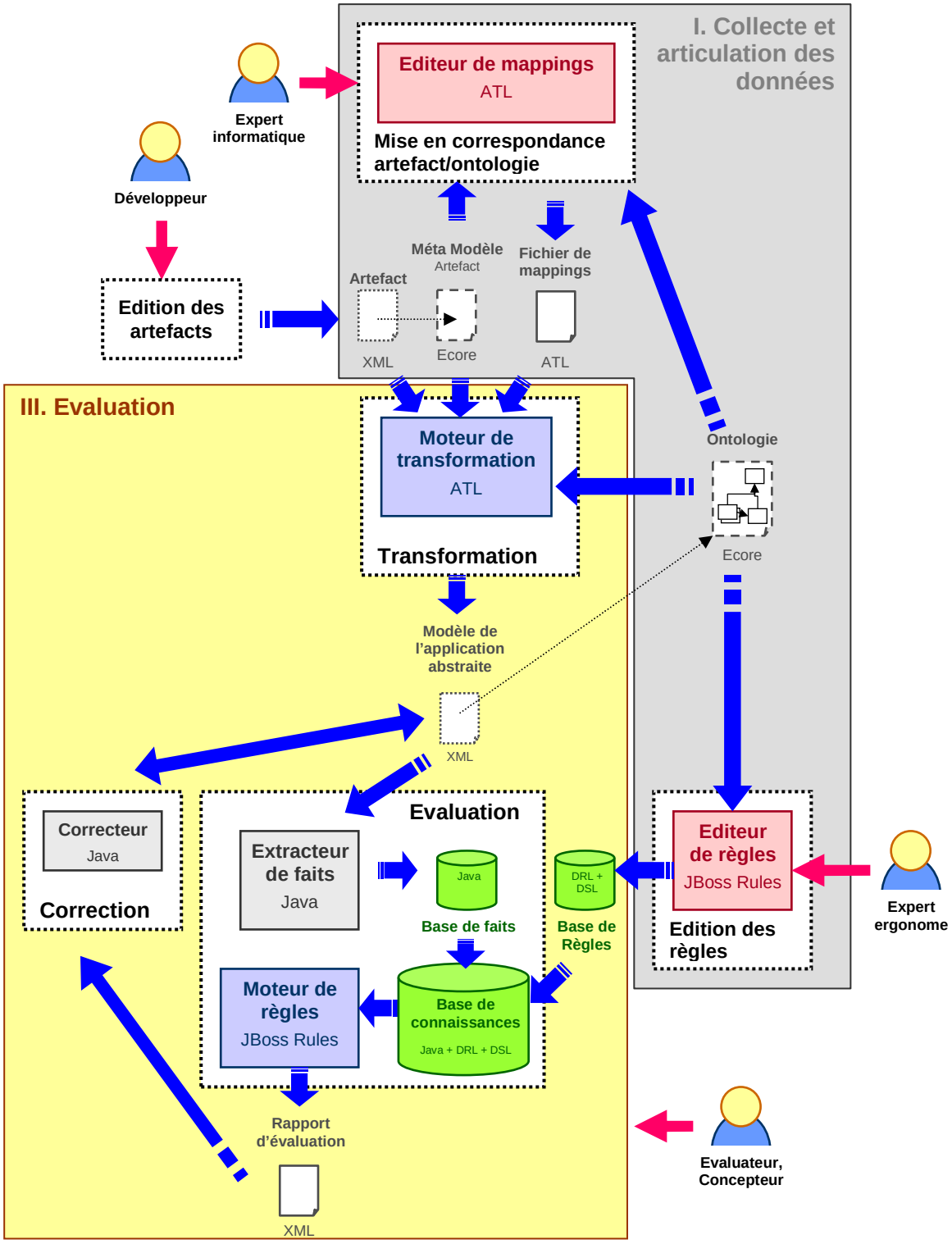


Figure 29 – Architecture détaillée de l'outil d'évaluation

Ce rapport d'évaluation sert ensuite de base à la correction des règles sur le modèle de l'application abstraite. Cette tâche est effectuée par le *Correcteur* (présenté dans la section 5.2.5) prenant en entrée le rapport d'évaluation et modifie en sortie le modèle de l'application abstraite.

5.2.1. Moteur de transformation

L'activité de transformation est une réponse au besoin de l'inspection ergonomique d'artefacts de différentes natures (modèles, code, etc.). Cette activité permet d'obtenir, à partir d'un artefact produit dans le cycle de vie, un modèle de l'application abstraite contenant sensiblement les mêmes informations, mais où les concepts représentés appartiennent à l'ontologie. Alors que l'artefact en entrée n'est pas directement évaluable, le modèle de l'application abstraite ainsi obtenu par transformation est quant à lui évaluable.

La transformation est réalisée par le *Moteur de transformation* (voir Figure 30) conformément à une approche dirigée par modèles [Schmidt 06], tel que détaillé dans la section 4.3.3.1, à partir de l'artefact, ou modèle source M_{source} (conforme à un méta modèle MM_{source}), pour obtenir le modèle de l'application abstraite, modèle cible M_{cible} (conforme à un méta modèle MM_{cible}, autrement dit, ici, l'ontologie).

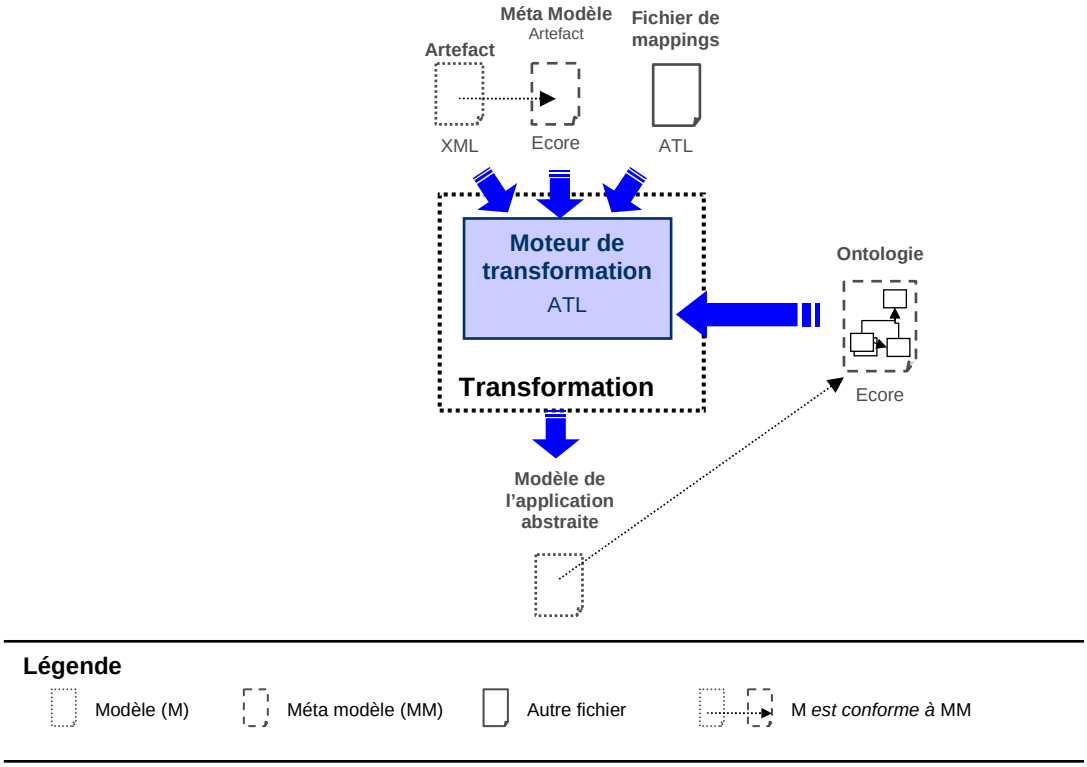


Figure 30 – Moteur de transformation

Les outils issus du domaine de l'IDM et dédiés à la transformation de modèles sont nombreux (voir section 4.3.3.1). Pour des raisons d'intégration à la plateforme Eclipse, nous avons choisi d'utiliser ATL (Atlas Transformation Language) [Jouault 05]. ATL désigne à la fois le langage et les outils de transformation de modèles, mais aussi le projet Eclipse du même nom. ATL a été développé par le groupe ATLAS[28] (INRIA & LINA).

Les métas modèles utilisés sont décrits en Ecore [Budinsky 04], le langage de méta modélisation d'Eclipse. Notre ontologie est notamment décrite avec ce langage et est perçue par le moteur de transformation comme étant un méta modèle (le méta modèle cible). Toutefois, si elle est vue comme un méta modèle à ce niveau, elle reste conceptuellement une ontologie

[28] http://www.sciences.univ-nantes.fr/lina/atl/

puisqu'elle est écrite en OWL et que le format Ecore de cette ontologie n'a été obtenue que par transformation (du format OWL vers le format Ecore)[29].

5.2.2. Editeur de mappings

La transformation d'un artefact en un modèle de l'application abstraite nécessite que les concepts manipulés par cet artefact soient mis en correspondance avec les concepts de l'ontologie. Ces mises en correspondance sont faites à l'aide de l'éditeur de mappings (voir Figure 31), par un expert informatique ayant les compétences requises nécessaires pour savoir quel(s) concept(s) de l'artefact correspond(ent) à un concept donné de l'ontologie.

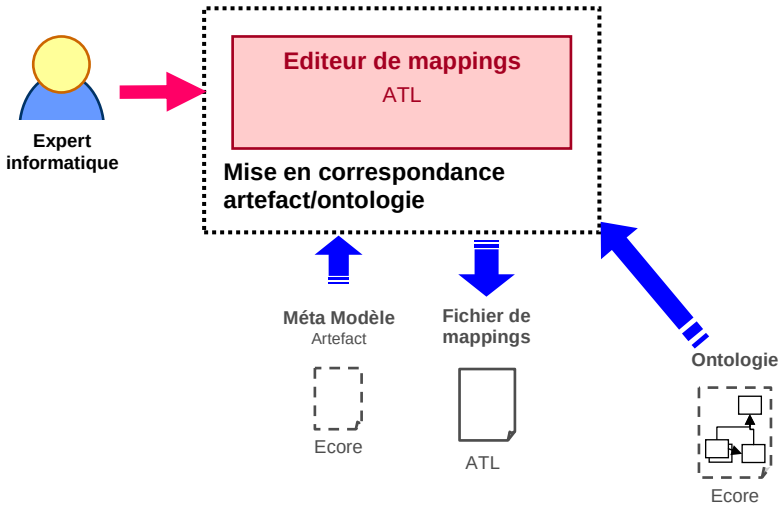


Figure 31 – Editeur de mappings

Plusieurs cas de mises en correspondance ont été illustrés dans la section 4.3.3 sans toutefois préciser comment de telles règles pouvaient être décrites. Dans le langage ATL, une mise en correspondance peut être vue comme une règle dans laquelle est spécifiée un *motif* (le concept de l'artefact à mettre en correspondance ainsi que des conditions sur celui-ci) et son *homologue* (le concept de l'ontologie correspondant).

Dans la partie motif de chaque mise en correspondance sont mentionnés les concepts de l'artefact que nous désirons transformer. Ces concepts sont fournis par le méta modèle de l'artefact (un artefact est défini formellement selon la sémantique donnée par le méta modèle). Ainsi, étant donné le méta modèle à notre disposition, nous pouvons, grâce à l'éditeur de mappings, spécifier une mise en correspondance pour chaque concept de l'artefact. En outre, pour pouvoir écrire des mises en correspondance conditionnelles, un concept, toujours dans la partie motif, peut être accompagné d'une ou de plusieurs conditions. Ceci est utile dans le cas où la mise en correspondance ne dépend pas uniquement du concept seul, mais aussi de la valeur de ses attributs. Par exemple, nous avons vu dans le chapitre 4 qu'un état dans la notation SWC correspondait à une page Web lorsque celui-ci était de type statique ou dynamique, alors qu'il ne correspondait à aucun concept lorsqu'il était de type externe ou transitoire. La mise en correspondance entre un état SWC et le concept Page est donc conditionnée par le type de l'état SWC.

Par ailleurs, l'éditeur de mappings permet de spécifier des mises en correspondance simples (un concept de l'artefact correspond exactement à un concept de l'ontologie), mais également complexes (par exemple, un concept de l'artefact correspond à une structure composée de plusieurs concepts (voir chapitre 4)).

[29] Le lecteur pourra trouver plus de détails sur cette transformation dans le projet EODM (EMF Ontology Definition Metamodel) : http://www.eclipse.org/modeling/mdt/?project=eodm#eodm

L'éditeur de mappings permet d'obtenir le fichier de mappings qui contient toutes les mises en correspondances spécifiées. Il se peut que lorsqu'un artefact manipule un grand nombre de concepts, le nombre de mises en correspondance soit important et que, par conséquent, certaines mises en correspondance entrent en conflit (c'est-à-dire qu'il existe au moins deux mises en correspondance spécifiant le même motif). Dans ce cas, l'éditeur signale ce conflit à l'expert informatique qui se chargera de corriger cette erreur.

L'écriture d'une mise en correspondance peut être complexe comme nous avons pu le voir dans le chapitre 4, et doit être réalisée, comme mentionné précédemment, par un expert informatique ayant des connaissances avancées et une bonne maîtrise des techniques de transformation de modèles. En outre, le fichier de mappings n'est établi qu'une seule fois, lors de l'introduction d'un nouvel artefact à évaluer, et il est rare que ce fichier soit modifié de manière fréquente. Par conséquent, nous n'utilisons qu'un éditeur de texte simple où les mises en correspondance sont entrées en langage informatique, langage supposé être maîtrisé par l'expert informatique.

Nous avons utilisé l'éditeur de règles de transformation fourni par ATL pour éditer les mises en correspondance dans lequel l'ontologie apparaît toujours comme étant le méta modèle cible. La Figure 32 montre un extrait du fichier de mappings où la mise en correspondance est effectuée entre les concepts de l'artefact SWC et ceux de l'ontologie. Ce fichier de mappings est présenté dans son intégralité en Annexe E.

```
1  module Swc2Ontology;
2  create OUT : WebApplication from IN : Swc;
3
4  ...
5
6  rule StaticState2Page {
7     from
8          state: Swc!StaticState (
9               not state.oclIsKindOf(Swc!CompositeState)
10              and state.type = #static
11         )
12    to
13         page: WebApplication!Page (
14              contents <- links,
15              url <- state.file
16         ),
17         links: distinct WebApplication!TextLink foreach (link in
18 state.getOutgoingLinks()) (
19              targetPath <- link.getTargetStateFile()
20         )
21 }
```

Figure 32 – Extrait du fichier de mappings SWC vers l'ontologie

Nous avons choisi, dans cet extrait, de ne présenter qu'une seule mise en correspondance pour illustrer la description de l'éditeur de mappings ATL où les mises en correspondance sont décrites en langage ATL.

Les méta modèles source (IN) et cible (OUT) sont spécifiés dans l'en-tête du fichier (voir Figure 32 ligne 2). Ainsi, « Swc » est une étiquette faisant référence au méta modèle de SWC et « WebApplication », une étiquette faisant référence à l'ontologie. Ces étiquettes servent d'espaces de nom pour identifier à quel méta modèle appartient chaque concept mentionné dans le fichier de mappings.

Une mise en correspondance est considérée comme une règle (ici, le nom donné à la règle est « StaticState2Page ») où est spécifié le motif (ici, un élément de type statique, voir Figure 32 ligne 8, suivi de plusieurs conditions entre parenthèses) ainsi que son homologue (ici, un élément de type Page appartenant à l'ontologie, voir Figure 32 ligne 13). Dans cette règle, nous pouvons voir une mise en correspondance simple : l'attribut file d'un état statique correspond à l'url d'une page). Mais nous avons également une mise en correspondance complexe : le contenu d'une page correspond à la liste des liens sortants de l'état statique (cette liste est

représentée ici par la variable links), liste dont la construction n'est pas triviale et fait appel à une fonction auxiliaire (*getTargetStateFile()*).

5.2.3. Editeur et outil de gestion des règles ergonomiques

L'ajout, la suppression et la modification des règles ergonomiques autorise une gestion flexible des règles qui vont être stockées dans une base de règles et ensuite exécutées par le moteur d'évaluation sur le modèle de l'application abstraite afin de détecter des problèmes d'ergonomie. Cependant, séparer ainsi les règles du moteur d'évaluation, impose que les règles soient décrites dans un langage informatique (compréhensible par le moteur d'évaluation). L'écriture de ces règles est supportée par un éditeur de règles ergonomiques qui va aider l'expert ergonome à définir ou mettre à jour les règles contenues dans la base de règles (voir Figure 33).

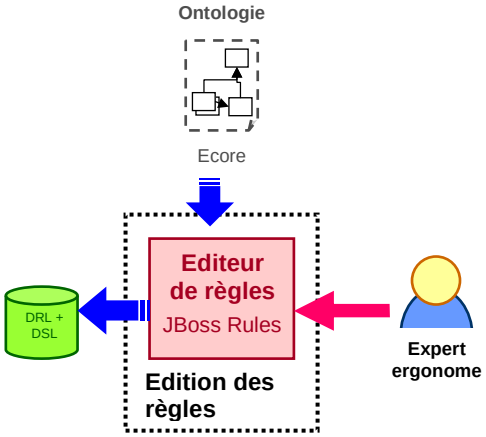


Figure 33 – Editeur de règles

L'éditeur de règles permet de spécifier en premier lieu la condition pour laquelle une règle est violée. Cette condition se fait sur un concept de l'ontologie ou une combinaison de ces concepts. Celle-ci peut être simple (tester la valeur d'un attribut ; opérations arithmétiques tels que « supérieur à », « égal à », « inférieur à », etc.) mais également complexe (opérateurs de la logique du premier ordre « Pour tout » et « Il existe » ; opérateurs logiques « NON, ET, et OU » ; opérations ensemblistes tels que « contient », « est inclus dans », etc.). En second lieu, l'éditeur permet de spécifier l'action à exécuter si la règle est violée. A l'heure actuelle, les seules actions possibles sont l'ajout d'une erreur ou l'ajout d'une alerte dans la liste des erreurs.

L'éditeur de règles est destiné à être utilisé par des experts ergonome n'ayant pas ou peu de compétences en informatique. Lorsque des conditions complexes doivent être écrites, la syntaxe des règles peut devenir illisible et par conséquent, difficile à comprendre. Pour faciliter l'écriture des règles ainsi que leur compréhension (un expert peut être amené à gérer et à comprendre des règles écrites par un autre expert), l'éditeur a été paramétré avec des conditions écrites, non plus en langage informatique, mais dans un langage proche du langage naturel. Par exemple, pour tout concept de l'ontologie, nous avons défini la condition « Il n'existe pas d'élément {*concept*} » où *concept* sera remplacé par un concept de l'ontologie lors de l'écriture de la règle. La condition écrite sous cette forme n'est rien d'autre qu'une étiquette qui sera traduite, lors de l'exécution de la règle, par son équivalent en langage informatique. Ecrire des règles en utilisant des phrases en langage naturel est plus simple à lire et à comprendre pour un expert ergonome. Ce mécanisme fonctionne également pour les actions.

Enfin, pour aider l'expert, l'éditeur guide ce dernier dans l'écriture d'une règle. Lorsque l'expert sélectionne un élément à évaluer dans la condition, l'éditeur propose (sous la forme d'une liste déroulante) la liste des conditions pouvant s'appliquer à cet élément.

Pour implémenter l'éditeur de règles, nous avons utilisé l'éditeur de règles JBoss Rules (voir section 5.2.4) que nous avons paramétré pour prendre en compte les concepts de notre ontologie (voir Figure 34).

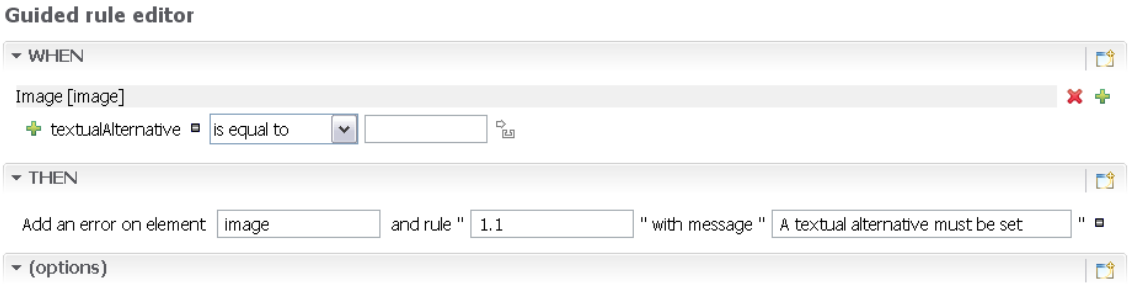


Figure 34 – Editeur de règles guidé

La règle présentée en Figure 34 spécifie une condition (dans la partie WHEN) sur l'attribut *textualAlternative* de tout élément de type Image (Image étant un concept de notre ontologie). Cette condition peut se lire de la manière suivante : « Si un élément de type Image possède une alternative textuelle égale à "" ». L'action exécutée lorsque cette condition est vérifiée est l'ajout d'une erreur dans la liste des erreurs. Pour cela, l'utilisation d'une instruction en langage naturel a été utilisée pour faciliter la lecture de la règle : « Add an error on element … ». Cette action prend trois paramètres : l'élément sur lequel l'erreur a été identifiée, le numéro de la règle violée, ainsi que le texte descriptif de l'erreur.

Chaque règle ainsi créée est ajoutée dans la base de règles. Cette base est constituée d'un simple fichier (avec l'extension .drl) contenant toutes les règles écrites en langage informatique. Nous avons un extrait de la base de règles en Figure 35 où est présentée la règle précédemment créée. Nous pouvons voir que la traduction de l'étiquette « Add an error on element… » a été réalisée et que cette règle est stockée en langage informatique dans la base de règles.

```
...

rule "1.1"
   dialect "mvel"
   when
         image : Image( textualAlternative == "" )
   then
   errors.add(new RuleError(RuleError.ERROR, image, "1.1", "A textual alternative must
be set"));
end
...
```

Figure 35 – Extrait de la base de règles

En outre, nous utilisons l'outil de gestion des règles fourni par JBoss Rules[30] qui permet de supporter l'ajout, la modification, et la suppression des règles dans la base de règles, mais également la sélection des règles à exécuter.

5.2.4. Extracteur de faits et Moteur d'évaluation

Comme précisé dans l'introduction de cette section, les modules supportant l'activité d'évaluation sont au nombre de deux : l'extracteur de faits et le moteur d'évaluation. Cette raison s'explique par le choix du moteur JBoss Rules qui impose le fonctionnement de ces deux modules indépendants qui communiquent entre eux.

JBoss Rules est un système expert à base de règles de production permettant de représenter et de traiter la connaissance, c'est-à-dire un ensemble de faits et de règles sur ces faits, relative à un domaine particulier (dans notre cas, cette connaissance, l'ontologie, est

[30] Le lecteur pourra trouver plus d'informations sur l'outil de gestion des règles fourni par JBoss Rules à l'adresse suivante : http://downloads.jboss.com/drools/docs/4.0.5.19064.GA/html_single/index.html#d0e4955

relative au domaine de l'ergonomie du Web). Le moteur de règles (appelé aussi moteur d'inférence) est capable d'exécuter un grand nombre de règles et de supporter un ensemble important de faits.

L'évaluation du modèle de l'application abstraite est réalisée par le moteur d'évaluation sur les faits (c'est-à-dire les concepts extraits de ce modèle) contenues dans la base de faits par rapport aux règles ergonomiques issues de la base de règles. Ces deux bases constituent la base de connaissances (voir Figure 36).

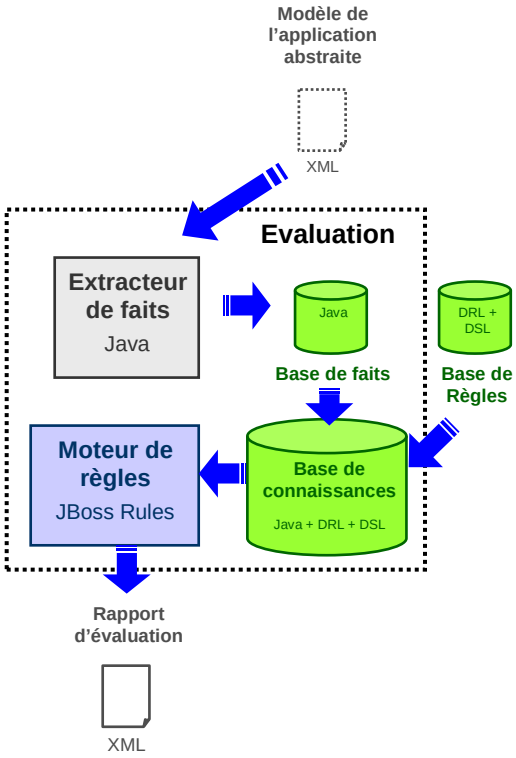


Figure 36 – Extracteur de faits et Moteur d'évaluation

La base de faits est alimentée avant toute évaluation par l'extracteur de faits, sans quoi, aucune règle ne sera déclenchée. L'implémentation de cet extracteur consiste en un parcours du modèle de l'application abstraite afin de collecter au fur et à mesure ces différentes instances dans la liste des faits. Nous avons utilisé pour cela le patron de conception *Visiteur* [Gamma 95] connu pour être efficace dans la lecture et le traitement d'une structure arborescente (le modèle de l'application abstraite est un fichier XML et peut être par conséquent représenté par un arbre).

Enfin, suite à l'exécution des règles, le rapport d'évaluation est produit en sortie. Actuellement, les outils étant intégrés dans la plateforme Eclipse, ce rapport d'évaluation est généré directement dans une vue Eclipse (une vue correspond à une zone de l'environnement graphique de développement). Il se présente sous la forme d'une liste d'erreurs exprimées en langage naturel, par exemple « Une alternative textuelle doit être renseignée pour les images. Ligne 54. » Le contenu et la structure du rapport d'évaluation seront abordés plus en détail dans la section 5.3.3.

Le format EARL [EARL] a été proposé par le W3C pour l'uniformisation des rapports d'évaluation de l'accessibilité des applications Web. Ce format est encore à l'étude et sa structure est instable : de nombreuses disparités apparaissent ainsi dans les outils d'inspection existants qui l'utilisent pour la rédaction de leurs rapports. Les rapports que nous produisons actuellement pourraient être convertis dans ce format standard, de manière à :

- évaluer la qualité des rapports produits par notre outil par rapport aux autres outils d'inspection ;
- une fois notre outil positionné par rapport aux autres, coupler nos rapports avec ceux produits par un outil complémentaire au nôtre.

5.2.5. Correcteur

Le Correcteur doit permettre la correction des erreurs identifiées lors de l'évaluation. Pour réaliser cette tâche, sont utilisés en entrée : le rapport d'évaluation contenant les informations sur les erreurs détectées, et le modèle de l'application abstraite à réparer. Pour chaque correction que l'outil est en mesure d'effectuer, le modèle de l'application est corrigé en conséquence (voir Figure 37).

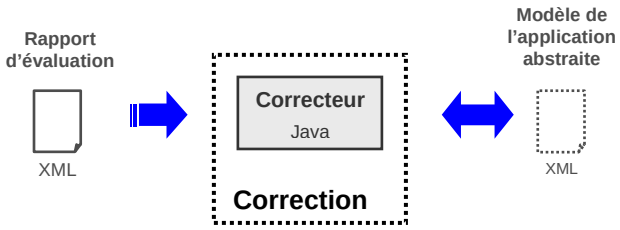


Figure 37 – Correcteur

Actuellement, ce module n'a pas été implémenté. Cependant, nous présentons ici les fonctionnalités devant être supportées par ce module au sein de l'architecture proposée.

Toutes les corrections sont faites sur le modèle de l'application abstraite puisque les erreurs détectées sont relatives à ce modèle. Lorsqu'une correction peut être faite de manière automatique par le Correcteur, ce dernier identifie l'élément sur lequel l'erreur a été détectée et répare automatiquement la faute afin d'être conforme à la règle ergonomique non respectée. Lorsqu'une correction automatique n'est pas possible, une correction semi-automatique ou manuelle (accès à des ressources pour une évaluation manuelle) doit être proposée par le Correcteur.

Le Correcteur doit pouvoir s'intégrer dans l'environnement de développement utilisé par l'évaluateur. Il doit être capable lorsque cela est possible de pointer de manière efficace l'endroit (ligne de code, élément XML, etc.) où l'erreur a été détectée. Enfin, il doit pouvoir supporter l'annulation des corrections dans le cas où l'évaluateur le désire.

Le report des erreurs sur l'artefact source correspondant au modèle de l'application abstraite doit être étudié. Notre outil guidera alors véritablement la conception d'interfaces ergonomiques. Si l'artefact est décrit sous forme de fichier textuel, le rapport d'inspection peut désigner la ligne et la colonne pointant sur l'erreur dans l'artefact. A partir de ces informations, l'élément à corriger dans l'artefact peut être identifié. La correction de cet élément pour le rendre valide consiste généralement à remplacer une valeur par une autre. Par exemple : changement d'une balise dépréciée <b> par sa remplaçante (règle WCAG 1.0 n°11.2), ajout de l'attribut alt sur une image (règle WCAG 1.0 n°1.1).

5.3. INTEGRATION DES DIFFERENTS MODULES DANS UN ENVIRONNEMENT INDUSTRIEL

L'outil, tel que nous l'avons présenté dans la section précédente, peut fonctionner de manière autonome. Cependant, selon les besoins d'un projet, il arrive que l'outil doive être intégré à un environnement de développement dont une partie des modules nécessaires à son fonctionnement existent déjà. Pour répondre à ce besoin, nous avons proposé, dans la section précédente, une architecture où chaque module était indépendant. Autrement dit, tous les modules qui composent l'outil peuvent fonctionner seuls dès l'instant où les différentes entrées

attendues (modèles, ontologie, etc.) sont fournis à ces modules, et chacun de ces modules peut être remplacé selon les besoins et contraintes du projet. Cette souplesse dans l'architecture facilite l'intégration de l'outil dans un environnement de développement.

Nous présentons dans cette section l'intégration de notre outil dans une plateforme de développement existante pour la conception d'e-services, plateforme utilisé dans un contexte industriel.

e-Citiz [eCitiz] est une plateforme permettant de concevoir des e-services (démarches administratives électroniques, également appelées « télé-services ») suivant une approche dirigée par modèles. Nos travaux de thèse ont été appliqués sur cette plateforme dont le besoin était de s'assurer de l'accessibilité des e-services développés depuis cette plateforme afin de se conformer à la loi française sur l'accessibilité [Loi Accessibilité France 05]. Nous avons contribué à cette plateforme en développant un outil d'évaluation de l'accessibilité respectant l'architecture présentée dans la section 5.2, afin d'accompagner cette évaluation tout au long de la conception des e-services. L'évaluation de l'accessibilité dans e-Citiz est faite conformément aux règles d'accessibilité WCAG 1.0 de niveau A et AA et est réalisée dans les différentes étapes du processus de conception.

Cet outil a été intégré directement au Studio e-Citiz. Toutefois, certains modules apparaissant dans cette architecture n'ont pas été développés pour les raisons suivantes :

- L'éditeur de mappings et le Moteur de transformation : l'ensemble des modèles étant de petite taille (au nombre de cinq), clairement défini dès le début du projet e-Citiz et étant peu susceptible d'augmenter, il a été décidé que la mise en correspondance soit faite manuellement afin d'apporter une solution plus légère à l'utilisateur du Studio (inutile d'embarquer un moteur de transformation alors que l'ensemble des modèles est fixe et connu) ; cette décision sera revue si le nombre de modèles vient à augmenter ;
- L'éditeur de règles : actuellement, les règles sont détachées du moteur et contenues dans une base de règles. Cependant, puisque les règles actuellement écrites sont les WCAG 1.0 (niveau A et AA) et qu'elles ne sont pas amenées à évoluer dans l'immédiat (ces règles datent de 1999 et aucune révision n'a été faite jusqu'à présent, une version 2.0 est en cours de réflexion depuis quelques années mais aucun consensus n'a encore été trouvé), l'éditeur de règles n'a pas été implémenté ; à terme, cet éditeur sera implémenté.

Nous présentons dans la section 5.3.1 la plateforme e-Citiz, puis dans la section 5.3.2, le processus de conception adopté pour le développement d'e-services, ainsi que les artefacts produits dans ce processus, et enfin, la section 5.3.3 est consacrée au module d'évaluation de l'accessibilité intégré à e-Citiz.

5.3.1. La plateforme e-Citiz

L'avènement des nouvelles technologies a changé la manière de traiter l'information au sein de l'administration : les procédures et démarches qui autrefois étaient traitées manuellement sont aujourd'hui automatisées entièrement ou partiellement. L'administration actuelle se tourne vers une administration électronique où les démarches administratives sont réalisées sur support électronique et à distance. De telles démarches sont appelées « e-services » et sont réalisées à travers une téléprocédure (ou procédure administrative effectuée à distance).

e-Citiz est une plateforme de spécification, de génération et de gestion d'e-services. Cette plateforme se compose d'un environnement de développement appelé *Studio e-Citiz* permettant la spécification d'e-services via l'utilisation intensive de modèles (approche dirigée par modèles) et d'un serveur appelé *Moteur e-Citiz* pour l'exécution des e-services développés.

Le *Studio e-Citiz* est un environnement de développement basé sur la plateforme Eclipse permettant, à l'aide d'éditeurs graphiques, de créer et d'éditer les différents modèles destinés à la spécification d'un e-service. Il permet notamment de :

- définir le workflow associé à chaque e-service ;
- définir l'interface utilisateur abstraite pour chaque étape du workflow ;
- définir l'ensemble des acteurs interagissant avec les différents e-services à mettre en œuvre ;
- spécifier, pour chaque acteur, l'ensemble des e-services auxquels il a le droit d'accéder ;
- définir les organisations d'appartenance des acteurs quand il s'agit d'agent de traitement back office (c'est-à-dire côté administration) ;
- définir les utilisateurs autorisés à se connecter aux services et les rôles qui leurs sont attribués ;
- définir l'intégralité du modèle de données comme, par exemple, le contenu des dossiers personnels des utilisateurs des e-services ;
- gérer les notifications ou alertes entre acteurs et/ou organisations.

Le *Moteur e-Citiz* est un serveur logiciel dont la responsabilité est d'exécuter les e-services déployés. Côté front office (utilisation de l'e-service), il gère et répond à toutes actions déclenchées par l'utilisateur sur l'e-service (exécution du workflow, requêtes sur la base de données, exécution des règles métier, etc.). Côté back office (administration des e-services), le moteur facilite la gestion des e-services déployées. Il donne ainsi la possibilité à l'organisation détentrice de l'e-service d'accéder et de tenir à jour la base de données, de recueillir les différentes démarches utilisateur pour les traiter ensuite et d'avoir une vue d'ensemble de l'état des e-services (par exemple, statistiques sur le nombre de démarches envoyées).

Le développement d'un e-service est ainsi réalisé en utilisant conjointement le Studio et le Moteur. Il implique la participation de plusieurs acteurs ayant des rôles et des compétences différentes. Les experts métier ou fonctionnel utilisent le Studio e-Citiz pour spécifier les besoins fonctionnels du e-service, en collaboration avec les experts technique (développeurs) qui spécifient les besoins techniques. Au final, l'e-service obtenu est déployé, exécuté grâce au Moteur e-Citiz et est évalué par l'équipe qualité.

5.3.2. Processus de conception

Le processus de conception proposé par e-Citiz est un processus itératif faisant intervenir plusieurs modèles pour le développement d'applications de type e-service. Ce processus (voir la partie gauche de la Figure 38 intitulée « e-Citiz ») permet de concevoir et réaliser des e-services en partant des besoins, puis en élaborant un prototype jusqu'à l'obtention des e-services mis en production. Les différents prototypes, ainsi que l'e-service final, sont validés fonctionnellement par l'expert métier à chaque itération.

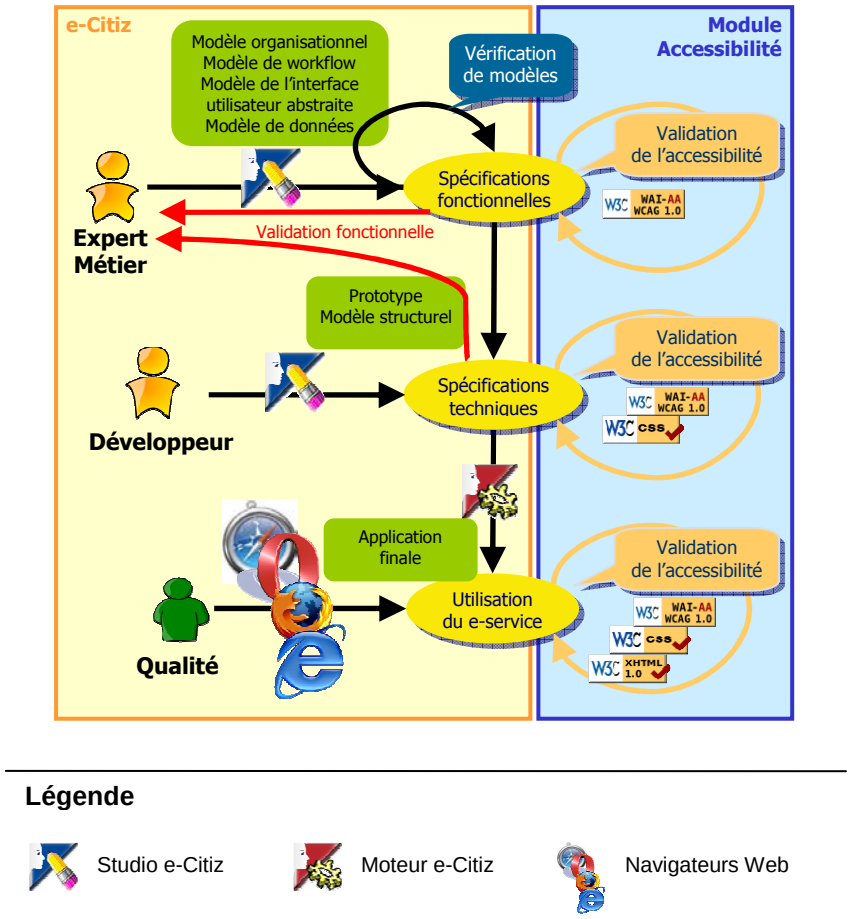


Figure 38 – Processus de conception e-Citiz

Dans la phase de *Spécifications fonctionnelles*, les besoins fonctionnels sont spécifiés par l'expert métier via l'utilisation de plusieurs modèles : modèle organisationnel, modèles de workflow, modèles de l'interface utilisateur abstraite, et modèles de donnée. Le modèle organisationnel permet de modéliser les organisations intervenant sur l'e-service spécifié ; les acteurs pouvant gérer une organisation mais également effectuer des démarches administratives, et les utilisateurs pouvant se voir attribuer plusieurs rôles (autrement dit, avoir le rôle de plusieurs acteurs). Les modèles de workflow permettent de modéliser l'enchaînement des différentes étapes d'un processus (ou démarche administrative). Les modèles de l'interface utilisateur abstraite permettent de spécifier, à un haut niveau d'abstraction, les formulaires (le domaine étant celui des e-services, le formulaire est le concept central) à partir duquel l'utilisateur du e-service pourra entrer des informations. Chaque formulaire contient des objets d'interactions concrets (au sens de [Vanderdonckt 93]). Le modèle de données est un modèle structurant l'ensemble des données qui vont être collectées par l'utilisateur pendant l'exécution de l'e-service. Une fois les spécifications fonctionnelles validées par l'expert métier, les spécifications techniques sont détaillées dans l'étape suivante.

Dans la phase de *Spécifications techniques*, des détails sur l'implémentation sont fournis par l'équipe de développement afin de générer automatiquement un prototype exécutable. Pour cela, un modèle structurel est établi et permet de spécifier la structure (ou interface utilisateur) des pages finales à travers la disposition des éléments de l'interface utilisateur dans les pages finales, ainsi que la charte graphique. Le prototype obtenu par génération est un e-service exécutable mais où les fonctionnalités ne sont pas encore toutes supportées. Ce prototype est ensuite validé fonctionnellement. S'il n'est pas conforme aux besoins ou si de nouveaux besoins sont exprimés, les spécifications fonctionnelles peuvent ou doivent être redéfinies, auquel cas, une nouvelle itération dans le processus de conception a lieu. Si le prototype est validé

fonctionnellement, l'application finale est générée et personnalisée en fonction des besoins du client (contraintes de sécurité, contraintes technologiques, etc.).

Dans la phase d'*Utilisation du e-service*, l'application finale est déployée et testée par l'équipe qualité. A ce stade, les besoins fonctionnels sont validés et l'évaluation ne porte que sur l'utilisabilité en situation d'exécution avec de vrais utilisateurs. Si l'évaluation est concluante, l'e-service est déployé et peut alors être utilisé par les utilisateurs finaux.

5.3.3. Evaluation de l'accessibilité dans e-Citiz

L'évaluation de l'accessibilité est réalisée sur les artefacts produits dans les trois étapes du processus de conception (voir la partie droite de la Figure 38 intitulée « Module Accessibilité »). Dans l'étape de Spécifications fonctionnelles, les modèles de workflow, les modèles de l'interface utilisateur abstraite ainsi que les modèles de données sont évalués (le modèle organisationnel n'est pas impacté par les règles d'accessibilité qui ne traitent que de l'interface utilisateur finale). Dans l'étape de Spécifications techniques, les différents prototypes sont vérifiés, ainsi que le modèle structurel de l'application. Ce dernier intègre des informations sur la disposition des éléments de l'interface utilisateur via une ou plusieurs feuilles de styles CSS. L'accessibilité est par conséquent réalisée également sur les feuilles de styles. Enfin, dans l'étape d'Utilisation du e-service, l'application finale est évaluée à partir du code XHTML.

L'évaluation des prototypes ainsi que l'application finale est supporté par un outil tiers, ATRC Checker [ATRC], destiné à évaluer l'accessibilité sur le code HTML. Cet outil est intégré directement au Moteur e-Citiz et permet, lors de l'exécution de l'application, de vérifier l'accessibilité de chacune des pages. L'intégration de cet outil a été motivé d'une part, pour sa qualité d'analyse du code source et d'autre part, parce qu'il apparaît comme un outil supplémentaire pour la validation de la qualité des e-services déployés. Nous ne discuterons pas de cet outil puisqu'il s'agit d'un outil indépendant pour l'évaluation du code HTML de l'application finale mais ne supportant pas notre méthode. L'évaluation des modèles est quant à elle supportée par l'outil d'évaluation de l'accessibilité que nous avons développé et intégré à la plateforme e-Citiz. Ces deux outils sont utilisés conjointement, mais à différentes étapes : notre outil est utilisé dans les étapes de spécification (Spécifications fonctionnelles et Spécifications techniques) et ATRC Checker est utilisé quant à lui dans l'étape d'Utilisation du e-service sur les prototypes générés et sur l'application finale (avant d'être déployé).

Avant de pouvoir réaliser des évaluations d'accessibilité, un paramétrage doit être effectué. Il s'agit d'une part, de sélectionner les règles à évaluer, et d'autre part, de sélectionner les artefacts que nous voulons évaluer : modèles (l'évaluation des modèles est appelée « audit des spécifications » et est supportée par notre outil), prototypes et application finale (l'évaluation de ces deux types d'artefacts est appelée « audit à l'exécution » et est supportée par ATRC Checker).

Comme mentionné précédemment (voir l'introduction de cette section), l'évaluation de l'accessibilité porte sur les règles d'accessibilité WCAG 1.0 niveau A et AA. Si à l'heure actuelle, il n'est pas possible de modifier ces règles (l'éditeur de règles n'ayant pas été implémenté), il est cependant possible de sélectionner l'ensemble des règles à évaluer : A, AA ou aucunes (dans ce dernier cas, aucune évaluation n'est lancée). La sélection des règles (voir Figure 39) est faite à partir des propriétés d'un e-service (ce dernier est stocké dans un fichier XML ayant pour nom « eservice.ego »).

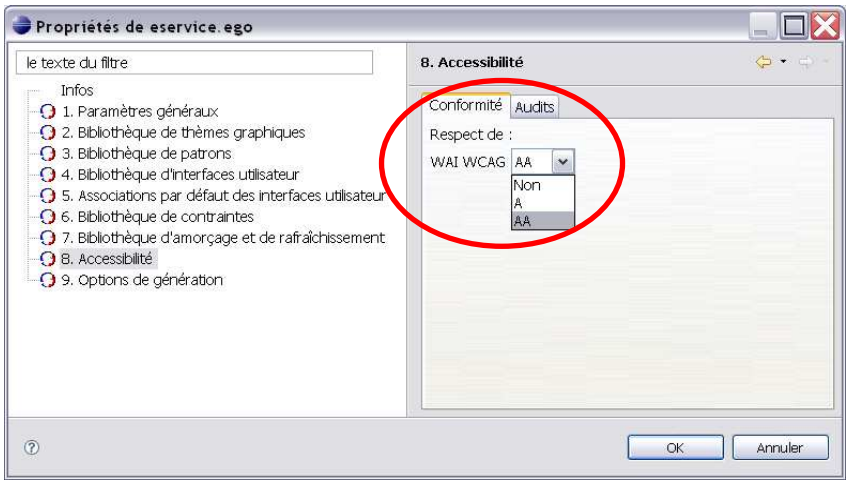


Figure 39 – Sélection des règles d'accessibilité à évaluer dans le Studio e-Citiz

La sélection du type d'évaluation que nous voulons effectuer est réalisée également à partir des propriétés du e-service. Dans la Figure 40, l'audit des spécifications (évaluation des modèles), ainsi que l'audit à l'exécution (évaluation des prototypes et de l'application finale) ont été sélectionnés, activant ainsi notre outil, ainsi que ATRC Checker.

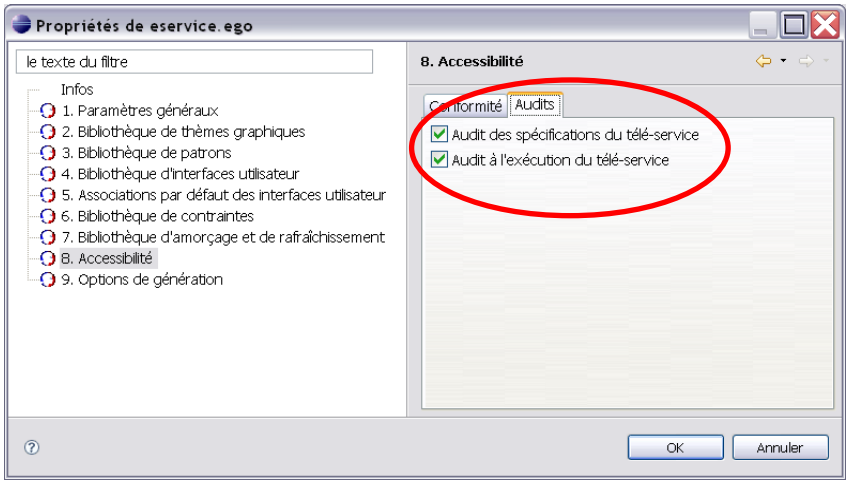


Figure 40 – Activation des modules d'évaluation de l'accessibilité dans e-Citiz

L'évaluation des spécifications est faite sur les modèles dans le Studio. L'évaluation de l'accessibilité des spécifications est lancée chaque fois que celles-ci sont modifiées et sauvegardées. Dans la Figure 41, nous pouvons voir une capture d'écran du Studio e-Citiz après une évaluation automatique des spécifications.

Le rapport d'évaluation (voir l'encadré en bas à droite de la Figure 41) affiche la liste des erreurs qui sont au nombre de 19 dans cet exemple, et apparaît dans la vue « Erreurs ». Actuellement, le rapport d'évaluation n'est affiché que dans cette vue et n'est pas produit physiquement (aucun fichier n'est généré après une évaluation) par le moteur d'évaluation. Double-cliquer sur une erreur dans ce rapport d'évaluation ouvre l'éditeur du modèle associé. Dans notre exemple, nous avons double-cliqué sur la première erreur portant sur l'interface utilisateur de l'objet d'interaction concret « Demande acceptée » (voir la flèche dans la Figure 41). L'éditeur associé à l'étape « Traitement d'une inscription » (une étape définit un modèle de l'interface utilisateur abstraite) s'ouvre et l'erreur peut être corrigée.

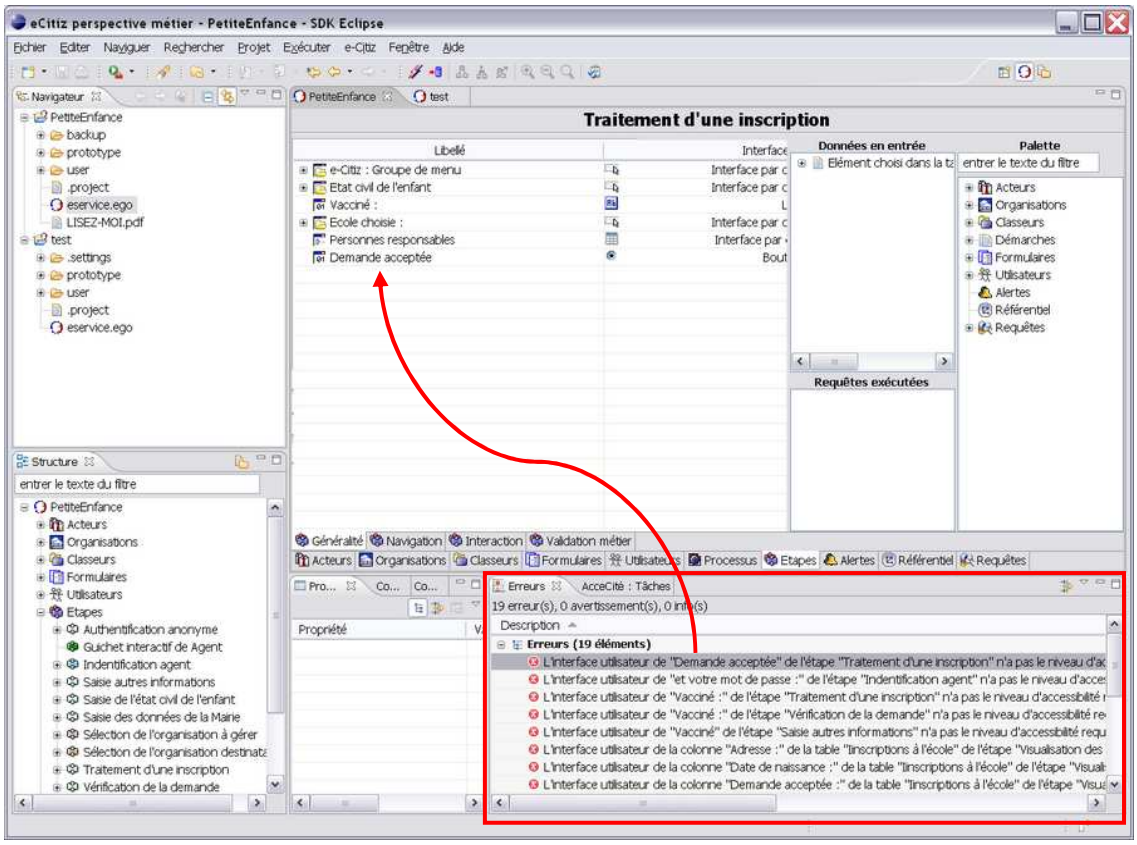


Figure 41 – Evaluation de l'accessibilité sur les spécifications d'un e-service dans e-Citiz

5.4. DISCUSSION

Les besoins identifiés afin de supporter la méthode proposée dans nos travaux de thèse ont permis l'élaboration d'un outil dont l'architecture modulaire a été présentée dans la section 5.2. Un premier besoin concerne la gestion efficace des règles. Actuellement, cette gestion est supportée grâce à un éditeur de règles permettant à un expert ergonome d'alimenter une base de règles, de mettre à jour et de supprimer les règles. Cependant, l'éditeur de règles doit être configuré en amont pour pouvoir écrire des règles par rapport aux concepts de l'ontologie et pour pouvoir guider l'expert ergonome dans cette écriture. Cette configuration passe par l'apprentissage du langage d'écriture des règles (dans notre solution, le langage utilisé est JBoss Rules) pour écrire les conditions applicables sur chaque concept.

D'autres besoins ont été identifiés concernant l'évaluation ergonomique : le support d'un grand ensemble de règles et d'artefacts de taille importante, la prise en compte de nouveaux artefacts, ainsi que le support de l'évaluation dans toutes les étapes du cycle de vie de l'application. La prise en compte d'un artefact (nouveau ou existant) est possible grâce à la présence du fichier de mappings. L'écriture de ce fichier peut être longue et fastidieuse, comme nous avons pu l'expérimenter, mais elle n'est faite qu'une fois, lorsqu'un nouvel artefact doit être supporté par l'outil d'évaluation. L'édition de ce fichier de mappings est supportée par un éditeur qui permet, pour un expert informatique, de faire les mises en correspondance nécessaires pour la transformation. Par ailleurs, le moteur d'évaluation utilisé (JBoss Rules) est un moteur de règles puissant capable de supporter un grand nombre de règles et des artefacts de grande taille (contenu large). L'évaluation sur une application de grande taille, telle que celles développées dans le cadre de projets industriels avec e-Citiz, a prouvé qu'il n'y avait pas de pertes de performance. Enfin, la combinaison du moteur de transformation et du moteur d'évaluation permet d'évaluer les différents artefacts produits dans les différentes étapes du cycle de vie.

Concernant le rapport d'évaluation, les besoins exprimés concernaient le détail et l'explication pertinente des erreurs détectées lors de l'évaluation, mais également la possibilité de configurer, de sauvegarder et de versionner celui-ci. Le rapport d'évaluation est actuellement intégré dans une vue de la plateforme Eclipse, plateforme sur laquelle l'outil développé est intégré. Il est par conséquent impossible de configurer, sauvegarder et versionner ces rapports. A terme, nous prévoyons de pouvoir convertir ce rapport au format EARL [EARL], format qui permettrait d'une part, d'évaluer la qualité des rapports produits par notre outil par rapport aux autres outils d'inspection et d'autre part, de coupler nos rapports avec ceux produits par un outil complémentaire au nôtre.

Les besoins sur la correction des erreurs étaient de pouvoir proposer une correction automatique dans le cas où cela est possible, et de proposer une correction semi-automatique ou manuelle autrement. La correction des erreurs n'est, pour l'instant, pas supportée dans notre outil. Actuellement, puisque les erreurs sont relatives au modèle de l'application abstraite, la correction des erreurs est prévue pour être effectuée sur ce modèle, puis, une fois ce modèle corrigé, ces corrections sont ensuite reportées sur l'artefact source. Ce module doit à terme pouvoir guider efficacement la conception d'interfaces ergonomiques en assistant de manière interactive l'évaluateur dans les phases de correction.

Enfin, les besoins sur l'outil de manière générale, étaient qu'il soit modulaire pour pouvoir être facilement intégré à tout environnement de développement, et en outre, être facile à prendre en main. En réponse à ces besoins, nous avons proposé une architecture constituée de plusieurs modules indépendants les uns des autres. Nous avons vu que cette indépendance autorisait le choix des différents modules selon les besoins et contraintes d'un projet, mais permettait également d'intégrer l'outil dans n'importe quel environnement de développement. En outre, l'utilisation de l'outil s'avère simple : dans la plateforme e-Citiz, l'évaluation, une fois configurée, se fait automatiquement à chaque fois que l'e-service spécifié est modifié et sauvegardé, sans intervention de l'utilisateur. Si l'utilisation de l'outil par un évaluateur reste simple, un effort initial doit toutefois être entrepris par l'expert informatique pour éditer les fichiers de mappings mais également pour configurer l'éditeur de règles.

5.5. CONCLUSION

Nous avons présenté dans ce chapitre un outil capable de supporter l'inspection ergonomique de manière automatique dans les différentes étapes du processus de conception selon la méthode proposée dans nos travaux de thèse. L'architecture de cet outil est composée de plusieurs modules indépendants : le moteur de transformation, l'éditeur de mappings, l'éditeur de l'outil de gestion des connaissances ergonomiques, le moteur d'évaluation et l'extracteur de faits, ainsi que le correcteur. Ces modules, puisque indépendants les uns des autres, peuvent être changés et adaptés selon les besoins et contraintes d'un projet. Notamment, l'intégration de notre outil est facilitée dans un environnement de développement utilisant déjà une partie des modules nécessaires au fonctionnement de l'outil.

Notre outil a été intégré dans la plateforme e-Citiz de développement d'e-services. Certains modules, comme le moteur d'évaluation, étaient déjà présents sur cette plateforme ; d'autres, ont été implémentés manuellement, comme le moteur de transformation (la transformation est faite manuellement dans le code). L'intégration de notre outil sur cette plateforme a montré qu'il pouvait être utilisé sur une plateforme employée à l'échelle industrielle.

6. Etude de cas

Résumé

Ce chapitre présente une étude de cas simple sur la conception du site Web de la mairie fictive de Bergen.

A travers les artefacts produits dans les différentes étapes du cycle de vie de l'application, nous mettons en évidence l'ensemble des éléments conceptuels représentés sur chacun des artefacts. Ces éléments conceptuels sont la base pour une inspection automatique des règles d'accessibilité WCAG 1.0 et de navigation EvalWeb.

Nous présentons et discutons tout au long de ce chapitre les résultats obtenus sur ces deux corpus de règles suite à l'inspection ergonomique.

Ce chapitre est organisé de la manière suivante :

6.1 Description informelle de l'étude de cas

6.2 Etape de spécification : Modèles de navigation

6.3 Etape de conception : Modèles de l'interface abstraite

6.4 Etape d'implémentation : Pages Web

6.5 Etape d'évaluation : Application Web finale

6.6 Résultats détaillés de l'inspection des règles de navigation et d'accessibilité

6.7 Conclusion

Nous présentons dans ce chapitre une étude de cas sur la conception d'un site Web. Cette étude de cas illustre les fondements de notre méthode présentée au chapitre 4. Les objectifs de cette étude de cas sont les suivants :

- Montrer que les artefacts produits dans le cycle de vie manipulent les mêmes concepts que ceux mentionnés dans les règles ergonomiques ;
- Donner un aperçu des règles vérifiables dans les différentes étapes du cycle de vie ;
- Montrer que certaines règles ne peuvent être vérifiées qu'à certaines étapes du cycle de vie.

Dans un souci de présenter une étude de cas proche de la réalité, notre choix s'est porté sur la conception et le développement du site Web d'une mairie. L'avantage d'une telle application est que non seulement nous trouvons des pages informationnelles, c'est-à-dire dont le contenu est uniquement dédié à apporter de l'information, mais également, des pages interactives où l'utilisateur saisit des informations, où des traitements côté serveur sont effectués et où des interactions avec une base de données peuvent avoir lieu. C'est le cas, par exemple, des services électroniques qu'une mairie peut offrir à ses citoyens via le site Web de la commune.

Pour des raisons de propriété intellectuelle, la mairie présentée est fictive. Toutefois, les éléments présentés tout au long de ce chapitre sont comparables à ceux de vrais projets tels que ceux développés dans un cadre industriel avec e-Citiz. Nous nous intéressons dans ce site Web à la démarche d'inscription d'un enfant à l'école, et plus particulièrement à une étape précise de cette démarche : la saisie de l'état civil de l'enfant. Au fil des différentes phases du processus de conception, l'artefact associé à l'étape est présenté. L'étude de chacun de ces artefacts est présentée par rapport aux règles d'accessibilité WCAG 1.0 et aux règles de navigation EvalWeb.

La section 6.1 présente le site Web de la mairie, les exigences attendues, ainsi que le processus de conception adopté. Les artefacts produits dans ces étapes sont décrits dans les sections 6.2 à 6.5. La section 6.6 donne un récapitulatif de l'évaluation de ces artefacts à chaque étape. La conclusion de ce chapitre est donnée dans la section 6.7.

6.1. DESCRIPTION INFORMELLE DE L'ETUDE DE CAS

La mairie de Bergen désire avoir son portail sur le Web afin de regrouper et présenter les informations utiles sur la ville (actualité, scolarité, transports en commun, urgences, associations, culture, tourisme, etc.), mais également pour moderniser son administration en offrant des services électroniques à l'ensemble de ses citoyens (demande d'acte de naissance, inscription d'un enfant à l'école, etc.).

Afin de permettre à tout citoyen d'accéder à ce portail Web, la mairie exige un respect des standards d'accessibilité ainsi qu'une navigation bien étudiée pour éviter que l'utilisateur ne se perde dans le flot d'informations. Pour cela, les règles choisies sont les WCAG 1.0 pour l'accessibilité et les règles EvalWeb pour la navigation, décrites dans la section 3.1.1.

Le processus de conception adopté pour le développement de cette application Web est celui de Scapin et al. [Scapin 00a] (voir Figure 42). Les artefacts produits dans les différentes étapes de ce processus et sur lesquels nous nous basons pour cette étude de cas sont les suivants (rectangles gris aux coins arrondis dans la Figure 42) : des *modèles de navigation* sont développés dans l'étape de spécification afin de formaliser la navigation et la structure de l'application Web ; les *interfaces utilisateur abstraites* de chaque page d'un modèle de navigation sont ensuite modélisées dans l'étape de conception ; les *pages Web* finales où est renseigné le contenu, autant informatif que décoratif, sont développées dans l'étape d'implémentation ; enfin, les *données d'usage* sont collectées dans l'étape d'utilisation et d'évaluation.

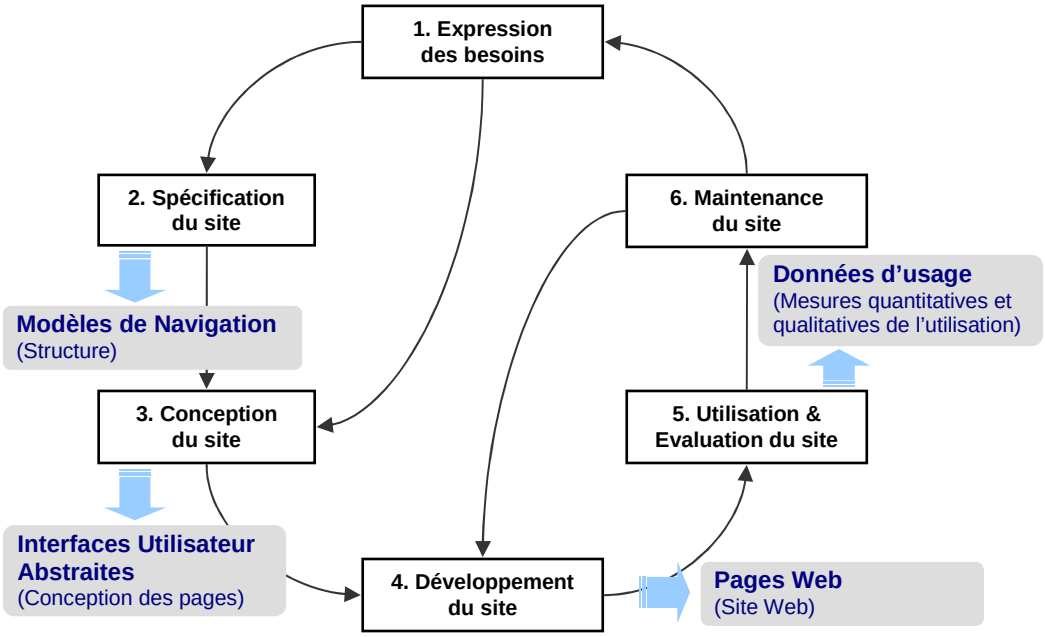


Figure 42 – Artefacts sélectionnés dans le cycle de vie pour l'étude de cas

6.2. ETAPE DE SPECIFICATION : MODELES DE NAVIGATION

Pour modéliser la navigation de cette application, nous utilisons la notation StateWebCharts (SWC) [Winckler 03]. Le site Web est d'abord modélisé dans son ensemble avec les principales rubriques de l'application (voir Figure 43). Ce site Web est composé de deux régions concurrentes (séparées par un trait pointillé dans la Figure 43), dont la première constitue le menu du site (état intitulé « menu » dans la région gauche) et la seconde, la page Web courante (l'état courant lors de l'accès à l'application est l'état initial intitulé « Actualités » symbolisé par le pseudo-état initial qui lui est rattaché). Ces deux régions sont affichées en permanence pour l'utilisateur : le menu reste ainsi toujours disponible pour l'utilisateur quelle que soit la rubrique visitée. Les états ayant une bordure supérieure dans la Figure 43 (par

exemple, l'état intitulé « Démarches administratives ») dénotent les états composites mais dont les états fils ne sont pas représentés pour des raisons de lisibilité.

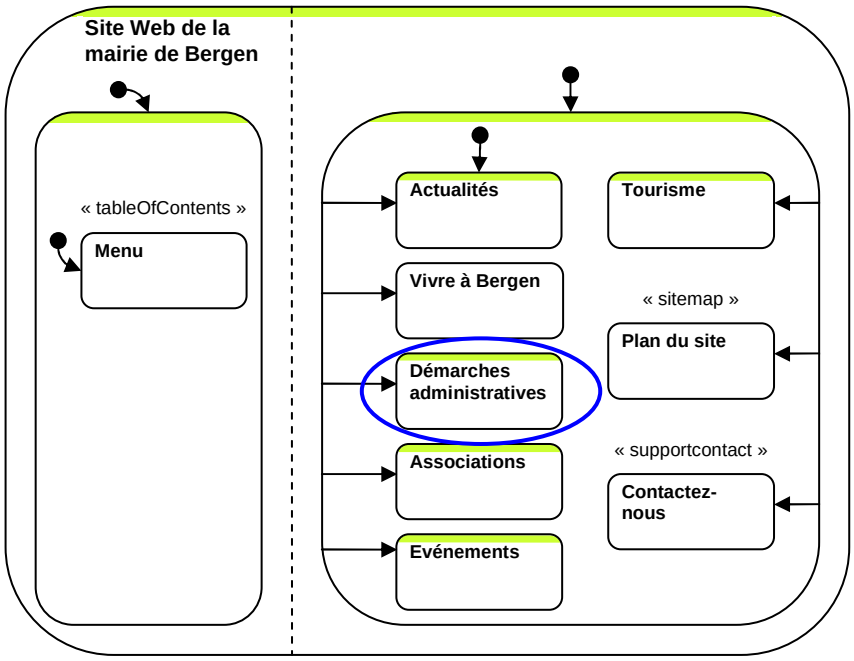


Figure 43 – Modèle de navigation SWC du site Web de la mairie de Bergen

Chaque rubrique est ensuite modélisée. Comme mentionné dans l'introduction, nous nous intéressons dans cette étude de cas à la modélisation de la démarche administrative d'« Inscription d'un enfant à l'école » accessible depuis la rubrique « Démarches administratives » (entourée dans la Figure 43). Cette rubrique est un état composite dont le détail est donné dans la Figure 44. Cinq démarches administratives sont offertes à l'utilisateur depuis l'index. Nous détaillons dans la suite l'inscription d'un enfant à l'école (état entouré dans la Figure 44).

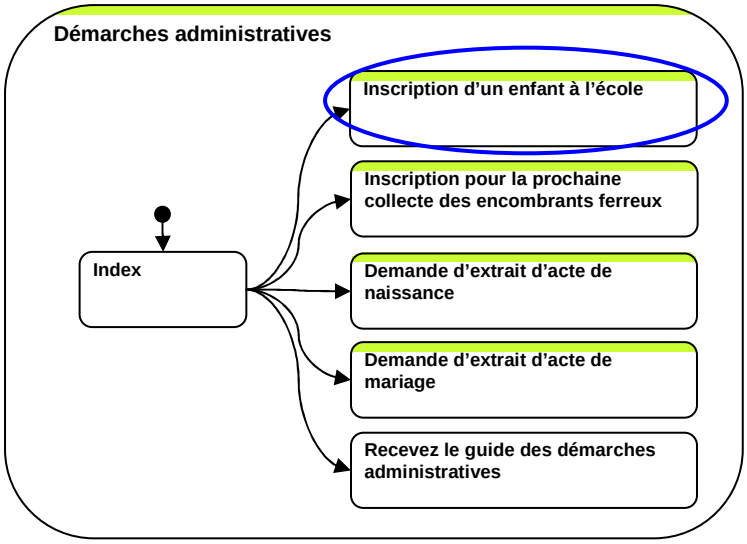


Figure 44 – Modèle de navigation de la page « Démarches administratives »

La démarche d'inscription d'un enfant à l'école (voir Figure 45) est composée d'une première étape d'identification (état « Identification »). Une fois le login et le mot de passe saisis par l'utilisateur, une phase de vérification (état transitoire « Vérification ») a lieu. Si ces données ne sont pas valides, une page d'erreur est affichée à l'utilisateur (état « Identification impossible ») et l'utilisateur se voit automatiquement redirigé vers la page d'identification. Si

l'authentification réussit, l'utilisateur entre alors dans la procédure d'inscription. Il est alors invité à saisir l'état civil de l'enfant à inscrire (état « Saisie Etat civil »). Une fois saisi, l'utilisateur valide en cliquant sur le lien « Suivant ». Cette action enclenche un processus de vérification des données entrées. Si des données ne sont pas valides (données obligatoires non saisies, date de naissance non valide, format des données non respectées, etc.), la page de saisie de l'état civil est réaffichée avec les erreurs identifiées à corriger. Lorsqu'il n'y a plus d'erreurs, l'utilisateur peut passer à la page suivante pour saisir des informations supplémentaires telles que les « personnes responsables de l'enfant » (état « Saisie Autres informations »). De même que pour l'étape précédente, les données saisies sont vérifiées et des erreurs sont affichées si ces données ne sont pas correctes. Enfin, la dernière étape de cette démarche propose un récapitulatif des données saisies avant la validation finale. Notons qu'à chaque étape, un bouton précédent permet de revenir en arrière dans la procédure (à l'exception de l'étape de saisie de l'état civil).

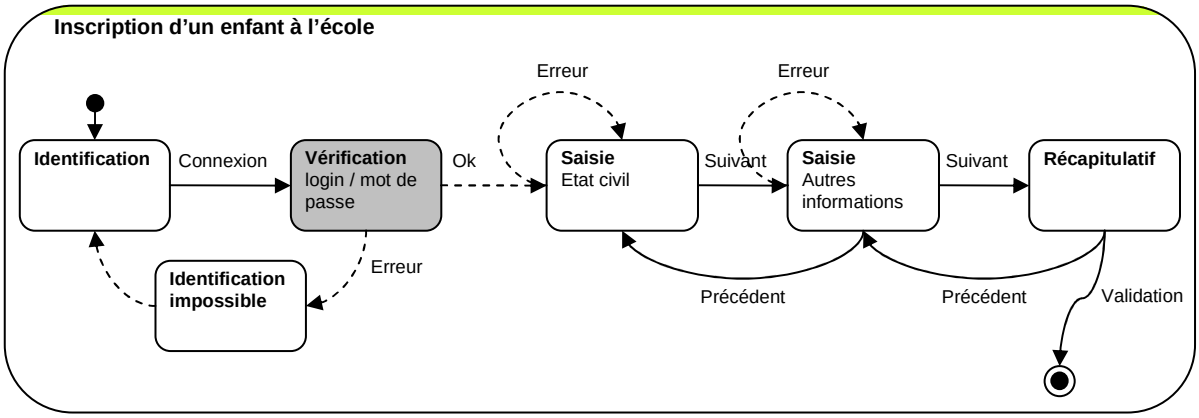


Figure 45 – Modèle de navigation SWC de la démarche d'inscription d'un enfant à l'école

A travers les modèles de navigation, nous définissons, comme nous venons de le voir, les différentes pages de l'application Web ainsi que les liens de navigation entre ces pages. Le contenu des pages n'est cependant pas modélisé.

Une transition utilisateur (flèches pleines) permet de modéliser un lien hypertexte, et son étiquette associée, le libellé du lien. Par exemple, dans la Figure 45, la transition utilisateur avec pour étiquette « Connexion » sera représentée par un lien ayant la même étiquette (à ce stade de la modélisation, nous ne nous préoccupons pas de savoir si ce lien sera au final un lien hypertexte classique ou, par exemple, un bouton). La pertinence du libellé des liens (règle WCAG 1.0 n°13.1 ; règles EvalWeb n°2, n°5, n°10, n°14 et n°15) peut donc être vérifiée sur ces modèles. Dans le meilleur des cas, ces libellés peuvent être vérifiés de manière semi-automatique : toutes les transitions utilisateurs seraient passées en revue par un évaluateur humain à l'aide d'un outil qui amènerait l'évaluateur de transition en transition.

Outre les liens, les états enrichis sémantiquement peuvent adresser certaines règles d'accessibilité et de navigation. C'est le cas, par exemple, des trois états ayant respectivement pour stéréotype « tableOfContents », « sitemap » et « supportcontact » en Figure 43. Les deux premiers états représentent respectivement une table des matières et le plan du site. Leur présence au sein de l'application valide automatiquement les règles WCAG 1.0 n°13.3 et n°13.4, ainsi que les règles EvalWeb n°39 et n°41. Le troisième état représente une page contenant des informations pour contacter le support, par exemple, pour qu'un utilisateur puisse indiquer un problème d'accessibilité à l'utilisation. La présence de cette page valide en partie la règle WCAG 1.0 n°11.4.

Enfin, le contenu des pages n'étant pas modélisé à cette phase de la conception, les règles de navigation et d'accessibilité portant sur le contenu ne peuvent pas être vérifiées.

6.3. ETAPE DE CONCEPTION : MODELES DE L'INTERFACE ABSTRAITE

Une interface utilisateur abstraite est produite pour chaque page de l'application Web. Pour cette étude de cas, nous avons modélisé l'interface utilisateur abstraite (IUA) de l'état « Saisie Etat civil » vue précédemment en Figure 45. L'IUA a été modélisée en UsiXML [Limbourg 04]. Un aperçu de cette interface (présenté en Figure 46) a été obtenu à partir du code UsiXML de l'IUA dont un extrait est donné en Figure 47. Dans le haut de l'interface présentée en Figure 46 (zone 1), nous trouvons l'icône de la ville de Bergen, ainsi que le titre de la page, à savoir « Mairie de Bergen ». Sur la gauche (zone 2), nous avons le menu permettant d'accéder aux différentes rubriques de la page. Dans la région centrale de l'interface (zone 3), nous avons le formulaire de saisie de l'état civil constitué de trois informations à saisir (le nom, le prénom et la date de naissance) ainsi que du bouton « suivant » permettant d'accéder à la page suivante. Le coin inférieur gauche de l'interface (zone 4) permet d'afficher le copyright de l'application. Enfin, dans la région du bas (zone 5), des informations sur l'adresse de contact de la mairie, ainsi qu'un texte législatif relatif à la procédure administrative, sont affichés.

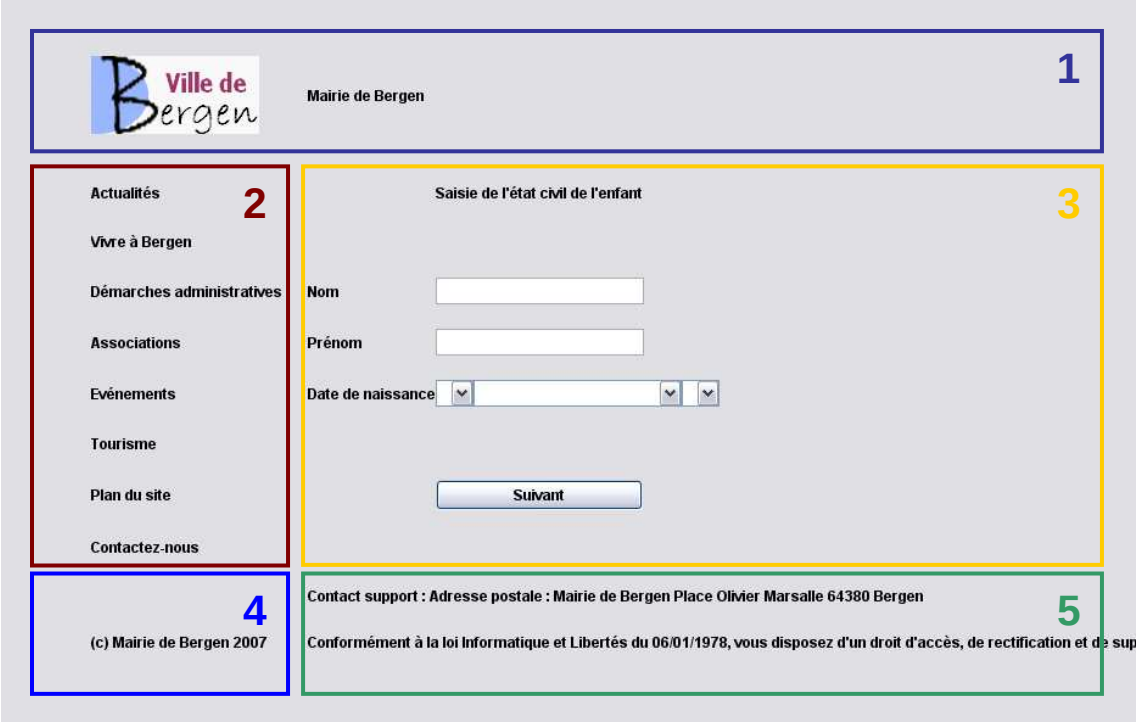


Figure 46 – Interface abstraite UsiXML de l'étape "Saisie de l'état civil de l'enfant" pour la démarche d'inscription d'un enfant à l'école

```
<constraint gridx="17" gridy="12" gridwidth="4"
   gridheight="1" weightx="1.0" weighty="1.0"
   fill="both" insets="0,0,0,0">
   <outputText id="output_text_component_47"
        name="output_text_component_47"
        content="/uiModel/resourceModel/cioRef[@cioId='output_text_component_47']/
                 resource/@content"
        defaultContent="Nom" isVisible="true"
        isEnabled="true" fgColor="#ffffff" isBold="true"
        textColor="#000000" textHorizontalAlign="right"/>
</constraint>
...
<constraint gridx="22" gridy="12" gridwidth="6"
   gridheight="1" weightx="1.0" weighty="1.0"
   fill="both" insets="0,0,0,0">
   <inputText id="input_text_component_40"
        name="input_text_component_40"
        isMandatory="true" isVisible="true"
        isEnabled="true" textColor="#000000"
        maxLength="50" numberOfColumns="15" isEditable="true"/>
</constraint>
```

Etiquette « Nom »

Champ de saisie pour l'étiquette « Nom »

Figure 47 – Extrait du code UsiXML pour le champ de saisie « Nom »

UsiXML est un langage de description d'interface utilisateur contenant des informations exploitables pour la vérification des règles d'accessibilité. Certaines règles de navigation et d'accessibilité sur les liens, les images, les couleurs, et les champs de formulaire peuvent être vérifiées de manière automatique comme nous allons le voir dans la suite. D'autres règles peuvent faire l'objet d'une évaluation manuelle.

Le langage UsiXML permet d'associer une URL à une étiquette (composant label) ou à une image, ces dernières jouant ainsi le rôle d'un lien textuel et d'un lien image (ou image cliquable). Les URLs peuvent être exploitées afin de vérifier qu'elles sont valides, autrement dit, que les pages vers lesquelles mènent ces URLs existent (règles EvalWeb n°1, n°2, n°3 et n°4). En outre, le libellé d'un lien textuel, qui n'est autre que le texte de l'étiquette, peut également être vérifié (règles EvalWeb n°2, n°5, n°10, n°14 et n°15). Nous pouvons noter que le bouton « Suivant » (voir Figure 46) est une réification du lien étiqueté « Suivant » sortant de l'état « Saisie Etat civil » dans le modèle de navigation présenté en Figure 45. Ainsi, sur l'interface abstraite, les règles qui vont être vérifiées sur ce composant sont celles relatives aux contrôles de formulaire de type bouton, alors que dans le modèle de navigation précédent, ce sont les règles relatives aux liens hypertextes qui étaient vérifiées sur le lien sortant « Suivant ».

Concernant les images, une alternative textuelle peut être renseignée. Toutefois, dans notre étude de cas, le logo de la mairie de Bergen n'en contient pas volontairement. La règle d'accessibilité WCAG 1.0 n°1.1 n'est donc pas respectée et une erreur est détectée.

Pour chaque composant de l'interface, la couleur de fond mais également celle du texte peuvent être spécifiées. Elles correspondent, dans l'étiquette « Nom » de la Figure 47, aux attributs respectifs « fgColor » et « textColor ». Il est ainsi possible, conformément à la règle WCAG 1.0 n°2.2, de vérifier que la différence de contraste entre ces deux couleurs est assez grande[31].

L'interface abstraite modélisée contient également un formulaire composé de champs et de leurs étiquettes associées. Toutefois, l'association étiquette/champ ne peut pas être faite directement dans le code UsiXML et ne peut se faire que visuellement par l'évaluateur. La règle WCAG 1.0 n° 12.4 n'est donc vérifiable que manuellement. Chaque champ de saisie textuel peut en outre être rempli avec un texte par défaut, par exemple : « Entrez votre nom ici ». Dans notre étude de cas, aucun texte par défaut n'est donné pour les champs du formulaire (l'absence de ce texte par défaut est marqué par l'absence de l'attribut « defaultContent » dans les champs de saisie, comme nous pouvons le voir pour l'étiquette « Nom » en Figure 47). La présence d'un

[31] Le W3C/WAI donne un algorithme de référence pour décider si la différence de contraste est assez grande à l'adresse suivante : http://www.w3.org/TR/AERT#color-contrast

texte par défaut dans un champ de formulaire est recommandée par la règle WCAG 1.0 n° 10.4. L'absence de ce texte dans les champs de formulaire signifie que cette règle n'est pas respectée.

Certaines règles ne peuvent être vérifiées que manuellement, telles que, par exemple, la position visuelle des étiquettes par rapport à leur champs (WCAG 1.0 n° 10.2) ou la clarté du langage écrit (WCAG 1.0 n°14.1). Parmi ces règles vérifiables manuellement, certaines ne peuvent pas être vérifiées automatiquement par manque de sémantique. C'est le cas par exemple des mécanismes de navigation. L'interface abstraite modélisée contient, par exemple, une table des matières (ou menu) et un lien vers un plan du site. Si conceptuellement, un évaluateur est capable d'identifier la table des matières (Figure 46, zone 2), cette information sémantique n'est pas contenue dans le code UsiXML. Une telle information aurait validé automatiquement les règles EvalWeb n°39 et n°41, ainsi que la règle WCAG 1.0 13.3 relative à la présence d'une table des matières et/ou d'un lien vers un plan du site. Nous pouvons noter que cette sémantique était présente dans l'étape de spécification précédente, validant ainsi automatiquement ces règles, mais qu'elle est perdue à cette étape.

Enfin, une partie des règles demeure non vérifiable car certains éléments ne peuvent pas être modélisés. C'est le cas, par exemple, des scripts et applets (règles WCAG 1.0 n°6.3, n°6.4, n°8.1, n°9.3), des feuilles de styles (règles WCAG 1.0 n°3.3, n°3.4, n°6.1), des raccourcis clavier (règle WCAG 1.0 n°9.5) ou encore des listes (règle WCAG 1.0 n°3.6).

6.4. ETAPE D'IMPLEMENTATION : PAGES WEB

La page Web finale de saisie de l'état civil, modélisée de manière abstraite dans l'étape de conception précédente, est présentée en Figure 48. L'interface de cette page a été développée à partir de l'IUA présentée en Figure 46 qui a servi de modèle pour l'agencement des différentes zones et la présentation du contenu. Cette page a été développée en XHTML pour le contenu (texte, champs de saisie, liens, etc.) et en CSS pour les différents styles et images (texte en gras, couleur du texte, images en vert dégradé utilisés pour les menus, etc.).

Figure 48 – Page Web HTML de l'étape "Saisie de l'état civil de l'enfant" pour la démarche d'inscription d'un enfant à l'école

L'ensemble des règles vérifiables sur l'interface abstraite vues dans la section 6.3 précédente, reste vérifiable sur la page Web finale. Toutefois, le niveau d'automatisation peut être différent. C'est le cas, par exemple, de la règle WCAG 1.0 n°2.2 sur le contraste entre couleur de fond et couleur du texte. Alors que ces deux informations étaient disponibles dans le code UsiXML de l'interface abstraite, sur l'application finale, ces informations ont été déportées dans la feuille de styles CSS et n'apparaissent pas dans la page XHTML évaluée. A moins de procéder à une évaluation des feuilles CSS, il n'est pas possible d'évaluer automatiquement cette règle.

En outre, puisque le contenu est pleinement renseigné dans la page Web finale, l'évaluation est plus complète, autrement dit, plus de règles peuvent être vérifiées. Par exemple, la hiérarchisation des titres (règle WCAG 1.0 n°3.5) peut être vérifiée : « Mairie de Bergen » (voir Figure 48) est ainsi un titre de niveau 1 (ce titre est contenu dans une balise h1 dans le code source), tandis que « Saisie de l'état civil de l'enfant » est un titre de niveau 2 (contenu dans une balise h2). D'autres règles peuvent également être vérifiées automatiquement telles que la présence d'une feuille de styles (règle WCAG 1.0 n°3.3), la langue du document (règle WCAG 1.0 n°4.3), la présence d'ordre de tabulation et de raccourcis clavier (règles WCAG 1.0 n°9.4 et n°9.5), l'association d'une étiquette avec son champ (règle WCAG 1.0 n°12.4) et la concordance du libellé d'un lien avec le titre de la page cible (règle EvalWeb n°44).

Les mêmes remarques faites précédemment sur l'interface abstraite et concernant la représentation d'éléments conceptuels tels que les éléments de navigation (voir section 6.3) sont valables également sur la page Web finale : la présence d'une table des matières ne peut être vérifiée que manuellement par l'évaluateur (règle WCAG 1.0 n°13.3) puisqu'un tel élément n'existe pas dans le langage XHTML.

6.5. ETAPE D'EVALUATION : APPLICATION WEB FINALE

A l'exécution, l'application Web peut fournir des informations supplémentaires pour l'inspection des règles de navigation et d'accessibilité. Concernant la navigation, les liens brisés posent un souci majeur d'utilisabilité (règles EvalWeb n°2, n°3 et n°4). S'il est possible de vérifier que tous les liens sont bien fonctionnels lors de l'implémentation des différentes pages, il est nécessaire que cette vérification soit effectuée de manière constante tout au long de l'existence de l'application. Celle-ci peut se faire lors de l'utilisation de l'application en testant automatiquement les liens de chaque page visitée. Les informations ainsi collectées peuvent être analysées et si des liens sont brisés, une correction plus rapide (c'est-à-dire plus réactive) peut être effectuée.

Nous avons également connaissance de l'URL exacte affichée à l'utilisateur lors de l'exécution de l'application. Les règles EvalWeb n°18 et n°43 stipulant que « l'URL doit être un chemin lisible et compréhensible » et que « l'URL doit être inclus en bas de chaque page » ne peuvent par conséquent être vérifiée qu'à cette étape du cycle de vie. Dans notre étude de cas, la vérification de la lisibilité et de la compréhension de l'URL, reste manuelle. En outre, cette URL n'est pas dupliquée en bas de la page comme le montre la Figure 48. La règle EvalWeb n°43 n'est donc pas respectée.

Concernant l'accessibilité, comme nous pouvons le voir dans le Tableau 21 (voir section 6.6), l'inspection de l'application à l'exécution n'apporte rien de plus pour les règles WCAG 1.0. Toutefois, ce constat ne signifie pas que les informations récoltées lors de l'exécution de l'application Web ne permettent pas de vérifications supplémentaires pour l'accessibilité. Par exemple, d'autres corpus de règles font état de règles ne pouvant être vérifiées qu'à l'exécution. Par exemple, la règle d'accessibilité n°12.8 du corpus AccessiWeb 1.0[32] stipule que : « Le poids des pages doit être limité à 70Ko ». Pour vérifier cette règle, il faut s'intéresser au poids de chaque donnée transitant entre le serveur et le client lorsqu'une page doit être affichée à l'utilisateur. Le poids d'une page Web lors de l'implémentation diffère ainsi du poids réel des pages Web à l'exécution puisque, par exemple, les images ne sont pas prises en compte dans le poids d'une page Web lors de l'implémentation.

[32] Le corpus de règles d'accessibilité AccessiWeb 1.0 a été remplacé au 9 juin 2008 par une nouvelle version 1.1. Les règles AccessiWeb 1.0 restent toutefois disponibles dans ce document : http://www.accessiweb.org/_repository/files/referentiel_accessibilite_version_2004.rtf

6.6. RESULTATS DETAILLES DE L'INSPECTION DES REGLES DE NAVIGATION ET D'ACCESSIBILITE

L'étude de cas présenté dans ce chapitre a porté sur l'inspection des règles d'accessibilité WCAG 1.0 et de navigation EvalWeb dans le cycle de vie. Les artefacts choisis ont permis de couvrir toutes les étapes du cycle de vie en O excepté :

- l'étape d'*expression des besoins* où les informations contenues dans le cahier des charges sont relatives aux besoins de l'application et ne sont par conséquent pas pertinentes pour une inspection automatique des règles d'accessibilité et de navigation ;
- l'étape de *maintenance du site* qui est une phase transitoire et qui ne consiste qu'à revenir soit sur le développement de l'application soit sur l'étape d'expression des besoins (voir Figure 42) : aucun artefact n'est par conséquent produit dans cette étape.

Les artefacts étudiés sont les suivants : trois modèles de navigation SWC dans l'étape de spécification, un modèle UsiXML de l'interface utilisateur abstraite dans l'étape de conception, et un fichier HTML pour les étapes de développement et d'utilisation & évaluation. Ce dernier est accompagné d'une feuille de styles CSS que nous n'avons pas pris en compte pour l'étude de cas. Par conséquent, nous n'en discuterons pas ici. Chaque artefact est replacé dans le cycle de vie dans la Figure 49.

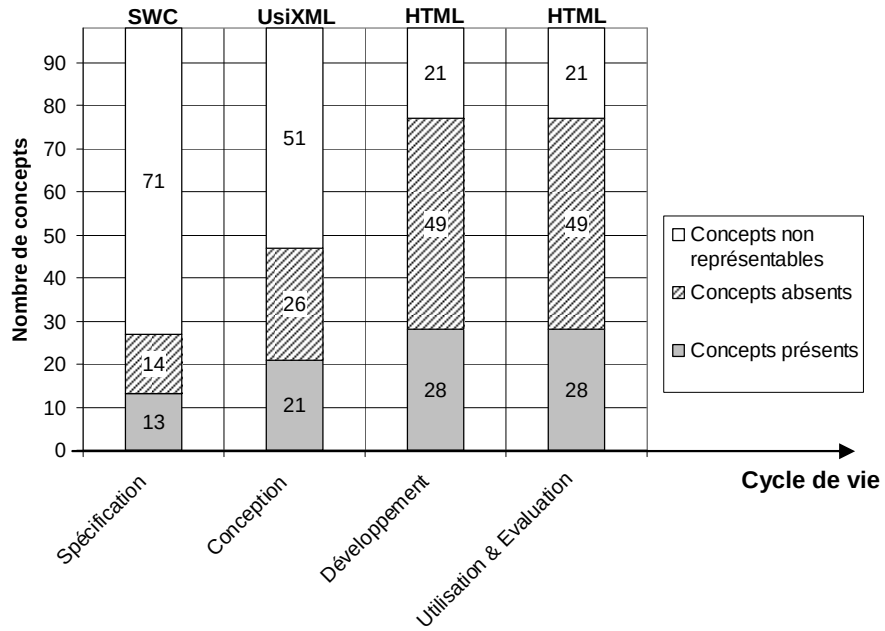


Figure 49 – Nombre de concepts représentés pour chaque artefact dans l'étude de cas

Nous avons vu tout au long de ce chapitre que ces artefacts pouvaient manipuler les mêmes concepts que ceux contenus dans l'ontologie. Par exemple, les modèles de navigation SWC produits permettent de représenter 13 concepts sur les 98 de l'ontologie tels que Page, Link, SupportContact, SiteMap, ou encore TableOfContents. Les concepts pouvant être représentés en SWC mais n'apparaissant pas dans les modèles SWC produits pour l'étude de cas sont au nombre de 14 : par exemple, les concepts ImageLink ou NavigationBar peuvent être modélisés en SWC mais n'apparaissent pas dans notre étude de cas. Enfin, la notation SWC ne permet pas de modéliser certains concepts tels que Image, List, Language, ou encore Button. Ces concepts sont au nombre de 71. Pour chacun des artefacts produits dans le cadre de l'étude de cas, nous avons indiqué en Figure 49 le nombre de concepts qui y étaient représentés, le nombre de concepts qui pouvaient être représentés par l'artefact mais qui sont absents, et le nombre de concepts qui ne sont pas représentables par l'artefact.

Nous présentons dans cette section les résultats détaillés de l'inspection ergonomique qui peut être menée sur les artefacts produits pour cette étude de cas. La méthode utilisée pour l'obtention détaillée de ces résultats est présentée dans la section 6.6.1 avant de présenter et d'analyser les résultats dans la section 6.6.2.

6.6.1. Méthode adoptée pour l'obtention des résultats de l'inspection automatique

Pour chacune des règles et pour chaque étape du cycle de vie, nous nous intéressons aux niveaux d'automatisation de chacune des règles inspectées. Certains de ces niveaux d'automatisation ont été présentés précédemment dans la section 3.5.3, nous les rappelons brièvement ici :

- Eléments absents : l'artefact considéré peut inclure cet élément absent, dans l'absolu. Mais, dans le cas de l'artefact évalué au moment de l'inspection, l'élément à vérifier n'est pas présent (par exemple, une page HTML peut inclure une image ; toutefois, certaines pages n'en contiennent pas). L'inspection ignore alors cette règle ;
- Non Vérifiable (NV) : la règle n'est pas vérifiable sur l'artefact concerné, car l'élément à évaluer ne peut pas être représenté sur cet artefact (par exemple, une image ne peut pas être représentée sur un modèle de navigation SWC) ;
- Analyse (An) : la règle peut être automatiquement inspectée par l'outil ;
- Aucun ou None (No) : l'outil prend en compte la règle mais ne la vérifie pas automatiquement. La vérification de la règle doit être manuelle.

Pour déterminer le niveau d'automatisation des règles, nous avons utilisé la méthode présentée dans la Figure 50.

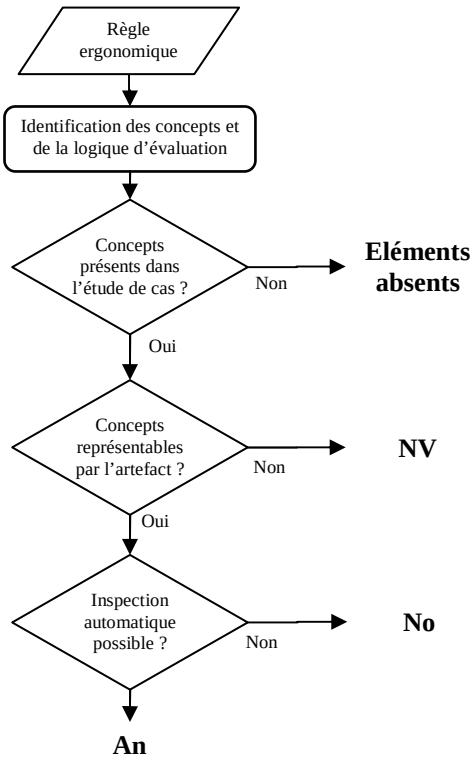


Figure 50 – Obtention des résultats sur l'inspection ergonomique des artefacts de l'étude de cas

L'inspection d'une règle ergonomique pour un artefact donné nécessite tout d'abord d'identifier les concepts à évaluer et la logique d'évaluation sur ces concepts. Si les concepts à évaluer sont absents de l'étude de cas, la règle ergonomique ne sera pas inspectée car des

éléments sont manquants (niveau d'automatisation *Eléments absents*), par exemple, l'étude de cas ne comporte pas d'éléments de type vidéo. Les règles relatives aux vidéos seront alors supposées comme validées. Si ces concepts sont présents, nous vérifions alors qu'ils peuvent être représentés par l'artefact. Dans le cas où au moins un des concepts ne peut pas être représenté, la règle ne pourra pas être vérifiée (niveau d'automatisation *Non Vérifiable*). Enfin, si tous les concepts peuvent être représentés, il faut étudier la logique d'évaluation de la règle pour vérifier si elle se prête à une automatisation ou non. Par exemple, une règle demandant de vérifier la pertinence du libellé d'un lien, ne pourra pas être inspectée automatiquement. Si l'inspection peut se faire automatiquement, le niveau d'automatisation de la règle sera Analyse (An), autrement, cette règle devra être inspectée manuellement (No).

6.6.2. Présentation et analyse des résultats

Les résultats de l'évaluation de la navigation et de l'accessibilité sur les artefacts produits pour le site Web de la mairie de Bergen sont affichés dans les figures de cette section. La Figure 51 présente les résultats de l'inspection relativement aux règles de navigation EvalWeb ; la Figure 52 présente les résultats de l'inspection relativement aux règles d'accessibilité WCAG 1.0. Les résultats détaillés pour ces deux corpus de règles sont présentés en fin de section dans le Tableau 20 et le Tableau 21.

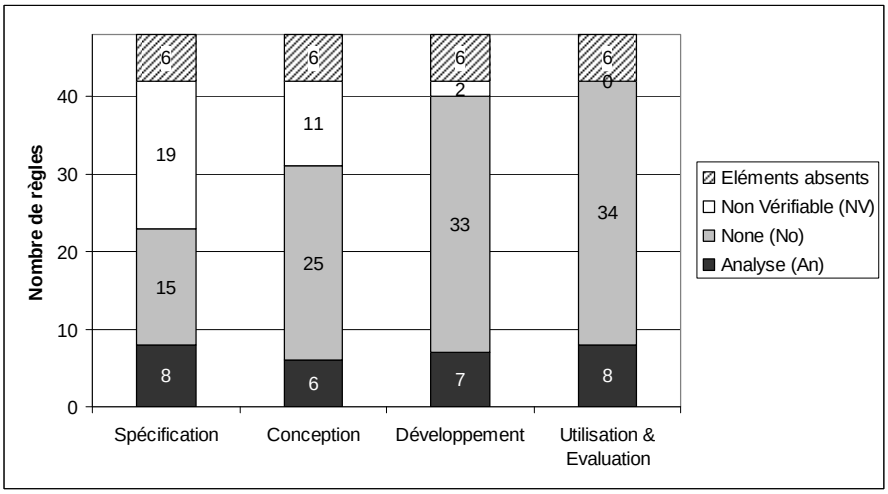


Figure 51 – Résultats de l'inspection des règles de navigation EvalWeb par niveaux d'automatisation dans le cadre de l'étude de cas

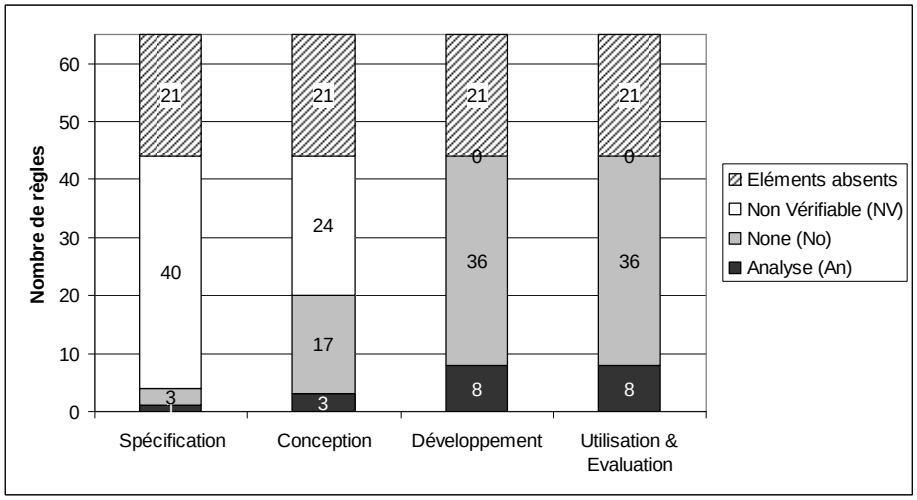


Figure 52 – Résultats de l'inspection des règles d'accessibilité WCAG 1.0 par niveaux d'automatisation dans le cadre de l'étude de cas

Le nombre de règles de navigation EvalWeb pouvant être supportées de manière automatique (An) reste constant dans chaque étape du cycle de vie (7 règles sur 48 en moyenne par artefact soit 14,6 % des règles, voir Figure 51). Ces règles concernent pour la plupart les liens hypertexte qui sont des éléments pouvant être représentés sur chacun des artefacts utilisés, expliquant ainsi pourquoi nous retrouvons sensiblement le même pourcentage de règles à chaque étape. Concernant les règles d'accessibilité (voir Figure 52), plus on avance dans le cycle de vie, plus le nombre de règles pouvant être automatiquement inspectées augmente : 1 règle sur 65 peut être inspectée sur les modèles SWC dans l'étape de spécification (soit 1,5 %), alors que 8 règles sur 65 peuvent l'être sur la page HTML dans l'étape d'utilisation & évaluation (soit 12,3 %). Ces chiffres s'expliquent par le fait que les règles d'accessibilité sont liées au contenu des pages Web, contenu qui est renseigné au fur et à mesure que nous avançons dans le processus de conception. Par exemple, la règle d'accessibilité n°12.4 « (pour un formulaire) associer les étiquettes avec leurs éléments de contrôle de manière explicite » ne peut pas être vérifiée sur les modèles de navigation SWC dans l'étape de spécification (cette notation ne permet pas de modéliser les formulaires) ; dans l'étape de conception, elle ne peut pas être inspectée automatiquement sur un modèle UsiXML (association impossible entre un contrôle de formulaire et son étiquette) ; dans les étapes de développement et d'utilisation & évaluation du site, cette règle peut être automatiquement inspectée.

A l'inverse, les résultats de cette étude de cas montrent également que certaines règles qui pouvaient être inspectées automatiquement au début du cycle de vie, peuvent ne plus l'être par la suite. Dans l'étape de spécification, les modèles de navigation SWC utilisés permettent de modéliser des aides à la navigation comme par exemple, les tables des matières, qui ne peuvent pas être représentés en UsiXML et en HTML. Cette sémantique fait que les règles EvalWeb n°39 « L'utilisation des index aide à l'orientation des utilisateurs » et n°41 « Offrez plusieurs approches pour la navigation » peuvent être inspectées automatiquement dans l'étape de spécification (voir Tableau 20). Dans les étapes suivantes, ces deux règles ne peuvent être inspectées que manuellement.

Certaines règles ne peuvent être inspectées que lors de l'exécution de l'application. Par exemple, la règle Evalweb n°43 « Il est souhaitable d'inclure l'adresse URL de la page en bas de la page » (voir Tableau 20) ne peut être vérifiée et inspectée que lorsque l'application est déployée, où l'URL de la page est alors accessible. Dans l'étape d'implémentation, cette URL n'est pas disponible et cette règle ne peut pas être inspectée automatiquement. Il est important de noter que l'artefact inspecté reste le même dans ces deux étapes : la page HTML. La seule différence est que des informations supplémentaires peuvent être collectées lors de l'utilisation de l'application. Cette étude de cas montre ainsi que l'inspection d'un même artefact dans des étapes différentes peut donner des résultats différents.

Actuellement, les outils d'inspection automatique ne permettent d'évaluer que l'application finale dans l'étape d'utilisation & évaluation. Le résultat précédent a montré que dans le cas des pages HTML, une partie des règles pouvant être inspectées pendant l'exécution de l'application pouvait également l'être pendant le développement des pages Web. Nous pourrions alors envisager l'implémentation d'outils de développement qui intègreraient directement des connaissances ergonomiques : cela permettrait de guider l'utilisateur pendant le développement des pages afin qu'il puisse se conformer à l'ensemble des règles qui peuvent être inspectées dans l'étape de développement. Ces outils permettraient ainsi de couvrir les règles pouvant être inspectées à cette étape et de laisser le soin aux outils d'inspection de vérifier les règles ergonomiques qui ne peuvent être inspectées que lors de l'exécution de l'application.

Tableau 20 – Résultats de la vérification des règles de navigation EvalWeb dans le cadre de l'étude de cas de la mairie de Bergen

N°de la règle	Spécification *SWC*	Conception *UsiXML*	Développement *HTML*	Utilisation & Evaluation *HTML*
1	An	An	An	An
2	An	An	An	An
3	An	An	An	An
4	An	An	An	An
5	No	No	No	No
6	NV	No	No	No
7	NV	No	No	No
8	NV	NV	No	No
9				
10	An	An	An	An
11	NV	NV	No	No
12	NV	NV	No	No
13	NV	NV	No	No
14	No	No	No	No
15	No	No	No	No
16				
17	NV	No	No	No
18	NV	NV	NV	No
19	NV	No	No	No
20	No	No	No	No
21	No	No	No	No
22	An	An	An	An
23				
24				
25	NV	No	No	No
26	No	No	No	No
27	No	No	No	No
28	No	No	No	No
29	NV	No	No	No
30	NV	No	No	No
31	No	No	No	No
32				
33	No	No	No	No
34	No	No	No	No
35	NV	NV	No	No
36	No	No	No	No
37	No	No	No	No
38	No	No	No	No
39	An	No	No	No
40	No	No	No	No
41	An	No	No	No
42	NV	No	No	No
43	NV	NV	NV	An
44	NV	NV	An	An
45				
46	NV	NV	No	No
47	NV	NV	No	No
48	NV	NV	NV	NV
Nombre de règles automatiquement inspectées	8	6	7	8
Pourcentage de règles automatiquement inspectées	16,7 %	12,5 %	14,6 %	16,7 %

Légende	
	Eléments absents
NV	Non Vérifiable
No	None (Aucun)
An	Analyse

Tableau 21 – Résultats de la vérification des règles d'accessibilité WCAG 1.0 dans le cadre de l'étude de cas de la mairie de Bergen

N°de la règle	Spécification *SWC*	Conception *UsiXML*	Développement *HTML*	Utilisation & Evaluation *HTML*
1.1	NV	An	An	An
1.2				
1.3				
1.4				
1.5				
2.1	NV	No	No	No
2.2	NV	An	No	No
3.1	NV	NV	No	No
3.2	NV	NV	No	No
3.3	NV	NV	An	An
3.4	NV	NV	No	No
3.5	NV	NV	An	An
3.6	NV	NV	No	No
3.7	NV	NV	No	No
4.1	NV	NV	No	No
4.2	NV	NV	No	No
4.3	NV	NV	An	An
5.1				
5.2				
5.3				
5.4				
5.5				
5.6				
6.1	NV	NV	No	No
6.2	NV	NV	No	No
6.3	NV	NV	No	No
6.4	NV	NV	No	No
6.5	NV	NV	No	No
7.1	NV	NV	No	No
7.2				
7.3	NV	NV	No	No
7.4				
7.5				
8.1	NV	NV	No	No
9.1				
9.2	NV	NV	No	No
9.3	NV	NV	No	No
9.4	NV	NV	An	An
9.5	NV	NV	An	An
10.1				
10.2	NV	No	No	No
10.3				
10.4	NV	An	An	An
10.5	NV	No	No	No
11.1	NV	NV	No	No
11.2	NV	NV	No	No
11.3	NV	No	No	No
11.4	An	No	No	No
12.1				
12.2				
12.3				
12.4	NV	No	An	An
13.1	No	No	No	No
13.2	No	No	No	No
13.3	No	No	No	No
13.4	NV	No	No	No
13.5				
13.6	NV	No	No	No
13.7				
13.8	NV	No	No	No
13.9	NV	No	No	No
13.10	NV	No	No	No
14.1	NV	No	No	No
14.2	NV	No	No	No
14.3	NV	No	No	No
Nombre de règles automatiquement inspectées	1	3	8	8
Pourcentage de règles automatiquement inspectées	1,5 %	4,6 %	12,3 %	12,3 %

Légende	
	Eléments absents
NV	Non Vérifiable
No	None (Aucun)
An	Analyse

6.7. CONCLUSION

Cette étude de cas simple a montré que les artefacts produits dans le cycle de vie lors de la conception d'une application Web, bien que de différentes natures (modèle de navigation, interface utilisateur abstraite, page Web et données d'usage), décrivent les mêmes éléments conceptuels tels que les liens, les images ou les contrôles de formulaire. La possibilité d'inspecter automatiquement ou manuellement les règles ergonomiques sur ces éléments a ainsi été mise en évidence sur les différents artefacts produits.

La conception de cette application a montré également que selon les artefacts, certains éléments ne pouvaient pas être représentés et que, en conséquence, certaines règles ne pouvaient simplement pas être vérifiées. C'est le cas, par exemple, des modèles de navigation SWC où le contenu des pages Web n'est pas modélisable. Ceci a une incidence notable sur les règles d'accessibilité WCAG 1.0 qui, en grande majorité, ne sont pas vérifiables sur les modèles SWC comme le montre le Tableau 21 (colonne Spécification).

Un des résultats intéressants que cette étude de cas a contribué à mettre en évidence, est que certaines règles ne pouvaient être vérifiées automatiquement que très tôt dans le processus de conception. Il semble logique de penser que plus on avance dans ce processus, plus des détails sur l'application sont disponibles, ce qui rend possible la vérification d'un plus grand nombre de règles et l'augmentation du niveau d'automatisation. Bien que ce raisonnement soit valide dans la majorité des cas, comme le montrent le Tableau 20 et le Tableau 21 (plus on avance dans le processus de conception, de gauche à droite dans ces tableaux, plus le nombre et le niveau d'automatisation des règles vérifiables augmente), certaines règles, au contraire, sont automatiquement vérifiables à certaines étapes mais pas dans les étapes ultérieures. Ceci est expliqué par le fait que certains éléments conceptuels peuvent être représentés dans des artefacts produits dans les étapes de spécification ou de conception et par conséquent, peuvent être inspectées de manière automatique. Mais lorsque ces éléments ne peuvent plus être représentés dans les étapes ultérieures, la sémantique de ces éléments n'est plus conservée et les règles ergonomiques liées à ces éléments ne peuvent plus être inspectées automatiquement dans la suite du cycle de vie.

Bien que cette étude de cas soit simple et minimaliste, la méthode d'inspection que nous avons proposée pour l'inspection automatique basée sur les artefacts produits dans le cycle de vie a été validée sur des projets de taille réelle, à l'échelle industrielle, sur la plateforme e-Citiz. Actuellement, certains téléservices développés depuis cette plateforme ont été déployés et font l'objet d'une utilisation quotidienne.

7. Conclusions et Perspectives

Actuellement, l'inspection ergonomique est souvent réalisée automatiquement sur l'application finale et si des erreurs sont identifiées, des corrections peuvent nécessiter une reconception de l'application. Pour éviter ces problèmes, les erreurs doivent être identifiées et corrigées le plus tôt possible dans le cycle de vie de l'application. Nos travaux de thèse se sont penchés sur l'étude et la proposition d'une méthode pour l'inspection automatique des applications Web tout au long du cycle de vie.

En début de conclusion, nous présentons tout d'abord un récapitulatif par chapitre des propositions que nous avons faites dans le cadre cette thèse, leurs limites, ainsi que les travaux qui restent à être effectués. Nous présentons ensuite les contributions que nous avons apportées dans le cadre de l'inspection automatique des applications Web. Enfin, nous discuterons des perspectives ouvertes par nos travaux.

Dans le chapitre 2, nous avons présenté un état de l'art sur les connaissances ergonomiques, les processus, méthodes et démarches de conception Web, et les outils de support basés sur des connaissances ergonomiques. Les connaissances ergonomiques existent sous plusieurs formes et sont collectées dans différentes sources. Ces connaissances sont cependant vastes et difficiles à réunir, et nécessitent par conséquent d'être organisées. Plusieurs travaux se sont attachés à l'organisation de ces connaissances [Mariage 05b] [Scapin 00b] [Grammenos 00] [Iannella 95] [Vanderdonckt 95] permettant ainsi, sur la base de ces connaissances organisées, de faciliter la conception et l'évaluation ergonomique des applications Web. L'étude des processus et méthodes de conception Web a montré qu'il était possible d'effectuer des évaluations ergonomiques sur les artefacts produits dans chacune des étapes du cycle de vie : modèles, descriptions d'interface, code de l'application, etc. Toutefois, les outils de support à la conception et à l'évaluation basés sur ces connaissances ergonomiques montrent qu'à l'heure actuelle, ces connaissances ne sont prises en compte que sur le code de l'application Web finale.

Dans le chapitre 3, nous avons proposé une méthode pour l'organisation des règles ergonomiques autour des éléments de l'interface utilisateur d'une application Web. L'ensemble de ces éléments a été identifié sur la base des corpus de règles EvalWeb et WCAG 1.0, et a été ensuite formalisé en une ontologie de concepts qui dresse ainsi un vocabulaire de termes sur lequel les règles peuvent être rattachées. En outre, cette ontologie a permis une étude sur l'inspection ergonomique avant l'implémentation de l'application finale. Les résultats de cette étude ont montré que certaines règles pouvaient être vérifiées automatiquement pendant les phases de conception, c'est-à-dire avant l'obtention de l'application Web finale, et par conséquent, des inspections ergonomiques pouvaient être réalisées à ces étapes.

L'ontologie couvre l'ensemble des règles ergonomiques que nous avons pu décrire dans nos travaux. Bien que ces règles ergonomiques proviennent de deux corpus de règles assez représentatifs pour le Web (les WCAG 1.0 constituant la référence internationale en matière d'accessibilité, et les règles EvalWeb étant une compilation de plusieurs sources de règles ergonomiques sur la navigation des applications Web), nous sommes conscients que l'ontologie doit être complétée, par exemple, par des corpus de règles régissant les applications mobiles, ou les interfaces plastiques afin de prendre en compte de nouveaux types d'interface. L'intégration de nouveaux concepts lors de la prise en compte de ces nouveaux corpus de règles nécessite alors que l'ontologie soit mise à jour. Pour cela, la méthode de développement que nous avons présentée dans ce chapitre peut être utilisée : elle nécessite toutefois une bonne connaissance de l'ontologie afin de ne pas introduire, par exemple, un concept synonyme d'un concept déjà présent.

Une ontologie d'un domaine (ici, l'ergonomie des applications Web) doit faire l'objet d'un consensus entre plusieurs personnes afin d'être acceptée. Elle nécessite par conséquent

d'être évaluée et les propriétés suivantes doivent être vérifiées : la syntaxe des définitions, la consistance, la complétude, et la concision. La validation de l'ontologie n'a pas été traitée dans cette thèse et reste à être réalisée.

Dans le chapitre 4, nous avons proposé une méthode pour l'inspection automatique des règles ergonomiques pouvant s'appliquer tout au long du cycle de vie de l'application dans les phases où des artefacts relatifs à l'interface utilisateur sont produits. Cette méthode se base fortement sur l'ontologie développée au chapitre 3. Les concepts de l'ontologie permettent de donner une description formelle des règles ergonomiques (en mettant en correspondance les règles avec les concepts de l'ontologie), de préparer un artefact pour son évaluation (en mettant en correspondance les concepts de l'artefact avec ceux de l'ontologie), et d'évaluer cet artefact (en mettant en correspondance à la fois les règles, l'artefact et l'ontologie).

L'application de notre méthode nécessite toutefois un effort initial avec des compétences bien précises (en ergonomie, informatique, et évaluation) pour les phases de mise en correspondance avec l'ontologie. De notre propre expérience, les mises en correspondance règles/ontologie et artefacts/ontologie sont longues et fastidieuses. Toutefois, ce travail n'est réalisé qu'une fois par règle et par artefact pour toutes les évaluations qui pourront être réalisées dans le cycle de vie d'une application. Ce travail de mises en correspondance peut également bénéficier au développement d'autres applications qui utiliseraient les mêmes artefacts (ou un sous-ensemble de ces artefacts) : l'inspection automatique de ces artefacts sera alors immédiate au sein de ces nouveaux projets.

Notre méthode propose d'utiliser la transformation de modèles pour réaliser les mises en correspondance et construire le modèle de l'application abstraite sur lequel l'inspection automatique est réalisée. Si des erreurs sont identifiées, les corrections à effectuer sur ce modèle doivent par conséquent être reportées sur l'artefact source. Ces corrections automatiques n'ont pas été discutées dans notre méthode : des études doivent encore être menées pour une proposition bien définie de la correction des artefacts dans le cadre d'une approche dirigée par les modèles.

Enfin, notre méthode ne permet pas de prendre en compte l'inspection combinée de plusieurs artefacts, autrement dit, les règles qui nécessitent l'évaluation simultanée de plusieurs artefacts. Même si nous n'avons été confrontés qu'à une règle de ce type, il est nécessaire, dans le futur, d'étudier l'inspection combinée afin d'augmenter la couverture des évaluations ergonomiques.

Dans le chapitre 5, nous avons détaillé l'outil que nous avons développé et qui supporte la méthode d'inspection proposée. Cet outil propose une architecture modulaire où chaque module est indépendant : les modules peuvent ainsi être remplacés selon les besoins et les contraintes d'un projet. Cet outil a été intégré dans e-Citiz, une plateforme de conception et de développement de téléservices. Cette plateforme est actuellement utilisée dans des projets de taille industrielle validant ainsi les fonctionnalités attendues de l'outil.

L'outil, tel que nous l'avons développé, reste toutefois à être amélioré : des développements sont encore nécessaires pour que les rapports d'évaluation générés puissent être configurés, sauvegardés et versionnés. En outre, ces rapports doivent pouvoir être produits au format EARL [EARL] proposé par le W3C/WAI pour d'une part, que l'évaluateur puisse évaluer la qualité des rapports produits par notre outil avec ceux d'autres outils, et d'autre part, pour coupler nos rapports avec ceux produits par un outil complémentaire. Concernant le développement des modules, seul le module de correction n'a pas été implémenté. Bien qu'aujourd'hui, nous ayons identifié les besoins en termes de fonctionnalités de ce dernier (correction automatique, semi-automatique ou manuelle), des problèmes d'ordre technique doivent cependant être résolus, notamment sur la correction d'artefacts stockés dans des formats différents (fichier texte, XML, ou binaire).

L'outil est actuellement intégré à la plateforme Eclipse, mais il n'a aucunes dépendances avec cette dernière. L'avantage d'être intégré à cette plateforme est que son utilisation par des évaluateurs familiers à Eclipse facilite son apprentissage et son adoption. Actuellement, nous

n'avons pas de retours d'expérience de l'outil d'évaluation de l'accessibilité intégré dans e-Citiz. Des études sur l'utilisabilité de l'outil restent encore à être menées pour s'assurer que les utilisateurs de l'outil, même novices, puisse l'utiliser sans soucis majeurs.

Dans le chapitre 6, nous avons présenté une étude de cas sur le site Web de la mairie fictive de Bergen. A travers les différents artefacts utilisés dans le cycle de vie, nous avons pu mettre en évidence les éléments nécessaires à l'inspection ergonomique des règles d'accessibilité (WCAG 1.0) et de navigation (EvalWeb). Les résultats ont montré que certaines règles qui étaient automatiquement inspectables à certaines phases du processus de conception ne l'étaient plus dans les phases ultérieures. Ceci est expliqué par le fait qu'il existe des concepts de haut niveau qui peuvent être représentés par exemple dans la phase de spécification, mais ne peuvent plus l'être sur les artefacts produits dans les phases suivantes. Enfin, cette étude de cas, bien que fictive, reste très proche de la réalité comme nous avons pu l'expérimenter avec e-Citiz. Elle est représentative, à la fois, des applications Web réelles, mais aussi, des artefacts qui sont produits dans les différentes activités du cycle de vie.

Les contributions de cette thèse sont multiples. Les artefacts produits dans le cycle de vie de l'application ont aujourd'hui un rôle qui se limite à accompagner les concepteurs dans l'avancement de l'application. La méthode que nous avons proposée contribue à une mise en avant de ces artefacts qui ne jouent plus ici un rôle de données transitoires mais de données sur lesquelles nous pouvons raisonner et travailler pour améliorer la qualité ergonomique dans toutes les phases du cycle de vie.

L'ontologie des éléments de l'interface utilisateur d'une application Web est une contribution originale puisqu'elle intègre des concepts directement issus des connaissances ergonomiques. Ainsi, contrairement aux ontologies entièrement dédiées à la conception (par exemple, [Plessers 05] et [Rossi 06]), nous trouvons des concepts relatifs à la conception et à l'évaluation ergonomique tels que les informations sur l'accessibilité, les aides à la navigation (que nous ne retrouvons pas nécessairement dans toutes les ontologies actuellement proposées), etc. L'utilisation de cette ontologie, puisque intégrant ces informations, donne la possibilité d'inspecter plus de règles et d'en inspecter plus tôt dans le processus de conception [Xiong 08a]. Cette ontologie apparaît également comme un moyen d'organiser la connaissance ergonomique par rapport aux éléments de l'interface utilisateur. Le vocabulaire utilisé dans les différentes sources de règles ergonomiques pour désigner ces éléments de l'interface est ainsi formalisé en un ensemble de concepts auxquels vont pouvoir être rattachées les règles ergonomiques. Ces concepts sont liés entre eux par des relations d'héritage ou d'agrégation.

La méthode d'inspection que nous proposons permet de prendre en compte des artefacts de différente nature et qui sont produits tout au long du cycle de vie [Xiong 06a]. Cette méthode rend ainsi possible l'inspection automatique dans toutes les étapes du cycle de vie produisant des artefacts : les règles qui jusqu'à présent ne pouvaient être inspectées automatiquement que sur l'application finale peuvent maintenant l'être dans tout le cycle de vie, autorisant ainsi des phases d'évaluation pendant la conception de l'application. Cette méthode a en outre l'avantage d'être générique puisqu'elle ne tient compte ni d'une méthode de conception ni d'un cycle de vie particulier.

Une contribution supplémentaire de cette thèse concerne l'outil supportant la méthode d'inspection proposée. Cet outil est composé de plusieurs modules indépendants et communiquant entre eux, et supporte les phases de notre méthode qui peuvent être automatisées. L'avantage est que cet outil suffit à lui seul pour accompagner l'évaluateur dans sa tâche dans toutes les étapes du cycle de vie. Il a en outre l'avantage d'être multiplateforme et de pouvoir être intégré à plusieurs environnements de développement selon les besoins et contraintes d'un projet. Cet outil a ainsi été intégré dans e-Citiz, un outil de conception de téléservices commercialisé et utilisé actuellement à grande échelle dans l'industrie.

Les travaux présentés dans cette thèse offrent des perspectives multiples dans le domaine. Les méthodes de conception actuelles manipulent des artefacts produits en conception dans l'objectif d'aboutir à l'application finale. Ces artefacts ne tiennent cependant pas compte des

connaissances relatives aux règles ergonomiques, autrement dit, les artefacts n'intègrent pas d'informations spécifiques pour augmenter les possibilités d'inspection. Des travaux pourraient être menés dans cette direction pour la conception d'artefacts sémantiquement riches et pouvant contenir des informations relatives aux connaissances ergonomiques. Il sera alors nécessaire d'étudier en détail la pertinence de l'enrichissement de ces artefacts, notamment pour éviter l'apparition de modèles intégrant un maximum d'informations mais dont une grande partie n'est pas pertinente pour la nature de ces modèles. Par exemple, un modèle de navigation a pour objectif de spécifier la navigation au sein de l'application ; il ne serait pas pertinent d'intégrer dans ce type de modèles des informations relatives à l'ergonomie d'autres aspects tels que le contenu (dont la modélisation n'est pas pertinente dans un modèle de navigation).

L'automatisation de l'inspection ergonomique sur l'application finale a abouti aujourd'hui à un ensemble d'outil permettant de certifier qu'une application est conforme à un ensemble de règles ergonomiques. C'est le cas par exemple des outils d'évaluation de l'accessibilité certifiant qu'une application Web est conforme aux recommandations WCAG 1.0. Nos travaux de thèse ont montré que nous pouvions également développer des outils pour automatiser l'inspection ergonomique pendant la conception et ainsi certifier, de la même manière, qu'une application est conforme à des recommandations à chaque étape du processus de conception. Des travaux pourraient être menées dans cette direction afin de définir clairement un processus d'évaluation qui permettrait de réaliser des inspections ergonomiques et d'assurer ainsi la conformité avec un ensemble de règles ergonomiques. Cette démarche consisterait ainsi, non plus à certifier l'application finale, mais le processus de développement de l'application.

L'automatisation de l'inspection ergonomique fournie par les outils permet, entre autre, un gain de temps considérable. Toutefois, un tel outil doit également proposer une correction automatique des erreurs détectées pour alléger également les phases de correction. Les artefacts produits dans le cycle de vie diffèrent selon la méthode de conception adoptée mais également selon leur nature : fichier texte, fichier XML, ou fichier binaire. Cela pose par conséquent des problèmes pour la correction automatique. Pour simplifier ce travail, il sera certainement nécessaire de se restreindre à quelques méthodes de conception (ou à quelques artefacts) et des études plus poussées devraient préciser à quelles conditions les corrections pourront être réalisées. Il faut alors être conscient que si nous gagnons en automatisation, nous perdrons toutefois en compatibilité : seuls certains artefacts bien précis pourront être corrigés automatiquement.

Cette thèse a ouvert la voie à l'inspection ergonomique tout au long du cycle de vie d'une application Web afin d'en assurer la qualité ergonomique à toutes les étapes de la conception. Les travaux futurs envisagés permettront de combler notamment les points suivants : la validation de l'ontologie, l'inspection combinée d'artefacts, la correction automatique (méthode et outil), et la production de rapports d'évaluation au format EARL pouvant être configurés, sauvegardés et versionnés. Les perspectives de travail offrent quant à elles des pistes intéressantes pour de futures recherches : l'intégration de connaissances ergonomiques directement incorporées dans les artefacts utilisés en conception afin d'augmenter les possibilités d'inspection, l'étude de la certification du processus de développement d'une application Web, et l'étude de la correction automatique d'artefacts dans une approche dirigée par les modèles.

Bibliographie

[A-Prompt] A-Prompt, Web Accessibility Verifier, Available at http://www.aprompt.ca/

[Abascal 04] Abascal, J.; Arrue, M.; Fajardo, I.; Garay, N.; Tomás, J. The use of guidelines to automatically verify Web accessibility, In *Universal Access in the Information Society*, special issue on "Guidelines, standards, methods and processes for software accessibility", Springer Verlag, Vol.3, N.1, 2004, pp.71-79.

[Abascal 06] Abascal J.; Arrue M.; Fajardo I.; Garay N. An expert-based usability evaluation of the EvalAccess web service. In R. Navarro-Prieto, J. Lorés (Eds.): *HCI related papers of Interacción 2004*, Springer, Netherlands, 2006, pp. 1-17.

[Abrams 99] Abrams, M.; Phanouriou, C.; Batongbacal, A.L.; Williams, S.; Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language, In A. Mendelzon, editor, *Proceedings of 8th International World Wide Web Conference, WWW'8*, Toronto, May 11-14, 1999, Elsevier Science Publishers, Amsterdam, 1999.

[AccessiWeb] AccessiWeb. Accessible à l'adresse suivante : http://www.accessiweb.org/

[Acerbis 04] Acerbis, R.; Bongio, A.; Butti, S.; Ceri, S.; Ciapessoni, F.; Conserva, C.; Fraternali, P.; Carughi, G. T. WebRatio, an Innovative Technology for Web Application Development, In *Proceedings of ICWE'2004*, Munich, Germany, July 28-30, 2004.

[Acerbis 08] Acerbis, R.; Bongio, A.; Brambilla, M.; Butti, S.; Ceri, S.; Fraternali, P. Web applications design and development with WebML and WebRatio 5.0, In *Proceedings of TOOLS Europe 2008*, 2008.

[Adobe Dreamweaver] Adobe Dreamweaver CS3. Available at http://www.adobe.com/products/dreamweaver/

[Alexander 77] Alexander, C. et al., *A Pattern Language*, Oxford University Press, New York, USA, 1977.

[Ali 03] Ali, M.F.; Pérez-Quiñones, M.A.; Abrams, M. Building Multi-Platform User Interfaces with UIML, In Seffah, A., Javahery, H. (eds.), *Multiple User Interfaces: Engineering and Application Framework*, John Wiley and Sons, New York, 2003.

[Apple 06] Apple Computer, *Apple Human Interface Guidelines*, 2006, Available at http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html

[Arrue 07] Arrue, M.; Vigo, M.; Abascal, J. Including heterogeneous web accessibility guidelines in the development process, In *Proceedings of Engineering Interactive Systems, EIS*, Salamanca, Spain, March 22-24, 2007.

[ATRC] ATRC Web Accessibility Checker, Available at http://checker.atrc.utoronto.ca

[Baresi 06] Baresi, L.; Colazzo, S.; Mainetti, L.; Morasca, S. W2000: A Modelling Notation for Complex Web Applications, In *Web Engineering*, Chapter 11, Springer-Verlag, pp. 335-364, 2006.

[Beirekdar 02] Beirekdar, A.; Vanderdonckt, J.; Noirhomme-Fraiture, M. A Framework and a Language for Usability Automatic Evaluation of Web Sites by Static Analysis of HTML Source Code, Chapter 29, in *Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces (CADUI 2002)*, Valenciennes, France, 15-17 May, Kluwer Academics Pub., Dordrecht, 2002, pp. 337-348

[Beirekdar 04] Beirekdar, A.; Noirhomme-Fraiture, M.; Mariage, C.; Vanderdonckt, J. DESTINE: outil d'aide à l'évaluation contextualisée de l'ergonomie des sites web, In

Proceedings of the 16th French-speaking Conference on Human-Computer Interaction, Namur, Belgium, 1-3 September, 2004

[Beirekdar 05] Beirekdar, A.; Keita1, M.; Noirhomme-Fraiture, M.; Randolet, F.; Vanderdonckt, J.; Mariage, C. Flexible Reporting for Automated Usability Evaluation of Web Sites, In *Proceedings of 10th IFIP TC13 International Conference on Human-Computer Interaction*, *INTERACT 2005*, 12-16 September, Rome, Italy, LNCS 3585, Springer-Verlag, pp. 281-294, 2005

[Berners-Lee 98] Berners-Lee, T. Style Guide for online hypertext, 1998, Available at http://www.w3.org/Provider/Style/

[Bichler 96] Bichler, M. and Nusser, S. Modular design of complex Web-applications with W3DT. In *the 5th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE'96)*, 1996.

[Boehm 86] Boehm, B. A spiral model of software development and enhancement, In *ACM SIGSOFT Software Engineering Notes*, 11(4), 1986, pp. 14-24.

[Bowers 96] Bowers, Neil. Weblint: Quality assurance for the World-Wide Web. In *Proceedings of 5th International World Wide Web Conference*, Paris, France, May 1996. Amsterdam, the Netherlands: Elsevier Science Publishers. Available at http://www.ra.ethz.ch/CDstore/www5/www162/overview.htm

[Borges 96] Borges, J.A.; Morales, I.; Rodríguez, N.J. Guidelines for Designing Usable World Wide Web Pages, In *Companion Proceedings of ACM Conference on Human Aspects in Computing Systems*, CHI'96, Vancouver, April 14-18, ACM Press, New York, 1996, pp. 277–278.

[Brambilla 07] Brambilla, M.; Cabot, J.; Moreno, N. Tool Support for Model Checking of Web application designs, In *Proceedings of 7th International Conference on Web Engineering*, *ICWE 2007*, 16-20 July, Como, Italy, LNCS 4607, Springer-Verlag, pp. 533-538, 2007.

[Brinck 02] Brinck, T.; Hermann, D.; Minnebo, B.; Hakim, A. AccessEnable: A Tool for Evaluating Compliance with Accessibility Standards, In *Proceedings of the Workshop on Automatically Evaluating the Usability of Web Sites*, CHI 2002, Minneapolis, MN, 21-22 April, 2002.

[Budinsky 04] Budinsky, F.; Steinberg, D.; Ellersick, R.; Grose, T. Ecore Modeling Concepts, Chapter 5, In *Eclipse Modeling Framework*, Addison Wesley Professional, 2004.

[Calvary 03] Calvary, G.; Coutaz, J.; Thevenin, D.; Limbourg, Q.; Bouillon, L.; Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. In *Interacting with Computers*, vol. 15, 3, 2003, pp. 289-308.

[Centeno 07] Centeno, V.L.; Kloos, C.D.; Blázquez, J.M.; Gaedke, M. Web Accessibility Evaluation via XSLT, In *Proceedings of 1st International Workshop on Web Usability and Accessibility*, *IWWUA 2007*, Nancy, France, December 3-7, 2007, LNCS 4832, Springer, pp. 459-469.

[Ceri 00] Ceri, S.; Fraternali, P.; Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites, In *Proceedings of the 9th international World Wide Web conference on Computer networks: the international journal of computer and telecommunications networking*, Amsterdam, the Netherlands, pp. 137-157, 2000.

[Chandrasekaran 99] Chandrasekaran, B.; Josephson, J.R.; Benjamins, V.R. What Are Ontologies, and Why Do We Need Them?, In *IEEE Intelligent Systems*, vol. 14, n° 1, pp. 20-26, 1999.

[Corcho 02] Corcho, O.; Fernández-López, M.; Gómez-Pérez, A. Methodologies, tools and languages for building ontologies. Where is their meeting point? In *Data and Knowledge Engineering*, vol. 46, N. 1, pp. 41-64, 2002.

[Correani 06] Correani, Francesco; Leporini, Barbara; Paternò, Fabio. Automatic Inspection-based Support for Obtaining Usable Web Sites for Vision-Impaired Users. In *Universal Access in the Information Society*, Springer-Verlag, 11 may 2006.

[Cooper 99] Cooper, M.; Limbourg, Q.; Mariage, C.; Vanderdonckt, J. Integrating Universal Design into a Global Approach for Managing Very Large Web Sites, In *Proceedings of 5th ERCIM Workshop on User Interfaces for All*, *UI4ALL'99*, Dagstuhl, 28 November-1 December, 1999, A. Kobsa & C. Stephanidis (eds.), GMD Report 74, GMD - Forschungszentrum Informationstechnik GmbH, Sankt Augustin, 1999, pp. 131-150. Available at: http://ui4all.ics.forth.gr/UI4ALL-99/Cooper.pdf

[Cowan 95] Cowan, D. D.; Lucena, C. J. P. Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse", *IEEE Transactions on Software Engineering*, Vol.21, No.3, March 1995.

[Coyette 07] Coyette, A.; Kieffer, S.; Vanderdonckt, J. Multi-fidelity Prototyping of User Interfaces, In *Proceedings of 11th IFIP TC 13 International Conference on Human-Computer Interaction*, *INTERACT'2007*, Rio de Janeiro, 10-14 September 2007, Springer-Verlag, LNCS Vol. 4662, 2007, pp. 149-162.

[Czarnecki 03] Czarnecki, K.; Helsen, S. Classification of Model Transformation Approaches, In *Proceedings of the OOPSLA'03 Workshop on the Generative Techniques in the context of Model-Driven Architecture*, Anaheim, California, USA, 2003.

[Devedžic 02] Devedžic, V. Understanding Ontological Engineering, *Communications of the ACM*, vol. 45, n°4, pp.136-144.

[Dix 03] Dix, A.; Finlay, J. E.; Abowd, G. D.; Beale, R. *Human-Computer Interaction* (3rd Edition), Prentice-Hall, Inc., Upper Saddle River, NJ, 2003

[Doctor Watson] Doctor Watson, version 5.1. Available at http://watson.addy.com/

[EARL] Evaluation and Report Language (EARL) 1.0. Available at www.w3.org/TR/EARL10/

[eCitiz] e-Citiz, Le monde des téléservices à portée de clic, http://www.e-citiz.com/

[Faraday 00] Faraday, Pete. Visually Critiquing Web Pages. In *Proceedings of the 6th Conf. on Human Factors & the Web*, Austin, Texas, June 2000. Available at http://facweb.cs.depaul.edu/cmiller/faraday/Faraday.htm

[Farenc 96] Farenc, C.; Liberati, V.; Barthet, M.-F. Automatic Ergonomic Evaluation: What are the Limits?, In *Proceedings of 2nd International Workshop on Computer-Aided Design of User Interfaces*, *CADUI'96*, Namur, 5-7 June, 1996, J. Vanderdonckt (ed.), Presses Universitaires de Namur, Namur, 1996.

[Farenc 99] Farenc, C.; Palanque, P.; Bastide, R. Embedding Ergonomic Rules as Generic Requirements in the Development Process of Interactive Software, In *Proceedings of 7th IFIP Conference on Human-Computer Interaction*, *Interact'99*, Edinburgh, Scotland, 30 August - 3 September, 1999.

[Farenc 01] Farenc, C; Palanque, P.; Bastien, C.; Scapin, D.; Winckler, M. Towards a General Guidance and Support Tool for Usability Optimization, In *Proceedings of 1st International Conference on Universal Access in Human-Computer Interaction*, *UAHCI2001*, 5-10 August, 2001, New Orleans, USA.

[Fleming 98] Fleming, J., Web Navigation: Designing the User Experience, O'REILLY, 1998.

[Gamma 95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. *Design Patterns: Elements of Reusable Design*. Reading, Massachusetts: Addison-Wesley, 1995

[Gardner 03] Gardner, T.; Griffin, C.; Koehler, J.; Hauser, R. *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard*, OMG Document: ad/03-08-02.

[Gómez 03] Gómez, J.; Cachero, C. OO-H Method: extending UML to model web interfaces, *Information modeling for internet applications*, 2003, pp. 144-173.

[Gómez 04] Gómez, J. Model-Driven Web Development with VisualWADE, In *Proceedings of ICWE'2004*, Munich, Germany, July 28-30, 2004, pp. 611-612.

[Gómez-Pérez 95] Gómez-Pérez, A. Some ideas and examples to evaluate ontologies, In *Proceedings of 11th Conference on Artificial Intelligence Applications*, IEEE Computer Society Press, 1995.

[Gómez-Pérez 01] Gómez-Pérez, A. Evaluation of ontologies, *International Journal of Intelligent Systems*, vol. 16, n°3, pp. 391-409, 2001.

[Grammenos 00] Grammenos, D.; Akoumianakis, D.; Stephanidis, C. Integrated Support for Working with Guidelines: the Sherlock Management System, In *Interacting with Computers*, 12(3), 2000, pp. 281-311.

[Gruber 93] Gruber T. R. A translation approach to portable ontology specifications, *Knowledge Acquisition*, vol. 5, n°2, pp. 199-220, 1993.

[Gruninger 95] Gruninger, M., Fox, M.S. Methodology for the design and evaluation of ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing*, International Joint Conference on Artificial Intelligence, 1995.

[Gupta 89] Gupta, A. ; Forgy, C.; Newell, A. High-speed implementations of rule-based systems, In *ACM Transactions on Computer Systems (TOCS)*, Volume 7, Issue 2, 1989, pp.119-146.

[Harel 87] Harel, D., Statecharts: a Visual Formalism for Complex Systems, *Science of Computer Programming*, 8, 1987, pp. 231-274.

[Hennicker 00] Hennicker, R.; Koch, N. A UML-based Methodology for Hypermedia Design. In *Proceedings of the Unified Modeling Language Conference, UML'2000*, 2000, Evans A. and Kent S., Eds. LNCS 1939, Springer Verlag, pp. 410-424.

[HFES ANSI 200] HFES/ANSI 200. Draft HFES/ANSI 200 Standard, Section 5: Accessibility, 1997.

[Hix & Hartson 93] Hix, D.; Hartson, R. *Developing user interfaces: ensuring usability through product and process*, New York, USA, John Wiley & Sons, 1993.

[Iannella 95] R. Iannella, HyperSAM: a management tool for large user interface guideline sets, In *ACM SIGCHI Bulletin* 27 (2), 1995, pp. 42-45.

[IBM 04] IBM Web Accessibility Checklist, Version 3.5, 2004. Available at http://www-03.ibm.com/able/guidelines/web/accessweb.html

[IBM 06] IBM, Style Guidelines, 7th edition, November 2006. Available at http://www-03.ibm.com/easy/page/1387

[ISO9241] ISO, Draft International Standard (DIS) 9241. *Ergonomic Requirements for Office Work with Visual Display Terminals*, Geneva, 1999.

[Ivory 03] Ivory, M.Y. *Automated Web Site Evaluation: Researchers' and Practitioners' Perspectives*, Kluwer Academic Publishers, Dordrecht, 2003.

[Jouault 05] Jouault, F.; Kurtev, I. Transforming Models with ATL, In *Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005*, Montego Bay, Jamaica, 2005.

[Koch 02] Koch, N.; Kraus, A. The expressive Power of UML-based Web Engineering, In *the 2nd International Workshop on Web-oriented Software Technology (IWWOST02)*, 2002.

[Kruchten 00] Kruchten, P. *The Rational Unified Process: An Introduction*, 2nd ed., Addison-Wesley Professional, 2000.

[Ivory 01] Ivory, M. Y., Hearst, M. A.: The state of the art in automating usability evaluation of user interfaces, *ACM Computing Surveys (CSUR)*, v.33 n.4, December, 2001, pp.470-516.

[LawStanca 04] Law Stanca, Law no. 4, January 9, 2004: Provisions to support the access to information technologies for the disabled. www.pubbliaccesso.it/normative/law_20040109_n4.htm_20040109_n4.htm

[Leporini 04] Leporini, Barbara; Paternò, Fabio. Increasing Usability when Interacting through Screen Readers. *International Journal Universal Access in the Information Society (UAIS)*, Special Issue on "Guidelines, standards, methods and processes for software accessibility", Springer Verlag, Vol.3, N.1, pp. 57-70.

[Leporini 06] Leporini, B.; Paternò, F.; Scorcia, A. Flexible tool support for accessibility evaluation, In *Interacting with Computers*, vol. 18, issue 5, Sept. 2006, pp. 869-890.

[LIFT Machine] LIFT Machine. http://www.usablenet.com/usablenet_liftmachine.html

[Limbourg 04] Limbourg, Q.; Vanderdonckt, J.; Michotte, B.; Bouillon, L.; Víctor López Jaquero. UsiXML: a Language Supporting Multi-Path Development of User Interfaces, In *Proceedings of 9th IFIP Working Conference on Engineering for Human-Computer Interaction* jointly with *11th International Workshop on Design, Specification, and Verification of Interactive Systems*, *EHCI-DSVIS'2004*, Hamburg, July 11-13, LNCS 3425, Springer-Verlag, Berlin, 2005, pp. 200-220.

[Loi Accessibilité France 05] Loi n° 2005-102 du 11 février 2005 pour l'égalité des droits et des chances, la participation et la citoyenneté des personnes handicapées, Journal Officiel de la République Française. Disponible à l'adresse suivante : http://www.legifrance.gouv.fr/affichTexte.do?cidTexte=JORFTEXT000000809647&dateTexte=

[Lynch & Horton 02] Lynch, P. J.; Horton, S. *Web Style Guide*, 2nd edition, Yale University, 2002. Available at http://webstyleguide.com/

[Mariage 05a] Mariage, C.; Vanderdonckt, J.; Pribeanu, C. State of the Art of Web Usability Guidelines, Chapter 41, In R.W. Proctor, K.-Ph.L. Vu (eds.), *The Handbook of Human Factors in Web Design* , Lawrence Erlbaum Associates, Mahwah, 2005.

[Mariage 05b] Mariage, C. MetroWeb : logiciel de support à l'évaluation de la qualité ergonomique des sites web, Thèse de doctorat, Université Catholique de Louvain, Mars 2005.

[McDermid & Ripken 84] McDermid, J.A.; Ripken, K. *Life Cycle Support in the Ada Environment*, New York, NY, USA: Cambridge University Press, 1984. Available at http://www.amazon.fr/Life-Cycle-Support-Ada-Environment/dp/0521260426/ref=sr_1_1/171-1675315-1781838?ie=UTF8&s=english-books&qid=1186578472&sr=8-1

[Microsoft 02] Microsoft Corporation, *Windows XP Visual Guidelines*, 2002, Available at http://www.microsoft.com/whdc/System/platform/pcdesign/XPguidelines.mspx

[Microsoft Expression Web] Microsoft Expression Web, Available at http://www.microsoft.com/france/expression/expression-web/default.mspx

[MIL-STD-1472C] Military Standard: *Human Engineering Design Criteria for Military Systems, Equipment and Facilities*. Washington, DC: Department of Defense, 1st September, 1983.

[mobileOK] W3C mobileOK Basic Tests 1.0, W3C Candidate Recommendation (30 November 2007), Available at www.w3.org/TR/mobileOK-basic10-tests

[MonashUniversity 06] Monash University, Web Style Guide, Version 3, November 2006. Available at http://www.monash.edu.au/staff/web/

[Moreno 08] Moreno, L.; Martínez, P.; Ruiz, B. A MDD approach for modelling web accessibility, To Be Published in *Proceedings of 7th International Workshop on Web-Oriented Software Technologies*, *IWWOST'2008*, in Conjunction with ICWE'2008, Yorktown Heights, New York, USA, 14th July 2008.

[NetMechanic] NetMechanic. http://www.netmechanic.com/

[Nielsen 93] Nielsen, J. *Usability Engineering*, Academic Press, 1993.

[Nielsen 94] Nielsen, J.; Mack, R.L. *Usability inspection methods*, John Wiley and Sons, New York, 1994.

[Nielsen 00] Nielsen, J. *Designing Web Usability: The Practice of Simplicity*, New Riders Publishing Thousand Oaks, CA, USA, 2000.

[NIST Web Metrics] NIST Web Metrics, Available at http://zing.ncsl.nist.gov/WebTools/index.html

[Noy & McGuinness 01] Noy, N.F.; McGuinness, D.L. *Ontology Development 101: A Guide to Creating Your First Ontology*, Knowledge Systems Laboratory, Stanford University, CA, 2001.

[Ocawa] Ocawa. Disponible à l'adresse http://www.ocawa.com/

[Ohnemus 97] Ohnemus, K. R. Web Style Guides: Who, What, Where, In *Proceedings of the 15th annual ACM international Conference on Computer Documentation, SIGDOC97,* Salt Lake City, October 19-22, 1997, pp.189-197.

[OWL 04] OWL Web Ontology Language: Overview. W3C Recommendation 10 February 2004. Available at: http://www.w3.org/TR/owl-features/

[Pastor 06] Pastor, O.; Fons, J.; Pelechano, V.; Abrahão, S. Conceptual Modelling of Web Applications: The OOWS Approach, In *Web Engineering*, Chapter 9, Springer-Verlag, pp. 277-302, 2006.

[Paternò 97] Paternò, F.; Mancini, C. and Meniconi, S. ConcurTaskTree: A diagrammatic notation for specifying task models, In *Proceedings of 13th International Conference on Human-Computer Interaction, Interact'97,* Chapman & Hall, Sydney, July 1997, pp.362-369.

[Paternò 03] Paternò, F.; Santoro, C. A Unified Method for Designing Interactive Systems Adaptable to Mobile and Stationary Platforms, In *Interacting with Computers*, vol.15, Issue 3, 2003, pp. 349-366.

[Plessers 05] Plessers, P.; Casteleyn, S.; Yesilada, Y.; De Troyer, O.; Stevens, R.; Harper, S.; Goble, C. Accessibility: a Web engineering approach, In *Proceedings of 14th World Wide Web Conference*, WWW'05, ACM, 2005.

[Raggett 98] Raggett, D. Clean Up Your Web Pages with HP's HTML Tidy. In *Computer Networks 30(1-7)*, 1998, pp. 730-732.

[RGAA] Référentiel Général d'Accessibilité des Administrations. Accessible à l'adresse suivante : http://rgaa.referentiels.modernisation.gouv.fr/

[Ricca & Tonella 01a] Ricca, F.; Tonella, P. Analysis and testing of web applications, In *Proceedings of 23rd International Conference on Software Engineering, ICSE2001*, 2001, pp. 25-34.

[Ricca & Tonella 01b] Ricca, F.; Tonella, P. Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions, In *Proceedings of TACAS'2001, Tools and Algorithms for the Construction and Analysis of Systems*, Geneva, Italy, April 2-6, 2001, LNCS 2031, pp.373-388.

[Rossi 06] Rossi, G.; Schwabe, D. Model-Based Web Application Development, In *Web Engineering*, Chapter 10, Springer-Verlag, pp. 303-333, 2006.

[Royce 70] Royce, W.W. Managing the development of large software systems: concepts and techniques, In *Proceedings of WESCON*, Monterey, California, USA, 1970, pp. 1-9.

[Salmre 05] Salmre, I. Characteristics of Mobile Applications, Chapter 2, In *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications*, Addison Wesley Professional, 2005.

[Scapin 99] Scapin, D.L. ; Garrigues, S. ; Farenc, C. ; Vanderdonckt, J. ; Palanque, P. ; Bastide, R. ; Bastien, J.M.C. ; Leulier, C. *Conception ergonomique d'interfaces web : démarche et outil logiciel de guidage et de support*, INRIA, 1999.

[Scapin 00a] Scapin, D.; Vanderdonckt, J.; Farenc, C.; Bastide, R.; Bastien, C.; Leulier, C.; Mariage, C.; Palanque, P., *Transferring Knowledge of User Interfaces Guidelines to the Web*, Springer Verlag, 2000.

[Scapin 00b] Scapin, D.; Leulier, C.; Vanderdonckt, J.; Mariage, C.; Bastien, C.; Farenc, C.; Palanque, P.; Bastide, R. A Framework for Organizing Web Usability Guidelines, In *Proceedings of 6th Conference on Human Factors and the Web*, 19 June, Austin, Texas, 2000.

[Scharl 99] Scharl, A. A Conceptual, User-Centric Approach to Modeling Web Information Systems. In *the 5th Australian WWW Conference*, 1999.

[Schmidt 06] Schmidt, D.C. Model-Driven Engineering, *IEEE Computer*, Vol. 39, No. 2, February 2006, pp. 41-47.

[Schwabe 98] Schwabe, D.; Rossi, G. Developing hypermedia applications using OOHDM, *Workshop on Hypermedia Development Processes: Methods and Models*, *Hypertext'98*, 1998.

[Sierkowsky 02] Sierkowsky, B. Achieving Web Accessibility, In *Proceedings of the 30th annual ACM conference on User Services*, Providence, Rhode Island, USA, 2002.

[Smith & Mosier 86] Smith, S.L.; Mosier, J.N. *Guidelines for Designing User Interface Software*, Technical Report ESD-TR-86-278, The Mitre Corporation, Bedford, 1986.

[Souchon 03] Souchon, N.; Vanderdonckt, J. A Review of XML-Compliant User Interface Description Languages, In *Proceedings of 10th International Conference on Design, Specification, and Verification of Interactive Systems, DSV-IS'2003*, Madeira, 4-6 June, 2003, Jorge, J., Nunes, N.J., Falcao e Cunha, J. (Eds.), LNCS 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.

[Sun 99] Sun Microsystems, *Java Look and Feel Design Guidelines*, Microsystems, Palo Alto, CA, 1999, Available at http://java.sun.com/products/jlf/ed1/dg/index.htm

[Takata 04] Takata, Y.; Nakamura, T.; Seki, H. Accessibility verification of WWW documents by an automatic guideline verification tool, In *Proceedings of 37th Annual Hawaii International Conference on System Sciences*, 5-8 January, 2004.

[TAW] TAW, *Web Accessibility Test*. Available at http://www.tawdis.net/taw3/cms/en

[Thatcher 02] Thatcher, J.; Bohman, P.; Burks, M.; Henry, S.L.; Regan, B.; Swierenga, S.; Urban, M.; Wadell, C. *Constructing Accessible Web Sites*, Glasshaus, Birmingham, UK, 2002.

[Thatcher 06] Thatcher, J.; Burks, M.; Heilmann, C.; Henry, S.L.; Kirkpatrick, A.; Lauke, P.H.; Lawson, B.; Regan, B.; Rutter, R.; Urban, M.; Wadell, C. *Web Accessibility: Web Standards and Regulatory Compliance*, Friends of Ed, July, 2006.

[Theng 98] Theng, Y. L. and Marsden, G. Authoring tools: Towards continuous usability testing of web documents. In *Proceedings of the 1st int. Workshop on Hypermedia Development*,

Pittsburgh, PA, June, 1998, Available at http://www.eng.uts.edu.au/»dbl/HypDev/ht98w/YinLeng/HT98 YinLeng.html.

[Tratt 05] Tratt, L. Model transformations and tool integration, *Journal of Software and Systems Modelling*, 4(2), May 2005, pp. 112-122.

[Uschold & Gruninger 96] Uschold, M.; Gruninger, M. Ontologies: Principles, Methods and Applications, In *Knowledge Engineering Review*, vol. 11, n°2, pp.93-136, 1996.

[UseIt] Useit.com: usable information technology, Jakob Nielsen's Website. http://www.useit.com/

[Vanderdonckt 93] Vanderdonckt, J.; Bodart, F. Encapsulating knowledge for intelligent automatic interaction objects selection. In S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, and T. White, editors, *Proceedings of the ACM Conference on Human Factors in Computing Systems*, *InterCHI'93*, Amsterdam, 24-29 April 1993, New York, ACM Press, 1993, pp. 424-429.

[Vanderdonckt 94] Vanderdonckt, J. *Guide ergonomique des interfaces homme-machine*, Presses Universitaires de Namur, Namur, 1994, ISBN 2-87037-189-6.

[Vanderdonckt 95] Vanderdonckt, J. Accessing Guidelines Information with SIERRA, In *Proceedings of 5th IFIP Conference on Human-Computer Interaction*, *INTERACT'95*, Lillehammer, Norway, June 25-29, 1995, pp. 311-316.

[Vanderdonckt 99] Vanderdonckt, J. Development Milestones towards a Tool for Working with Guidelines, *Interacting with Computers*, Vol. 12, N°2, November 1999, pp. 81-118.

[WCAG] Web Content Accessibility Guidelines 1.0 (WCAG), *W3C Recommendation 5 May 1999*, available at http://www.w3.org/TR/WCAG10/

[WebRatio] WebRatio, You think You get. http://www.webratio.com

[WebXACT] Watchfire WebXACT, Available at http://webxact.watchfire.com/

[Winckler 03] Winckler, M.; Palanque, P. StateWebCharts: a Formal Description Technique Dedicated to Navigation Modelling of Web Applications, *DSV-IS*, Portugal, 2003

[Winckler 04] Winckler, M. Modelling Web Applications, Chapter 2, In *StateWebCharts: a Formal Notation for Navigation Modelling of Web Applications*, PhD Thesis, University of Toulouse 1, France, 2 April, 2004.

[Winckler 07] Winckler, M.; Xiong, J.; Noirhomme-Fraiture, M. Accessibility Legislation and Codes of Practice: an Accessibility Study of Web Sites of French and Belgium Local Administrations. In *Proceedings of 1st International Workshop on Design & Evaluation of e-Government Applications and Services*, *DEGAS'07*, held in conjunction with IFIP TC13 INTERACT'2007, Rio de Janeiro, Brazil, September 11th, 2007.

[Xiong 06a] Xiong, J.; Diouf, M.; Farenc, C.; Winckler, M. Automating Guidelines Inspection: From Web Site Specification to Deployment. In *Proceedings of 6th International Conference on Computer-Aided Design of User Interfaces*, *CADUI 2006*, Bucharest, Romania, June 5-8, 2006.

[Xiong 06b] Xiong, J.; Farenc, C.; Winckler, M. Extending Automated Guidelines Inspection with an Ontology of User Interface of Web Applications, In *Proceedings of 2nd International Workshop on Automated Specification and Verification of Web Systems*, *WWV 2006*, Poster Session, Cyprus, November 15-19, 2006.

[Xiong 07] Xiong, J.; Farenc, C.; Winckler, M. Analyzing Tool Support for Inspecting Accessibility Guidelines during the Development Process of Web Sites, In *Proceedings of 1st International Workshop on Web Usability and Accessibility*, *IWWUA 2007*, Nancy, France, December 3-7, 2007, Springer LNCS 4832, pp. 470-480.

[Xiong 08a] Xiong, J.; Farenc, C.; Winckler, M. Towards an Ontology-based Approach for Dealing with Web Guidelines, to be published in *Proceedings of 2nd International Workshop on Web Usability and Accessibility, IWWUA 2008*, September 1-4, 2008, Auckland, New Zealand.

[Xiong 08b] Xiong, J.; Winckler, M. An Investigation of Tool Support for Accessibility Assessment Throughout the Development Process of Web Sites, to be published in Journal of Web Engineering, vol. 7, n°4, 2008.

Liste des figures

Liste des tableaux

Annexe A. Description détaillée de l'ontologie

1. Site

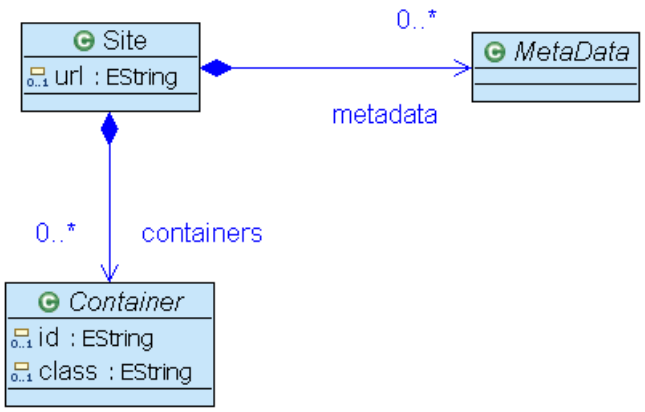


Figure 53 – Class diagram for the *Site* element

A *Site* element represents a Web application. It is composed of metadata elements (e.g. author name, the support contact, etc.) and of containers (i.e. Web pages).

Attribute name	Attribute type	Description
url	string	The URL of the Web application.
containers	List(Container)	The list of containers (a set of Web pages and/or windows).
metadata	List(MetaData)	The set of metadata attached to this site.

2. Container

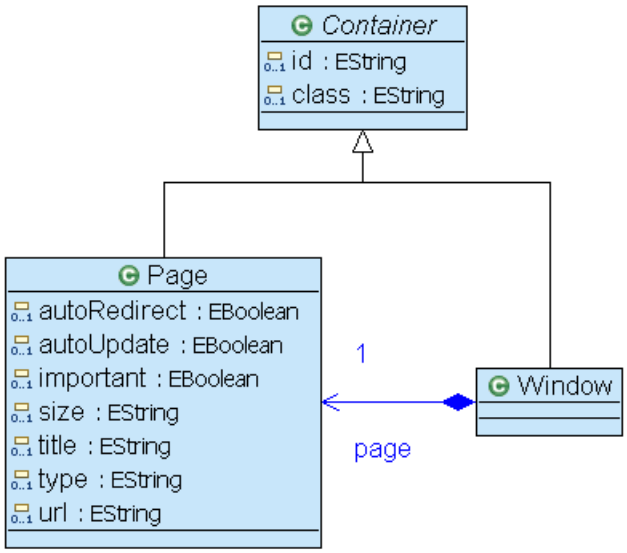


Figure 54 – Class diagram for the *Container* element

A *Container* element represents an abstract element that can hold Web contents. Concrete containers are *Page* and *Window*.

Attribute name	Attribute type	Description
id	string	The identifier. It is unique and can be used for several purposes such as for styling effects on this element.
class	string	The class. The name of this class can be shared with other elements. It can be used for several purposes such as for styling effects on this element.

2.1. WINDOW

A *Window* element represents a container for one Web page (i.e. a *Page* element). Pages that are opened in a new window (such as pop-ups) are embedded in a *Window* Container.

Attribute name	Attribute type	Description
page	Page	The Web page contained in this window

3. Page

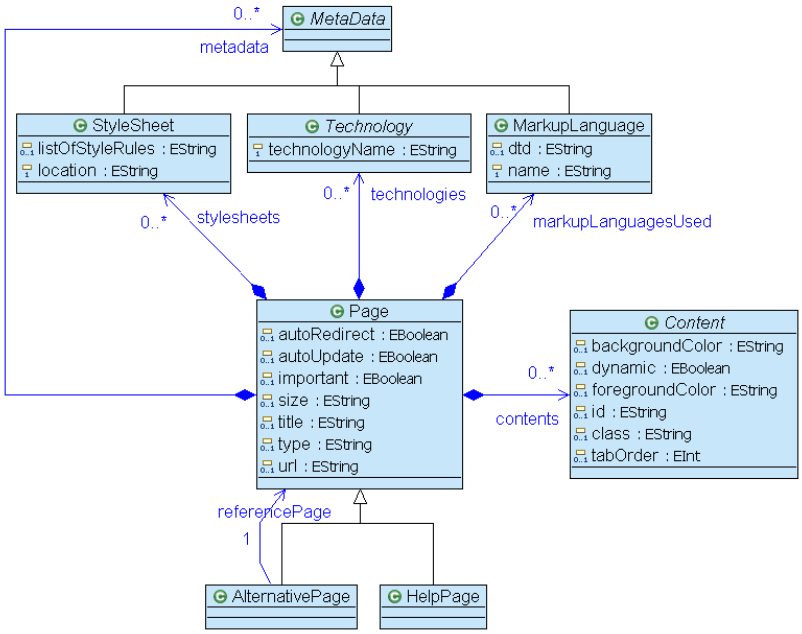


Figure 55 – Class diagram for the *Page* element

A *Page* element represents a simple Web page. More specialized pages can be alternative pages or help pages.

One Web page can mention the technologies and/or markup languages used. Besides, style sheets, metadata elements and contents can be attached to this page.

Attribute name	Attribute type	Description
autoRedirect	boolean	Indicates if this page automatically redirects the user to another page.
autoUpdate	boolean	Indicates if this page performs an automatic update (or refresh) at specific intervals.
important	boolean	Indicates if this page is important or not.
size	string	The size of this page (e.g. in Kb).
title	string	The title of this page.
type	string	The type of this page, i.e. the role this page plays in the Web site (e.g. “homepage”).
url	string	The URL of this page.
contents	List(Content)	The list of contents of this page.
markupLanguagesUsed	List(MarkupLanguage)	The list of markup languages used in this page.
metadata	List(MetaData)	The list of metadata attached to this page.
stylesheets	List(StyleSheet)	The list of style sheets attached to this page.
technologies	List(Technology)	The list of technologies used in this page.

3.1. ALTERNATIVEPAGE

An *AlternativePage* element represents a page that contains the same elements as a “reference page”. The only difference is that the alternative page is a version of the reference

page that should be accessible whatever the modality used. Using alternative pages helps ensuring users can always access the information.

Attribute name	Attribute type	Description
referencePage	Page	The reference page for this alternative page.

3.2. HELPPAGE

A *HelpPage* element represents a page that contains help information about the Web site, the navigation, etc.

4. MetaData

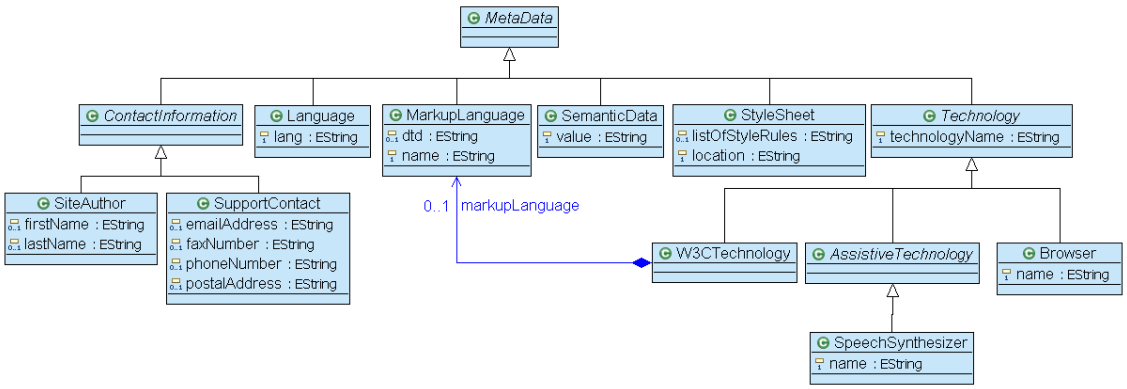


Figure 56 – Class diagram for the *MetaData* element

A *Metadata* element represents data about a Web site or a Web page. It can be the contact information, the language used, a markup language, semantic data, a style sheet, or a technology used to view the Web site.

4.1. CONTACTINFORMATION

A *ContactInformation* element represents a metadata intended to give the user a means to reach the contact person of a Web site. It can be the site author information or the support contact information.

4.2. SITEAUTHOR

A *SiteAuthor* element represents information about the Web site's author.

Attribute name	Attribute type	Description
firstName	string	The first name of the author.
lastName	string	The last name of the author.

4.3. SUPPORTCONTACT

A *SupportContact* element represents information about the support contact for the Web site.

Attribute name	Attribute type	Description
emailAddress	string	The e-mail address of the support contact.
faxNumber	string	The fax number of the support contact.
phoneNumber	string	The phone number of the support contact.
postalAddress	string	The postal address of the support contact.

4.4. LANGUAGE

A *Language* element represents the natural language used in the Web site/Web page.

Attribute name	Attribute type	Description
lang	string	The string representing the natural language used (e.g. “en” for English, or “fr” for French).

4.5. MARKUPLANGUAGE

A *MarkupLanguage* element represents a language that structures contents with the use of surrounding tags. Each of these tags provides a semantic to the content of the tag. Examples or markup languages are HTML[33] or XML[34].

Attribute name	Attribute type	Description
dtd	String	The Data Type Definition (DTD) of this markup language.
name	String	The name of this markup language.

4.6. SEMANTICDATA

A *SemanticData* element represents information that will provide additional semantics to its associated element (e.g. a Web page, an image, etc.). One can assimilate it to a tag that decorates an element. It could be for example a tag labeled “Home Page” to designate a page as the home page of the Web site. This semantic data is human-readable only.

Attribute name	Attribute type	Description
value	string	The human-readable value of the semantic data.

[33] HyperText Markup Language, specifications available at http://www.w3.org/TR/html4/
[34] eXtensible Markup Language, specifications available at http://www.w3.org/TR/xml11/

4.7. STYLESHEET

A *StyleSheet* element represents a list of style rules that is applied to a Web page for presentation purposes.

Attribute name	Attribute type	Description
listOfStyleRules	string	The list of style rules.
location	string	The location of the style sheet.

4.8. TECHNOLOGY

A *Technology* element represents a technology used in the Web domain. It can be a W3C technology, an assistive technology, or a browser.

Attribute name	Attribute type	Description
technologyName	string	The name of this technology.

4.9. W3CTECHNOLOGY

A *W3CTechnology* element represents a technology recommended by the World Wide Web Consortium[35] (W3C). A W3C technology can have a reference to a markup language.

Attribute name	Attribute type	Description
markupLanguage	MarkupLanguage	The referenced markup language if this W3C technology is a markup language.

4.10.ASSISTIVETECHNOLOGY

An *AssistiveTechnology* element represents any device or technology that helps a disabled person when navigating through the Web application.

4.11.SPEECHSYNTHESIZER

A *SpeechSynthesizer* element represents a device that can “read” any computer text aloud.

Attribute name	Attribute type	Description
name	string	The name of this speech synthesizer.

4.12.BROWSER

A *Browser* element represents an application used to view Web pages.

[35] http://www.w3.org/

Attribute name	Attribute type	Description
name	string	The name of this browser.

5. Content

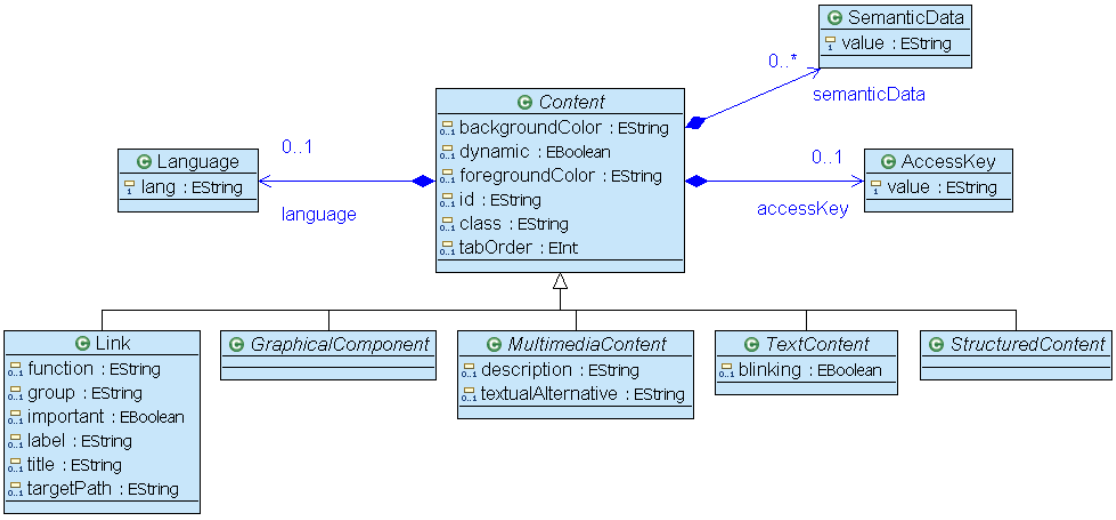


Figure 57 – Class diagram for the *Content* element

A *Content* element represents any content that can be included in a Web page. It can denote a link, a graphical component, a multimedia content, a textual content or a structured content. Furthermore, information about the natural language, the access key (in order to directly reach this content element), and semantic data can be supplied for each content element.

Attribute name	Attribute type	Description
backgroundColor	string	The background color given by either its hexadecimal code, its RGB code, or its name (when it exists).
dynamic	boolean	Indicates if this content element is dynamic (i.e. is dynamically generated and/or dynamically changes).
foregroundColor	string	The foreground color given by either its hexadecimal code, its RGB code, or its name (when it exists).
id	string	The identifier. It is unique and can be used for several purposes such as for styling effects on this element.
class	string	The class. The name of this class can be shared with other elements. It can be used for several purposes such as for styling effects on this element.
tabOrder	int	The tabulation order when navigating by the means of the keyboard.
language	Language	The language used for this content element (when relevant).
accessKey	AccessKey	The access key used to directly reach this content element.
semanticData	List(SemanticData)	The list of semantic data associated to this content element.

5.1. LINK

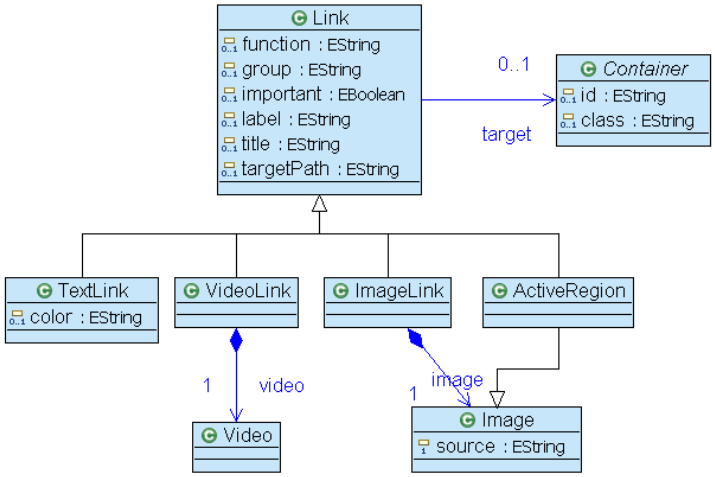


Figure 58 – Class diagram for the *Link* element

A *Link* element represents a connection between two containers allowing the user to move from the current container (the source) to the connected container (the target). A link can be a textual link, an image link, a video link, or an active region of an image map. A link has a reference to its target container. The source container of the link is supposed to be its parent container.

Attribute name	Attribute type	Description
function	string	The function of this link (e.g. “Move to the next page”).
group	string	The name of the group this link belongs to (e.g. “Navigation links”).
important	boolean	Indicates if this link is important or not.
label	string	The label of this link.
title	string	The title of this link (i.e. additional supplied information provided to complement the information given by the label).
targetPath	string	The path to the target of this link, generally an URL.
target	Container	The target container.

5.1.1. TextLink

A *TextLink* element represents a textual link. Clicking on the text of this textual link leads to moving to the target container.

Attribute name	Attribute type	Description
color	string	The color of the text given by either its hexadecimal code, its RGB code, or its name (when it exists).

5.1.2. ImageLink

An *ImageLink* element represents an image link. Clicking on the image leads to moving to the target container. The image link has a reference to its image.

Attribute name	Attribute type	Description
image	Image	The image of this image link.
contents	List(Content)	The list of contents of this image link.

5.1.3. VideoLink

A *VideoLink* element represents a video link. Clicking on the video leads to moving to the target container.

Attribute name	Attribute type	Description
video	Video	The video of this video link.

5.1.4. ActiveRegion

An *ActiveRegion* is a reactive and clickable region of an image map (see element *ImageMap* in section 5.3.7). An active region is both a link and an image. Clicking on the active region leads to moving to the target container.

5.2. GRAPHICALCOMPONENT

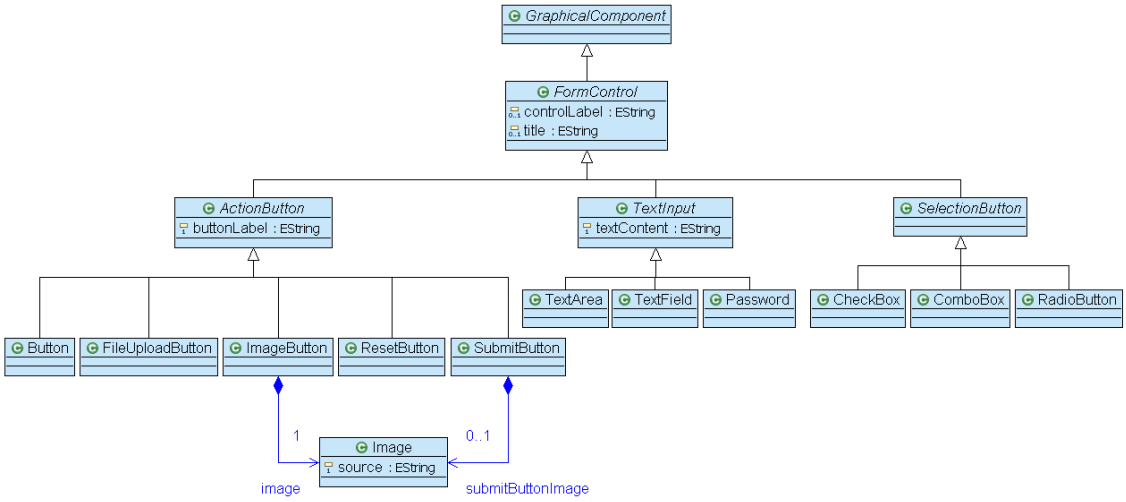


Figure 59 – Class diagram for the *GraphicalComponent* element

A *GraphicalComponent* element represents any element composing a graphical user interface.

5.2.1. FormControl

A *FormControl* element represents a control (or field) contained within a form. A form control can be an action button, a text input, or a selection button.

Such form controls allow a user to interact with the Web application by sending data to the Web server. The server then handles the user request and sends the result to the user in response.

Attribute name	Attribute type	Description
controlLabel	string	The label associated with this form control.
title	string	The title of this form control (i.e. additional supplied information provided to complement the information given by the control label).

5.2.2. ActionButton

An *ActionButton* element represents a form button that performs an action when clicked. An action button can be a simple button, an image button, a file upload button, a reset button or a submit button.

Attribute name	Attribute type	Description
buttonLabel	string	The label displayed on the button.

5.2.3. Button

A *Button* element represents a simple form button.

5.2.4. FileUploadButton

A *FileUploadButton* element represents a form control that may let the user include one or more files into the form submission.

5.2.5. ImageButton

An *ImageButton* element represents an image button in a form (the image plays the same role as a button).

Attribute name	Attribute type	Description
image	Image	The image associated to this image button.

5.2.6. ResetButton

A *ResetButton* element represents a reset button, i.e. a button that resets each field in a form.

5.2.7. SubmitButton

A *SubmitButton* element represents a submit button, i.e. a button that sends the filled form to the Web server when clicked.

Attribute name	Attribute type	Description
submitButtonImage	Image	The image associated to this submit button if this button appears as a clickable image button.

5.2.8. TextInput

A *TextInput* element represents an input control in a form. A text input control can be a text area, a text field, or a password field. Each of these controls allows filling the form with texts.

Attribute name	Attribute type	Description
textContent	string	The text content in the body of this text input control.

5.2.9. TextArea

A *TextArea* element represents a text area control in which a user can enter multi-lines texts.

5.2.10. TextField

A *TextField* element represents a text field in which a user can enter single-line texts.

5.2.11. Password

A *Password* element represents a password field in which a user can enter a hidden password. In general, each character the user enters is displayed with the same symbol (most of the time, this symbol is '*') so as to hide this entered character.

5.2.12. SelectionButton

A *SelectionButton* element represents a form control used to select or unselect an item in a form. It can be a checkbox button or a radio button.

5.2.13. CheckBox

A *CheckBox* element represents a selection button used to select or unselect an item in a form.

5.2.14. ComboBox

A *ComboBox* element represents a selection list in which the user can select a single item.

5.2.15. RadioButton

A *RadioButton* represents a selection button used to select an item in a set of items. The particularity of radio buttons is that the user can only select one item for a same group of choices (for instance, for the 'Gender' group in a form, one has to choose either 'Male' of 'Female' but not both).

5.3. MULTIMEDIACONTENT

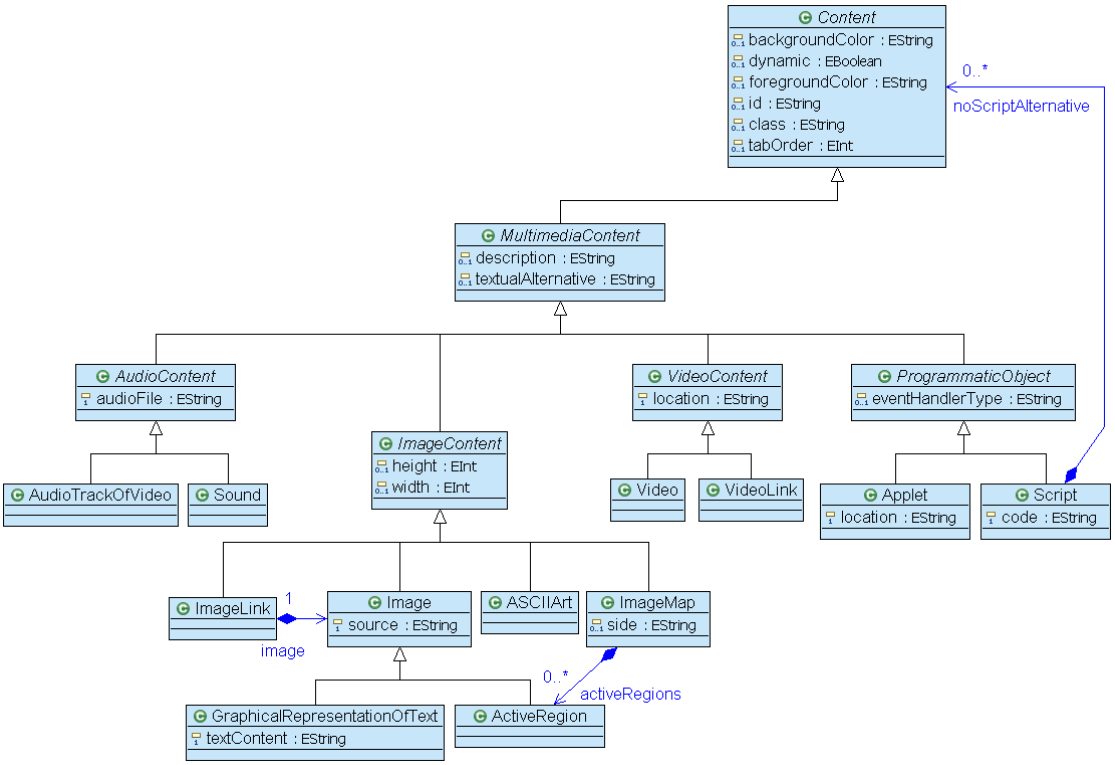


Figure 60 – Class diagram for the *MultimediaContent* element

A *MultimediaContent* element represents an abstract multimedia content. It can be an audio content, an image content, a video content, or a programmatic object.

Attribute name	Attribute type	Description
description	string	The description associated with this multimedia content. The purpose of this description is to give supplementary information about this content in the case where it is not accessible.
textualAlternative	string	The textual alternative of this content element (when relevant).

5.3.1. AudioContent

An *AudioContent* element represents an abstract audio content. It can be a simple sound or an audio track of video.

Attribute name	Attribute type	Description
audioFile	string	The location of the audio file.

5.3.2. Sound

A *Sound* element represents a simple sound.

5.3.3. AudioTrackOfVideo

An *AudioTrackOfVideo* element represents an audio track of a video. This audio content is to be used in conjunction with a video.

5.3.4. ImageContent

An *ImageContent* element represents a multimedia content of type Image. It can be an image, an image link, an image map, or ASCII art.

Attribute name	Attribute type	Description
height	integer	The height of the image (in pixel).
width	integer	The width of the image (in pixel).

5.3.5. Image

An *Image* element represents a simple image. An image can be a graphical representation of text.

Attribute name	Attribute type	Description
source	string	The location of the image file.

5.3.6. GraphicalRepresentationOfText

A *GraphicalRepresentationOfText* represents an image that reproduces a text. Most of the times, such images are used to represent a text with a specific font, styling, 3D effects, etc. that cannot be reproduced with style sheets.

Attribute name	Attribute type	Description
textContent	string	The text content that is represented by this image.

5.3.7. ASCIIArt

An *ASCIIArt* element refers to text characters and symbols that are combined to create an image. More information can be found at http://www.w3.org/TR/WCAG10/#ascii-art.

5.3.8. ImageMap

An *ImageMap* element represents an image segmented into clickable regions (*ActiveRegion* elements, see section 5.1.4). Each region is specified by the image map and is a link to a container.

Attribute name	Attribute type	Description
side	string	Indicates if this image map is server-side or client-side. Possible values are SERVER_SIDE or CLIENT_SIDE.
activeRegions	List(ActiveRegion)	The list of active regions.

5.3.9. VideoContent

A *VideoContent* element represents a multimedia content of type video. It can be a simple video or a video link.

Attribute name	Attribute type	Description
location	string	The location of the video file.

5.3.10. Video

A *Video* element represents a simple video.

5.3.11. ProgrammaticObject

A *ProgrammaticObject* element represents a programmatic object. It can be an applet, or a script.

Attribute name	Attribute type	Description
eventHandlerType	string	The type of the event handler. Possible values are LOGICAL or INTERFACE_DEPENDENT

5.3.12. Applet

An *Applet* element represents a Java applet[36].

Attribute name	Attribute type	Description
location	string	The location of the applet.

5.3.13. Script

A *Script* element represents a script (e.g. a block of Javascript code). An alternative version of this script can be provided via the *noScriptAlternative* attribute in order to comply with accessibility requirements.

Attribute name	Attribute type	Description
code	string	The code of the script.
noScriptAlternative	List(Content)	The alternative version of this script.

[36] More information about Java applets can be found at http://java.sun.com/applets/

5.4. TEXTCONTENT

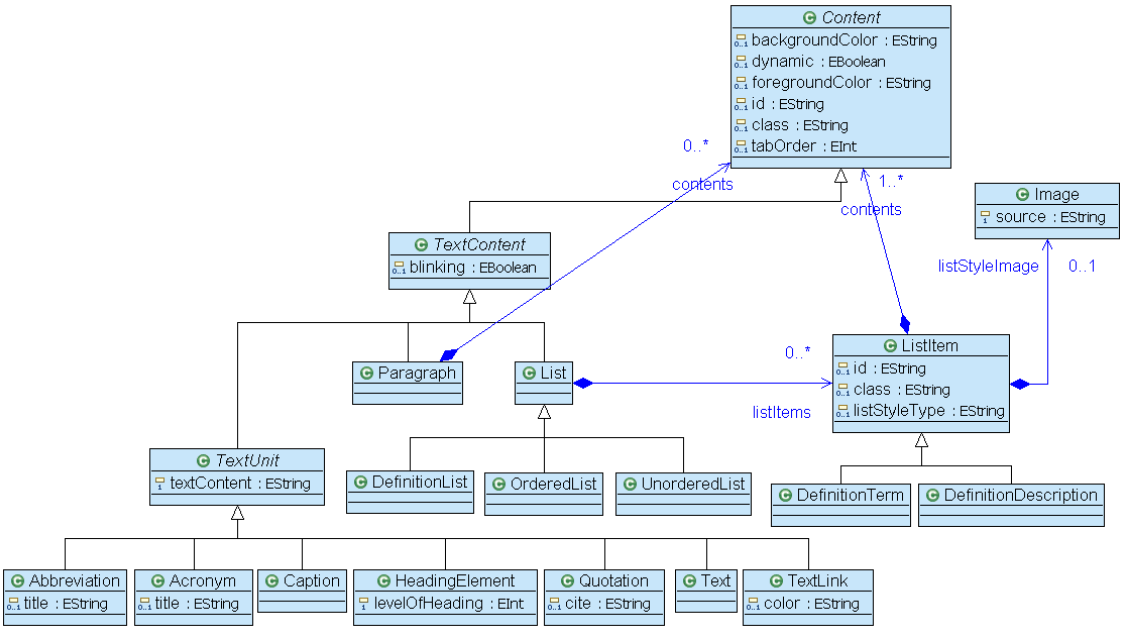


Figure 61 – Class diagram for the *TextContent* element

A *TextContent* element represents an abstract textual content. It can be a simple text unit, a paragraph, or a list.

Attribute name	Attribute type	Description
blinking	boolean	Indicates if this textual content is blinking or not.

5.4.1. TextUnit

A *TextUnit* element represents a simple text unit. It can be an abbreviation, an acronym, a caption, a heading element, a quotation, a text, or a textual link.

Attribute name	Attribute type	Description
textContent	string	The text content of this text unit.

5.4.2. Abbreviation

An *Abbreviation* element represents an abbreviation.

Attribute name	Attribute type	Description
title	string	The full text for this abbreviation.

5.4.3. Acronym

An *Acronym* element represents an acronym.

Attribute name	Attribute type	Description
title	string	The full text for this acronym.

5.4.4. Caption

A *Caption* element represents a caption for a table.

5.4.5. HeadingElement

A *HeadingElement* element represents a heading element for a section of the Web page. Actually, it briefly describes the topic of the section it introduces. Using heading elements allows hierarchically structuring a Web page.

Attribute name	Attribute type	Description
levelOfHeading	integer	The level of heading. Values range from 1 (the highest level) to 6 (the lowest level).

5.4.6. Quotation

A *Quotation* element represents a quotation.

Attribute name	Attribute type	Description
cite	string	The URL of the quote, if it is taken from the Web.

5.4.7. Text

A *Text* element represents a simple text.

5.4.8. Paragraph

A *Paragraph* element represents a paragraph. It can contain any kind of content.

Attribute name	Attribute type	Description
contents	List(Content)	The list of contents in this paragraph.

5.4.9. List

A *List* element represents a list. It can be a definition list, an ordered list, or an unordered list. A list contains several list items.

Attribute name	Attribute type	Description
listItems	List(ListItem)	The list of items.

5.4.10. DefinitionList

A *DefinitionList* element represents a list of terms followed by their respective definition. Thus, a definition list contains a collection of (*definition term*, *definition description*) items.

5.4.11. OrderedList

An *OrderedList* element represents an ordered list.

5.4.12. UnorderedList

An *UnorderedList* element represents an unordered list.

5.4.13. ListItem

A *ListItem* element represents a list item.

Attribute name	Attribute type	Description
id	string	The identifier. It is unique and can be used for several purposes such as for styling effects on this element.
class	string	The class. The name of this class can be shared with other elements. It can be used for several purposes such as for styling effects on this element.
listStyleType	string	The type of bullets for this list. Possible values are DISC, CIRCLE, SQUARE, DECIMAL, LOWER_ROMAN, UPPER_ROMAN, LOWER_ALPHA, UPPER_ALPHA, or NONE.
listStyleImage	Image	The image representing the bullets.
contents	List(Content)	The list of contents in this list item.

5.4.14. DefinitionTerm

A *DefinitionTerm* element represents a term to be defined. A definition term is a possible item of a definition list. It is to be used in conjunction with a definition description item (see section 5.4.15). This last one gives the description of the term.

5.4.15. DefinitionDescription

A *DefinitionDescription* element represents a description for a term in a definition list. It is to be used in conjunction with a definition term item (see section 5.4.14).

5.5. STRUCTUREDCONTENT

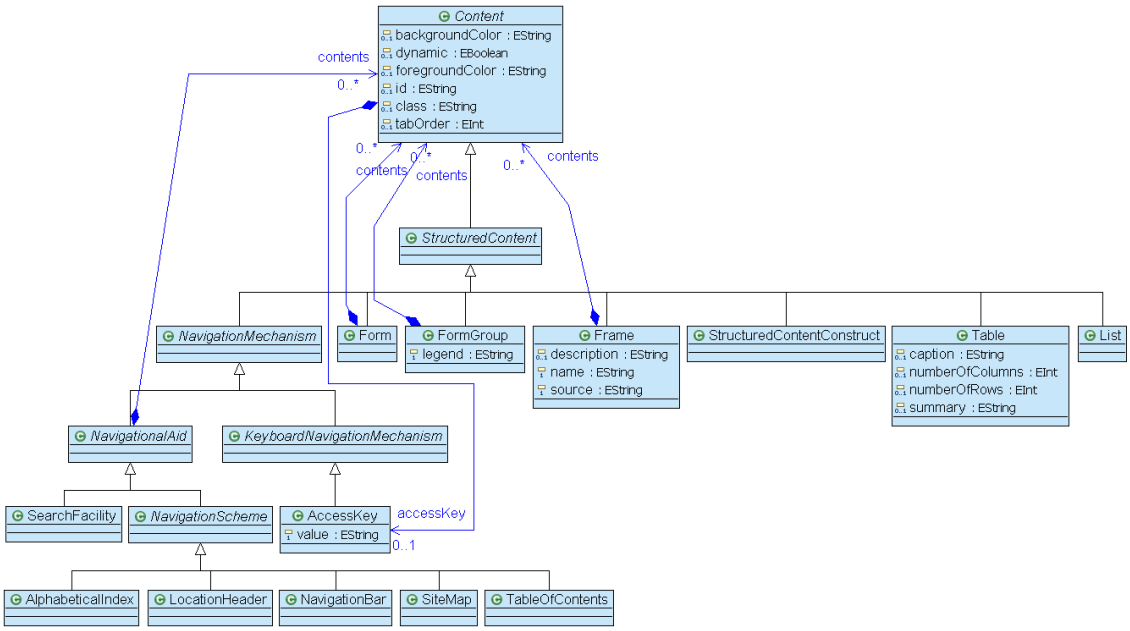


Figure 62 – Class diagram for the *StructuredContent* element

A *StructuredContent* element represents a structured content, i.e. a content which can hold other contents following a certain structure. It can be a navigation mechanism, a form, a form group, a frame, a table, or a list.

5.5.1. NavigationMechanism

A *NavigationMechanism* element represents any mechanisms dedicated to navigation. It can be a navigational aid or a keyboard navigation mechanism.

5.5.2. NavigationalAid

A *NavigationalAid* element represents a complex structure inserted into a Web page dedicated to navigation. It can be a search facility or a navigation scheme. A navigational aid is composed of many contents defining the navigation paths (e.g. a list of links).

Attribute name	Attribute type	Description
contents	List(Content)	The list of contents in this navigational aid.

5.5.3. SearchFacility

A *SearchFacility* element represents any kind of search facilities. The aim of such navigational aids is to provide the user with a quick search feature. First, a query is given by the user, and subsequently a search engine returns the corresponding result (most of the time the search is done in the scope of the Web site).

5.5.4. NavigationScheme

A *NavigationScheme* element represents any navigational aids in which the navigation items are organized (e.g. by theme, by categories, etc.). A navigation scheme can be an alphabetical index, a location header, a navigation bar, a site map, or a table of contents.

5.5.5. AlphabeticalIndex

An *AlphabeticalIndex* element represents a navigational aid in which navigation items are organized in the alphabetical order.

5.5.6. LocationHeader

A *LocationHeader* element represents a navigational aid that shows the path of the current Web page. Thus, a location header reveals the structure of the Web site.

5.5.7. NavigationBar

A *NavigationBar* element represents a navigational aid that contains links to the most representative categories (or the most visited categories such as "what's new", "contact", or "faq" categories).

5.5.8. SiteMap

A *SiteMap* element represents a navigational aid that visually describes the whole map of the Web Site.

5.5.9. TableOfContents

A *TableOfContents* element represents a table of contents for the Web site. This navigational aid shows the different sections of Web site in a linear manner (even though the user can navigate between the different sections in a non-linear order).

5.5.10. KeyboardNavigationMechanism

A *KeyboardNavigationMechanism* element represents a navigation mechanism with the use of the keyboard.

5.5.11. AccessKey

An *AccessKey* element represents a keyboard navigation mechanism that brings the user to a target element in the Web page when activating a certain character: the access key[37].

5.5.12. Form

A *Form* element represents a form, i.e. a structure containing fields intended to receive user input data. A form generally contains form controls (i.e. fields) but actually, it can contain any kind of contents (images, text, tables, etc.). Form controls can be grouped within form groups.

One may be surprised that there's no explicit composition relationships between *Form* and *FormGroup* in the *StructuredContent* diagram class. In fact, this relationship is deduced from the relationship between *Form* and *Content*.

Attribute name	Attribute type	Description
contents	List(Content)	The list of contents in this form.

[37] The activation of the access key depends on the browser used: with Firefox the key sequence is *ALT + character* while on Microsoft Internet Explorer the key sequence is *ALT + character + Enter*.

5.5.13. FormGroup

A *FormGroup* element represents group of form controls in a form. It allows grouping elements that are semantically related.

Attribute name	Attribute type	Description
legend	string	The legend (or title) of the form group.
contents	List(Content)	The list of contents in this form group.

5.5.14. Frame

A *Frame* element represents an area in a Web page. A frame can be seen as a Web page contained in a Web page.

Attribute name	Attribute type	Description
description	string	The description associated with this multimedia content. The purpose of this description is to give supplementary information about this content in the case where it is not accessible.
name	string	The name of this frame.
source	string	The location of this frame.
contents	List(Content)	The list of contents of this frame.

5.5.15. StructuredContentConstruct

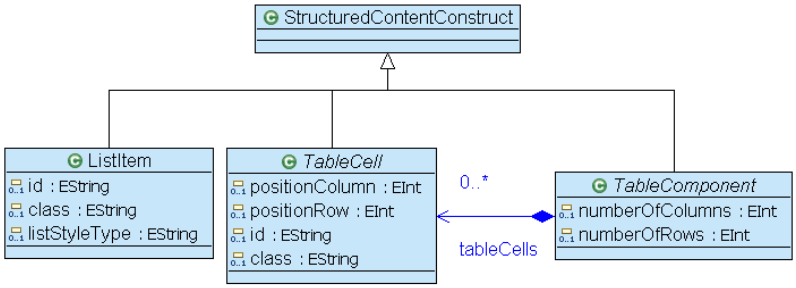


Figure 63 – Class diagram for the *StructuredContentConstruct* element

A *StructuredContentConstruct* element represents any construct for structured content such as tables or lists. It can be a table component, a table cell, or a list item.

5.5.16. Table

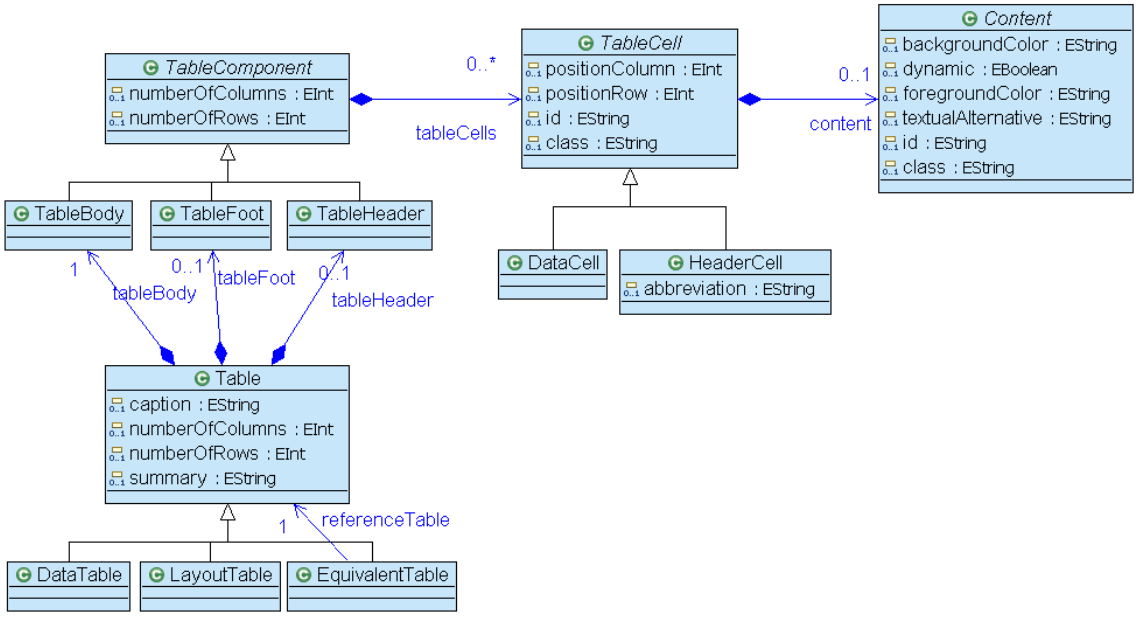


Figure 64 – Class diagram for the *Table* element

A *Table* element represents a structured content that arranges contents within table cells. Each table cell can be identified by its row and its column. A table can be a data table, a layout table or an equivalent table. A table is composed of a header, a body, and a foot.

Attribute name	Attribute type	Description
caption	string	The caption of this table.
numberOfColumns	integer	The number of columns of this table.
numberOfRows	integer	The number of rows of this table.
summary	string	The summary for this table.
tableHeader	TableHeader	The table header.
tableBody	TableBody	The table body.
tableFoot	TableFoot	The table foot.

5.5.17. DataTable

A *DataTable* element represents a table intended to present data in a tabular form.

5.5.18. LayoutTable

A *LayoutTable* element represents a table used for presentation layout. This kind of tables helps managing the presentation of the page, each cell containing a presentation element: an image, the title of the page, the main area of the page, etc.

5.5.19. EquivalentTable

An *EquivalentTable* element represents a table that contains the same elements as a “reference table”. The only difference is that the equivalent table is a version of the reference table that should be accessible whatever the modality used. Using equivalent table helps ensuring users can always access the information.

Attribute name	Attribute type	Description
referenceTable	Table	The reference table for this equivalent table.

5.5.20. TableComponent

A *TableComponent* element represents a part of one table. It can be a table header, a table body, or a table foot.

Attribute name	Attribute type	Description
numberOfColumns	integer	The number of columns of this table component.
numberOfRows	integer	The number of rows of this table component.
tableCells	List(TableCell)	The list of cells for this table component.

5.5.21. TableHeader

A *TableHeader* element represents the header part of a table.

5.5.22. TableBody

A *TableBody* element represents the body part of a table.

5.5.23. TableFoot

A *TableFoot* element represents the foot part of a table.

5.5.24. TableCell

A *TableCell* element represents a cell contained in a table. It can contain any possible content.

Attribute name	Attribute type	Description
positionColumn	integer	The column number of this table cell.
positionRow	integer	The row number of this table cell.
id	string	The identifier. It is unique and can be used for several purposes such as for styling effects on this element.
class	string	The class. The name of this class can be shared with other elements. It can be used for several purposes such as for styling effects on this element.
content	Content	The content of this table cell.

5.5.25. DataCell

A *DataCell* element represents a table cell intended to contain any kind of data.

5.5.26. HeaderCell

A *HeaderCell* element represents a table cell intended to play the role of a header for other cells. A header cell is typically contained within the header part of a table (element *TableHeader*).

Attribute name	Attribute type	Description
abbreviation	string	The abbreviation form of the text contained within this header cell.

Annexe B. Description OWL de l’ontologie

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns="http://www.owl-ontologies.com/unnamed.owl#"
  xml:base="http://www.owl-ontologies.com/unnamed.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="ImageContent">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="MultimediaContent"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Acronym">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="TextUnit"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="CheckBox">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="SelectionButton"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="DefinitionDescription">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="ListItem"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="TextContent">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Content"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#ListItem">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="StructuredContentConstruct"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="AudioContent">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#MultimediaContent"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Video">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="VideoContent"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="ImageMap">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:ID="side"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >SERVER_SIDE</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:FunctionalProperty rdf:about="#side"/>
        </owl:onProperty>
        <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >CLIENT_SIDE</owl:hasValue>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#ImageContent"/>
  </owl:Class>
  <owl:Class rdf:ID="Paragraph">
    <rdfs:subClassOf rdf:resource="#TextContent"/>
```

```
</owl:Class>
<owl:Class rdf:ID="Frame">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="StructuredContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="UnorderedList">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="List"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ImageLink">
  <rdfs:subClassOf rdf:resource="#ImageContent"/>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Link"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FormGroup">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StructuredContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Script">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="ProgrammaticObject"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="MarkupLanguage">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="MetaData"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TextLink">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextUnit"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Link"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NavigationMechanism">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StructuredContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="AlternativePage">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Page"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="KeyboardNavigationMechanism">
  <rdfs:subClassOf rdf:resource="#NavigationMechanism"/>
</owl:Class>
<owl:Class rdf:about="#ProgrammaticObject">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#MultimediaContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TextArea">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TextInput"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#VideoContent">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#MultimediaContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DataCell">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TableCell"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Abbreviation">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextUnit"/>
  </rdfs:subClassOf>
```

```
</owl:Class>
<owl:Class rdf:ID="Container"/>
<owl:Class rdf:ID="TextField">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextInput"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TableHeader">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="TableComponent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="StyleSheet">
  <rdfs:subClassOf rdf:resource="#MetaData"/>
</owl:Class>
<owl:Class rdf:about="#StructuredContentConstruct">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StructuredContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SearchFacility">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="NavigationalAid"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ASCIIArt">
  <rdfs:subClassOf rdf:resource="#ImageContent"/>
</owl:Class>
<owl:Class rdf:ID="Site"/>
<owl:Class rdf:ID="Language">
  <rdfs:subClassOf rdf:resource="#MetaData"/>
</owl:Class>
<owl:Class rdf:ID="ResetButton">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="ActionButton"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#SelectionButton">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="FormControl"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SupportContact">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="ContactInformation"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="AlphabeticalIndex">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="NavigationScheme"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DefinitionTerm">
  <rdfs:subClassOf rdf:resource="#ListItem"/>
</owl:Class>
<owl:Class rdf:ID="HeaderCell">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TableCell"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#NavigationalAid">
  <rdfs:subClassOf rdf:resource="#NavigationMechanism"/>
</owl:Class>
<owl:Class rdf:ID="GraphicalRepresentationOfText">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Image"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Sound">
  <rdfs:subClassOf rdf:resource="#AudioContent"/>
</owl:Class>
<owl:Class rdf:about="#Image">
  <rdfs:subClassOf rdf:resource="#ImageContent"/>
</owl:Class>
<owl:Class rdf:ID="VideoLink">
  <rdfs:subClassOf rdf:resource="#VideoContent"/>
  <rdfs:subClassOf>
```

```
    <owl:Class rdf:about="#Link"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="DefinitionList">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#List"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="EquivalentTable">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Table"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="LocationHeader">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NavigationScheme"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ImageButton">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ActionButton"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SemanticData">
  <rdfs:subClassOf rdf:resource="#MetaData"/>
</owl:Class>
<owl:Class rdf:ID="AssistiveTechnology">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Technology"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#ContactInformation">
  <rdfs:subClassOf rdf:resource="#MetaData"/>
</owl:Class>
<owl:Class rdf:ID="HeadingElement">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextUnit"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="NavigationBar">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NavigationScheme"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Button">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ActionButton"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#MultimediaContent">
  <rdfs:subClassOf rdf:resource="#Content"/>
</owl:Class>
<owl:Class rdf:ID="Form">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#StructuredContent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#TableCell">
  <rdfs:subClassOf rdf:resource="#StructuredContentConstruct"/>
</owl:Class>
<owl:Class rdf:ID="GraphicalComponent">
  <rdfs:subClassOf rdf:resource="#Content"/>
</owl:Class>
<owl:Class rdf:about="#StructuredContent">
  <rdfs:subClassOf rdf:resource="#Content"/>
</owl:Class>
<owl:Class rdf:ID="SpeechSynthetizer">
  <rdfs:subClassOf rdf:resource="#AssistiveTechnology"/>
</owl:Class>
<owl:Class rdf:about="#TextInput">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#FormControl"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="W3CTechnology">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Technology"/>
```

```
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TableBody">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TableComponent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="HelpPage">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Page"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TableFoot">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TableComponent"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Link">
  <rdfs:subClassOf rdf:resource="#Content"/>
</owl:Class>
<owl:Class rdf:ID="Text">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextUnit"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Caption">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#TextUnit"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Window">
  <rdfs:subClassOf rdf:resource="#Container"/>
</owl:Class>
<owl:Class rdf:ID="Password">
  <rdfs:subClassOf rdf:resource="#TextInput"/>
</owl:Class>
<owl:Class rdf:about="#List">
  <rdfs:subClassOf rdf:resource="#TextContent"/>
</owl:Class>
<owl:Class rdf:ID="ActiveRegion">
  <rdfs:subClassOf rdf:resource="#Image"/>
  <rdfs:subClassOf rdf:resource="#Link"/>
</owl:Class>
<owl:Class rdf:ID="DataTable">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Table"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="RadioButton">
  <rdfs:subClassOf rdf:resource="#SelectionButton"/>
</owl:Class>
<owl:Class rdf:about="#Page">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >HOMEPAGE</owl:hasValue>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="type"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#type"/>
      </owl:onProperty>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >NORMAL</owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >NAVIGATION</owl:hasValue>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#type"/>
      </owl:onProperty>
```

```
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Container"/>
</owl:Class>
<owl:Class rdf:ID="Browser">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Technology"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="SiteAuthor">
  <rdfs:subClassOf rdf:resource="#ContactInformation"/>
</owl:Class>
<owl:Class rdf:ID="SiteMap">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NavigationScheme"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#FormControl">
  <rdfs:subClassOf rdf:resource="#GraphicalComponent"/>
</owl:Class>
<owl:Class rdf:ID="LayoutTable">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Table"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="TableOfContents">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#NavigationScheme"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#TableComponent">
  <rdfs:subClassOf rdf:resource="#StructuredContentConstruct"/>
</owl:Class>
<owl:Class rdf:ID="Applet">
  <rdfs:subClassOf rdf:resource="#ProgrammaticObject"/>
</owl:Class>
<owl:Class rdf:ID="SubmitButton">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#ActionButton"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="OrderedList">
  <rdfs:subClassOf rdf:resource="#List"/>
</owl:Class>
<owl:Class rdf:about="#NavigationScheme">
  <rdfs:subClassOf rdf:resource="#NavigationalAid"/>
</owl:Class>
<owl:Class rdf:about="#TextUnit">
  <rdfs:subClassOf rdf:resource="#TextContent"/>
</owl:Class>
<owl:Class rdf:about="#Table">
  <rdfs:subClassOf rdf:resource="#StructuredContent"/>
</owl:Class>
<owl:Class rdf:ID="ComboBox">
  <rdfs:subClassOf rdf:resource="#SelectionButton"/>
</owl:Class>
<owl:Class rdf:ID="AccessKey">
  <rdfs:subClassOf rdf:resource="#KeyboardNavigationMechanism"/>
</owl:Class>
<owl:Class rdf:about="#ActionButton">
  <rdfs:subClassOf rdf:resource="#FormControl"/>
</owl:Class>
<owl:Class rdf:ID="FileUploadButton">
  <rdfs:subClassOf rdf:resource="#ActionButton"/>
</owl:Class>
<owl:Class rdf:ID="Quotation">
  <rdfs:subClassOf rdf:resource="#TextUnit"/>
</owl:Class>
<owl:Class rdf:ID="AudioTrackOfVideo">
  <rdfs:subClassOf rdf:resource="#AudioContent"/>
</owl:Class>
<owl:Class rdf:about="#Technology">
  <rdfs:subClassOf rdf:resource="#MetaData"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="containers">
  <rdfs:domain rdf:resource="#Site"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="stylesheets">
  <rdfs:domain rdf:resource="#Page"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="formControls"/>
<owl:ObjectProperty rdf:ID="textUnits"/>
<owl:ObjectProperty rdf:ID="tableCells">
  <rdfs:domain rdf:resource="#TableComponent"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="listItems">
  <rdfs:domain rdf:resource="#List"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="semanticData">
  <rdfs:domain rdf:resource="#Content"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="technologies">
  <rdfs:domain rdf:resource="#Page"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="metadata">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Page"/>
        <owl:Class rdf:about="#Site"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="compositeContainer"/>
<owl:ObjectProperty rdf:ID="markupLanguagesUsed">
  <rdfs:domain rdf:resource="#Page"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="activeRegions">
  <rdfs:domain rdf:resource="#ImageMap"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="noScriptAlternative">
  <rdfs:domain rdf:resource="#Script"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="contents">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#ListItem"/>
        <owl:Class rdf:about="#Page"/>
        <owl:Class rdf:about="#Paragraph"/>
        <owl:Class rdf:about="#Form"/>
        <owl:Class rdf:about="#Frame"/>
        <owl:Class rdf:about="#NavigationalAid"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Pages"/>
<owl:FunctionalProperty rdf:ID="url">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Page"/>
        <owl:Class rdf:about="#Site"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="color">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#TextLink"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="legend">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#FormGroup"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="content">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#TableCell"/>
```

```
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="lang">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Language"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="function">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Link"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="number">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="image">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#ImageButton"/>
        <owl:Class rdf:about="#ImageLink"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="emailAddress">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#SupportContact"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="dtd">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#MarkupLanguage"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="length">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="tabOrder">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Content"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="listStyleImage">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#ListItem"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="numberOfRows">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TableComponent"/>
        <owl:Class rdf:about="#Table"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="size">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Page"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="autoUpdate">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#Page"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="listStyleType">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#ListItem"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="tableFoot">
```

```
    <rdfs:domain rdf:resource="#Table"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="phoneNumber">
    <rdfs:domain rdf:resource="#SupportContact"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="target">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Link"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="faxNumber">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#SupportContact"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:about="#side">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ImageMap"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="important">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Link"/>
          <owl:Class rdf:about="#Page"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="class">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Content"/>
          <owl:Class rdf:about="#Container"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="controlLabel">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#FormControl"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="dynamic">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Content"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="caption">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#Table"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="positionColumn">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdfs:domain rdf:resource="#TableCell"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="accessKey">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Content"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="title">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Link"/>
          <owl:Class rdf:about="#Page"/>
```

```
        <owl:Class rdf:about="#FormControl"/>
        <owl:Class rdf:about="#Abbreviation"/>
        <owl:Class rdf:about="#Acronym"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="levelOfHeading">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#HeadingElement"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="firstName">
  <rdfs:domain rdf:resource="#SiteAuthor"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="video">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#VideoLink"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="audioFileName">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="tableBody">
  <rdfs:domain rdf:resource="#Table"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="id">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Content"/>
        <owl:Class rdf:about="#Container"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="lastName">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#SiteAuthor"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="numberOfColumns">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TableComponent"/>
        <owl:Class rdf:about="#Table"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="markupLanguage">
  <rdfs:domain rdf:resource="#W3CTechnology"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="backgroundColor">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Content"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="group">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Link"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="location">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
```

```
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#StyleSheet"/>
        <owl:Class rdf:about="#VideoContent"/>
        <owl:Class rdf:about="#Applet"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="eventHandlerType">
  <rdfs:domain rdf:resource="#ProgrammaticObject"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="height">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#ImageContent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="textualAlternative">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#MultimediaContent"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="technologyName">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Technology"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="abbreviation">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#HeaderCell"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="width">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#ImageContent"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="semantic">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="audioFile">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#AudioContent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="listOfStyleRules">
  <rdfs:domain rdf:resource="#StyleSheet"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="blinking">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#TextContent"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="referencePage">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#AlternativePage"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#type">
  <rdfs:domain rdf:resource="#Page"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="value">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#SemanticData"/>
        <owl:Class rdf:about="#AccessKey"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
```

```
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="tableHeader">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Table"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="postalAddress">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#SupportContact"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="targetPath">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:domain rdf:resource="#Link"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="code">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Script"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="cite">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Quotation"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="label">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Link"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="textContent">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#TextUnit"/>
        <owl:Class rdf:about="#TextInput"/>
        <owl:Class rdf:about="#GraphicalRepresentationOfText"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="description">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#MultimediaContent"/>
        <owl:Class rdf:about="#Frame"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="source">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Image"/>
        <owl:Class rdf:about="#Frame"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="autoRedirect">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
  <rdfs:domain rdf:resource="#Page"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="WebApplication-v2_Slot_8">
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="summary">
```

```
    <rdfs:domain rdf:resource="#Table"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="foregroundColor">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Content"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="positionRow">
    <rdfs:domain rdf:resource="#TableCell"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="colorValue">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="page">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#Window"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="name">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#MarkupLanguage"/>
          <owl:Class rdf:about="#Frame"/>
          <owl:Class rdf:about="#SpeechSynthetizer"/>
          <owl:Class rdf:about="#Browser"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="language">
    <rdfs:domain rdf:resource="#Content"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="referenceTable">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#EquivalentTable"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="buttonLabel">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#ActionButton"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="submitButtonImage">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    <rdfs:domain rdf:resource="#SubmitButton"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Annexe C. Règles d'accessibilité WCAG 1.0

Guideline 1: Provide equivalent alternatives to auditory and visual content					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
1.1	Provide a text equivalent for every non-text element (e.g., via "alt", "longdesc", or in element content). *This includes*: images, graphical representations of text (including symbols), image map regions, animations (e.g., animated GIFs), applets and programmatic objects, ascii art, frames, scripts, images used as list bullets, spacers, graphical buttons, sounds (played with or without user interaction), stand-alone audio files, audio tracks of video, and video.	1	1.1.1	Sur une image, vérifier la présence d'une alternative textuelle	Image
			1.1.2	Sur une image, vérifier que l'alternative textuelle n'est pas vide	Image
			1.1.3	Sur une image, vérifier que le descriptif de l'alternative textuelle est correct	Image
			1.1.4	Si une image est utilisée dans un lien, vérifier la présence d'une alternative textuelle	ImageLink
			1.1.5	Si une image est utilisée dans un lien, vérifier que l'alternative textuelle n'est pas vide	ImageLink
			1.1.6	Si une image est utilisée dans un lien, vérifier que le descriptif de l'alternative textuelle est correct	ImageLink
			1.1.7	Si une image est utilisée dans un lien textuel (i.e. le lien est constitué d'une image et d'un texte), vérifier que l'alternative textuelle est égale à " "	ImageLink
			1.1.8	Ne pas mettre d'images pour les puces de liste. Si c'est le cas, vérifier que l'alternative textuelle des puces de liste est égale à "* "	ListItem
			1.1.9	Si une image est importante pour la compréhension d'une page Web et doit être décrite, donner une description longue de l'image	Image
			1.1.10	Sur un bouton graphique, vérifier la présence d'une alternative textuelle	SubmitButton
			1.1.11	Sur un bouton graphique, vérifier que l'alternative textuelle n'est pas vide	SubmitButton
			1.1.12	Sur un bouton graphique, vérifier que le descriptif de l'alternative textuelle est correct. Cette alternative textuelle doit donner la **fonction du bouton**. La plupart du temps, celle-ci est la même que le texte du bouton	SubmitButton
			1.1.13	Vérifier que chaque composant audio a une alternative textuelle qui lui est associée	AudioContent
			1.1.14	Vérifier que chaque composant audio-vidéo a des sous-titres (alternatives textuelles) qui lui sont associés	AudioContent, VideoContent
			1.1.15	Sur un script, la règle 6.2.1 doit être vérifiée.	No Objects
1.2	Provide redundant text links for each active region of a server-side image map.	1			ImageMap, TextLink
1.3	Until user agents can automatically read aloud the text equivalent of a visual track, provide an auditory description of the important information of the visual track of a multimedia presentation.	1			VideoContent, AudioContent
1.4	For any time-based multimedia presentation (e.g., a movie or animation), synchronize equivalent alternatives (e.g., captions or auditory descriptions of the visual track) with the presentation.	1			VideoContent, AudioContent
1.5	Until user agents render text equivalents for client-side image map links, provide redundant text links for each active region of a client-side image map.	3			ImageMap, TextLink
Guideline 2: Don't rely on color alone					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
2.1	Ensure that all information conveyed with color is also available without color, for example from context or markup.	1			Site
2.2	Ensure that foreground and background color combinations provide sufficient contrast when viewed by someone having color deficits or when viewed on a black and white screen.	2	2.2.1	Vérifier que le code RVB est utilisé pour les couleurs et pas le nom (des couleurs).	Content
			2.2.2	Vérifier que la combinaison (couleur de premier plan, couleur de fond) des couleurs offre suffisamment de contraste	Content
Guideline 3: Use markup and style sheets and do so properly					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
3.1	When an appropriate markup language exists, use markup rather than images to convey information.	2			Image
3.2	Create documents that validate to published formal grammars.	2	3.2.1	Vérifier que chaque page Web mentionne les langages qu'elle utilise (vérification de la balise <!DOCTYPE>).	Page
			3.2.2	Valider chaque page XHTML	Page

			3.2.3	Valider chaque page CSS	Page
3.3	Use style sheets to control layout and presentation.	2	3.3.1	Vérifier que la page a une feuille de style.	Page
			3.3.2	Fournir la possibilité de gérer la mise en page.	Page
3.4	Use relative rather than absolute units in markup language attribute values and style sheet property values.	2			ImageContent, Page
3.5	Use header elements to convey document structure and use them according to specification.	2			HeadingElement
3.6	Mark up lists and list items properly.	2			List
3.7	Mark up quotations. Do not use quotation markup for formatting effects such as indentation.	2			Quotation
Guideline 4: Clarify natural language usage					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
4.1	Clearly identify changes in the natural language of a document's text and any text equivalents (e.g., captions).	1			Language, Content
4.2	Specify the expansion of each abbreviation or acronym in a document where it first occurs.	3	4.2.1	Identifier les acronymes et les abréviations dans le document.	Abbreviation, Acronym
			4.2.2	Vérifier que la première occurrence de l'acronyme ou de l'abréviation est signalée.	Abbreviation, Acronym
4.3	Identify the primary natural language of a document.	3			Language
Guideline 5: Create tables that transform gracefully					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
5.1	For data tables, identify row and column headers.	1	5.1.1	Vérifier qu'un tableau contient une cellule d'en-tête pour chaque colonne.	DataTable
			5.1.2	Vérifier que les cellules d'en-tête ne sont pas vides.	DataTable
			5.1.3	Vérifier que le descriptif de chaque cellule d'en-tête est correct.	DataTable
5.2	For data tables that have two or more logical levels of row or column headers, use markup to associate data cells and header cells.	1	5.2.1	Pour un tableau, vérifier que chaque cellule d'en-tête (th) comporte un attribut id	DataTable
			5.2.2	Chaque cellule d'en-tête doit être reliée explicitement à sa colonne	DataTable
5.3	Do not use tables for layout unless the table makes sense when linearized. Otherwise, if the table does not make sense, provide an alternative equivalent (which may be a linearized version).	2			Table, LayoutTable, EquivalentTable
5.4	If a table is used for layout, do not use any structural markup for the purpose of visual formatting.	2			LayoutTable
5.5	Provide summaries for tables.	3	5.5.1	Vérifier que chaque tableau contient une description (ou résumé)	Table
			5.5.2	Vérifier que cette description n'est pas vide	Table
			5.5.3	Vérifier que le descriptif de cette description est correct	Table
5.6	Provide abbreviations for header labels.	3	5.6.1	Vérifier qu'une abréviation est donnée pour une cellule d'en-tête qui contient un texte long	Table
			5.6.2	Vérifier que cette abréviation n'est pas vide.	Table
			5.6.3	Vérifier que le descriptif de cette abréviation est correct	Table
Guideline 6: Ensure that pages featuring new technologies transform gracefully					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
6.1	Organize documents so they may be read without style sheets. For example, when an HTML document is rendered without associated style sheets, it must still be possible to read the document.	1	6.1.1	Pour tout contenu généré, assurez-vous que le contenu important apparaît dans le code source de la page (ex. le contenu généré par une CSS n'apparaît pas dans le code source de la page).	Page
			6.1.2	Pour tout contenu généré avec des feuilles de styles CSS, fournir un équivalent textuel dans le code source.	Page
			6.1.3	Faire apparaître visuellement la notion de séparation dans le document.	Page
			6.1.4	S'assurer que la séparation visuelle est bien faite.	Page
			6.1.5	L'ordre d'apparition des éléments doit être le même avec ou sans feuilles de styles.	Page
			6.1.6	La structure visuelle (hiérarchie des différentes parties du document, listes, tableaux, etc.) doit se refléter à la fois dans le document avec les feuilles de styles activées et désactivées.	Page
6.2	Ensure that equivalents for dynamic content are updated when the dynamic content changes.	1	6.2.1	Vérifier que chaque script a une alternative	Script
			6.2.2	Vérifier que cette alternative est bien renseignée et permet d'avoir accès aux mêmes fonctionnalités que le script	Script
			6.2.3	Dans le cas où l'alternative à un script est textuelle, s'assurer qu'elle change à chaque fois que les informations données par le script change.	Script
			6.2.4	Vérifier que le seul élément contenu dans un cadre (frame) est une page Web	Frame
6.3	Ensure that pages are usable when scripts, applets, or other programmatic objects are turned off or not supported. If this is	1			ProgrammaticObjects, AlternativePage

	not possible, provide equivalent information on an alternative accessible page.				
6.4	For scripts and applets, ensure that event handlers are input device-independent.	2			ProgrammaticObject
6.5	Ensure that dynamic content is accessible or provide an alternative presentation or page.	2	6.5.1	Vérifier que des méta données sont utilisées pour désigner les pages alternatives lorsque de telles pages sont présentes	Page
			6.5.2	Ne pas générer du contenu à la volée.	Page
			6.5.3	Ne pas créer de liens faisant appel directement à un script (p.ex. du code javascript) directement depuis l'URI	Link
			6.5.4	La règle 1.4 doit être vérifiée.	No objects
Guideline 7: Ensure user control of time-sensitive content changes					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
7.1	Until user agents allow users to control flickering, avoid causing the screen to flicker.	1			Site
7.2	Until user agents allow users to control blinking, avoid causing content to blink (i.e., change presentation at a regular rate, such as turning on and off).	2			TextContent
7.3	Until user agents allow users to freeze moving content, avoid movement in pages.	2			Content
7.4	Until user agents provide the ability to stop the refresh, do not create periodically auto-refreshing pages.	2			Page
7.5	Until user agents provide the ability to stop auto-redirect, do not use markup to redirect pages automatically. Instead, configure the server to perform redirects.	2			Page
Guideline 8: Ensure direct accessibility of embedded user interfaces					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
8.1	Make programmatic elements such as scripts and applets directly accessible or compatible with assistive technologies	1			ProgrammaticObject
Guideline 9: Design for device-independence					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
9.1	Provide client-side image maps instead of server-side image maps except where the regions cannot be defined with an available geometric shape.	1			ImageMap
9.2	Ensure that any element that has its own interface can be operated in a device-independent manner.	2		Une page doit être conçue pour être accessible au minimum au clavier. Une page accessible au clavier l'est généralement avec d'autres dispositifs d'entrée.	Page
9.3	For scripts, specify logical event handlers rather than device-dependent event handlers.	2			Script
9.4	Create a logical tab order through links, form controls, and objects.	3			Content
9.5	Provide keyboard shortcuts to important links (including those in client-side image maps), form controls, and groups of form controls.	3			Link, Form, FormGroup
Guideline 10: Use interim solutions					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
10.1	Until user agents allow users to turn off spawned windows, do not cause pop-ups or other windows to appear and do not change the current window without informing the user.	2			Window
10.2	Until user agents support explicit associations between labels and form controls, for all form controls with implicitly associated labels, ensure that the label is properly positioned.	2	10.2.1	Vérifier que l'étiquette est alignée à gauche du champ (collée au champ)	FormControl
			10.2.2	La règle 12,4 doit être vérifiée.	No objects
			Note	Cette règle est cependant **obsolète**. Elle n'existait que pour les technologies assistantes qui ne géraient pas la balise <label> et son attribut « for ». Par conséquent, il fallait les associer visuellement (critère de proximité). Or, aujourd'hui, toutes les technologies assistantes supportent cette balise (et son attribut « for »).	No objects
10.3	Until user agents (including assistive technologies) render side-by-side text correctly, provide a linear text alternative (on the current page or some other) for all tables that lay out text in parallel, word-wrapped columns.	3			Table
10.4	Until user agents handle empty controls correctly, include	3	10.4.1	Vérifier que les contrôles de formulaires ne sont pas vides par défaut en fournissant un texte	TextInput

	default, place-holding characters in edit boxes and text areas.		10.4.2	Vérifier que le descriptif de ce texte est correct	TextInput
			Note	Cette règle est **obsolète.** Elle n'existait que pour les lecteurs d'écran qui n'arrivaient pas à lire les contrôles vides, c'est-à-dire qu'ils ne détectaient pas les contrôles vides et par conséquent ne les signalaient pas à l'utilisateur. Aujourd'hui, les lecteurs d'écran gèrent les contrôles vides.	No objects
10.5	Until user agents (including assistive technologies) render adjacent links distinctly, include non-link, printable characters (surrounded by spaces) between adjacent links.	3			Link
Guideline 11: Use W3C technologies and guidelines					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
11.1	Use W3C technologies when they are available and appropriate for a task and use the latest versions when supported.	2			W3CTechnology
11.2	Avoid deprecated features of W3C technologies.	2			W3CTechnology
11.3	Provide information so that users may receive documents according to their preferences (e.g., language, content type, etc.)	3			Site
11.4	If, after best efforts, you cannot create an accessible page, provide a link to an alternative page that uses W3C technologies, is accessible, has equivalent information (or functionality), and is updated as often as the inaccessible (original) page.	1	11.4.1	Si le recours à une page alternative est nécessaire, fournir des liens en haut de chaque page (principale et alternative) pour permettre à l'utilisateur de naviguer entre ces deux pages.	Page
			11.4.2	Utiliser des méta données pour désigner les pages alternatives. Les navigateurs devraient charger automatiquement les pages alternatives selon les préférences de l'utilisateur et le type du navigateur.	Page
			11.4.3	Si une page reste inaccessible, fournir un n° de tél., un n° de fax, une adresse e-mail ou une adresse postale pour contacter le support afin d'obtenir de l'information	Page
			11.4.4	Vérifier qu'au moins une des informations précédentes (n° tél, n° fax, email, adresse postale) est renseignée.	Page
Guideline 12: Provide context and orientation information					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
12.1	Title each frame to facilitate frame identification and navigation.	1	12.1.1	Vérifier que chaque cadre (frame) contient un titre	Frame
			12.1.2	Vérifier que ce titre n'est pas vide	Frame
			12.1.3	Vérifier que le descriptif de ce titre est correct	Frame
12.2	Describe the purpose of frames and how frames relate to each other if it is not obvious by frame titles alone.	2			Frame
12.3	Divide large blocks of information into more manageable groups where natural and appropriate.	2	12.3.1	Pour un formulaire, rassembler les champs qui sont liés sémantiquement en groupes	Form, FormGroup
			12.3.2	Vérifier qu'une description est donnée pour chaque groupe	FormGroup
			12.3.3	Vérifier que, pour chaque groupe, cette description n'est pas vide	FormGroup
			12.3.4	Vérifier que, pour chaque groupe, la description donnée est correcte	FormGroup
			12.3.5	A l'intérieur d'une combobox, si nécessaire, grouper les choix offerts à l'utilisateur	Combobox
			12.3.6	Dans le cas des boutons radio, l'utilisation des groupes est possible. La question posée est contenue dans la description du groupe (en html, cette description est représentée par l'élément <legend>) et les choix possibles sont contenus dans le groupe (en html, un groupe est représenté par l'élément <fieldset>). Dans ce cas, la règle 12.4.4 ne doit pas être prise en compte.	RadioButton
12.4	Associate labels explicitly with their controls.	2	12.4.1	Pour chaque contrôle de formulaire, vérifier que son étiquette, s'il en possède une, lui est associée explicitement dans le code source	FormControl
			12.4.2	Dans le cas des contrôles de formulaires qui n'ont pas d'étiquettes, utiliser l'attribut title. Le descriptif de cet attribut doit être court et concis, par ex. "Search text" pour le champ de saisie textuel d'un moteur de recherche.	FormControl
			12.4.3	Dans le cas des tables, mettre dans l'attribut title de chaque contrôle de formulaire les informations sur la ligne et la colonne qui lui sont associées. Ex. "Jean Dupont, âge" pour désigner que ce contrôle de formulaire se trouve à la ligne "Jean Dupont" et dans la colonne "âge".	FormControl, Table
			12.4.4	Dans le cas des boutons radio, mettre la question posée et la réponse dans l'attribut title. Par ex. si nous utilisons des boutons radio pour déterminer le sexe d'une personne, nous aurons dans l'attribut title du 1er bouton radio "sexe, masculin" et pour le 2ème "sexe, féminin". Si cette technique est	RadioButton

				utilisée, la règle 12.3.6 ne doit pas être prise en compte.	
			12.4.5	En html, ne jamais utiliser la balise <label> en tant que container, c'est-à-dire, ne jamais englober un contrôle de formulaire avec cette balise. Préférer plutôt l'utilisation des attributs for et id. Les lecteurs d'écran gèrent mal le cas du label en tant que container.	FormControl
Guideline 13: Provide clear navigation mechanisms					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
13.1	Clearly identify the target of each link.	2	13.1.1	Le libellé de chaque lien doit être explicite.	Link
			13.1.2	Chaque lien avec un libellé identique doit pointer vers la même ressource, sauf si les descriptifs de chaque lien (donné en en HTML par l'attribut title) sont différents	Link
			13.1.3	Pour une suite de liens (adjacents) liés sémantiquement, fournir de l'information contextuelle sur le 1er lien.	Link
13.2	Provide metadata to add semantic information to pages and sites.	2	13.2.1	Vérifier que le titre de la page est indiqué dans l'entête de la page (en html, dans la partie <head>)	Page
			13.2.2	Eventuellement, fournir l'adresse du support (auteur de la page, ou support technique de l'application)	Page
			13.2.3	Vérifier que le descriptif de l'adresse est correct	Page
			13.2.4	Les règles 7.4, 7.5, 3.2, 13.9, 6.5 et 3.6 doivent être vérifiées	No objects
13.3	Provide information about the general layout of a site (e.g., a site map or table of contents).	2			NavigationScheme
13.4	Use navigation mechanisms in a consistent manner.	2			NavigationMechanism
13.5	Provide navigation bars to highlight and give access to the navigation mechanism.	3			NavigationBar
13.6	Group related links, identify the group (for user agents), and, until user agents do so, provide a way to bypass the group.	3			Link
13.7	If search functions are provided, enable different types of searches for different skill levels and preferences.	3			SearchFacility
13.8	Place distinguishing information at the beginning of headings, paragraphs, lists, etc.	3			HeadingElement, List, Paragraph
13.9	Provide information about document collections (i.e., documents comprising multiple pages.).	3	13.9.1	Fournir des méta données concernant les mécanismes de navigation et l'organisation dans le site.	Page
			13.9.2	Fournir un paquetage des pages (p.ex. archive zip) pour faciliter la lecture des pages hors connexion.	Site
13.10	Provide a means to skip over multi-line ASCII art.	3			ASCIIArt
Guideline 14: Ensure that documents are clear and simple					
N°	**Checkpoint**	**Priority**	**N°**	**Sub-checkpoint**	**Ontology Concepts**
14.1	Use the clearest and simplest language appropriate for a site's content.	1			Site
14.2	Supplement text with graphic or auditory presentations where they will facilitate comprehension of the page.	3			Page
14.3	Create a style of presentation that is consistent across pages.	3			Page

Annexe D. Règles de navigation EvalWeb

N°	Règle	N°	Interprétation	Concepts de l'ontologie	Source
1	Vérifier que les liens connectent vers des pages qui existent			Link, Page	[9]
2	Testez la navigation.	2.1	Chaque page doit contenir au moins un lien	Page, Link	[3]
		2.2	Chaque lien doit mener vers une page existante	Link, Page	
		2.3	Le libellé d'un lien doit être explicite pour que l'utilisateur ait une idée de l'information qui va lui être affichée	Link	
		2.4	Toute page de l'application doit pouvoir être atteinte	Page, Link	
3	Eliminez les liens erronés ou équivoques. Ne faites pas référence à de l'information manquante.			Link	[8]
4	Evitez les références vers des documents inachevés. Ils augmentent le temps de téléchargement et les utilisateurs bénéficiant d'une faible connexion ne vont pas apprécier.			Link	[10]
5	Un lien doit apparaître avec son contexte.			Link	[11]
6	Regroupez ensemble les éléments de navigation.			NavigationMechanism, Link	[1]
7	Afin d'aider le lecteur à choisir un lien, les positions des liens doivent être évidents.			Link	[12]
8	Indiquez à un utilisateur quand un lien a déjà été sélectionné. Si un utilisateur sélectionne un lien et qu'il existe d'autres liens vers la même cible, assurez-vous que tous ces liens changent de couleur.			Link	[2]
9	Ne faites pas confiance aux éléments graphiques réagissant au passage de la souris. Utilisez toujours le soulignement ou d'autres indicateurs visuels pour indiquer que tels mots sont des liens.			TextLink	[5]
10	Etiquetez les liens de façon descriptive afin que l'utilisateur puisse différencier les liens similaires.			Link	[2]
11	Indiquez clairement quand un lien amène l'utilisateur vers (a) la même page, (b) une page différente du même site ou (c) une page sur un site différent.			Link	[2]
12	Mettez en valeur les textes différents (par ex. quand plusieurs liens commencent par les mêmes mots).			Link	[11]
13	Du feedback visuel peut être donné à l'utilisateur en l'intégrant directement dans les éléments de navigation.			NavigationalScheme, Link	[13]
14	Utilisez des libellés indiquant clairement la fonction des liens.			Link	[3]
15	Ecrivez de bons libellés pour les liens.			Link	[4]
16	Les liens sous forme d'icône doivent bien représenter la page vers laquelle elles mènent.			ImageLink	[9]
17	Les éléments de navigation dans un site Web doivent être affordants (un élément de l'interface est affordant lorsque l'utilisateur peut deviner sa fonction de par son apparence).			NavigationMechanism, Link	[13]
18	Une URL doit être un chemin lisible et compréhensible par un utilisateur contenant des noms de fichiers qui reflètent la nature de l'information.			Page	[16]
19	Utilisez la marge droite pour l'index principal du site Web.		L'index est considéré comme une table des matières	Page, TableOfContents	[1]
20	Utilisez le même schéma de navigation pour toutes les pages.			NavigationScheme	[1]
21	Utilisez les éléments de navigation de façon constante.			NavigationMechanism	[3]
22	Utilisez des liens textuels. N'utilisez pas de liens sous forme d'image.			TextLink, ImageLink	[2]
23	Assurez-vous que les images cliquables peuvent être perçues par les personnes malvoyantes.			ImageLink	[3]
24	Concevez judicieusement les icônes de navigation.			ImageLink	[4]
25	Dans un site Web bien conçu, le schéma de navigation est visible et aisément compréhensible tout en ayant une présentation attrayante.			NavigationScheme	[13]
26	Essayez de fournir constamment des liens vers la page d'accueil et vers les catégories importantes du site.			Link, Page	[3]
27	Assurez-vous que l'utilisateur peut accéder au contenu le plus important depuis plus d'un lien textuel.			Page, TextLink	[6]

28	Intégrez des liens vers les principales sections dans l'aide à la navigation.			NavigationalAid, Link, Page	[6]
29	Placez les aides à la navigation en haut et en bas des pages nécessitant un défilement (ou dans des frames).			Page, NavigationalAid	[6]
30	Intégrez un lien vers le début de la section dans l'aide à la navigation.			NavigationalAid, Page	[6]
31	Utilisez les aides à la navigation partout où vous le pouvez.			NavigationalAid	[1]
32	(En ce qui concerne les pop-ups et les notes de bas de pages,) les liens vers plus d'informations présentés aux utilisateurs (devraient être présentés) sans quitter le contexte de la page courante.			Window, Link	[7]
33	Intégrez des liens vers des services de bases (tels que la rubrique « aide ») dans l'aide à la navigation.			Link, Page	[6]
34	Intégrez une aide à la navigation dans chaque page.			NavigationalAid, Page	[6]
35	Fournissez du feedback indiquant à l'utilisateur où il se trouve dans votre site (appliqué aux liens).			Link	[3]
36	Aidez les utilisateurs venant de l'extérieur à s'orienter afin d'accéder facilement à la page d'accueil et aux autres sections majeures grâce à l'aide à la navigation (text/bar).			LocationHeader	[6]
37	Une structure commune pour un site Web est d'avoir une ou plusieurs pages de navigation pour guider l'utilisateur vers les pages contenant l'information désirée (approche hiérarchique de la navigation).			Page	[13]
38	L'utilisateur doit pouvoir connaître la section dans laquelle il se trouve et où exactement il se trouve dans cette section en ayant soit directement cette information à l'écran soit la possibilité d'accéder à cette information par un mécanisme dont il a connaissance.			NavigationMechanism	[12]
39	L'utilisation des index à travers l'espace des documents aide à l'orientation des utilisateurs, minimisant ainsi le phénomène de « perte dans l'hyperespace ».		L'index est considéré comme une table des matières	TableOfContents	[14]
40	Intégrez des éléments visuels indiquant la position actuelle de l'utilisateur, le chemin qu'il a emprunté pour arriver jusque là et les options s'offrant à lui pour continuer sa navigation.	40.1	Utilisez des en-têtes de localisation pour indiquer à l'utilisateur sa position dans le site.	LocationHeader	[15]
41	Offrez plusieurs approches pour la navigation.			NavigationMechanism, Link	[4]
42	Pensez à dupliquer les en-têtes de navigation en bas de vos pages.			LocationHeader	[11]
43	Il est souhaitable d'inclure l'adresse URL de la page en bas de la page.			Page	[9]
44	Essayez de faire correspondre le libellé du lien avec le titre de la page résultante.			Link, Page	[11]
45	Placez les onglets utilisés pour les liens en haut de la page et assurez-vous qu'ils ont l'air cliquables, tels de vrais onglets.			NavigationBar	[5]
46	Les liens typés permettent à l'utilisateur d'affecter un sens aux différentes relations entre documents.			Link	[7]
47	Organisez les liens en thèmes primaires et secondaires.			Link	[9]
48	Choisissez vos liens de façon qu'ils supportent votre phrase et la structure du concept			Link	[11]

Références

[1] http://usability.gov/guidelines

[2] Spool, J.M.; Scanlon, T.; Schroeder, W.; Snyder, C.; DeAngelo, T. Web Site Usability: A Designer's Guide, North Andover, MA User Interface Engineering, 1997

[3] http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/748

[4] http://builder.cnet.com/webbuilding/pages/Graphics/CTips2/ss12a.html

[5] Bailey, R.W.; Koyani, S.; Nall, J. Usability testing of several health information Web sites, National Cancer Institute Technical Report, September 7-8, 2000.

[6] Detweiler, M.C.; Omanson, R.C. Ameritech Web Page User Interface Standards and Design Guidelines, 1996 (www.ameritech.com).

[7] Instone, K. Opportunities for Improving the World Wide Web,1996. Available at http://www.acm.org/sigchi/webchi/chi96workshop/

[8] Levi, M.D.; Conrad, F.G. A Heuristic Evaluation of a World Wide Web Prototype, Interactions, III.4, pp. 51-61, 1996

[9] Borges, J.A.; Morales, I.; Rodriguez, N.J. Guidelines for Designing Usable World Wide Web Pages, In M.J. Tauber (Eds.), Conference on Human Factors in Computing Systems, CHI'96 Conference, April 13-18, pp. 277-278, New York, NY, 1996.

[10] DPIE. PIENet Web Publishinng Standards and Guidelines. Australian Department of Primary Industries and Energy, 1996. Available at http://www.dpie.gov.au/dpie/web/construction.html

[11] Levine, R. Guide to Web Style, Sun Microsystems, Inc., 1996. Available at http://www.sun.com/styleguide/

[12] Hardman, L.; Sharratt, B.S. User-centered hypertext design: the application of HCI design principles and guidelines, In R.M. Aleese & C.Green (Eds.), Hypertext: State of the Art, Chapter 28, London: Intellect Limited. 1990

[13] IBM Corporation, IBM Web Guidelines - Complete set, 1997, Available at http://www.ibm.com/IBM/HCI/guidelines/web/print.html

[14] Catledge, L.; Pitkow, J.E. Characterizing Browsing Strategies in the World-Wide Web, In Proceedings of 3rd International World Wide Web Conference, Darmstadt, Germany, 1995. Available at http://www.gatech.edu/lcc/idt/Students/Catledge/browsing/UserPatterns.Paper4.formatted.html

[15] Thüring, M.; Hannemann, J.; Haake, J. Designing for Comprehension: A Cognitive Approach to Hypermedia Development, Communications of the ACM, vol. 38, n° 8, pp. 57-66, 1995. Available at http://space.njit.edu:5080/cacm/overview.html

[16] Nielsen, Top Ten Mistakes in Web Design, 1996. Available at http://www.useit.com/alertbox/9605.html

Annexe E. Fichier de mappings ATL StateWebCharts/Ontologie

```
module Swc2Ontology; -- Module Template
create OUT : WebApplication from IN : Swc;

------
-- Return the children of a composite state
------
helper context Swc!CompositeState def: getChildren() : OrderedSet(Swc!StateVertex) =
    let allChildren : OrderedSet(Swc!StaticState) = self.getAllChildren()
    in allChildren->select(state | state.refImmediateComposite() = self);

helper context Swc!CompositeState def: getChild(state : Swc!StateVertex) :
OrderedSet(Swc!StateVertex) =
    if state.oclIsUndefined()
    then
        OrderedSet{}
    else
        OrderedSet{state}
    endif;

------
-- Return the list of all the children (recursively) for a composite state
------
helper context Swc!CompositeState def: getAllChildren() : OrderedSet(Swc!StateVertex) =
    let staticStates : OrderedSet(Swc!StaticState) = self.StaticState
    in let children : OrderedSet(Swc!StateVertex) = OrderedSet{}
            ->union(staticStates)
            ->union(self.getChild(self.DeepHistory))
            ->union(self.getChild(self.ShallowHistory))
            ->union(self.getChild(self.EndState))
    in let compositeStates : OrderedSet(Swc!CompositeState) = self.CompositeState
    in if compositeStates.oclIsUndefined()
    then
        children
    else
        let elem : OrderedSet(Swc!StaticState) =
            compositeStates->iterate(child ; elements: OrderedSet(Swc!StateVertex) =
OrderedSet{} |
            if child.oclIsTypeOf(Swc!CompositeState)
            then
                elements->union(child.getAllChildren())
            else
                elements->append(child)
            endif
            )
        in children->union(elem)
    endif;

helper context Swc!Transition def: getTargetState() : Swc!StateVertex =
    let allStates : OrderedSet(Swc!StateVertex) = Swc!StateVertex.allInstances()
    in allStates->select(s | s.id = self.target);

helper context Swc!Transition def: isValidTransition() : Boolean =
    let targetState : Swc!StateVertex = self.getTargetState()
    in targetState.oclIsTypeOf(Swc!StaticState) or
targetState.oclIsTypeOf(Swc!CompositeState);

------
-- Return the list of outgoing transitions for a state vertex
------
helper context Swc!StateVertex def: getOutgoingLinks() : OrderedSet(Swc!Transition) =
    let parent : OclAny = self.refImmediateComposite()
    in let parentOutgoingLinks : OrderedSet(Swc!Transition) =
        if parent.oclIsTypeOf(Swc!CompositeState) = false
            then
                OrderedSet{}
            else
                parent.getOutgoingLinks()
            endif
    in let allTransitions : OrderedSet(Swc!Transition) = Swc!Transition.allInstances()
    in allTransitions
```

```
            ->select(t | t.source = self.id and t.isValidTransition())
            ->union(parentOutgoingLinks)
   ;

------
-- Return the target state of a transition
------
helper context Swc!Transition def: getTargetState() : Swc!StaticState =
   let allStaticStates : OrderedSet(Swc!StaticState) = Swc!StaticState.allInstances()
   in allStaticStates->any(s | self.target = s.id);

------
-- Return the URL of a transition's target state
------
helper context Swc!Transition def: getTargetStateFile() : String =
   let state : Swc!StaticState = self.getTargetState()
   in if state.oclIsUndefined()
        then
            ''
        else
            state.file
        endif;

----------------------------------------
-- Mapping Rules                      --
----------------------------------------
rule Swc2Site {
   from
        sm: Swc!Swc
   using {
        root: Swc!CompositeState = sm.CompositeState;
   }
   to
        site: WebApplication!Site (
            startLine    <- root.startLine,
            startColumn  <- root.startColumn,
            endLine      <- root.endLine,
            endColumn    <- root.endColumn,
            containers   <- root.getAllChildren(),--root.StaticState
            url          <- root.file
        )
}

rule StaticState2Page {
   from
        state: Swc!StaticState (
            not state.oclIsKindOf(Swc!CompositeState)
            and state.type = #static
        )
   to
        page: WebApplication!Page (
            startLine    <- state.startLine,
            startColumn  <- state.startColumn,
            endLine      <- state.endLine,
            endColumn    <- state.endColumn,
            contents     <- links,
            url          <- state.file
        ),
        links : distinct WebApplication!TextLink foreach (link in
state.getOutgoingLinks()) (
            targetPath   <- link.getTargetStateFile()
        )
}
```

Title: An automatic guidelines inspection method throughout the development process of Web applications

---

Abstract

The increasing use of the Web as a software platform together with the advance of technology has promoted Web applications as a start point for delivering information and services. Facing to the ever growing number of users, usability became a major requirement for the universal access of Web applications. In the last years, a number of evaluation methods have been developed by researchers, practitioners and Information Technology companies to help organizations to identify and to fix usability problems. However, usability evaluation of Web sites is not a straightforward process. On one hand, usability evaluation requires some knowledge and expertise in software ergonomics. One the other hand, due to constant evolution of Web application, it is required frequent evaluations to make sure that content updates do not introduce new usability problems. In order to overcome these limitations, much effort has been devoted in the development of tools for automating the inspection of usability and accessibility guidelines. However, tools currently available can only evaluate the final applications. In this thesis we propose a model-based evaluation method that allows ensuring the ergonomic quality of Web applications throughout the lifecycle. We developed an ontology that organizes recommendations around interface elements of a Web application and that allows identifying precisely what elements to evaluate at each phase of the lifecycle. We exploit this ontology to assess the various artifacts produced throughout the lifecycle. This work has been applied and validated at an industrial scale on an e-service development platform.

Auteur : Joseph Xiong

Résumé :
Les applications Web actuelles offrent de plus en plus de services. Pour éviter les difficultés d'usage de ces applications, l'utilisabilité doit être assurée. L'évaluation de l'utilisabilité est une tâche qui requiert une expertise en ergonomie logicielle. Cette expertise peut être capitalisée sous forme de recommandations qui sont l'expression d'une connaissance en ergonomie et qui vont aider à l'évaluation. Toutefois, puisqu'il est nécessaire d'appliquer celles-ci de manière systématique, leur inspection manuelle peut s'avérer laborieuse.
Pour ne pas être limité par l'inspection manuelle, des outils ont été développés pour guider et supporter l'inspection automatique. Un des avantages de ces outils est que les connaissances en ergonomie y sont directement intégrées. De plus, le manque d'experts et le coût élevé des autres méthodes d'évaluation font que l'inspection automatique est une méthode adaptée. Cependant, ces outils ne peuvent évaluer que l'application finale et si des erreurs sont détectées, des modifications importantes de l'application peuvent avoir lieu.

Dans cette thèse nous proposons une méthode d'évaluation basée sur modèles permettant de s'assurer tout au long du cycle de vie de l'utilisabilité des applications Web développées. Nous avons établi une ontologie qui organise les recommandations autour des éléments de l'interface pour identifier précisément quels éléments évaluer à chaque étape du cycle de vie. Cette ontologie est exploitée pour vérifier l'utilisabilité sur les différents artefacts produits dans le cycle de vie. Ces travaux ont été appliqués et validés à l'échelle industrielle sur une plateforme de développement de téléservices.

Mots clés : Ergonomie, Web, Utilisabilité, Inspection automatique, Ingénierie Dirigée par les Modèles, Ontologie