

Rapport de stage

Développement d'une API PHP 5 pour la génération de document Open XML

Du 1^{er} avril au 30 juin 2010

A. Remerciements

Je tiens tout d'abord à remercier **Michel Hoël**, mon tuteur à Urbilog, qui m'a permis de réaliser ce stage au sein de l'équipe web. Il m'a suivi tout au long de ce stage et a toujours veillé au bon déroulement de celui-ci.

Je remercie également tous les membres de la société Urbilog, **Joffrey, Julien, Kamal, Adrien, Antoine, Martine, Ziad, Vincent, Mathieu, Benoît** pour leur accueil, leur bonne humeur et leurs conseils.

Je voudrais remercier tout particulièrement **Cyril Biencourt** pour l'aide qu'il m'a apportée durant la réalisation du projet.

Enfin je remercie **Bruno Beaufils**, mon tuteur de l'IUT qui m'a suivi lors de ce stage, ainsi que l'IUT A de Lille 1 pour l'enseignement que j'ai reçu.

B. Résumé

J'ai effectué mon stage au sein de la société Urbilog et plus précisément au sein du département recherche et développement. C'est une société de service qui développe des moyens de communications interactifs et de publication sur le Web, experte en accessibilité.

Ma mission consistait à écrire une API PHP 5 permettant de générer des documents Open XML en fusionnant un template et un fichier XML contenant les différentes informations à insérer. L'API est destinée à générer des rapports et présentations PowerPoint afin de voir l'étendue des choses possibles dans le domaine, avec une éventuelle commercialisation auprès de clients potentiellement intéressés.

Développé en objet avec PHP 5, l'api est basé sur un pseudo modèle MVC dont les fondements seront expliqués plus tard dans le rapport. Étant donné l'ampleur du projet et la quantité de choses à faire, quelques semaines supplémentaires seraient nécessaires pour avoir un produit pleinement commercialisable, néanmoins dans l'état actuel l'api gère déjà l'insertion de texte, d'image, de liste, de boucle, de tableau et intègre un moteur d'inférence. En outre un effort particulier a été effectué dans le projet en vue d'évolutions futures.

Ce stage m'a permis de réaliser une API PHP de A à Z et donc d'approfondir mes connaissances dans ce langage. Mais également, d'acquérir de nouvelles connaissances techniques sur le langage XML et d'apprendre de nouveaux concepts avec CLIPS.

C. Abstract

I made my internship in the company Urbilog and more exactly in the research and development department. It's a service company which develops interactive ways of communications and publication on Web, expert in accessibility.

My mission consisted in writing a PHP 5 API capable of generating Open XML documents by merging a template and an XML file containing the various information to insert. The API will be use to generate reports and PowerPoint presentations, in order to see the extent of possible things that can be done in the domain, with a possible marketing to potentially interested customers.

Developed in object with PHP 5, the API is based on an MVC model of which foundations will be explained later in the report. In view of the scale of the project and the quantity of things to do, some additional weeks would be necessary to have a completely marketable product, nevertheless in the current state the API already manages the insertion of text, loop, images, list, table and integrates an inference engine. In addition, a special effort was made to facilitate future evolutions.

Thanks to this internship, I had the opportunity to realize a PHP API from scratch and thus to improve my knowledge in this language. But also, to acquire new technical knowledge on the language XML and to lean new concepts like CLIPS.

SOMMAIRE

A. REMERCIEMENTS	3
B. RESUME	4
C. ABSTRACT	5
A. INTRODUCTION.....	8
B. LA SOCIETE URBILOG	9
1. Son identité	9
a. Historique.....	9
b. Partenaires	9
c. Quelques chiffres.....	10
2. Ses activités	10
a. Accessibilité du Web.....	10
b. Réseaux informatiques	11
c. Développement Web	11
d. Formations	11
e. Audit et conseils	12
3. Ses produits.....	12
a. Ocawa.....	12
b. Expert Fixer	13
c. Clavier virtuel	14
d. Ranking.....	14
4. Organisation de l'entreprise	15
C. LE SUJET DU STAGE.....	16
1. Présentation	16
2. Objectif	16
3. Les outils.....	17
a. Technologies employées	17
b. Environnement de travail et organisation du travail.....	17
c. Contraintes.....	18
D. L'EXISTANT... PAS SI EXISTANT.....	19
1. Détail de la structure OpenXML	19
2. Applications internes	23
3. Fichier XML d'entrée	23
E. L'ANALYSE	25
1. Des recherches et discussions obligatoires	25
a. Des outils trop simples ou trop complexes	25
b. Des discussions obligatoires	26
2. Analyse	28

a. Le package transformer.....	28
b. Le package document	29
c. Le package utils	30
d. Les autres packages.....	30
F. UTILISATION.....	31
G. CONCLUSION	32
H. BIBLIOGRAPHIE ET WEBOGRAPHIE	33
I. MOTS-CLES ET KEYWORDS	34
J. GLOSSAIRE	35
K. ANNEXES.....	36
1. Annexe 1 : diagramme de classes UML du package transformer	36
2. Annexe 2 : diagramme de classes UML du package document.....	37
3. Annexe 3 : diagramme de classes UML du package utils.....	38
4. Annexe 4 : diagramme de classes UML des autres packages	39
5. Annexe 5 : documentation utilisateur.....	40
A.DESCRPTION DE L'API	42
1. Le template	42
2. CLIPS	42
3. L'api	43
B.DOCUMENTATION UTILISATEUR.....	44
1. Création d'un template	44
a.La balise image.....	47
b.La balise liste.....	48
c.La balise loop.....	48
d.La balise loopRel	50
e.La balise rules.....	51
f.La balise slide.....	53
g.La balise text.....	54
2. Modification des règles CLIPS	55
3. Utilisation de l'api	56

A. Introduction

Que ce soit en entreprise, à l'école et de manière générale dès que l'on veut échanger, présenter des informations, les documents du type docx, pptx ou encore odt se révèlent indispensable et peuvent devenir vos meilleurs amis comme vos pire ennemis. En effet une présentation bien faite peut vous rapporter un nouveau client ou une bonne note, néanmoins on imagine facilement le temps et l'argent que l'on perd à créer ces documents.

Il existe des solutions pour générer ces documents, pourtant aucune n'est satisfaisante. Elles sont soit trop complexes, ce qui fait qu'il aurait été plus rapide de créer le document à la main, soit incroyablement simplistes, mais incomplètes. C'est pourquoi Urbilog souhaite créer une API permettant de trouver le juste milieu, c'est-à-dire une approche basée sur des templates créés dans un environnement WYSIWYG dans lesquelles on va venir injecter des informations afin de créer de nouveaux documents.

Le but du stage était donc d'écrire une API qui soit suffisamment puissante et rapide pour pouvoir générer des documents presque immédiatement exploitables, avec la possibilité d'ajouter de nouveaux éléments, concepts sans pour autant devenir trop simple ou trop complexe.

Dans un premier temps nous verrons l'activité de la société urbilog, son histoire, son fonctionnement, ses outils. Ensuite, je présenterai le sujet que l'on m'a confié, puis je détaillerais l'analyse du projet. Et enfin nous étudierons les aspects techniques de l'api, avec des exemples concrets.

B. La société Urbilog

Dans cette partie, je vais vous présenter la société Urbilog, ainsi que ses activités et ses produits.

1. Son identité

Urbilog développe des applications pour le Web allant de la création d'un simple site Internet jusqu'à la mise en place d'un Intranet, Extranet avec les applications associées. Rendre le WEB accessible est la priorité d'Urbilog. En effet, tout le monde a le droit à l'information quelque soit le lieu, la personne ou les outils utilisés. Depuis sa création en 1995, la société a su diversifier ses activités pour en arriver aujourd'hui à 5 pôles d'actions principaux :

- les réseaux informatiques
- le développement WEB
- la formation
- l'audit conseil
- et principalement l'accessibilité du WEB.

a. Historique

La Société Urbilog est une Société de services informatiques créée par Benoît Thieffey et Michel Hoël en 1995 sous la forme de cybercafé avec services informatiques, dans le quartier de la fac de droit de Lille-Moulins. Son activité démarre réellement en 1998 lors de l'arrivée d'un troisième actionnaire, Mathieu Froidure, malvoyant, donnant l'idée à la Société de se spécialiser dans l'accessibilité, un marché prometteur dû à une législation grandissante.

Les débuts sont durs, mais en 2002 Urbilog réussit à se trouver une place en décrochant un contrat avec la branche Recherche et développement d'Orange pour la création d'un outil d'audit, nommé « Ocawa », permettant de tester l'accessibilité d'un site. Suite à cela les contrats s'enchaînent. En 2003, IBM contacte Urbilog pour développer une application d'aide à la réparation de sites : l'outil « Expert-Fixer » qui audite les sites, mais peut aussi les réparer. En 2004, le développement de « Ocawa » reprend avec la création d'un nouveau site Internet et l'insertion de règles d'ergonomie dans l'audit des pages.

Depuis, Urbilog est devenue l'une des sociétés références dans le monde de l'accessibilité du web. À ce titre, elle opère pour nombre de grands noms du monde industriel et commercial ainsi que dans le secteur public, auprès des collectivités territoriales en France et en Europe.

b. Partenaires

Grâce à la qualité de ses prestations, sa réactivité et ses investissements en recherches et développement, Urbilog a gagné la confiance de grands groupes du monde de l'informatique, de la communication et du monde industriel et commercial. Entre autres, Urbilog travaille désormais aux côtés de :

- France Télécom (Développement d'Ocawa) ;

- IBM (Développement d'un outil d'audit avancé d'aide à la réparation de sites Web : Expert Fixer) ;
- Crédit du nord ;
- Le groupe La Poste ;
- La banque Postale.

De nombreuses TPE et PME de la région du Nord font également partie de ses références : Autonomie et Solidarité, Actes architectures, Mission locale de Lille, Eurologiciel, DCA Architecture, Le Crête, Atout et Handicap, GKD...

c. Quelques chiffres

Au début de son ascension, vers l'an 2000, la société Urbilog avait un capital d'environ 120.000 €, qui est actuellement estimé à plus de 220.000 €. Son chiffre d'affaires a également évolué depuis 1995, en voici l'évolution de 2004 à 2009 :

- 2004 : 405 000 €
- 2005 : 530 000 €
- 2006 : 650 000 €
- 2007 : 750 000 €
- 2008 : 870 000 €
- 2009 : 700 000 €

2. Ses activités

La société Urbilog compte désormais cinq activités de compétences majeures.

a. Accessibilité du Web

L'accessibilité du web signifie que les personnes handicapées peuvent l'utiliser. Plus spécifiquement elle signifie que ces gens peuvent percevoir, comprendre, naviguer, interagir avec le web et y contribuer. L'accessibilité du Web bénéficie également à d'autres, notamment les personnes âgées ayant des capacités diminuées dues au vieillissement.

Elle est un droit universel, selon l'article neuf de la Convention relative aux droits des personnes handicapées adoptée en 2006 par l'Organisation des nations unies.

Il est important de souligner que les utilisateurs peuvent ne pas être en mesure de voir, d'entendre ou de bouger : ils peuvent utiliser un environnement différent du notre. Le traitement de l'information peut être donc altéré par la présence ou l'absence de certains périphériques chez l'utilisateur.

Urbilog développe donc des solutions logicielles destinées à accroître l'accessibilité du Web. Urbilog en partenariat avec France Télécom a conçu OCAWA, un outil d'audit et d'analyse de sites Web, indiquant le non-respect de règles d'accessibilité (définie par le WAI).

Urbilog propose des prestations d'expertise en accessibilité adaptées aux besoins spécifiques de ses clients tels que :

- Sensibilisation
- État des lieux, audit/conseil
- Recommandations, expertises (assistance)
- Formations, pérennisation

b. Réseaux informatiques

La société propose l'hébergement de sites Web, de domaines, de données et de machines. Cette activité reste cependant beaucoup plus « classique » et se retrouve dans beaucoup de SSII de la région.

Au niveau des prestations liées à l'installation de réseaux, Urbilog conçoit des architectures réseaux ou étudie le réseau existant, dans le but d'optimiser ses performances et d'améliorer l'échange des informations entre les postes. Elle propose une large gamme de serveurs « Clés en main » (Windows ou Linux). Urbilog opère également dans la création d'Extranet, pour que les entreprises et les clients privilégiés puissent rester connectés à leur réseau depuis n'importe quel point du globe. Ils ont ainsi accès aux informations les plus poussées tels que les tarifs et les promotions, ou un accès favorisé pour la prise de commande.

c. Développement Web

Urbilog propose tout type de sites Web : statiques, dynamiques, plates-formes de e-commerce, mais également des applications utilisables sur le Web. Les applications conçues permettent par exemple de gérer des commandes, des relations clients, mais aussi des cyberboutiques.

On peut citer parmi toutes ces applications : GRC-Plus, qui est un logiciel de Gestion de Relation Clients. Ce dernier permet de gérer les emplois du temps, les agendas et les carnets d'adresses de toutes les personnes d'une société.

d. Formations

Urbilog est depuis 1998 un organisme de formation déclaré auprès de la DRTEFP (Direction Régionale du Travail, de l'Emploi et de la Formation Professionnelle). Les formations développées par Urbilog sont essentiellement des formations professionnelles et peuvent se diviser en deux catégories : les formations aux nouvelles technologies d'un côté, et les formations bureautiques destinées aux déficients visuels de l'autre.

Les formations dispensées par Urbilog sont reconnues par les professionnels, si bien qu'en 2002, la société s'est vue décernée le trophée de la formation continue dans la catégorie « Insertion Réinsertion ».

e. Audit et conseils

Urbilog assiste des sociétés, et les aide à optimiser leurs outils de communications internes et externes. Ses experts et consultants assurent des prestations complètes en audits et préconisent le développement Web afin de garantir le respect des référentiels de l'accessibilité :

- WAI
- ADAE, RGAA
- Section 508
- Stanca
- BITV

De plus, Urbilog propose d'élaborer des chartes qualité/accessibilité, permettant d'associer les contraintes de l'accessibilité aux caractéristiques existantes d'un projet. Cette société permet également de mettre en place une démarche qualité/accessibilité au sein des systèmes d'informations, dans le but :

- d'améliorer les performances des outils de communication interactive
- d'optimiser le système d'informations en fonction des exigences métiers et des standards du Web
- de respecter les référentiels de l'accessibilité.

Urbilog conçoit des architectures de réseaux ou étudie le réseau existant dans le but d'optimiser ses performances et d'améliorer l'échange des informations entre les postes. Ceci concerne aussi bien la topologie que les équipements (câblage, hubs, switches, routeurs, serveurs d'accès, serveurs de terminaux). Elle propose son expérience, ses compétences et sa méthodologie pour la réussite des projets. Grâce aux outils d'audits et d'analyse développés par la Société, Urbilog assure rigueur et rapidité en toutes circonstances. L'audit peut être fait sur des sites Internet, des installations réseaux, des conduites de projets, etc.

3. Ses produits

Depuis sa création, la société Urbilog a mis en place différentes applications, liées à l'accessibilité.

a. Ocawa



Figure 1 : Logo d'Ocawa Web Accessibility Expert

Ocawa est un outil de contrôle en accessibilité développé en partenariat avec France Télécom.

Le principe de ce service est de pouvoir être utilisé via un serveur Web placé sur un réseau intranet ou directement sur internet. Il fonctionne sous Windows et

sous Linux, et s'appuie sur le serveur Web Apache. Sa commercialisation s'effectue sous deux formes différentes, en mode Web-service ou serveur dédié.

C'est l'un des outils les plus puissants du marché. Il se démarque des autres par l'intégration d'un système expert (base de l'intelligence artificielle) dans l'analyse de la qualité et de l'accessibilité d'un site Web. Cette particularité d'Ocawa permet de découpler la puissance de l'analyse, et le moteur d'inférence facilite la rédaction de nouveaux jeux de règles. Celle-ci se fait dans un langage spécifique XML (OKWML).

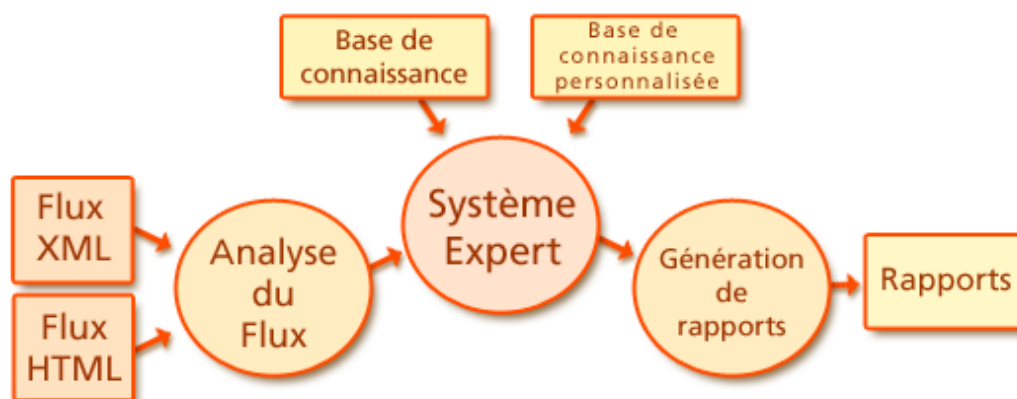


Figure 2 : Schéma de l'intégration du système expert

Ocawa intègre OcawaToolbar, une extension pour Mozilla Firefox en 2006 et en prévoit une pour le navigateur Chrome de Google. Ces extensions permettent de visualiser lors de la navigation sur un site Web quels sont les éléments qui ne respectent pas les référentiels d'accessibilité.

Les rapports d'audit retournés par Ocawa sont issus d'un flux XML pouvant être redéfini selon les besoins de l'utilisateur (les possibilités sont donc infinies). Ce flux de sortie étant générique il est possible d'utiliser les données de différentes façons. Il est par exemple possible de stocker l'ensemble des remarques dans une base de données ou de les visualiser sous la forme d'un rapport HTML structuré.

Les rapports Ocawa présentent d'abord une page statistique de l'audit, puis chaque page analysée est présentée de façon à ce que le ciblage de la balise posant problème soit simplifié.

b. Expert Fixer



Figure 3 : Logo d'Expert Fixer

Expert fixer est un outil d'aide au développement accessible développé en partenariat avec IBM EMEA. Il permet aux techniciens Web de contrôler le niveau d'accessibilité pendant la création des pages Web et d'y apporter les corrections éventuelles.

c. Clavier virtuel

Le clavier virtuel est un moyen d'accès pour toute personne à son ou ses comptes bancaires de façon sécurisé via Internet. Urbilog a développé une solution accessible aux personnes handicapées et aux seniors. Ce clavier virtuel peut être intégré dans la plupart des sites Web des organismes bancaires sans altérer le niveau de sécurité initial de l'application.

Accédez à l'espace sécurisé de BNPPARIBAS.NET :


1 Saisissez votre numéro client à l'aide du clavier

Numéro client :

2 Cliquez pour composer les 6 chiffres de votre code.

	2	5		8
7	6			
		0		
		1	9	
		3	4	

Code secret

 **Corriger**

3

Figure 4 : Illustration du clavier virtuel

d. Ranking

Le ranking est un service gratuit proposé par Urbilog qui propose un classement d'accessibilité des sites institutionnels français et étrangers des secteurs publics et privés. Il permet de :

- faire un point sur l'accessibilité des sites ;
- faire passer le message de l'accessibilité au grand public et aux décideurs des différentes structures ;
- fournir une ressource commune, gratuite et accessible pour déterminer l'état de l'environnement numérique Français et Internationale ;
- promouvoir les services proposés par Urbilog.

Le problème de ce service est que les audits se font manuellement et que les statistiques et classements ne sont pas assez mis en valeur.

Classement d'accessibilité (wcag10_3_fr) des Banques

Classement d'accessibilité (wcag10_3_fr) des Banques au 03-02-2009 sur 30 pages					
Pos		Banque	Err. Totales	Err. moyennes	Err. 1ère page
1	↗	Citibank	0	0.00	0
2	↗	BNP Paribas	0	0.00	0
3	↗	Société Générale	19	0.63	19
4	↘	Natwest	188	6.27	6
5	↘	Barclays France	200	6.67	9
6	↘	The Royal Bank of Scotland	262	8.73	6
7	↘	HSBC UK	267	8.90	13
8	↘	Le Crédit Lyonnais	300	10.00	259
9	↘	Crédit Agricole	404	13.47	10
10	↘	CIC	427	14.23	13
11	↘	Bank of Scotland	487	16.23	7
12	↘	Lloyds	514	17.13	30
13	↘	Halifax	524	17.47	3
14	↘	HSBC USA	551	18.37	40
15	↘	HSBC France	583	19.43	17
16	↘	Barclays UK	597	19.90	3
17	↗	Alliance- & Leicester	784	26.13	30
18	↘	Wachovia	863	28.77	33
19	↗	Chase Manhattan Bank	898	29.93	39
20	↘	Abbey	1074	35.80	13

Figure 5 : Illustration de l'application de Ranking

4. Organisation de l'entreprise

Vous trouverez ci-dessous l'organigramme de l'entreprise qui permet de voir la répartition des rôles et de l'autorité au sein d'Urbilog.

Organigramme

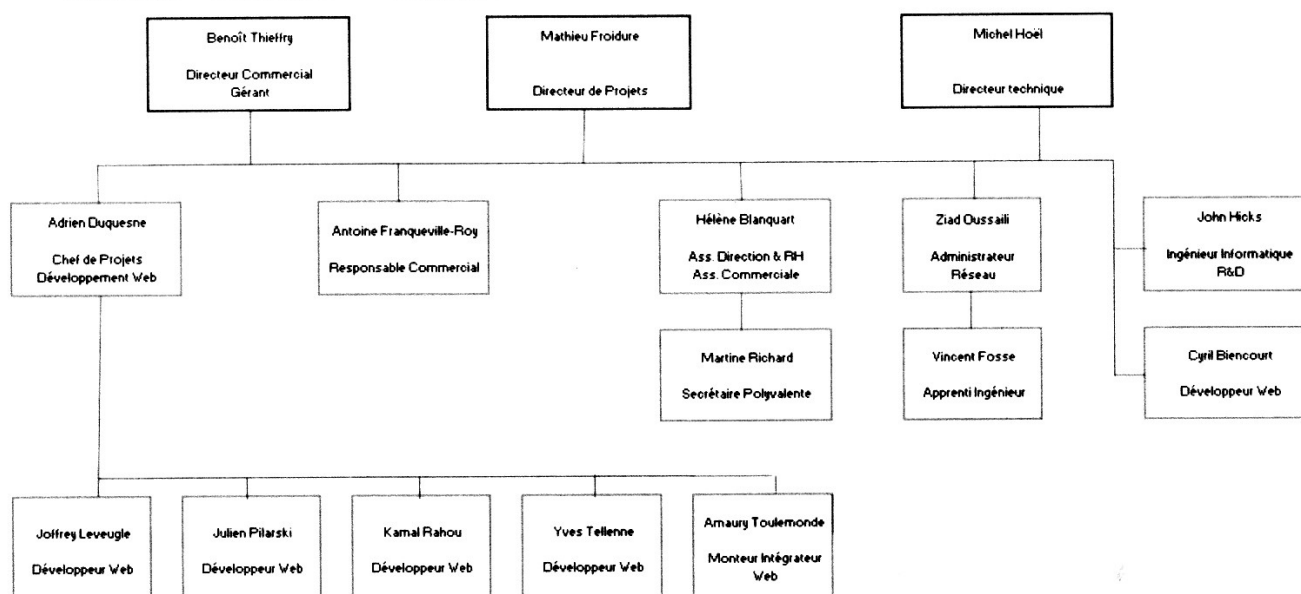


Figure 6 : Organigramme de l'entreprise

C. Le sujet du stage

1. Présentation

Comme vous le savez les logiciels tels que Microsoft Word, Microsoft PowerPoint, ou encore Open Office sont indispensables en entreprise, que ce soit pour écrire des rapports, de la documentation, préparer des présentations etc. L'idée d'écrire une API permettant la génération de document Open XML à partir de template est partie d'une constatation simple. En effet chez Urbilog le logo de l'entreprise a changé souvent au cours du temps, Urbilog effectuant régulièrement des interventions ou formations, cela prenait énormément de temps (et donc d'argent) à chaque fois de changer, dans tous leurs documents, uniquement le logo de l'entreprise généralement repris sur toutes les pages.

Ainsi l'idée de départ du projet a été de se dire, comment faire pour générer rapidement des documents sachant que le design (c'est-à-dire les types et tailles de polices utilisées, l'emplacement des informations, la taille des marges, les styles des paragraphes, etc.) du document risquait fortement de ne pas changer à l'inverse du contenu (les images, le texte, ...). D'où l'idée d'utiliser des templates dans lesquels on positionne des marqueurs qui seront remplacés par la suite par du contenu en provenance d'un ou plusieurs fichiers XML.

Ci-après nous détaillerons dans l'ordre l'objectif, à court et moyen terme du projet, les bases de l'application sur lesquelles j'ai pu m'appuyer pour mener à bien la mission, les outils qui m'ont permis d'exploiter au mieux les douze semaines de travail, le déroulement du stage par étapes chronologiques et les contraintes auxquelles j'ai été confronté.

2. Objectif

Des premières discussions autour du projet de ce stage, il est ressorti qu'il était illusoire de croire que l'on pourrait générer des documents parfaits dont la présentation n'exploserait pas et qui soit exactement bien formé dès le départ. L'objectif était pour l'api de réduire un maximum l'effort exigé pour produire des documents de traitement de texte bien formatés et que ce n'est pas grave si l'on doit passer cinq minutes pour corriger à la main les quelques défauts de présentation.

En effet certaines personnes sont réfractaires à l'idée de réaliser des documents, soit parce que cela requiert généralement une part de travail assez lourde qui n'est guère passionnant, soit parce qu'ils ne peuvent pas, comme Mathieu qui travaille à Urbilog. Mathieu est mal voyant et a besoin qu'une personne l'assiste pour qu'il crée ses slides dans Microsoft PowerPoint, car ce n'est pas un logiciel accessible. Ainsi maintenant grâce à l'api il lui suffirait de fournir un document texte, dont on extrairait les informations pour créer un fichier XML qui permettrait de créer automatiquement ses documents de présentation.

À court terme l'api pourra également être utilisé pour générer les rapports d'audit fournis par Ocawa. En-effet, actuellement les rapports d'audit sont déjà générés à partir de deux fichiers RDF et le design des audits ne change que très peu. Ce type de rapport se prête donc bien au mode de fonctionnement de l'API.

Enfin, nous nous sommes rendu compte qu'il n'existait pas énormément d'outils permettant de générer des documents à partir de template, c'est donc un marché ouvert où il nous reste donc à innover afin de poser les bases d'un nouveau type d'outils, de façon de penser. Nul doute que de nombreuses entreprises seraient intéressées.

3. Les outils

a. Technologies employées

Chez Urbilog la politique est qu'il n'existe pas de bon langage, tous ont leurs points faibles et leurs points forts, le tout est de choisir le bon. Le choix du langage de programmation pour l'API s'est porté sur le langage PHP pour plusieurs raisons.

La première est que tout le projet tourne autour de la manipulation de fichier XML et PHP est connu pour sa puissance et simplicité dans les traitements XML. PHP supporte DOM (Document Object Model), SAX (Simple API for XML) et SimpleXML (un outil permettant de manipuler très simplement des documents XML, basé sur le back-end de DOM). Il propose également un support natif pour SOAP et diverses fonctionnalités supplémentaires (transformations XSLT, XML-RPC, etc.).

La deuxième raison est que le temps imparti pour développer l'application était relativement court, les principaux avantages de PHP, c'est-à-dire sa simplicité, sa rapidité et sa fiabilité, que ce soit au niveau du développement ou de l'exécution, en font un langage très primé lors des développements de courte durée.

Un des objectifs était que l'on puisse ajouter facilement de nouvelles fonctionnalités à l'API, mais pour cela il faut que les gens comprennent le langage dans lequel le programme a été écrit. Et force est de constater que le PHP est probablement le seul langage de programmation que tout le monde connaît à peu près chez Urbilog. De plus depuis PHP 5, on dispose d'un langage objet qui permet également de faciliter la maintenance, la réutilisation et adaptation des classes.

Outre PHP, qui représente le langage de développement majeur du logiciel, je me suis appuyé sur une technologie « annexe » : CLIPS qui est un moteur d'inférence, dont je parlerais plus tard en détail, qui permet de conduire des raisonnements logiques et de dériver des conclusions à partir de faits que l'on émet.

b. Environnement de travail et organisation du travail

L'ensemble du développement a été réalisé sous notepad++ dont les principales forces sont la simplicité, rapidité, légèreté, coloration syntaxique et

possibilité de réduire le contenu d'une balise afin de clarifier le code. Il existe des IDE pour PHP néanmoins ceux-ci souffrent de quelques lourdeurs et les interfaces sont souvent surchargées, il n'est pas rare qu'une même fonction se retrouve dans plusieurs menus, une même fenêtre peut avoir plusieurs utilisations... et au moins avec notepad ++ on est obligé de connaître son code.

Sur le plan humain, le développement du stage s'est réalisé en monôme. Je n'ai donc pas eu de problème au niveau de la répartition des tâches, du partage et gestion de version des sources.

Pour d'autres raisons, le développement de l'API, s'est effectué de manière très libre, sans cahier des charges, en concertation avec mon tuteur d'entreprise. Bien sûr, cette « liberté » et ce permanent échange d'idées a participé pour beaucoup à l'évolution constante du projet.

c. Contraintes

L'une des principales contraintes de ce projet provient de l'utilisation de CLIPS et ce pour plusieurs raisons. La première est que CLIPS est basé sur la notion de faits et de règles, un concept tout à fait nouveau pour moi, couplé à une syntaxe pas forcément évidente, ce qui fait que j'ai dû passer quelques heures à lire la documentation avant de me lancer dedans. La deuxième est que CLIPS n'est pas complétement supporté nativement par PHP, néanmoins Michel, mon maître de stage a intégré lui-même clips à PHP et recompilé sa propre version de PHP. Néanmoins, c'est une version basée sur PHP 5.1.3, une ancienne version de PHP, ce qui m'a privé donc de toutes les nouveautés apportées par PHP 5.3.2 comme par exemple les fonctions de gestion de fichier zip très utiles pour dans le cas présent, puisque les documents Open XML sont des fichiers zip.

Les seules autres contraintes ont été des contraintes de performances, en effet l'API devait générer des documents rapidement, c'est-à-dire de l'ordre de la seconde même pour des documents conséquents. On revient au choix de l'utilisation de PHP, très rapide pour la manipulation XML et l'utilisation de CLIPS pour régler rapide tous les problèmes de mise en page.

Enfin l'utilisation de la version de PHP compiler par Michel et intégrant donc CLIPS, m'a contraint à utiliser PHP en ligne de commande, puisqu'elle ne permet de ne faire que ça. Ce qui implique que l'API sera principalement utilisée en mode batch.

D. L'existant... pas si existant

S'il aura un avenir, le développement de l'API possède aussi un précédent que l'on peut décomposer en trois parties : le format Open XML, des applications internes à Urbilog utilisées pour la manipulation de document et enfin un fichier XML d'exemple contenant les informations à insérer.

Dans leurs parties respectives, nous passerons en revue ces trois éléments fondateurs qu'il a fallu comprendre et intégrer avant de se lancer corps et âme dans le développement de l'api.

1. Détail de la structure OpenXML

Open XML est le nouveau format de fichier adopté par les documents de la suite Office, à partir de la version 2007. Ce format, fruit de la collaboration de Microsoft, d'Intel et d'Apple, entre autres, est totalement libre de royalties, et sa pérennité et son indépendance vis-à-vis de tout éditeur sont garanties par son élévation au rang de norme par l'ECMA.

Open XML rompt radicalement avec les formats binaires propriétaires des précédentes versions d'Office (jusqu'à la version 2003, qui a amorcé le tournant), en adoptant XML, langage ouvert, comme format de stockage. Il sera désormais possible aux développeurs de lire, créer, modifier, afficher sur différents médias des documents Office, sans dépendre d'applications Microsoft, en utilisant des outils comme XSLT, SAX ou DOM, directement ou par l'intermédiaire de bibliothèques Open XML.

Les documents Office Open XML sont des fichiers compressés selon le format Zip. On peut donc visualiser leur contenu en les décompressant à l'aide de n'importe quel utilitaire reconnaissant le format Zip. Un fichier Word 2007 dézippé présente la structure typique suivante:

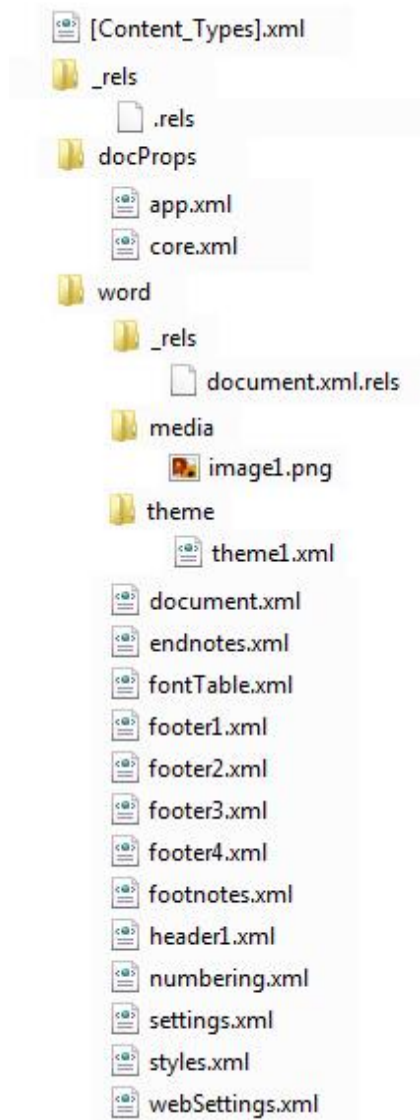


Figure 7 : arborescence typique d'un document Open XML

Une fois décompressé, un fichier Open XML révèle une kyrielle d'autres fichiers, l'ensemble de ces fichiers est appelé, dans la terminologie Open XML, un paquet. Ce paquet regroupe une collection de fichiers appelés parties, répartis dans une arborescence dont ils constituent les feuilles. Cette structure éclatée tranche avec le format monolithique employé par Office 2003, dans lequel l'intégralité du document se retrouvait au sein du seul et même fichier XML. Ici, nous avons affaire à une structure modulaire, chaque partie ou répertoire contenant des parties étant un élément du fichier ZIP.

Les parties qui composent un fichier Open XML peuvent être de deux catégories :

- Les données (texte, images, vidéos, etc.) qui constituent le document lui-même. Ces parties peuvent contenir des données définies dans Open XML, d'autres données XML dont le schéma ne fait pas partie des spécifications Open XML, des données binaires (objets OLE, images JPEG ou PNG, des vidéos AVI, etc.), du texte simple...

- Les données concernant la structure interne du paquet, notamment le type de contenu des autres parties, et les liens logiques qu'elles ont entre elles. Ces parties contiennent des données XML dont le schéma est défini par la norme Open XML (par les Open Packaging Conventions).

Les parties de la deuxième catégorie, qui contiennent les informations vont permettre de connaître précisément le rôle de chacune des autres parties du paquet, ainsi que les liens qui les unissent entre elles. Ces parties sont de deux types, le fichier des types de contenu et les fichiers de relations.

Les fichiers de relations permettent de retrouver toutes les données contenues dans la structure et d'y naviguer. Ils contiennent la description des liens entre les parties. Ces deux types de fichiers sont les clés de l'accès aux données du document Open XML. Ils vont permettre de recomposer le document dans son intégralité.

Le fichier central est sans aucun doute le fichier document.xml du dossier « word » qui représente le contenu du document final. Concernant la story principale d'un document, son contenu se trouve sous la balise body selon la structure suivante :

```
<w:document
xmlns:w="http://schemas.openXMLformats.org/wordprocessingml/2006/main">
  <w:body>
    <w:p>
      <w:r>
        <w:t>L'existant... pas si existant</w:t>
      </w:r>
    </w:p>
    <w:p>
      <w:r>
        <w:t>S'il aura un avenir, le développement de l'api possède aussi un
précédent que l'on peut décomposer en trois parties : le format Open XML, des
applications internes à Urbilog utilisées pour la manipulation de document et enfin un
fichier XML d'exemple contenant les informations à insérer.</w:t>
      </w:r>
    </w:p>
  </w:body>
</w:document>
```

Et équivaldrait dans Micro\$oft Word à cela :

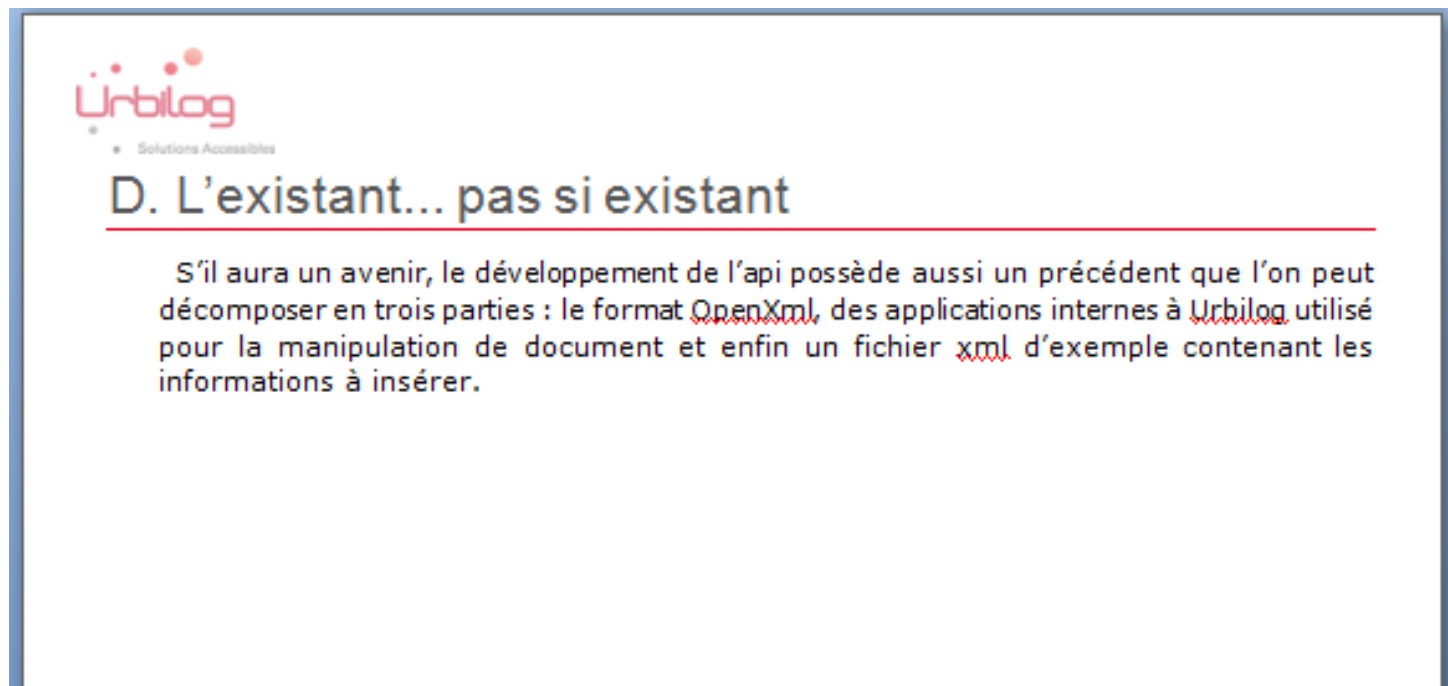


Figure 8 : équivalence entre le fichier document.xml et sa représentation dans Micro\$oft Word

L'exemple décrit un document possédant deux paragraphes. Voici le détail de cette structure :

- document : c'est l'élément racine du document principal.
- body : cet élément peut contenir plusieurs paragraphes.
- p : représente un paragraphe, et peut contenir un ou plusieurs run et des propriétés dans une balise fille pPr communes à tous les runs de ce paragraphe.
- r : représente un run, c'est-à-dire un ensemble contigu de caractères ayant les mêmes propriétés. Tout comme pour les paragraphes, chaque run peut posséder un élément fils rPr définissant les propriétés spécifiques de ce run. Nous reviendrons juste après sur ce concept fondamental de Open XML.
- t : le contenu textuel du run parent. Le contenu est principalement du texte non formaté. Le formatage de ce texte est hérité des propriétés du run et des propriétés du paragraphe. Cet élément contient aussi souvent l'attribut XMLspace="preserve" afin de bien spécifier la prise en compte des espaces dans le texte.

Dans la réalité le XML généré par Micro\$oft Word n'est pas aussi épuré et ressemblerait plutôt à cela :

```
- <w:p w:rsidR="009A4668" w:rsidRDefault="009A4668" w:rsidP="009A4668">
- <w:pPr>
  <w:pStyle w:val="URBTitre1" />
- <w:numPr>
  <w:ilvl w:val="0" />
  <w:numId w:val="22" />
</w:numPr>
</w:pPr>
<w:bookmarkStart w:id="0" w:name="_Toc264397998" />
- <w:r>
  <w:lastRenderedPageBreak />
  <w:t>L'existant... pas si existant</w:t>
</w:r>
<w:bookmarkEnd w:id="0" />
</w:p>
```

Figure 9 : morceau de code d'un fichier document.xml

2. Applications internes

Urbilog possède déjà quelques applications dont ils se servent pour générer des documents. L'une est actuellement utilisée pour générer les rapports d'audit OCAWA au format doc (Office 2003). Le format doc est tellement complexe, dans le sens où le fond et la forme ne sont absolument pas séparés, qu'au final l'application reste assez limitée.

Urbilog pense qu'il est utopique de penser qu'un programme peut à la fois tout faire et le faire bien. C'est dans cette optique que le projet Assigma a été développé, il permet en effet de convertir un grand nombre de fichiers, même les fichiers plats comme les fichiers texte, en fichier XML. La conversion s'effectue en deux parties, dans un premier temps le fichier est converti dans le format XML grâce à des moteurs déjà existants (comme l'activeX de Microsoft Word 2007) et dans un second temps on donne du sens au fichier grâce à une série de règles écrites par les employés d'Urbilog. En fait Urbilog est contre les usines à gaz et préfère avoir plusieurs applications qui font bien leur travail et qui passent le résultat d'une application à l'entrée de l'autre. C'est donc dans cette optique qu'Urbilog essaye de se construire un petit parc d'outils pour la manipulation de documents.

3. Fichier XML d'entrée

Vous l'aurez compris le projet Assigma, permet de s'affranchir du type des fichiers à supporter, en plus du fait que le format XML a prouvé sa grande force pour structurer des informations, c'est pour cela qu'il ne m'a été demandé de ne prendre en charge que les fichiers XML. C'est d'ailleurs la seule chose qui m'a été fournie en début du stage et dont je n'ai pas eu à inventer la structure. Ces fichiers sont des fichiers XML classiques dont voici un morceau de l'exemple qu'il m'a été fourni :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes" ?>
- <rgaa>
  <titre>Référentiel Général d'Accessibilité pour les Administrations</titre>
- <directive id="1">
  <titre>Fournir des alternatives équivalentes aux contenus visuels et auditifs.</titre>
  
- <pc id="1.1">
  <titre>Fournir une alternative textuelle aux éléments non textuels</titre>
- <description>
  - <libelle>
    <p>S'assurer que chaque élément non textuel (visuel ou auditif) possède un
      équivalent textuel, via l'attribut approprié dans son contenu ou dans le
      contenu adjacent à l'élément.</p>
    </libelle>
  - <obj_int>
    <p>Le contenu des éléments non textuels ne sera pas perceptible pour
      différentes catégories d'utilisateurs tels que :</p>
  - <ul>
    <li>Les personnes handicapées visuelles accédant au site via la synthèse
      vocale d'un lecteur d'écran,</li>
    <li>Les utilisateurs ayant désactivé l'affichage des éléments graphiques ou
      multimédias, pour réduire notamment les temps de téléchargement des
      pages en bas débit.</li>
    </ul>
    <p>La présence d'alternatives textuelles aux contenus non textuels permet
      également à des robots d'indexation d'en exploiter l'information, ou à des
      traducteurs automatiques d'en traduire le sens.</p>
    </obj_int>
    <wcag10 niv="A" />
    <reference type="WCAG 1.0" href="http://www.w3.org/TR/WAI-WEBCONTENT/#gl-
      provide-equivalents">directive 1</reference>
  - <reference type="DGME 1.0">
    <point>1.1</point>
    <point>1.2</point>
    <point>1.3</point>
```

Figure 10 : première exemple fourni

E. L'analyse

Urbilog est composé de deux départements, le pôle développement et le pôle Recherches et Développement. Pour ma part j'ai rejoint la R&D, comme son nom l'indique le but de ce département est d'accroître la somme des connaissances de l'entreprise ainsi que d'utiliser cette somme de connaissances pour de nouvelles applications. J'ai été surpris de voir l'importance du temps consacré aux recherches effectuées, que se soit sur Internet ou grâce à des magazines, dans le but de se tenir informé des dernières nouveautés technologiques, nouveaux standards ou tout simplement des derniers potins du web. Pour perpétuer la tradition, ma première tâche a été de voir s'il n'existait pas déjà des applications semblables au sujet de mon stage.

1. Des recherches et discussions obligatoires

Nous verrons ici, ce que le Web nous offre en terme d'outil de génération de document et l'importance des «réunions» effectuées pour encadrer le projet.

a. Des outils trop simples ou trop complexes

La génération de documents bien formatés avec PHP est une tâche plutôt difficile. Il existe habituellement deux approches principales. Avec suffisamment de temps et de patience, ces deux approches font l'affaire, mais laissent tout de même à désirer.

L'approche HTML vers le format de notre choix est très employée dans des applications traditionnelles. Dans ce cas, un document est programmé en HTML et converti dans le format de notre choix en utilisant une des nombreuses bibliothèques Open Source (domPDF, HTML_ToPDF, etc.). Cependant, le HTML n'est pas un format orienté page (tel le PDF) et donc ce n'est pas possible d'exécuter une conversion fidèle. Les caractéristiques typiques de format d'un fichier de traitement de texte, tels que les en-têtes et pieds de page, les veuves et orphelins ou encore la numérotation de pages ne peuvent tout simplement pas être représentés en HTML.

L'approche programmation offre un contrôle total sur le document résultant. Cependant, il faut que toutes les coordonnées x et y de chaque ligne de texte, les formes géométriques et les graphiques soient fixés par le code source. Non seulement, c'est une solution qui prend énormément de temps, mais elle est également très fragile : à chaque fois que la mise en page d'un document est changée, le programmeur doit retravailler son code source.

Il existe une troisième approche, elle repose sur des modèles qui ont été créés dans un environnement WYSIWYG, tels que Microsoft Word, puis chargés avec des données dans PHP. Le document résultant peut être enregistré dans de nombreux formats tels que le DOCX, DOC et RTF. Nous allons détailler les 2 principales solutions qui existent à l'heure actuelle, la première et la plus utilisée actuellement est LiveDocx.

LiveDocx est un service SOAP pour la génération de document basé sur les composants de traitement de texte. Il permet de remplir des modèles de traitement de texte avec des données dans tous les langages de programmation supportant SOAP (notamment PHP). Le document résultant peut être enregistré sous tout format supporté. Les principales forces de LiveDocx sont le nombre important de formats supportés que ce soit en entrée ou en sortie, une grande facilité d'utilisation, et la gratuité du service. Néanmoins cette simplicité d'utilisation est à double tranchant puisqu'elle limite grandement les possibilités de l'application. En effet il n'est possible d'insérer que du texte, et ce sont des remplacements du type un pour un, c'est-à-dire un marqueur est remplacé par une donnée textuelle.

Il existe une autre solution, TOPCASED GenDoc permet de générer des documents à partir de modèles et d'un document template. A l'inverse de LiveDocx, GenDoc ne permet pas de changer de format entre le document template et le document généré, en revanche l'application est bien moins limitée notamment grâce au fait que l'on peut mettre des informations dans les marqueurs. Par exemple le marqueur : `<gendoc:generatedFragment name='2' template= 'my_template' model='MyModel' />` fait référence à un modèle nommé 'MyModel', contenant les données à insérer, présent dans le document template 'my_template', idée que l'on a reprise.

Nous avons donc choisi de prendre le meilleur des deux, pour garder une certaine facilité d'utilisation, sans pour autant être limité.

Un autre type de recherches que j'ai eu à effectuer concerne la lecture de documentation, que ce soit la documentation de CLIPS, ou de la norme Open XML. En effet une grande partie du projet concerne l'injection de code à l'intérieur de documents Open XML. Ainsi pour pouvoir générer des documents valides il est nécessaire de connaître cette norme. Concernant CLIPS, c'était une manière de raisonner tout à fait nouvelle pour moi, dont je parlerais plus en détail plus tard.

b. Des discussions obligatoires

Tout au long du stage, afin d'organiser le travail, j'ai entretenu très régulièrement de longues et intenses discussions avec mon tuteur de stage, Michel, afin de parler des améliorations à apporter à l'API, de nouveaux concepts que l'on pourrait mettre en œuvre etc. Ces dernières ont aussi permis de faire le point sur l'évolution du projet.

Dès mon arrivée, la première chose dont nous avons parlé est la structure du template, puis les différents marqueurs qu'il serait intéressant d'avoir, les langages à utiliser et savoir quelle forme auraient les données.

Dans un premier temps, nous avons discuté de la structure des templates en général. Michel m'a donné un rapport d'audit OCAWA que j'ai transformé en y insérant différents marqueurs.

Voici la première version du template que j'ai effectué ainsi que la façon dont il a évolué au cours du temps grâce aux discussions que nous avons eu.

Niveau d'accessibilité {{{Urbi:text referentiel}}}

A. Pour l'ensemble des pages auditées

{{{Urbi:text pourcent}}}

B. Répartition par directive {{{Urbi:text referentiel}}}

{{{Urbi:text tabNum}}}	{{{Urbi:text tabText}}}
---------------------------	-------------------------

Figure 11 : template de première génération

Dans cette première version, nous nous contentions de faire des remplacements de type un pour un, c'est-à-dire un marqueur pour un contenu, la syntaxe restait de ce fait très simple. Quand au template qui suit il est un des derniers que j'ai réalisé.

{{{Urbi:text rgaa:path=<rgaa/directive/titre>}}}

{{{Urbi:loop rgaa:path=<rgaa/directive/pc>}}}

{{{Urbi:text rgaa:path=<rgaa/directive/pc/titre>}}}

{{{Urbi:text rgaa:path=<rgaa/directive/pc/description/libelle/p>}}}

{{{Urbi:loop rgaa:path=<rgaa/directive/pc/evaluation/test>
sort:string_desc=<titre>}}}

<pre> {{{Urbi:text rgaa:path=<rgaa/directi ve/pc/evaluation/test/titre >}}} </pre>	<pre> {{{Urbi:text rgaa:path=<rgaa/directive/p c/evaluation/test/champ_ap pli/p>}}} </pre>
--	--

{{{Urbi:endLoop}}}

{{{Urbi:endLoopRel}}}

{{{Urbi:endLoop}}}

{{{Urbi:endLoop}}}

{{{Urbi:endLoop}}}

Figure 12 : template de dernière génération

Vous avez certainement pu apercevoir une obscure syntaxe beaucoup plus riche, contenant des mots tels que, loop, path, sort. Ne vous inquiétez pas, les fondements de cette grammaire vous seront bientôt expliqués.

Au fil du temps nous avons choisi de retirer certains marqueurs, d'en ajouter d'autres, d'utiliser de nouveaux outils tels que CLIPS pour palier à certains problèmes. Ces discussions se sont donc révélées très intéressantes et ont contribué à l'évolution constante de l'API.

2. Analyse

Comme dit précédemment, le projet a largement évolué au cours du temps et ne ressemble plus du tout à ce qu'il était au départ. Ainsi l'analyse a été effectuée au fur et à mesure selon le principe des méthodes agiles. En effet le projet a avancé à vue ce qui a permis d'obtenir une grande souplesse et réactivité. De plus je pense que cela favorise la communication, ainsi, dès que quelque chose me bloquait, ou qu'il fallait revoir un concept, Michel était au courant très rapidement et nous pouvions réfléchir sur comment résoudre ce problème. Un des inconvénients majeurs de cette méthode est le manque d'une planification prévisionnelle ainsi que d'une analyse détaillée au départ.

Une des choses qui a largement facilité la modélisation de l'API est que le format docx et pptx sont très semblables, à part une petite différence au niveau du nom des namespaces utilisés par l'un ou par l'autre, on retrouve la même structure, les mêmes schémas. Dans la plupart des cas j'écrivais une classe abstraite contenant des fonctions communes aux deux formats et dans les classes qui en héritent, tout ce qui différait. La plus grosse différence résidait dans le fait que le contenu d'un document Microsoft Word est stocké dans un seul fichier, alors que dans le cas des documents Microsoft PowerPoint il y a un fichier XML par slide.

L'API est constitué de quatre packages principaux : transformer, document, utils, une série d'autre petit package. Pour détailler les différentes classes de ces packages, j'expliquerai d'abord les classes liées à la conversion des marqueurs mis en WYSIWYG.

a. Le package transformer

Les classes liées à la conversion des marqueurs ont été les premières classes que j'ai écrites. Nous nous sommes rendu compte que le XML généré par Microsoft Word était loin d'être épuré ainsi les marqueurs que je mettais sous Microsoft Word étaient difficilement manipulables puisque chaque marqueur se retrouvait réparti dans une multitude de balises XML, une première étape de conversion est donc nécessaire.

Dans l'exemple qui suit, le marqueur `{{{Urbi:text path=rgaa/directive/pc/evaluation/test}}}` se retrouve réparti dans deux runs différents.

```
- <w:r w:rsidRPr="00B66EF6">
- <w:rPr>
  <w:sz w:val="20" />
  <w:szCs w:val="20" />
  <w:lang w:val="en-US" />
</w:rPr>
<w:t>{{{{Urbi:text path=rgaa/directive/pc/evaluation/test/ </w:t>
</w:r>
- <w:r>
- <w:rPr>
  <w:sz w:val="20" />
  <w:szCs w:val="20" />
  <w:lang w:val="en-US" />
</w:rPr>
<w:t>titre}}}}</w:t>
</w:r>
```

Figure 13 : illustration d'une ligne se retrouvant dans deux runs différents

Le rôle des classes de ce package est donc de réunir chaque marqueur en une seule balise XML dont la syntaxe est la suivante :

```
<Urbi type='TYPE_DU_MARQUEUR' NOM_ATTRIBUT='VALEUR_ATTRIBUT'>>>
```

Dans l'exemple ci-dessus cela donnerait :

```
<Urbi type='text' path='rgaa/directive/pc/evaluation/test/>
```

En annexe 1 vous trouverez le diagramme de classe UML de ce package.

b. Le package document

Ce package regroupe principalement les fonctions permettant l'insertion des données dans le template de départ, c'est sans aucun doute la plus grosse partie de l'API. Le principe est simple, la méthode parsing du document va venir découper le document en une foultitude de document fils, chacun de ces fils peut avoir également des documents fils etc. Ces documents sont reliés entre eux par les managers présents dans le sous package manager et stockent les documents fils dans un tableau. L'avantage de cette méthode est que l'on peut maintenant trier et filtrer ces documents facilement, on peut par exemple trier par ordre alphabétique les sous documents sur leur titre (toutes ces informations doivent être précisées dans le template).

Une fois que la création des documents fils est finie, il suffit d'appeler la méthode «regenerate» qui va rassembler tous les documents entre eux en remplaçant au passage les marqueurs par les données en provenance des différents fichiers XML. En fait je me suis basé sur le principe du toString, mais en l'adaptant, c'est-à-dire que le premier document va appeler sa propre méthode toString qui va appeler la méthode toString de ses fils, et ainsi de suite, jusqu'à tomber sur des

feuilles c'est-à-dire un document qui n'a pas de fils. Ensuite cela va remonter récursivement une énorme chaîne de caractère par laquelle on va venir remplacer le contenu du document. Les classes s'occupant de faire ce remplacement s'appellent de la même manière que leur marqueur associé et sont chargés automatiquement par une factory.

En annexe 2 vous trouverez le diagramme de classe UML de ce package.

c. Le package utils

Comme son nom l'indique ce package est composé de classes contenant une pléthore de fonctions statiques utiles tout au long de l'API. Comme par exemple la classe ZipUtils qui permet de créer et décompresser des archives au format zip, puisque je vous le rappelle, les documents Open XML sont juste des fichiers zip renommés.

L'API ne manipule presque que des données XML, ainsi il fallait trouver un moyen efficace de traiter les fichiers XML. Cela passe par la classe FileUtils qui s'occupe de charger un document XML et de renvoyer un objet XmlDocument permettant de manipuler ce fichier. Le traitement n'est malheureusement pas instantané, me servant plusieurs fois des mêmes fichiers XML, une astuce pleine de bon sens a été de mettre dans un tableau tout les documents ouverts. Ainsi lorsque l'on demande à la fonction de parser le fichier XML elle regarde d'abord s'il n'est pas déjà chargé. Un gros travail d'optimisation a été réalisé afin de garder des performances respectables, en effet la moindre fonction mal écrite peut provoquer une perte de performances importantes sur les gros documents.

La classe AlgoUtils contient toutes sortes de fonctions qui effectuent des traitements algorithmiques précis, j'ai par exemple recodé un quicksort, que j'ai renommé brocoliSort, puisque celui d'origine ne me convenait pas.

En annexe 3 vous trouverez le diagramme de classe UML de ce package.

d. Les autres packages

Ce sont des packages qui contiennent peu d'objets, mais qui sont très utiles, comme le package log qui permet de créer des fichiers de log aussi bien dans le format html, dans un fichier texte ou encore dans un document Open XML, afin par exemple d'insérer à la fin d'un document généré grâce à l'api, un rapport de génération avec différentes remarques. La classe CLIPS permet de faire le pont entre le moteur d'inférence et PHP et TxtToXML permet de transformer un fichier texte en un fichier XML, afin de patienter en attendant que le projet Assigma soit fini.

En annexe 4 vous trouverez le diagramme de classe UML de ces package.

F.Utilisation

Dans cette partie je comptai expliquer comment se servir de l'API, comment créer un template, comment faire de petites modifications au niveau de CLIPS. Néanmoins cela correspond à la documentation utilisateur que j'ai eu à réaliser lors du stage, vous trouverez donc votre bonheur en annexe 5.

G. Conclusion

Mon stage au sein de la société Urbilog, a été très enrichissant, d'une part pour l'aspect technique du sujet, j'ai en effet pu améliorer mes connaissances dans le langage PHP notamment dans la conception orientée objet. D'autre part, j'ai pu me rendre compte de la difficulté de mener à bien un projet du début à la fin sans analyse fixe au départ. Je regrette néanmoins de n'avoir pas pu mettre en pratique, lors de ce stage, ce que l'entreprise m'a appris sur l'accessibilité.

L'état actuel de l'API me permet de dire que j'ai répondu aux attentes d'Urbilog. Même si je souhaiterais aller plus loin dans les fonctionnalités, l'API est fonctionnelle et peut être utilisée dès aujourd'hui. Quelques semaines supplémentaires n'auraient pas été du luxe afin d'avoir les mêmes fonctionnalités dans la gestion des documents Microsoft Word et PowerPoint. Néanmoins, la documentation que j'ai écrite devrait permettre de finir ce travail sans trop de difficultés.

Enfin je suis très content de ce stage car je n'aurais jamais cru arriver à réaliser quelque chose d'une telle envergure, c'est sans aucun doute grâce à Michel qui a astucieusement présenté le sujet en le faisant évoluer au fur et à mesure, toujours en me poussant vers le haut.

H. Bibliographie et Webographie

Bibliographie :

- Office Open XML Part 4: Markup Language Reference
- CLIPS Online Documentation

Webographie :

- <http://www.php.net>
- <http://www.developpez.com/>

I. Mots-clés et keywords

Mots-clés : API, Open XML, Word, PowerPoint, modèle, PHP, XML, génération, CLIPS, programmation orientée objet, Urbilog, accessibilité

Keywords : API, Open XML, Word, PowerPoint, template, PHP, XML, generation, CLIPS, object-oriented programming, Urbilog, accessibility

J. Glossaire

ActiveX : Solution de création d'applications développée par Microsoft. Les principes de fonctionnement sont très semblables au fonctionnement des applets.

API : Une interface de programmation (*Application Programming Interface* ou *API*) est un ensemble de fonctions, procédures ou classes mises à disposition des programmes informatiques.

Docx, pptx, odt : suffixe des fichiers des principaux logiciels de traitements de texte

DOM : Le Document Object Model (ou DOM) est une recommandation du W3C qui décrit une interface indépendante de tout langage de programmation, permettant à des programmes informatiques et à des scripts d'accéder ou de mettre à jour le contenu, la structure ou le style de documents.

Factory : design pattern qui a pour rôle de charger les classes qu'on lui passe en argument. Ainsi, quand on effectue une modification dans son code, il suffit de modifier le masque d'usine pour que la plupart des modifications prennent effet.

Modèle : le terme modèle (template en anglais) renvoie au fichier d'entrée contenant les marqueurs. Les modèles peuvent être dans le format docx ou pptx.

Modèle MVC : Le Modèle-Vue-Contrôleur (en abrégé MVC, de l'anglais *Model-View-Controller*) est une architecture et une méthode de conception qui organise l'interface homme machine d'une application logicielle. Ce paradigme divise l'IHM en un modèle (modèle de données), une vue (présentation, interface utilisateur) et un contrôleur (logique de contrôle, gestion des événements, synchronisation), chacun ayant un rôle précis dans l'interface.

Namespace : Le terme espace de noms (*namespace*) désigne en informatique un lieu abstrait conçu pour accueillir des ensembles de termes appartenant à un même répertoire. Les technologies XML font un usage intensif des espaces de noms, car les documents XML doivent avoir la capacité de mélanger sans ambiguïté plusieurs ontologies, c'est-à-dire plusieurs vocabulaires distincts qui peuvent contenir des homonymes.

Package : D'une manière générale, le mot paquetage désigne un ensemble de plusieurs entités regroupées en une seule. En informatique c'est une archive comprenant les fichiers informatiques, les informations et procédures nécessaires à l'utilisation d'une application.

PHP : langage de scripts libre principalement utilisé pour produire des pages Web dynamiques, mais pouvant également fonctionner comme n'importe quel langage interprété de façon locale, en exécutant les programmes en ligne de commande.

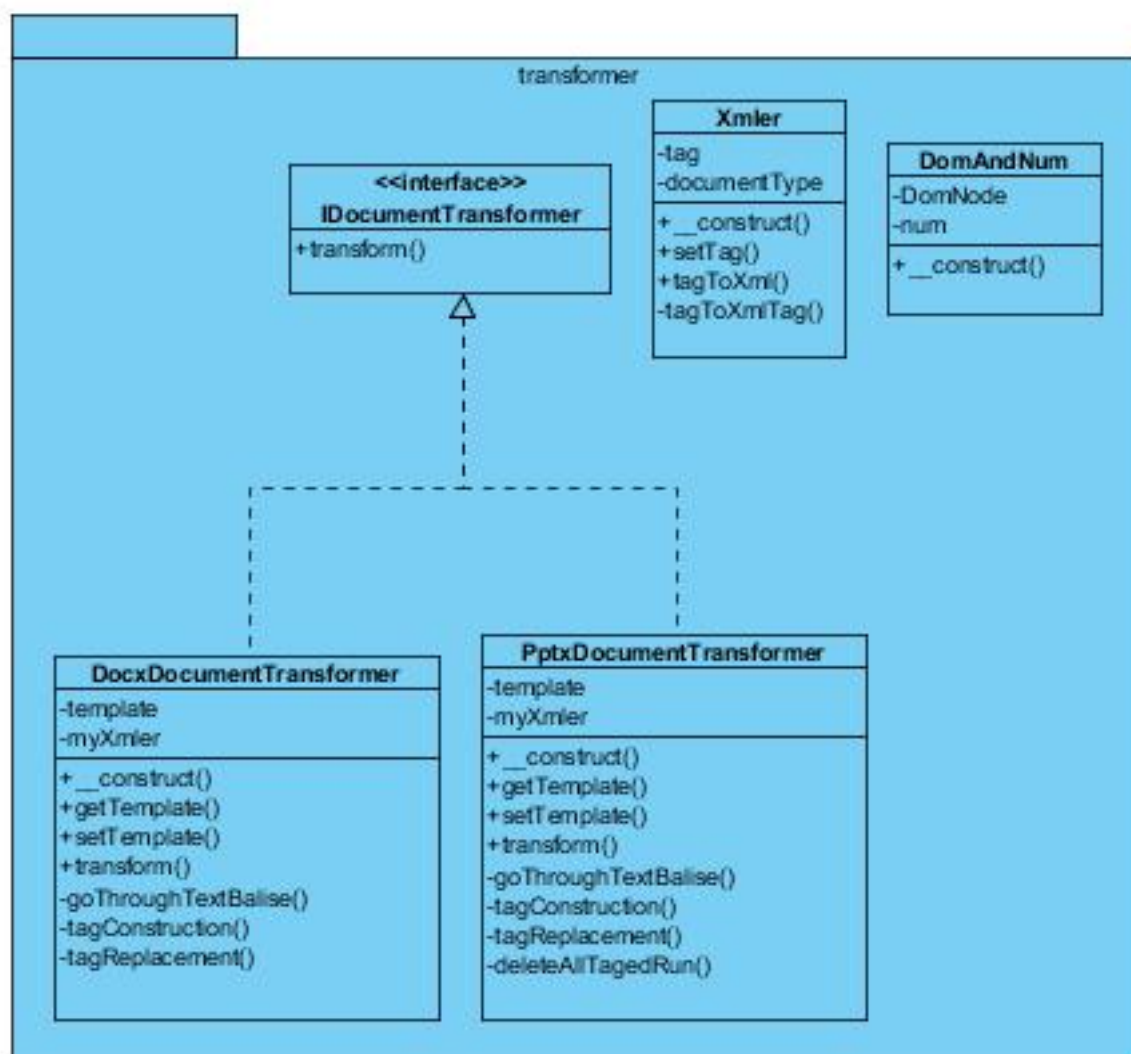
UML : Unified Modeling Language, est un langage graphique de modélisation de données et des traitements.

XML : *Extensible Markup Language* (français : « langage extensible de balisage ») est un langage informatique de balisage générique. Il sert essentiellement à stocker/transférer des données de type texte structurées en champs arborescents.

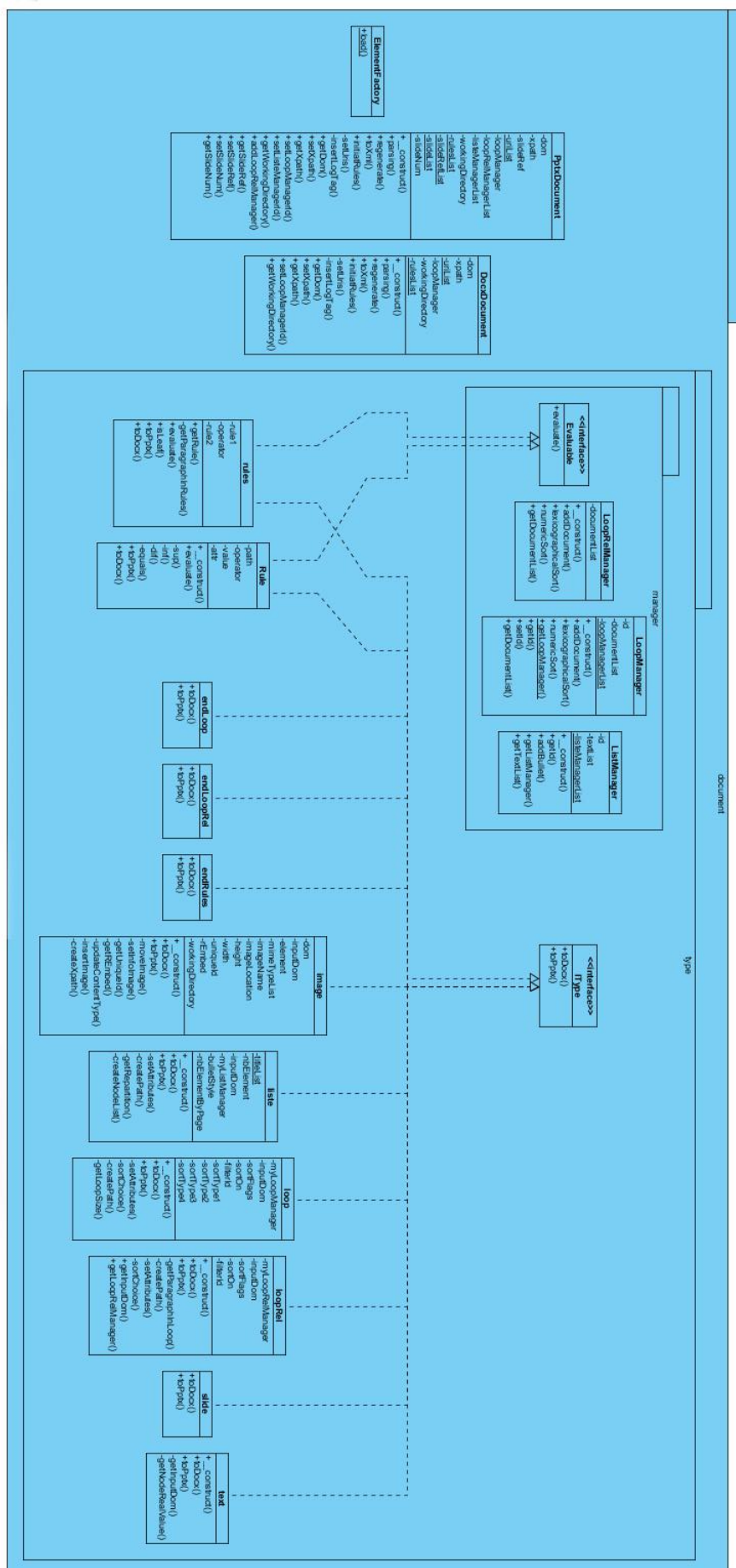
WYSIWYG : « WYSIWYG » est l'acronyme de la locution anglaise « *What you see is what you get* », signifiant littéralement en français « ce que vous voyez est ce que vous obtenez ».

K. Annexes

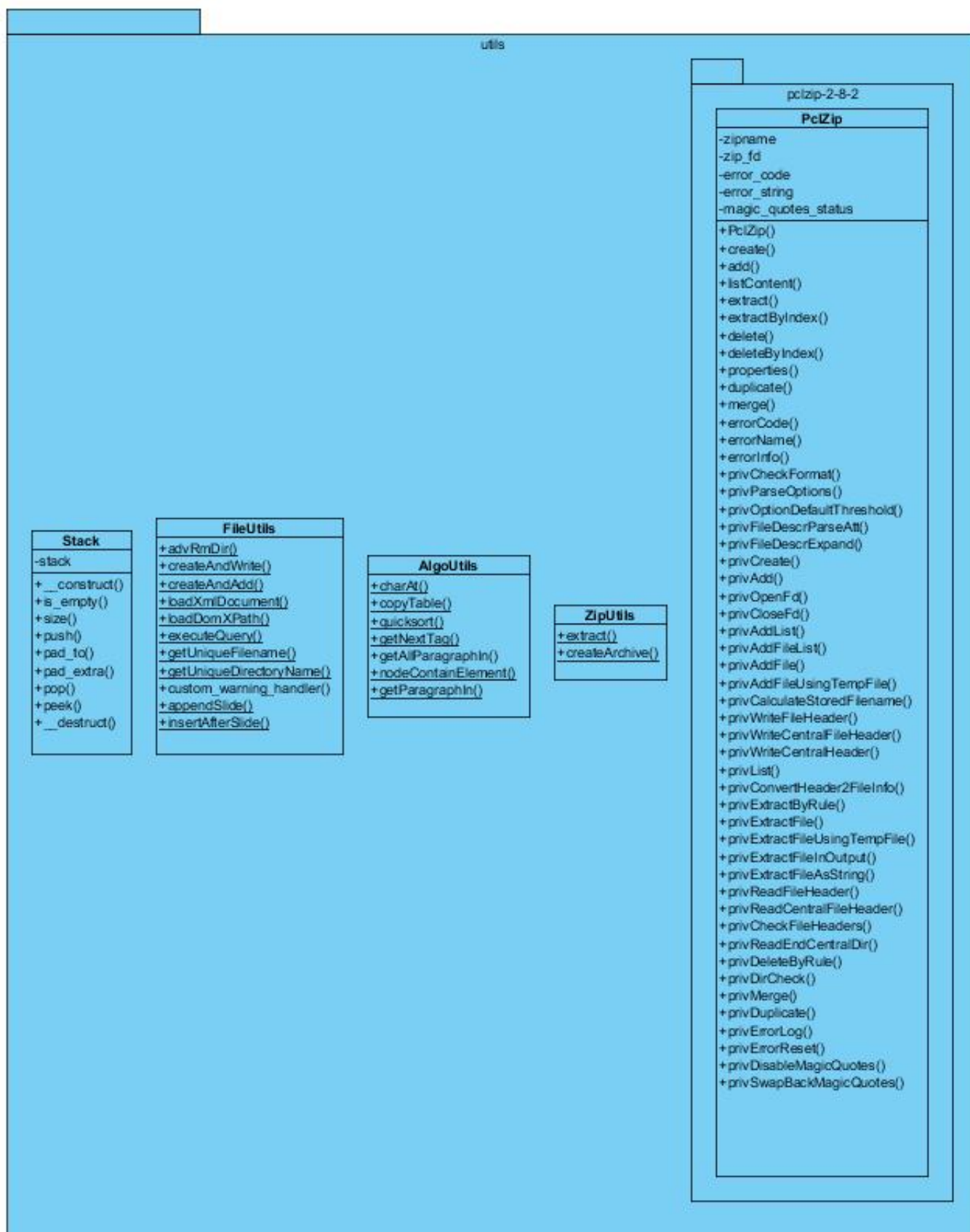
1. Annexe 1 : diagramme de classes UML du package transformer



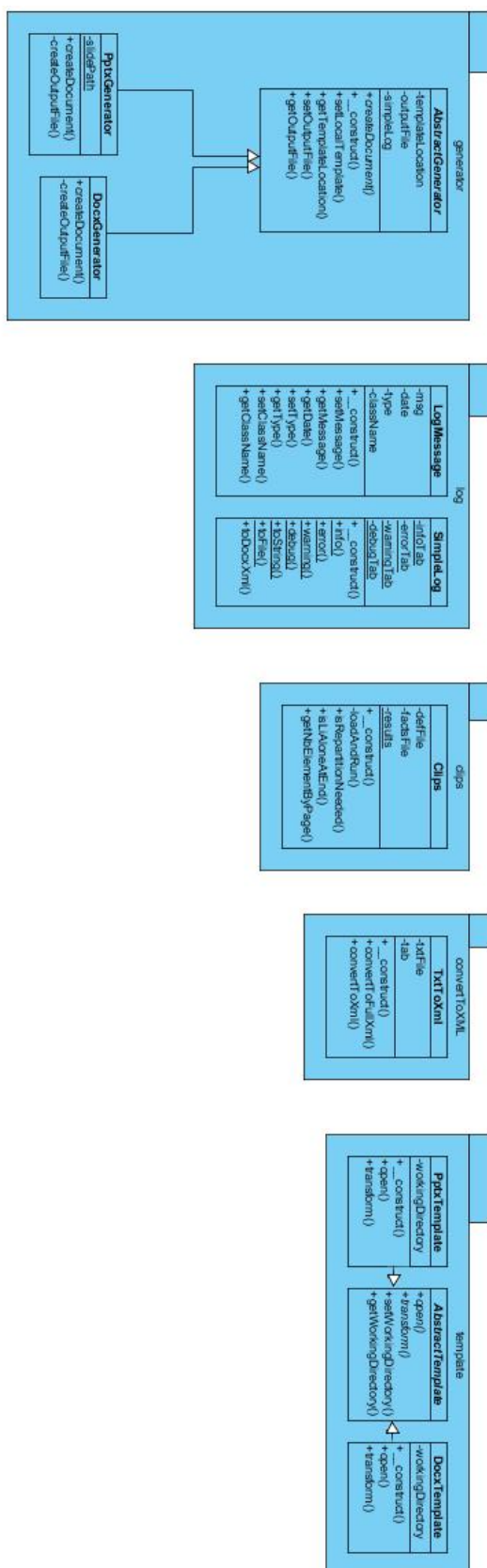
2. Annexe 2 : diagramme de classes UML du package document



3. Annexe 3 : diagramme de classes UML du package utils



4. Annexe 4 : diagramme de classes UML des autres packages



Documentation utilisateur

API PHP 5 pour la génération de document OpenXML

Rédacteur : Loïc Torrez

SOMMAIRE

A. DESCRIPTION DE L'API	42
1. Le template	42
2. CLIPS.....	42
3. L'api	43
B. DOCUMENTATION UTILISATEUR	44
1. Création d'un template	44
a.La balise image	47
b.La balise liste	48
c.La balise loop	48
d.La balise loopRel	50
e.La balise rules	51
f.La balise slide.....	53
g.La balise text	54
2. Modification des règles CLIPS	55

A. Description de l'API

Cette API permet de faire gagner du temps dans la création de documents Word et PowerPoint. En effet, elle contient un ensemble de fonctions permettant de générer automatiquement des documents en fusionnant un document template et des données provenant d'un ou plusieurs fichiers XML.

Afin d'acquérir une certaine flexibilité tout en gardant un grand nombre de fonctionnalités, l'application se découpe en trois parties : l'API, le template et CLIPS. Ce mode de fonctionnement peut être associé à un modèle MVC où le modèle serait le template, la vue CLIPS et le contrôleur l'API.

Dans la partie utilisateur, je m'efforcerai d'expliquer comment se servir de l'API et quant à la partie technique, elle sera consacrée à l'explication du fonctionnement de l'API en elle-même ainsi qu'à la façon d'ajouter facilement de nouvelles fonctionnalités.

1. Le template

Le but du template est de s'occuper de la partie présentation, c'est-à-dire stocker les styles du document, comme par exemple mettre un pied de page, un sommaire, un style particulier pour les titres de niveau deux, définir la taille des marges, en somme toutes les choses qui peuvent se faire automatiquement. Afin que l'API comprenne où et comment insérer des informations il est nécessaire d'utiliser des marqueurs. Afin de ne pas être limité à de simples remplacements, une grammaire a été élaborée permettant de réaliser des choses un peu plus complexes telles que des boucles.

2. CLIPS

CLIPS (*C Language Integrated Production System*) est un environnement et un langage de programmation créés en 1985, faisant partie du paradigme des langages déclaratifs et logiques.

Il s'agit avant tout d'un outil de construction de systèmes experts à base de règles et d'objets. Ses caractéristiques notables sont :

- La gestion de trois paradigmes à la fois : programmation objet, programmation procédurale et programmation par règles.
- Une grande flexibilité dans la représentation des connaissances, grâce à ces trois paradigmes.
- Portabilité et rapidité : il est écrit en C.

- Intégration : CLIPS peut être embarqué dans des logiciels écrits dans d'autres langages.
- Extensibilité : ces autres langages peuvent servir à étendre ses fonctionnalités en respectant certains protocoles.
- Interactivité avec l'interpréteur de commandes.
- Fonctionnalités poussées de validation de systèmes experts, de vérification de contraintes.

Dans notre cas, nous utilisons CLIPS afin d'améliorer, de personnaliser la façon dont sont affichées les choses.

3. L'API

L'API est écrite en PHP 5.1.3 pour plusieurs raisons, la première est que le PHP est un langage simple que la plupart des gens connaissent et qu'il est connu pour sa puissance et simplicité dans les traitements XML. Quand au choix de la version il était imposé, en effet afin d'utiliser CLIPS il faut se servir d'une version spécifique de PHP, compilé par Michel et intégrant nativement CLIPS. L'API est écrite en objet et se compose de plus de 45 classes réparties dans quatre principaux packages : transformer, document, utils et une série d'autres petits packages.

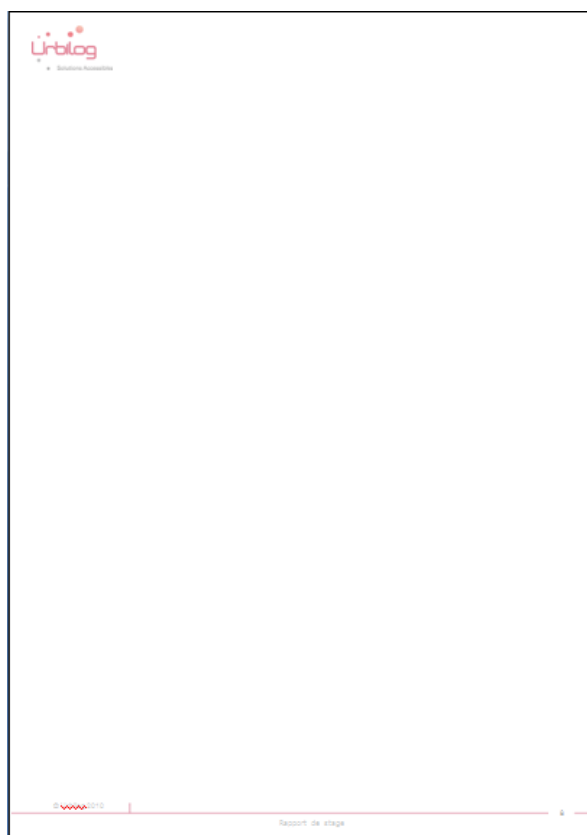
B. Documentation utilisateur

Cette partie va vous permettre de vous servir de l'API pour générer des documents Word et PowerPoint. Comme dit précédemment l'API est basé sur un pseudo MVC. Le but de ce mécanisme est qu'une fois le projet fini, on touche le moins possible à l'API et qu'il soit quand même possible de changer de manière significative la façon dont sont générés les documents, soit en modifiant le template, soit les règles.

1. Création d'un template

Le premier pas dans la génération d'un document est la création d'un template. Pour cela, vous pouvez utiliser soit Open Office ou Microsoft Word. Dans notre cas, nous allons utiliser Microsoft Word 2007.

Démarrez en créant un nouveau fichier dans Microsoft Word 2007 et enregistrez ce template sous le nom de votre choix. Vous pouvez alors commencer à concevoir le modèle en y insérant du texte, un sommaire, un pied de page, un logo repris sur chaque page, définir un style pour les différents éléments du document, etc.



Au bout d'un moment vous devriez avoir un template contenant tous les styles du document final.

La prochaine étape est l'insertion de marqueurs qui seront remplacés par des données. La capture d'écran suivante vous montre à quoi peut ressembler un template fini, il nous servira d'exemple tout au long de cette documentation.



• Solutions Accessibles

```
{{{Urbi:loop rapport:uri=</cygdrive/c/wamp/www/v3/usercases/models/rapport.xml>
rapport:path=<rapport>}}}
```

```
{{{Urbi:text rapport:path=</rapport/titre>}}}
```

SOMMAIRE

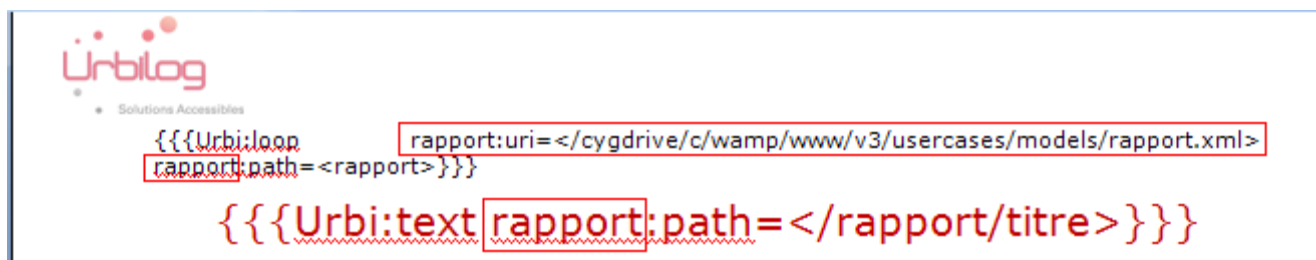
```
{{{Urbi:loop rapport:path=</rapport/sommaire/chapitre>
sort:string_asc=</rapport/sommaire/chapitre/titre> filter=<r1>}}}
    • {{{Urbi:text rapport:path=</rapport/sommaire/titre>}}}
    {{{Urbi:endLoop}}}
    {{{Urbi:image rapport:path=</rapport/sommaire/img>}}}
    {{{Urbi:endLoop}}}

{{{Urbi:rules id=<r1>}}}
    {{{Urbi:rule path=</titre> operator=<\<> value=<toto>}}}
    and
    {{{Urbi:rules}}}
        {{{Urbi:rule path=</titre> operator=<!=> value=</.Microsoft./> }}}
        and
        {{{Urbi:rule path=</description> operator=<=> value=<1> attr=<id>}}}
    {{{Urbi:endRules}}}
{{{Urbi:endRules}}}
```

Ces marqueurs respectent une grammaire que nous avons inventée, pour comprendre cette grammaire il est nécessaire dans un premier temps de définir les différents marqueurs pouvant être utilisés.

Mais avant toute chose il est utile de faire quelques remarques concernant la syntaxe générale du template. En effet, vous l'aurez peut-être remarqué, mais nous n'utilisons pas les caractères " et " comme délimiteur de chaîne de caractère, mais les caractères < et >. Si vous voulez utiliser ces caractères pour autre chose que délimiter une chaîne, il suffit de mettre un « \ » devant pour le protéger. Dans la plupart des balises on doit préciser le chemin du fichier XML où se trouvent les données à insérer.

Pour éviter de réécrire le chemin complet à chaque fois, il est possible d'utiliser un alias que l'on met dans la première balise du document, la syntaxe pour cela est la suivante : **ALIAS:uri=<CHEMIN_DU_FICHIER>**.



Dans l'exemple ci-dessus je définis l'alias rapport qui correspond au fichier XML dont le chemin est /cygdrive/c/wamp/www/v3/usercases/models/rapport.XML puis j'utilise cet alias à deux reprises.

Et enfin, parfaite transition pour vous parler de la notation des chemins, partout dans l'API vous devez utiliser la notation cygdrive, c'est-à-dire séparer les répertoires par des « / » et les chemins doivent commencer par « /cygdrive ».

Je présenterai les marqueurs qui ont le plus d'intérêt sous la forme suivante : la syntaxe, suivie d'une brève description, la liste des options disponibles et souvent, un exemple d'utilisation. Par convention je mettrai en majuscule et en gras les choses qui ne sont pas fixes.

La majorité des marqueurs sont à la fois disponibles pour les docx et pptx, néanmoins ce n'est pas le cas de tous, soit par manque de temps soit parce que ce sont des balises spécifiques à un format. Par exemple cela n'aurait aucun sens d'avoir une balise slide pour les documents Word. Voici donc un tableau récapitulatif des types de marqueurs supportés avec leurs attributs disponibles :

Nom du marqueur	Attributs supportés pour les docx	Attributs supportés pour les pptx
Image	path	path
List	Balise non supportée	path
Loop	path, sort, filter	path, sort
loopRel	Balise non supportée	path, sort
Rules	id	Balise non supportée
Slide	Balise non supportée	id
Text	path	path

a. La balise image

Syntaxe :

```
{{{Urbi:image ALIAS:path=<XPATH>}}}
```

Description :

Comme son nom l'indique cette balise permet d'insérer des images, l'image est positionnée au même endroit que le marqueur qu'elle vient remplacer, de plus elle garde ses dimensions initiales. Seules les images au format png et gif sont supportées pour le moment. Si vous souhaitez insérer une image d'un autre format il vous faut dans un premier temps la convertir avec par exemple Paint ou Photoshop.

Attributs :

alias = alias du fichier XML dans lequel se trouve les informations à insérer

xpath = xpath permettant d'aller chercher l'information dans le fichier XML précisé juste avant (vous pouvez donc utiliser toutes les règles de xpath afin de sélectionner un nœud particulier)

Exemple :

Reprenons notre exemple cité plus haut

```
{{{Urbi:endLoop}}}
{{{Urbi:image rapport:path=</rapport/sommaire/img>}}}
{{{Urbi:endLoop}}}
```

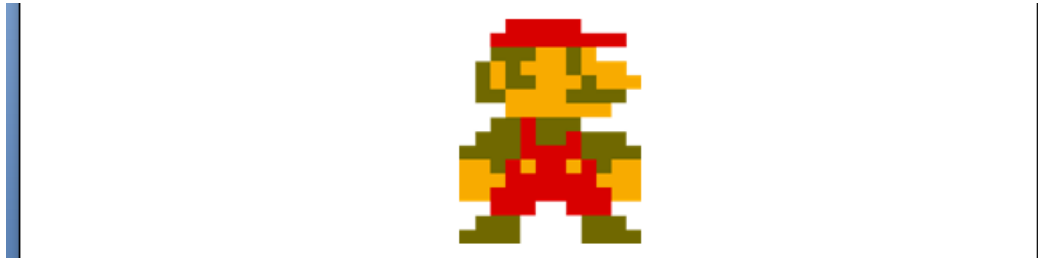
Ce marqueur sera remplacé par une image dont le chemin peut être retrouvé grâce au xpath « /rapport/sommaire/img » dans le fichier XML qui a pour alias rapport.

```

1  <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
2  <rapport>
3      <titre>Documentation utilisateur</titre>
4      <sommaire>
5          
6          <chapitre id="1">
7              <titre>DESCRIPTION DE L'API</titre>
8          </chapitre>
9          <chapitre id="2">
10             <titre>DOCUMENTATION UTILISATEUR</titre>
11         </chapitre>
12         <chapitre id="3">
13             <titre>DOCUMENTATION TECHNIQUE</titre>
14         </chapitre>
15     </sommaire>
16 </rapport>

```

Produisant ainsi le morceau de document suivant :



b. La balise liste

Syntaxe :

```
{{{Urbi:liste ALIAS:path=<XPATH>}}}
```

Description :

La balise liste permet de créer ... des listes. Cette balise s'utilise exactement comme la balise image, à l'exception que le xpath donné doit pointer sur un nœud dont les fils constitueront la liste. C'est une balise spécialement conçue pour les documents PowerPoint, qui créent automatiquement de nouvelles slides si la liste est trop grande pour tenir sur une seule slide.

Attributs :

alias = alias du fichier XML dans lequel se trouve les informations à insérer

xpath = xpath permettant d'aller chercher l'information dans le fichier XML précisé juste avant. Dans le cas d'une liste le xpath doit mener à un nœud dont les fils constitueront la liste. Chacun de ces fils doit posséder un attribut « lvl » représentant le niveau de profondeur du texte, si celui-ci n'est pas présent la valeur par défaut sera prise, c'est-à-dire zéro.

c. La balise loop

Syntaxe :

```
{{{Urbi:loop ALIAS:path=<XPATH> sort:TYPE_DE_TRI=<XPATH_2>
```

```
Filter=<ID_DU_FILTRE>}}}
```

Description :

La balise loop permet de faire des boucles, c'est-à-dire répéter plusieurs fois une information. Cela peut paraître sans grand intérêt mais c'est l'une des balises les plus complexes de l'api qui permet de générer des documents de plusieurs dizaines de pages à partir d'un template de 10 lignes. La balise permettant de marquer la fin d'une boucle s'appelle « endLoop », le principe est que l'on va prendre tout ce qui se trouve dans la boucle et le répéter un certain nombre de fois en changeant le xpath suivant notre position dans la boucle.

Attributs :

alias = alias du fichier XML dans lequel se trouve les informations à insérer

xpath = xpath permettant d'aller chercher l'information dans le fichier XML précisé juste avant. Dans le cas d'une boucle il permet de connaître la taille de la boucle et ainsi de faire varier le xpath au cours du temps.

Attributs optionnels :

TYPE_DE_TRI et XPATH_2 = il est possible de trier les groupes d'éléments répétés suivant l'ordre alphabétique ou numérique sur un nœud précisé par XPATH_2. TYPE_DE_TRI peut prendre comme valeur « string_asc », « string_desc », « numeric_asc » et « numeric_desc ».

Id_du_filtre = il est également possible de filtrer les groupes d'éléments répétés, c'est-à-dire ne pas afficher ceux qui ne respectent pas la règle dont l'identifiant est spécifié. Pour plus d'information voir les marqueurs « rules » et « rule ».

Exemple :

Reprenons notre fameux exemple

SOMMAIRE

```

{{{Urbil:loop rapport:path=</rapport/sommaire/chapitre>
sort:string_asc=</rapport/sommaire/chapitre/titre> filter=<r1>}}}}
    • {{{Urbil:text rapport:path=</rapport/sommaire/titre>}}}}
{{{Urbil:endLoop}}}}

```

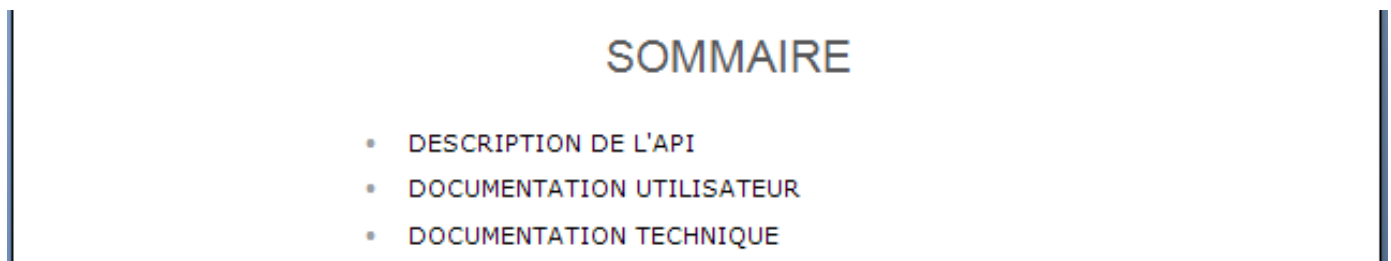

Ces balises indiquent à l'API qu'il faut répéter tout ce qui se trouve entre le marqueur « loop » et le marqueur « endLoop ». Si l'on regarde le fichier XML associé, on observe que le xpath « /rapport/sommaire/chapitre » cible plusieurs nœuds, trois pour être exact.

```

1  <?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
2  <rapport>
3    <titre>Documentation utilisateur</titre>
4    <sommaire>
5      
6      <chapitre id="1">
7        <titre>DESCRIPTION DE L'API</titre>
8      </chapitre>
9      <chapitre id="2">
10       <titre>DOCUMENTATION UTILISATEUR</titre>
11     </chapitre>
12     <chapitre id="3">
13       <titre>DOCUMENTATION TECHNIQUE</titre>
14     </chapitre>
15   </sommaire>
16 </rapport>

```

Tout ce qui se trouve à l'intérieur de la boucle sera donc répété trois fois, la seule différence sera le xpath qui vaudra à tour de rôle « /rapport/sommaire/chapitre[1] », « /rapport/sommaire/chapitre[2] » et enfin « /rapport/sommaire/chapitre[3] », produisant le résultat suivant :



Là où cette balise devient extrêmement puissante est qu'à l'intérieur des boucles on peut mettre toutes les autres balises, y compris des boucles ou encore du contenu fixe qui sera répété comme une puce donnant l'illusion de créer une liste.

d.La balise loopRel

Syntaxe :

```
{{{Urbi:loopRel FICHIER:path=<XPATH> id=<NUM_SLIDE>}}}
```

Description :

Cette balise se comporte exactement comme une boucle normale à l'exception qu'au lieu de répéter ce qui se trouve à l'intérieur de la boucle celle-ci répète une slide. Pour connaître la taille de la boucle, l'API se base sur le nombre de nœuds que renvoie le xpath, et crée autant de nouvelles slides avec chacune un xpath différent.

Attributs :

alias = alias du fichier XML dans lequel se trouve les informations à insérer

xpath = xpath permettant d'aller chercher l'information dans le fichier XML précisé juste avant. Dans le cas d'une boucle il permet de connaître la taille de la boucle et ainsi de faire varier le xpath au cours du temps.

num_slide = identifiant de la slide à répéter. Pour donner un identifiant à une slide voir le marqueur slide.

e.La balise rules

Syntaxe :

```
{{{Urbi:rules id=<ID_DE_LA_REGLE>}}}
```

Description :

Cette balise permet d'écrire des règles et de les affecter ensuite à une boucle dans les documents Microsoft Word. Une « rules » est constituée d'une « rule », d'un opérateur et d'une « rule ». Une « rule » est une expression qui peut être immédiatement évaluée et retourne un booléen. Ainsi la valeur d'une « rules » est l'évaluation de ces règles auxquelles on applique un opérateur. De plus une « rules » peut également être constituée de « rules ».

Attributs :

Id_de_la_regle = identifiant de la règle qui permet de l'utiliser dans une boucle

Exemple :

Je ne sais pas pourquoi, mais je sens que vous l'attendiez celui là. Dans l'exemple qui va suivre je définis une règle nommée r1 que j'affecte à une boucle. Pour cela il faut utiliser l'attribut « filter » des boucles et lui donner la même valeur que l'identifiant de la règle à appliquer.

```

{{{Urbi:loop rapport:path= </rapport/sommaire/chapitre>
sort:string_asc= </rapport/sommaire/chapitre/>filter= <r1>}}}

    • {{{Urbi:text rapport:path= </rapport/sommaire/titre>}}}

{{{Urbi:endLoop}}}

{{{Urbi:image rapport:path= </rapport/sommaire/img>}}}

{{{Urbi:endLoop}}}

{{{Urbi:rules id= <r1>}}}

    {{{Urbi:rule path= </titre> operator= <\> value= <toto>}}}

    and

    {{{Urbi:rules}}}

        {{{Urbi:rule path= </titre> operator= <!=> value= </. *Microsoft.*/> }}}

        and

        {{{Urbi:rule path= </description> operator= <=> value= <1> attr= <id>}}}

    {{{Urbi:endRules}}}

{{{Urbi:endRules}}}

```

Règle 1

Opérateur

Règle 2

Comme vous pouvez le constater la règle « r1 » est constituée de deux règles séparées par l'opérateur logique « ET ». Prenons les règles une par une, si l'on regarde la règle une, on s'aperçoit qu'elle est du type « rule » c'est donc une règle qui peut immédiatement être évaluée. Ses attributs sont :

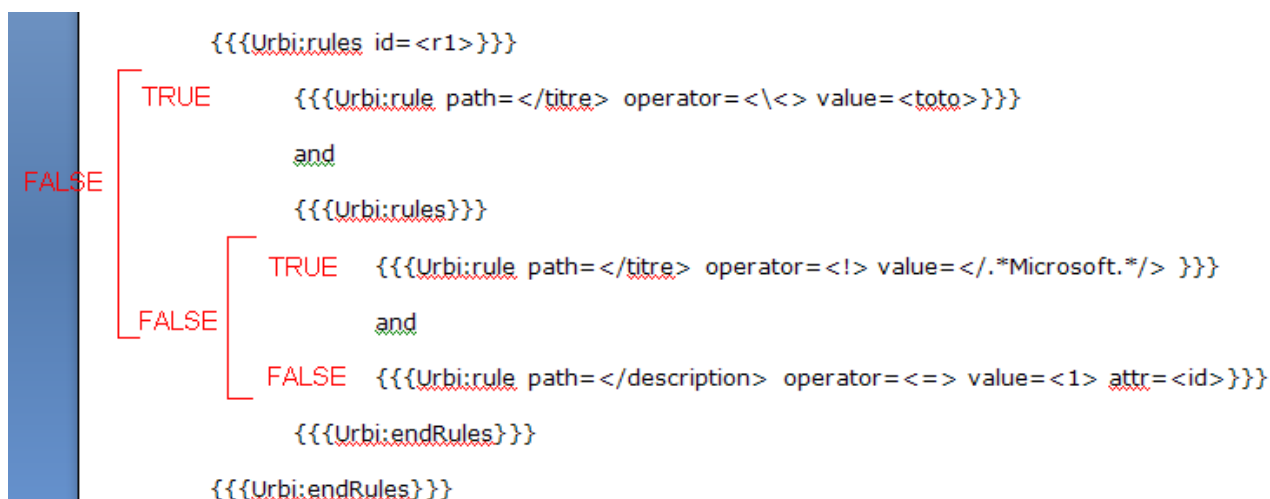
- un xpath qui doit mener à un nœud sur lequel la comparaison s'effectuera. Si vous souhaitez récupérer la valeur d'un attribut à la place de la valeur du nœud, vous pouvez soit utiliser la notation suivante dans le xpath « /description[@id] » soit utiliser l'attribut « attr » comme dans l'exemple ci-dessus.
- un opérateur de comparaison qui peut prendre les valeurs suivantes : <, >, =, !. Pour les opérateurs « = » et « ! » la valeur doit être une expression régulière.
- une valeur qui sera comparée à la valeur du nœud que le xpath cible

En français la règle une veut donc dire, je ne garde la ligne que si la valeur du nœud ciblé par « /rapport/sommaire/chapitre[NUM_DANS_LA_BOUCLE]/titre » est inférieur à la chaîne « toto ».

Observons maintenant la règle deux, contrairement à la règle un, c'est une « rules », c'est-à-dire qu'elle est constituée de deux règles et d'un opérateur logique. La

valeur de cette règle sera donc l'évaluation des règles qui la composent auxquelles on applique l'opérateur logique « ET ».

Au final la règle « r1 » peut donc être traduite par : je ne garde la ligne que si la valeur du nœud ciblé par « /rapport/sommaire/chapitre[NUM_DANS_LA_BOUCLE]/titre » est inférieure à la chaîne « toto » et ne contient pas le mot « Microsoft » et dont la description possède un attribut « id » de valeur un.



Dans notre cas il n'y a pas de nœud description dans notre fichier XML, le résultat retourné vaudra donc toujours faux et aucune ligne ne sera affichée.

f.La balise slide

Syntaxe :

```
{{{Urbi:slide id=<ID_DE_LA_SLIDE>}}}
```

Description :

Vous avez pu avoir un aperçu de cette balise grâce au marqueur loopRel, en effet ce marqueur ne sert qu'à donner un identifiant à une boucle. Par exemple dans le cas d'une loopRel cela sert à préciser quelle slide répéter.

Attributs :

ID_DE_LA_SLIDE = une chaîne de caractère qui permet de distinguer de façon unique les différentes slides

g. La balise text

Syntaxe :

{{{Urbi:text **ALIAS**:path=<**XPATH**>}}}

Description :

Et enfin, voici le dernier marqueur supporté, mais pas des moindres, puisque c'est sans aucun doute la balise la plus utilisée, celle qui permet d'insérer du texte. Comme pour toutes les autres balises, si un style particulier (une police, une taille de police,...) est appliqué au marqueur, ce style sera également repris par le texte inséré.

Attributs :

alias = alias du fichier XML dans lequel se trouve les informations à insérer

xpath = xpath permettant d'aller chercher l'information dans le fichier XML précisé juste avant.

Exemple :

Un exemple spécifique pour la balise text n'est pas nécessaire puisqu'elle est utilisée dans les exemples cités plus haut. En revanche, voici à quoi ressemble le document complet une fois généré.



2. Modification des règles CLIPS

Actuellement CLIPS n'est utilisé dans l'API que dans le cas où l'on utilise le marqueur « list ». En fait l'utilisation de CLIPS s'est décidée lorsque j'ai été confronté à un problème lié au PowerPoint. En effet à la différence des documents Word, dans un document PowerPoint lorsque l'on arrive à la fin d'une slide, le texte ne continue pas sur une nouvelle slide. Et pire, il n'existe pas de moyen de savoir si l'on déborde d'une slide ou non. Ainsi dans un premier temps nous avons décidé d'utiliser CLIPS afin de poser des limites pour empêcher ces débordements et proposer une répartition plus judicieuse. Puis dans un second temps nous avons décidé d'élargir l'utilisation de CLIPS à un rôle de personnalisation, par exemple avec la possibilité de définir le style des listes à puces, en définissant un caractère pour chaque niveau de profondeur.

Pour l'heure, du point de vue utilisateur deux choses sont possibles, la première est de changer le style des listes à puces et la deuxième est de changer la façon dont est réparti, sur plusieurs slides, une liste trop grande.

Tout se passe dans le fichier def.clp du répertoire clips. Dans notre cas ce sont les premières lignes qui nous intéressent :

```
(defrule MAIN::focus
  =>
  (assert(nbLiBySlide 5))
  (assert(bulletStyle "•,o,>"))
  (focus MAIN MAKE_REPARTITION DISPLAY END)
)
```

C'est ce que l'on appelle une règle, néanmoins ne paniquez pas, il n'est aucunement nécessaire de connaître ce langage, seul quelques menus modifications sont éventuellement à effectuer. Cette règle spécifie juste le nombre maximum d'éléments d'une liste sur une slide, précise le style des listes à puces et appelle les différents modules qui composent ce fichier.

Ainsi si vous souhaitez changer le nombre maximum d'éléments d'une liste sur une slide, il suffit de remplacer dans « (assert(nbLiBySlide 5)) » le cinq par le nombre de votre choix. De même que pour changer le style des listes à puces il suffit de modifier les différents caractères de la chaîne « (assert(bulletStyle "•,o,>")) », sachant que le « • » représente le niveau zéro d'indentation et « > » le niveau deux.

Pour changer l'algorithme de répartition d'une liste il est nécessaire de comprendre le langage CLIPS (se référer à la partie technique de cette documentation), néanmoins si vous vous sentez d'attaque voici une partie de l'algorithme en question :

```
(defrule select_li_withChild
  (nbLiBySlide ?nb)
  ?li <- (object(is-a TAG_LI)(childs $?clds)(num ?num&:(and (eq 0
(modulo ?num ?nb)) (neq ?num 0))))
  (test
    (> (length$ ?clds) 0)
  )
=>
  (bind ?*stringRepartition* (str-cat ?*stringRepartition* (- ?num 1)))
  (bind ?*stringRepartition* (str-cat ?*stringRepartition* ","))
)
```

3. Utilisation de l'api

Maintenant que les règles et le template sont prêts, il ne reste plus qu'à insérer des données dans le template.

Cet exemple montre comment générer un document Word, la procédure est exactement la même pour générer un document PowerPoint à l'exception qu'il faut utiliser, non pas un objet de type DocxGenerator, mais un objet PptxGenerator.

```
// $inputFile = emplacement du template
// $outputName = emplacement du fichier généré

$documentGen = new DocxGenerator() ;
$documentGen->setLocalTemplate($inputFile);
$documentGen->setOutputFile($outputName);
$documentGen->createDocument() ;
```

Pour vous faciliter la tâche j'ai écrit un lanceur qui permet de lancer le traitement facilement, la syntaxe de la commande est la suivante :

```
php_win\php.exe -q launcher.php -i "chemin_du_fichier_entrée" -o
"chemin_et_nom_fichier_sortie"
```

Quelque soit la façon que vous choisissiez pour lancer le traitement vous devez **impérativement** vous trouver à la racine du projet.

Le document résultant est enregistré sur le disque à l'emplacement spécifié et peut être ouvert dans votre lecteur favori, tel que le Micro\$oft Word. A noter qu'à la fin du document fraîchement créé se trouve un rapport dans lequel se trouvent toutes les remarques générées lors du traitement. Si le template contient trop d'erreurs, l'insertion des données sera interrompue et le document ne contiendra que ce rapport afin de vous aider à corriger le template utilisé.

Generation report

Information

- 18/06/10 17:18:59 - beginning of the generation of the document
- 18/06/10 17:19:00 - the document was generated in 0.760000 seconds

Error

Warning

- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document
- 18/06/10 17:19:00 - the tag 'endLoopRel' don't do anything in this type of document



Debug

Vous l'avez peut-être remarqué, mais le rapport de génération est en anglais, en effet il est possible de changer la langue de l'API. Pour cela il suffit de modifier le fichier « config.php » présent dans le répertoire « inc » et de changer la ligne « define('lang','en'); » par la ligne « define('lang','fr'); ». Dans ce fichier il est également possible de modifier tout un tas de paramètres tels que l'emplacement des fichiers temporaires, l'emplacement des fichiers de log, etc.