# Sydonie : modèle de document et ingénierie du Web

Jean-Marc Lecarpentier

**UNIVERSITÉ de CAEN/BASSE-NORMANDIE**

**U.F.R.** : Sciences

**ÉCOLE DOCTORALE** : SIMEM

**THÈSE**

présentée par

Jean-Marc LECARPENTIER

et soutenue

**le** 5 décembre 2011

en vue de l'obtention du

**DOCTORAT de l'UNIVERSITÉ de CAEN**

**spécialité : Informatique & Applications**

***(Arrêté du 7 août 2006)***

# Sydonie : Architecture de Document et Ingénierie du Web

**MEMBRES du JURY**

| | | | |
|---|---|---|---|
| Florence SÈDES | Professeur | Université Toulouse 3 | (rapporteur) |
| Vincent QUINT | Directeur de Recherche | INRIA | (rapporteur) |
| Manuel ZACKLAD | Professeur | CNAM - Paris | |
| Gaël DIAS | Professeur | Université de Caen Basse-Normandie | |
| Jacques MADELAINE | Maître de Conférences | Université de Caen Basse-Normandie | |
| Hervé LE CROSNIER | Maître de Conférences HDR | Université de Caen Basse-Normandie | (directeur) |

Mis en page avec la classe thloria.

# Remerciements

***First things first***

Thank you Michelle for your love and support. Thank you for everything, your time, your patience, our jogs, our laughs and even the tantrums. It all helped us move forward and make it through.

Thank you kids for your patience. Your shiny eyes, bright faces and smiles are always a source of energy.

Un grand merci à toute l'équipe de Sydonie.
Merci à Hervé Le Crosnier de m'avoir embarqué dans cette aventure. Merci d'avoir partagé ton enthousiasme et réflexions nocturnes avec l'équipe. Merci aussi pour le soutien moral au long de ces trois années. Enfin, merci de nous avoir convaincu que le web est un outil merveilleux.

Un énorme merci à Cyril Bazin. Sydonie ne serait pas ce qu'il est aujourd'hui sans toi. Ce manuscrit n'aurait pu être terminé sans tes nombreuses relectures. Merci infiniment Cyril.

Merci à Jacques Madelaine de m'avoir accompagné pendant ces trois années.

Merci à Nicolas Taffin d'avoir fait confiance à Sydonie pour créer des projets innovants et passionnants.

Merci à Charline Énée d'avoir essuyé maintes fois les plâtres. C'est aussi grâce à toutes les expérimentations réalisées que nous avons réussi.
Merci aux étudiants qui ont eux-aussi expérimenté avec Sydonie pendant leurs projets.

Mille remerciements aux rapporteurs de cette thèse, Florence Sèdes et Vincent Quint. Merci d'avoir accepté d'évaluer mon travail malgré un calendrier serré.
Merci à Manuel Zacklad et Gaël Dias d'avoir accepté de participer au jury.

Merci à l'Université de Caen Basse-Normandie qui m'a permis de bénéficier d'un aménagement de service pendant la durée de cette thèse.
Merci au GREYC et aux membres de DLU et de feu Dodola.
Merci à Fabrice Maurel d'avoir géré l'enseignement en ce début d'année, à Romain Brixtel de m'avoir remplacé au pied levé. Enfin, merci à Valérie Cauchard et Maroua Bouzid d'avoir décalé ma rentrée.
Merci à tous les collègues qui n'ont pas manqué de m'encourager dans la dernière ligne droite.

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction

## Contexte

Cette thèse de doctorat est articulée autour des réflexions sur les évolutions du web et de l'approche des documents numériques. Elle se concrétise dans la mise au point d'un framework reprenant nos propositions de modèle de document, d'interactions et d'ingénierie du web. Nous appelons ce framework Sydonie : SYstème de gestion de DOcuments Numériques pour l'Internet et l'Édition, distribué en logiciel libre.

L'architecture du web et des sites a considérablement changé en l'espace de quelques années. D'une architecture basée sur la diffusion de documents, le web est passé à une architecture de services, incorporant des documents, mais aussi des applications et des données structurées. Ce changement de paradigme a eu plusieurs conséquences. Tout d'abord, un changement d'approche dans la normalisation de HTML. En proposant de nouvelles balises pour la navigation, l'audio, la vidéo, etc. et de nombreuses APIs, par exemple pour le stockage de données sur le client, la communication entre différents contextes de navigation, etc., le web devient avec HTML5 un web d'applications. Les informations deviennent ré-utilisables, plaçant les métadonnées au centre du processus d'échange. Enfin, les sites web - et leurs producteurs - deviennent dépendants de prestataires de services en mode Ajax, par exemple pour l'affichage de cartes interactives. Les sites web sont présentés dans plusieurs langues pour répondre aux besoins d'un public planétaire. Les documents présentés dans les sites peuvent alors être disponibles dans diverses versions linguistiques et intègrent divers types de médias (texte, images, sons, vidéos). Pour les producteurs de sites et d'applications web, cet ensemble d'évolutions et de facteurs rend la création de systèmes opérationnels de plus en plus difficile. Les systèmes de gestion de contenu ou CMS (Content Management System), logiciels utilisés pour la gestion de sites, montrent alors leurs limites. En effet, si le cœur de métier des CMS était avant tout la gestion de la publication de contenu, la gestion des documents numériques composites et multilingues n'est pas toujours prévue.

Les bibliothèques ont, elles-aussi, beaucoup changé ces dernières années. La publication du rapport sur les spécifications fonctionnelles des notices bibliographiques, désigné par « les

FRBR », est une révolution du mode de catalogage des documents. Les FRBR proposent des concepts généraux plus adaptés au mode de fonctionnement du lecteur, qui cherche une œuvre dans un catalogue, puis choisit une version et enfin une édition de l'ouvrage puis l'exemplaire qu'il lira. La conception des FRBR constitue pour les bibliothèques un changement radical. Alors que le catalogue était auparavant constitué à partir de la description « de l'ouvrage dans les mains du bibliothécaire », les FRBR font découler l'exemplaire présent dans la bibliothèque d'un ensemble issu de volontés « intellectuelles ». La conception du travail original, dit Work dans les FRBR, et les divers travaux intellectuels nécessaires pour établir des Expressions de ce travail (par exemple les traductions, ou les abrégés,...), se réalisent ensuite dans des Manifestations dont l'exemplaire dans les mains du bibliothécaire est un des items (un des semblables). Cette inversion du mode de pensée est aussi une métaphore de ce qui se passe dans le web. Celui-ci a dans un premier temps été conçu autour de documents isolés, souvent appelés simplement « contenus », dont l'élément essentiel est leur « présentation » aux yeux du lecteur, lui même souvent réduit à une personne humaine derrière un navigateur. Plus récemment, nous allons vers un web de documents et de données, dans lequel la volonté intellectuelle de produire une information aisément exploitable est au moins aussi déterminante que la façon dont elle est présentée au lecteur. Un lecteur qui, par ailleurs, doit maintenant être considéré dans toute sa complexité : un humain derrière plusieurs types de terminaux, notamment des mobiles, ou bien un robot ayant besoin d'accéder directement aux connaissances enregistrées dans le document.

Notre expérience dans le domaine du développement et de l'enseignement des technologies du web nous a amené à approfondir les relations entre les documents sur le web et les concepts élaborés par les FRBR. Les travaux présentés dans ce mémoire sont centrés autour des articulations entre le monde des bibliothèques et des professionnels du développement web. C'est en filant cette métaphore que nous essayons de situer notre travail. Nous proposons un « modèle de document » qui soit, à l'image de ce qu'ont réalisé les bibliothèques dans la dernière décennie, un retournement des concepts pour mettre en avant la notion de relation de dépendances entre les éléments intellectuels et les réalisations présentées aux divers types de « lecteurs ».

Une orientation a guidé nos travaux : partir des pratiques réelles des personnes concernées par la production, la circulation et le stockage des documents. Ainsi, nous cherchons à mettre en avant le travail des « web designers », qui façonnent l'expérience de lecture collective. Ce sont souvent des innovations dans la présentation, dans l'ergonomie, dans les choix des métaphores exprimant les actions qui sont à la base de l'évolution du web. C'est pour répondre aux besoins de leurs clients (des sites d'informations ou des sites marchands) que les web designers innovent, poussent aux limites de leurs possibles les normes et les outils proposés par les chercheurs. Nombre des évolutions du web sont dues à cette innovation ascendante, et nous essayerons de nous en inspirer pour définir les outils et les modèles que nous allons proposer. Nous avons aussi toujours gardé comme guide l'indépendance de décision de celui ou celle qui développe un site ou

une application web. Nous croyons que le « modèle » d'une application doit être définie par les besoins auxquels celle-ci doit répondre, par le contexte toujours particulier de son déploiement. Nous avons cherché à proposer des outils qui permettent de définir les objets d'un système sans devoir dépendre de conceptions préalables ou extérieures à la problématique. Les éditeurs du web, qui organisent le flux d'information en fonction de chaque public particulier et aux objectifs économiques ou culturels, doivent, à notre sens, garder la dernière main sur la structure même de l'information et des documents de leurs sites.

Enfin, nous pensons qu'au delà des objets documentaires, ce sont les relations entre ces objets qui constituent la qualité principale des systèmes documentaires. Les relations sont à la fois liées à la schématisation des contenus et à la capacité d'utiliser ces contenus dans une organisation sociale. Nous retrouvons ces deux aspects au travers de nos contributions, d'une part, d'un modèle de document, et, d'autre part dans l'utilisation des relations pour la gestion des permissions et des dépendances entre objets.

Cette thèse a été réalisée au laboratoire GREYC de l'Université de Caen Basse-Normandie, dans le contexte d'un temps partagé entre la recherche et l'enseignement au département Informatique de l'UFR de Sciences.

Afin de fournir un cadre applicatif à nos travaux, nous avons créé le projet Sydonie, un framework de développement pour le web. Sydonie est le moyen de mettre en application nos résultats et d'associer le monde de la recherche et du développement web.

Sydonie, pour SYstème de gestion de DOcuments Numériques pour l'Internet et l'Édition, est un projet de l'équipe Documents, Langues et Usages du laboratoire GREYC. Il a reçu le soutien du Conseil Régional de Basse-Normandie, du projet TGE-Adonis, et est développé en partenariat avec la maison d'édition multimédia C&Féditions. Nous présentons les partenariats, les acteurs et les projets collectifs au chapitre 7 et dans la conclusion.

## Contributions

L'objectif de cette thèse est de proposer un modèle pour la gestion de documents numériques multilingues et de produire un framework adapté au développement d'applications web.

Les contributions présentées dans ce mémoire s'articulent autour de trois axes.

Le premier axe étudie l'impact des travaux réalisés dans le cadre des FRBR pour la modélisation de documents numériques composites multilingues. En utilisant ces concepts, nous proposons un modèle basé sur une métaphore autour des FRBR. Notre modèle regroupe les diverses versions linguistiques et divers formats d'un même travail intellectuel sous la forme d'une structure arborescente. Le modèle propose également un mode de reconstitution des documents com-

posites multilingues utilisant le principe de négociation de contenu. L'ensemble est implémenté dans le framework Sydonie.

La deuxième contribution présentée dans ce mémoire est centrée sur la réalisation concrète d'une plate-forme pour la création d'applications web intégrant la gestion de documents numériques multilingues, des métadonnées, et l'ergonomie de développement. En mettant en perspective les études portant sur l'ingénierie du web et notre expérience dans le domaine de la formation, nous proposons des solutions pour le développement d'applications web. Nous proposons une architecture de développement mettant en œuvre le modèle de document élaboré précédemment. En apportant des solutions concrètes pour la création du modèle de données des objets, pour la gestion des documents composites et multilingues, le framework Sydonie apporte une flexibilité de développement. La couche d'abstraction, qui gère complètement les aspects base de données, concentre la création d'une application web sur le modèle de celle-ci. Les relations entre objets, basées sur le modèle RDF de triplets (sujet, prédicat, objet), permettent la gestion simple des liens entre les objets du système. Nous proposons un modèle pour définir finement la politique de permissions d'un site. Avec ce modèle, basé sur les relations entre les objets et utilisateurs du système, un développeur spécifie des formules logiques pour exprimer les permissions. Nous présentons des mécanismes pour la gestion des interactions dans une application web, par exemple pour la saisies de données. Une bibliothèque Javascript simplifie la création des applications Ajax. En proposant la définition des zones à rafraîchir de façon simple à l'aide de classes au sein du HTML, un graphiste peut définir les interactions Ajax de son application sans coder de Javascript.

Enfin, la troisième contribution présentée est transverse aux deux premières. Il s'agit d'étudier l'interopérabilité entre les documents et les métadonnées qu'ils contiennent. Nous présentons un modèle pour la gestion des métadonnées dans les applications web. Basées sur le modèle de document que nous avons proposé, les interactions s'appuient sur l'architecture du framework Sydonie pour proposer une gestion des métadonnées plus accessible aux développeurs web. En utilisant le modèle de document sous forme d'arbre, cette gestion des métadonnées évite la redondance d'informations et permet la réutilisation des données lors de l'ajout de nouvelles versions ou formats d'un document. Pour intégrer la gestion des métadonnées au sein de son application, le développeur doit définir dans un fichier de configuration les correspondances entre les informations placées aux différents niveaux de l'arbre du document et les schémas de métadonnées.

## Organisation du mémoire

Ce mémoire est organisé en deux parties.

La première partie réalise un état de l'art autour des trois thèmes principaux de nos recherches.

Le premier thème présente les évolutions du web et permet de cerner le contexte dans lequel nous avons travaillé. Nous traçons un bref historique des techniques utilisées dans les applications web et présentons les notions essentielles pour la lecture de ce mémoire. Nous mettons les technologies du web en perspective avec les jeux de pouvoir des grandes firmes du web. Enfin, nous montrons comment le terminal de lecture occupe désormais une place centrale dans les affrontements du web.

Le deuxième thème est axé autour du document numérique. Tout d'abord, nous présentons les travaux des bibliothécaires, consignés dans le rapport sur les spécifications fonctionnelles des notices bibliographiques (désigné par « les FRBR »). Ces travaux sont un des points de départ de nos réflexions pour la création d'un nouveau modèle de document. La suite du deuxième thème présente les problématiques qui nous sont posées dans le cadre de la gestion de documents numériques multilingues.

Enfin, le troisième thème de cette première partie est axé sur l'ingénierie du Web. Nous attacherons une importance particulière aux acteurs du Web. Nous analysons leurs besoins et les spécificités du développement web. Nous présentons les faiblesses des systèmes actuels à nos yeux. Cette partie se termine par une synthèse.

La deuxième partie de ce mémoire présente nos contributions. Elle est organisées en trois thèmes (sous la forme de trois chapitres).

Le premier thème présente le modèle de document que nous avons élaboré. Ce modèle propose l'utilisation d'entités inspirées des FRBR pour gérer les documents composites multilingues. Les versions linguistiques et les formats de fichier d'un même document sont groupés dans une structure arborescente. Le premier thème présente cette approche et son implémentation.

Le deuxième thème présente l'intégration des métadonnées dans un système de gestion de documents. À partir de l'analyse faite dans la première partie, nous proposons un modèle d'interaction pour la gestion des métadonnées dans les applications web. L'utilisation du modèle FRBR pour les documents apporte une gestion plus fine des métadonnées.

Le troisième thème de cette deuxième partie présente les aspects de génie logiciel qui ont été mis en œuvre dans nos travaux. La souplesse et l'ergonomie de développement ont été des aspects importants de nos réflexions. Nos résultats sont détaillés dans ce chapitre.

Nous concluons ce mémoire avec un bilan des travaux menés, et présentons les travaux en cours et nos perspectives de recherche.

# Première partie

# État de l'art

# Introduction

Cette première partie reflète les divers aspects de notre travail de réflexion et de réalisation au cours des trois années écoulées : tenter de recouvrir les multiples facettes du web, du document numérique et des acteurs de ces deux domaines. Nous avons choisi de présenter cet *état de l'art* en trois thèmes.

Le premier thème traite de l'évolution du web du point de vue des techniques, des contenus et de leur interopérabilité, de la normalisation, puis de la place du terminal de lecture. En traçant un bref historique et en mettant en perspective les évolutions récentes du monde du web, ce thème permet de cerner le contexte dans lequel nous avons travaillé.

Le deuxième thème est centré sur le document numérique. Avant tout, nous présentons les travaux réalisés dans le cadre des bibliothèques. Ces travaux seront constamment utilisés dans la suite de ce mémoire. Nous nous intéressons ensuite au document numérique composite, puis au document numérique multilingue. Nous regardons quelles relations établir entre diverses traductions. Nous ne traiterons pas du traitement de la langue et de la traduction. Nous considérons ici que les diverses versions linguistiques d'un document existent et ont été obtenues indépendamment de notre système. Nous verrons comment gérer les données de l'interface de navigation d'un site. Cette partie fait le point sur les concepts qui seront appliqués dans notre réalisation.

Enfin, le troisième thème de cette partie est axé sur l'ingénierie du Web. Nous attacherons une importance particulière aux acteurs du Web. Notamment, une catégorie d'acteurs rarement mentionnées, qui occupe une place centrale à notre avis, est celle qui rassemble les concepteurs et développeurs de sites et d'applications web. En nous appuyant sur notre expérience de formation de professionnels du web, nous nous attacherons dans cette partie à présenter les spécificités du développement web.

La conclusion fera une synthèse des points abordés dans cette première partie afin de faire le lien avec la présentation des travaux réalisés dans le cadre de cette thèse.

# Chapitre 1

# Web : évolutions des techniques, contenus et logiciels

**Sommaire**



Le Web est un espace de création : création informatique, création artistique, création de nouveaux modèles économiques. Mais le Web est aussi un espace d'échanges : échanges de documents (partiels ou complets), de métadonnées, de techniques et de services. Créer un site, c'est s'intégrer dans un écosystème qui gère ces divers échanges.

Dans ce chapitre nous reprenons en partie les contributions faites dans des conférences axées sur le Document Numérique afin de présenter les évolutions du Web en termes de techniques et de contenus [Lecarpentier *et al.* 08]. Nous montrons ensuite comment le navigateur, ou plus précisément le terminal de lecture, est devenu une pièce centrale autour de laquelle se joue les jeux de pouvoirs des grandes firmes du Web [Le Crosnier & Lecarpentier 10]. Notre discours s'articulera autour de la relation complexe entre l'évolution des normes établies par le World Wide Web Consortium, les besoins et pratiques des web designers, et les forces économiques et techniques qui dominent le Web.

## 1.1 Introduction

Les développements techniques autour du Web sont nombreux, dans des domaines comme les langages de programmation, les connexions (haut débit, connexion permanente et pervasive), la multiplication des terminaux de lecture par exemple. La composition des « pages web » fait désormais intervenir de nombreux acteurs, avec la syndication, les services web ou les widgets entre autres. Ces facteurs ont amené une évolution rapide et profonde de l'architecture du web et de ses contenus. Dans cette partie nous pointons les transformations qu'imposent ces évolutions à la notion de « document numérique ».

Nous tracerons ces évolutions dans une double optique. D'une part celle des web designers, qui, au quotidien, cherchent à transformer l'expérience utilisateur et à permettre à leurs clients (les sites commerciaux principalement) de mieux assurer leur retour sur investissement. Pour cela, les designers poussent dans leurs derniers retranchements les protocoles et les usages établis, ils inventent avec les outils qui leur sont proposés (par exemple Flash ou Ajax) des modes ergonomiques de navigation et de sélection, souvent sans se préoccuper de la maintenance à long terme de leurs créations. D'autre part, dans l'optique des opérateurs de la normalisation du web, qui cherchent à définir des formes d'encodage, des protocoles de transfert et des règles d'accessibilité afin de maintenir le contexte documentaire des réseaux numériques (ré-utilisabilité, maintenance et archivage). De plus en plus, les pages Web sont constituées par la réutilisation de blocs d'information instituant un modèle quasi industriel de média.

## 1.2 Évolution des techniques : retour sur HTML

### 1.2.1 Pages Web : évolutions des normes et des pratiques

La débat qui a fait rage au sein du W3C [1], instance de normalisation des technologies du web, pendant la fin des années 2000 a opposé deux approches. D'un côté les partisans de la conception d'un web de documents qui sont « transformés » pour être présentés à des usagers au travers d'un navigateur et de l'autre celle d'un web d'applications dans lequel les « pages web » sont des supports à l'interaction des utilisateurs. La première logique emploie des méthodes appuyées sur les feuilles de style (CSS ou XSLT et XSL :FO) s'appliquant à des documents XML. Ces documents XML représentent l'architecture logique du contenu, avec indépendance de la forme et du fond, ou plus précisément de la logique du document et de sa présentation. La seconde approche part du résultat présenté à l'usager et remonte le fil des besoins. Elle utilise le balisage comme un langage de description de pages conçues comme une composition de services. Cette approche utilise un langage de balisage ouvert permettant d'intégrer à la fois des contenus documentaires, mais aussi des applications, depuis les forums jusqu'au commerce électronique. D'un côté nous avons une logique centrée sur le Document et de l'autre une logique axée autour de services.

Avec le fort soutien des fabricants de navigateurs, la logique de web d'applications l'a finalement emporté, avec la publication de la recommandation de HTML5 [Hickson 11b] et l'abandon du groupe de travail XHTML au W3C [2]. Dans la partie suivante nous revenons sur les évolutions du langage HTML depuis sa version 4.0.

### 1.2.2 De HTML4 à HTML5 en passant par XHTML

Avec l'arrivée de HTML 4.0 en avril 1998, il devient possible de décrire une structure complexe de document et de présenter des informations diverses sur une même page écran. La plupart des pages web étaient jusqu'alors un document dit *standalone* connecté à d'autres documents *via* les liens hypertexte. D'une collection de pages web reliées par ces liens, on passe à des sites créés avec une optique de navigation et d'ergonomie afin de guider l'utilisateur. L'usage des *frames* a simultanément renforcé cette évolution. L'omniprésence du « paratexte » de navigation s'est imposé dans tous les sites web (bannières, menus, découpage en rubriques, pied de page...).

Les web designers, afin d'améliorer l'interface utilisateur et la navigation, détournèrent alors largement l'esprit du balisage HTML. Par exemple en utilisant des tableaux (élément `<table>`)

1. La majeure partie des techniques décrites dans cette partie ont une page de présentation sur le site du World Wide Web Consortium. Nous renvoyons globalement à ce site pour les définitions plus précises. Nous n'avons cherché à préciser que les normes liées précisément au sujet de cette partie.
http://w3c.org

2. Annonce du 2 juillet 2009 :http://www.w3.org/News/2009#entry-6601

pour présenter le contenu d'une page sur plusieurs colonnes. Cette pratique permettait d'avoir ainsi des menus latéraux permanents pour la navigation, même si le document normatif de HTML 4 précise explicitement que l'élément `<table>` ne doit pas être utilisé pour mettre en forme un document [3]. On trouve ici une première illustration de cette différence entre les normalisateurs, cherchant la maintenance ultime des documents, et les designers, cherchant à partir des situations techniques (balisage + modèle d'implémentation dans les navigateurs) à obtenir des présentations élaborées et adaptées aux besoins du lectorat.

XHTML 1 [Pemberton 02] (première publication en janvier 2000) reprend les grandes lignes de HTML 4 en le « purifiant » afin de le rendre compatible avec XML. XHTML élimine ainsi la plupart des balises de mise en forme du texte (telles que `<font>` par exemple) afin de se concentrer sur la description de la structure du document présenté. Les feuilles de style CSS deviennent le pivot central de la présentation du document. Les concepteurs de XHTML s'attachent à promouvoir la séparation forte entre la structure du document et sa réalisation sur tout type de support. L'utilisation de feuilles de styles programmables, telles XSLT et XSL :FO, permet de créer des modèles adaptés à chaque nouveau support ou terminal pouvant apparaître : documents imprimables (en PDF), ou présentation pour les mobiles (notamment les systèmes WAP qui apparaissaient à l'époque).

Avec l'arrivée de navigateurs prenant en charge de mieux en mieux les feuilles de style, les premiers détournements du langage HTML (les tables par exemple) tendent à disparaître, au profit de « zones » définies sur une page web grâce à l'élément division `<div>`. Ces zones sont placées les unes par rapport aux autres par la feuille de style, voire par l'usage de bibliothèques Javascript (par exemple *jQuery* ou *Mootools*). Elles évoluent de plus en plus vers des *widgets*, zones à contenu variable positionnées et dimensionnées par l'usager sur le canevas de son écran. Avec l'arrivée du web dit « 2.0 » et le développement des réseaux à haut débit, les pages web intègrent de plus en plus d'éléments divers qui n'entrent pas dans la métaphore du document tel qu'il est prévu dans la norme XHTML :

- divisions dont le contenu est géré via des programmes Javascript ;
- utilisation de programmes tiers par le navigateur (*plugins*, dont Flash en particulier) pour intégrer des contenus multimédias ou des applications (par exemple lecteur vidéo comme sur la page d'accueil du NY Times ou sites de jeux en ligne) ;
- transformation de la page web en support à du *streaming*, donc un mode connecté en permanence, qui s'éloigne de la métaphore du document.

D'autres usages ont parallèlement été inventés, qui vont au contraire renforcer la métaphore du document. Notamment les usages permettant de donner une sémantique de domaine au contenu HTML de certaines pages. Quand la sémantique propre des balises HTML se concentre sur la structure logique du texte, et permet au mieux une indexation « documentaire » par

3. Extrait : *Tables should not be used purely as a means to layout document content.*

analyse syntaxique des pages, il s'agit par ces pratiques de promouvoir la réutilisation de cette structure dans un cadre plus global d'extraction de connaissances. Ces processus d'échanges d'information seront explorés dans les sections suivantes. Un des symptômes de cette continuité de l'approche par des documents est l'usage de XHTML dans le format de livres numériques au format ePub, et en particulier l'insistance sur XHTML5 dans la version 3 de la norme ePub parue en septembre 2011 [Conboy *et al.* 11].

Le groupe de travail sur XHTML2, aujourd'hui abandonné, suivit l'évolution de XHTML et produisit une norme entièrement basée sur les technologies XML, dont tout le monde s'accorde à dire qu'elles sont le meilleur gage de pérennité en termes d'archivage ou d'indépendance des terminaux. Ce choix permet l'intégration de données sémantiques avec RDF mais aussi de nombreuses autres modèles : mathématiques (MathML), graphiques (SVG), etc. Il s'agit en fait d'élargir le vocabulaire des balises XHTML valides en définissant des mécanismes d'intégration de dialectes XML dans le cadre de diffusion documentaire de (X)HTML.

Ce choix induisait cependant une coupure avec le web actuel : en tant qu'application XML, un navigateur interprétant du XHTML 2 aurait dû impérativement générer une erreur si le document à afficher était mal formé (*The user agent* must *parse and evaluate an XHTML 2 document for wellformedness*) [Pemberton *et al.* 06]. Ceci aurait impliqué que la compatibilité antérieure ne soit plus assurée. En effet les navigateurs actuels sont très tolérants avec les documents (X)HTML mal formés et s'en arrangent pour les afficher « au mieux ». La communauté des web designers estime intolérable l'application d'une règle trop rigide, en partant du principe que 90% (au moins) du contenu présent sur le web l'est dans des documents mal formés (ce qui est communément appelé *tag soup*). Si XHTML 2 ajoutait des possibilités pour des contenus dynamiques, il reste dans la métaphore du document générique. Or les web designers et les fabricants de navigateurs voulaient voir d'autres fonctionnalités utilisées dans les pages web directement au travers du balisage. Créé en 2004 pour envisager un futur à HTML4 différent de celui étudié par le W3C, le groupe de travail WHATWG (Web Hypertext Application Technology Working Group [4]) a préparé de façon indépendante du W3C une nouvelle version de HTML. Pour éviter une scission, Tim Berners-Lee propose en 2007 la création du groupe de travail HTML5, qui va prendre une direction distincte de l'option du « tout XML ». Le document de travail de HTML 5 précisait clairement cette contradiction [5] :

> XHTML2 définit un nouveau vocabulaire HTML qui améliore les fonctionnalités des liens hypertextes, l'insertion des contenus multimédia, l'annotation et l'édition de documents, la richesse des métadonnées, la construction déclarative des formulaires interactifs, et permet de décrire la sémantique des œuvres littéraires tels que des

4. http://www.whatwg.org/

5. XHTML2 ayant disparu, cette note ne figure plus dans la version actuelle de HTML5. On peut la retrouver dans la première version : http://www.w3.org/TR/2008/WD-html5-20080122/#relationship0

poèmes et des articles scientifiques.

Cependant, il lui manque des éléments pour exprimer la sémantique d'un grand nombre de contenus non-document, qui sont pourtant fréquents sur le Web. Par exemple, les forums, les sites d'enchères, les moteurs de recherche, les boutiques en ligne, et autres, ne trouvent pas leur place dans la métaphore du « document » et ce faisant ne sont pas couverts par XHTML 2.

### 1.2.3 HTML5 et ses APIs

La recommandation HTML5 [Hickson 11b], ainsi que ses recommandations « satellites », montre clairement les différences en introduisant nombre de nouveautés basées sur les problèmes des web designers, dans le but de leur simplifier le travail. Certaines de ces nouveautés sont dans une logique de développement d'applications et non plus dans la métaphore du document :

- balises `<menu>` et `<aside>` pour définir des menus contextuels ou de type « barre d'outils », élément `<canvas>` pour intégrer des dessins ou animations pilotées par Javascript ;
- APIs intégrées pour les interactions utilisateurs : pour glisser-déposer des éléments sur une page web, pour avoir un contenu modifiable (*contenteditable*) ou encore pour les formulaires (web forms) ;
- API pour navigation *offline* ;
- APIs de stockage de données sur le client : du stockage simple [Hickson 11d] à la base de données intégrée au client [Hickson 10] [Mehta *et al.* 11] ;
- API de Géolocalisation [Popescu 10], technique devenant extrêmement importante vu l'explosion des *smart phones* permettant de consulter le web à tout instant ;
- méthodes d'accès à des services web [Van Kesteren 10] quel que soit le domaine ;
- API Web Workers [Hickson 11e] pour exécuter du Javascript avec des threads différentes de l'interface utilisateur ;
- API Web messaging [Hickson 11c] pour la communication entre différents contextes de navigation.

La liste ci-dessus montre bien que l'objectif des ces nouvelles fonctionnalités est de faciliter l'intégration de diverses options dans les pages web, sans avoir recours à de nombreux programmes Javascript comme c'est le cas actuellement. L'exemple des pages personnalisées comme iGoogle ou Netvibes, des éditeurs en ligne comme Google Docs (traitement de texte et tableurs entre autres) ou Flickr (gestion de collection d'images) montrent les nouvelles utilisations de la page web. Celle-ci devient une application, le navigateur n'étant plus le logiciel qui permet d'afficher un document, mais le logiciel qui permet à l'application de fonctionner. L'adoption de XHTML5 dans la nouvelle version de la norme ePub version 3 (ePub3) [Conboy *et al.* 11]) rend possible le passage du livre numérique *stricto sensus* au livre-application tout en conservant la

métaphore du document.

Cependant, malgré toutes ces APIs prometteuses et les progrès accomplis ces dernières années, l'implémentation de toutes ces fonctionnalités dans tous les navigateurs est loin d'être complètement réalisée[6]. Le travail du développeur web reste donc compliqué pour assurer le fonctionnement des applications sur les navigateurs les plus utilisés. Si les principaux navigateurs tendent à être de plus en plus compatibles, l'arrivée des navigateurs pour les appareils mobiles (par exemple iOS Safari, Opera Mini, Android Browser) complique de nouveau la donne. L'évolution des contenus, dont la partie suivante donne un aperçu, apporte d'autres contraintes et problèmes de compatibilité pour le développeur.

## 1.3 Évolution des contenus

Le « site web » formé de pages HTML reliées entre elles fait désormais partie de la préhistoire du développement web. Aujourd'hui, un site web est un ensemble d'informations interconnectées avec le reste du monde. En effet, l'architecture du web change en proportion de la recomposition des « pages web » comme un ensemble de services. Quand le schéma du premier web suggérait la logique de reproduction des médias, avec un émetteur (le serveur) qui composait une page qui serait servie à un lecteur (le client) et remise en forme par son agent utilisateur (le navigateur) au moyen des feuilles de style, le nouveau web vise au contraire à juxtaposer des parties documentaires et des parties interactives ou des « services ». Le développement des widgets est un symptôme de cette évolution des pratiques.

A l'origine du web, les pages étaient « statiques », c'est-à-dire qu'une page web correspondait à un document dont le contenu était figé. Chaque lecture d'une page se traduisait par le même observable. Puis l'intégration de langages de programmation permettant une interaction entre un serveur web et une base de données a permis la création de pages dites « dynamiques ». Le contenu est alors extrait d'une base de données, permettant ainsi des affichages différents selon les situations, par exemple selon la date, les dernières données entrées dans la base de données, les préférences de l'utilisateur, la géolocalisation, etc.

Avec l'arrivée du web dit « 2.0 », ce schéma a évolué. Les données servant à constituer le contenu des pages web ne sont plus uniquement sur le serveur et la base de données du site web consulté. Les contenus sont obtenus sur le web via une architecture orientée services en collectant des informations venant de flux RSS ou en interrogeant des services web. Le modèle de base de diffusion du Web et l'architecture du Web dit 2.0 sont résumés sur la figure 1.1.

Il est important de noter que les informations collectées sur le web transitent toujours, dans ce modèle, par le serveur interrogé par l'usager. Même dans le cadre d'une interrogation de type

6. Voir par exemple http://caniuse.com/ ou http://html5test.com/ pour visualiser les fonctionnalités de chaque navigateur.

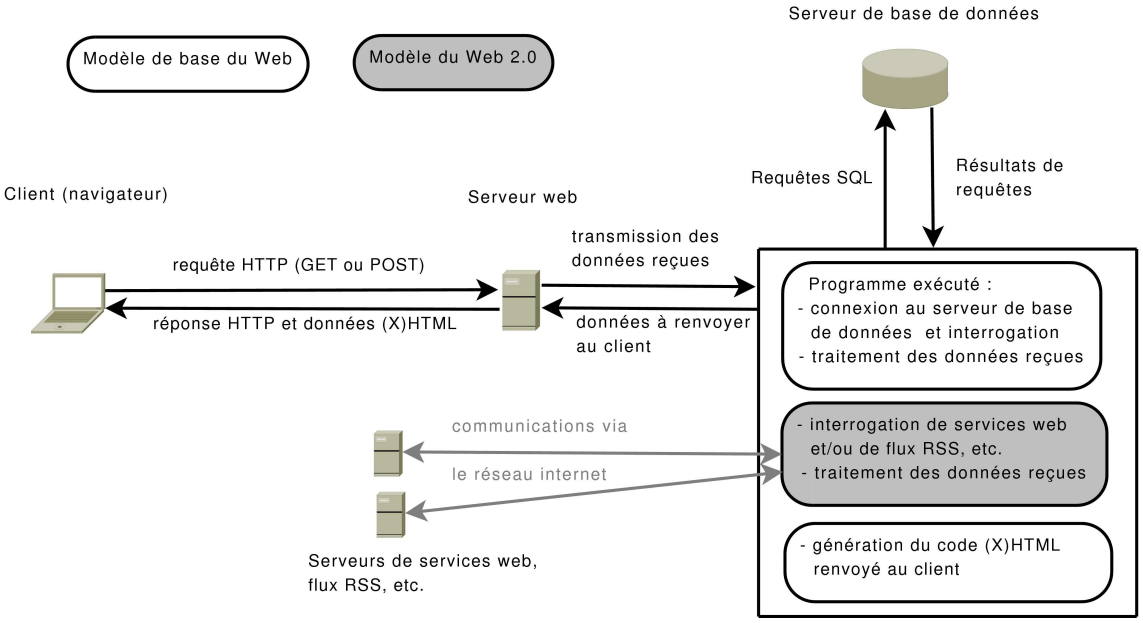



FIGURE 1.1 – Modèles de diffusion du Web de base et du Web 2.0, architecture orientée services

Ajax où seulement une partie de la page est recalculée et rafraîchie, le serveur reste responsable du contenu envoyé à l'usager. Le programme du serveur peut soit choisir de reproduire intégralement ce qui aurait été fourni par un tiers (modèle d'agence, ou de marque blanche), soit de traiter l'information du tiers pour y apporter diverses valeurs ajoutées (mise en contexte, tri des informations... ou insertion de la publicité pour les webmédias, *etc*).

Deux techniques peuvent être utilisées pour interroger des services web. La première, soutenue par le W3C, est basée sur les technologies XML et doit permettre une automatisation totale : de la recherche du service web à utiliser (UDDI[7]) à l'interrogation du service *via* le protocole SOAP [MITRA & LAFON 07] en passant par la description du service proposé spécifié en WSDL [CHRISTENSEN *et al.* 01].

Au vu de la complexité de mise en œuvre de ces technologies, l'enthousiasme n'est pas de mise chez les développeurs web. Seules les structures disposant de compétences avancées en XML investissent réellement dans l'utilisation des services web avec SOAP et WSDL.

D'autant plus qu'une nouvelle technique, basée sur l'architecture REST [FIELDING 00] apparaît. Soutenue au départ par Yahoo ! et ses filiales, cette technique est vite devenue la plus utilisée par les web designers. Le service web est accessible via un URI et les informations nécessaires pour l'interroger sont transmises directement via une requête HTTP (en méthode GET pour les plus simples). Par exemple, une recherche sur le web service REST de Flickr pour

7. Universal Description Discovery and Integration, à l'origine un projet de recommandation du W3C, aujourd'hui un modèle de recherche de web services soutenu par quelques grands industriels, mais en perte de vitesse. Voir http://en.wikipedia.org/wiki/UDDI.

des images de l'utilisateur `x32b` s'effectue avec l'URL :

`http://api.flickr.com/services/?api_key=1234&user_id=x32b&method=flickr.people.getPhotos.`

Le service web répond alors en envoyant des données XML (ou sous d'autres formats, comme le populaire JSON[8]) contenant les réponses.

La principale raison du succès de cette méthode est son extrême simplicité de mise en œuvre. En particulier, elle ne nécessite pas de connaissances avancées en XML, compétence assez peu développée chez les web designers. Une autre raison du succès de la méthode REST réside dans le fait que XML n'est plus le seul format d'échange possible, contrairement à l'utilisation de SOAP. Par exemple, le service web de Flickr propose ses résultats en XML, PHP sérialisé, JSON ou JSON avec appel de fonction *callback* (technique appelée JSONP[9]).

Avec le format JSON, les informations reçues peuvent être immédiatement traitées par le programme Javascript qui met à jour la page web. De plus, si un appel de fonction *callback* est ajouté à la réponse du serveur (JSONP), alors les échanges peuvent se faire directement entre le client et le serveur tiers en utilisant uniquement Javascript. La contrainte liée à l'utilisation d'Ajax qui limite les échanges au même domaine, se trouve alors levée. Les développeurs peuvent dans ce cas utiliser directement dans leurs pages les APIs proposées par les fournisseurs de service. Cette technique permet de développer très rapidement diverses fonctionnalités, de l'affichage des dernières photos publiées sur Flickr aux cartes interactives de Google maps par exemple.

On retrouve ici l'opposition entre l'objectif de pérennité et d'indépendance représenté par XML, qui nécessite des opérations de traitement lourdes, et l'objectif de rapidité et simplicité de développement demandé par les web designers. Cependant, les informations transmises dans les données JSON ne sont justement *que* des données. L'affichage est renvoyé au programme Javascript qui devient le point nodal de reconstruction du document. Or, un document est plus qu'un simple ensemble de données, il est associé à une présentation et un processus éditorial. Nous montrerons à la section 1.4 que d'autres possibilités existent pour les échanges de documents, et non simplement de données.

L'utilisation de JSON et JSONP, couplée aux fonctionnalités DOM et Javascript des navigateurs, permet donc d'interroger directement des services tiers, sans transiter par le serveur du site web qui a délivré la page web. Cette architecture, qui ne change apparemment rien du point de vue de l'internaute lecteur, est fondamentalement différente. En effet, les informations ne transitent plus via le serveur du site web qui distribue le contenu et en est responsable, mais les communications se font directement entre le client et le serveur tiers. La figure 1.2 montre les deux modes de fonctionnement présentés ci-dessus. Pour fonctionner, un programme Javascript exécuté sur le navigateur est nécessaire. Celui qui fournit ce programme possède la maîtrise de

8. Javascript Object Notation : http://json.org/
9. JSON with Padding : http://en.wikipedia.org/wiki/JSONP

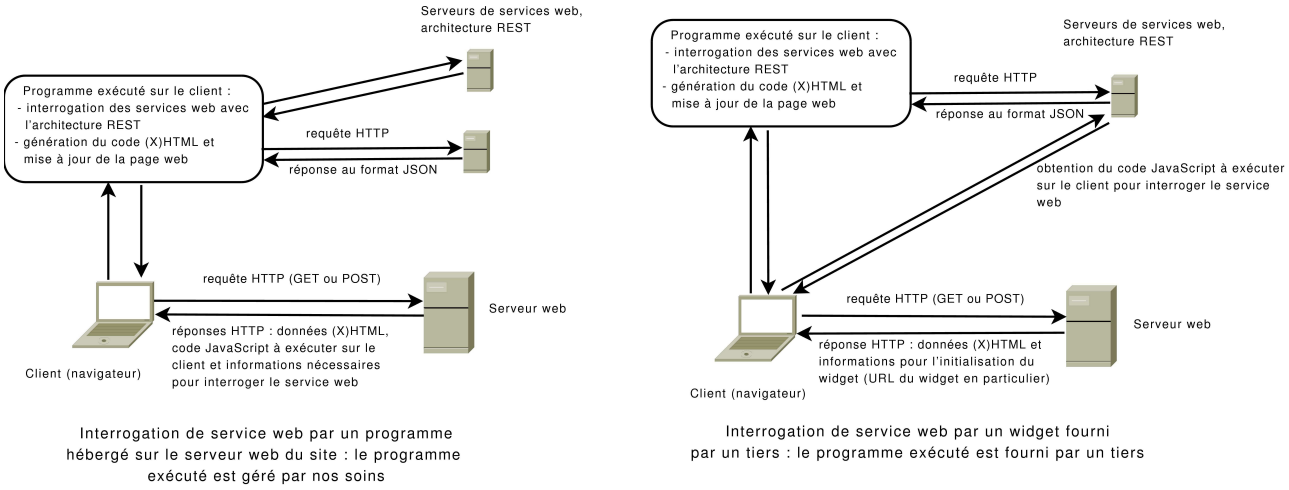



FIGURE 1.2 – Comparaison entre interrogation de services web par un programme exécuté sur la machine client et interrogation de service web via un *widget*

la présentation de l'information. Il importe donc de savoir s'il est fournit par le distributeur ou par le fournisseur de l'information tierce, ce qui a un impact immédiat sur la publicité associée. Google maps est l'exemple le plus répandu de cette contradiction. On parle de widgets pour désigner des packages complets fournis par le producteur de l'information tierce. C'est par exemple le choix de Google pour l'usage de toutes ses APIs, que ce soit Google maps, les APIs de visualisation et de recherche documentaire ou de diffusion de vidéo par YouTube... et bien évidemment les APIs de publicité AdSense.

L'architecture de web services, telle qu'elle existait au départ, est une forme de relation B2B (« business to business ») entre serveurs, assimilable à la relation entre la presse et les agences. Elle est cependant en train de se voir supplanter par une architecture dans laquelle une page présentée par un distributeur (i.e. un « média-web ») contient non pas des informations produites par un tiers, qui auraient été éventuellement retraitées (architecture de web services) mais incorpore une application (en langage de script) qui interagit directement avec le prestataire de cette information tierce.

## 1.4 Descriptions sémantiques dans le contenu

Le Web est aussi un espace d'échanges de documents et d'information sur ces documents. Très vite est apparu le besoin de faire apparaître dans les pages web des informations relatives au contenu de la page. La balise `<META>`, apparue avec HTML 2.0 en 1995, permet d'indiquer des informations aussi diverses que des mots-clés, des mentions d'auteur ou d'éditeur, des précisions sur l'encodage de la page ou sur le logiciel qui a servi pour créer la page. Dans cette conception, les métadonnées décrivent la page entière, principalement pour un usage par les moteurs de

recherche. Toutefois la pratique du « meta tag spam » ou « spamdexing » a amené à dénigrer cette approche, les moteurs étant contraints de ne plus tenir compte des balises `<META>`.

On comprend néanmoins que l'interprétation des textes est un exercice difficile dans un espace polysémique comme le web. La recherche textuelle montre ici ses limites. Il devient nécessaire de fournir une « explication de texte » de certains termes présents dans le document pour favoriser l'émergence de systèmes de recherche plus évolués. Les informations complémentaires contenues dans le texte et non visibles par le lecteur humain facilitent la création de systèmes d'extraction de connaissance.

Enfin la capacité d'utiliser des parties d'un texte comme moyen d'accès à des ressources communes grâce aux Linked Data [Bizer *et al.* 08] va transformer la conception des documents et leur édition. C'est ce que cherchent à réaliser les microformats, RDFa et les microdata que nous allons présenter.

Les web designers ont voulu compléter HTML pour donner une sémantique de domaine aux contenus de leurs pages, en enrichissant par des métadonnées certains termes présents. Quand la sémantique propre des balises HTML se concentre sur la structure logique du texte, et permet au mieux une indexation « documentaire » par analyse syntaxique des pages, il s'agit ici de promouvoir la ré-utilisation de cette structure dans un cadre plus global d'extraction de connaissances.

### 1.4.1 Microformats

Les microformats [10], créés dès 2005, permettent d'utiliser les éléments HTML, leurs attributs, en particulier l'attribut de classe, pour baliser sémantiquement des données sans avoir à élaborer de norme spécifique. Des groupes de développeurs s'entendent sur un ensemble d'attributs et de noms de classes, formant un modèle simplifié de vocabulaire partagé. Ces informations peuvent à la fois être utilisées par les feuilles de style CSS pour la présentation, et par tout robot d'extraction de connaissances pour produire une représentation sémantique. Ainsi le microformat hCalendar permet d'intégrer des données de type iCalendar [IETF-RFC2445 98] dans une page HTML. Les informations peuvent alors être reprises et intégrées dans des logiciels personnels de gestion d'agenda. De même le microformat hCard implémente le format standard vCard [IETF-RFC2426 98] à l'aide d'éléments et attributs HTML, comme le montre le code source de la figure 1.3.

Ces méthodes utilisant les possibilités existantes du langage HTML étaient, au moment de leur adoption par les web designers, bien en avance sur ce que proposait le W3C. Elles ont été poussées par une large communauté de développeurs et de designers qui s'en sont emparés, proposant des extensions aux navigateurs pour exploiter les informations formatées à l'intérieur

10. http://microformats.org

```
<div id="hcardJeanMarcLecarpentier" class="vcard">
  <a class="url fn" href="http://users.info.unicaen.fr/~jml/">Jean-Marc Lecarpentier</a>
  <div class="org">Université de Caen BasseNormandie</div>
  <a class="email" href="mailto:jml@info.unicaen.fr">jml@info.unicaen.fr</a>
  <div class="adr">
    <div class="streetaddress">Boulevard Maréchal juin</div>
    <span class="locality">Caen</span>
    <span class="postalcode">14000</span>
    <span class="countryname">France</span>
  </div>
  <div class="tel">+33 231 567 375</div>
</div>
```

FIGURE 1.3 – Exemple de code HTML utilisant le microformat hCard

des pages HTML.

De son côté, l'instance de normalisation vise à concevoir un formalisme de ré-utilisation des informations sémantiques intégrées dans les pages web qui soit plus formel et plus généralisable. C'est le processus dit de « glanage » et la norme GRDDL [CONNOLLY 07]. La métaphore et le choix du terme sont significatifs. Quand le moissonnage est l'organisation globale de la récolte de documents (par exemple définis par l'exposition de métadonnées par des entrepôts de type OAI-PMH [NELSON & WARNER 08]), le glanage vient après la dispersion des documents et cherche à récupérer, à titre individuel ou en petites communautés, des informations à partir de l'analyse document par document des contenus. C'est un « bénéfice secondaire » ouvert à la communauté. Un document « glanable » a un profil et doit indiquer quel est le logiciel de transformation qui permettra d'extraire automatiquement les connaissances balisées sémantiquement en son sein. Il s'agit de faire coopérer une logique individuelle (glanage et interprétation locale, par exemple avec les extensions des navigateurs web) avec l'organisation plus globale d'un web sémantique, permettant de partager les connaissances inscrites dans les documents.

### 1.4.2 RDFa

Pour modéliser et élargir ce processus de glanage, le W3C a cherché à développer une méthode pour inscrire les triplets relationnels de RDF (Ressource Description Framework) [MANOLA & MILLER 04] dans les documents (X)HTML en exploitant de nouveaux attributs. C'est le sens de la norme RDFa [ADIDA & BIRBECK 08], généralisation du processus initié par les microformats. La figure 1.4 montre deux exemples de code HTML intégrant RDFa. Mais à nouveau, une telle généralisation se fait au détriment de la simplicité. Ce qui était adapté à des communautés de développeurs focalisées sur quelques domaines, comme dans les microformats, devient bien plus lourd à mettre en place pour un processus ouvert et extensif (nouveaux vocabulaires, namespaces,...). Il ne s'agit plus de se servir des outils disponibles pour des bénéfices secondaires, mais

```
<div xmlns:dc="http://purl.org/dc/elements/1.1/">
<h2 property="dc:title">The trouble with Bob</h2>
<h3 property="dc:creator">Alice</h3>
...
</div>
```

```
<div typeof="foaf:person"
     about="http://www.w3.org/People/Berners-Lee/card#i">
   L'inventeur du Web a pour nom
   <span property="foaf:name">Timothy Berners-Lee</span>
   et pour surnom <span property="foaf:nick">Timbl</span>.
</div>
```

FIGURE 1.4 – Exemples d'utilisation de RDFa

de penser la nouvelle architecture du web et de la circulation des informations sémantiques.

RDFa permet d'intégrer des données sémantiques dans des pages web au format XHTML en respectant les relations de type « sujet-prédicat-objet » qui caractérisent le modèle de RDF. Détail non négligeable : les outils définis par le W3C ne sont utilisables que dans des documents XML puisque faisant appel aux espaces de nommage XML alors que les outils créés par la communauté du web design fonctionnent avec (X)HTML, et donc sont utilisables avec la plupart des documents actuels. Faut-il abandonner HTML et son succès majoritaire pour ne concevoir les applications web uniquement sous l'angle d'applications XML+RDF avec les contraintes d'apprentissage auprès de toute la communauté des web designers ? Ces derniers sont souvent habitués à outrepasser les limitations des techniques et de leur implémentation, mais peu enclins à soumettre leurs projets à une technique exigeante de précision et de formalisme comme la planète XML, et plus encore RDF.

### 1.4.3 Microdata

Avec le choix de HTML5 plutôt que de XHTML2, les tensions entre les partisans d'un « web HTML » et ceux d'un « web XML », et par extension entre la communauté des web designers et les instances de normalisation, est là encore apparue de façon flagrante. Le groupe de travail sur HTML5 a choisi une nouvelle technique, appelée Microdata [HICKSON 11A]. La syntaxe des Microdata, dont la figure 1.5 présente un exemple, est plus simple de la syntaxe RDFa. Étant plus proche des Microformats, elle est surtout plus proche des habitudes des développeurs web. Ce choix a été renforcé par l'initiative schema.org[11] portée conjointement Google, Yahoo ! et Bing et sur laquelle nous reviendrons.

HTML5 n'étant pas un langage XML, tous les travaux basés sur les techniques RDF devenaient inutilisables avec HTML5, ce qui semblait impensable pour toute la communauté travaillant sur RDF et le web de données. Finalement des solutions ont été trouvées pour permettre d'intégrer du RDFa dans des pages réalisées en HTML5 [SPORNY & MCCARRON 11]. De plus, pour répondre aux critiques de complexité de RDFa, la nouvelle version de RDFa [ADIDA *et al.* 11] intègre des mécanismes simplifiant son utilisation.

Le développeur web peut donc choisir entre plusieurs techniques pour intégrer des informa-

11. http://schema.org

```
<div itemscope itemtype="http://schema.org/Person">
  <span itemprop="name">Jean-Marc Lecarpentier</span>
  <img src="jml.jpg" itemprop="image" alt="Jean-Marc"/>

  <span itemprop="jobTitle">Doctorant</span>
  <span itemprop="telephone">02 31 56 73 75</span>
  <a href="mailto:lecarpentier@unicaen.fr"
     itemprop="email">lecarpentier@unicaen.fr</a>

  Page personnelle :
  <a href="http://jml.perso.info.nicaen.fr"
     itemprop="url">jml.perso.info.nicaen.fr</a>
</div>
```

FIGURE 1.5 – Exemple de code HTML utilisant les Microdata

tions dans ses pages web. Mais quelle technique utiliser ? Si les microformats semblent désormais un peu dépassés par les possibilités des Microdata et de RDFa, la présence de deux formats « concurrents » pose problème. Nous reviendrons sur les forces en jeu dans ce domaine dans la section 3.3.8.

Les possibilités d'échanges de métadonnées et d'informations au sein des pages web ont beaucoup évolué. Les échanges sont cependant loin de se limiter à des informations incluses dans les pages web. Nous reviendrons plus largement sur ce sujet dans la section 3.3, traitant des documents et de leurs métadonnées de façon plus générale.

## 1.5 Échanges de documents

Dans le cadre des échanges de documents, XML est le format de prédilection puisque indépendant des langages de programmation et des plates-formes. Plusieurs standards de vocabulaire XML sont utilisés, chacun ayant ses spécificités. Nous en présentons quelques uns ici.

### 1.5.1 PRISM

Le standard PRISM (Publishing Requirements for Industry Standard Metadata) [IDEALLIANCE 09] définit un ensemble de vocabulaires pour les professionnels de l'édition en général, et plus particulièrement dans le cadre des journaux, magazines et lettres d'information. PRISM offre un cadre pour l'échange de métadonnées et de contenus grâce à un ensemble d'éléments

les décrivant. PRISM se concentre sur les besoins des éditeurs pour la publication, l'envoi et la réception de documents composites. PRISM est basé sur les technologies XML et utilise RDF, DublinCore (présentés section 3.3) et d'autres spécifications existantes (DOI, RSS par exemple). De plus, XHTML peut être utilisé pour les échanges de contenus textuels. La figure 1.6 montre un exemple de code PRISM [12]. Cet exemple utilise PRISM Aggregator Message (PAM) qui sert à la transmission du contenu et des métadonnées.

PRISM permet les échanges de contenus très variés, mais ne spécifie pas la présentation des contenus échangés. Initialement développé pour des usages dans le domaine de l'édition imprimée, il a été étendu pour permettre les usages sur diverses plates-formes (web, mobile, etc.).

### 1.5.2 newsML

NewsML [IPTC 11a] est développé par l'IPTC (International Press Telecommunications Council) pour l'échange d'informations dans le domaine des médias d'actualité. NewsML permet l'échange des métadonnées et des contenus des brèves d'actualité de type texte, audio ou vidéo. NewsML permet ainsi d'automatiser les échanges d'informations entre différents acteurs du domaine de l'information et de l'actualité. Il inscrit les normes sociales (délais d'embargo par exemple) et juridiques (copyright,...) de la profession. NewsML est par exemple utilisé par les agences Reuters, Associated Press et l'Agence France Presse. La figure 1.7 montre un exemple de code newsML simple [13].

NewsML et PRISM sont principalement complémentaires, avec cependant quelques redondances. La spécification de PRISM précise comment utiliser NewsML à l'intérieur de données PRISM.

### 1.5.3 TEI

La TEI ou Text Encoding Initiative [TEI 07] est issue des travaux de chercheurs de Vassar College (États-Unis) en 1987. Basée initialement sur SGML et s'appuyant désormais sur XML, la TEI est une DTD qui offre un marquage des données textuelles, notamment pour les Sciences Humaines et Sociales dans le cadre d'études sur les textes littéraires ou les sources anciennes par exemple [Dornier & Buard 08]. La TEI permet de décrire le document lui-même mais aussi les apports extérieurs (annotations, commentaire éditorial, analyse, etc.). Les Presses Universitaires de Caen [14] ont par exemple défini, en collaboration avec le CLÉO [15] et revues.org [16], un sous-

12. Extrait de http://www.prismstandard.org/resources/PRISMCookbookRecipe5.pdf
13. Extrait de http://xml.coverpages.org/NewsMLForDummies.pdf
14. http://www.unicaen.fr/puc/
15. http://cleo.cnrs.fr/
16. http://revues.org

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pam:message xmlns:dc="http://purl.org/dc/elements/1.1/"
             xmlns:pam="http://prismstandard.org/namespaces/pam/2.0/"
             xmlns:pim="http://prismstandard.org/namespaces/pim/2.0/"
             xmlns:prl="http://prismstandard.org/namespaces/prl/2.0/"
             xmlns:prism="http://prismstandard/org/namespaces/basic/2.0/">
<pam:article xml:lang="en-US">
  <head>
    <dc:identifier>20080128_200801280128kennedy</dc:identifier>
    <pam:status>A</pam:status>
    <prism:originPlatform prism:platform="web"/>
    <dc:title>Kennedy Evokes JFK in Obama Endorsement</dc:title>
    <dc:creator>Liz Halloran</dc:creator>
    <prism:publicationName>USNews.com</prism:publicationName>
    <prism:publicationDate>2008-01-28</prism:publicationDate>
    <prism:killDate>2008-12-31</prism:killDate>
    <prism:channel>news</prism:channel>
    <prism:url>http://www.usnews.com/articles/news/campaign-
2008/2008/01/28/kennedy-evokes-jfk-in-obama-endorsement.html</prism:url>
    <prism:keyword>American University</prism:keyword>
    <prism:section>Nation &amp; World</prism:section>
    <prism:subsection1>Campaign 2008</prism:subsection1>
    <dc:subject>Presidential Election 2008</dc:subject>
    <prism:person>Barack Obama</prism:person>
    <prism:person>Ted Kennedy</prism:person>
    <prism:copyright>Copyright 2008 U.S. News &amp; World Report</prism:copyright>
    <prism:wordCount>455</prism:wordCount>
  </head>
  <body>
    <h1>Kennedy Evokes JFK in Obama Endorsement</h1>
    <p class="deck">Sen. Ted Kennedy's endorsement of Obama could give
the campaign a boost</p>
    <p class="byline">By Liz Halloran</p>
    <p>They made a powerful entrance...</p>
    ...
    <p>Obama paid homage to the Kennedy ....</p>
    <pam:media>
      <dc:type>Picture</dc:type>
      <pam:mediaReference pam:mimetype="image/jpg" pam:refid="FE_PR_080128kennedy185x123.jpg"/>
      <pam:credit>Jeffrey MacMillan for USN&amp;amp;WR</pam:credit>
      <pam:caption>Ted, Patrick and Caroline Kennedy all endorse
Barack Obama at a rally at American University in
Washington, DC. JFK spoke here in 1963.</pam:caption>
    </pam:media>
  </body>
</pam:article>
</pam:message>
```

FIGURE 1.6 – Exemple de code PRISM

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE NewsML PUBLIC "urn:newsml:iptc.org:20001006:NewsMLv1.0:1"
                        "http://www.afp.com/dtd/NewsMLv1.0.dtd"
  [<!ENTITY % nitf SYSTEM "http://www.afp.com/dtd/nitf-2-5.dtd"> %nitf; ]>
<NewsML>
  <Catalog Href="http://www.afp.com/dtd/AFPCatalog.xml"/>
  <NewsEnvelope>
    <DateAndTime>20000811T0818Z</DateAndTime>
  </NewsEnvelope>
  <NewsItem>
    <Identification>
      <NewsIdentifier>
        <ProviderId>afp.com</ProviderId>
        <DateId>20000811</DateId>
        <NewsItemId>010607144425.x6pxrl6k</NewsItemId>
        <RevisionId PreviousRevision="0" Update="N">1</RevisionId>
        <PublicIdentifier>urn:NewsML:afp.com:20000811:010607144425.x6pxrl6k:1</PublicIdentifier>
      </New sIdentifier>
    </Identification>
    <NewsManagement>
      <NewsItemType FormalName="News"/>
      <FirstCreated>20000811T0818Z</FirstCreated>
      <ThisRevisionCreated>20000811T0818Z</ThisRevisionCreated>
      <Status FormalName="Usable"/>
    </NewsManagement>
    <New sComponent>
      <NewsLines>
        <HeadLine>George W. Bush president (television)</HeadLine>
        <CopyrightLine>© 2001 AFP</CopyrightLine>
      </NewsLines>
      <AdministrativeMetadata>
        <Provider>
          <Party FormalName="AFP"/>
        </Provider>
      </AdministrativeMetadata>
    </NewsComponent>
  </NewsItem>
</NewsML>
```

FIGURE 1.7 – Exemple de code NewsML

```
<body>
  <div type="chapitre">
    <head type="main" aid:pstyle="T_3_Article">Les collèges jésuites et la
formation des élites : l'impact de la loi Debré</head>
    <p aid:pstyle="txt_Resume">Résumé :
      <hi rend="italic" aid:cstyle="typo_Italique">Alors qu'une grande partie ...</hi>
    </p>
    <p aid:pstyle="txt_Normal">On sait que la loi Debré du 31 décembre...
    </p>
    <p aid:pstyle="txt_Normal">Nous limiterons notre réflexion...
        <note place="foot" aid:pstyle="txt_Note" xml:id="ftn1">Par
        recteur, il faut entendre ...</note>
        ...estiment dès les...
    </p>
    <p aid:pstyle="txt_Normal">Ce constat se révélera prémonitoire.</p>
  ...
  </div>
...
</body>
```

FIGURE 1.8 – Exemple de corps d'un fichier encodé avec TEI

ensemble de la TEI adapté aux publications scientifiques en Sciences Humaines et Sociales.

La TEI est très utilisée dans le milieu des *Digital Humanities* et de l'édition universitaire. Son champ d'application est plus général que celles de PRISM ou newsML par exemple. La figure 1.8 montre un exemple de code TEI simple, limité au contenu du document.

### 1.5.4 RSS et Atom

Bien que n'étant pas spécifiquement des formats d'échanges de documents, les formats RSS1 [BEGED-DOV *et al.* 00] et Atom [NOTTINGHAM & SAYRE 05] servent à décrire des flux de documents. Le champ description de ces flux peut toutefois être utilisé pour échanger le contenu lui-même. Ils permettent la désignation de contenu multimédia pour l'intégrer dans des sites. RSS1 étant basé sur XML et RDF, le processus générique d'extension s'applique et permet de définir d'autres utilisations (podcasts par exemple).

Ces deux formats sont adaptés à la diffusion en flux (nouvelles d'agence, annonces de parution d'articles,...) ou en réponse à des requêtes. Par exemple, Atom est le format utilisé pour la navigation dans les bibliothèques ou librairies de livres numériques au format OPML.

La figure 1.9 montre un exemple de codes RSS et Atom du site du W3C.

### 1.5.5 Conclusion

Cette panoplie de systèmes d'échanges de documents permet de couvrir plusieurs usages spécifiques. Dans le cadre de la publication de documents sur le web, un utilisateur peut avoir

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
  xmlns:admin="http://webns.net/mvcb/"
  xmlns:cc="http://web.resource.org/cc/"
  xmlns="http://purl.org/rss/1.0/">
<channel rdf:about="http://www.w3.org/">
  <title>W3C News</title>
  <link>http://www.w3.org/</link>
  <description></description>
  <dc:language>en</dc:language>
  <dc:creator>W3C Staff and Contributors</dc:creator>
  <dc:date>2011-10-07T12:41:18-05:00</dc:date>
  <admin:generatorAgent rdf:resource="http://www.movabletype.org/" />
  <items>
    <rdf:Seq>
      <rdf:li rdf:resource="http://w3.org/News/2011.html#entry-9226" />
      <rdf:li rdf:resource="http://w3.org/News/2011.html#entry-9225" />
      ...
  </rdf:Seq>
  </items>
</channel>
<item rdf:about="http://www.w3.org/News/2011.html#entry-9226">
  <title>SVG Open 2011 is Around the Corner: 17-20 October</title>
  <link>http://www.w3.org/News/2011.html#entry-9226</link>
  <description>
    <![CDATA[<p>Registration is coming to a close... Learn more about
    <a href="http://www.w3.org/standards/webdesign/graphics">Graphics
    at W3C</a>.</p>]]>
  </description>
  <dc:subject>Web Design and Applications</dc:subject>
  <dc:creator>Ian Jacobs</dc:creator>
  <dc:language>en</dc:language>
  <dc:date>2011-10-07T12:41:18-05:00</dc:date>
</item>
<!-- more items -->
</rdf:RDF>
```

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
    <title>W3C News</title>
    <link rel="alternate" type="text/html" href="http://www.w3.org/" />
    <link rel="self" type="application/atom+xml"
          href="http://www.w3.org/News/atom.xml" />
    <id>tag:www.w3.org,2008-09-29://4</id>
    <updated>2011-10-07T18:00:58Z</updated>
    <generator uri="www.movabletype.org/">Movable Type 4.34</generator>
  <entry>
    <title>SVG Open 2011 is Around the Corner: 17-20 October</title>
    <link rel="alternate" type="text/html"
          href="http://www.w3.org/News/2011.html#entry-9226" />
    <id>tag:www.w3.org,2011://4.9226</id>
    <published>2011-10-07T17:41:18Z</published>
    <updated>2011-10-07T17:41:18Z</updated>
    <summary>Registration is coming to a close...</summary>
    <author>
      <name>W3C Staff</name>
    </author>
    <category term="Web Design and Applications"
              scheme="http://www.sixapart.com/ns/types#category" />
    <category term="Home Page Stories"
              scheme="http://www.sixapart.com/ns/types#category" />
    <content type="html" xml:lang="en" xml:base="http://www.w3.org/">
      <![CDATA[<p>Registration is coming... Learn more about
      <a href="http://www.w3.org/standards/webdesign/graphics">Graphics
      at W3C</a>.</p>]]>
    </content>
  </entry>
  <!-- more entries -->
</feed>
```

FIGURE 1.9 – Exemple de code RSS1 et Atom pour le même flux (W3C)

besoin de reprendre un document existant. Par exemple, si un article est publié sur un site A, de quels moyens dispose le gestionnaire du site B pour y intégrer l'article ? En supposant un accord du site source ou une licence de publication le permettant, dans la plupart des cas la solution adoptée sera un copier/coller de l'article original est un reformatage « à la main ». En effet, si les mécanismes d'import/export de contenu entre mêmes systèmes existent (par exemple le CMS Spip propose un *plugin* Spip2spip), les mécanismes pour un import/export entre systèmes de publication différents sont rares. Les systèmes de gestion de sites web construisent effectivement des documents à partir de leur modèle interne, avec peu de moyens d'échanges de documents (un format texte étant le plus utilisé pour réaliser un export...).

On comprend dès lors l'attrait d'un format pivot pour la réutilisation de contenu. Cependant, la multiplicité des formats et des usages rend cette fonctionnalité difficile à réaliser de façon satisfaisante pour diverses communautés d'utilisation. Dans le cadre de la réalisation d'un framework, il semble donc peu réaliste de pouvoir implémenter des fonctions d'import/export génériques. En revanche, le développeur web qui utilisera un framework pour créer ses applications devra avoir la possibilité de réaliser ces fonctions d'import/export en adéquation avec ses besoins.

Dans la partie précédente nous avons pointé un certain nombre d'évolutions dans les technologies utilisées sur le web. Nous avons aussi montré l'évolution d'un web de documents vers un web d'applications. Cette évolution donne au navigateur une place de choix dans la palette des outils du web.

## 1.6 Le terminal de lecture au centre

Étant donné la multiplicité des moyens d'accès au web et aux documents, l'acte de création ne suffit plus pour construire un document numérique afin de le présenter au public. Il convient de l'accompagner d'une ingénierie de composition, de lui associer diverses formes de contenu (images, vidéos, animations) et le décliner pour de multiples dispositifs de lecture, depuis les écrans des mobiles jusqu'aux ordinateurs de bureau, et dorénavant les tablettes. Une fois rédigé, avec toutes ces complexités, le document numérique doit encore trouver un chemin parmi les divers modèles de diffusion, entre l'exposition web, le streaming, la circulation de livres numériques ou les Apps pour terminaux mobiles.

Engagé au début des années 90, le travail de normalisation dont nous avons résumé l'aspect HTML dans la partie précédente, avait pour objectif de réduire la pression sur les créateurs, les compositeurs et les éditeurs de contenus numériques afin de permettre l'accès le plus universel à leurs productions. Cette nécessité relevait de deux objectifs : en finir avec la « guerre des navigateurs » et redonner au document numérique une capacité à construire la mémoire collective,

en garantissant l'archivage, et la diffusion la plus étendue. Ces objectifs restent l'apanage de toutes les tentatives de normalisation autour du document numérique

Cependant, le numérique est aussi un vaste champ de bataille économique pour les firmes technologiques. Dans ce cadre, les documents numériques deviennent des produits d'appel pour toute une série de produits (mobiles, smartphones ou ordiphones, tablettes, liseuses, nouvelles consoles de jeux, télévision numérique,...). Proposer une large gamme de contenus devient essentiel pour lancer de nouveaux appareils sur le marché. On pourrait penser que cela favoriserait la normalisation, rendant les documents indépendants des outils de lecture. Mais c'est sans compter avec la tendance de l'économie des réseaux à favoriser les monopoles autour de ce qu'on appelle « l'effet de réseau » . Maîtriser toute la chaîne de production-diffusion-lecture et avoir les moyens d'en évincer les concurrents reste la stratégie principale des firmes du secteur. Elles créent pour cela des systèmes fermés d'achat (les Apps stores) et valorisent des formats propriétaires, notamment en les défendant par des brevets puisque certains pays, en particulier les États-Unis et le Japon, acceptent le dépôt de brevets sur le logiciel, les algorithmes et les formats de données.

Nous somme revenus sur l'historique de la guerre des navigateurs (ou *Browser war*) des années 90 dans un article [Le Crosnier & Lecarpentier 10]. Nous nous attarderons ici sur les points de tension actuels entre les géants de l'Internet. La nouvelle guerre des navigateurs a en effet commencé en 2007, suite à une plainte de l'éditeur du logiciel Opera [Opera 07] reprochant à Microsoft la vente liée du système d'exploitation et du navigateur Internet Explorer. Pour éviter de nouvelles poursuites, Microsoft a négocié une solution alternative. L'accord conclu entre la Commission Européenne et la firme de Redmond prévoit un « écran de choix » (*ballot screen*) du navigateur qui est proposé à l'installation de Windows7 et pour les mises à jour de Windows Vista et XP.

Cette nouvelle donne pour le choix du navigateur a relancé le marché de ce domaine. On aurait pu penser que la Fondation Mozilla et son produit phare Firefox seraient les grands gagnants de ce nouveau mode d'installation du navigateur. Mais c'est compter sans les nouveaux acteurs, en particulier Chrome, le navigateur de Google lancé en septembre 2008. Dès la fin 2009, Google a sorti l'artillerie lourde dans des campagnes de publicité massives et impressionnantes pour son navigateur. Nombre d'utilisateurs ont été séduits par ce nouveau venu, faisant passer Google Chrome de 7% d'utilisateurs à plus de 30% en deux ans [17]. En proposant aux développeurs web des outils leur facilitant la tâche et parfaitement intégrés à son navigateur, Google séduit une population clé pour la maîtrise des contenus sur le web. Comme par exemple avec Google Chrome Frame [18] qui permet d'installer un équivalent de navigateur Chrome à l'intérieur d'Internet Explorer. Une satisfaction évidemment pour les webdesigners qui rejettent profondément les multiples manquements à la normalisation d'IE, mais un marché faustien, car les designers

17. Statistiques mensuelles http://www.w3schools.com/browsers/browsers_stats.asp
18. http://www.chromium.org/developers/how-tos/chrome-frame-getting-started

deviennent alors ceux qui pilotent, à partir de leur site, l'installation de Google, et de ses cookies de traçage, chez l'usager. La prolifération des boutons *I like* de facebook, ou `+1` récemment introduit par Google, les a rendus indispensables sur les sites. L'utilisation par Facebook ou Google des données que ces boutons permettent de collecter est d'ailleurs mise en cause par plusieurs sources [Cubrilovic 11] [Singel 11b].

Dans le domaine de la vidéo, la dépendance au logiciel utilisé est encore plus marquante. Le passage de la diffusion télévisuelle sur internet (convergence) et l'arrivée de la vidéo à la demande s'accompagnent d'une délinéarisation : l'internaute décide du moment, du lieu et de l'outil avec lequel il va accéder aux programmes audiovisuels. Une pratique renforcée par l'accès à travers les mobiles. La technique d'encodage vidéo et l'usage dans le navigateur deviennent des questions clés. Jusqu'à présent, les technologies Flash (rachetées par Adobe en 2005) étaient la seule solution pour mettre en ligne des vidéos, via divers « players ». Avec HTML5, le navigateur prend directement en charge la lecture de l'audio et la vidéo. Pour que ces médias puissent être lus sur tous les navigateurs, la norme HTML5 doit définir le codec vidéo qui sera utilisé en standard. La guerre des navigateurs se déplace ici sur les questions de format et de codec. Au sein du groupe de travail HTML5, chacun défend donc « son » format. Pas étonnant dans ces conditions que Safari et IE choisissent le codec H.264 sur lequel Apple, Microsoft et Adobe possèdent des brevets. Un codec que refuse la Fondation Mozilla à cause de la licence annuelle en contradiction avec les principes du logiciel libre. Google, qui jusqu'en 2010 était resté « neutre », s'associe à Mozilla et Opera pour lancer en mai 2010 le projet WebM[19], dont l'objectif est de créer un format vidéo optimisé pour le web, ouvert et sans royalties. En mettant sa force de frappe dans la bataille, notamment en assurant que les vidéos YouTube seront disponibles au format WebM, Google fait d'une pierre deux coups en mettant Apple et Adobe plus ou moins hors jeu dans ce domaine. Reste à voir quel choix fera le W3C pour la norme HTML5. Dans une note faisant le point sur HTML5 et la vidéo [Le Hégaret 10], Philippe Le Hégaret, *Interaction Domain Leader* au W3C, explique que le problème n°7 (ISSUE-7) du choix du codec associé à l'élément `<video>` est retiré des tâches du groupe de travail sur HTML5 faute de consensus. Plus d'un an après, le choix du codec vidéo est toujours au point mort. Les développeurs web voulant utiliser les possibilités de HTML5 se voient donc contraints d'encoder les fichiers sous plusieurs formes et de prévoir une solution de repli (*fallback*) avec un lecteur Flash.

La dépendance au logiciel utilisé se retrouve aussi dans la consultation de sites via des dispositifs mobiles. En pleine explosion, celle-ci risque de rapidement devenir le premier moyen d'accès au web. Ceci donne aux mobiles un caractère central dans la guerre des navigateurs, que l'on retrouve dans l'affrontement entre Google/Android et Apple/iPhone. Une situation qui contraint Opera et Mozilla à proposer des versions mobiles de leurs navigateurs, profitant du retard de Microsoft dans ce domaine. En excluant Flash de ses produits, Apple élimine

19. http://www.webmproject.org/

Adobe du marché de la technologie des mobiles où la société espérait imposer Flash et lance une guerre entre les grandes firmes. Une guerre du mobile que l'on retrouve au centre du modèle de distribution de contenu et d'applications, puisque celle-ci dépasse le cadre strict du Web, incluant les *Apps*, la musique, la vidéo, les livres numériques et la publicité. Le choix d'un contrôle strict par Apple sur tout ce qui peut être installé sur ses dispositifs de lecture, ou celui d'Amazon de lancer un format propriétaire de livres numériques pour sa liseuse Kindle, sont des modèles loin de faire l'unanimité. En choisissant le modèle d'agence pour son service de livres électroniques iBook Store (liberté du prix laissé à l'éditeur et partage des revenus en pourcentages), Apple a choisi de mettre les éditeurs de son côté pour contrer Amazon. Ce dernier, leader sur le marché des livres numériques, avait mis les éditeurs en grogne en imposant un « prix unique » de 9,99 dollars pour tout livre numérique.

On voit ainsi que les créateurs de contenus numériques se retrouvent dépendants des stratégies commerciales des grandes firmes, et cela jusque dans la définition même des compositions numériques et des formats de données. Une tendance qui incite ces firmes à proposer des SDK adaptés à leurs visions, sans tenir compte de l'interopérabilité. C'est en ce sens qu'il faut lire l'affrontement entre Apple et Adobe.

Car au fond, tous ces combats entre firmes n'auraient guère d'importance si des organismes forts pouvaient imposer des normes solides, pérennes, garantissant l'indépendance des producteurs de contenu et des webdesigners. Or c'est au contraire que l'on assiste. Les firmes souhaitent gagner à leur service les producteurs de contenu, avec des arguments tantôt techniques (qualité, rapidité), tantôt commerciaux (partage des revenus, de la vente ou de la publicité), tantôt de facilité d'usage (les nombreuses APIs proposées aux développeurs de sites qui les lient ensuite à leur fournisseur, notamment dans le domaine de la cartographie en ligne). Le rêve d'une instance se positionnant au delà des guerres commerciales a certainement été celui présidant à la création du W3C, qui associait à côté des entreprises du secteur des universités et des centres de recherches. Devons-nous admettre que ce rêve n'est plus d'actualité, et qu'un faisceau convergent d'intérêts, de complicités et de marchés vont balkaniser la production de contenu elle-même ? Il ne semble plus gère possible aujourd'hui pour les webdesigners et les développeurs d'application de rester en dehors de ce questionnement. Les nouveaux sujets, comme les polices de caractères embarqués (webfont), l'évolution des CSS, l'évolution du format ePub pour les livres numériques multimédias, les métadonnées embarquées (microdata et RDFa),... viennent reposer à chaque fois ces questions. Dans tous ces domaines, on assiste aussi à la volonté des grandes firmes de maîtriser la chaîne de production/diffusion/lecture de bout en bout.

La guerre à laquelle nous assistons aujourd'hui est à l'échelle du volume de données et des moyens de diffusion actuels. L'indépendance réelle du contenu vis-à-vis des plates-formes de diffusion, des outils de lecture, et des modèles d'affaire des grands acteurs du secteur devient une extension indispensable de la normalisation et de l'interopérabilité. La recherche en ce

domaine ne peut plus se limiter à l'aspect technique, mais inclure les approches issues des sciences économiques, humaines et sociales.

## 1.7 Synthèse

Dans cette présentation des évolutions du Web, nous avons cherché à souligner les points qui nous semblent importants pour notre projet Sydonie :

- faire coïncider la logique des documents de XHTML et celle des applications de HTML5 ;
- penser que les documents valent en relation avec leur métadonnées globales (description de l'ensemble du document) ou attachées à des termes spécifiques contenus dans le document ;
- même s'il semble difficile de définir un format pivot universel, la réalisation d'un framework d'édition doit intégrer la logique d'un format d'échange ;
- la production et la diffusion de documents numériques est en réalité un champ de bataille commun dans lequel les acteurs de la normalisation et les web designers jouent un rôle essentiel mais sont confrontés à des puissances économiques capables de leur imposer des choix.

Nous avons utilisé dans toute cette partie le fil conducteur du travail des web designers. Dans le chapitre suivant, nous nous appuierons sur l'analyse faite par les bibliothécaires pour la description des documents et de l'organisation de l'espace documentaire.

# Chapitre 2

# FRBR ou la révolution des bibliothèques

**Sommaire**



Dès le début des années 1990, un groupe de travail au sein de l'IFLA (International Federation Library Association) [20] a commencé à réfléchir sur de nouvelles méthodes pour la description de documents dans l'univers bibliographique. Ces travaux ont abouti en 1998 à la publication du rapport final sur les spécifications fonctionnelles des notices bibliographiques, plus connu sous l'acronyme FRBR (Functional Requirements for Bibliographic Records).

Avec un changement radical dans la vision bibliographique des documents, FRBR est une véritable révolution pour les bibliothèques. Ce chapitre présente les principes des FRBR et détaille en particulier les aspects qui seront repris dans nos travaux.

20. http://www.ifla.org/

## 2.1 Documents et bibliothèques

Les bibliothèques doivent gérer un ensemble de documents au travers de catalogues. Un catalogue est un registre des objets présents dans une bibliothèque. Les objectifs d'un catalogue [Cutter 76] sont de permettre à un utilisateur de :

- trouver un livre dont il connaît l'auteur, le titre, le sujet ou la catégorie ;
- lister les ouvrages de la bibliothèque pour un auteur, un sujet ou un type de livre donné ;
- aider au choix d'un livre. Ceci implique de pouvoir connaître les différentes éditions d'une même œuvre.

Les catalogues gèrent désormais bien plus que des livres mais les objectifs peuvent être résumés par la définition ci-dessus. Les notices bibliographiques composant les catalogues contiennent des informations (métadonnées) de plusieurs types :

- métadonnées descriptives : auteur(s), sujet, type d'ouvrage, éditeur, traducteur, format, etc. Ces métadonnées renseignent sur le contenu intellectuel et son origine ;
- métadonnées administratives : n$^o$ d'inventaire, statut de prêt, etc. Ces métadonnées renseignent sur l'objet concret présent dans la bibliothèque ;
- métadonnées documentaires permettant de retrouver la ressource : indexation par mots-clés, classification par exemple ;
- éventuellement des métadonnées de structure, par exemple composition d'une œuvre en plusieurs volumes.

L'expérience des bibliothèque est intéressante puisque l'objectif du catalogue est de permettre la découverte de documents sans avoir à consulter leur contenu. Nous devrions typiquement retrouver ces fonctionnalités dans tout système de gestion de documents en ligne. En effet, un utilisateur doit pouvoir accéder rapidement à une information synthétique sur les documents qui l'intéresse, afin de pouvoir procéder à son choix. Ces principes se retrouvent au sein du web. Un document sur le web doit pouvoir être « catalogué ». Pour l'instant, les catalogues du Web sont réalisés par les moteurs de recherche qui indexent et listent les documents. Cependant, de nombreuses applications plus spécifiques (secteurs professionnels, communautés d'usage,...) ont besoin d'une autre forme d'organisation et de mise en relation entre les documents. C'est à ce titre que nous pouvons apprendre de la façon dont les bibliothèques ou les musées conçoivent des catalogues.

Les notions et principes de catalogage ont été bouleversé par un rapport d'étude publié en 1998 sur les spécifications fonctionnelles des notices bibliographiques. Débutés en 1990, ces travaux de l'IFLA avaient pour objectif de recommander un niveau minimal de spécifications et de fonctionnalités appliquées aux notices. En élaborant un cadre conceptuel simplement et précisément exprimé, le rapport final sur les Functional Requirements for Bibliographic Records [Madison 98] définit un modèle de catalogage dont nous montrerons les applications

possibles pour la gestion de documents numériques composites.

Le rapport sur les FRBR apporte des changements radicaux dans la manière de voir et de rédiger des notices bibliographiques. En effet, les règles traditionnelles de rédaction de notices étaient jusqu'à présent basées sur l'exemplaire dans les mains du bibliothécaire. Cela entraîne une multiplication des fiches pour les diverses éditions, traductions, supports, exemplaires d'une même œuvre. Renversant la méthodologie, le modèle des FRBR reconstruit l'ensemble des documents issus d'une même production intellectuelle et offre une nouvelle vue sur les structures et les relations bibliographiques. FRBR définit un vocabulaire précisant les termes « œuvre », « édition », « exemplaire », « personne », etc. et définit trois groupes d'entités. Nous nous intéressons d'abord ici au premier, en reprenant largement le rapport FRBR lui-même et sa traduction française.

## 2.2 Entités du groupe 1 des FRBR

### 2.2.1 Terminologie

L'analogie avec le mot « livre » utilisée par [Tillett 03] présente bien les termes fondamentaux des FRBR. En langage courant, parler d'un « livre » peut signifier plusieurs choses.

On peut par exemple utiliser le mot « livre » pour désigner un objet physique qui est un ensemble de pages reliées, et qui peut éventuellement servir à caler une table. FRBR désigne cet usage du mot « livre » par le terme Item.

On peut aussi utiliser le mot « livre » chez son libraire. On connaît alors peut-être l'ISBN, mais l'objet physique lui-même importe peu, du moment qu'on en trouve un. FRBR désigne cet usage du mot « livre » par le terme Manifestation.

Mais on utilise « livre » en signifiant « qui a traduit ce livre », dans le sens où on recherche le livre dans une langue précise. FRBR appelle cela une Expression.

Enfin, on peut parler d'un « livre » en parlant de « qui a écrit ce livre ». Cela signifie que l'on désigne le livre avec un niveau d'abstraction supérieur, désignant plutôt l'histoire racontée ou le contenu intellectuel du livre (indépendamment des versions linguistiques). Les FRBR utilisent le terme Work dans ce cas.

Malgré l'existence d'une traduction française du rapport final des FRBR, notons dès à présent que nous avons volontairement choisi les dénominations anglaises des entités du groupe 1. En effet la traduction française produite par la BnF [Madison 01] utilise le terme « Œuvre » pour désigner l'entité Work et « Document » pour l'entité Item. Ce choix nous semble porteur de confusion, en particulier celui de « Document » pour l'entité Item puisque le terme document, de façon similaire à « livre » ci-dessus, peut être interprété de diverses façons. Le choix de la traduction nous semble en contradiction avec la définition élargie de Roger T. Pédauque [Pédauque 06]

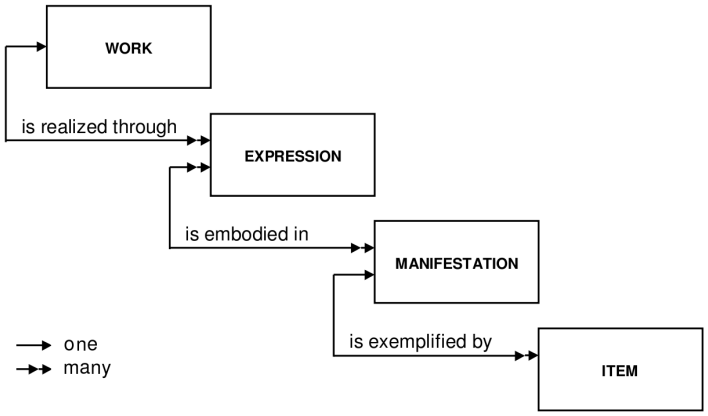



FIGURE 2.1 – FRBR : Entités du Groupe 1 et relations fondamentales

de ce qu'est un document numérique.

Les entités du premier groupe, à savoir Work, Expression, Manifestation et Item (voir figure 2.1) représentent les différents aspects de ce qu'un utilisateur peut trouver dans les produits d'une activité intellectuelle ou artistique.

Les entités définies comme Work (c'est-à-dire, une création intellectuelle ou artistique déterminée) et comme Expression (c'est-à-dire, la réalisation intellectuelle ou artistique d'une entité Work) en expriment le contenu intellectuel ou artistique.

Les entités définies comme Manifestation (c'est-à-dire, la matérialisation de l'une des Expressions d'une entité Work) et comme Item (c'est-à-dire, un exemplaire isolé d'une Manifestation), à l'inverse, en expriment la forme matérielle.

Les relations décrites dans le schéma 2.1 montrent qu'une entité Work peut trouver sa réalisation dans une ou plus d'une Expression. Une Expression, de son côté, constitue la réalisation d'une seule et unique entité Work. Une Expression peut se concrétiser en une ou plus d'une Manifestation et une Manifestation peut matérialiser une ou plus d'une Expression. Enfin, une Manifestation peut être représentée par un ou plus d'un Item, mais un Item ne peut jamais représenter plus d'une Manifestation.

Chacune des entités définies dans le modèle possède un ensemble de caractéristiques ou d'attributs, grâce auxquels les utilisateurs peuvent effectuer des recherches et visualiser les résultats. Le rapport FRBR distingue les attributs inhérents à une entité et les attributs forgés.

La première catégorie comprend les caractéristiques physiques (par exemple, la matière et les dimensions d'un objet) et les éléments distinctifs que l'on peut qualifier d'informations principales (par exemple, des mentions figurant sur la page de titre, la couverture, ou le boîtier). Ces attributs peuvent en général être renseignés par un examen de l'objet à cataloguer.

La deuxième catégorie, les attributs forgés, comprend des identifiants affectés à une entité

(par exemple, un numéro de catalogue pour un ouvrage), et des informations contextuelles (par exemple, le contexte politique dans lequel une œuvre a été conçue). Le contenu de ces attributs fait souvent appel à une source extérieure.

Les FRBR définissent les attributs des entités à un niveau logique, exprimés en termes d'éléments qu'un utilisateur pourrait juger caractéristiques d'une entité plutôt que comme des données spécifiques définies par les personnes chargées du catalogage. Dans le cadre strict du rapport des FRBR, un attribut a en général une valeur unique. Dans certains cas cependant, un attribut pourra avoir une valeur multiple. Par exemple, dans le cas où une œuvre est connue sous plusieurs noms, l'attribut titre pourra apparaître plusieurs fois.

Les parties suivantes s'attachent à présenter plus en détails les entités Work, Expression, Manifestation et Item.

### 2.2.2 Entité Work

L'entité Work (Œuvre en français) représente une création intellectuelle ou artistique déterminée. Work est une entité abstraite ne correspondant à aucun objet matériel.

On reconnaît l'entité Work à travers des réalisations individuelles que sont les Expressions, mais le Work n'existe que par sa présence conceptuelle dans les diverses Expressions. Par exemple, l'entité Work « Le rouge et le noir » ne réfère pas à à tel ou tel état du texte de l'œuvre ou à une version linguistique particulière, ni à une version abrégée par exemple. L'entité Work réfère à la création intellectuelle sous-jacente à toute la palette d'Expressions qui lui sont rattachées.

En raison du caractère abstrait de la notion de Work, les frontières entre Work et Expression sont difficiles à définir précisément et peuvent dépendre du contexte culturel. Dans le cadre des FRBR, les différents états d'un texte, comme les versions révisées, annotées, augmentées ou abrégées par exemple, constituent des Expressions de la même entité Work. De même, les traductions d'une langue dans une autre, les transcriptions et arrangements musicaux, les versions doublées ou sous-titrées d'un film représentent des Expressions de la même entité Work.

#### Exemples

L'œuvre de Stendhal *Le rouge et le noir* constitue une entité Work. Le texte original en français est une Expression, en l'occurrence il s'agit de la première Expression apparue. Diverses traductions existent. Le processus de traduction étant un travail intellectuel conséquent, une nouvelle Expression est créée pour chaque traduction, même si la langue cible est la même. En ce qui concerne la version audio, on considère là aussi que la lecture à voix haute est un travail intellectuel qui mérite la création d'une Expression. Une version audio lue par une autre personne amènerait à une Expression différente. L'ensemble peut alors être représenté par la structure suivante :

**Work** *Le rouge et le noir* de Stendhal

- **Expression 1** Texte original en français
- **Expression 2** Texte en français lu par Michel Vuilermoz de la Comédie Française
- **Expression 3** *The Red and the Black*, traduit en anglais par Charles Kenneth Scott-Moncrieff en 1926
- **Expression 4** *The Red and the Black*, traduit en anglais par Burton Raffel en 2003
- **Expression 5** *Rot und Schwarz*, traduit en allemand par Arthur Schurig

Dans le cadre d'un film, les versions doublées ou sous-titrées représentent de nouvelles Expressions, par exemple :

**Work** *Jules et Jim* (film)

- **Expression 1** la version originale en français
- **Expression 2** la version originale sous-titrée en anglais

Lorsque l'intervention sur l'entité Work implique un degré significatif de recherche intellectuelle ou artistique autonome, le résultat est alors une nouvelle entité Work. Par exemple les réécritures, adaptations pour enfants, parodies, adaptations théâtrales ou cinématographiques constituent de nouvelles entités Work. Les *abstracts*, *digests* et résumés sont également réputés constituer de nouveaux Work. Par exemple :

**Work 1** *Le rouge et le noir* de Stendhal

**Work 2** *Le rouge et le noir*, film franco-italien de Claude Autant-Lara, 1954, avec Gérard Philipe

**Work 3** *Le rouge et le noir*, téléfilm de Jean-Daniel Verhaeghe, 1998, avec Carole Bouquet

La figure 2.2, extraite de [Tillett 01] montre la frontière entre la création d'une nouvelle entité Expression ou d'une nouvelle entité Work, qui dépend du travail intellectuel entre les versions.

Les attributs de l'entité Work définis par les FRBR sont :

- titre de l'œuvre : le titre sous lequel elle est connue. Plusieurs titres peuvent être mentionnés, le « titre uniforme » étant l'appellation communément acceptée ;
- forme de l'œuvre : par exemple roman, pièce de théâtre, symphonie, concerto, carte, dessin, photographie, etc ;
- date de l'œuvre : date originelle de la création. Cela peut être un intervalle de dates ;
- autre caractéristique distinctive qui permet de distinguer deux Works ayant le même titre ;
- complétude visée : indique si l'œuvre est terminée ou non ;
- public visé ;
- contexte de l'œuvre : contexte historique, social, intellectuel, artistique ou autre au sein duquel l'œuvre a été conçue.

D'autres attributs existent qui dépendent du type de l'œuvre (musicale, cartographique par

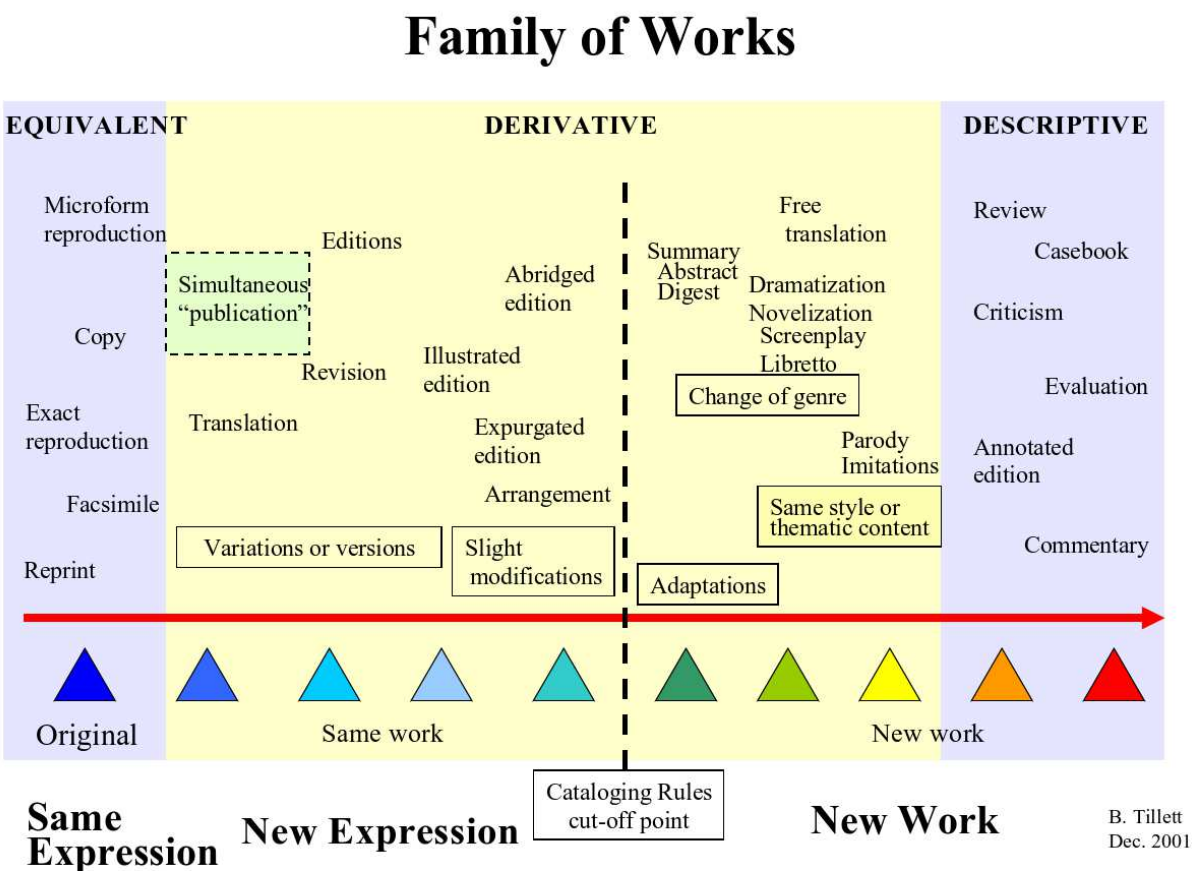



FIGURE 2.2 – Nuances entre création d'une nouvelle entité Expression ou Work

exemple).

D'un point de vue pratique, l'entité Work permet de nommer la création intellectuelle ou artistique abstraite qui englobe toutes les Expressions. Elle permet de plus d'établir des relations entre entités. Par exemple, une entité Work *Critique littéraire du rouge et le noir* pourra être reliée au Work *Le rouge et le noir*, puisque cette critique ne se réfère pas à une Expression particulière mais à l'œuvre en général. D'autres relations existent que nous présenterons au chapitre 3. Il est important de noter ici que le rapport FRBR ne donne pas de liste exhaustive de relations ou de règles *normatives* pour les utiliser.

### 2.2.3 Entité Expression

Une Expression est la réalisation intellectuelle ou artistique d'une entité Work sous une forme quelconque (sonore, visuelle, etc.). Par exemple les versions linguistiques d'une même œuvre (c.f. exemple précédent avec *Le rouge et le noir*), les divers enregistrements d'une même œuvre sonore, ou diverses éditions revues et corrigées constituent différentes Expressions de la

même entité Work. L'entité Expression permet de rendre compte des différences de contenu intellectuel ou artistique qui peuvent exister entre deux réalisations d'une même entité Work.

**Exemple**

L'exemple ci-dessous complète ceux présentés section 2.2.2. De même que pour les exemples précédents, la création de nouvelles Expressions se justifie par le travail intellectuel que nécessite l'interprétation de l'œuvre.

**Work** Le quintette *La truite* de Franz Schubert
- **Expression 1** la partition du compositeur
- **Expression 2** interprétation par le quatuor Amadeus avec Hephzibah Menuhin au piano
- **Expression 3** interprétation par le Cleveland quartet avec Yo-Yo Ma au violoncelle

Les attributs de l'entité Expression sont nombreux, nous n'en citerons que les principaux :
- titre de l'expression : de même que pour l'entité Work, il peut y en avoir plusieurs ;
- forme de l'expression : il s'agit ici du moyen par lequel le Work est réalisé (par exemple, sous forme de sculpture, de danse, de mime, etc.) ;
- date de l'expression : il s'agit de la date à laquelle l'Expression a été créée, par exemple la date d'une traduction ;
- langue de l'expression : langue dans laquelle le Work est exprimé au travers de cette Expression ;
- résumé du contenu : cela englobe un abstract, un sommaire, un synopsis, etc., ou bien une table des matières, une liste des chansons contenues dans l'Expression ;
- restrictions d'usage de l'expression : par exemple des mentions de copyright.

L'entité Expression permet de rendre compte des différences de contenu intellectuel ou artistique entre deux réalisations d'une même entité Work. De même que pour les entités Work, des relations sont définies au sein du modèle FRBR entre les Expressions pour exprimer les variations entre celles-ci (abrégé, traduction, etc.), par exemple pour identifier le texte sur lequel a été établie une traduction. Nous reviendrons là aussi sur ces relations dans le chapitre 3.

### 2.2.4 Entité Manifestation

Manifestation, troisième entité définie dans le modèle, représente la matérialisation de l'Expression d'un Work. Quand une entité Work a trouvé sa réalisation, l'Expression qui en résulte peut se concrétiser matériellement sur divers supports (papier, vidéo, toile, CD, etc). Chaque matérialisation constitue une Manifestation différente. Il peut n'exister qu'un seul exemplaire de

cette Manifestation (par exemple l'original d'un tableau) ou de multiples exemplaires (diffusion ou distribution publique d'un ouvrage).

Le contenu intellectuel et la présentation matérielle délimitent les frontières entre une Manifestation et une autre. Quand le processus de production induit des modifications dans la présentation matérielle, alors le produit qui en résulte est représenté par une nouvelle Manifestation. C'est le cas par exemple lors de modifications visuelles (réédition d'un ouvrage avec une couverture différente) ou de support (VHS/DVD par exemple). De même lorsque les modifications impliquent un éditeur, producteur ou distributeur différent, alors une nouvelle Manifestation est créée. Cependant si les modifications affectent le contenu intellectuel ou artistique, alors une nouvelle Expression doit être créée pour l'entité Work.

#### Exemple

Nous complétons ici la représentation de l'œuvre *Le rouge et le noir* avec diverses éditions [21] :

**Work** *Le rouge et le noir* de Stendhal

- **Expression 1** Texte original en français
  - **Manifestation 1** Édition originale chez Levasseur, 1830
  - **Manifestation 2** Édition chez Robert Laffont, 1991
  - **Manifestation 3** Édition chez Gallimard, 1935
- **Expression 2** Texte en français lu par Michel Vuilermoz de la Comédie Française
  - **Manifestation 1** Édition Thélème, 2008
- **Expression 3** *The Red and the Black*, traduit en anglais par Charles Kenneth Scott-Moncrieff en 1926
  - **Manifestation 1** Édition chez The Modern library, 1929
  - **Manifestation 2** Édition chez Limited Editions Club, 1947
- **Expression 4** *The Red and the Black*, traduit en anglais par Burton Raffel en 2003
  - **Manifestation 1** Édition chez The Modern Library, 2003

De même que pour Expression, les entités Manifestation peuvent avoir de nombreux attributs, en particulier pour contenir des détails techniques. Nous n'en citons ici aussi que quelques-uns :

- titre de la manifestation : de même que pour les entités Work et Expression, il peut y en avoir plusieurs ;
- éditeur/diffuseur : le nom de l'éditeur ;
- date d'édition/diffusion : date à laquelle la Manifestation a été mise au public ;
- lieu d'édition/diffusion : localisation de l'éditeur ;
- mention d'édition/numéro : $2^e$ édition par exemple ;

21. Les éditions sont choisies au hasard parmi les nombreuses éditions existantes.

– type de support : papier, cassette audio, CD, etc. ;

L'entité Manifestation permet de nommer et décrire les caractéristiques communes à un ensemble d'exemplaires, qui seront chacun matérialisé par des entités Item. Ces caractéristiques peuvent être physiques (support, format, etc.) ou liées aux conditions de distribution afin de permettre à un utilisateur de choisir une Manifestation appropriée à ses besoins. Elle permet aussi d'établir des relations entre des Manifestations particulières d'une œuvre.

### 2.2.5 Entité Item

L'entité Item représente un exemplaire isolé d'une Manifestation : l'objet physique dans les mains du bibliothécaire. Un Item constituant un exemplaire d'une Manifestation a un contenu identique au contenu de la Manifestation elle-même. Des variantes peuvent exister entre les Items, par exemple un exemplaire avec dédicace de l'auteur, un exemplaire annoté par l'auteur ou une autre personne, un exemplaire détérioré, etc.

#### Exemple

**Work** *Le rouge et le noir* de Stendhal

- **Expression 1** Texte original en français
  - **Manifestation 1** 2$^e$ Édition chez Levasseur, 1831
    - **Item 1** Exemplaire conservé à la BnF, cote Y2- 69747
    - **Item 1** Exemplaire conservé à la BnF, cote Y2- 69752
  - **Manifestation 2** Édition chez Robert Laffont, 1991
    - **Item 1** Exemplaire conservé à la BnF, cote 84/34 STEN 2
    - **Item 1** Exemplaire conservé à la Bibliothèque de Caen, cote A 31516
- **Expression 2** Texte en français lu par Michel Vuilermoz de la Comédie Française
  - **Manifestation 1** Édition Thélème, 2008
    - **Item 1** Exemplaire conservé à la BnF, cote SDC 12- 240215
- **Expression 3** *The Red and the Black*, traduit en anglais par Charles Kenneth Scott-Moncrieff en 1926
  - **Manifestation 1** Édition chez The Modern library, 1929
    - **Item 1** Exemplaire conservé à la British Library, cote MP1.0000333765.0.1
    - **Item 1** Exemplaire conservé à la British Library, cote MP1.0000333765.0.4
  - **Manifestation 2** Édition chez Limited Editions Club, 1947
    - **Item 1** Exemplaire conservé à la Library of Congress, cote PQ2435.R7 E5

L'entité Item possède des attributs permettant, entre autres, de spécifier la provenance de celui-ci (achat, don, etc.), les annotations ou inscriptions figurant sur l'Item, son état.

L'entité Item permet d'identifier chaque exemplaire de la Manifestation et éventuellement d'établir des relations entre exemplaires isolés.

### 2.2.6 Structure arborescente et logique du lecteur

Les exemples ci-dessus montrent que les entités FRBR structurent le document sous la forme d'un arbre dont l'entité Work est la racine. Concernant un exemplaire particulier (entité Item), l'ensemble de ses caractéristiques sont recomposées en fusionnant les informations obtenues en parcourant la liste des ancêtres de l'Item. La structure arborescente est illustrée par la figure 2.3.

Cette logique reprend les pratiques des lecteurs : le moment de recherche globale d'une œuvre précède le choix d'une édition puis d'un exemplaire en fonction de ce que propose le catalogue. Le catalogage permet dans ce cas de proposer le choix de l'entité Work (recherche d'un titre dans le catalogue par exemple) puis de proposer la liste des éditions ou versions disponibles (liste des entités Expression). La Manifestation peut alors être choisie et le lecteur peut trouver l'exemplaire disponible (Item) qu'il recherche.

Les entités du groupe 1 modélisent les documents présents dans un système documentaire, en introduisant quatre niveaux de conceptualisation. Les attributs des entités permettent de stocker les informations correspondant à chaque niveau conceptuel. Le système documentaire peut ainsi reconstituer les informations complètes d'une entité en remontant la branche de l'arbre ainsi constitué.

Cependant, les entités ne possèdent pas d'attribut permettant de mentionner un auteur, traducteur, compositeur, organisme, etc. qui intervient dans la création d'un Work ou d'une Expression. Pour cela le rapport FRBR définit deux entités, Person et CorporateBody, qui composent le groupe 2. Les entités du groupe 3 permettent de décrire des concept, évènements, objets et lieux qui sont éventuellement présents dans les œuvres.

## 2.3 Entités des groupes 2 et 3 et relations

Les entités du groupe 2 représentent les personnes physiques ou morales qui ont la responsabilité du contenu intellectuel ou artistique, de la production matérielle et de la distribution, ou de la gestion juridique des entités du premier groupe. Le rapport définit à cet effet les entités Person et Corporate Body [22].

22. Nous avons choisi de continuer à utiliser les versions originales des noms des entités par souci de cohérence avec la section précédente.

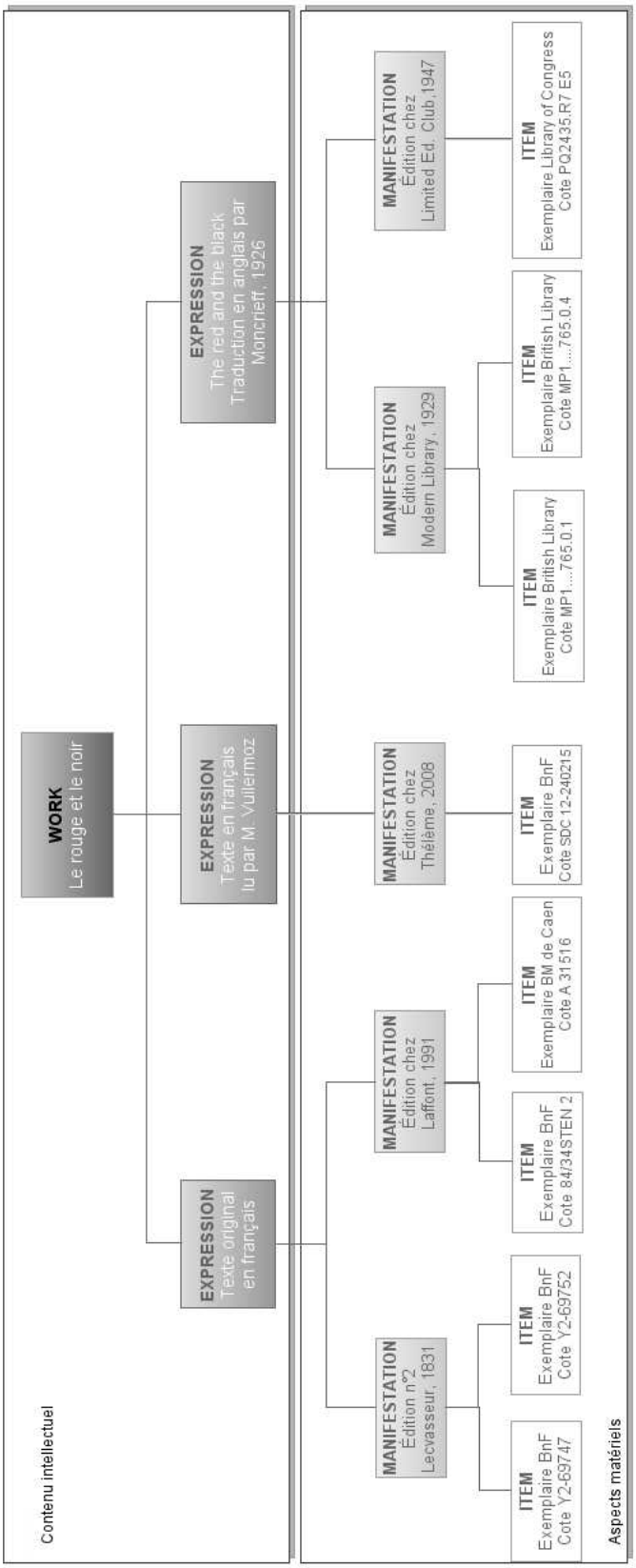



FIGURE 2.3 – FRBR : Structure arborescente du Document

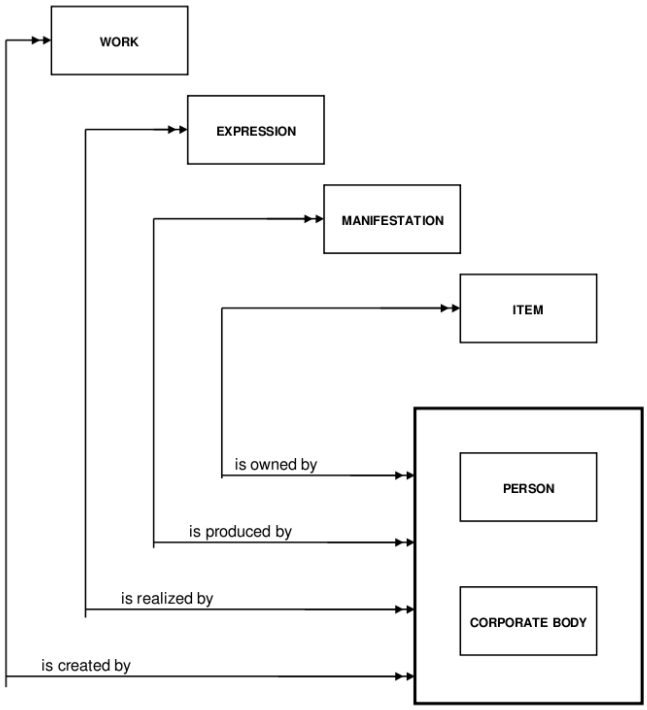



FIGURE 2.4 – Liens de responsabilité entre entités du $1^{er}$ et du $2^{ème}$ groupes

Les relations de responsabilité entre les entités du premier groupe et celles du second groupe sont définies dans le rapport des FRBR et sont illustrées figure 2.4. Ces relations reflètent les rôles des personnes par rapport aux entités Work, Expression, Manifestation et Item. Ces informations sont importantes pour les opérations effectuées par les utilisateurs et la navigation dans l'univers bibliographique.

Enfin l'ensemble des entités du groupe 3 représentent les sujets des œuvres. Les entités du groupe 3 sont Concept, Object, Event et Place. Des relations de sujet peuvent exister entre une entité Work et les entités du $3^{ème}$ groupe, mais aussi entre le Work et des entités du $1^{er}$ ou $2^{ème}$ groupe, comme le représente la figure 2.5.

L'expression de « relations » entre entités est un des points clés des FRBR, permettant ainsi d'exprimer les notions de *rôle*, de *sujet* comme définis pour les groupes 2 et 3. Le rapport FRBR définit aussi d'autres relations, celles-ci entre les contenus. Ces relations permettent d'exprimer la notion de *travail dérivé*, par exemple dans le cas de traductions, d'adaptations, etc. Elles expriment aussi des relations d'ensemble à partie, par exemple dans le cas d'œuvres en plusieurs volumes ou de d'œuvres incluses dans une autre. Ces relations sont établies entre des entités Work ou Expression selon les cas, en fonction des contenus mais aussi des besoins du catalogue constitué. Nous reviendrons sur ces relations dans le chapitre 3.

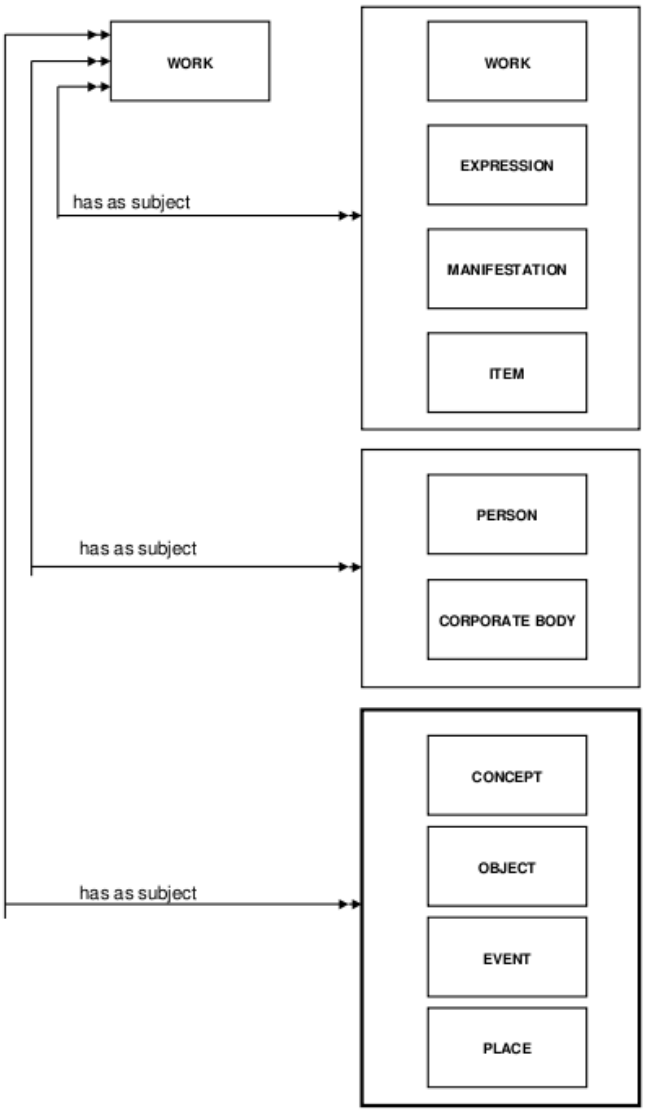



FIGURE 2.5 – Liens de sujet entre Work et entités du $1^{er}$, $2^{ème}$ ou $3^{ème}$ groupes

## 2.4 Synthèse

Nous avons présenté dans cette partie le modèle élaboré par les spécifications fonctionnelles des notices bibliographiques. En proposant un cadre conceptuel pour la description de documents, les FRBR posent les bases d'une représentation des divers formats, versions, etc. d'une même création intellectuelle représentée dans l'entité Work.

Si les attributs et les relations définis dans le rapport sont clairement orientés pour les besoins des bibliothèques, les concepts posés sont la base de notre travail et du modèle pour les documents numériques que nous proposons dans la partie II et que nous introduirons dans le chapitre suivant.

Les relations entre entités du groupe 1 décrivent les relations fondamentales, expriment les liens ensemble/partie ainsi que les relations entre versions linguistiques. Nous présenterons celles-ci plus en détail dans le chapitre 3 et étudierons leurs applications possibles dans le domaine des documents composites et des documents multilingues.

Enfin, les groupes 2 et 3 sont encore peu utilisés dans le framework Sydonie à l'heure actuelle. Cependant, certaines parties du framework comme la politique des permissions utilisent directement les concepts posés par le groupe 2 et les relations qu'il définit.

De plus, les groupes 2 et 3 s'insèrent directement dans le monde du web actuel où les personnes, lieux, concepts, etc. peuvent être identifiés par un URI sur le web et reliés entre eux.

# Chapitre 3

# Document numérique

## Sommaire



Sans reprendre ici l'ensemble des travaux sur le document numérique, ce chapitre traite de trois aspects du document numérique sur lesquels notre travail est principalement axé. Document composite d'abord. Nous détaillons comment les FRBR, présentés précédemment, interprètent

cette notion et le modèle qu'ils proposent. Nous abordons ensuite la problématique des document multilingue. Là encore, nous étudierons les concepts introduits par le rapport sur les FRBR, puis nous nous intéressons au découplage de l'interface des applications web et de leur contenu. Enfin, nous abordons le domaine des métadonnées et des documents.

## 3.1 Document composite

### 3.1.1 Introduction

Nous appelons document composite un document dont le contenu provient de plusieurs sources, par exemple un texte illustré par une ou plusieurs images. Ou encore un texte agrémenté de divers « encarts » présentant des compléments d'informations.

Comment donc considérer cet ensemble d'informations qui constitue un document ? Une erreur fréquente est de considérer un document composite comme un ensemble de fichiers, le tout étant organisé de façon à être présenté de façon compréhensible par un lecteur. Cette erreur est liée aux débuts du Web et au fait que les contenus étaient pilotés par la technique plutôt que par la conception. Par exemple pour une image, le contenu était simplement exprimé en HTML 4, et la problématique des web designers était plus de savoir comment mettre des images dans les pages web que de réaliser l'insertion d'un document image.

Dans l'exemple d'un texte illustré par une ou plusieurs images, considérer ce document comme du texte agrémenté de fichiers images revient à négliger complètement la dimension documentaire de chacun de ces élément :

- titre, description, auteur(s) au minimum ;
- informations complémentaires sur chacun des documents de la composition : cela peut aller des informations sur la création ou production du document jusqu'au informations techniques sur sa présentation (résolution par ex.), en passant par des informations sur les droits liés au document.

Il y a entre ces éléments des relations de dépendance qui sont de l'ordre sémantique et ne présument en rien de la *présentation* des composants les uns par rapport aux autres. Ces relations peuvent être de diverses nature : illustration (image, audio ou vidéo), encart, commentaire, etc. Ce sont donc des relations nommées. Notons de plus que l'on retrouve dans ce cas certaines des problématiques présentées dans le chapitre 1 avec l'échange de documents et dont PRISM est une des solutions proposées. Ces divers points illustrent la nature transversale des problématiques abordées au cours de nos travaux.

### 3.1.2 Relations FRBR pour les documents composites

Nous avons présenté au chapitre 2 comment les FRBR modélisent un document au moyen des quatre entités Work, Expression, Manifestation et Item. Ces entités sont liées par les relations de réalisation et de matérialisation qui permettent d'obtenir la structure arborescente d'une œuvre. Nous avons présenté de plus les relations entre les quatre entités du groupe 1 et des entités des groupes 2 et 3 qui permettent d'exprimer des informations concernant la responsabilité ou le sujet des œuvres.

En plus des relations fondamentales entre les entités du premier groupe, nous nous intéressons ici aux relations supplémentaires définies par FRBR. Le rapport sur les FRBR définit différents types de relations entre instances des ces entités. Pour chaque type, le rapport donne une liste de relations possibles, mais « sans prétendre à l'exhaustivité ».

FRBR définit des relations entre entités Work ou entre un Work et une Expression pour exprimer par exemple des relations :

- de suite : par exemple *La château de ma mère* de Pagnol est la suite de *La gloire de mon père* ;
- d'adaptation : par exemple le film *La gloire de mon père* d'Yves Robert est une adaptation du roman du même nom ;
- de complément : par exemple *Forest Gump, The Soundtrack* est un complément du film *Forest Gump*.

FRBR définit en particulier des relations ensemble/partie qui expriment diverses notions du document composite. Ces relations ensemble/partie peuvent être établies entre œuvres (entités Work), Expressions ou Manifestations. Nous nous attarderons ici sur ce type de relations et regardons leur signification selon les entités considérées.

Entre entités du type Work, la relation *a une partie* ou *est une partie de* permet de spécifier qu'une œuvre est composée d'un ensemble d'autres œuvres. Le tableau 3.1 reprend les exemples du rapport FRBR.

Les relations ensemble/partie sont réparties en deux catégories. La première, les relations avec des parties dépendantes, décrit les parties composantes d'une entité Work qui sont destinées à être utilisées dans le contexte de ce Work. Par exemple, les parties, chapitres, etc. appartiennent à cette catégorie. Dans le cadre du catalogage, ces parties dépendantes n'ont, *à priori*, pas vocation à avoir une notice bibliographique leur correspondant.

La seconde catégorie correspond aux parties indépendantes, c'est-à-dire celles qui ne dépendent pas, pour leur signification, du contexte fourni par l'œuvre plus large. En général, les parties composantes indépendantes ont des noms/titres distincts. Dans le cadre des bibliothèques, ces parties indépendantes sont souvent amenées à avoir leur propre notice bibliographique.

| **Type de relation** | **Partie dépendante** | **Partie indépendante** |
|---|---|---|
| Ensemble/partie<br>a une partie $->$<br>$<-$ est une partie de | Chapitre, section,<br>partie, *etc.*<br><br>Volume ou numéro d'un<br>périodique<br>Partie intellectuelle d'une<br>œuvre en plusieurs parties<br><br>Illustration d'un texte<br>Bande son d'un film | Monographie dans une<br>collection<br>Article de journal ou de<br>revue<br><br>Partie intellectuelle d'une<br>œuvre en plusieurs parties |

TABLE 3.1 – Relations ensemble/partie entre une œuvre et une œuvre

| **Type de relation** | **Partie dépendante** | **Partie indépendante** |
|---|---|---|
| Ensemble/partie<br>a une partie $->$<br>$<-$ est une partie de | Table des matières, *etc.*<br><br>Volume ou numéro d'un<br>périodique<br>Illustration d'un texte<br>Bande son d'un film | Monographie dans une<br>collection<br>Article de journal<br>Partie intellectuelle d'une<br>œuvre en plusieurs parties<br>Rectificatif |

TABLE 3.2 – Relations ensemble/partie entre une expression et une expression

FRBR se refuse à tracer des frontières rigides entre ce qui est dépendant ou non. Une photographie insérée dans un article peut relever des deux catégories. D'une part, elle existe comme document indépendant avec ses métadonnées spécifiques (auteur, sujet, date, etc.). Mais elle existe aussi comme document dépendant, par exemple la légende associée peut varier et sera adaptée au texte.

Les relations ensemble/partie entre Expressions sont du même type que celles définies au niveau Work, avec quelques variations puisque ces parties doivent faire référence à des caractéristiques de l'Expression. Le tableau 3.2 reprend les exemples du rapport FRBR.

Enfin, les relations entre Manifestations, présentées dans le tableau 3.3, sont plus simples puisque dépendant uniquement des aspects *matériels*. Le rapport ne définit pas de relations pour les entités Item.

Il est important de rappeler que le rapport FRBR ne donne pas de liste exhaustive pour les relations ou de règles *normatives* pour les utiliser. Il appartient au bibliothécaire de prendre la décision au cas par cas, le rapport fournissant les grandes lignes guidant le choix, et laissant

| Type de relation | Manifestation |
|---|---|
| Ensemble/partie<br>a une partie − ><br>< − est une partie de | volume d'une manifestation en plusieurs volumes<br>bande son d'un film sur un support distinct<br>bande son d'un film intégrée dans le film |

TABLE 3.3 – Relations ensemble/partie entre une manifestation et une manifestation

ainsi une place importante au facteur humain dans la classification et la création des notices bibliographiques.

Il est important aussi de noter que ces relations ensemble/partie n'impliquent pas que cet ensemble est *uniquement* composé de parties distinctes, mais que l'ensemble peut être du type `ensemble = texte + {parties}`, et ceci aux divers niveaux Work, Expression et Manifestation. Considérer un document comme composite ne se réduit pas à une juxtaposition de documents au sein d'une même page. Il existe des relations sémantiques spécifiques entre les parties d'un document composite. Il existe de plus des notions de document englobant qui est le maître d'œuvre de ces relations sémantiques. C'est en général par ce document englobant que l'on nomme le document composite et que l'on y accède.

De plus, l'existence de relations ensemble/partie entre entités permet de voir, pour un document donné, dans quels documents il figure comme composant. Il sera possible, pas exemple pour une image, de savoir dans quels documents elle apparaît comme illustration.

Nous venons de voir que les FRBR posent les bases permettant de modéliser les documents numériques composites, en établissant des relations souples entre divers composants d'un document. Dans la partie suivante, nous regardons comment le Document Object Model va, lui aussi, permettre de modéliser les documents composites, à un niveau différent de celui des FRBR, mais utilisant la modélisation des pages Web.

### 3.1.3 Document Object Model

Le Document Object Model (DOM) [LE HORS *et al.* 04] est une API (*Application Programming Interface*) pour les documents HTML valides et les documents XML bien formés. Le DOM définit la structure logique des documents, les façons d'accéder à cette structure et de la manipuler. La notion de document est d'ailleurs vue au sens large du terme puisque le DOM est utilisé aussi bien pour des documents au sens premier du terme que pour des données XML « brutes ».

Un des objectifs de la spécification du W3C est de fournir une interface de programmation utilisée dans un large panel de langages et d'applications. L'implémentation du DOM la plus utilisée dans le domaine du Web étant certainement celle pour ECMAScript [ECM 11] (plus

communément appelé Javascript), il existe de nombreuses autres implémentations de cette API (Java, PHP, etc.).

Le Document Object Model permet à un programme de construire des documents, de naviguer dans le document et d'ajouter, modifier ou supprimer quasiment toute information de ce document.

Le Document Object Model permet la réalisation concrète du document et ouvre une interface de programmation qui permet de le modifier. Notamment par l'utilisation de programmes qui peuvent insérer des données provenant de services web (c.f. section 1.3). Ces documents issus de services web font partie des composants indépendants tels qu'ils sont présentés dans la section précédente.

Nous utiliserons le DOM comme outils technique pour la représentation interne du modèle de document en arbre de Sydonie présenté partieII.

## 3.2 Documents multilingues

### 3.2.1 Introduction

De la même façon que dans la section 3.1, posons-nous la question de ce qu'est un document multilingue et quel sens nous mettons dans ce terme pour la suite de nos travaux.

Nous traiterons dans cette partie deux aspects du multilinguisme. Tout d'abord, nous nous intéressons aux documents dont plusieurs versions linguistiques existent. Ces documents peuvent être de natures diverses, aussi bien textuelle que visuelle ou sonore. Nous analyserons comment la plupart des systèmes de gestion de contenu prennent en charge le multilinguisme et les leçons que nous en tirons. Puis nous regarderons comment les FRBR présentés au chapitre 2 modélisent l'existence d'un document en plusieurs versions de langues. Nous ne parlerons pas des documents qui *mélangent* plusieurs langues dans leur contenu. Pour de tels documents, nous considérerons que le document a une langue « maître » et nous nous bornerons à préciser la langue des diverses parties. De plus, nous ne cherchons pas à effectuer des travaux d'alignement, sans exclure cependant cet axe pour des perspectives ultérieures.

Enfin, nous présenterons la distinction entre *internationalization* et *localization* et présenterons des aspects de la *localization* qui doivent être gérés par Sydonie. Nous terminerons par présenter plusieurs méthodes qui permettent le stockage et l'échange de données de traduction pour divers affichages dans une application web.

### 3.2.2 Multilinguisme et CMS

Nous nous attarderons plus en détails sur les systèmes de gestion de contenu (ou Content Management System, CMS) au chapitre 4. Nous souhaitons ici nous intéresser à la façon dont

les CMS gèrent le multilinguisme pour les documents et leurs composants.

Wordpress[23], un des logiciels de blog les plus utilisés (plus de 60 millions de blogs), ne prend pas en charge le multilinguisme par défaut. Pour cela, un plugin doit être utilisé (nous avons testé WPML[24]). Un billet de blog peut alors être traduit et le blog affiche des liens vers les diverses versions linguistiques disponibles. Une langue par défaut est définie et un billet doit d'abord exister dans cette langue avant de pouvoir saisir une autre version. Le blog dispose d'une bibliothèque de médias (images, vidéos) dont les éléments sont décrits par un titre, une légende, une description, un titre alternatif (pour la balise `alt` des images). Cependant, ces informations sont saisies uniquement dans la langue par défaut du blog. L'insertion d'une image ou d'une vidéo dans un billet de blog reprend ces informations pour l'affichage en les copiant dans le code HTML qui est directement inséré dans le billet. Il en résulte plusieurs problèmes :

- les informations insérées sont alors dans la langue par défaut, quelle que soit la traduction en cours ;
- les informations concernant le média sont dupliquées. Si par exemple la description de l'image change dans la bibliothèque de médias, alors la description présente dans le billet ne sera pas cohérente avec la description présente dans la bibliothèque ;
- il n'existe aucun moyen de savoir dans quels billets est inséré un élément donné de la bibliothèque de média.

Le premier problème est directement lié au multilinguisme du site, les deux suivants sont liés à la gestion des documents composites de Wordpress. Nous reviendrons sur ces deux points dans le chapitre 5 lorsque nous présenterons le modèle de document de Sydonie.

Pour les affichages des billets de blog, le système n'affiche que ceux publiés dans la langue d'interface choisie. En d'autres termes, lorsqu'un utilisateur consulte le blog dans une langue autre que celle par défaut, il ne va pas voir tous les billets du blogs puisque ceux qui ne sont pas traduits dans la langue choisie ne seront pas affichés. Le système fait le choix pour l'utilisateur, alors que celui-ci devrait avoir, au minimum, l'information de la présence de billets non traduits et pouvoir avoir accès à la version originale (ou dans une autre langue).

Dans le cas de Drupal[25], le logiciel libre le plus utilisé dans le domaine des CMS, la version 7 prend en charge le multilinguisme façon native. Cependant des plugins sont nécessaires pour le rendre ergonomique. Drupal gère le contenu avec des nœuds (`node` dans le vocabulaire de Drupal), et chaque version linguistique d'un contenu est un nœud distinct. Un plugin est nécessaire pour regrouper les versions linguistiques d'une même contenu intellectuel.

Pour une installation de base de Drupal, les images insérées dans du contenu sont simplement des fichiers agrémentés de quelques informations complémentaires (titre, texte alternatif par

23. http://wordpress.org/
24. http://wpml.org/
25. http://drupal.org/

exemple).

Le support multilingue pour les CMS est souvent développé comme un ajout à des systèmes existants. Les relations entre un document et ses versions linguistiques ne sont pas intégrées dans le cœur du système, rendant ainsi parfois difficiles leur gestion. Enfin, les composants d'un document sont en général considérés comme des fichiers et non comme des documents à part entière. C'est ce qui entraîne des pertes d'informations lors de l'insertion de ces composants dans diverses langues.

Le modèle que nous proposerons au chapitre 5 pallie aux défauts observés dans le domaine du multilinguisme. Il s'inspire du modèle proposé par les FRBR, que nous présentons dans la section suivante.

### 3.2.3 FRBR et multilinguisme

Nous avons introduit les travaux de l'IFLA sur les Functionnal Requirements for Bibliographic Records dans le chapitre 2 et vu comment le modèle permet de définir des relations entre entités du groupe 1. Rappelons que les entités Work et Expression expriment le contenu intellectuel de l'œuvre.

L'entité Work représente une création intellectuelle ou artistique, et se réalise en une ou plusieurs Expressions. Les FRBR définissent des relations entre Expressions de la même œuvre (c'est-à-dire dépendant de la même entité Work). Ces relations sont utilisées lorsqu'une Expression a été obtenue à partir d'une autre. Les Expressions obtenues sont autonomes, c'est-à-dire qu'il n'est nul besoin d'avoir recours à l'Expression antérieure pour la comprendre ou l'utiliser. Le tableau 3.4 reprend les relations entre Expression de la même œuvre définies par le rapport FRBR.

Il existe plusieurs moyens d'obtenir une nouvelle Expression d'une œuvre. En particulier, les différents états d'un texte sont réputés n'être que des Expressions d'une même œuvre. Par exemple, une réédition d'un ouvrage ou une édition révisée constitue une nouvelle Expression. De même pour les versions abrégées ou augmentées d'un texte existant. Les traductions d'une langue dans une autre, les transcriptions et arrangements musicaux, les versions doublées ou sous-titrées d'un film sont également réputées n'être que des Expressions différentes de la même œuvre originale. Comme nous l'avions présenté au chapitre 2, le passage d'une Expression à une autre requiert un travail intellectuel (traduction, résumé, etc.) alors que le passage d'une Manifestation à une autre (par exemple changement de format) peut être automatisé. Dans le cadre d'un système informatique, ceci soulève un problème dans le cas de la traduction automatique. Est-ce bien dans ce cas une nouvelle Expression ? On peut considérer que le travail intellectuel a été inscrit dans le programme de traduction, mais cela montre que des processus de validation

| Type de relation | Expression autonome |
|---|---|
| Abrégé<br>a un abrégé − ><br>< − est un abrégé de | Version abrégée<br>Version condensée<br>Version expurgée |
| Révision<br>a une révision − ><br>< − est une révision de | Édition révisée<br>Édition augmentée<br>État (image fixe) |
| Traduction<br>a une traduction − ><br>< − est une traduction de | Traduction littérale<br>Transcription (musique) |
| Arrangement (musique)<br>a un arrangement − ><br>< − est un arrangement de | Arrangement (musique) |

TABLE 3.4 – Relations entre une expression et une expression

doivent être pensés et mis en place au sein du système de gestion des documents.

En utilisant ce modèle conceptuel, chaque version linguistique d'une œuvre est donc une nouvelle Expression d'une même entité Work. On obtient ainsi une représentation sous forme arborescente, avec des relations entre les nœuds Expression. Notons que parmi toutes les expressions représentant les versions linguistiques, l'une d'entre-elles est l'expression originale, ou l'expression de référence. Notons de plus que le rapport FRBR ne précise pas comment exprimer cette notion de *référence*. La figure 3.1 donne un exemple pour une œuvre textuelle ayant deux traductions. Cette représentation peut aussi être utilisée pour des documents non textuels, par exemple des images comme le montre l'exemple de la figure 3.2, ou des œuvres de type audio ou vidéo (par exemple un film aurait une Expression pour la version originale en français et une autre expression pour la version sous-titrée en anglais).

FRBR définit aussi des relations entre expressions d'œuvres différentes, par exemple *est un résumé de*, *est un complément de* ou *est une suite de* entre autres. En effet dans ces cas là, la création intellectuelle n'est plus la même et donc il y a lieu de référer à un Work différent et de définir des relations entre les documents (au niveau Work ou Expression selon les cas).

Pour les documents composites et multilingues, si le document « englobant » a plusieurs versions linguistiques et qu'il possède une relation ensemble/partie avec un autre document « composant », il faudra alors déterminer quelle version linguistique de ce « composant » in-

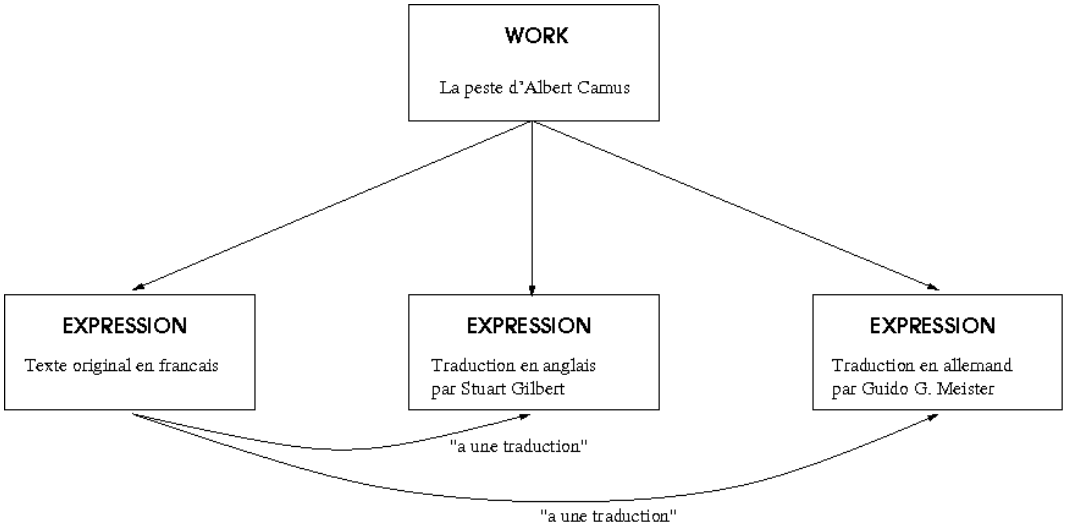



FIGURE 3.1 – Représentation arborescente des entités Work et Expression

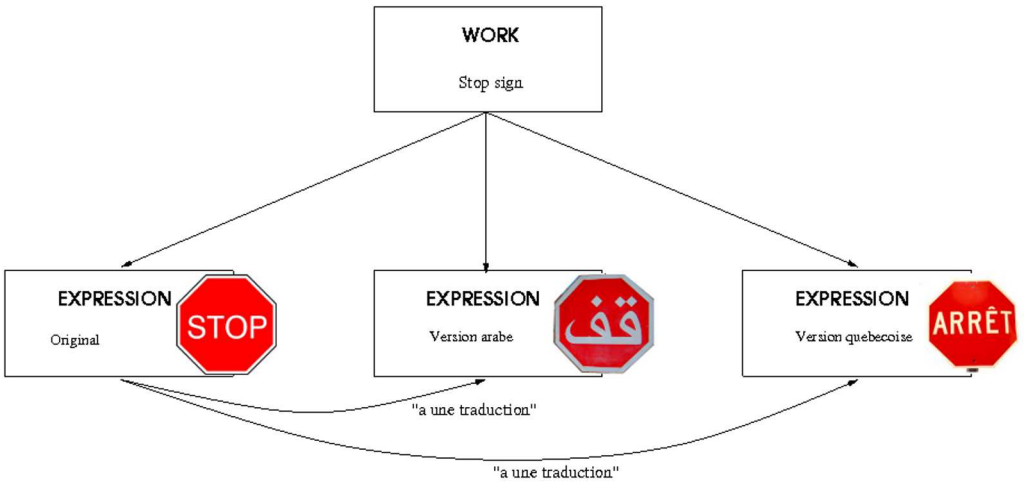



FIGURE 3.2 – Représentation arborescente des entités Work et Expression pour une image

clure dans la présentation du document « englobant ». Cette décision se fera en fonction de l'Expression du document « maître » qui doit être visualisée et des Expressions disponibles pour le document « composant ». Nous proposerons dans la partie II un moyen de recomposer les documents pour les proposer à un utilisateur.

Dans le domaine du multilinguisme comme dans celui des documents composites, les travaux présentés dans le rapport sur les FRBR nous offrent des éclairages qui posent les bases d'une modélisation pour les documents. Dans cette section, nous avons vu comment, grâce aux entités Work et Expression, les diverses versions linguistiques d'un document permettent d'être représentées et mises en relation. Enfin, l'entité Work offre un point d'entrée commun à toutes ces versions par exemple lors d'une requête documentaire. Cette conception permet simplement d'accéder à la liste des Expressions disponibles (i.e. diverses traductions entre autres) à partir d'une Expression donnée.

### 3.2.4 Multilinguisme, document et interface

Dans toute page ou application web, on doit distinguer la langue qui sert à la navigation et aux actions des utilisateurs, que nous appellerons langue d'interface, de la langue plus spécifique du document. Un site présenté en français par exemple peut contenir des documents dans une autre langue.

L'usager choisit en général sa langue d'interface, soit directement à l'aide d'un sélecteur, soit automatiquement grâce à la configuration des langues de son navigateur. Quand on doit présenter un document à l'utilisateur, il est préférable lui présenter une version dans la langue qu'il a choisi pour l'interface. Nous traiterons ici de la langue d'interface et de la gestion des labels, menus, boutons de navigation, etc.

#### Internationalization, localization

Lorsque l'on aborde le sujet du support de différentes langues pour un ensemble de programmes, les termes *internationalization* et *localization* apparaissent, et chacun de ces termes a une signification précise.

*Internationalization*, souvent raccourci en *i18n*[26], réfère au processus mis en place pour qu'un logiciel soit *capable* de fonctionner dans divers environnements linguistiques. Par exemple, préparer un site web pour l'i18n signifie que la conception et la réalisation des programmes gérant le site permettent de le faire fonctionner dans diverses langues sans programmation supplémentaire. Il sera cependant nécessaire d'y ajouter le processus de *localiza-*

26. *internationalization* est trop long à écrire, on simplifie en écrivant la première et la dernière lettre et le nombre de lettres en les deux (il y en a bien 18 entre le $i$ et le $n$).

*tion* pour les langues envisagées pour le site. Par exemple, le langage HTML prévoit pour toutes ses balises des attributs `lang` et `dir` pour préciser la langue et la direction de la langue (gauche/droite ou droite/gauche). La gestion de l'encodage des caractères (avec Unicode) est une des problématiques de base à gérer lors d'un processus i18n.

*Localization*, ou *l10n*[27], réfère au processus d'adaptation d'un système à une langue ou culture spécifique. l10n concerne donc l'ensemble des données qui seront fournies à un programme afin que celui-ci puisse fonctionner en ayant des entrées/sorties adaptées à diverses langues. Par exemple, l'affichage de dates ou nombres selon les habitudes d'une langue donnée (`12,345.67` en anglais, `12.345,67` en allemand et `12 345,67` en français) relève de la *localization*.

Les deux processus sont intrinsèquement liés puisque le processus de localisation ne peut avoir lieu si le système n'a pas été *internationalisé*. Les processus i18n et l10n interviennent à deux niveaux :

- la partie visible de l'application : l'interface utilisateur. Les affichages doivent alors être adaptés à la langue de l'interface choisie ;
- la partie non visible : traitement et stockage des données saisies dans divers formats et encodages.

Nous traiterons ici de la problématique de l'interface utilisateur, et nous présenterons dans la partie II les stratégies mises en place pour la gestion des documents dans diverses langues. Dans le cadre de l'interface utilisateur, les données d'une application web qui peuvent être adaptées pour l10n sont par exemple :

- les messages : informations affichées à l'utilisateur ou messages d'erreurs ;
- les labels, c'est-à-dire les textes des boutons, liens de navigation, etc ;
- les dates, nombres, monnaies ;
- numéros de téléphone, adresses ;

L'ensemble des informations spécifiques à une langue (ou un pays) qui permettent de traiter des dates, nombres, valeurs monétaires selon le bon format et vocabulaire est appelé *locale*. La plupart des langages de programmation offrent les possibilité de formatage de données selon une *locale* donnée, permettant ainsi un affichage des nombres, dates, monnaies, etc. dans diverses langues. Le processus de localisation peut dans ce cas être automatisé.

En revanche, en ce qui concerne l'affichage des messages et des labels définis plus haut, si le processus d'affichage lui-même sera aussi automatisé, il est nécessaire de fournir aux programmes les diverses versions linguistiques des messages et labels. Par exemple, les programmes doivent pouvoir utiliser les labels « Accueil » ou « Home », « Valider » ou « Submit » pour ne citer

27. C.f. *supra*

que deux exemples simples en français et anglais. Il faut déterminer comment enregistrer les diverses versions linguistiques des messages et labels, comment pouvoir les échanger et comment en ajouter.

De quels moyens dispose-t-on pour gérer les labels en diverses langues et quels sont leurs avantages ou inconvénients ? Nous présentons dans cette partie trois différents types de gestion des labels.

#### gettext

GNU `gettext` est un ensemble de programmes pour les développeurs d'applications et les traducteurs, leur offrant un ensemble d'outils intégrés à l'environnement. Ces outils incluent :

- un ensemble de conventions sur la façon dont les programmes doivent être écrits pour pouvoir utiliser des catalogues de messages ;
- une structure de répertoires et de nommage de fichiers pour l'organisation des catalogues ;
- un ensemble de programmes pour gérer les fichiers contenant les catalogues de messages.

L'objectif principal de GNU `gettext` est de minimiser l'impact de l10n sur les sources des programmes. GNU `gettext` est un des premiers outils permettant de gérer des catalogues de messages.

Les messages et leurs diverses versions linguistiques sont stockés dans des fichiers. Chaque fichier contient une traduction des messages dans une langue cible. GNU `gettext` gère les fichiers et l'arborescence des répertoires. Par exemple, une entrée de catalogue pourrait être formatée comme suit :

```
# une erreur inconnue
msgid "Unknown system error"
msgstr "Error desconegut del sistema"
```

Les avantage principaux de GNU `gettext` sont :

- intégration dans les systèmes GNU ;
- indépendance des fichiers selon la langue. Deux traducteurs peuvent donc travailler indépendamment à la traduction des messages dans deux langues distinctes ;
- format texte simple.

L'indépendance des fichiers selon la langue est aussi l'un de ses principaux défauts car il n'est pas possible d'avoir une vue globale des toutes les versions disponibles d'un message donné. Mais le plus problématique est l'absence d'un format d'échange des catalogues de messages qui existent au format texte uniquement.

Pour répondre à ces deux défauts, des groupes de travail ont réfléchi sur des structures XML pour l'échange et l'édition de catalogues de messages. Les deux parties suivantes présentent les résultats principaux.

```
<tu tuid="message0002"
    srclang="fr" >
   <prop type="Domain">Sports</prop>

   <!-- un exemple simple -->
   <tuv xml:lang="en-US">
      <seg>race car</seg>
   </tuv>
   <tuv xml:lang="fr">
      <seg>voiture de course</seg>
   </tuv>
   <tuv xml:lang="de">
      <seg>rennwagen</seg>
   </tuv>

   <!-- un exemple avec formatage du texte en HTML -->
   <tu tuid="message0182">
      <tuv xml:lang="en">
         <seg><bpt i="1">&lt;em&gt;</bpt>world
              <ept i="1">&lt;/em&gt;</ept> championship</seg>
      </tuv>
      <tuv xml:lang="fr">
         <seg>championnat du <bpt i="1">&lt;em&gt;</bpt>
              monde<ept i="1">&lt;/em&gt;</ept></seg>
      </tuv>
  </tu>

</tu>
```

FIGURE 3.3 – Exemple de contenu d'un fichier TMX

### Translation Memory eXchange

TMX (Translation Memory eXchange) [OSCAR 05] est une spécification XML créée par le groupe de travail OSCAR [28] de l'association LISA [29] (Localisation Industry Standards Association). L'objectif de la spécification est de définir un standard pour l'échange de données de traduction créées par les logiciels pour localisation ou d'aide à la traduction. Un certain nombre de logiciels pour les traducteurs acceptent le format TMX (en import et en export). Si les messages doivent être traduits en plusieurs langues, les données à traduire transiteront *via* divers traducteurs. L'utilisation de formats standards d'échange facilite donc le travail et la circulation des données entre les divers traducteurs.

Contrairement à `gettext` qui nécessite autant de fichiers que de langues cibles, TMX conserve toutes les versions linguistiques dans un même fichier. TMX est tout particulièrement destiné à fournir des traductions de phrases dans différentes langues. Un fichier TMX est donc un fichier XML avec pour élément racine `<tmx>`. Il est constitué d'un en-tête (élément `<head>`) et d'un corps (élément `<body>`).

L'en-tête permet d'inclure des métadonnées sur le fichier TMX (date de création et de modification, logiciel utilisé, *etc.*) ainsi que des notes et commentaires.

28. Open Standards for Container/Content Allowing Re-use
29. http://www.lisa.org/

Le corps du document TMX contient les unités de traduction (*translation unit*, éléments `<tu>`). Chaque élément `<tu>` doit contenir **au moins** une variante de traduction dans un élément `<tuv>`. Le texte de la traduction est lui-même dans un élément `<seg>` et peut contenir divers éléments pour le formatage du segment. L'exemple présenté en figure 3.3 montre les possibilités de TMX.

TMX produit des fichiers facilement et humainement lisibles, et donc éditables par un humain avec un minimum de connaissances en XML. Le fait de grouper toutes les traductions dans un même élément XML permet de facilement détecter la présence ou l'absence de traduction pour un message donné. En revanche le fait de grouper toutes les versions peut amener à des fichiers volumineux.

De notre point de vue, le point fort de TMX est son extrême simplicité pour des messages textuels. Le format XML assure une indépendance des données par rapport au type d'application créée et au langage de programmation utilisé.

Cependant, les travaux sur TMX sont au point mort depuis 2007, année de la publication d'un brouillon pour la spécification TMX 2.0 qui devait susciter les commentaires du public. De plus, l'association LISA s'est dissoute en février 2011, en laissant les travaux publiés dans le domaine public. Malgré le fait que TMX soit toujours utilisé par un certain nombre de logiciels, il semble qu'aucun groupe de travail n'ait repris les discussions autour de TMX à la date de rédaction de ce rapport.

Ceci renforce la position de XLIFF, un autre standard utilisé dans l'industrie de la localisation. Nous le présentons dans la partie suivante.

### XML Localisation Interchange File Format

XML Localisation Interchange File Format [Committee 08] (XLIFF), est un langage XML définissant un standard pour les échanges liés à la localisation. XLIFF est publié par l'association OASIS [30] (Organization for the Advancement of Structured Information Standards), consortium d'industriels et d'organismes publics dont l'objectif est de créer des standards ouverts pour la société de l'information.

Les objectifs de XLIFF sont proches de ceux de TMX dont il s'inspire en partie. XLIFF élargit cependant le champ d'application de TMX puisqu'il ne se concentre pas uniquement sur l'échange de données de traduction, mais sur le processus de traduction. Pour cela XLIFF définit des mécanismes permettant d'inclure des informations d'aide à la traduction (par exemple des suggestions de traduction ou des versions alternatives de traduction). Un fichier XLIFF contient donc une traduction *à l'état en cours* alors qu'un fichier TMX est destiné à contenir une traduction terminée.

30. http://www.oasis-open.org/

```
<xliff version='1.2'
       xmlns='urn:oasis:names:tc:xliff:document:1.2'>
<file original='hello.txt'
      source-language='en'
      target-language='fr'
      datatype='plaintext'>
   <body>
      <trans-unit id='hi'>
         <source>Hello world</source>
         <target>Bonjour le monde</target>
         <alt-trans>
            <target>Salut le monde</target>
         </alt-trans>
      </trans-unit>
   </body>
</file>
</xliff>
```

FIGURE 3.4 – Exemple de contenu d'un fichier XLIFF

Une différence principale avec TMX réside dans le fait qu'un fichier XLIFF ne contient que deux langues, une langue source et une langue traduite. De même que pour `gettxt`, il faut donc autant de fichiers XLIFF que de langues cibles pour un même ensemble de messages. Cela simplifie les échanges avec un traducteur pour une langue donnée mais rend plus complexe la gestion et le stockage de toutes les données lorsque le nombre de versions disponibles est grand. Cela n'est *à priori* pas un problème dans le domaine de la production logicielle puisqu'un utilisateur installe le produit dans une seule langue, mais complique la tâche d'un web designer qui doit permettre à un utilisateur de choisir la langue de l'interface du site à tout moment.

Un fichier XLIFF est un document XML de racine `<xliff>`. Le document est composé d'une ou plusieurs sections contenues dans un élément `<file>`. Un élément `<file>` correspond à une source dont les données à traduire sont extraites. XLIFF devant fonctionner avec deux langues, l'élément `<file>` spécifie en attributs la langue source et la langue cible.

Un élément `<file>` contient un en-tête optionnel (élément `<header>`) pour préciser des informations complémentaires sur la source de données. Le corps (élément `<body>`) contient la liste des messages à traduire. Notons que ces messages peuvent être groupés (par exemple la liste des items d'un menu) et que des informations sur le processus de traduction peuvent être insérés. Nous renvoyons à la lecture détaillée de la spécification pour ces détails.

L'élément `<body>` contient un ensemble de données à traduire. Chaque message correspond à un élément `<trans-unit>`, qui contient le message source dans l'élément `<source>` et la traduction dans l'élément `<target>`. L'élément `<alt-trans>` peut contenir des traductions alternatives.

Comme pour TMX et de façon similaire, des éléments sont définis pour exprimer la segmentation de texte. La figure 3.4 donne un exemple de code XLIFF simple.

**Un choix difficile**

Le format XLIFF présenté dans cette section permet de définir des données de traduction et de les échanger entre divers outils et traducteurs. Comme le format TMX, il permet de formaliser la segmentation des texte traduits. XLIFF exprime de plus diverses informations sur l'état de la traduction et permet d'inclure des suggestions alternatives, rendant le format XLIFF à la fois plus complet que TMX mais aussi globalement plus difficile à utiliser. De plus, un fichier XLIFF n'est destiné qu'à contenir deux langues, multipliant ainsi le nombre de fichiers à gérer si l'application doit être disponible en plusieurs langues. En centralisant les informations dans un seul fichier, TMX simplifie les traitements, en particulier si un label n'existe pas dans une langue donnée, le mécanisme de *fallback* est simple à réaliser.

Cependant, suite à la dissolution de LISA mentionnée auparavant, le standard XLIFF risque de devenir le seul standard mis à jour dans le domaine de la localisation.

Dans cette section, nous avons présenté deux aspects du multilinguisme. Le premier aspect concerne les diverses versions linguistiques d'un même document. Le rapport sur les FRBR, en définissant la relation `est une traduction` entre les Expressions d'une même entité Work, permet d'exprimer les liens entre ces diverses versions. Le deuxième aspect relève des techniques de localisation qui permettent de présenter un logiciel dans la langue choisie par l'utilisateur. En considérant l'interface du site web (interface de navigation entre autres) comme étant similaire à l'interface d'un logiciel, nous avons présenté des techniques et standards élaborés pour l'industrie de la localisation. Ces standards pourront être utilisés dans notre travail.

La section suivante, la dernière du thème Document, traite des documents et de leurs métadonnées.

## 3.3 Documents et métadonnées

Les deux parties précédentes ont abordé les questions du document numérique composite et multilingue et présenté des travaux qui serviront à la définition d'un modèle de document pour notre framework. Dans cette section, nous abordons la problématique des métadonnées et des documents.

### 3.3.1 Introduction

La définition, classique, d'une métadonnée est une « donnée sur une donnée ». Pour être plus précis, les métadonnées sont un ensemble structuré d'informations sur une ressource. Anne Gilliland [Gilliland 08] définit les métadonnées comme la « somme de ce qu'on peut dire sur un objet informationnel, sur plusieurs niveaux de précision ». L'objet concerné peut être

considéré comme une entité par un humain ou par une machine. L'introduction aux métadonnées publiée par la NISO [NISO 04] définit les métadonnées comme les « informations structurées qui décrivent, expliquent, localisent, ou en tout cas aident à retrouver, utiliser ou gérer une ressource, la ressource étant numérique ou non ».

Les métadonnées peuvent être regroupées en plusieurs catégories, notamment :

- les métadonnées descriptives servent à décrire et identifier la ressource. Par exemple pour un livre, le titre, l'auteur, l'éditeur d'un ouvrage mais aussi le format, les mentions, la collection... ce qu'on peut décrire à partir du livre en main ;
- les métadonnées administratives décrivent comment est gérée la ressource, donnent des informations techniques sur celle-ci, etc. Par exemple dans une bibliothèque, cela correspond au numéro d'inventaire, en prêt ou non, emplacement sur les étagères,... ;
- les métadonnées documentaires permettent de caractériser la ressource pour mieux la retrouver : indexation par mots-clés, classification, tags,... ;
- les métadonnées de structure servent à relier différentes parties de la ressource, au niveau logique (volumes, parties, chapitres par exemple) ou au niveau physique (liens entre fichiers qui composent la ressource).

À ce lot qui forme le cœur des métadonnées, on peut ajouter les métadonnées techniques (type de fichier, niveau de compression, etc.), d'usage (droits d'utilisation, traces des usages par exemple) et de préservation (informations sur les actions réalisées ou à faire pour la préservation de la ressource).

Historiquement, les métadonnées furent d'abord utilisées par les professionnels de l'information (bibliothèques, musées, *etc.*) avec plusieurs objectifs :

- description de ressources afin d'aider à leur découverte ;
- gestion de collections (classification par exemple) ;
- préservation de ces ressources.

Avec l'arrivée du web, les recherches ne se font plus principalement sur un (ou plusieurs) catalogue(s) géré(s) par des professionnels du domaine qui ont décrit et classé consciencieusement les ressources. On cherche à présent dans un ensemble de documents non organisés, et pour lesquels la production des métadonnées n'est plus l'apanage des professionnels. Les producteurs de documents utilisent les métadonnées non plus pour décrire les ressources mais pour en assurer la promotion.

Les métadonnées deviennent-elles alors une utopie comme le clamait de façon provocative Cory Doctorow en 2001 [Doctorow 01] ? L'argument principal de Doctorow pour ne pas faire confiance aux métadonnées se résume en « les gens mentent ». La balise `<META>` de HTML 2 apparue pour permettre d'inclure des métadonnées sur la page web parcourue par un moteur de recherche devient vite un moyen pour les webmestres peu scrupuleux de faire monter leur site en tête des listes de réponses de moteurs de recherche. Avec le « `META` tag spam », les moteurs

se doivent donc de détecter les mensonges dans les pages web.

En conséquence, les moteurs de recherche utilisent une indexation plein texte plutôt que les balises `<META>` et déploient des algorithmes secrets pour détecter les tricheurs. Avec l'indexation plein texte, ce sont les données textuelles elles-mêmes qui servent de clé d'accès pour la recherche. Les fonctions de rapprochement ou de discernement entre les documents qui étaient réalisés par les bibliothécaires sont maintenant confiées aux algorithmes des moteurs.

Les moteurs de recherche utilisent leurs capacités d'analyses des liens pour « créer » des métadonnées d'usage des pages web afin de détecter les pages que le web estime être en adéquation avec une recherche. Cette approche est renforcée avec la prise en compte par Google des avis des internautes exprimés par l'utilisation du bouton +1 (similaire au *I like* de Facebook) pour les pages référencées par le moteur [Singel 11a].

### 3.3.2 Comment stocker les métadonnées

Les métadonnées associées à un objet numérique peuvent être stockées dans le fichier lui-même ou dans une base de données (ou un fichier) externe. Les guides de bonnes pratiques [NISO 07] recommandent l'utilisation des deux solutions, en faisant attention à leur synchronisation.

L'inscription des métadonnées à l'extérieur de la ressource simplifie la recherche sur l'ensemble de la collection. Elle apporte souplesse avec par exemple la recherche multicritères, l'utilisation de vocabulaires contrôlés, etc. En évitant de parcourir le contenu de chaque ressource, elle offre de plus la rapidité nécessaire. Les catalogues de bibliothèques en sont l'exemple typique. Cependant, les métadonnées externes ne sont pas transmises lors de l'échange du document par téléchargement ou copie du fichier par exemple. Pour éviter ces pertes de données qui peuvent être dommageables, pour la gestion des droits associés par exemple, les informations sur le document peuvent être également inscrites à l'intérieur du fichier.

Les informations associées à un document peuvent parfois avoir une valeur intrinsèque au moins aussi importante que le contenu lui-même. Prenons deux exemples simples. Lors de l'ajout d'un fichier MP3 dans son lecteur favori, un utilisateur aime à voir s'afficher le nom de l'artiste, le titre du morceau, une image, et éventuellement d'autres informations. Sans ces données, le fichier `toto.mp3` lui sera de fort peu d'utilité parmi les centaines de fichiers MP3 présents sur son baladeur. Deuxième exemple [Salven 07], un photographe ayant fourni des images à un client en incluant des droits réservés sur l'utilisation de celles-ci, les responsables du journal *Miami Herald* ont pu s'apercevoir que la publication d'une image par le client ne pouvait se faire sans l'autorisation du photographe. Sans mention de droits *dans le fichier*, cette utilisation non permise n'aurait pu être détectée. La bibliothèque du Congrès, *Library of Congress*, et la *Stock Artists Alliance* ont lancé l'initiative *Photo Metadata Project* [SAA Photo Metadata Project]. L'objectif du projet est de sensibiliser les acteurs de l'édition d'images et en premier

lieu les photographes à l'inclusion de métadonnées dans toute ressource numérique à l'aide de standards existants.

### 3.3.3 Resource Description Framework

Il est nécessaire de définir un langage pour exprimer les différentes métadonnées des documents. Dans ce domaine, le langage RDF (Resource Description Framework) [Manola & Miller 04] est le plus utilisé.

Le concept fondamental de RDF est de pouvoir décomposer des informations sous leur forme la plus simple possible et de permettre l'utilisation d'une architecture distribuée [Klyne & Carroll 04]. La description d'une ressource est donc décomposée en triplets de la forme (sujet, prédicat, objet). Le sujet est la ressource qui est décrite, le prédicat est le terme qui définit la relation. L'objet est pris parmi un ensemble de descripteurs (liste de personnes, de documents, etc.). Chaque triplet étant indépendant, il est toujours possible d'ajouter des informations. RDF est donc particulièrement adapté pour l'expression de façon simple des métadonnées d'un document en listant ses propriétés sous la forme de triplets dont les prédicats et les objets seront dépendants du document décrit.

Par exemple, une image du Mémorial de Caen pourra avoir les propriétés suivantes :

- titre : Mémorial de Caen ;
- localisation : latitude : 49.198251, longitude : -0.382851 ;
- photographe : Jean-Marc Lecarpentier.

RDF représente ces informations sous la forme de quatre triplets que l'on peut écrire sous diverse formes, la notation N3 étant la plus « lisible » par un humain :

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

<http://monsite.com/lememorial.png>
  dc:title "Mémorial de Caen"
  dc:contributor <http://monsite.com/jml.rdf>
  geo:lat "49.198251"
  geo:long "-0.382851"
```

RDF peut être exprimé en différentes syntaxes, la version RDF/XML étant la plus utilisée en machine puisque basée sur un langage permettant l'échange de données. Souple et extensible, RDF est cependant d'une manipulation difficile, ce qui a conduit un certain nombre d'acteurs à écrire leurs métadonnées sous un autre format XML (par exemple Encoded Archival Description pour la description d'archives), voire de simples formats tabulés (par exemple les tags ID3 pour les fichiers de type mp3).

| Type de fichier | Métadonnées |
|---|---|
| Images | – données Exif (images Jpeg)<br>– données IPTC |
| Audio | – Format MP3 : Tags ID3<br>– Format WAV : Tags INFO |
| Vidéos | – Format AVI : Tags INFO<br>– Format MP4 : spécifique au format<br>– Ogg : conteneur libre |

TABLE 3.5 – Méthodes d'inclusion de métadonnées dans divers fichiers

Comme le montre l'exemple des métadonnées dans les fichiers mp3 incluses au moyen des tags ID3, l'inscription des métadonnées dans un document ne repose pas toujours sur RDF, ce qui ne simplifie pas le traitement des fichiers. La partie suivante présente XMP, un standard permettant d'insérer des métadonnées dans de nombreux types de fichiers.

### 3.3.4 Métadonnées dans les documents : XMP

La plupart des types de fichiers permettent l'inclusion de métadonnées, mais le procédé d'inclusion est souvent spécifique. Le tableau 3.5 donne un exemple de méthodes d'inclusion de métadonnées dans les fichiers images, son ou vidéos. Sur ces quelques exemples, on voit la multiplicité des méthodes. De plus, ces méthodes s'appuient souvent sur une liste finie de propriétés, ne reflétant pas la réalité et limitant les nouveaux usages. Pour pallier à ces limitations, Adobe a créé le standard XMP en collaboration avec l'IPTC [31].

Créé en 2005, XMP (eXtensible Metadata Platform) [ROSZKIEWICZ 08] [32] est un standard pour le traitement et le stockage de métadonnées relatives au contenu d'un fichier. XMP uniformise les méthodes de définition et de stockage des métadonnées dans divers types de fichiers.

En permettant d'encapsuler des données au format RDF, XMP rend possible l'ajout de métadonnées provenant de divers schémas. XMP fait cohabiter les autres modèles de métadonnées dans les champs d'en-tête des fichiers numériques.

XMP est en quelque sorte un méta schéma de métadonnées. Il permet de différencier les schémas de métadonnées, les formats d'écriture et les formats de stockage. Le principal avantage de XMP est l'homogénéité de la lecture et de l'écriture des métadonnées. XMP peut être intégré dans la plupart des types de fichiers images, pdf, audio ou vidéo. Le même méthode d'inscription des métadonnées pourra alors être utilisée pour tous ces types de fichiers.

31. International Press Telecommunications Council, http://www.iptc.org/
32. http://www.adobe.com/products/xmp/

XMP nous semble être un bon moyen de gérer l'inclusion des métadonnées dans les fichiers en offrant un mécanisme uniforme pour les types de fichiers courants. La problématique de l'utilisation et de l'inclusion des métadonnées dans les fichiers est alors gérée de façon centralisée et celle-ci se déplace sur le choix du schéma de métadonnées à utiliser.

### 3.3.5 Multiplicité des vocabulaires

La problématique des métadonnées se complique pour le développeur dès que se dessine la question du choix du schéma de métadonnées à utiliser. En effet, le développeur se trouve souvent confronté à l'impression que « plus on réfléchit aux métadonnées et plus cela devient compliqué »[33]. De fait la multiplicité des schémas de métadonnées rend le travail du développeur compliqué : quel schéma choisir pour l'application à développer ?

Dans le poster[34] illustrant l'univers des métadonnées [Riley & Becker 09] [Landesman 11], repris figure 3.5, l'auteure classe les standards les plus utilisés dans le domaine de l'héritage culturel en quatre catégories principales selon que la cible visée est le domaine, le rôle, l'usage ou le public :

- domaine, c'est-à-dire les types de données pour lesquelles le standard s'avère utile. Par exemple les données géolocalisées, les ensemble de données (*datasets*), les images, etc. ;
- rôle, c'est-à-dire le rôle joué par le standard pour la création et le stockage des métadonnées. Par exemple les standards de structure, de vocabulaire contrôlé, etc. ;
- usage, c'est-à-dire spécifier la dimension des métadonnées à exprimer. Par exemple les métadonnées de droits d'utilisation, les données techniques, etc.
- public, c'est-à-dire les communautés utilisatrices du standard. Les principales communautés sont les bibliothèques, les services d'archives, les musées et l'industrie de l'information.

L'ensemble montre la multiplicité des standards mais surtout des domaines d'application, des rôles, des usages, et des publics. Confronté à cette multitude, il est alors difficile pour un développeur de choisir tel ou tel standard. Cette multiplicité est pourtant nécessaire : on ne peut pas présumer des besoins de chaque communauté d'utilisateurs.

Le développement d'une application utilisant les métadonnées doit donc passer par une étape de choix des métadonnées à traiter, puis du choix de la modélisation de celles-ci, et enfin par le choix d'un ou plusieurs schémas de métadonnées. Le tableau 3.6 présente une sélection de schémas de métadonnées parmi les plus utilisés [Haslhofer & Klas 10]. Le schéma qui ressort en tête de liste, et qui est désormais inclus dans la plupart des applications, est le schéma Dublin Core, que nous présentons dans la partie suivante.

33. Citation de Andy Powell (Eduserv Foundation) : http://orweblog.oclc.org/archives/001717.html
34. Disponible à http://www.dlib.indiana.edu/~jenlrile/metadatamap/

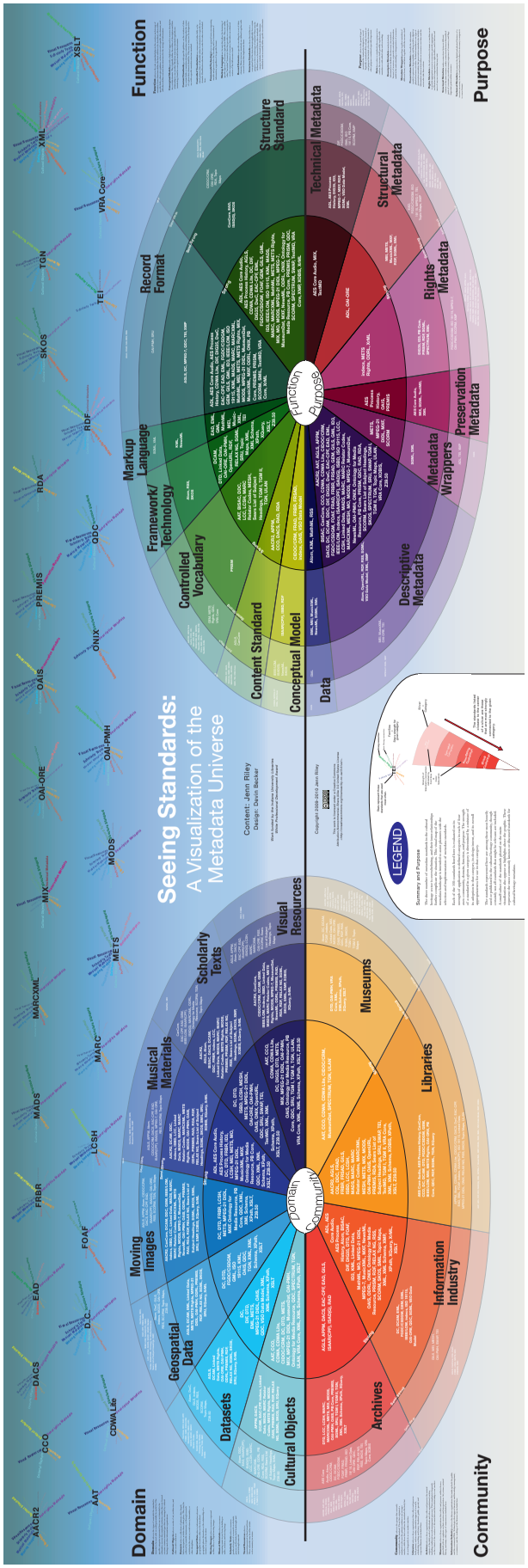



FIGURE 3.5 – L'univers des métadonnées

| Schéma | Domaine d'application | Objectifs | Dernière version |
|---|---|---|---|
| Dublin Core (DC) | Indépendant | Description générale de ressources | 2008 |
| Guidelines for Electronic Text Encoding and Interchange (TEI) | Sciences Humaines et Sociales | Représentation de textes numérisés | 2007 |
| Learning Objects and Metadata (LOM) | eLearning | Description de ressources pédagogiques | 2002 |
| Sharable Content Object Reference Model (SCORM) | eLearning | Description, agrégation et séquences d'objets pédagogiques | 2004 |
| Online Information Exchange (ONIX) | Commerce et publication | Fournir des informations sur les produits en vente en ligne | 2005 |
| MARC 21 DFormat for Bibliographic Data | Bibliothèques | Échange de données bibliographiques | 2006 |
| Metadata Object Description Schema (MODS) | Bibliothèques numériques | Sous-ensemble de MARC | 2006 |

TABLE 3.6 – Une sélection de schémas de métadonnées

### 3.3.6 Dublin Core

Le Dublin Core [DCMI 08] est un format descriptif à la fois simple et générique, comprenant 15 éléments différents, qui a été créé en 1995 à Dublin (Ohio) par l'OCLC [35] et le NCSA [36]. La création de ce format a donné naissance à l'association Dublin Core Metadata Initiative [DC 95].

D'après la présentation du Dublin Core de la BnF [37], l'objectif du Dublin Core est de fournir un socle commun d'éléments descriptifs pour améliorer le signalement et la recherche de ressources au-delà des diverses communautés et des nombreux formats descriptifs propres à chaque spécialité, tout en restant suffisamment structuré.

Le Dublin Core prévoit 15 éléments tous facultatifs et tous répétables, qui portent sur la description :

- du contenu : Title, Subject, Description, Source, Language, Relation, Coverage ;
- de la propriété intellectuelle : Creator, Contributor, Publisher, Rights ;
- de l'instanciation : Date, Type, Format, Identifier.

Le Dublin Core est indépendant des formats d'encodage et de stockage de l'information. La DCMI propose néanmoins des recommandations et bonnes pratiques pour permettre l'emploi

35. Online Computer Library Center, http://http://www.oclc.org
36. National Center for Supercomputing Applications, http://www.ncsa.illinois.edu/
37. Présentation du Dublin Core sur le site de la Bibliothèque Nationale de France (BnF) : http://www.bnf.fr/fr/professionnels/formats_catalogage/a.f_dublin_core.html

uniforme de Dublin Core dans tous les types d'usages.

En proposant un schéma relativement souple et généraliste, le Dublin Core est désormais utilisé par de nombreuses applications et standards. Cependant, cette généralité lui fait perdre en précision, les règles énonçant le contenu des quinze éléments étant parfois floues [Baker 00].

Au cœur des métadonnées que représente Dublin Core, on peut ajouter d'autres schémas en fonction du domaine d'application du projet. Cette multiplicité des schémas et complexité d'utilisation pour un développeur non averti est d'ailleurs le principal frein à l'utilisation efficace des métadonnées dans les projets web. Il est donc capital pour un développeur de se préoccuper d'abord des données de son modèle, et non pas des métadonnées qu'il devra traiter. Le développeur doit de plus avoir à sa disposition des outils qui lui rendent ces schémas de métadonnées le plus transparent possible, afin d'éviter de se perdre dans le dédale des schémas. Il pourra alors se concentrer sur les données manipulées par son application. Bien entendu, un développeur averti doit cependant pouvoir faire ce qu'il veut avec les métadonnées. Dans le cadre du projet de framework Sydonie, la difficulté sera donc de proposer des outils adaptés aux besoins des développeurs web qui leur offre une souplesse de travail et leur minimise les difficultés.

### 3.3.7 Un framework pour les métadonnées

Le framework que nous voulons développer doit donc imaginer des outils qui aident les développeurs web à travailler avec les métadonnées. Dans ce cadre, nous avons identifié un certain nombre de tâches relatives aux métadonnées qui sont présentes dans le développement d'applications web.

#### Extraction de métadonnées

Lors de l'import de fichier dans le système, les métadonnées éventuellement présentes dans le fichier doivent être extraites pour les présenter à l'utilisateur. Par exemple, lors de l'import d'une image, le système peut lire les métadonnées déjà présentes dans l'image afin de remplir automatiquement les champs de description de l'image avant toute intervention d'un utilisateur. Le framework doit proposer des routines permettant de travailler avec divers types de fichiers, en commençant par les types les plus utilisés (images, PDF par exemple), et traiter divers schémas (Exif, XMP, IPTC, etc) et divers vocabulaires, en particulier Dublin Core. Tout développeur doit de plus pouvoir ajouter des routines pour son application, lui permettant d'enrichir celles proposées par le framework.

#### Inclusion des métadonnées

Le système proposé doit permettre l'opération inverse, c'est-à-dire la possibilité d'inscrire les métadonnées dans les fichiers. Par exemple, lorsqu'un utilisateur dépose un fichier PDF sur le site et saisit des informations le concernant, ces informations doivent être inscrites dans le fichier, assurant ainsi la synchronisation des données recommandée par [NISO 07]. Là encore, ces opérations doivent paraître le plus transparentes possible au développeur mais aussi à l'utilisateur.

#### *Mappings* modèle de l'application/métadonnées

Le modèle d'une application ne doit pas être dépendant du schéma de métadonnées choisi. De plus si plusieurs schémas sont utilisés, il faudra gérer la redondance entre ceux-ci. Là encore, le framework doit proposer des moyens d'effectuer simplement la correspondance entre les données du modèle de l'application et les schémas de métadonnées. Par exemple, lorsqu'une photographie est associée à un objet *photographe* de l'application, la correspondance entre *photographe* et le champ `dc:contributor` doit être formulée de façon simple par le développeur, et le reste du travail, principalement l'écriture du RDF correspondant, lui sera alors transparent.

Ces mécanismes permettent au développeur de se concentrer sur le modèle de son application. En rendant le plus transparent possible les mécanismes d'import/export et de correspondance entre les données et métadonnées, la création d'applications qui utilisent les métadonnées en est facilitée. De plus, cette gestion devient elle aussi le plus transparent possible pour un utilisateur final, qui dans le cas général n'est pas un professionnel de l'information ou spécialiste des métadonnées. La gestion des métadonnées dans un système n'est donc pas seulement un problème de technique, mais aussi un problème de processus de gestion des documents et des interfaces mise en œuvre dans l'application web. Le développement de telles applications ne doit donc pas se perdre dans les considérations techniques ou les difficultés du sujet, mais doit se concentrer sur les processus de traitements des documents et leurs acteurs.

Nous présenterons au chapitre 6 les solutions apportées par le framework Sydonie pour la production et la gestion des métadonnées.

### 3.3.8 Le cas des métadonnées dans les pages web

L'intégration des métadonnées au moyen de XMP permet d'utiliser un processus commun à la plupart des formats de fichiers. Le cas des documents HTML est cependant différent. Les métadonnées peuvent y être insérées à deux niveaux :

– pour l'ensemble de la page web dans l'en-tête HTML ;

```
<html xmlns:og="http://ogp.me/ns#">
<head>
...
<meta property="og:audio" content="http://example.com/amazing.mp3" />
<meta property="og:audio:title" content="Amazing Song" />
<meta property="og:audio:artist" content="Amazing Band" />
<meta property="og:audio:album" content="Amazing Album" />
<meta property="og:audio:type" content="application/mp3" />
...
</head>
```

FIGURE 3.6 – Exemple de métadonnées OPG dans l'en-tête de page HTML

– dans les textes des pages pour préciser et contextualiser le sens de certains textes, au moyen des microformats, microdata ou de RDFa introduits à la section 1.4.

### Métadonnées dans l'en-tête

Les balises `<META>`, apparues dès 1995 avec HTML 2.0, permettent d'inclure des métadonnées diverses. Le développement des schémas de métadonnées tels que le Dublin Core ont mené à la création de mécanismes pour intégrer les métadonnées Dublin Core dans les en-têtes de pages web avec les balises `<META>` et `<LINK>` [JOHNSTON & POWELL 08]. L'utilisation des *namespace* XML permet aussi d'intégrer des métadonnées complémentaires à l'aide des balises `<META>`. C'est le choix de Facebook avec Open Graph Protocol [OPG 10] pour la description de ressources présentes dans les pages web, en particulier pour l'intégration des informations qui seront utilisées par le bouton « *I like* ». Les données peuvent ainsi être simplement ajoutées sur la page, comme illustré figure 3.6.

Cependant, les métadonnées spécifiées dans l'en-tête de la page web sont nécessairement globales pour l'ensemble de la page. Il n'y est donc pas possible d'associer des détails à chaque section de la page par exemple. Pour exprimer des informations plus spécifiques à diverses parties de la page web, les métadonnées doivent alors être exprimées au sein du contenu de la page.

### Métadonnées au sein du contenu

Comme nous l'avons présenté à la section 1.4, plusieurs moyens existent pour inclure les explications de texte exprimées par les métadonnées dans le contenu des pages web. En plus du choix déjà difficile des vocabulaires à utiliser, le développeur web se retrouve devant un autre choix à effectuer : doit-il utiliser les microformats, RDFa ou microdata ?

Ni les développements récents au sein du W3C, ni les choix des géants du web n'aident

les développeurs. En effet, l'annonce en juin 2011[38] de schema.org[39], une collaboration entre les trois principaux moteurs de recherche Google, Bing et Yahoo !, proposant un schéma de métadonnées exprimé dans les pages web *via* le format microdata relance le débat. Si nombre de développeurs utilisaient déjà RDFa, l'annonce de Schema.org repose la question de ce choix [Corlosquet 11].

Le débat est lui aussi relancé par le groupe TAG (Technical Architecture Group) du W3C qui demande aux deux groupes de travail sur les microdata et RDFa de s'accorder sur une syntaxe commune pour éviter la publication par le W3C de deux recommandations sur le même sujet[40]. Afin de travailler sur ce point, le W3C lance deux nouveaux groupes de travail [Herman 11]. Le premier groupe, *Web Schemas Task Force*[41] travaille sur la problématique des vocabulaires, et le second groupe, *HTML Data Task Force*[42] se concentre sur RDFa et microdata pour analyser comment les deux peuvent être combinés.

De notre point de vue, la majorité des développeurs web se ralliera à l'utilisation des microdata, ceci pour plusieurs raisons. En premier lieu, qui osera prendre le risque de ne pas utiliser les technologies recommandées par les trois moteurs de recherche les plus utilisés ? Même si ceux-ci continuent de parser les pages web qui contiennent des informations au format RDFa, le choix est clairement de mettre en avant les microdata. De plus, les développeurs web, et en particulier ceux qui n'ont pas une formation avancée en informatique, rechignent à apprendre de nouveaux schémas XML et en particulier la syntaxe compliquée de RDFa. Les microdata et les outils mis à leur disposition par l'initiative schema.org semblent plus accessibles pour un développeur qui écrit son code HTML « à la main ». Un certain nombre d'indicateurs semble confirmer cette montée en charge des microdata. Schema.org a adopté le standard rNews [IPTC 11b], un standard de l'IPTC pour inclure des métadonnées sur les brèves d'actualité, initialement proposé en RDFa, renforçant ainsi la position des deux acteurs. Le projet LRMI (Learning Resource Metadata Initiative)[43], une initiative commune à l'Association of Educational Publishers (AEP) et Creative Commons, travaille à la création d'un vocabulaire avec schema.org pour la description de ressources éducatives, à destination des enseignants et apprenants. Aneesh Chopra, United States Chief Technology Officer, travaille aussi avec schema.org sur la problématique de la transparence de l'information [Franzon 11]. D'autres vocabulaires, comme le populaire GoodRelations[44] ou *Product Type Ontology*[45] par exemples, ont d'ores et déjà ajouté la possibilité d'utiliser les microdata. Toute cette activité s'est réalisée moins de quatre mois après

38. http://googleblog.blogspot.com/2011/06/introducing-schemaorg-search-engines.html
39. http://schema.org
40. http://lists.w3.org/Archives/Public/public-html/2011Jun/0366.html
41. http://www.w3.org/2001/sw/interest/webschema.html
42. http://www.w3.org/wiki/Html-data-tf
43. http://lrmi.net/
44. http://www.heppnetz.de/projects/goodrelations/
45. http://www.productontology.org

le lancement de schema.org, montrant le poids des trois géants Google, Yahoo et Bing dans la direction suivie par le web. L'inconvénient le plus reproché à Microdata est son incapacité, dans certains cas, à utiliser plusieurs vocabulaires dans une même page [Tennison 11]. La recommandation n'étant pas encore publiée, des évolutions sont cependant toujours possibles.

Nous entrons donc dans une période d'incertitude sur le devenir des microdata et de RDFa 1.1. Les groupes de travail réussiront-ils à s'accorder ou les deux formats vont-ils coexister jusqu'à ce que l'un deviennent le standard *de facto* ou qu'ils soient remplacés par un nouveau format ? Il est encore trop tôt pour répondre à cette question. Cependant, quel que soit le langage utilisé, le principe du marquage du contenu reste commun. Dans ce cadre, une difficulté complémentaire est liée au mode de création du contenu des pages web.

#### Annotations au sein du texte et IHM

L'inclusion de métadonnées dans le contenu des pages web, que ce soit en microdata ou en RDFa, pose des problèmes d'ergonomie et d'IHM.

Grâce à la multiplication des systèmes de gestion de contenu, les textes des pages web sont le plus souvent créés *via* un formulaire où le rédacteur saisit son texte, choisit ses images, *etc.* Le rédacteur peut de plus remplir des champs qui donnent des informations complémentaires sur le contenu saisi (tags, classification, etc.). Ces informations complémentaires seront transformées par le système de gestion de contenu en métadonnées dans la page web, au niveau global dans l'en-tête HTML. Ces données complémentaires peuvent aussi être exprimées en RDFa dans les éléments les affichant. C'est le choix de Drupal pour intégrer des données RDFa [Corlosquet *et al.* 09] dans ses pages.

Cependant, le corps du texte saisi par l'utilisateur ne contiendra *à priori* aucune annotation, à moins que l'utilisateur ne soit suffisamment averti pour les y inclure lui-même.

Pour permettre l'inclusion de métadonnées dans le texte produit par un utilisateur, des méthodes automatiques d'extraction d'information peuvent être utilisées pour identifier des entités. Le résultat est obtenu sous la forme de texte complété d'annotations au format RDFa. OpenCalais [Butuc 09], un service fourni par Reuters, procède ainsi. Il permet de repérer dans un texte d'actualité les mentions de lieux, de personnes et d'évènement qui sont encodées en RDFa. Epiphany [Adrian *et al.* 10] est un projet dont le but est d'extraire des informations des pages web en spécifiant les domaines à traiter, ce qui évite les restrictions d'OpenCalais au domaine des brèves d'actualité.

La problématique de l'enrichissement du texte saisi par un utilisateur ne s'arrête pas à l'extraction des termes et l'inclusion des informations sémantiques correspondants. Il faut ajouter à cela, une fois de plus, le processus éditorial qui doit l'accompagner. En effet, une fois les informations ajoutées au texte, il est indispensable que ces informations puissent être revues et validées

par l'utilisateur. Par exemple, dans le texte « Paris Hilton était hier à Paris », OpenCalais repère deux fois l'occurrence de la personne Paris Hilton mais ne trouve pas la ville de Paris. Le « tout automatique » n'est donc pas suffisamment fiable pour remplacer complètement l'humain. Il est indispensable qu'un utilisateur puisse modifier les termes trouvés, comme le montre l'exemple ci-dessus, mais aussi ajouter et supprimer des termes. Des systèmes d'aides doivent cependant lui être proposées, en utilisant des vocabulaires contrôlés ou d'autres mécanismes. On retrouve ici les problématiques d'interface homme-machine : imaginer de nouvelles formes d'interaction pour la gestion de ces informations complémentaires, et comment « mixer » les aspects automatiques et interactions avec l'humain dans les systèmes de gestion de documents de demain.

### 3.3.9 Qualité des métadonnées

Les métadonnées associées aux documents sont importantes pour le référencement de ces documents dans le système d'information ou sur le web de façon plus large. Cependant, afin d'être réellement utiles, les métadonnées doivent atteindre un certain niveau de qualité. Dans leur analyse, Bruce et Hillman [Bruce & Hillmann 04] notent que la multiplication des schémas de métadonnées est contre-balancée par le manque d'expérience des développeurs puis des utilisateurs des systèmes d'informations utilisant ces schémas. Ainsi, si la multiplication des métadonnées elle-même peut être un point positif, la qualité des métadonnées n'est pas toujours à la hauteur des espoirs suscités.

Mesurer la qualité de métadonnées est nécessairement abstrait. Bruce et Hillman proposent sept paramètres :

- complétude : le schéma de métadonnées choisi doit couvrir les besoins le mieux possible (en restant dans la limite des possibilités techniques et économiques). En sens inverse, le schéma de métadonnées choisi doit être utilisé le plus complètement possible. En effet, il est inutile de pouvoir enregistrer une multitude de métadonnées si seulement quelques items de la collection sont annotés ;
- précision : les informations doivent être précises. Au minimum l'information doit être correcte et factuelle. Les termes utilisés doivent être communément reconnus (sigles, abréviations, *etc*) ;
- provenance : la connaissance de l'organisme qui a fournit les métadonnées donne une idée de la confiance que l'on peut leur accorder ;
- respect des attentes : l'utilisation de standards pour les métadonnées et de profils d'application doit être en concordance avec les attentes de sa communauté d'utilisateurs. On peut imaginer d'implémenter tout ce que l'on veut, mais il est préférable d'implémenter un système qui correspond aux attentes et besoins des utilisateurs ;

- cohérence : l'utilisation de standards et de profils d'application permet de s'assurer de la cohérence du schéma utilisé ;
- validité dans le temps : les informations doivent rester valides au fil du temps, ce qui représente un des aspects les plus difficiles à vérifier. La maintenance des données réparties est encore plus complexe que celles contenues dans un catalogue ;
- accessibilité : les informations doivent être accessibles pour les humains et les machines. Cet aspect est relativement facile à mettre en œuvre.

Plus que des règles de développement, les paramètres de qualité fournissent des règles de bonne conduite qui nous serviront dans les développements de système de gestion des métadonnées. Nous présentons au chapitre 6 les solutions mises en œuvre dans Sydonie pour parvenir à un niveau acceptable de qualité des métadonnées.

### 3.3.10 FRBR et métadonnées

Le modèle défini par les FRBR pour les documents avec les entités du groupe 1 Work, Expression, Manifestation et Item expriment les divers niveaux d'informations pour un document. En spécifiant des attributs pour chaque niveau d'entité, les FRBR répartissent les métadonnées d'un ensemble de documents sur les divers nœuds de l'arbre le représentant.

L'utilisation des entités Work et Expression pour représenter le contenu intellectuel et artistique permet d'associer les métadonnées au plus haut niveau possible et évite ainsi de dupliquer les informations aux niveaux d'entité plus bas.

Nous verrons au chapitre 6 comment le modèle des FRBR est utilisé dans Sydonie pour la gestion des métadonnées.

## 3.4 Synthèse

L'approche traditionnelle distingue les documents et les métadonnées. Nous avons montré comment la réalité est plus complexe, documents et métadonnées étant intriquement liés. De la même façon, les différents composants d'un document ou les versions linguistiques sont traditionnellement considérés indépendants. Le modèle des FRBR recentre la notion de document autour de l'ensemble des ces informations organisées sous la forme d'un arbre. Ces questions recoupent la complexité en trois parties définies par le Pédauque [Pédauque 06] où la forme est caractérisée par les métadonnées de structuration et de présentation, le signe par les métadonnées incluses au sein du texte (RDFa, etc.) et le medium correspond à l'architecture des diverses Expressions dans des catalogues pour diversifier leur usage.

La réalisation de Sydonie a essayé de prendre en compte cette complexité. Nous présentons nos résultats dans la partie II.

# Chapitre 4

# Web Engineering

**Sommaire**



Le web recouvre une palette de métiers désormais vaste : du développeur au graphiste pour la production, mais aussi du rédacteur au webmestre pour le fonctionnement au quotidien, les acteurs sont nombreux. Dans ce chapitre, nous nous intéressons à l'ingénierie du Web, ses acteurs et les outils disponibles. Nous abordons les problématiques d'authentification et des politiques de permissions d'accès pour terminer.

## 4.1 Ingénierie du Web

Les sites web : qui, quoi et comment ? Peu d'études parlent de la population des « web designers » ou développeurs web et de leurs méthodes de travail. Celle-ci est constituée de personnes de profils très divers, et non uniquement d'informaticiens. Cette partie tente de répondre à ces questions en s'intéressant aux travaux académiques et aux travaux de l'industrie du web.

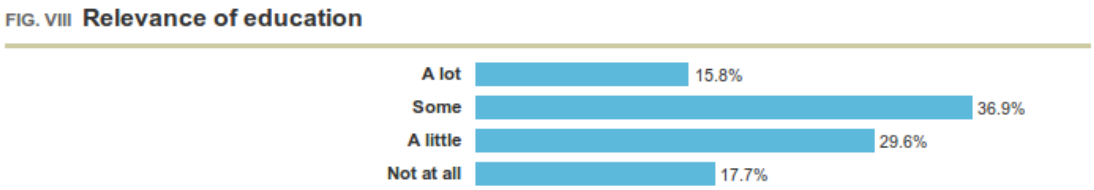



FIGURE 4.1 – Adéquation formation/travail, issu de l'étude du site *A List Apart*

### 4.1.1 Acteurs de l'Ingénierie du Web

Qui crée des sites web ? C'est en se posant cette question en 2007 que les responsables du site *A list apart*[46] se sont aperçus du peu d'études sur la population des développeurs web.

La première étude date de 1998 et fut réalisée par Pawan Vora [VORA 98]. En l'absence totale d'informations sur les personnes qui font des sites et leurs pratiques de développement, l'objectif de l'étude était d'obtenir un profil des développeurs, de leurs méthodes de travail et de leurs besoins. Il ressort de cette étude une population aux profils très divers, avec les graphistes et ergonomes majoritaires devant les programmeurs. D'autres études [ROSSON *et al.* 05A] [ROSSON *et al.* 05B] [CONNELL 08] confirment cette variété de profils, avec environ 50% seulement de profils informaticiens. Le sondage réalisé annuellement par le site *A list apart* [STEVENS 10] est cependant peut-être le plus fiable puisque réalisé sur un échantillon beaucoup plus large[47]. La réponse à la question de l'adéquation de leur formation avec leur travail actuel, illustrée figure 4.1, montre que les informaticiens sont loin d'être majoritaires, mais peut-être aussi que les formations informatiques ne sont pas adaptées au développement web.

La diversité des profils montre bien la distinction entre développement informatique et développement web. Historiquement, le web a été créé comme média de diffusion de documents et a donc attiré à lui nombre de personnes liées aux documents plus qu'à l'informatique (documentalistes, graphistes, etc.). L'évolution vers un web d'applications, présenté à la section 1.2, qui rend le web beaucoup plus techno-centré, ne donne cependant pas nécessairement la part belle aux informaticiens comme le montre l'étude du site *A list apart*. Malgré cette évolution, beaucoup considèrent le développement web comme un problème lié à la rédaction et aux documents plutôt que du développement d'application [GINIGE & MURUGESAN 01]. Les professionnels du web constituent une population aux profils très divers, qui maintient à jour ses compétences grâce aux ressources disponibles sur le web lui-même (FAQs, tutoriels, documentations, etc.). Toutes les études mentionnées ci-dessus soulignent ce mode de fonctionnement en auto-apprentissage pour la « mise à jour » des connaissances.

46. http://www.alistapart.com

47. On connaît la marge d'erreur des sondages sur le web auxquelles ne répondent que les personnes motivées. Cependant, cette étude est axée sur les professionnels, à l'image du site lui-même. Plus de 16000 personnes, certes anglophones, ont répondu à cette enquête.

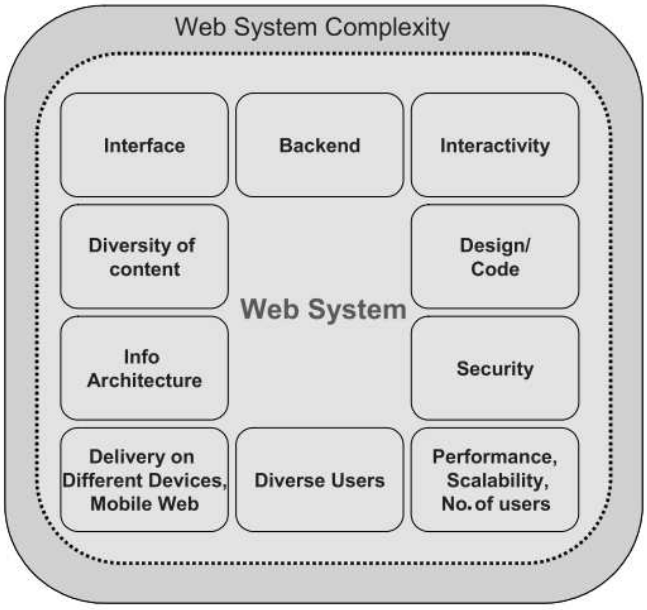



FIGURE 4.2 – Complexité des systèmes Web illustrée par Murugesan

La multiplicité des acteurs et des technologies mises en jeu (techniques plus réflexion sur la place de celles-ci) a de plus fait émerger la nouvelle discipline qu'est l'Ingénierie du web. Dans son article [ROSSON *et al.* 05B], Rosson insiste sur l'absence de corrélation entre le profil du développeur et les difficultés rencontrée. Elle montre aussi que des fonctionnalités en apparence simples, comme par exemple un système d'inscription à un site, se révèlent complexes à mettre en œuvre. Dans leur introduction de la discipline « web engineering », Ginige et Murugesan [GINIGE & MURUGESAN 01] présentent le développement web comme multidisciplinaire, un mélange entre la publication « print » et le développement logiciel, entre marketing et informatique, entre communication interne et relations extérieures, entre art et technologie. Même si certains contestent l'apparition d'une nouvelle discipline [KAUTZ & NØ RBJERG 03] et estiment que le développement web n'est qu'une évolution dans le développement des systèmes d'information, de nouvelles problématiques sont apparues. Mendes [MENDES *et al.* 06] liste les différences entre le développement logiciel classique et le développement web, classés par catégories. Il ressort, entre autres, que les applications web sont développées pour une audience large et parfois peu ou pas définie au départ, avec des architectures plus complexes, comprenant plus d'intervenants et faisant appel à des spécialités les plus diverses. De plus, les applications web doivent être opérationnelles à tout moment. Murugesan [MURUGESAN 08] illustre cette complexité par le schéma repris figure 4.2.

Au-delà des informations sur la population des développeurs web, les diverses études mentionnées ont permis de faire émerger un certain nombre de besoins récurrents.

### 4.1.2 Développement web, quels besoins ?

Un certain nombre de fonctionnalités se retrouvent de façon générale dans la plupart des applications web : authentification, gestion des droits d'accès, etc. En analysant les réponses des développeurs web, Rosson [Rosson *et al.* 05b] et Rode [Rode & Rosson 06] établissent une liste des besoins des développeurs web, dont les plus demandés sont :

- persistance des données : généralement en assurant la gestion des accès à une base de données ;
- gestion de session : permettre de conserver des informations de page en page ;
- gestion des saisies : formulaires de saisie et gestion de la validité des données ;
- gestion des droits d'accès : gérer quels utilisateurs ont accès à quelles ressources. On peut y inclure la procédure de création de compte ;

Il est intéressant de noter que l'étude de Rode [Rode & Rosson 06] montre que le tiers des besoins des utilisateurs pourraient être couverts par un outil offrant les fonctionnalités de saisie et de stockage de données. Un autre 40% des besoins seraient couverts par 5 types d'applications génériques :

- réservation de ressources ;
- panier électronique et paiement ;
- forum et tableau d'affichage ;
- gestion de contenu ;
- calendrier.

Le reste des besoins réside dans des applications plus avancées ou spécifiques.

Nombre d'outils permettent aujourd'hui de réaliser une application du type mentionné ci-dessus : les systèmes de gestion de contenu, généraliste ou spécialisé, s'acquittent très bien de ces tâches. Les 25% d'applications ne rentrant pas dans les catégories classiques sont alors ceux qui posent problème pour les outils existants.

### 4.1.3 CMS *vs.* framework

La grande majorité des sites internet sont aujourd'hui gérés à l'aide de Systèmes de Gestion de Contenu ou CMS (Content Management System). Parmi les plus connus, on peut citer Drupal [48], Spip [49] ou Joomla [50]. À côté de ces CMS on trouve des frameworks de développement spécialisés pour le web comme Ruby On Rails [51] ou Zend Framework [52]. Le titre de cette section est volontairement réducteur et semble opposer CMS et frameworks. Dans cette section, nous

48. http://drupal.org/
49. http://www.spip.net/
50. http://www.joomla.fr/
51. http://rubyonrails.org/
52. http://framework.zend.com/

discutons des solutions qui s'offrent au développeur web pour réaliser des sites et applications web.

La première méthode, utilisée par nombre d'agences de développement web, est de développer l'ensemble de l'application web directement à partir d'un langage de programmation. Dans le domaine du web, le langage le plus utilisé est PHP, désormais enseigné dans la plupart des formations informatiques, suivi par le langage propriétaire de Microsoft ASP.net, les autres langages étant largement minoritaires[53]. Généralement, les agences web travaillant avec ces méthodes ont, au fil des ans, développé un certain nombre de routines leur permettant de réutiliser des parties de code pour accélérer le développement. D'autres ont préféré l'utilisation d'un CMS ou se posent la question de leur utilisation. La frontière entre programmation et utilisation d'un CMS est cependant loin d'être étanche.

Un système de gestion de contenu, communément appelé CMS (Content Management System), est un logiciel destiné à la création et à la gestion d'un site ou d'une application web. En général, un CMS gère la persistance des données, l'authentification des utilisateurs et leurs droits d'accès, la saisie de données par les utilisateurs du site et leur publication. Pour le développeur, les CMS lui permettent de développer un site en se concentrant sur les vues à créer pour obtenir un rendu propre au site développé. Certains CMS peuvent désormais s'intégrer dans des logiciels de création web comme Dreamweaver par exemple, simplifiant le développement des vues pour le web designer. D'autres CMS comme Spip ou Typo3 par exemple requièrent l'apprentissage d'un pseudo-langage pour la réalisation des vues et du site.

La plupart des CMS permettent donc à un (presque) novice de gérer le contenu d'un site et sa publication, souvent à l'aide d'un hébergement internet de base. En plus des fonctionnalités de gestion du contenu, certains CMS permettent d'étendre leurs fonctionnalité via des *plugins* que l'on peut sélectionner dans une liste fournie par la communauté des usagers ou développer pour ses besoins spécifiques.

C'est souvent là que se perd le développeur non expérimenté, comme nous le voyons souvent avec les personnes en formation. Comment se retrouver dans la multitude de *plugins* qui en font soit trop, soit trop peu ? Quant à créer ses propres *plugins*, cela requiert des capacités de développeur et une connaissance du CMS utilisé qui vont souvent au-delà des capacités du développeur web type. Dans ce cas, la préférence va à un développement classique qui évite l'apprentissage parfois difficile des mécanismes internes du CMS. De plus la multiplication des *plugins* pose parfois des problèmes de conflits entre eux. Plus gênant, la mise à jour du système dépend alors de nombreux acteurs puisque chaque *plugin* est développé indépendamment du CMS et ne suit pas forcément les évolutions du CMS. Pire, il est parfois complètement abandonné.

Ce casse-tête des *plugins* et de leurs mises à jour justifie pour certains la recherche de

53. Source : http://w3techs.com/technologies/overview/programming_language/all

nouvelles méthodes de développement [Hinton 11], où chaque fonctionnalité est alors une API indépendante.

Une autre solution pour le développement d'applications web, entre le développement complet et l'utilisation d'un CMS, est l'utilisation d'un framework de développement web. Un framework est un kit de composants logiciels structurels, qui servent à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel [54].

Les frameworks de développement web sont des frameworks spécialisés offrant, sous la forme de classes, des fonctionnalités qui accélèrent le développement d'applications web, par exemple pour la gestion de l'authentification. Certains frameworks comme Ruby On Rails ou CakePHP [55] utilisent la méthode du *scaffolding* [56] pour créer une application de base à partir d'une spécification du modèle de l'application. L'application de base permet alors immédiatement de créer, modifier, enregistrer et lire des informations stockées dans une base de données.

De notre point de vue, les CMS sont parfaitement adaptés à la publication sur le web (qui représente la majeure partie des besoins). Toutefois, ils rencontrent leurs limites dans le développement d'applications web. La gestion de la multitude des *plugins* engendre une application difficile à maintenir [Hinton 11]. Le développement d'applications génériques est tout à fait envisageable avec un CMS. Cependant, les applications plus spécifiques qui ne rentrent pas dans des catégories prédéfinies sont beaucoup plus difficiles à développer avec un CMS. On pourrait arguer du fait que des CMS, Drupal par exemple, sont aussi des frameworks de développement. Ils offrent effectivement des moyens de programmer des fonctionnalités spécifiques (au moyen de *hook* pour Drupal), mais le développeur doit alors commencer par « défaire » ce que le CMS fait par défaut avant de réaliser ce qu'il souhaite. De plus, l'apprentissage de la programmation de ce type d'outils est souvent difficile, et parfois au-delà des capacités d'un développeur non expérimenté. Le problème est donc souvent que le CMS a effectué des choix de développement, d'interface, de gestion, et qu'ils ne sont pas nécessairement les choix dont l'application a besoin [Johnson 09]. Nous sommes donc convaincus qu'il existe un besoin pour l'aide au développement d'applications qui correspondent aux 25% non classiques décrits par l'étude de Rode [Rode & Rosson 06].

Dans cette section et la précédente, nous avons présenté des travaux montrant la diversité des personnes travaillant autour du développement web et les besoins de nouvelles méthodes pour l'ingénierie du web. Les outils existants permettent le développement d'applications type mais un certain nombre de besoins ne peuvent être satisfaits de façon générique. Ces observations seront le point de départ pour le développement de notre propre framework qui sera présenté dans la partie II.

54. Définition Wikipedia http://fr.wikipedia.org/wiki/Framework
55. http://cakephp.org/
56. http://en.wikipedia.org/wiki/Scaffold_(programming)

Dans la section suivante, nous nous intéressons plus précisément aux problématiques d'authentification des utilisateurs et de gestion des droits d'accès.

## 4.2 Droits d'accès et permissions

### 4.2.1 Introduction

La gestion des droits d'accès dans un système d'information est un aspect important de la sécurité des données. Nous n'abordons pas dans cette section la sécurité du point de vue cryptographique mais du point de vue de la définition d'une politique de permissions pour déterminer les autorisations d'accès à une ressource.

Dans le cadre d'une application web, les utilisateurs peuvent effectuer un certain nombre d'actions sur les objets du système. L'exemple le plus classique est la création de contenu et sa gestion (modification, publication). Afin que les données ne soient pas accessibles ou modifiables par tous, une application web doit donc implémenter une politique de contrôle d'accès. Cette politique est définie spécifiquement pour le site par l'organisme dont il dépend, en fonction de ses besoins, de son organisation interne et des données à gérer.

Par exemple, la politique de permissions pour un site de forums pourrait être exprimée par des phrases comme :

- tout utilisateur peut créer un compte ;
- tout utilisateur peut voir les questions et leurs réponses ;
- un utilisateur inscrit peut répondre à une question.

Ou bien, pour la politique de permissions d'une application de réseau social, nous aurions :

- tout utilisateur peut vérifier si un compte existe ;
- mon ≪ mur ≫ non public ne peut être vu que par mes amis ;
- seules les applications autorisées peuvent accéder à mon mur.

Les applications web doivent donc gérer des objets stockés en général dans une base de données. Par objets nous entendons ici par exemple un compte utilisateur, un billet de blog, un commentaire, etc. La politique de permissions détermine les actions qu'un utilisateur pourra effectuer sur un objet du système. Cette politique est souvent exprimée en langage naturel.

Polifile [57] est une application créée par C&Féditions pour créer et gérer des livres numériques au format ePub pour des individus ou de petites maisons d'éditions. Dans le cadre de Polifile par exemple, cette politique s'exprime en partie par :

- l'administrateur peut tout faire ;
- un utilisateur connecté peut créer un nouveau livre ;
- un utilisateur connecté peut lire et modifier les livres dont il est contributeur ;

57. http://polifile.fr

– un membre d'une maison d'édition peut lire tous les livres de cette maison d'édition ;
– un directeur d'une maison d'édition peut modifier tous les livres de cette maison d'édition ;

C'est en fonction de cette politique que l'application web peut répondre à des questions du type :

– « L'utilisateur X peut-il effectuer l'action Y sur l'instance Z ? »
  Exemple : l'utilisateur `jml` peut-il modifier l'objet d'identifiant `x32b` ;
– « L'utilisateur X peut-il effectuer l'action Y sur le type d'objet Z ? »
  Exemple : l'utilisateur `jml` peut-il créer un nouvel objet de type Ebook.

La réponse à ces questions permet de déterminer si l'utilisateur a le droit de faire ces actions.

La plupart des applications web utilisent des modèles de permissions existants, basés sur les listes d'accès ou sur l'existence de rôles. La section suivante présente ces modèles.

### 4.2.2 Modèles de permissions

Deux modèles de permissions sont principalement implémentés pour les applications web : *Role Based Access Control* ou *Access Control List* que nous présentons brièvement ici. Nous renvoyons à la lecture de [Tolone *et al.* 05] pour un état de l'art plus détaillé sur les modèles de permissions.

#### RBAC

RBAC, ou *Role Based Access Control* [Sandhu *et al.* 96], est un modèle permettant gérer les droits d'accès des utilisateurs d'une application (au sens général et pas nécessairement une application web) en leur affectant un rôle auquel est associé une liste de permissions. Par exemple le rôle *admin* permet en général de réaliser toutes les actions possibles sur le système. Tout utilisateur ayant ce rôle est donc autorisé à gérer l'application. Un autre exemple de rôle est celui de contributeur, qui peut rédiger des contenus mais ne peut pas les rendre public sur le site.

La différence avec l'utilisation de groupes d'utilisateur réside dans le fait que le rôle est le lien entre un ensemble d'utilisateurs et un ensemble de permissions. Cela permet une stabilité dans la gestion des permissions puisque les rôles sont en général plus stables que les groupes d'utilisateurs. Le modèle RBAC s'est imposé comme le plus utilisé dans les systèmes web et permet de couvrir la plupart des besoins. RBAC ne permet cependant pas de définir des permissions fines sur certains utilisateurs et/ou certains objets précis du système. Dans les applications de travail collaboratif par exemple, un utilisateur aura besoin de permissions, non pas sur un type d'objet, mais sur une instance d'un objet particulier.

L'utilisation de groupes, avec le modèle *Team-Based Access Control* (TMAC) [Thomas 97],

associe les permissions à un groupe d'utilisateurs plutôt qu'à un individu. Cette approche est en général utilisée dans les systèmes collaboratifs pour lesquels RBAC montre ses limites, en particulier pour regrouper des instances de rôles ou utiliser une gestion plus fine des permissions. TMAC cependant ne prend pas en compte l'administration des droits accordés et la gestion des droits peut donc être difficile.

D'autres variantes existent, par exemple *Task-Based Access Control*, *Spacial Access Control*, ou *Context-Aware Access Control* mais, dans le cadre d'applications web, ces modèles sont relativement peu appliqués.

### ACLs

Les *Access Control List* (ACLs) sont un autre mode de contrôle d'accès. La liste des permissions est rattachée à un objet du système. Cet objet peut être un fichier dans le cas d'un système d'exploitation. Dans le cas d'une application web, l'objet auquel les permissions sont rattachées correspondra souvent à un objet géré par l'application (un document, un fichier, un utilisateur par exemple). Les ACLs sont le plus souvent implémentées en utilisant une table de base de données qui liste les autorisations d'accès aux objets. Un des inconvénients des ACLs, outre le fait de générer de nombreux enregistrements en base de données, est l'impossibilité de savoir quel utilisateur a posé une autorisation sur un objet.

En ce qui concerne les CMS, la méthode RBAC est souvent utilisée, comme par exemple dans Drupal ou Wordpress[58], deux des logiciels les plus utilisés sur le web. Dans le cas de Wordpress, RBAC est utilisé avec un ensemble de rôles prédéfinis qui correspondent aux différents intervenants d'un blog :

- *Subscriber* est le lecteur simple, il ne peut que lire des billets de blog ;
- *Contributor* peut créer des billets mais ne peut pas les publier ;
- *Author* peut créer et gérer ses propres billets seulement ;
- *Editor* peut créer et gérer ses propres billets mais aussi ceux d'autres utilisateurs ;
- *Administrator* peut tout faire ou presque.

Cela devient difficile à utiliser lorsque l'on veut étendre l'utilisation du logiciel à d'autres types d'applications, même si des *plugins* permettent d'altérer le comportement par défaut. Drupal définit de même des rôles par défaut et permet d'en définir d'autres. Des *plugins* permettent là aussi de changer le comportement du CMS et d'y ajouter des ACLs par exemple. La gestion des droits devient alors malheureusement difficile pour l'administrateur du site qui se retrouve avec de multiples paramètres à gérer, se traduisant souvent par des tableaux avec d'innombrables cases à cocher comme le montre l'exemple de la figure 4.3.

Comme Steve Barker [Barker 09], nous pensons que RBAC n'est pas suffisant pour la

58. http://wordpress.org/

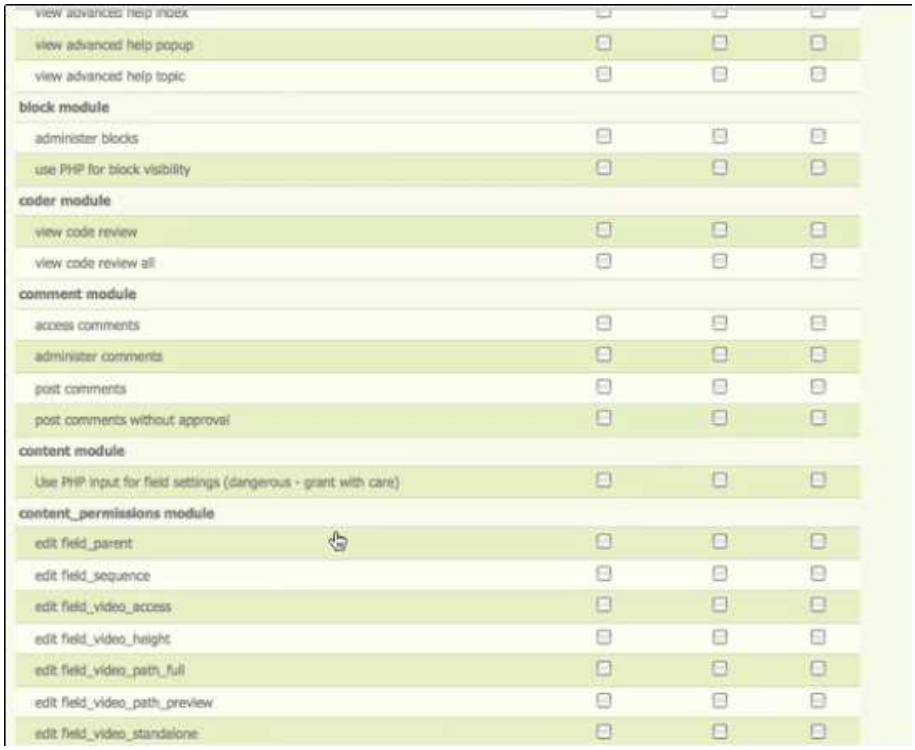



FIGURE 4.3 – Exemple d'interface de gestion des permissions

gestion des droits d'une application web. Certains aspects des permissions ne sont couverts ni par RBAC, ni par les ACLs. Par exemple dans le cas d'un site permettant le partage de documents dans un groupe, la politique de permissions sera plutôt du type TMAC. Un framework doit offrir au développeur le choix de son modèle de permissions et la possibilité d'en mixer plusieurs.

## 4.3 Pourquoi un nouveau CMS/framework ?

Nous avons abordé dans les sections précédentes certaines insuffisances que nous avons constaté dans les systèmes de gestion de contenu pour le développement d'applications web. Se tourner vers les frameworks de développement web semblait alors une solution plus satisfaisante. Pourquoi dans ce cas ne pas utiliser un framework existant pour nos travaux ?

### 4.3.1 Motivations

Les frameworks existants offrent un certain nombre de fonctionnalités mais, de notre point de vue, restent en deçà de ce qu'un développeur web aimerait trouver. En particulier, il manque des mécanismes qui vont aider le développeur dans sa tâche quotidienne. En effet, en plus de la gestion des enregistrements (ce que fait par défaut un framework utilisant le *scaffolding*), un développeur a besoin de mécanismes pour gérer les métadonnées des documents, les rela-

tions entre les documents et en particulier entre versions linguistiques. En bref, un développeur aimerait avoir une application de base qu'il peut configurer, modifier et étendre. D'un côté les CMS nous semblent trop contraignants dans leur version de base et trop difficiles d'accès pour modifier leur comportement en les utilisant comme framework. Et de l'autre côté les frameworks n'apportent pas suffisamment de *routines* préprogrammées pour faciliter et accélérer le processus de création d'applications web. Nous estimons donc qu'il y a un manque dans les outils à la disposition du public des développeurs, un outil qui se situe entre les CMS et les frameworks. Comme nous l'avons dit précédemment, il existe un vide pour l'aide au développeur qui veut créer des applications web spécifiques et partir de son modèle d'application plutôt que d'un modèle existant. Nous avons alors décidé de commencer la création de Sydonie, SYstème de gestion de DOcuments Numériques pour l'Internet et l'Édition, dont nous présenterons certains mécanismes dans la partie suivante.

### 4.3.2 Gestion de documents

Contrairement aux CMS qui, comme leur nom l'indique, permettent de gérer le *contenu* des sites web, nous souhaitons réaliser un framework qui permet la gestion des *documents* publiés dans le site web.

Dans le cadre de la gestion de documents, la plupart des CMS gèrent de façon simple les documents liés. Dans le cas d'une image illustrant un article par exemple, les métadonnées de l'image sont dans la plupart des cas limitées au titre, légende et texte alternatif de l'image, alors que de, notre point de vue, l'image elle-même est un document qui mérite d'être traité en tant que tel, avec une liste de métadonnées descriptives, techniques et administratives, permettant par exemple de gérer les droits artistiques [SAA Photo Metadata Project]. Le système que nous souhaitons créer doit donc gérer les documents intégrés au sein d'autres documents. Les métadonnées associées aux documents doivent pouvoir être gérées de façon simple, à la fois par l'utilisateur final au moyen d'interfaces adaptées, mais aussi par le développeur d'applications, auquel le framework doit fournir les outils adaptés à ces traitements.

Le système ainsi créé doit proposer en standard la gestion des documents : création, stockage, publication, métadonnées, import/export. Nous souhaitons aussi avoir un système qui permet de travailler avec des documents composites multilingues. Le modèle de document utilisé, le mode de création de nouveaux types de documents et la gestion des métadonnées seront présentés dans la partie suivante.

### 4.3.3 Souplesse et ergonomie de développement

#### CMS : des avantages et des inconvénients

Les systèmes de gestion de contenu permettent de réaliser des applications de type donné, où les fonctionnalités sont similaires à chaque développement. Nous avons cependant souligné certains inconvénients de l'utilisation des CMS, en particulier le fait d'enfermer le développeur dans la logique du CMS. De plus, nous avons vu que l'utilisation de *plugins*, pratique dans le cas d'applications assez génériques, pose rapidement des problèmes de compatibilité et de maintenance de l'application. Cette utilisation transforme vite le développement en un jeu de puzzle plutôt qu'en une réflexion en amont du développement de l'application. Nous avons maintes fois observé ce phénomène lors de projets d'étudiants. Les fonctionnalités du site deviennent dépendantes des *plugins* que le développeur a réussi à mettre en place. La technique vient alors forcer la main à la créativité et à l'imagination du début de projet, alors même que cette technique ne devrait qu'être au service du projet et non lui imposer des limites.

#### Modélisation des applications

Nous pensons que la priorité du développeur doit être la modélisation de son application. Dans ce cadre, le système utilisé doit lui permettre de penser son modèle applicatif sans contraintes *à priori*. Là encore, combien de fois nous sommes-nous vus opposer le rédhibitoire « ce n'est pas possible » à cause de telle ou telle limitation ou conflit de *plugin*. Le développeur ne doit pas être contraint de faire entrer son modèle d'application dans un moule. L'accent doit être mis sur l'extensibilité du modèle pour le développeur.

#### Langages et souplesse de développement

Nous pensons de plus que le développeur doit utiliser les langages qu'il maîtrise déjà. HTML et CSS sont bien entendu indispensables pour le développement web. Pour le langage côté serveur, nous avons montré que PHP est le langage le plus utilisé pour le développement d'applications web. Il est de plus enseigné dans la plupart des formations informatiques. Nous avons donc opté pour celui-ci dans le cadre du framework Sydonie.

Il est aussi important d'éviter la création de pseudo-langage spécifique au framework. Le développeur souhaite en général utiliser ses connaissances sans devoir apprendre une nouvelle syntaxe pour commencer. C'est une raison pour laquelle certains n'utilisent pas Spip par exemple qui fonctionne avec une logique de « boucles » pour la création des pages. De même Typo3 utilise une logique particulière pour la création des pages de contenu. En revanche, le choix de Wordpress, qui utilise uniquement HTML et PHP dans les templates, nous semble tout à fait adapté aux besoins des développeurs.

Enfin, la souplesse de développement doit aussi se traduire par la possibilité pour le programmeur de travailler de façon « classique ». Les développements réalisés de façon indépendante doivent pouvoir s'intégrer dans le fonctionnement de l'application réalisée avec le framework. De nouveau, le développeur ne doit pas être contraint de travailler de façon unique.

En plus de la flexibilité, une réflexion sur l'ergonomie de développement a été menée pour faciliter le travail du web designer. En observant les pratiques des développeurs et grâce à notre expérience dans ce domaine, en tant que formateur et développeur, le framework Sydonie propose des outils qui facilitent la tâche du développeur : utilisation de fichiers de configuration simples, routines pour les interactions Ajax et les boites modales, etc.

La partie II présente les solutions que nous avons mises en œuvre pour offrir aux développeurs une flexibilité de travail qui permet de se concentrer sur l'application à réaliser.

## 4.4 Synthèse

Dans ce chapitre, nous avons tenté de tracer un panorama du paysage de l'Ingénierie du Web. Nous avons montré la diversité des acteurs et la complexité du développement web. Les outils existants, s'ils permettent de satisfaire les principaux besoins des sites web, deviennent un carcan pour développer des applications web spécifiques. Nous pensons que la technique ne doit pas être un frein à la créativité et nous avons montré comment, dans certains cas, l'utilisation de CMS opère en ce sens.

Après avoir présenté les principaux modèles de permissions, nous avons montré leur utilisation par les systèmes de gestion de contenu. Ces politiques de permissions définies selon un modèle prédéfini par le logiciel nous semblent réductrices. Là encore, la technique passe devant la modélisation de l'application à créer.

Nous avons enfin dans ce chapitre présenté les motivations qui nous ont amené à vouloir penser et développer de nouveaux outils. Pour terminer nous avons tracé les grandes lignes des fonctionnalités que nous souhaitons voir dans ce nouvel outil. La partie II présente en détails le modèle et les stratégies mises en place pour la création de Sydonie.

# Synthèse de l'état de l'art

L'état de l'art que nous avons présenté dans cette partie est principalement articulé autour de trois points.

Tout d'abord nous avons cherché à pointer les évolutions des techniques et des contenus sur le web. Celles-ci nous montrent comment les logiques de documents et d'applications doivent coïncider sur le web. Elles montrent aussi comment documents et métadonnées sont intriquement liés. Les deux aspects ne peuvent être dissociés et la gestion des documents dans un système tel que nous l'imaginons doit permettre la gestion fine des métadonnées.

Le deuxième aspect est présenté au travers des travaux réalisés par les bibliothécaires, et en particulier le rapport sur les spécifications fonctionnelles des notices bibliographiques (FRBR). Les FRBR proposent un modèle pour la description de documents et de leurs divers formats, versions linguistiques, sous la forme d'une structure arborescente. Nous avons montré au chapitre 3 comment les concepts des FRBR peuvent s'appliquer pour la modélisation des documents composites et des documents multilingues. Le chapitre 6 montrera comment nous avons utilisé ce modèle pour la gestion de documents et le leurs métadonnées, faisant ainsi le lien entre les deux premiers points de cet état de l'art.

Enfin, les problématiques de l'ingénierie du web constituent le troisième aspect abordé dans cette partie. Nous avons souligné la coexistence de deux types d'acteurs, les web designers et les concepteurs informaticiens. Si les besoins en développement peuvent être couverts en grande partie par l'utilisation de CMS, nous avons montré leurs limites pour la création d'applications web. L'analyse des besoins en ingénierie du web et notre expérience dans le domaine de la formation de professionnels du web nous a amené à préciser nos attentes dans l'outil que nous souhaitons créer.

La partie suivante s'attachera à présenter les contributions réalisées dans le cadre de cette thèse.

# Deuxième partie

# Contributions

# Introduction

La première partie de ce mémoire a présenté un certain nombre de concepts, modèles et techniques pouvant être utilisés dans la gestion de documents et la création web. Cette partie présente nos travaux s'appuyant sur ces concepts.

Comme pour la partie précédente, celle-ci est articulée en trois chapitres. Le premier chapitre présente le modèle de document que nous avons choisi pour travailler avec des documents numériques composites, basé sur une métaphore autour des FRBR. La second chapitre présente notre travail sur les métadonnées et les documents, et comment nous introduisons un pivot pour l'extraction de métadonnées et leur inscription dans les documents. Le troisième chapitre sera axé sur le génie logiciel mis en œuvre afin de réaliser un framework opérationnel.

Nous parlerons beaucoup de Sydonie dans cette partie. Qu'est-ce que Sydonie et quels sont les objectifs de ce projet ?

Sydonie, pour SYstème de gestion de DOcuments Numériques pour l'Internet et l'Édition, est un projet de l'équipe DLU du laboratoire GREYC. Il a reçu le soutien du Conseil Régional de Basse-Normandie, du projet TGE-Adonis, et est développé en partenariat avec la maison d'édition C&Féditions [59].

Sydonie peut être abordé sous deux angles différents mais qui se rejoignent. La première vision de Sydonie est celle qui fournit un modèle de document pour la création, la gestion et la publication de documents composites multilingues. Partant de l'expérience des bibliothèques et intégrant l'interdisciplinarité des projets menés, nous avons établi un modèle pour les documents gérés par le framework Sydonie. La vision *document* peut donc être la première approche de Sydonie. Le premier chapitre présente cette approche et son implémentation dans le framework.

La deuxième approche de Sydonie est sous l'angle du développement web et du génie logiciel. En effet, un des points de départ du projet Sydonie est l'insatisfaction, au sein de notre équipe, avec les outils disponibles pour le développement de sites et d'applications web. Sydonie a donc pour objectif de fournir aux développeurs web des outils adaptés aux besoins spécifiques du web d'aujourd'hui, et si possible de demain. Sydonie n'est donc ni un CMS, ni un framework généraliste tels que nous les avons définis à la section 4.1, mais plutôt un framework applicatif

59. http://cfeditions.com

pour le développement web. Ce travail implique une réflexion au niveau génie logiciel qui prend en compte les règles de l'art du développement informatique, mais qui doit aussi prendre en compte les contraintes et réalités du monde professionnel du web. Notre expérience dans le domaine de la formation de professionnels en Ingénierie du web nous a apporté nombre de leçons. Les relations avec les entreprises et professionnels du domaine nous ont permis d'avoir une bonne idée du concret des développeurs web au quotidien. Le chapitre 7 présente Sydonie sous l'angle *génie logiciel*.

Enfin le chapitre 6 présente la gestion des métadonnées dans le framework Sydonie. Ce chapitre est transverse aux deux approches mentionnées ci-dessus puisqu'elle utilise le modèle de document implémenté dans Sydonie et fournit aux développeurs web des outils pour gérer efficacement les métadonnées dans les applications web.

# Chapitre 5

# Sydonie : modèle de document

**Sommaire**



La partie précédente a posé les bases de notre travail. Nous avons présenté au chapitre 1 les évolutions du web et les conséquences pour le développement d'applications web et la gestion de documents. Au chapitre 4, nous avons analysé les limites des systèmes de gestion de contenu et nous en avons tiré des conclusions quant à ce que nous attendions d'un nouveau framework en termes de flexibilité et ergonomie de développement.

Dans le domaine de la gestion de documents, nous avons présenté au chapitre 2 les travaux de l'IFLA, consignés dans le rapport fonctionnel sur les notices bibliographiques FRBR. Nous avons choisi d'étudier le modèle défini par les FRBR et nous avons présenté au chapitre 3 comment ce modèle représente les documents composites et les documents multilingues.

Ce chapitre présente le modèle de document utilisé par le framework Sydonie, que nous avons présenté lors de la conférence Document Engineering [Lecarpentier *et al.* 10]. Ce modèle est directement inspiré des FRBR. Nous présentons tout d'abord comment nous avons adapté la structure arborescente de document autour des entités du groupe 1. Notre interprétation des recommandations du rapport FRBR nous amène à un modèle de document qui permet de gérer l'aspect composite multilingue des documents numériques et leurs métadonnées. Nous présentons ensuite comment le framework permet de définir de nouveaux types de documents. Enfin, nous

présentons les stratégies mises en place pour la recomposition des documents. Nous concluons par un premier bilan et une discussion autour de ces différents points.

## 5.1 Sydonie : un modèle comme métaphore des FRBR

Les FRBR ont été pensés et élaborés pour les notices bibliographiques des bibliothèques. Dans le cadre de bibliothèques numériques ou d'applications web, l'exemplaire physique n'existe pas, ou au moins n'a pas le même sens. De plus, là où une bibliothèque numérique ne stocke que les *métadonnées* des documents présents sous la forme des notices, une application web devra conserver à la fois les métadonnées *et* le contenu d'un document.

Il est donc nécessaire d'adapter le modèle de structure fourni par les FRBR aux besoins des applications web. Pour cela, nous revenons sur les différents aspects des entités du groupe 1 et analysons comment ils peuvent être utilisés dans le contexte de notre travail.

Soulignons de nouveau ici, comme le fait P. Le Boeuf [Le Bœuf 03], que les FRBR *ne sont pas* un modèles de données. Les FRBR fournissent un cadre intellectuel pour typer les données et les mettre en relation. En particulier les attributs définis pour chaque type d'entité ne revêtissent absolument pas un caractère normatif. Nous citons[60] :

> ≪ Il s'agissait d'élaborer un cadre conceptuel permettant de comprendre clairement, sous une forme précisément exprimée et dans un langage qui soit parlant pour tout le monde, l'essence même de ce sur quoi la notice bibliographique est censée renseigner, et l'essence même de ce que nous attendons de la notice en termes d'adéquation aux besoins des utilisateurs ≫

Nous nous plaçons donc dans ce cadre et essayons de transposer les concepts des FRBR au domaine des applications web et de la gestion de documents sur le web.

L'entité Work représente la création intellectuelle ou artistique de l'œuvre, et par conséquent est une entité complètement abstraite. Nous pouvons donc y assigner des attributs similaires à ceux définis dans les FRBR, en considérant comme obligatoire l'attribut titre uniforme et en laissant la possibilité d'indiquer toute autre métadonnée susceptible de renseigner sur l'œuvre.

L'entité Expression représente la forme intellectuelle ou artistique de l'œuvre chaque fois que celle-ci est réalisée. Une entité Work se réalise donc en une ou plusieurs expressions. Ces expressions peuvent éventuellement avoir des relations entre elles comme nous l'avons indiqué dans le tableau 3.4 de la section 3.2.3. Nous considérerons, pour l'instant, uniquement les relations de type *est une traduction de* entre deux expressions. L'ensemble des entités Expression représente dans ce cas l'ensemble des versions linguistiques d'une entité Work. Nous qualifierons la version originale de l'expression de *référence*, à partir de laquelle les autres expressions sont créées. Une

60. Rapport final sur les FRBR, version française, page 8.

entité Expression doit donc avoir les attributs titre et langue. De même que pour l'entité Work, l'éventualité d'indiquer d'autres métadonnées doit être ouverte.

L'entité Manifestation est la *matérialisation* de l'Expression d'une entité Work. Le terme matérialisation est ici ambigu puisque nous sommes dans le domaine du numérique. Dans le rapport FRBR, le concept de matérialisation implique souvent une notion de support, par exemple (extrait des FRBR) :

- Work : *Les Six suites pour violoncelle seul* de J. S. Bach
  - Expression 1 : les interprétations de Janos Starker enregistrées en 1963 et 1965
    - Manifestation 1 : les enregistrements commercialisés en 1965 sur 33 tours par Mercury
    - Manifestation 2 : les enregistrements réédités en 1991 sur disque compact par Mercury
  - Expression 2 : les interprétations de Yo-Yo Ma enregistrées en 1983
    - Manifestation 1 : les enregistrements commercialisés en 1983 sur 33 tours par CBS
    - Manifestation 2 : les enregistrements réédités en 1992 sur disque compact par CBS

En ce qui concerne le numérique, la traduction en français FRBR propose l'exemple suivant :

- Work : Les Functional requirements for bibliographic records
  - Expression 1 : le texte original
    - Manifestation 1 : l'édition papier chez Saur en 1998
    - Manifestation 2 : l'édition électronique en ligne en format PDF
    - Manifestation 3 : l'édition électronique en ligne en format HTML

Dans le cadre de Sydonie, les Manifestations « physiques » n'existent pas. Nous considérons donc que les différentes Manifestations d'une Expression sont les divers formats sous laquelle l'Expression est disponible. De même que pour les Expressions, une Manifestation de *référence* est celle utilisée pour créer les autres Manifestations par un processus automatique ou non. Enfin, à chaque Manifestation peut être associé une ressource qui représentera le contenu de la Manifestation.

En ce qui concerne l'entité Item, nous avons choisi de ne pas l'utiliser puisque l'exemplaire physique n'existe pas dans le cadre de notre application. Les ressources étant en relation unaires avec les Manifestations, ce ne sont donc pas des Items.

Le modèle de document que nous venons de décrire est une métaphore sur les FRBR. La structure arborescente est similaire, mais les significations des différentes entités sont adaptées à l'environnement numérique. De plus, nous appellerons *Document* l'ensemble des entités, organisées sous la forme d'une arbre, qui composent les versions linguistiques et formats dans lesquels un document est disponible dans le système. Les figures 5.1 et ref 5.2 illustrent la structure obtenue.

Dans cette section, nous avons vu comment, grâce à la représentation issue des FRBR, un document connaît ses différentes versions et formats. La section suivante présente la façon les

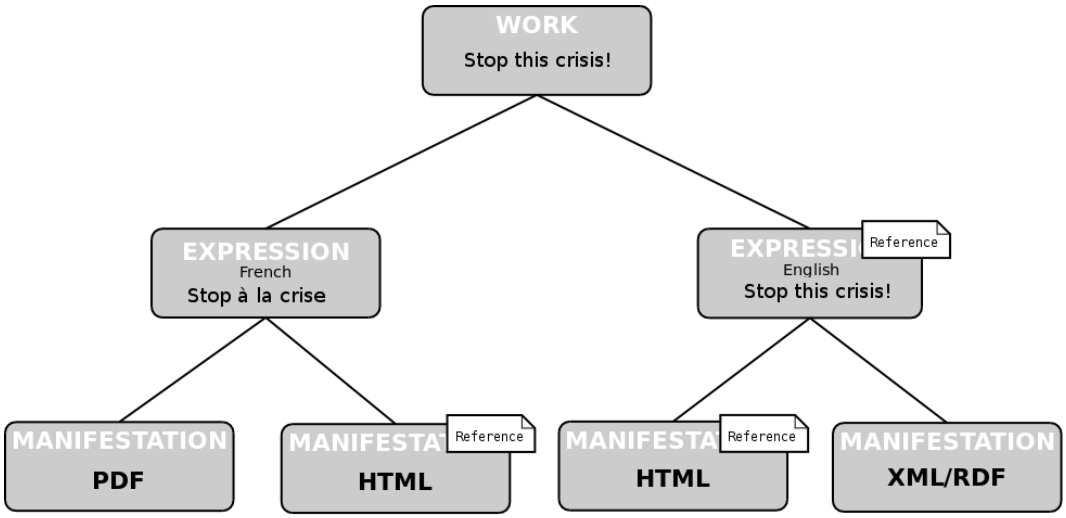



FIGURE 5.1 – Structure arborescente d'un document

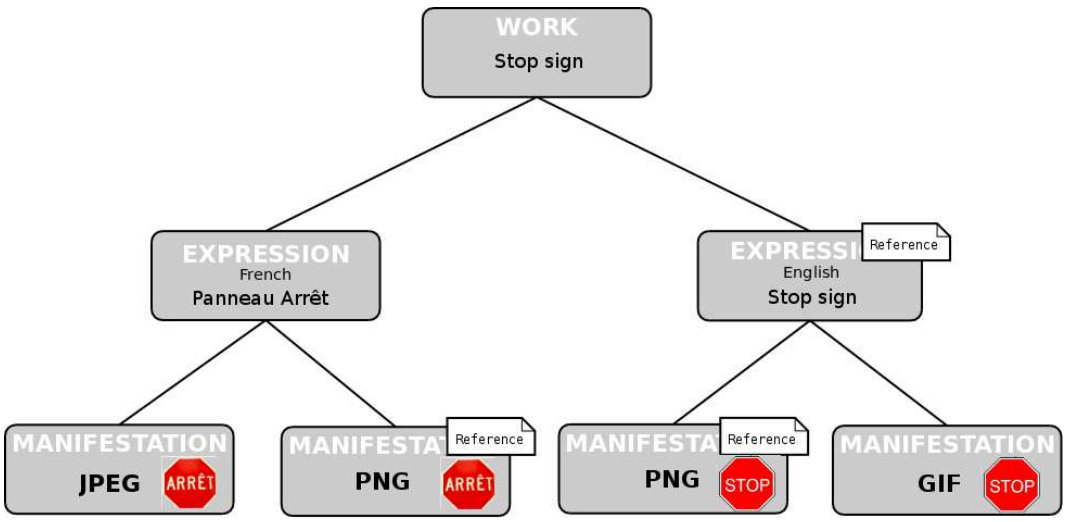



FIGURE 5.2 – Structure arborescente d'un document de type image

données et métadonnées d'un document sont gérées au sein du modèle de Sydonie.

## 5.2 Structure des données et implémentation

Les documents gérés par Sydonie sont représentés sous la forme d'un arbre, avec les entités Work, Expression et Manifestation. Nous avons présenté, section 2.1, comment les FRBR définissent des attributs pour chaque type d'entité. Une différence entre le modèle des FRBR et le modèle que nous élaborons pour Sydonie réside dans le fait que Sydonie devra gérer les métadonnées des documents ainsi que le contenu des documents, alors que les notices bibliographiques ne contiennent que des métadonnées et pas le contenu. Nous allons donc présenter dans cette section comment Sydonie gère les informations d'un document, qu'elles soient des données ou métadonnées.

### 5.2.1 Entités, attributs et ressources

Sydonie se voulant être un framework généraliste pour les applications web, il est nécessaire d'avoir une structure de données qui soit le plus souple possible afin de s'adapter à des types de documents aussi variés que possible. Il est donc exclu d'assigner à chaque type d'entité une liste fixe d'attributs, ce que d'ailleurs les FRBR se refusent à faire. Afin de rendre la liste des attributs flexible, Sydonie utilise un modèle similaire au modèle RDF, sous la forme de triplets sujet-prédicat-objet, où :

- sujet est l'entité (i.e. le nœud de l'arbre) à laquelle l'attribut est rattaché ;
- prédicat est la définition du nom de l'attribut ;
- objet est le contenu de l'attribut, lui-même sous la forme d'un objet, permettant ainsi de définir des valeurs autres que des scalaires.

De même que pour le modèle FRBR, un attribut peut être de type simple ou multiple, sous la forme d'une liste (ordonnée ou non). La figure 5.3 reprend l'exemple figure 5.1 de la section précédente, en ne présentant qu'une branche de l'arbre avec les attributs qui pourraient être définis pour ce document, leurs valeurs et leurs types. Dans cet exemple, le *contenu* du document n'est pas représenté.

L'exemple représenté par la figure 5.3 montre les différents attributs que l'on peut imaginer pour la branche du document représenté. Afin de laisser la possibilité de définir le type d'attribut, Sydonie définit une classe abstraite SydonieAttribute. Pour créer les types d'attributs nécessaires à une application, il suffit de créer les classes dérivant de SydonieAttribute en définissant leur comportement.

Les types d'attributs peuvent être simples comme du texte seul ou plus complexes, composés de diverses données, comme par exemple une adresse postale avec les informations de $n^o$, de rue, de code postal et de ville.

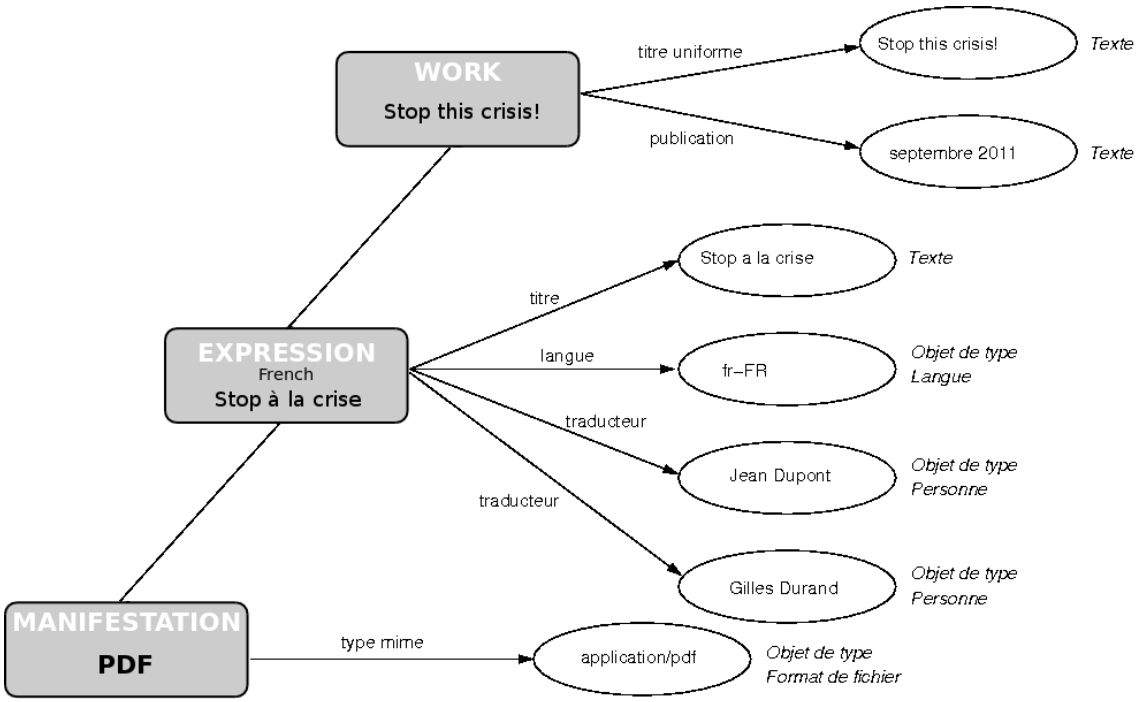



FIGURE 5.3 – Branche d'un document avec ses attributs, leurs valeurs et leurs types

Nous avons souligné plus haut que, à la différence d'une bibliothèque, Sydonie doit gérer le contenu des documents. Le contenu d'un document peut être simplement textuel mais il peut aussi, comme dans le cas de l'exemple précédent, être un fichier. Dans ce cas, un attribut de type *Ressource* permet de gérer le contenu-fichier du document. Nous présenterons plus en détails les avantages de cette solution à la section 7.5.1.

Sydonie, grâce à la structure arborescente inspirée des FRBR, permet donc de modéliser diverses versions de document, en stockant les informations à trois niveaux différents : Work, Expression et Manifestation. Au sein du framework Sydonie, un document est donc un ensemble cohérent de contenu et de métadonnées, organisé dans une structure d'arbre.

La flexibilité du système permet de choisir quels types de données et de métadonnées seront « attachées » à tel ou tel nœud de l'arbre. Il faut cependant un mécanisme pour définir les données à stocker en attributs. La section suivante explique comment le framework Sydonie réalise cette fonctionnalité.

### 5.2.2 Types de documents

Les types de documents et les données à stocker pour chaque type varient selon l'application à laquelle ils sont destinés. Nous avions envisagé au départ de créer des types de documents « standards » avec un ensemble de données à différents niveaux de l'arbre, types de documents qui seraient cependant flexibles pour pouvoir s'adapter aux divers cas rencontrés.

Cette démarche est vite apparue contraire aux principes énoncés dans la partie précédente. En effet, pour que Sydonie soit réellement un framework généraliste, il fallait que chaque développeur puisse définir ses propres types de documents, sans être contraint d'utiliser les objets prédéfinis (ce que nous reprochons aux CMS en général). En sens inverse, nous avons prévu un processus pour définir des types de documents à partir de types génériques prédéfinis.

Sydonie dispose donc d'une classe abstraite SydonieDocument qui, comme pour les AttributeType, définit les fonctionnalités de base d'un document. La classe SydonieDocument met en œuvre le modèle de document sous forme d'arbre présenté précédemment. Les attributs définis par défaut sont ceux qui nous ont paru indispensable dans le cadre du modèle de Sydonie :

- au niveau Work : titre uniforme, informations sur la date de première publication ;
- au niveau Expression : titre, description, langue de l'expression ;
- au niveau Manifestation : type mime et taille du contenu, contenu.

Pour définir un type de document, il suffit alors de créer une classe qui dérive de SydonieDocument. On définit les données attachées aux nœuds des entités dans un fichier de configuration. Ce fichier, au format XML, permet à un développeur web de créer les types de documents qu'il souhaite. Pour chaque niveau d'entité et pour chaque attribut, il faut spécifier le type, le nom du prédicat associé, sa cardinalité, plus un nombre d'options éventuelles. La figure 5.4 montre un exemple de configuration partielle du type de document *Blog Post* pour la réalisation d'un blog. Ce processus permet à un développeur, et surtout à un concepteur de site, de définir préalablement les objets dont il a besoin pour son application avant de se poser la question de l'implémentation. Il n'aura pas à « plier » son modèle pour l'adapter à des objets préconçus.

Nous aborderons au chapitre 7 les stratégies logicielles mises en place pour faciliter ce travail lors de la création de types de documents.

Un type de document est donc modélisé sous la forme d'un arbre avec des relations de type RDF sujet-prédicat-objet vers les divers attributs qui le composent. Ce modèle permet à un document d'encapsuler les différentes versions linguistiques et les formats disponibles. Dans le cadre d'une application web, l'accès aux documents peut se faire à tout niveau dans l'arbre représentant le document. Cependant, la vue qui sera servie à un lecteur (ou agent utilisateur, *User-Agent*[61]) est toujours la vue d'une Manifestation, ce qui correspond à une vue sur une branche particulière de l'arbre. Pour constituer la vue de la Manifestation choisie, la fusion de tous les attributs des nœuds de la branche de l'arbre sera utilisée.

Le système doit donc pouvoir déterminer quelle branche utiliser pour construire la vue qui sera présentée. De plus, lorsque le document est un document composite, il faudra déterminer les Manifestations des composant à inclure dans la vue du document. La section suivante présente les solutions apportées par Sydonie.

61. Définition de User-Agent sur Wikipedia : http://fr.wikipedia.org/wiki/User-Agent

```
<configuration>
  <class>
    <name>SydonieDocument_BlogPost</name>
    <extends>Abstract_SydonieDocument</extends>
  </class>
  <attribute entityLevel="work"
             minOccur="1"
             maxOccur="1" >
    <predicate>publicationDate</predicate>
    <objectClass>AttributeType_Date</objectClass>
  </attribute>

  <attribute entityLevel="work"
             minOccur="1"
             maxOccur="1" >
    <predicate>commentsOpen</predicate>
    <objectClass>AttributeType_Boolean</objectClass>
  </attribute>

  <attribute entityLevel="expression"
             minOccur="1"
             maxOccur="1" >
    <predicate>lang</predicate>
    <objectClass>AttributeType_Lang</objectClass>
  </attribute>

  <attribute entityLevel="manifestation"
             minOccur="1"
             maxOccur="1" >
    <predicate>content</predicate>
    <objectClass>AttributeType_Text</objectClass>
  </attribute>
</configuration>
```

FIGURE 5.4 – Configuration du type de document BlogPost


FIGURE 5.5 – Préférences de Firefox pour le choix des langues

### 5.2.3 Principe de négociation

Lorsqu'un utilisateur demande à visualiser un document, il faut déterminer l'Expression à lui présenter et choisir parmi les Manifestations. Si l'on reprend l'exemple illustré figure 5.1, le système devra déterminer s'il doit présenter la version anglaise ou la traduction française de l'article. Selon la langue utilisée, le système devra ensuite déterminer si le format à renvoyer au client doit être HTML, PDF ou RDF.

Pour aborder la négociation de contenu, nous avons utilisé le rapport du W3C *Cool URIs for the semantic web* [SAUERMANN *et al.* 08], publié en décembre 2008. Ce rapport donne des indications sur la stratégie à utiliser pour déterminer quelle version d'un document et sous quel format un serveur web devra servir chaque client.

Deux types de négociations de contenu entre un client et le serveur sont principalement utilisés et préconisés dans ce rapport. Ces négociations sont basées sur le protocole HTTP qui permet à deux machines d'entrer en négociation pour déterminer le contenu adapté à une requête grâce aux en-têtes échangés.

Le premier type, *Language-Negotiation*, permet au serveur de connaître les préférences linguistiques du *User-Agent* qui accède à la ressource. Dans le cas d'un utilisateur humain par exemple, cela correspond aux préférences de langues du navigateur qu'il utilise. Dans le cas d'un robot, ce sera la langue définie par les développeurs du robot. Les préférences linguistiques peuvent lister un ensemble de langues avec un ordre de priorité. Par exemple, la figure 5.5 montre la configuration du navigateur Firefox pour les langues.

Selon le même principe, le deuxième type de négociation, dit *Content-Negotiation*, permet

```
GET /~jml/cf/sydV1/SydonieSite/article/a_la_une/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; fr; rv:1.9.2.18)
            Gecko/20110628 Ubuntu/10.04 (lucid) Firefox/3.6.18

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip,deflate
Accept-Charset: UTF-8,*
```

FIGURE 5.6 – En-têtes HTTP envoyés par un navigateur pour indiquer les préférences linguistiques et de format

au serveur de connaître les préférences de format du *User-Agent* qui accède à la ressource. Dans la cas d'un utilisateur humain, le navigateur demandera en général du HTML en priorité. Un robot, en revanche, préférera certainement un format d'échange plus adapté comme RDF ou un autre format XML. De même que pour les langues, ces préférences sont hiérarchisées. La figure 5.6 montre un exemple d'en-têtes HTTP transmis à un serveur par un navigateur web, les caractères * signifiant que toute autre langue ou format est accepté en dernier recours. On peut remarquer que l'ordre dans les préférences de langues et de format sont traduites dans la requête HTTP par des coefficients compris entre 0 et 1.

Le système de Sydonie va donc utiliser ces deux techniques pour déterminer la vue à construire pour un utilisateur donné. Avant toute négociation, le système doit déterminer à quel niveau d'entité le document est demandé. Rappelons que l'utilisateur recevra la vue construite à partir d'une branche de l'arbre. Le système devra donc réagir en fonction du nœud de l'arbre demandé par le client :

- si un nœud de type Work est demandé, alors le processus *Language-Negotiation* permet au serveur de déterminer quelle version linguistique utiliser, c'est-à-dire quelle Expression utiliser. La négociation devra alors continuer au niveau Expression pour déterminer le format à utiliser ;
- si un nœud de type Expression est demandé, alors le processus *Language-Negotiation* n'est pas nécessaire et seul le processus *Content-Negotiation* aura lieu pour déterminer quel format utiliser, c'est-à-dire quelle Manifestation utiliser ;
- si un nœud de type Manifestation est demandé, alors le serveur peut construire la vue sur la branche de l'arbre partant de la Manifestation demandée sans avoir à effectuer de négociation.

Si aucune des langues de prédilection de l'utilisateur n'est disponible, alors le système prendra par défaut la langue de l'interface de l'application web à l'instant de la demande. Si une


FIGURE 5.7 – Version de référence en anglais et version en français à recomposer

Expression dans cette langue n'est pas disponible, alors l'Expression de référence sera utilisée. Le même principe s'applique pour le choix de la Manifestation : si aucun des formats préférés de l'utilisateur n'est disponible, alors la manifestation de référence sera utilisée. Dans tous les cas, la vue générée peut contenir un encart informant l'utilisateur sur les autres versions et formats disponibles.

Les processus détaillés ci-dessus permettent à Sydonie de déterminer la vue la plus adaptée aux besoins d'un utilisateur. Ces négociations peuvent aussi avoir lieu dans les relations entre les documents dans le cadre des documents composites. La section suivante explique comment Sydonie réalise cette recomposition.

### 5.2.4 Négociation, Relations et Documents composites

Lorsque Sydonie doit construire la vue d'un document composite, la négociation présentée dans la section précédente permet de déterminer la Manifestation à utiliser pour le document englobant, et par conséquent la branche de l'arbre à construire. Il faut désormais, pour reconstruire le document composite, déterminer la Manifestation à utiliser pour chaque composant.

Reprenons l'exemple de l'article *Stop this crisis !* de la figure 5.1. La version originale, c'est-à-dire l'Expression de référence pour Sydonie, est en anglais. Lorsque l'Expression traduite en français devra être présentée à un utilisateur, il faudra recomposer le document : le système devra déterminer quelle Expression du document Image inclure, comme le montre la figure 5.7.

Nous avons présenté, section 3.1, les relations d'ensemble/partie définies par le rapport FRBR et qui peuvent être établies entre entités. Dans le cas de l'exemple ci-dessus, le système a donc à gérer un document englobant qui est l'article. Celui-ci possède un composant qui est une image. Cette image est elle-même un document au sein de Sydonie, avec sa représentation sous forme d'un arbre. Une relation ensemble/partie est donc établie entre le document article et l'image. La figure 5.8 montre les structures internes des deux documents et leurs relations.

Lorsque le système recompose le document pour la version en français, la relation avec le document de type image permet d'accéder à son arbre, au niveau de l'entité Work. La même

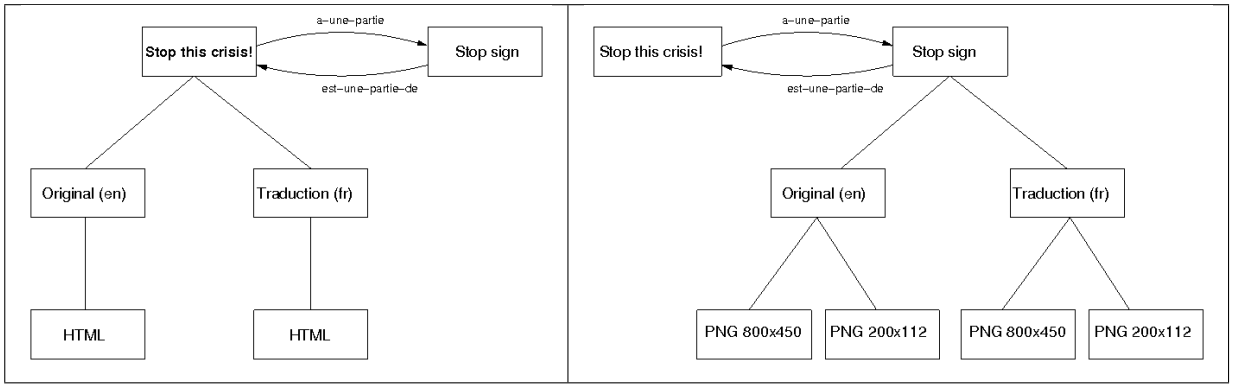



FIGURE 5.8 – Structures du document englobant et du composant


FIGURE 5.9 – Document reconstruit après négociation de contenu

négociation que celle décrite à la partie précédente permet alors de déterminer la Manifestation de l'image à utiliser. Si pour l'image une Expression en langue française est disponible alors elle sera utilisée pour reconstruire le document comme illustré figure 5.9. Si, pour une Expression donnée du document englobant, l'image ne possède pas d'Expression dans cette langue alors l'Expression de référence (en anglais dans le cas de l'exemple) sera utilisée. La figure 5.9[62] montre le résultat obtenu avec cette solution de repli.

Cette négociation permet de réaliser une reconstruction du document cohérente. En assurant une solution de repli, le système propose nécessairement un contenu, éventuellement dégradé, à l'utilisateur.

## 5.3 Modélisation des relations entre objets

Nous avons présenté au chapitre 2 les relations ensemble/partie entre entités. Dans la section précédente, nous avons utilisé cette relation pour exprimer l'inclusion d'un document dans un autre, permettant ainsi la recomposition du document englobant.

Dans certains cas, cette relation d'ensemble à partie mérite d'être qualifiée plus précisément. Par exemple, la relation pourrait exprimer la présence d'une illustration, d'un encart, etc. Cepen-

62. Les traductions de cet exemple sont des traductions automatiques, d'où leur qualité contestable.

dant, le rapport FRBR précise ne pas prétendre être exhaustif pour la liste des relations possibles.

Nous avons donc imaginé l'utilisation de relations entre objets du système. Elles sont définies par le modèle de l'application à développer. Pour cela, Sydonie définit un type de relation générique, appelé Statement. Inspiré directement de RDF, les Statements permettent de lier deux objets du système en définissant un triplet (sujet, prédicat, objet). Le prédicat définit le type de relation entre les objets. Les relations sont symétriques et chaque relation entre deux objets est traduite par deux statements symétriques.

La flexibilité de ce système permet de modéliser les relations présentes dans l'application. Par exemple, dans le cadre de l'application Polifile, un objet eBook peut contenir plusieurs images. L'objet eBook est donc en relation avec chacune des images qu'il contient. Ces relations sont représentées par deux statements de prédicats *contient l'image* et *appartient à*, sous la forme (eBook, *contient l'image*, image) et (image, *appartient à*, eBook). En revanche, une image et une seule ne peut être utilisée pour la couverture du livre. L'objet eBook est en relation avec l'image de couverture par deux statements symétriques de prédicats *a pour couverture* et *est couverture de*.

Nous reviendrons sur le fonctionnement des Statements à la section 7.4.

Les sections précédentes ont présenté le modèle de document utilisé par Sydonie et comment il est implémenté. Pour terminer ce chapitre, nous réalisons un bilan des expériences utilisant ce système et les perspectives d'améliorations et de développements futurs.

## 5.4 Discussion

Le modèle de document de Sydonie, en groupant le(s) contenu(s) des versions d'un document et les métadonnées dans une structure d'arbre, permet de réaliser la négociation de contenu décrite par le rapport *Cool URIs for the semantic web* et la recomposition de documents composites multilingues.

Quelques points restent à améliorer et tester plus en avant. Premièrement, les expériences dans un environnement multilingue sont à multiplier pour tester le modèle et son implémentation sur des données plus volumineuses. Dans le cadre des FRBR, il est spécifié que lors de l'élaboration d'une notice bibliographique d'une traduction, l'Expression qui a servi de « modèle » pour la traduction n'est pas nécessairement connue. Notre modèle part du principe que la première Expression saisie est l'Expression dite référence, à partir de laquelle les autres Expressions ont été obtenues par traduction. Cette supposition est réductrice de la réalité et doit donc être modifiée pour permettre les cas où la traduction a été réalisée soit à partir d'une autre Expression, soit à partir d'une source non connue. De plus, des mécanismes permettant de gérer la fiabilité de la traduction sont à mettre en place. Nous reviendrons sur ces problématiques dans nos conclusions

et perspectives.

En utilisant uniquement les relations de traduction entre les diverses Expressions, nous réduisons là aussi le modèle défini par les FRBR. La possibilité d'avoir des relations autres que de traduction (telles que relations de révision ou d'abrégé par exemple), ouvrirait des champs d'applications nouvelles.

Deuxièmement, les problématiques d'IHM autour du document composite ne sont pas encore réglées. Les interfaces WYSIWYG en ligne telles que CKeditor[63] ou TinyMCE[64] posent de nombreux problèmes pour gérer l'insertion de composants dans le texte d'un document en cours d'édition. Les difficultés pour créer un *plugin* permettant de réaliser cette fonctionnalité de façon satisfaisante n'ont pas encore été surmontées. En effet, l'insertion ou la suppression d'un composant dans l'interface de l'éditeur doit être synchronisée avec la création ou suppression de la relation correspondante entre les documents. Il sera intéressant de voir ce que le W3C proposera dans ce domaine, puisqu'une recommandation est en cours d'élaboration pour l'édition de documents dans le navigateur (Web editing API) [Aryeh 11].

Le système de définition de document grâce à un fichier de configuration est opérationnel et testé par plusieurs développeurs qui ont apprécié sa simplicité et rapidité de mise en œuvre. De même, la définition de nouveaux types d'attributs a été elle aussi appréciée des développeurs. Ces deux systèmes, couplés aux aides qui seront présentées chapitre 7 pour la réalisation d'applications web, permettent de développer rapidement des contenus dont les données et métadonnées sont adaptés aux applications spécifiques envisagées.

63. http://ckeditor.com
64. http://tinymce.com

# Chapitre 6

# Métadonnées et document

**Sommaire**



Dans le chapitre précédent, nous avons présenté le modèle de document utilisé dans le framework Sydonie, basé sur le cadre conceptuel des FRBR. En fonction de la nature des données, notre modèle de document permet de placer des informations à divers niveaux d'abstraction grâce aux entités Work, Expression et Manifestation.

Dans ce chapitre, nous présentons les problématiques liées à la gestion des métadonnées et comment le framework Sydonie tire parti du modèle défini précédemment. Cette gestion à différents niveaux va se révéler utile pour la gestion et la qualité des métadonnées stockées pour les documents.

## 6.1 Un framework pour les métadonnées

Nous avons décrit dans la section 3.3 les problématiques auxquelles fait face un développeur d'applications lorsqu'il doit gérer les métadonnées des documents. Nous en avons déduit les points de blocage et préconisé les apports qu'un framework doit fournir pour simplifier le travail du développeur, que nous résumons ici :

– gestion de l'extraction des métadonnées lors de l'ajout de fichiers dans le système ;

– gestion de l'inclusion des métadonnées dans les fichiers gérés par le système. Cette opération doit permettre la cohérence des données présentes dans le système et dans le fichier ;
– gestion des correspondances entre les données du modèle de l'application et le(s) schéma(s) de métadonnées utilisé(s).

Ce chapitre présente les solutions apportées par Sydonie pour répondre aux besoins exprimés ci-dessus et la façon dont les développeurs peuvent tirer profit de l'utilisation du modèle de document de Sydonie pour la gestion des métadonnées.

Au fil de ce chapitre, nous utiliserons un exemple concret pour illustrer les processus en jeu et les solutions mises en place. L'application `lisez-moi.lu`, en cours de développement par C&Féditions, est une plate-forme de vente de livres numériques, à destination des petits éditeurs. Dans le cadre de cette plate-forme, un éditeur gère les livres dont il souhaite la vente en ligne. Un livre numérique peut être disponible sous divers formats comme ePub ou PDF par exemple. S'il est important de noter que les divers formats de fichiers ne sont pas générés par un processus automatique, on considère ici qu'un nouveau type de fichier correspond à une nouvelle Manifestation. En effet, on peut faire le parallèle avec une le cas d'édition différente puisque le contenu intellectuel est le même. Seuls changent l'apparence et le contenant.

Dans le cadre de `lisez-moi.lu`, un type de document eBook a été créé au sein de l'application, en utilisant le modèle présenté au chapitre précédent. Avec ce modèle, un objet de type eBook aurait donc les propriétés :

– au niveau de l'entité Work : titre uniforme, date de première publication, auteur (un ou plusieurs). Rappelons que ces propriétés sont valables pour l'ensemble de l'arborescence du document ;
– au niveau de l'entité Expression : titre, description, langue, auxquels on ajoute l'attribut éditeur. Chaque Expression correspond à une version linguistique différente de l'œuvre ;
– au niveau de l'entité Manifestation : type de fichier, taille du fichier, auxquels on ajoute les attributs prix, ISBN, copyright. Ces propriétés dépendent du fichier lui-même.

Notons que, comme nous l'avons recommandé dans la partie 3.3.7, le modèle de l'application est alors créé de façon complètement indépendante des métadonnées que celle-ci devra intégrer dans les fichiers ou pages web.

## 6.2 Processus de gestion des métadonnées

### 6.2.1 Exemple

Dans le cas de `lisez-moi.lu`, l'ajout d'un livre au catalogue commence par l'upload d'un fichier, par exemple au format ePub. Si le fichier ePub a été créé de façon professionnelle, le fichier doit d'ores et déjà contenir un certain nombre de métadonnées concernant le livre. En

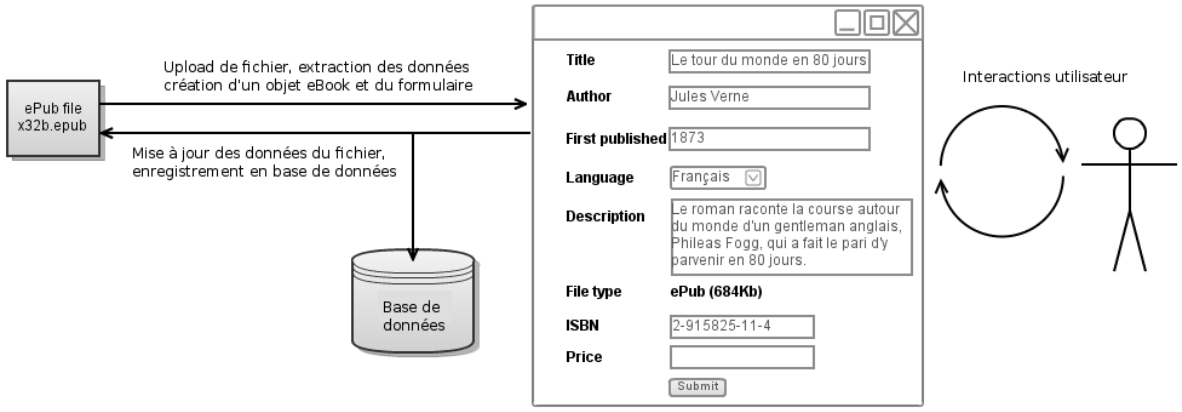



FIGURE 6.1 – Processus d'ajout d'un livre au catalogue

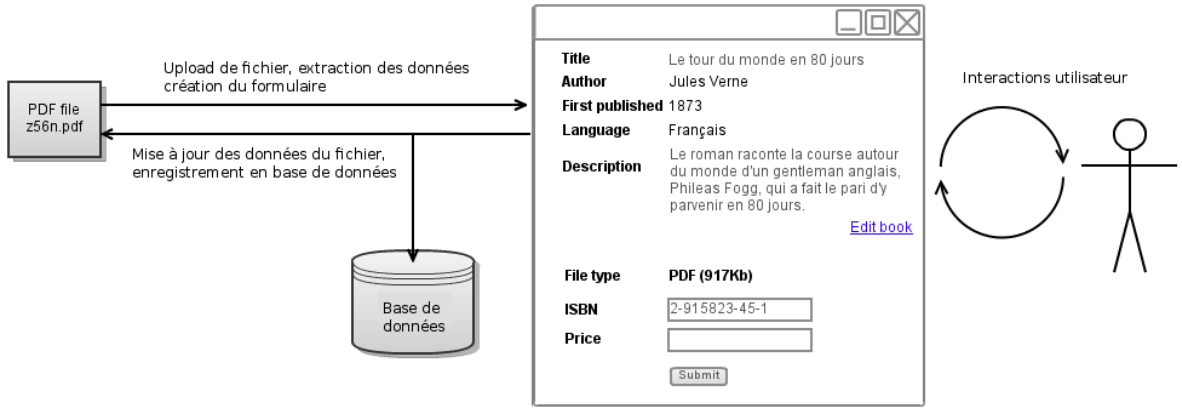



FIGURE 6.2 – Processus d'ajout d'un fichier pour un livre existant

effet, un fichier ePub définit son contenu à l'aide de données exprimées en XML au format OPF (Open Packaging Format) [OPF 07]. Ces données décrivent principalement les composants du fichier ePub et un ensemble de métadonnées concernant le livre numérique. Dans la norme ePub version 2, les métadonnées sont exprimées à l'aide des éléments du Dublin Core.

Lors de l'ajout d'un fichier, il serait donc aberrant de ne pas proposer à l'éditeur d'utiliser ces informations, charge à lui de les vérifier. L'éditeur se voit alors proposer un formulaire pré-rempli avec les informations que le système a pu extraire du fichier. L'éditeur peut alors vérifier et compléter les informations concernant le livre. Une fois ces informations validées, les différentes données sont enregistrées dans le système en utilisant le modèle de document défini. Enfin, pour assurer la cohérence entre les données de l'application et les métadonnées du fichier, les métadonnées du fichier peuvent être réinscrites dans celui-ci. Le processus décrit est illustré figure 6.1.

Lors de l'ajout, pour le même livre, d'un autre type de fichier, par exemple au format PDF, le processus sera quelque peu différent. En effet, pour l'application il ne s'agit pas ici de créer un

nouvel objet eBook, mais il s'agit d'ajouter une Manifestation à un objet eBook existant. Cette distinction doit bien évidemment être transparente pour l'éditeur. Dans ce cas, les informations *déjà présentes* dans le système sont utilisées. Les données extraites du fichier déposé servent à pré-remplir un formulaire qui ne concerne que les informations spécifiques au fichier PDF ajouté pour le livre existant. Là encore, une fois les informations validées par l'éditeur, les données sont mises à jour dans le fichier pour assurer la cohérence globale. Ce processus est illustré figure 6.2.

### 6.2.2 Généralisation du processus

Le processus présenté dans la section précédente est généralisé pour la gestion des fichiers qu'un utilisateur ajoute au système. En effet, Sydonie a vocation à gérer des documents, en particulier nous avons souligné les manques des systèmes existants quant à la gestion des fichiers déposés, le cas des images illustrant un contenu étant le plus courant. L'ajout de fichier dans une application gérée par Sydonie passe par l'étape d'extraction des métadonnées présentes dans le fichier. Le développeur de l'application est alors responsable de ce qu'il advient de ces données et de la façon dont elles sont intégrées ou non au système. Afin de faciliter ces opérations, Sydonie réalise les principales opérations, que nous allons détailler dans les parties suivantes.

## 6.3 Conteneur de métadonnées

Malgré l'utilisation possible de XMP pour intégrer des métadonnées dans les fichiers, la gestion des multiples formats doit être assurée afin de garantir la cohérence des métadonnées. Par exemple, dans une image les métadonnées seront exprimées en utilisant différents schémas : Exif, IPTC et/ou XMP. De plus, comme l'exemple de `lisez-moi.lu` le montre, des métadonnées identiques peuvent être insérées dans des types de fichiers différents. Enfin, les métadonnées doivent pouvoir être lues (extraites) et écrites (insérées) dans les fichiers.

Les métadonnées pour un document peuvent être générées de deux façons : à partir du modèle de l'application (création d'un objet à partir de rien), ou bien à partir de fichiers (cas de l'import d'un fichier pour créer un document par exemple). Le framework Sydonie utilise un conteneur de métadonnées comme pivot entre le modèle d'application et les métadonnées intégrées dans les fichiers. Le conteneur stocke les informations de façon indépendante des fichiers et du modèle de l'application. Des lecteurs/écrivains travaillent uniquement entre le conteneur et un ou plusieurs fichiers, ils sont donc complètement indépendants du modèle d'application. Cette architecture permet la réutilisabilité des programmes effectuant ces opérations. La figure 6.3 illustre les interactions entre le conteneur et divers types de fichiers.

Les imports/exports entre le conteneur et les fichiers sont gérés au niveau du framework. Si un format de fichier n'est pas géré, le développeur peut alors créer cette fonctionnalité pour son application, et éventuellement ensuite proposer son intégration dans le framework. A terme

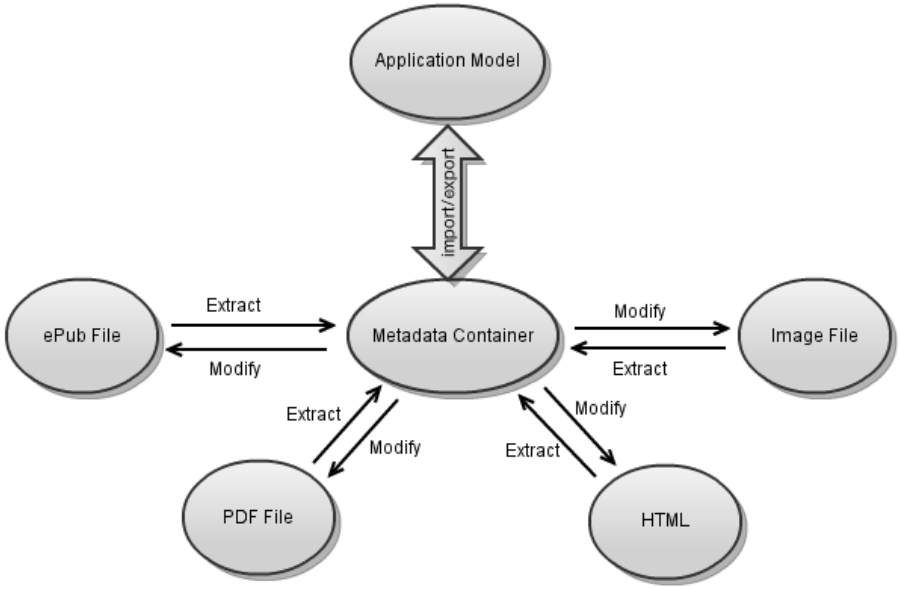



FIGURE 6.3 – Interactions entre le conteneur de métadonnées et divers types de fichiers

cependant, les formats de fichiers les plus courants seront gérés par le cœur de Sydonie. Notons que ces fonctions d'import/export ont vocation à utiliser des bibliothèques tierces pour interagir avec les fichiers. Par exemple, l'import/export dans les fichiers images est réalisé à l'aide de la bibliothèque ExifTool [65], utilisée entre autres par le site d'images Flickr.

Le conteneur décorrèle complètement la gestion de la lecture/écriture d'informations dans un fichier de la gestion des schémas de métadonnées utilisés. Certains schémas de métadonnées supportés par l'import/export sont définis par défaut dans le moteur de Sydonie. C'est le cas du Dublin Core, Exif, XMP. Cependant, lorsque des éléments d'autres schémas de métadonnées sont nécessaires, l'application peut spécifier les éléments qui seront gérés par les procédures d'import/export.

Le conteneur de métadonnées groupe les informations extraites par schéma utilisé, mais les informations à ce niveau sont complètement indépendantes du concept de document de Sydonie. La figure 6.4 donne un exemple de la structure des données pour un document PDF. Celui-ci contient des métadonnées natives au format PDF et des métadonnées incluses avec XMP. L'exemple suppose que les données XMP contiennent des informations du schéma Dublin Core et du schéma LOM (Learning Object Metadata) [HODGINS & DUVAL 02], dont l'objectif est la description de ressources éducatives. L'exemple illustré par la figure 6.4 montre de plus que des informations identiques peuvent être présentes dans plusieurs schémas.

L'étape suivante du processus interne à Sydonie consiste à établir les correspondances entre les métadonnées que le conteneur stocke et le modèle de document de l'application.

65. http://www.sno.phy.queensu.ca/~phil/exiftool/

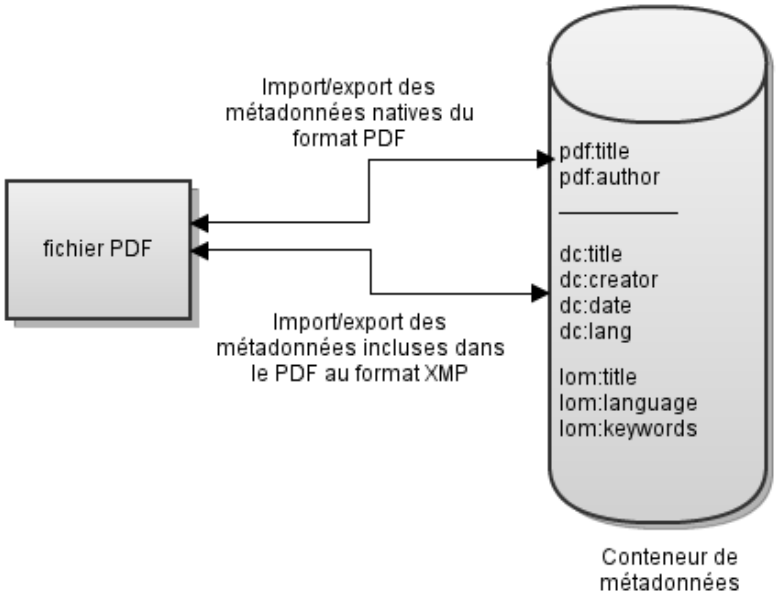



FIGURE 6.4 – Structure de données du conteneur de métadonnées

## 6.4 *Mappings* métadonnées/modèle de l'application

Dans le conteneur de métadonnées, les informations sont groupées par schémas de métadonnées. Il y a donc possibilité de redondances si plusieurs schémas sont utilisés, comme le montre l'exemple de la figure 6.4. De plus, la vue sur ces informations est *à plat*. La ressource décrite étant une Manifestation, les métadonnées exprimées dans le conteneur correspondent à la fusion des métadonnées prises en remontant les nœuds parents de cette Manifestation.

Le système doit donc effectuer une correspondance entre ces données et le modèle de l'application. Le *mapping* doit être défini par le développeur en fonction des données disponibles et du modèle de l'application. Il doit préciser à quel niveau dans l'arborescence du document (Work, Expression, Manifestation) les informations se situent. La figure 6.5 illustre cette étape.

Le développeur définit les procédures qui établissent les correspondances entre les données. Le processus peut alors appliquer des conversions à certaines valeurs lors du *mapping* (dates, coordonnées GPS par exemple). Dans le cas de redondances lors de l'extraction, comme dans l'exemple de la figure, le développeur définit le champ qui sera prioritaire. Le processus étant par la suite validé par un utilisateur, celui-ci pourra effectuer les corrections nécessaires. En revanche, l'écriture des métadonnées dans le fichier assure une cohérence des informations entre les divers schémas de métadonnées. Dans le cas de types de fichiers « classiques » comme les images par exemple, le framework fournit des procédures par défaut pour la création des documents de type Image. Ces procédures par défaut peuvent être modifiées dans l'application si besoin.

Il est de la responsabilité du développeur de vérifier la cohérence de l'ensemble du pro-

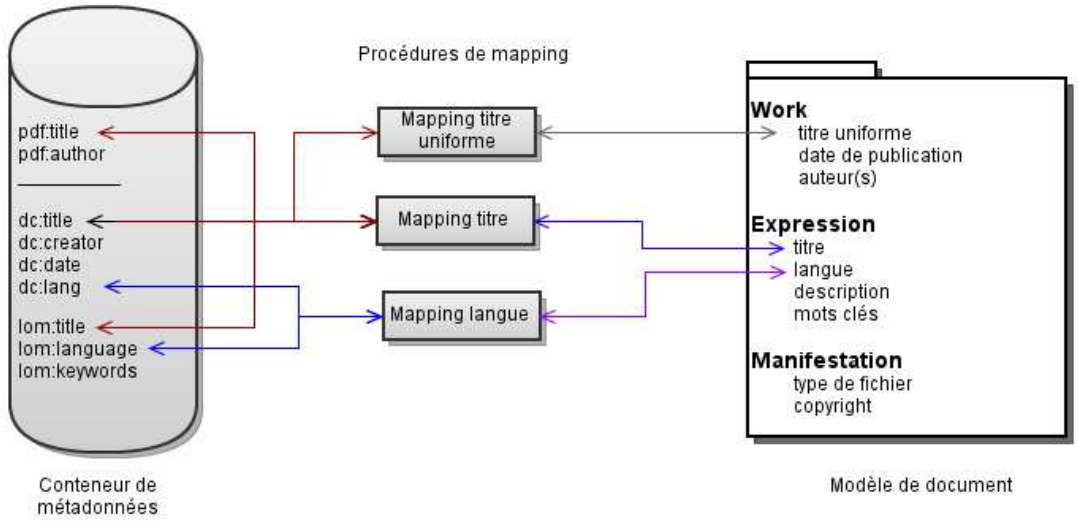



FIGURE 6.5 – Définitions des procédures de mapping

cessus et de créer les procédures de *mapping* en gardant à l'esprit les critères de qualité des données. En particulier, Bruce et Hillman [BRUCE & HILLMANN 04] indiquent que les éléments de métadonnées choisis doivent décrire les objets de façon complète mais réaliste. Le développeur doit donc proposer les éléments qui sont essentiels pour le public visé, mais sans inclure une masse d'informations dont la plupart ne sera jamais renseignée ou utilisée.

## 6.5 Processus complet, cohérence des données et IHM

Les interfaces de validation et de saisie des informations sont créées pour chaque application. Là encore, Sydonie peut fournir des interfaces par défaut comme c'est le cas pour les images par exemple. Les formulaires peuvent donc être pré-remplis avec les métadonnées extraites du fichier lui-même. L'utilisation du modèle de document de Sydonie assure la cohérence des données lors de l'ajout d'une nouvelle Expression ou Manifestation et évite la duplication. La figure 6.6 illustre, dans le cadre de l'application `lisez-moi.lu`, le processus de création d'un eBook lors de l'upload d'un fichier ePub.

Les étapes 1 et 7 montrent la couche d'abstraction implémentée avec le conteneur de métadonnées détaillé à la section 6.3. Ces étapes sont intégrées au moteur du framework Sydonie. Leur objectif est d'assurer, du point de vue informatique, l'accessibilité et la cohérence des données. Ces étapes sont réalisées par les lecteurs/écrivains qui gèrent les import/export dans divers types de fichiers. Ces programmes sont typiquement créés par l'équipe de Sydonie ou par des développeurs indépendants qui les mettent à disposition de la communauté. Ils sont destinés à être intégrés dans le moteur de Sydonie et à faire appel à des bibliothèques externes.

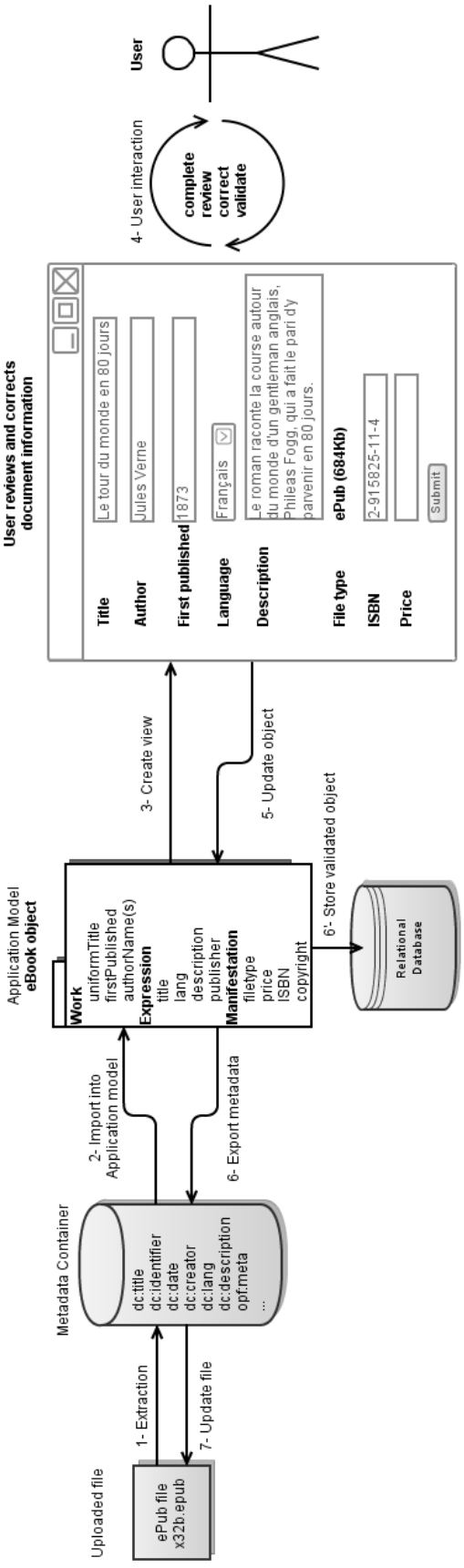



FIGURE 6.6 – Processus de création d'un eBook à partir d'un fichier ePub

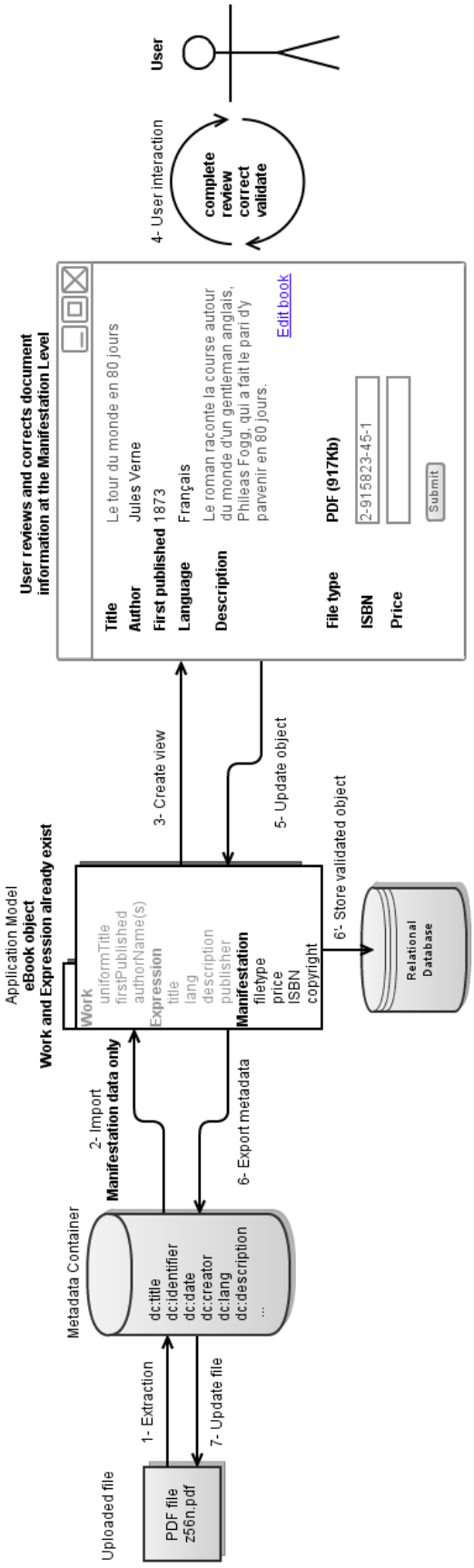



FIGURE 6.7 – Procédure d'ajout d'un PDF pour un eBook existant

Les étapes 2 et 6 montrent les procédures de *mapping* entre les données du conteneur et le modèle de document défini pour l'application. Cette étape est réalisée par le développeur web. il définit les correspondances à effectuer entre les données du conteneur et le modèle de son application.

Les étapes 3, 4 et 5 illustrent les interactions entre le modèle de l'application, les vues créées et l'utilisateur. Ces interactions sont réalisées par le développeur web avec le graphiste ou l'ergonome. La définition des processus de validation doit prendre en compte les besoins des utilisateurs de l'application. Les interactions avec l'utilisateur (étape 4) sont importantes pour la complétude et la précision des données et des métadonnées. Le producteur d'information (en l'occurrence un éditeur dans le cadre de `lisez-moi.lu`) est responsable de la validation des informations qui ont été extraites.

Lors de l'ajout d'un nouveau format pour un eBook existant, illustré figure 6.7, le modèle de document de Sydonie montre ses avantages, permettant ici d'utiliser les informations déjà présentes dans le système pour les niveaux Work et Expression. Seules les informations concernant le fichier ajouté, c'est-à-dire au niveau de la Manifestation, doivent alors être validées et complétées par l'éditeur.

Le travail du développeur est finalement facilité. Grâce à la couche d'abstraction fournie par le conteneur de métadonnées et les routines d'import/export existantes, il peut désormais se concentrer sur les procédures de correspondance entre les données du modèle de document à gérer et les métadonnées. L'utilisation du modèle de document de Sydonie permet d'assurer la cohérence des informations contenues dans divers fichiers. De plus, la gestion des interactions permet la construction de formulaires adaptés et la gestion des informations aux différents niveaux Work, Expression et Manifestation. La notion d'arbre FRBR et de schéma de métadonnées deviennent alors transparentes pour l'utilisateur final, qui en général n'est pas spécialiste de la gestion d'information. Les données saisies par l'utilisateur sont alors transformées en métadonnées grâce aux processus fournis par Sydonie.

## 6.6 Discussion

Les procédures et méthodes présentées dans ce chapitre fonctionnent bien pour les fichiers incluant des métadonnées, tels que les images, les fichiers PDF, ePub ou HTML. L'ajout d'autres types de fichiers gérés par Sydonie se fera au fil du temps et des contributions de divers projets.

La définition des correspondances implique pour l'instant que le développeur connaisse les éléments des schémas de métadonnées utilisés. Il sera donc nécessaire d'ajouter à des interfaces facilement utilisables pour la définition du mapping entre les données du modèle et les métadonnées.

Nous réfléchissons à l'utilisation d'une entité Manifestation spécifique au format XML/RDF,

qui exprimerait les métadonnées et pourrait répondre à des requêtes émanant de moteurs ou d'outils spécifiques.

D'autres travaux commencent pour expérimenter l'utilisation de méthodes de marquage de texte avec l'utilisation de la TEI. Ces travaux, menés par Pierre-Yves Buard [Dornier & Buard 08] dans le cadre d'une thèse, permettront d'améliorer les services proposés par le framework Sydonie.

Les développeurs qui ont uilisé ce système ont apprécié la couche d'abstraction apportée par le framework. Leur travail a alors pu se concentrer sur la gestion des interactions des utilisateurs avec l'application.

# Chapitre 7

# Architecture d'un framework

## Sommaire

Développer un framework est avant tout un travail d'équipe, où les heures de développement sont ponctuées de longues réunions de réflexion sur la définition des besoins et l'architecture du système.

Le projet Sydonie est développé au sein de "L'équipe Sydonie", qui est partie prenante de l'Équipe DLU (Document, Langues et Usages) au sein du Laboratoire GREYC. L'équipe Sydonie est constituée sous l'impulsion de Hervé Le Crosnier, notre directeur de thèse. Sa connaissance du monde des bibliothèques et de la gestion de documents a été déterminante pour la compréhension des FRBR et la modélisation de Sydonie.

Le projet Sydonie a obtenu le soutien de la Région Basse-Normandie dans le cadre du Contrat de Plan État-Région, ce qui nous a permis d'embaucher pour 18 mois un post-doctorant en la personne de Cyril Bazin. Celui-ci a largement contribué à la définition de l'architecture et à la programmation des outils. Il a su donner une impulsion majeure depuis son arrivée dans l'équipe en avril 2010.

Le framework Sydonie a été le support de trois projets d'étudiants de Master M2 "Ingénierie de l'Internet", réalisés entre mars et juin 2010 :

- Ingrid Moranne a développé une banque d'images. Le projet a permis des avancées dans le traitement des métadonnées des images par le framework ;
- Julien Chatelin a réalisé un site web pour construire et collectionner des jeux de cartes « Magic l'assemblée ». Ce projet a permis de tester le framework sur de gros volumes de données et d'améliorer des fonctionnalités Ajax du framework ;
- Benjamin Vallée a créé un blog à vocation multilingue. Débutant en programmation, Benjamin nous a aidé à identifier les points de blocages qu'un développeur non expert pourrait rencontrer. Le projet a donc permis l'amélioration de l'ergonomie de développement apportée par Sydonie.

Ces étudiants ont souvent été les alpha-testeurs du projet, dans un jeu de va-et-vient fort instructif entre leurs besoins et leur expérience et l'état d'avancement du framework. Nous parlerons ici d'eux globalement comme "les développeurs" ayant implémenté des sites avec Sydonie.

Enfin, Sydonie est aussi développé en collaboration avec l'entreprise d'édition multimédia C&F éditions[66]. Celle-ci a participé au travers de :

- Nicolas Taffin, graphiste et ergonome qui a proposé les exemples concrets d'utilisation en grandeur réelle de Sydonie, notamment le projet "Mémoire des catastrophes". Il a été particulièrement actif dans la définition de l'ergonomie de Polifile, application de création en ligne de livres numériques au format ePub. Enfin, il nous a laissé utiliser le framework CSS `<blank>` qu'il a créé, et qui est un des outils complémentaires de Sydonie ;
- Charline Enée, étudiante en contrat de professionnalisation en convention entre C&F

66. http://cfeditions.com

éditions et l'Université de Caen. Charline a été présente tout au long des deux dernières années et les applications qu'elle développait pour le compte de son employeur ont été les premières réalisations utilisant Sydonie. À ce titre les aller-retour et les réflexions communes ont permis d'éclairer les fonctionnalités nécessaires pour un framework orienté vers les applications d'édition sur le web. Charline a développé l'application Polifile, ainsi qu'une maquette de système de diffusion de livres numériques `lisez-moi.lu` et le site Mémoire des catastrophes.

Les chapitres précédents nous ont permis de présenter le modèle de document et les solutions apportées pour la gestion des métadonnées, qui sont implémentées dans le framework Sydonie. Ce chapitre présente l'architecture logicielle de Sydonie et les solutions apportées au problèmes soulevés au chapitre 4.

## 7.1 Développement d'applications

Nous avons insisté au chapitre 4 sur l'importance pour le développeur de pouvoir penser son modèle d'application sans contraintes *à priori*.

Le développeur doit pouvoir définir les objets selon les besoins de son application. L'intérêt de l'utilisation d'un framework est de fournir des outils pour gérer les fonctionnalités récurrentes dans le développement d'applications web, comme par exemple se connecter avec un identifiant et un mot de passe.

Dans ce cadre, après avoir défini ses besoins, le développeur peut regarder si le framework fournit déjà certaines fonctionnalités. Il y a alors trois possibilités. La première est le cas où le framework possède déjà la fonctionnalité nécessaire. Il suffit alors de la réutiliser, éventuellement en adaptant certains détails comme les affichages par exemple.

Deuxième possibilité, le framework dispose partiellement de cette fonctionnalité. Elle correspond en partie aux besoins de l'application mais il faudrait y ajouter des données et/ou des fonctionnalités. Dans ce cas, le développeur souhaitera la réutiliser pour son application et y ajouter les fonctionnalités dont il a besoin, afin d'éviter de tout refaire. Cela correspond au principes de dérivation en programmation objet.

Enfin, dernière possibilité, la fonctionnalité voulue n'existe pas dans le framework. Dans ce cas le développeur a toute latitude pour la créer en fonction de son modèle.

Les études des besoins des développeurs web ont montré qu'un certain nombre de fonctionnalités se retrouvent en standard dans les applications web. Nous avons listé à la section 4.1.2 les plus importantes : la persistance des données, la gestion des sessions, la gestion des saisies et formulaires et la gestion des permissions. Le framework doit donc proposer ces fonctionnalités de base. Nous y ajouterons les fonctionnalités liées aux documents que nous avons présentées dans les deux chapitres précédents. Cependant, nous venons de montrer que le comportement par

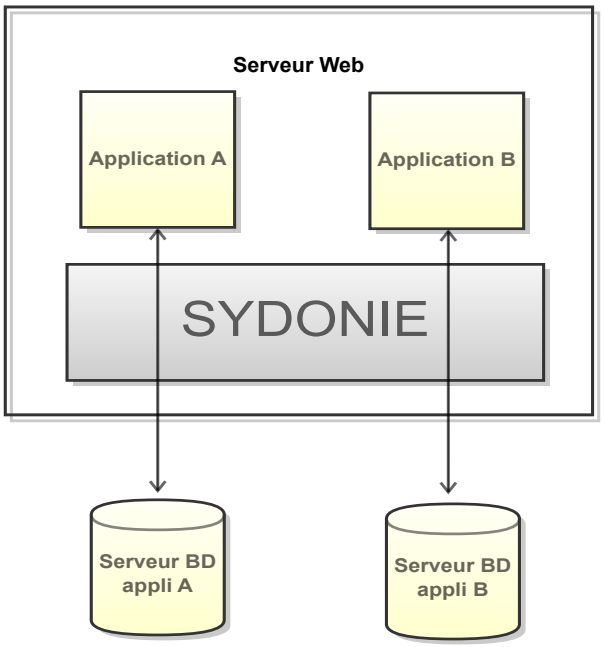



FIGURE 7.1 – Découplage moteur du framework et applications

défaut doit pouvoir être utilisé tel quel, adapté au modèle de l'application, ou encore développé complètement.

Sydonie utilise plusieurs mécanismes afin de garantir la flexibilité de développement mentionnée ci-dessus. Le premier aspect est le découplage du moteur du framework Sydonie et des applications hébergées sur un serveur. Le cœur de Sydonie peut ainsi être utilisé par plusieurs applications, ce qui évite de dupliquer le moteur du framework sur un serveur. Chaque application définit son point d'entrée sur le serveur, son serveur de base de données, et utilise le moteur de Sydonie pour les fonctionnalités que le développeur souhaite utiliser. Cela permet au développeur d'utiliser les composants du framework qui lui simplifient son travail, comme par exemple la gestion de la persistance des données, la gestion des templates, etc. Ce fonctionnement est illustré par la figure 7.1.

Sydonie gère la persistance des données, le gestion de templates pour les vues et formulaires, la gestion des permissions, etc. Nous détaillerons plus loin dans ce chapitre un certain nombre de ces outils qui n'ont pas, *à priori*, à être modifiées par le développeur mais qui doivent être configurés. Pour la création des fonctionnalités de l'application, le développement de celles-ci est réalisé au sein de ce que nous avons appelé *package*, que nous présentons à la section suivante.

## 7.2 Les packages dans Sydonie

### 7.2.1 Notion de package et héritage

#### Composition d'un package

Pour créer un nouveau type d'objet dans Sydonie, le développeur doit définir plusieurs éléments que l'on regroupe sous l'appellation de *package*. Il est composé de la définition des attributs de l'objet, un ensemble de vues, un ensemble d'actions, la configuration des formulaires, et enfin des labels servant pour l'interface et les messages.

Le type d'objet représenté par le package est modélisé par une classe dont le modèle est à définir. Les attributs de l'objet, qui constituent le modèle des données, sont définis de manière déclarative dans un fichier de configuration au format XML, comme introduit à la section 5.2.2. Nous présenterons plus en détail la configuration des objets à la section 7.2.2. Si l'on fait un parallèle avec le paradigme de la programmation orientée objet, les attributs définis dans la configuration correspondent aux propriétés d'une classe.

Les fonctionnalités de l'objet sont gérées par des actions. Celles-ci sont regroupées par types de fonctionnalités. Par exemple, les actions pour créer, enregistrer et modifier un objet correspondent à un groupe d'actions. En poursuivant le parallèle avec le paradigme objet, les actions correspondent aux méthodes de l'objet qui sont appelables *via* un URL, c'est-à-dire aux interactions avec le reste du monde. Le fonctionnement des actions sera détaillé à la section 7.2.3.

Les vues sur les objets que le framework doit produire sont gérées par un système de *templates*. Ils permettent de gérer les affichages des objets mais aussi de créer les formulaires de saisie ou de modification. Ces templates s'appuient sur des labels. Les labels sont utilisés pour la gestion multilingue de l'affichage des éléments d'interface comme le texte des boutons, des liens d'actions, des messages de confirmation ou d'erreur, etc. Nous présenterons le fonctionnement des templates et des labels aux sections 7.2.4 et 7.2.5. La gestion de la saisie de données et des formulaires est effectuée par le framework qui utilise pour cela la configuration des champs de formulaire définis pour l'objet utilisé. Leur fonctionnement sera expliqué à la section 7.2.6.

L'ensemble est contenu dans un répertoire spécifique au package, comme illustré par la figure 7.2. Lorsque l'URL `http://monsite.com./x32b/edit` est appelé, l'ensemble interagit de la façon suivante. Le système intercepte que l'objet d'identifiant `x32b` doit être utilisé. Il est nécessaire d'avoir une instance de l'objet demandé. Le fichier `config.xml` qui définit le modèle, permet de reconstituer l'objet en allant chercher les informations nécessaires en base de données. Le système peut alors déterminer à quelle action de cette instance le mot-clé `edit` correspond. Pour cela, le contrôleur du package détermine le groupe d'actions à utiliser grâce à la configuration contenue dans le fichier `actions.xml` et vérifie que l'action demandée peut être exécutée

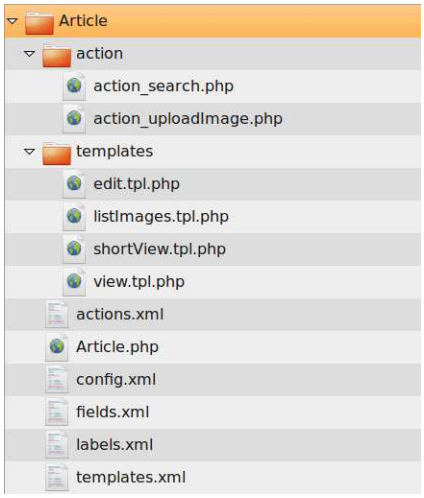



FIGURE 7.2 – Arborescence des fichiers composant le package Article

(gestion des permissions). Pour créer le formulaire, les instructions du template d'édition sont exécutées. Afin de créer les champs adéquats, le système regarde alors la configuration de chaque attribut, spécifiée dans le fichier `fields.xml`. Enfin, les labels stockés dans `labels.xml` permettent d'afficher les messages spécifiques au type d'objet utilisé. Nous détaillerons dans les sections suivantes les divers mécanismes mis en œuvre pour réaliser toutes ces opérations.

### Héritage de package

La création d'un package permet au développeur de créer les types d'objets dont l'application a besoin et les fonctionnalités associées. L'utilisation du framework doit cependant lui permettre de tirer profit des fonctionnalités déjà présentes au sein de celui-ci.

Afin de pouvoir utiliser des modèles et des fonctionnalités déjà définis, Sydonie permet d'utiliser un héritage entre les packages. Similaire à l'héritage « classique » du paradigme objet, il permet d'ajouter ou de redéfinir des données du modèle (héritage des attributs) et des actions (héritage des « méthodes »). Dans l'architecture de Sydonie, l'héritage de package va plus loin en permettant l'héritage de tous ses aspects : attributs, actions, vues (templates), configuration des formulaires, labels.

Sydonie définit un package de base (`SydonieEntity`) qui met en place les mécanismes qui permettent cet héritage. Tous les packages utilisés dans Sydonie dérivent de celui-ci. De la même façon, Sydonie définit un package `SydonieDocument` qui met en place l'architecture de documents inspirée des FRBR présentée au chapitre 5 et qui permet l'utilisation des entités Work, Expression et Manifestation. Tout package dont le modèle de données utilise cette structure de document dérive de `SydonieDocument`.

Enfin, le découplage entre le framework et l'application développée introduit une autre forme

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <class>
    <name>BlogComment</name>
    <extends>SydonieDocument</extends>
  </class>
  <attribute entityLevel="work"
             minOccur="1"
             maxOccur="1">
    <predicate>name</predicate>
    <objectClass>AttributeType_Text</objectClass>
  </attribute>
  <attribute entityLevel="work"
             minOccur="1"
             maxOccur="1">
    <predicate>email</predicate>
    <objectClass>AttributeType_Email</objectClass>
  </attribute>
  <attribute entityLevel="manifestation"
             minOccur="1"
             maxOccur="1" >
    <predicate>content</predicate>
    <objectClass>AttributeType_Text</objectClass>
  </attribute>
</configuration>
```

FIGURE 7.3 – Configuration du type de document BlogComment

d'héritage que nous avons appelée « double cascade » et que nous présentons à la section 7.3.

Les sections suivantes présentent plus en détail les notions introduites ici.

### 7.2.2 Définition du modèle de données

#### Configuration du modèle

Nous avons présenté section 5.2.2 la configuration d'un type de document, que nous reprenons ici. Le fichier de configuration d'un type d'objet détermine le modèle de données de celui-ci. Il permet de définir la liste des attributs qui compose l'objet. La figure 7.3 montre un exemple où sont spécifiés pour chacun des attributs :

- son nom (élément `predicate`) ;
- son type (élément `objectClass`) ;
- sa multiplicité (attributs `minOccur` et `maxOccur`) ;
- éventuellement des routines d'initialisation ;
- éventuellement des routines de validation.

Ces informations définissent les données d'un type d'objet quelconque. Si le type d'objet implémente le modèle de document présenté au chapitre 5, en dérivant du package qui implémente ce modèle (`SydonieDocument`), alors la configuration doit de plus préciser le niveau d'entité auquel chaque attribut est attaché (attribut `entityLevel`). Un développeur peut donc créer les types d'objets correspondant au modèle de son application.

### Types d'attributs

Nous avons introduit la notion de type d'attribut à la section 5.2.1. Le modèle de données d'un type d'objet est défini par des d'attributs. Les attributs ont pour valeur une instance d'objet `AttributeType`.

Les objets types d'attribut `AttributeType` peuvent être simples ou complexes. Par exemple, le type Text est simplement une donnée textuelle. En revanche, le type Address contient les éléments nécessaires modélisant une adresse postale (numéro, rue, ville, code postal).

Les objets `AttributeType` sont eux aussi implémentés sous la forme de package. Ils possèdent donc les fonctionnalités que nous décrivons dans cette section. En conséquence, si un objet de l'application nécessite un type d'attribut qui n'existe pas, le développeur peut le créer pour l'application.

### Attributs

Le modèle des données, géré au moyen de la configuration, utilise l'héritage de package : les attributs définis dans ce fichier s'ajoutent ou surchargent les attributs définis dans les packages parents.

Dans l'exemple de la figure 7.3, le type d'objet `BlogComment` dérive de `SydonieDocument`. Par conséquent, il aura les attributs de `SydonieDocument` (présentés section 5.2.2) plus les attributs définis dans sa configuration, soit finalement :

- au niveau Work :
  - hérités de `SydonieDocument` : uniformTitle, firstPublished ;
  - définis pour `BlogComment` : name ;
- au niveau Expression :
  - hérités de `SydonieDocument` : title, description, lang ;
  - (`BlogComment` n'a aucun attibut en plus pour l'Expression) ;
- au niveau Manifestation :
  - hérités de `SydonieDocument` : mimeType, fileSize. L'attribut content est redéfini par la configuration de `BlogComment` ;
  - définis pour `BlogComment` : content, qui ici redéfinit le type d'attribut content comme étant un attribut de type texte, alors que, par défaut, `SydonieDocument` le définit comme un attribut de type ressource.

Le modèle des types d'objets, exprimé de façon déclarative, rend simple pour le développeur la création du modèle de données de l'application. La flexibilité apportée par l'héritage de package et la possibilité de créer de nouveaux types d'attributs lui permet de répondre aux besoins de l'application.

### 7.2.3 Actions sur les objets

Les actions sur un type d'objet permettent de réaliser les fonctionnalités du package. Les actions sont regroupées par type de fonctionnalité. Chaque groupe d'actions est implémenté dans une classe, au sein de laquelle chaque action est réalisée par plusieurs méthodes. Par exemple l'action `save` d'enregistrement d'un objet appartient à la classe d'actions `manage`. Lors de l'exécution d'une action, un contexte d'exécution est transmis à la classe d'actions utilisée par le package.

#### Options d'une action

Pour chaque action, une méthode définit des options, dont une déclare les permissions nécessaires pour pouvoir exécuter l'action.

La première option définit si l'action est une action de classe, c'est-à-dire si elle nécessite la présence d'une instance d'objet. En programmation objet, cela pourrait correspondre à une méthode statique. Par exemple, l'action de création d'un objet doit pouvoir être exécutée sans instance. En revanche, pour exécuter l'action de modification, la classe d'actions doit obligatoirement avoir dans son contexte une instance de l'objet à modifier.

La deuxième option permet de définir les permissions nécessaires sur le type d'objet considéré pour pouvoir réaliser l'action à effectuer. Par exemple, pour pouvoir exécuter la modification d'une instance d'un objet, l'utilisateur doit avoir les droits d'écriture. Cette information est spécifiée dans les options. Cette option permet la vérification des permissions par le framework dans différents cas. Cette vérification peut être effectuée avant l'exécution de l'action. Elle peut aussi être réalisée pour l'affichage d'un lien vers une action. Ceci évite la création d'un lien vers une action que l'utilisateur ne peut pas effectuer. Nous expliquons le fonctionnement de la gestion des permissions à la section 7.5.3.

#### Contexte d'exécution d'une action

Toute classe d'actions dérive d'une classe abstraite qui permet de fournir un contexte d'exécution aux classes d'actions. Le contexte d'exécution fournit les informations suivantes :

- le type d'objet ou l'instance de l'objet sur laquelle l'action est exécutée ;
- les données du contexte global de l'exécution (typiquement les données GET ou POST de la requête HTTP) ;
- les options de l'action à exécuter.

Ce contexte permet de compléter les actions avec un mécanisme de *hook*. Par exemple, après l'enregistrement d'un objet lors de sa création, le développeur souhaite qu'un mail de confirmation soit envoyé. L'action d'enregistrement par défaut est fournie par le framework et n'a en rien besoin d'être modifiée, mais seulement complétée. À l'issue de l'action d'enregistrement et dans

le cas où l'objet a bien été enregistré, une méthode `onInstanceOk` est automatiquement appelée. Celle-ci faisant partie de la même classe que l'action d'enregistrement, elle possède donc le même contexte d'exécution, en particulier l'instance de l'objet à manipuler. Il suffit alors de (re)définir la méthode `onInstanceOk` qui effectuera l'envoi de mail. Grâce au contexte d'exécution, les informations nécessaires seront disponibles pour la méthode `onInstanceOk`. Le développeur peut ainsi programmer les instructions complémentaires sans avoir besoin de reprendre toute l'action d'enregistrement d'un objet. Ce type de mécanisme permet au développeur de gérer les relations à placer entre objets du système après l'enregistrement par exemple. Nous y reviendrons à la section 7.4.

#### Actions et dérivation de package

Les actions d'un type d'objet sont hérités de son parent. Par exemple, un package A héritant d'un package B possédera par défaut toutes les actions du package B. Comme dans le cas de l'héritage « classique », le développeur peut redéfinir des actions existantes ou en ajouter de nouvelles. Cette architecture permet de définir des actions qui sont utilisables par tous les types d'objets. C'est par exemple la cas avec le package de base du framework `SydonieEntity`. Celui-ci définit dans le moteur de Sydonie une classe d'actions pour créer, enregistrer, modifier, supprimer des objets, rendant ainsi ces fonctionnalités disponibles pour tout package qui en dérive.

Les groupes d'actions étant réalisés sous forme de classe, cela permet de plus l'existence d'actions « abstraites ».

#### Actions « abstraites »

L'utilisation d'une classe d'actions permet de définir des groupes d'actions « abstraites » qui sont définies dans le framework mais ne sont pas utilisables sans en implémenter certains composants. Par exemple, Sydonie définit une classe d'actions pour gérer l'upload de fichier. Cette classe définit des actions génériques pour réaliser l'affichage du bouton d'upload dans une page et le traitement du fichier déposé. Cette classe est inutilisable en l'état, et de plus il est impossible de la rendre générique pour toute application. En effet, chaque fonction d'upload de fichier aura ses contraintes : types(s) de fichier accepté, type d'objet à créer à partir de ce fichier, etc.

Afin de la rendre flexible, la classe d'actions upload définit différents points d'intervention au moyen de *hooks* comme expliqué plus haut. Dans ce cadre, un développeur peut donc créer une classe d'actions qui dérive de cette classe abstraite, afin de pouvoir utiliser les fonctionnalités communes à tout upload. Il lui suffit alors de définir les méthodes correspondant aux divers points d'intervention dont il a besoin. Dans le cas de l'upload, le développeur implémente, dans sa classe d'action, une méthode définissant le(s) type(s) de fichier acceptés et le type d'objet à créer à partir du fichier déposé. Il peut de plus y ajouter d'autres fonctionnalités si besoin. Par

exemple, si l'on souhaite pouvoir associer une image à un article, il suffit d'ajouter au package Article une classe d'actions pour cela. Cette classe d'actions dérive de la classe d'upload. Il suffit alors d'implémenter les méthodes pour définir les types de fichiers images acceptés, pour spécifier l'utilisation du fichier pour créer un type de document Image, et pour préciser qu'à l'issue du processus il faudra effectuer une association entre les instances de Article et de Image. Le reste du mécanisme d'upload est géré par la classe d'upload fournie par le framework.

Cette architecture permet au framework de proposer les briques de base pour implémenter les fonctionnalités dont les développeurs ont fréquemment besoin, tout en offrant une flexibilité dans les détails de ces fonctionnalités. Un développeur peut facilement ajouter une classe d'actions à un package existant. Il peut aussi créer des actions qui seront disponibles pour tous les types d'objets de Sydonie ou tous les types de documents par exemple. Dans la cadre des projets réalisés avec Sydonie, les développeurs ont apprécié la souplesse apportée par ce fonctionnement. Les différents points de contrôle permettent au développeur de réutiliser les fonctionnalités de base présentes dans le framework, pour se concentrer ainsi sur les détails spécifiques au modèle de l'application.

### 7.2.4 Templates : gestion des vues

#### Métiers et édition

La réalisation des interactions est une étape complexe du processus de développement d'une application web. De plus, les compétences nécessaires pour leur mise en œuvre sont multiples : ergonomie, graphisme, développement, etc. Les interactions correspondent aux divers affichages produits par l'application et les actions possibles de l'utilisateur au sein de ces affichages, sous la forme de saisie de données ou de choix à réaliser (menus, listes, etc.). Dans ce domaine, les web designers (graphistes et/ou ergonomes) doivent pouvoir intervenir sans avoir à toujours faire appel à un développeur. Il est donc important que le framework offre une souplesse de développement pour la création des vues, des masques de saisie d'informations et des interactions.

#### Templates

Les vues sur un objet sont réalisées par les différentes actions pour produire les affichages nécessaires à l'application. Les vues utilisent le langage HTML. Pour la gestion des vues, il serait possible d'utiliser un moteur de templates existant[67], mais cela implique dans la plupart des cas l'apprentissage d'un langage spécifique à ce moteur. Nous avons souligné à la section 4.3 qu'il est important, de notre point de vue, de ne pas avoir à apprendre un nouveau langage

67. Article présentant les principaux moteurs de templates pour PHP :
http://webification.com/best-php-template-engines

ou pseudo-langage pour prendre en main le framework Sydonie ou pour la réalisation des vues. Cet argument nous a paru particulièrement important puisqu'un graphiste pourra être amené à intervenir dans le cadre des templates, en plus du développeur web. Nous avons choisi de proposer un système de templates qui utilise uniquement le langage PHP en plus de HTML, puisque PHP est connu de la plupart des développeurs web.

Les templates sont définis pour chaque type de document. Un développeur peut créer les vues dont il a besoin pour le package qu'il utilise. Comme pour le modèle de données et les actions, les templates d'un package héritent de son parent. Un package pourra donc utiliser tous les templates de son parent, tout en y ajoutant les templates qu'il définit ou redéfinit. Selon le même principe que pour les modèles des types d'objets et des actions, cela permet de définir des vues par défaut pour tous les types d'objets.

#### Contexte et fonctions de raccourci

Les templates sont par défaut au format HTML et un développeur peut y insérer du code PHP à exécuter si besoin. Le gestionnaire de templates se charge de l'exécution des instructions PHP contenues dans le template. Afin de simplifier le code PHP à inclure, le template dispose d'un contexte d'exécution qui fournit un accès au gestionnaire de templates. Cela permet d'utiliser des méthodes pour accéder aux données de l'objet, afficher un label, créer un contrôle de formulaire, etc. Nous détaillerons les affichages des labels et la gestion des formulaires aux sections 7.2.5 et 7.2.6.

Sydonie fournit de plus un certain nombre de fonctions de raccourci pour afficher divers contenus dans les templates. Par exemple, des fonctions permettent de générer des liens vers des actions en ayant au préalable vérifié les droits de l'utilisation pour effectuer ces actions. En plus des fonctions proposées par le framework, le développeur peut créer les fonctions de raccourci dont l'application a besoin.

#### Templates et pages web

Les affichages produits par les templates sont des fragments HTML qui sont ensuite insérés dans la cadre plus global de la page web. Ce cadre global de la page web est réalisé par un ensemble de squelettes qui intègrent les contenus créés par les templates. Les squelettes de page et les feuilles de style sont réalisés par le web designer. Pour ce faire, il crée, dans un répertoire spécifique, un thème pour son application et utilise ses outils habituels de création web, incluant éventuellement dans son thème un framework CSS. Cette architecture est illustrée par la figure 7.4.

Comme pour les templates, le web designer peut insérer dans les squelettes des instructions faisant appel à des fonctions prédéfinies dans Sydonie ou créées spécifiquement pour l'application.

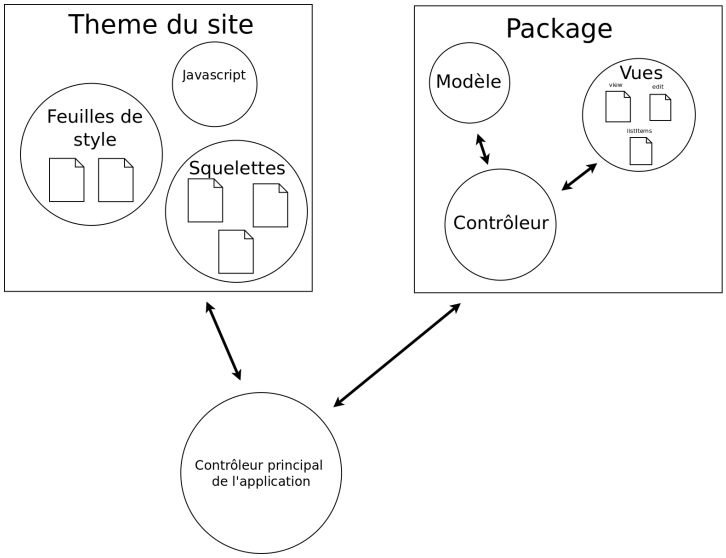



FIGURE 7.4 – Articulation entre les templates et les squelettes de l'application

### 7.2.5 Multilinguisme

#### Labels

Les textes correspondant aux éléments d'interface, appelés labels, sont les textes des liens utilisés pour la navigation ou les actions sur les objets, les textes des champs de formulaires (labels des contrôles, messages d'aide ou d'erreur), etc. Ils doivent être affichés dans la langue choisie par l'utilisateur. Chaque label est identifié par un code et les textes correspondants sont stockés dans un fichier au format TMX que nous avons présenté section 3.2.4. Un gestionnaire de labels se charge de parser les fichiers TMX des packages utilisés par l'application pour y extraire les textes dans la langue de l'interface que l'utilisateur a choisi (éventuellement avec une négociation de contenu). Les textes des labels sont alors disponibles pour le gestionnaire de templates qui peut demander l'affichage d'un label à partir de son code.

Les labels sont créés pour un package donné, selon les besoins de l'application. Une fois de plus cependant, l'héritage de package s'applique pour la définition des labels. Pour un package donné, les labels de son parent sont disponibles, mais il peut les redéfinir et/ou en définir de nouveaux.

Comme pour les types d'objets, les actions et les templates, cette architecture permet de définir des textes généraux utilisés par tous les packages. Des labels peuvent aussi être créés ou redéfinis pour un package particulier selon les besoins de l'application.

```
<?xml version="1.0" encoding="utf-8"?>
<fields>
  <field propName="adresse">
    <templateName>editAddress</templateName>
  </field>
  <field propName="description.text" uiMethod="Input_Textarea">
  </field>
  <field propName="content.data" uiMethod="Input_CkEditor">
    <option name="fieldCssClass">large</option>
  </field>
  <field propName="city.text" uiMethod="Input_CityAutocomplete">
  </field>
  <field propName="startDate.date" uiMethod="Input_CustomCalendar">
  </field>
</fields>
```

FIGURE 7.5 – Configuration de champs de formulaire

#### Templates et multilinguisme

Les labels permettent l'affichage de textes de l'interface dans plusieurs langues. Dans certains cas cependant, l'utilisation des labels n'est pas suffisante. C'est le cas lorsque les affichages ont besoin d'être adaptés à différents contextes culturels. Lorsque le template contient des textes « statiques » plus complexes que de simples labels, il est alors plus simple de réaliser plusieurs templates que « d'éclater » le texte en de multiples labels.

Pour répondre à ces problématiques, les templates peuvent être réalisés en plusieurs versions linguistiques. Les différentes versions sont déclarées dans le fichier de configuration des templates du package. En utilisant le principe de la négociation de contenu et en particulier le *language-negotiation* présenté section 5.2.3, le système utilisera le template le plus adapté à la situation.

### 7.2.6 Saisie de données et formulaires

La saisie de données et la gestion des formulaires est une des fonctionnalités fortement demandées par les développeurs web [ROSSON *et al.* 05A]. En effet, la création et la gestion des formulaires est souvent une tâche longue et répétitive, générant souvent de nombreuses erreurs (bugs, sécurité, etc.). Le framework doit proposer des solutions qui simplifient cette étape de développement, tout en lui garantissant une flexibilité de travail.

En particulier, l'agencement du formulaire doit pouvoir être réalisé par le développeur ou le graphiste. Nous avons choisi d'utiliser les templates pour cet agencement. Un formulaire est donc défini par un template qui utilise des méthodes spécifiques. Ces méthodes se chargent de créer les contrôles de formulaire définis par la configuration du package.

Pour créer la partie de formulaire correspondant à un attribut, c'est-à-dire un objet type d'attribut, un fichier de configuration (`fields.xml`) définit de manière déclarative les modes d'interaction à utiliser. Un exemple de configuration est proposé figure 7.5.

La configuration peut être de deux types. Rappelons que les types d'attributs (i.e. les ob-

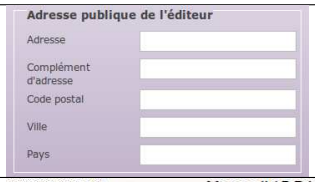
| Configuration | Template | Formulaire |
|---|---|---|
| `<field propName="adresse">`<br>`<templateName>editAdress</templateName>`<br>`</field>` | `<fieldset>`<br>`<legend><?php echo $ui->label('addresse');?>`<br>`</legend>`<br>`<?php echo $ui->input('addresse');?>`<br>`</fieldset>` |  |
| `<field propName="startDate.date"`<br>`uiMethod="Input_CustomCalendar">`<br>`</field>` | `<?php echo $ui->input('startDate'); ?>` |  |
| `<field propName="damageIntensity.text"`<br>`uiMethod="Input_SelectIntensity">`<br>`</field>` | `<?php echo $ui->input('damageIntensity'); ?>` |  |
| `<field propName="city.text"`<br>`uiMethod="Input_CityAutocomplete">`<br>`</field>` | `<?php echo $ui->statementInput('city',`<br>`'StatementCity'); ?>` |  |

TABLE 7.1 – Exemples de configuration, de template et de rendu

jets `AttributeType`) peuvent être simple (avec une seule donnée) ou complexes (composés de plusieurs informations).

Pour les types simples, le fichier de configuration indique le type d'interaction qui doit être utilisé. Par exemple, le champ pour le texte de l'attribut description sera un champ de type textarea. Dans l'exemple de la figure 7.5, le texte du contenu (donnée `data` de l'attribut `content`) sera présenté dans l'éditeur Wysiwyg CKeditor, modifiant ainsi la configuration par défaut du package.

Pour les types complexes, prenons l'exemple de l'attribut adresse qui correspond à un type d'attribut `Address`. Pour celui-ci, l'interaction est définie en précisant le template d'édition que l'objet `Address` doit utiliser. Rappelons que les types d'attributs étant eux-mêmes des packages, ils disposent des fonctionnalités des packages et donc définissent aussi leurs interactions via des formulaires. Dans le cas de l'exemple, le framework ou l'application définit le fragment de formulaire à créer pour un type d'attribut `Address`. Le template à utiliser est définit dans un template spécifié par le fichier de configuration. Dans le cas de l'exemple, le template `editAddress` sera utilisé, et produira le formulaire présenté dans le tableau 7.1.

Le tableau 7.1 montre divers types d'interactions, la façon dont ils sont configurés et comment le template de formulaire les insère.

Dans le cas d'attributs à cardinalité multiple, le framework propose par défaut de les ajouter

un à un en utilisant une boîte modale pour la saisie. Là encore, ce comportement peut être modifié si le développeur le souhaite. Les interactions utilisant les boîtes modales et Ajax seront présentées à la section 7.6.

Pour terminer le processus de saisie, le framework met en place des mécanismes pour la vérification des données saisies. Il assure la gestion des erreurs du formulaire : réaffichage du formulaire avec les messages d'erreur. Ces fonctionnalités sont fournies par des actions définies par le package `SydonieEntity`. Là encore, ces fonctionnalités peuvent être redéfinies si besoin.

## 7.3 Héritage en « double cascade »

L'architecture présentée à la section précédente permet au développeur de créer un package en définissant le modèle de données, les actions possibles sur l'objet, les affichages de l'objet et de l'interface, et enfin les interactions de saisie. L'héritage des packages offre une certaine flexibilité puisque le développeur choisit les éléments à (re)définir. Cependant, cela s'avère insuffisant. En effet, un développeur peut avoir besoin d'utiliser un package prédéfini dans le moteur de Sydonie et d'y apporter quelques modifications, sans pour autant vouloir créer un package dérivé. Le moteur du framework étant découplé de l'application, nous avons mis en place un mécanisme, que nous avons appelé héritage en double cascade, permettant de réaliser cela. Nous présentons celui-ci dans cette section.

### 7.3.1 Principe

L'architecture de packages permet la création de nouveaux types d'objets. Le développeur peut développer des packages pour créer des fonctionnalités n'existant pas encore dans le framework. Un des intérêts du framework est de pouvoir réutiliser des fonctionnalités existantes. Dans ce cadre, un développeur peut souhaiter simplement adapter un package existant sans pour autant créer un package dérivé. Par exemple, le framework possède un package `Article` pour gérer les articles dans un site. Un développeur qui souhaite utiliser ce type de document dans son application voudra certainement effectuer un certain nombre de « réglages ». Par exemple, il veut adapter le modèle de l'objet `Article`, en y ajoutant un attribut de « chapo », modifier le template d'affichage pour l'adapter à son site, modifier une fonctionnalité ou en ajouter une nouvelle. Pour permettre ces réglages fins et ajouter ainsi plus de souplesse de développement, nous avons créé un mécanisme de « double cascade » qui tire parti du découplage entre le moteur du framework et les applications.

Grâce à ce mécanisme, le développeur peut créer ce qu'il souhaite dans son application, il peut utiliser un package existant, le modifier et/ou le dériver.

### 7.3.2 Double cascade pour les fichiers

Pour profiter des fonctionnalités du moteur du framework et de l'application, les répertoires du framework et de l'application doivent avoir la même structure. Dans ce cas, lorsque le système a besoin d'un fichier donné pour un package, il commence par regarder si ce fichier existe dans les répertoires de l'application. Si c'est le cas alors il est utilisé, sinon le système prendra le fichier présent dans le cœur de Sydonie.

Ce mécanisme permet de redéfinir complètement un template donné. Par exemple, si le fichier de template `edit.tpl.php` de la classe Article est présent dans l'application, alors ce sera lui qui sera utilisé, et non le template `edit.tpl.php` présent dans le moteur de Sydonie.

De la même façon, une classe donnée pourra être redéfinie entièrement. Il suffit pour cela que le fichier définissant la classe soit présent dans l'application. Ce mécanisme s'ajoute aux possibilités d'héritage des classes présentés à la section précédente.

### 7.3.3 Application aux configurations

Le mécanisme de la double cascade permet de modifier les configurations d'un package. Cela inclut les configurations du modèle de document, des templates, des contrôles de formulaire, des labels et des actions. La double cascade donne la possibilité d'ajouter des informations de configuration mais aussi de modifier la configuration par défaut.

Prenons par exemple un type de document Article pour lequel on souhaite modifier le modèle de données, c'est-à-dire pour lequel le développeur veut redéfinir les attributs. Rappelons que le package `Article` dérive des packages `SydonieDocument` et `SydonieEntity`, comme illustré figure 7.6.

Le moteur de Sydonie et l'application étant découplés, chacun des packages parents de `Article` possède une configuration par défaut dans Sydonie et éventuellement une configuration spécifique dans l'application. Pour connaître la configuration d'un document de type Article pour l'application, Sydonie va alors reconstruire la liste des attributs d'un objet Article. Pour cela, on remonte la cascade de dérivation des packages en prenant à chaque fois en compte les configurations de l'application avant celles spécifiées dans Sydonie. Le fonctionnement est illustré par la figure 7.6.

Lors de l'utilisation d'un document de type Article par l'application, Sydonie va reconstruire la liste des attributs en commençant par les attributs définis dans la configuration de l'application pour le package `Article` (n$^o$ 1 sur la figure). Puis le framework prendra les attributs définis par défaut dans Sydonie pour le package `Article` (n$^o$ 2 sur la figure). Sydonie continue en remontant au package *parent* où le même processus se reproduit (n$^{os}$ 3 et 4 sur la figure), et ainsi de suite jusqu'à la « classe mère » de plus haut niveau.

Ce mécanisme permet de définir des attributs que l'on retrouvera pour tout un ensemble de

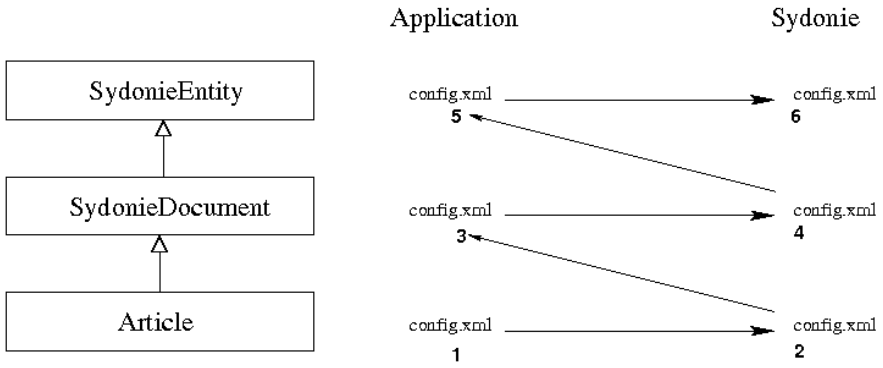



FIGURE 7.6 – *Héritage en cascade* pour la configuration d'un type de document

packages dérivés, mais aussi de modifier la nature d'un attribut au niveau de l'application.

Ce mécanisme est utilisé pour les affichages : templates, labels, formulaires. Pour chacun d'entre eux, la double cascade est mise en œuvre.

Dans le cadre de la configuration des templates, ce processus est combiné au mécanisme utilisé pour les fichiers. Par exemple, si le fichier de template `view.tpl.php` doit être utilisé pour le package `Article`, alors le système va d'abord regarder si ce fichier existe dans le package `Article` au sein de l'application. Si le fichier n'existe pas, le système cherche dans le cœur du framework. Tant que le fichier n'est pas trouvé, le système remonte ensuite la liste des packages parents tout en regardant à chaque fois dans l'application en premier. Le développeur bénéficie donc des templates définis par défaut dans Sydonie et peut, là encore de façon simple, définir ses propres templates sur les objets de l'application.

Le processus est similaire pour la définition des labels, et la configuration des formulaires.

### 7.3.4 Conclusion

Nous avons présenté dans cette section et la précédente les solutions qui répondent à certaines attentes que nous avions formulées au chapitre 4 et à la section 7.1.

Le développeur d'applications web a toute liberté pour définir le modèle de son application. Les mécanismes proposés par le framework lui permettent de bénéficier d'une flexibilité de développement. Il peut en effet créer les packages nécessaires pour les besoins de son application. Il peut aussi adapter un package existant en modifiant son modèle de données, ses actions, ses vues, ses labels et ses interactions de saisie.

Cette architecture a été appréciée des développeurs. Ils ont alors pu se concentrer sur les spécificités de l'application à développer.

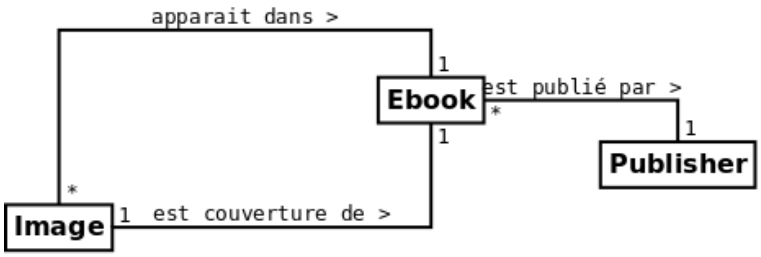



FIGURE 7.7 – Modèle de l'application Polifile

## 7.4 Relations entre objets

Nous avons introduit à la section 5.3 le modèle de relations entre objets utilisé par Sydonie. Ces relations sous la forme d'objets Statement peuvent être représentées par des triplets (sujet, prédicat, objet) similaires aux triplets RDF. Dans Sydonie, le sujet et l'objet sont des objets qui héritent de `SydonieEntity` (classe de base du framework) et le prédicat est une chaîne de caractères. Dans le framework, les relations entre objets correspondent aux relations modélisées en UML par l'association et l'agrégation.

Sydonie gère les statements de façon symétrique : si un statement existe entre un objet A et un objet B, alors le statement symétrique est placé entre l'objet B et l'objet A. Par exemple, pour représenter le fait que le document d'identifiant `x32b` est créé par l'utilisateur `jml`, deux statements réciproques sont ajoutés dans la base des statements :

| Sujet | Prédicat | Objet |
|---|---|---|
| `jml` | `OWNS` | `x32b` |
| `x32b` | `IS_OWNED_BY` | `jml` |

L'existence d'un statement et de son symétrique est une aide au développeur qui peut ainsi travailler indifféremment avec l'un des deux prédicats, et facilite ainsi les recherches d'associations entre objets.

Sydonie gère des statements internes, par exemple pour associer un document et son propriétaire comme ci-dessus. Ces statements pourront ensuite servir au framework pour déterminer les droits d'un utilisateur pour effectuer une action sur un objet. Un gestionnaire de Statement centralise la prise en charge de diverses opérations, par exemple pour déterminer l'existence d'un statement donné, lister les objets ayant un sujet et un prédicat donnés, etc.

Un développeur peut de plus définir les statements selon le modèle de l'application. Par exemple, dans le cadre de Polifile, une forme simplifiée du modèle de l'application est résumé par la figure 7.7. Le type d'objet Ebook représente le livre numérique. Il peut contenir des images et avoir une image de couverture. Un livre est publié par une maison d'édition, représentée par l'objet Publisher.

L'application définit alors les relations suivantes entre les objets :

- relations `IS_PUBLISHED_BY` et `PUBLISHES` entre les objets Ebook et Publisher pour exprimer le fait qu'un livre appartient à une maison d'édition ;
- relations `IS_IMAGE_OF` et `HAS_IMAGE` entre les objets Ebook et Image pour exprimer l'appartenance d'une image à un livre ;
- relations `IS_COVER_OF` et `HAS_COVER` entre les objets Ebook et Image pour exprimer l'utilisation d'une image pour la couverture du livre numérique.

Aux relations présentées et exprimées ci-dessus s'ajoutent les relations pour indiquer l'appartenance d'un utilisateur à une maison d'édition donnée. De plus, le framework définit en interne des relations entre l'utilisateur et les objets qu'il crée (Ebook, Image, etc.). Ces relations sont aussi utilisées par l'application. Elles serviront ici par exemple à vérifier qu'un utilisateur accède seulement à un Ebook de la maison d'édition auquel il appartient. Les relations entre les objets sont au cœur du système de permissions, que nous présentons à la section 7.5.3.

La gestion des relations donne la possibilité au développeur de gérer en toute liberté le modèle de son application. À terme, le framework pourrait proposer au développeur de déclarer les contraintes d'intégrité fonctionnelles, notamment les cardinalités.

Dans cette section, nous avons décrit les mécanismes mis en œuvre pour permettre au développeur d'appliquer son modèle d'application, en mettant l'accent sur les packages. En articulant le développement d'application autour des packages et des relations entre objets, le framework permet au développeur de concentrer sa conception sur le modèle de l'application à créer. La flexibilité ainsi obtenue simplifie le développement complet d'une application. L'existence de types d'objets et de fonctionnalités prédéfinies simplifie et accélère le processus de développement.

## 7.5 Gestion des ressources

### 7.5.1 Stockage des données

Dans le cadre du développement d'applications web, le stockage des données est un aspect incontournable. Il prend la forme de deux problématiques : le stockage de fichiers et la persistance des données dans une base de données. Cette section présente les solutions proposées par Sydonie.

#### Stockage des fichiers

En ce qui concerne le stockage de fichiers, l'avènement des web services et du cloud implique que les fichiers ne soient plus nécessairement stockés sur le serveur web de l'application, mais sur un ou des serveur(s) tiers. Il existe de nombreux services de stockage de fichiers, le plus connu

étant probablement la plate-forme Amazon S3 [68]. Amazon S3 permet à une application web de stocker des données en profitant de l'infrastructure des serveurs de Amazon. Par exemple, pour éviter de stocker de gros fichiers sur son serveur, une application peut choisir d'enregistrer les fichiers vidéos sur une plate-forme de stockage.

Lors de l'utilisation d'objets associés à des fichiers, le framework doit pouvoir gérer, de façon transparente pour le développeur, le moyen utilisé pour le stockage des fichiers. Par défaut, Sydonie enregistre les fichiers qu'il gère dans un répertoire spécifique. Cependant, afin de permettre l'utilisation de plusieurs emplacements de stockage, locaux ou distants, Sydonie implémente un type de données appelé Resource. Lorsqu'un objet du système utilise un fichier, cette utilisation se fait au travers de l'objet Resource.

Par exemple, pour le type de document Image, le contenu de la Manifestation est le fichier contenant les données de l'image. La configuration du package Image indique donc que l'attribut `content` est un type d'attribut de type Resource. L'objet Resource gère alors le fichier correspondant à l'image en faisant abstraction de la façon dont il est enregistré. Le développeur aura donc seulement à définir les paramètres de stockage pour les fichiers, et le reste de son développement se fera sans avoir à gérer ces aspects.

### Persistance des données

La structure de données interne aux objets du framework Sydonie est basée sur les principes de RDF, au moyen des attributs et des types d'attributs présentés aux sections 5.2.2 et 7.2.2. Nous avons montré que cette approche offre une flexibilité de développement pour créer des applications. En ce qui concerne la persistance des données, nous avons montré au chapitre 4 certains inconvénients des solutions adoptées par les CMS. En effet, la structure de base de données d'un CMS est déterminée par l'installation du système. Elle évolue en fonction des ajouts réalisés par les *plugins* installés. Mais, en général, le développeur n'intervient pas directement sur la structure de la base de données utilisée. Nous avons montré que cela implique, pour le développeur, de se « plier » au schéma existant pour la création de son application. En revanche, le CMS se charge de la persistance des données en gérant les interactions avec la base. Nous avons aussi présenté les frameworks comme offrant plus de liberté dans ce domaine. Avec les frameworks généralistes, il appartient cependant au développeur de concevoir le schéma de base de données à utiliser et d'écrire les requêtes qui permettront d'interroger celle-ci.

Dans le cadre du framework Sydonie, nous avons voulu profiter des avantages des deux solutions : gestion la persistance des données au sein du framework et liberté pour la définition du modèle d'application. Pour compléter le modèle de développement présenté dans ce chapitre, la structure de la base de données des applications utilisant Sydonie est alors générique. Les

68. Amazon Simple Storage Service (Amazon S3) http://aws.amazon.com/s3/

données des divers objets sont éclatées dans de multiples tables. Le système de gestion interne effectue la reconstruction des instances dans Sydonie.

Pour le développeur, ce système a des avantages :

- l'abstraction de données lui permet de travailler en se concentrant sur le modèle objet ;
- il n'a pas à écrire de requêtes SQL puisque celles-ci sont gérées par le framework, ce qui accélère le développement ;
- la base de données peut s'adapter aux évolutions de l'application.

L'abstraction des données est alors complète, le développeur n'ayant pas à gérer les accès à la base de données, que ce soit pour lire, enregistrer, rechercher, etc. De plus, il n'est pas nécessaire de modifier la base pour créer des objets adaptés à l'application. De nouveau, le développeur web peut rapidement développer une application et les fonctionnalités de Sydonie lui permettent de gérer la persistance des données sans avoir besoin de rédiger les requêtes SQL nécessaires.

### 7.5.2 Registre et URLs

L'utilisation des URIs pour identifier les objets du système reste à ajouter à Sydonie. Cependant, les briques nécessaires à cette mise en place sont présentes grâce au registre des objets et au parseur d'URL que nous présentons dans cette section.

#### Registre des objets

Nous avons présenté à la section précédente comment les informations sur les objets sont éclatées dans les diverses tables de la base de données.

Pour permettre la réification d'un objet, Sydonie gère un registre des identifiants des objets présents dans l'application. Le registre contient deux informations : l'identifiant de l'objet et son type. À partir de l'identifiant d'un objet, le système peut alors déterminer le type d'objet à instancier et réaliser sa réification en utilisant la configuration du package correspondant.

Ce choix implique qu'il ne peut pas exister, dans l'application, deux objets ayant le même identifiant. De plus, le système peut alors, à partir d'un identifiant, déterminer le type d'objet correspondant.

#### Traitement des URLs

Afin de mettre en œuvre les principes des Cool URIs [Sauermann *et al.* 08], un parseur d'URL a été mis en place dans Sydonie. Celui-ci a pour mission de décomposer l'URL demandée pour en extraire les informations nécessaires à l'application. L'utilisation du registre permet au parseur d'URL d'identifier rapidement l'objet demandé par la requête. Par exemple, le document d'identifiant `x32b` sera disponible à l'adresse `http://monsite.com/x32b/`. L'unicité de l'identifiant permet cet URL simple, qui par défaut affichera une vue de l'objet correspondant.

Si aucun identifiant ne figure dans l'URL, alors l'action à réaliser ne porte pas sur une instance particulière d'objet. L'application définit dans ses fichiers de configuration une liste de mots-clés qui peuvent figurer dans les URLs.

Pour éviter les URLs opaques du type `http://monsite.com/x32b/`, les rendre compréhensibles et faciliter leur échange sur la nappe de restaurant, un système d'alias est mis en place dans Sydonie. Un objet peut être associé à un alias, permettant alors de choisir l'URL qui servira d'accès à l'objet en question. Par exemple, l'article *Stop this crisis !* utilisé dans le chapitre 5 pourrait avoir l'alias `stop_this_crisis`, rendant l'article accessible avec l'URL `http://monsite.com/stop_this_crisis` qui est plus explicite qu'avec l'identifiant.

Le choix de l'alias peut être fait automatiquement par le système ou être délégué à l'utilisateur. Sydonie propose au développeur web les outils pour gérer les alias et l'interface utilisateur en fonction des besoins de son application. Sydonie utilise en interne un gestionnaire d'alias qui centralise les opérations sur les alias.

Il reste peu de choses à faire pour mettre complètement en application les principes des Cool URIS [Sauermann *et al.* 08] et identifier les objets d'une application par un URI. En particulier, la représentation en XML/RDF d'un objet, mentionnée à la section 6.6 doit être réalisée en standard dans le framework, ce qui permettra la négociation de contenu pour servir une représentation en XML/RDF d'un objet donné.

### 7.5.3 Gestion des permissions

#### Introduction

Nous avons discuté dans la section 4.2.2 des modèles de gestion des permissions dans les systèmes d'information et indiqué les inconvénients des modèles présentés. Pour mémoire, nous voulions imaginer un système souple, que le développeur peut facilement exprimer, et qui, de plus, lui évite l'exemple des innombrables cases à cocher.

Nous sommes partis d'une constatation simple : les règles de permissions sont d'abord énoncées en langage naturel. Par exemple, « seuls les modérateurs peuvent publier un texte déposé sur le site » ou « l'internaute authentifié peut déposer un commentaire ». Le principe de notre solution est de traduire ces règles en formules logiques entre les objets du système. Le framework peut alors évaluer ces formules logiques pour déterminer si l'accès est autorisé ou non. Notre solution propose un moyen de traduire une formule logique en fragments de requêtes SQL. Reprenant [Barker & Rosenthal 01] et [Stonebraker 75], ce fragment peut être ajouté au sein d'une requête. Cette méthode permet de limiter les résultats aux éléments pour lesquels l'utilisateur a les droits suffisants, par exemple pour n'afficher à un utilisateur que les objets auxquels il a le droit d'accès.

Nous avons présenté à la section 7.4 les relations entre objets du système sous la forme de

triplets (sujet, prédicat, objet), représentés dans Sydonie par les objets Statement. Ces relations sont à la base de la gestion des permissions dans Sydonie.

### Exemple

Prenons l'exemple de l'application Polifile, en le simplifiant quelque peu. Le système aura fréquemment à répondre à la question « Cet utilisateur peut-il modifier ce livre ? ». Considérons la politique de permissions suivante :

- l'administrateur peut tout faire ;
- un utilisateur peut lire et écrire un livre dont il est un contributeur ;
- le directeur d'une maison d'édition peut modifier tout livre de cette maison d'édition.

Nous allons regarder comment le système va pouvoir vérifier si un utilisateur peut modifier un livre donné.

Les relations entre les objets d'une application sont exprimés au travers de Statements, présentés section 7.4, sous la forme de triplets. Dans le cadre de Polifile, les relations entre les utilisateurs et les divers objets sont les suivantes :

- un triplet (*User*, « est membre de », *Publisher*) exprime l'appartenance d'un utilisateur à une maison d'édition ;
- un triplet (*User*, « est contributeur du », *Ebook*) exprime qu'un utilisateur est un contributeur d'un livre donné ;
- un triplet (*User*, « est directeur de », *Publisher*) exprime qu'un utilisateur est un directeur d'une maison d'édition ;
- un triplet (*User*, « appartient à », *Admin group*) exprime qu'un utilisateur appartient au groupe des administrateurs.

### Formules logiques et configuration

Rappelons que nous souhaitons répondre à la question « Cet utilisateur peut-il modifier ce livre ? ». Avec les règles énoncées ci-dessus, il faut donc vérifier si l'utilisateur est un administrateur, s'il est un contributeur du livre ou s'il est un directeur de la maison d'édition publiant ce livre. En utilisant deux entités « *User* » et « *Ebook* », cette condition peut être exprimée avec la formule logique suivante :

$$
\begin{aligned}
&\text{peutEcrire}(User, Ebook) \leftarrow \\
&\text{exists}(User, \text{'est membre du groupe'}, \text{admin group}) \\
\vee\ &\text{exists}(User, \text{'est contributeur de'}, Ebook) \\
\vee\ &\big(\exists X |\ \text{exists}(User, \text{'est directeur de'}, X) \\
&\wedge\ \text{exists}(X, \text{'possède'}, Ebook)\big)
\end{aligned}
$$

où la primitive $exists(a, p, b)$ renvoie VRAI lorsque un Statement de prédicat $p$ existe entre les deux objets $a$ et $b$. Il suffit de remplacer $\ll User \gg$ par l'identité de l'utilisateur et $\ll Ebook \gg$ par l'instance du livre pour vérifier si la formule est satisfaite ou non.

Afin d'obtenir des propositions indépendantes, la formule ci-dessus est transformée en :

$$\begin{aligned} & \text{peutEcrire}(User, Ebook) \leftarrow \\ & \text{exists}(User, \text{'est membre du groupe'}, \text{admin group}) \\ \vee\ & \text{exists}(User, \text{'est contributeur de'}, Ebook) \\ \vee\ & \text{existsPath}(User, \text{'est directeur de'}, \text{'possède'}, Ebook) \end{aligned}$$

où la primitive $existsPath$ est définie par :

$$\begin{aligned} & \text{existPath}(Sujet, Predicat1, Predicat2, Objet) \leftarrow \\ & \exists X \mid \text{exists}(Sujet, Predicat1, X) \\ & \wedge\ \text{exists}(X, Predicat2, Objet) \end{aligned}$$

Le framework gère la traduction des formules logiques en fragments de requêtes SQL, leur ajout aux requêtes SQL de base qui servent pour obtenir une instance, lister des objets, etc., et leur interprétation.

Cet aspect est complètement transparent pour le développeur qui ne doit donc que définir la politique de permissions *via* des formules logiques. Il est nécessaire de décrire plusieurs formules en fonction des actions (lecture, écriture, suppression par exemple). Ces formules sont stockées dans la configuration de l'application et peuvent être mises à jour à tout moment. La vérification des droits d'accès d'un utilisateur sur un objet du système est ensuite complètement gérée par Sydonie.

#### Permissions sur les actions

Cette gestion de permissions est utilisée par le système pour générer les liens d'actions sur des objets. Par exemple, lors de l'affichage d'une liste d'objets avec des liens Voir, Modifier, Supprimer, les fonctions créant les liens effectueront la vérification des droits. Par exemple, le lien Modifier n'apparaîtra que si l'utilisateur a effectivement le droit de réaliser cette action.

De même, un utilisateur ne verra s'afficher que les éléments qu'il a le droit de voir. Grâce à la gestion des fragments de requêtes SQL gérées par le système de gestion des permissions de Sydonie, cet aspect est lui aussi transparent pour le développeur.

#### Application aux modèles de permissions

Le système présenté ici permet d'appliquer divers modèle de permissions présentés à la section 4.2. Les modèles de permissions de type RBAC, TMAC et ACLs peuvent être émulés

par le système proposé.

Pour RBAC, il suffit de créer un nouveau type d'objet « *Role* » et d'ajouter des Statements entre les utilisateurs et les rôles qu'ils possèdent. Dans le cadre de Polifile, la formule vérifiant les permissions serait de la forme :

$$\begin{aligned} & \text{peutEcrire}(User, Ebook) \leftarrow \\ & \text{exists}(User, \text{'a pour rôle'}, \text{admin}) \\ \vee \quad & \ldots \end{aligned}$$

Le système doit alors vérifier la présence de Statement entre l'utilisateur et le rôle requis pour effectuer l'action demandée, par exemple sous la forme (*utilisateur*, « a pour rôle », *rôle admin*). Notons bien que Sydonie laisse toute latitude au développeur pour créer les rôles nécessaires à son application (administrateur, rédacteur, etc.).

*Team-Based Access Control* (TMAC) peut être réalisé en créant des Statement « *est membre de* » entre un utilisateur et son groupe. Dans l'exemple de Polifile, un utilisateur peut lire un livre s'il est membre de la maison d'édition qui possède le livre. Cela se traduit par la formule suivante :

$$\begin{aligned} & \text{peutLire}(User, Ebook) \leftarrow \\ & \text{existsPath}(User, \text{'est membre de'}, \text{'possède'}, Ebook) \\ \vee \quad & \ldots \end{aligned}$$

Notons bien que les Statements 'est membre de' et 'possède' doivent exister dans le modèle de l'application. Ils ne sont pas créés spécifiquement pour la gestion des permissions.

En ce qui concerne les ACLs, des permissions au cas par cas peuvent être ajoutés entre les objets du système et les utilisateurs pour leur permettre de réaliser certaines actions. Par exemple, un Statement d'*autorisation* du type « peut lire » doit être placé entre l'utilisateur et l'objet concernés. Le système assure la vérification avec la formule :

$$\begin{aligned} & \text{peutLire}(User, Ebook) \leftarrow \\ & \text{exists}(User, \text{'peut lire'}, Ebook) \\ \vee \quad & \ldots \end{aligned}$$

L'administrateur doit alors poser ou supprimer les Statements adéquats entre les objets et les utilisateurs. Le développeur doit concevoir une interface de gestion adaptée.

**Conclusion**

Le système de permissions présenté ici est implémenté dans le framework Sydonie. Il permet l'utilisation d'un ou plusieurs modèles de permissions au choix du développeur parmi les modèles courants utilisés pour les applications web. Il est extensible dans le sens où des primitives complémentaires peuvent être ajoutées au système (il suffit de pouvoir les transformer en requêtes SQL). Les fonctionnalités actuelles ont cependant suffit pour les applications développées avec Sydonie jusqu'à présent.

Les retours d'expérience sur ce système de la part des développeurs sont positifs, même si certains bémols sont à préciser, tout comme des demandes de fonctionnalités ou d'aide au développeur. En effet, l'écriture des formules logiques n'est pas à la portée de tout développeur, en particulier pour ceux n'ayant pas une formation informatique avancée. Pour pallier ce problème, nous pensons adapter le système pour proposer une déclaration des permissions sous la forme d'expressions « si, alors, sinon », le système traduisant ensuite ces propositions en formules logiques. Par exemple pour Polifile, cela se traduirait par « Si un utilisateur est contributeur du livre, alors il peut le modifier ». Cette formulation serait plus à la portée des développeurs.

## 7.6 Interactions Ajax

### 7.6.1 Introduction

Comme évoqué au chapitre 1, les pages web comportent désormais de nombreuses interactions Ajax[69]. Ajax permet de réaliser des interactions avec un serveur web par l'intermédiaire de Javascript. Cette technique permet de mettre à jour des zones de la page web sans recharger celle-ci complètement. Ajax est très utilisé pour les applications web, qui n'ont plus rien à envier aux logiciels interactifs graphiques.

Dans une application web, les boîtes modales permettent de rester sur la même page et de superposer à la page une zone qui peut servir à informer l'utilisateur ou à lui proposer des interactions (saisie de données, choix à effectuer par exemple). L'utilisation de boîtes modales est un mode d'interaction devenu lui-aussi courant. Les contenus des boîtes modales sont en général obtenus en utilisant Ajax.

Le développement d'interfaces faisant appel à des technologies Ajax est à la fois répétitif et spécifique :

- répétitif car les types de changements sont similaires : changer le contenu d'une zone, ouvrir une boîte modale, rafraîchir une zone tierce, etc. ;
- spécifique car la structure de la page et de l'interface changent à chaque projet. Les zones à mettre à jour et les URLs pour le chargement des données dépendent de l'application.

69. Terme apparu en 2005 : http://adaptivepath.com/ideas/ajax-new-approach-web-applications

Afin de proposer des solutions de développement aux web designers, nous avons observé les types d'interactions existants.

### 7.6.2 Types d'interactions Ajax

Nous avons identifié trois types d'interactions qui apparaissent fréquemment dans les interfaces utilisant Ajax :

- activer un lien ou un bouton pour modifier une zone de la page, comme illustré par la figure 7.8 ;
- déclencher la mise à jour d'une ou plusieurs zones suite à une interaction, comme illustré par la figure 7.9 ;
- ouvrir ou fermer une boîte modale. On retrouve les deux cas ci-dessus pour le contenu de la boîte modale. De plus la fermeture de la boîte modale peut déclencher la mise à jour de zone(s) de la page comme le montre la figure 7.10.

Ces interactions, simples pour l'utilisateur, sont en réalité un casse-tête pour le développeur. En effet, il est difficile d'en extraire des composants réutilisables pour diverses applications.

### 7.6.3 Exemples d'interactions

Prenons l'exemple de la figure 7.9, dans le cadre de l'application Polifile : lorsque le formulaire de création ou modification d'un chapitre est soumis, la zone contenant le formulaire est remplacée par la visualisation du chapitre, et la zone contenant la table des matières du livre est mise à jour.

Un cas en apparence aussi simple devient vite difficile à programmer. Il faut en effet :

1. effectuer une requête Ajax pour envoyer les données du formulaire ;
2. mettre à jour la zone du formulaire avec la réponse du serveur ;
3. pour chaque zone qui doit être mise à jour, lancer cette mise à jour avec l'URL correspondant ;
4. chaque partie doit alors effectuer une requête Ajax pour actualiser la zone concernée.

Un tel programme devrait donc stocker des informations spécifiques à l'application (les zones à mettre à jour et leur URL de mise à jour) mais aussi réaliser des actions génériques (requêtes Ajax, mise à jour de l'arbre DOM). Une première solution consiste à séparer ces éléments et associer au formulaire de l'exemple la liste des zones à mettre à jour et leur URL. Cette méthode devient vite limitée puisque la liste doit être mise à jour si l'interface générale de l'application change.

On voit sur cet exemple la difficulté d'extraire des composants réutilisables. Les programmes Javascript associés sont réalisés au cas par cas. De plus, pour réaliser ces tâches qui ont une importance capitale pour l'ergonomie, un graphiste devra faire appel à un développeur.

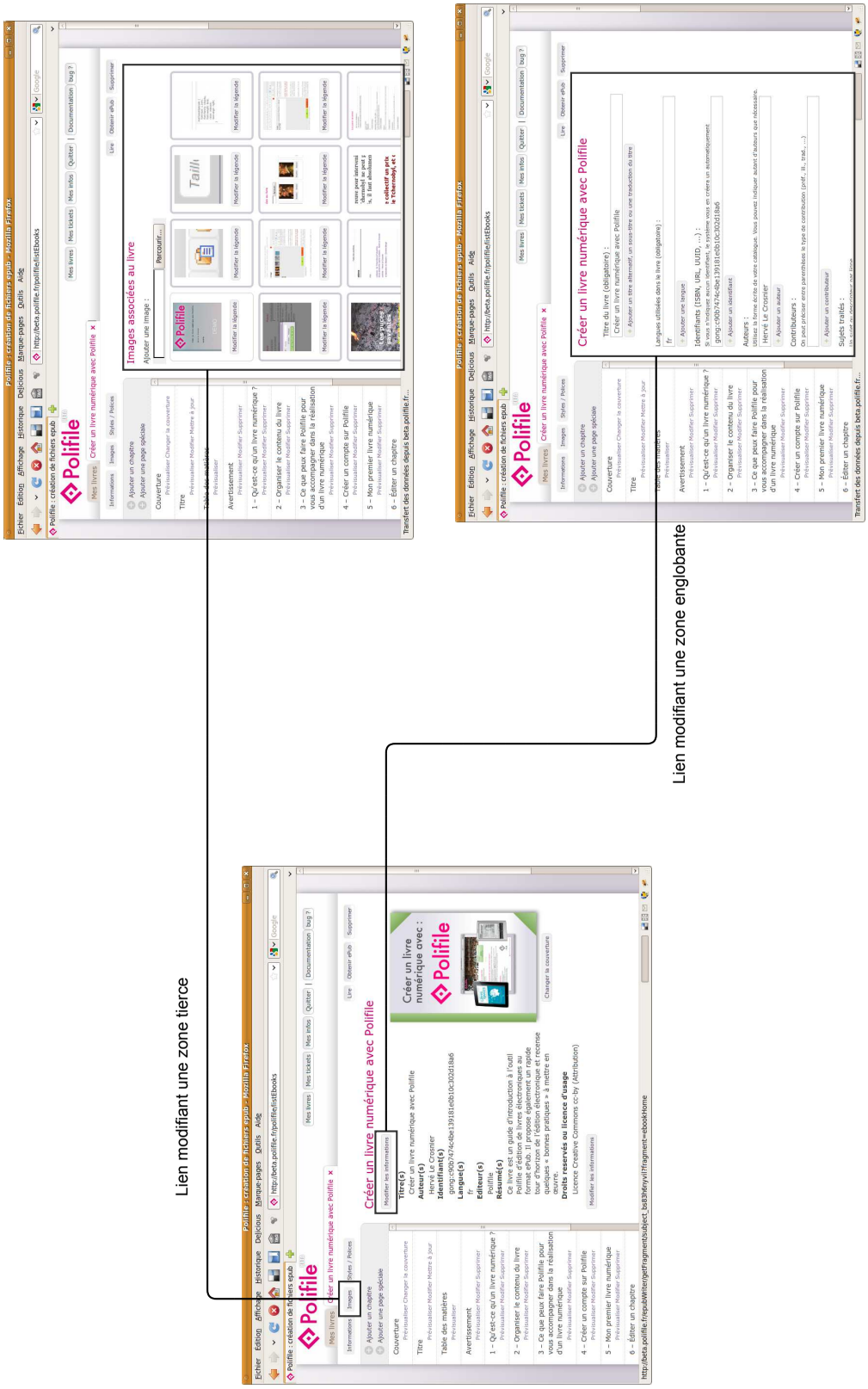



FIGURE 7.8 – Exemples de liens modifiant une zone de la page

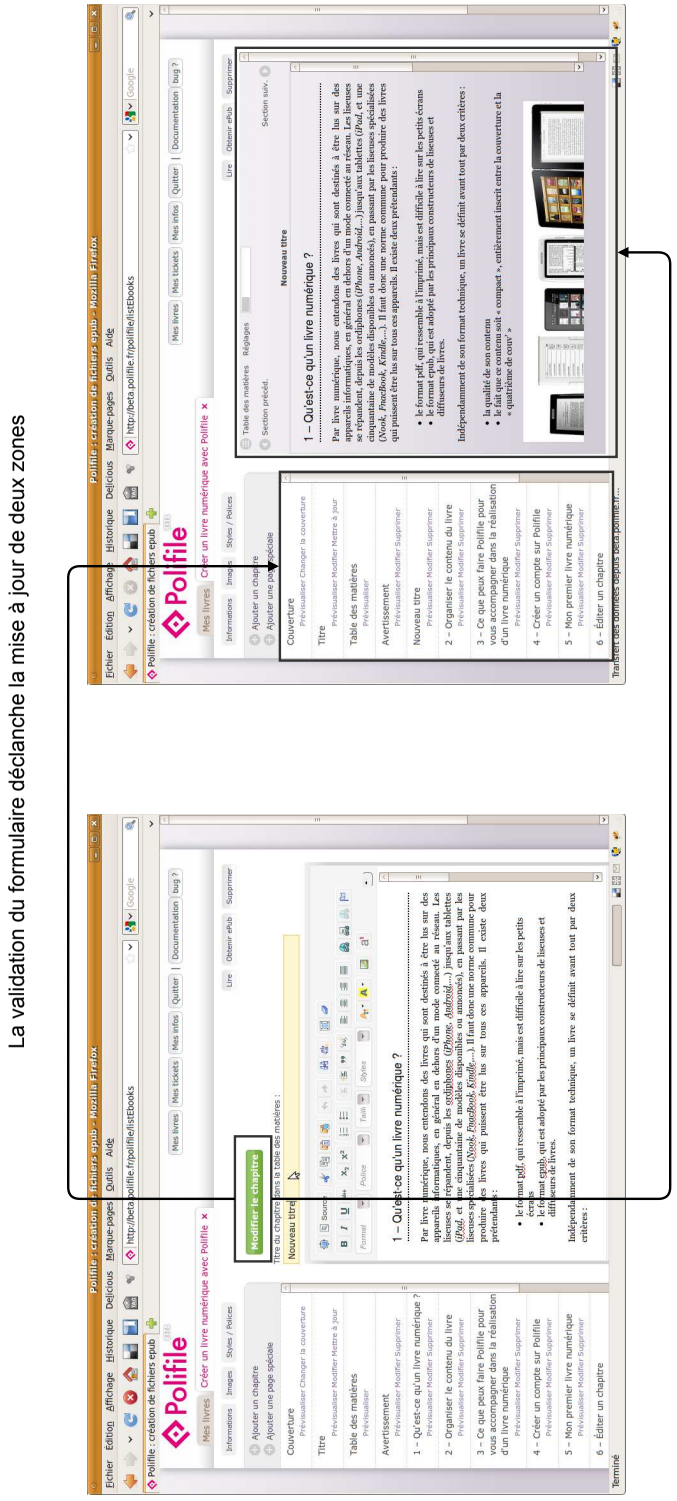



FIGURE 7.9 – Exemple de déclenchement de la mise à jour de plusieurs zones après soumission d'un formulaire

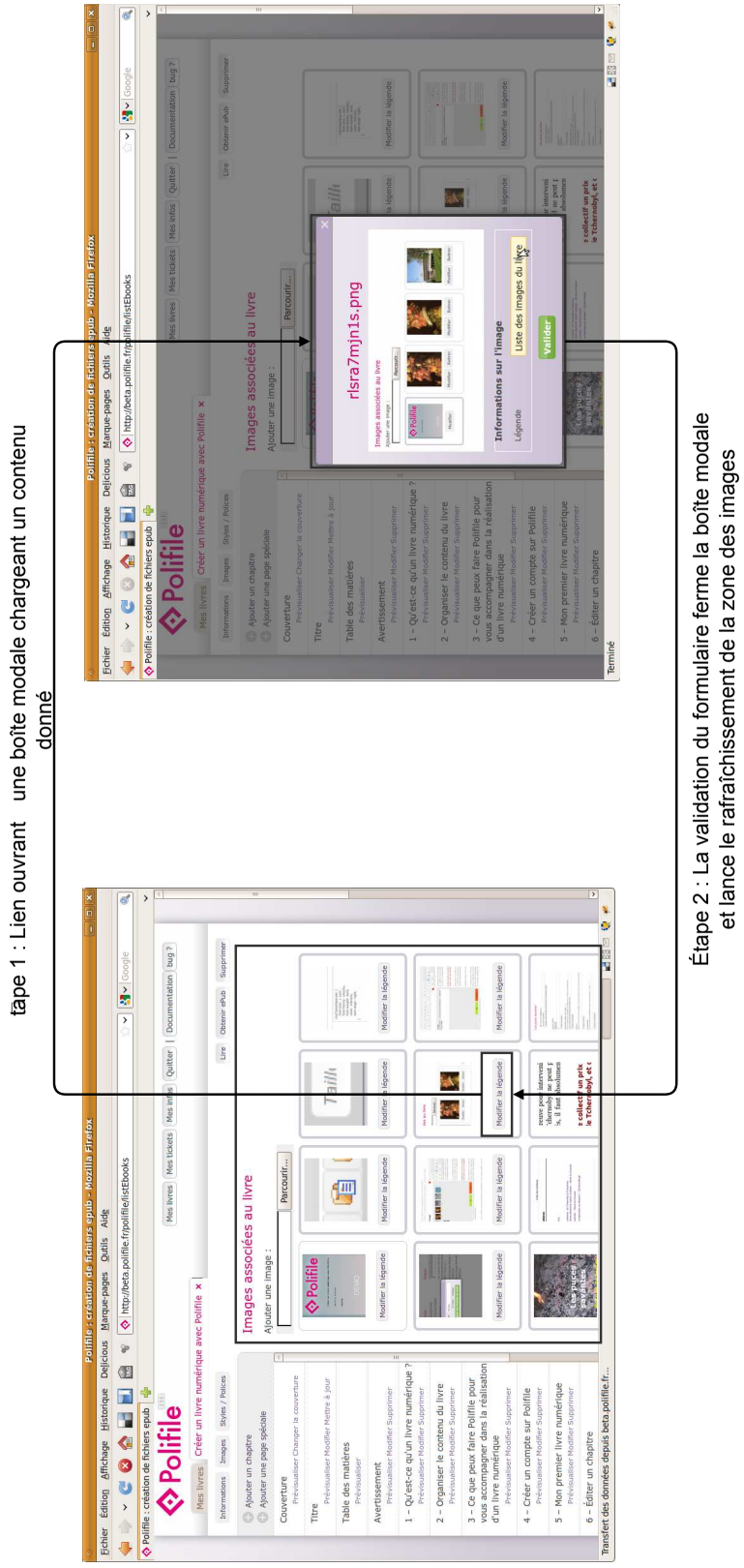



FIGURE 7.10 – Exemples de liens ouvrant une boîte modale

Nous avons imaginé une solution qui facilite le travail du développeur pour la réalisation de telles fonctionnalités. Dans la section suivante, nous présentons cette solution que nous avons nommée Majax.

### 7.6.4 Majax, ou l'Ajax magique

Nous avons réalisé une bibliothèque Javascript, nommée Majax [70], pour la gestion d'interactions Ajax du type de celles présentées auparavant.

Les objectifs de cette bibliothèque sont les suivants :

- gérer la mise à jour de zones de la page ;
- utiliser uniquement des classes (attribut HTML `class`), dans un esprit similaire à celui des microformats, afin d'éviter au développeur de créer des programmes Javascript pour chaque application.

Le principe de Majax réside dans la déclaration d'événements à écouter ou à déclencher par les zones de la page, par l'intermédiaire de classes.

Les événements à déclencher sont déclarés par une classe `majaxFire_<xxx>`, où `<xxx>` est le nom de l'événement à déclencher. Ils sont déclarés sur des éléments d'interaction comme les liens, les boutons de formulaire, ou d'ouverture de boîte modale.

Les événements à écouter sont déclarés par une classe `majaxListenTo_<xxx>`, où `<xxx>` est le nom de l'événement à capter. Lorsque l'événement est capté, la zone correspondante doit être mise à jour. Pour cela, il est nécessaire de connaître l'URL que la requête Ajax devra appeler. Cet URL est indiqué par l'élément concerné dans un attribut `data-url` spécifique à l'application [71].

La bibliothèque Majax gère aussi l'envoi des données d'un formulaire et la réponse du serveur, permet de déclencher un événement lors de cette réponse et peut mettre à jour une zone donnée avec la réponse du serveur.

Dans le cas de l'exemple 7.9, le formulaire d'édition d'un chapitre contient les classes suivantes :

- `majaxForm` indique à Majax que les données du formulaire doivent être envoyée via une requête Ajax. L'URL spécifié dans l'attribut `action` du formulaire est utilisé à cet effet ;
- `majaxFire_tocModified` indique à Majax que l'événement `tocModified`, c'est-à-dire la mise à jour des zones de la table des matières, doit être déclenché lorsqu'est reçue la réponse du serveur ;
- `majaxUpdate_workArea` indique à Majax que la zone `workArea` doit être mise à jour avec la réponse du serveur.

70. http://fr.wikipedia.org/wiki/Gerard_Majax

71. Avec HTML5, les attributs commençant par `data-` sont les attributs spécifiques à l'application (c.f. http://www.w3.org/TR/html5/elements.html).

La zone contenant le formulaire possède la classe `majaxContain_workArea`, ce qui indique que son contenu sera mis à jour avec la réponse du serveur. Le code ci-dessous présente cette zone et la balise du formulaire.

```
<div id="majaxContain_workingArea">
  <form class="majaxForm
               majaxFire_tocModified
               majaxUpdate_workingArea"
        method="post"
        action="saveUpdatedSection/subject_bs83h6nyvil">
  ...
  </form>
</div>
```

La zone englobant la table des matières possède la classe `majaxListenTo_tocModified`, ce qui indique que lorsque l'événement `tocModified` est capté, la zone est mise à jour en appelant l'URL contenu dans l'attribut `data-url`, comme le montre le code ci-dessous :

```
<div class="majaxListenTo_tocModified"
     data-url="getFragment/subject_bs83h6nyvil?fragment=toc">
... table des matières ...
</div>
```

La bibliothèque Majax se charge de la gestion des événements, de la gestion des appels Ajax, et de la gestion des parties à modifier dans l'arbre DOM. Le web designer peut ainsi se concentrer sur la création de son interface en utilisant des classes dans le HTML, langage qu'il maîtrise bien.

Pour l'utilisation des boîtes modales, un lien ou un bouton déclenche l'ouverture d'une boîte modale s'il possède une classe spécifique (`modalLink` par exemple). Par défaut, les liens contenus dans la modale seront activés en Ajax et la gestion des formulaires se fait aussi en Ajax dans la boîte modale. Ces comportements par défaut permettent au développeur de se concentrer sur les interactions à réaliser dans la modale. Il lui suffit alors de spécifier uniquement les liens ou actions qui ferment la boîte modale. Les actions à déclencher lors de la fermeture de la boîte modale sont spécifiées dans les classes du lien ou bouton ayant ouvert celle-ci, comme dans le cas l'envoi du formulaire détaillé dans l'exemple de Polifile.

Les retours d'expériences des développeurs qui ont utilisé ce système sont très positifs. Ils ont beaucoup apprécié la simplicité de mise en place qui leur permet de se concentrer sur l'ergonomie de l'application et leur demande simplement d'ajouter des classes sur les éléments concernés. Le fait de ne pas avoir à programmer de Javascript pour les cas les plus courants leur a fait gagner beaucoup de temps de développement et de débuggage.

## 7.7 Discussion

Le framework Sydonie a été utilisé par C&Féditions pour le développement de plusieurs applications. L'application de création de livres numériques Polifile est en ligne, la plate-forme de vente lisez-moi.lu est en cours de développement, et le site Mémoire des catastrophes [72] pour l'Institut de l'Histoire et de la Mémoire des Catastrophes sera prochainement lancé. Des étudiants en Master Ingénierie de l'Internet à l'Université de Caen Basse-Normandie ont aussi travaillé avec le framework Sydonie sur divers projets. Les retours d'expériences sont positifs de la part de C&Féditions et des étudiants. Deux développeurs de bon niveau ont pu prendre en main le framework assez rapidement. Pour deux autres développeurs ayant un an d'expérience dans le domaine du développement web, le développement a été un peu plus difficile, en particulier au début. Ils ont cependant réussi à mener à bien leurs développements. Des projets de recherche liés au framework Sydonie sont en cours, nous les présenterons à la fin de ce mémoire.

### FRBR et le framework

L'utilisation du modèle de document de Sydonie a apporté de réels bénéfices. La compréhension du modèle, au centre d'un projet de blog multilingue, a été un peu difficile à appréhender par un développeur. En revanche, la gestion des documents au moyen des entités inspirées des FRBR a été appréciée pour le développement de l'application `lisez-moi.lu`. L'utilisation de ce modèle a simplifié la gestion des divers formats de fichiers pour un même livre numérique. En effet, le regroupement des formats pour un livre est dans ce cas réalisé par la structure de document du framework.

### Stockage et abstraction de données

Pour s'assurer du bon fonctionnement de leur application, les développeurs web sont habitués à vérifier que les enregistrements sont bien réalisés en base de données. Avec Sydonie, cette étape de vérification s'avère difficile, puisque les données sont enregistrées dans les tables de façon « explosée ». Cet aspect fut déroutant pour les étudiants travaillant avec le framework. Cet obstacle est lié au niveau d'abstraction du système auquel ils n'étaient pas préparés. En revanche, le choix de déléguer la gestion de la base de données au framework leur a évité d'avoir à créer le schéma de base de données et d'écrire les requêtes SQL. Cet aspect a été apprécié et a permis d'accélérer le développement. Pour guider le développeur, nous créons des aides à la visualisation des objets présents dans Sydonie. La visualisation du contenu des objets avec un affichage spécifique permettant au développeur de comprendre le contenu de l'objet est en cours de création. De plus, une visualisation de configuration calculée par la « double cascade » pour

72. http://www.memoiredescatastrophes.org/

les configurations des objets, des formulaires, les templates, les actions et les labels, indique le résultat final de l'opération de cascade et dans quels fichiers chaque propriété est définie.

La gestion des ressources et de l'upload de fichier a permis aux développeurs de se concentrer sur les modes d'interaction et les processus à mettre en œuvre dans leur projets.

### Statements et permissions

La gestion des relations entre objets du système, à l'aide des Statements, apporte une flexibilité dans l'expression du modèle de l'application. Le développeur gère l'application en posant les Statements nécessaires entre les objets. Le gestionnaire de Statements permet de poser simplement des relations entre les objets. Lors du développement de Polifile, le modèle de l'application a évolué au cours du temps. À chaque fois, la flexibilité des Statements a permis de modifier rapidement le modèle de fonctionnement de l'application.

La gestion des permissions transparente a simplifié les processus de développement. Là encore dans le cadre de Polifile, la flexibilité des Statements a permis de modifier rapidement des politiques de permissions de l'application lors des évolutions des besoins. La définition de formules logiques pour définir une politique de permissions et leur transformation en requêtes SQL fonctionne bien. Des travaux sont en cours pour écrire la politique de permissions de façon plus naturelle pour les développeurs.

### Modèle de développement et framework

Avec Sydonie, le développeur se retrouve-t-il « enfermé » dans un modèle, comme nous l'avons beaucoup reproché aux CMS ? Le développeur doit effectivement se confronter au modèle de document et de données de Sydonie.

Il faut distinguer ce qui est une contrainte de langage (reprendre l'architecture de Sydonie avec les mêmes noms de fichiers, utiliser les templates, les labels, les actions et les Statements,. . . ) et ce qui est une contrainte de modèle.

Les contraintes de langage nécessitent un apprentissage, et l'étape de rédaction d'une documentation claire avec des exemples nombreux est déterminante. Nous allons nous y attaquer dans le cadre de la mise à disposition en logiciel libre de Sydonie dont nous parlerons dans la conclusion.

La liberté de choisir le modèle de l'application et de définir les relations entre les divers objets du système requiert là aussi un travail préalable. Cependant, la flexibilité du modèle de développement de Sydonie a permis la réalisation de projets où le modèle a évolué au fil du temps et des besoins. Par exemple, l'abstraction totale vis-à-vis de la base de données a permis la modification du modèle de données d'objets sans avoir à modifier les programmes ou a réécrire des requêtes SQL. Cette souplesse est un atout pour développer rapidement des fonctionnalités

de base pour une application et la compléter ensuite.

Dans tous les cas, l'utilisation de Sydonie a permis d'accélérer le développement. Pour l'application Polifile, le développement du système de paiement a été réalisé sans utiliser Sydonie, puis intégré dans l'application. Cela montre la souplesse du framework. Les fonctionnalités de création de formulaires, la gestion de erreurs de saisie ont été très utiles. La gestion de l'upload de fichier qu'il suffit d'adapter pour créer la fonctionnalité permet par exemple de développer très rapidement l'ajout d'une image à un objet. Enfin, les mécanismes Majax ont permis de se concentrer sur le modèle de l'application et sur les interactions à réaliser. Majax a, de plus, accéléré significativement les développements de ces interactions.

# Conclusion et perspectives

## Contributions

Les travaux présentés dans ce mémoire de thèse abordent les problèmes liés à la gestion et à la diffusion de document pour et sur le Web. Les solutions proposées passent par la construction de modèles inspirés de l'expérience des bibliothèques et la réalisation de collection d'outils pour l'ingénierie de développement web. Le framework réalisé illustre notre conception d'une recherche qui prend sens dans l'activité sur le web.

Nous avons orienté nos recherches autour de trois axes. Le premier axe étudie les travaux réalisés dans le cadre du rapport sur les spécifications fonctionnelles des notices bibliographiques. En utilisant les concepts élaborés par les bibliothécaires, nous avons proposé un modèle de document basé sur une métaphore autour des FRBR. Notre modèle propose de regrouper, sous la forme d'un arbre, les différentes versions linguistiques et divers formats de fichier d'un même document. L'entité Work offre un point d'entrée abstrait pour toutes les formes du document. En utilisant une négociation de contenu, le modèle permet de proposer la vue d'un document la plus adaptée pour un utilisateur donné. De plus, la négociation est utilisée au sein même des documents composites pour inclure les composants sous la forme la plus appropriée.

Le deuxième axe de nos recherches est centré sur le génie logiciel et l'ergonomie de développement. Nous avons associé les études sur l'ingénierie du web et notre expérience dans le développement web et l'encadrement de formations dans le domaine de l'ingénierie du web. Dans ce cadre, nous avons mené un travail de réflexion sur les processus mis en œuvre pour le développement d'applications web. L'implémentation dans le framework Sydonie propose des solutions souples et extensibles. Le modèle inspiré des FRBR simplifie la gestion des diverses versions linguistiques et des formats de fichier disponibles. La couche d'abstraction de données permet à un développeur de se concentrer sur le modèle de l'application web à créer. La gestion des relations entre objets à l'aide des Statements ajoute à la flexibilité du système, et permet de déterminer les droits d'un utilisateur sur les objets. La politique de permissions d'une application est définie à l'aide de formules logiques. Enfin, nous avons modélisé les interactions Ajax

et proposé une bibliothèque qui permet à un graphiste de simplement déclarer les zones d'une page à rafraîchir.

Le troisième axe de nos travaux étudie les mécanismes à proposer pour rendre la gestion des métadonnées plus accessible aux développeurs web. Cet axe est transverse aux deux premiers. Il utilise le modèle de document sous forme d'arbre pour affecter les métadonnées aux diverses entités de l'arbre. Cette représentation évite les redondances d'information et permet la réutilisation des données des entités Work ou Expression, dans le cas de la création de nouvelles Manifestations par exemple. Les mécanismes proposés dans le framework offrent au développeur des moyens de définir de façon simple les correspondances entre l'ensemble des métadonnées contenues dans un document et les nœuds de l'arbre.

## Travaux en cours

En proposant un modèle de document et un modèle d'interactions, Sydonie se veut être une plate-forme au service de la recherche et du développement d'applications. Grâce à la souplesse des mécanismes proposés, le champ d'applications est large. Les travaux en cours, présentés ici, prolongent les recherches déjà réalisées.

Dans le cadre du partenariat avec C&Féditions, le développement de la plate-forme de diffusion de livres numériques `lisez-moi.lu` continue. Il servira pour améliorer les mécanismes présentés au chapitre 6 pour la gestion des métadonnées dans les fichiers de types ePUB ou PDF. Ce projet verra le jour courant 2012.

C&Féditions envisage l'utilisation de Sydonie pour les autres projets qui ne manqueront pas de se présenter dans l'année qui vient.

Le Centre de Recherches Archéologiques et Historiques Anciennes et Médiévales (Craham)[73] de l'Université de Caen Basse-Normandie regroupe au sein du Centre Michel de Boüard près de 90 personnes. Les sujets de recherche sont axés autour de l'étude des sociétés du passé dans l'espace et dans le temps, en travaillant sur leurs dynamiques culturelles et la production de territoires.

L'exposition sur Michel de Boüard, créée au Musée de Normandie en 2009, a été l'occasion d'une prise de conscience des risques de dégradation encourus par le fonds photographique du Craham. Ce fonds compte environ 11000 diapositives et sans doute plus de 1000 négatifs. Il documente l'activité du centre de recherches et de ses membres des années 1950 aux années 1980. La dégradation des couleurs d'un grand nombre de ces diapositives a amené le Centre à débuter une opération de numérisation de l'ensemble du fonds. Environ 7000 diapositives

73. http://www.unicaen.fr/craham/

ont d'ores et déjà été numérisées. Au-delà de cette préservation de la mémoire de l'archéologie médiévale française, le Craham souhaite mettre en ligne, à la disposition de l'ensemble des chercheurs européens, l'ensemble du fonds de photographies.

L'équipe Sydonie intervient d'ores et déjà dans ce projet. Notre rôle est de modéliser et mettre en œuvre la gestion du fonds d'images numérisées. Ce projet de recherche en collaboration avec le Craham va permettre d'expérimenter le modèle de gestion des métadonnées au sein des images pour des collections de grande taille. Les modes de consultation du fonds par le grand public ou les chercheurs est à définir. Enfin, nous comptons utiliser le framework pour concevoir un outil d'organisation et de classification des images. Cet outil devra être simple, efficace et ergonomique. La réalisation de cet outil pourra ensuite être intégrée dans le framework.

Créé à l'initiative de chercheurs et de professionnels de l'édition, l'Institut Mémoires de l'Édition Contemporaine (IMEC) [74] rassemble, préserve et met en valeur des fonds d'archives et d'études consacrés aux principales maisons d'édition, aux revues et aux différents acteurs de la vie du livre et de la création. L'IMEC est actuellement dans un processus de ré-informatisation pour la gestion et l'accès en ligne à ses collections. Dans le cadre du Contrat de Plan État Région, l'IMEC et l'équipe de recherche Sydonie vont développer un système pour mettre en valeur certaines sous-collections du fonds. Ce projet est en cours de lancement.

Dans le cadre d'une thèse encadrée par Catherine Jacquemard et Hervé Le Crosnier, Pierre-Yves Buard étudie l'impact des FRBR sur la production des éditions critiques de sources primaires, réalisés par des chercheurs en Sciences Humaines et Sociales. L'objet de ses recherches se concentre sur les ensembles de manuscrits pour lesquels l'existence d'une « œuvre » n'est pas clairement établie. Le chercheur dispose dans ce cas d'un ensemble de Manifestations mais l'existence d'une entité Work ou Expression, ou « Œuvrexpression » pour reprendre [Le Bœuf 03], reste à établir. L'objectif de cette thèse est de fournir un modèle ainsi que des outils pour aider les chercheurs en Sciences Humaines et Sociales dans l'étude de tels corpus de fragments de manuscrits. Les outils réalisés sont développés avec le framework Sydonie et s'appuient sur l'utilisation du modèle FRBR déjà implémenté dans le framework. Ils permettront d'élargir les champs d'applications de FRBR pour la gestion de documents numériques.

Les différents projets en cours montrent le dynamisme de l'équipe et les passerelles qui sont construites avec les Sciences Humaines et Sociales. Dans ce cadre, l'équipe Sydonie participe aux travaux du pôle Document Numérique [75] animé par la Maison de la Recherche en Sciences Humaines de l'Université de Caen Basse-Normandie.

74. http://www.imec-archives.com/
75. http://www.unicaen.fr/recherche/mrsh/document_numerique

# Perspectives

## Sydonie logiciel libre

Comme nous avons pu le souligner maintes fois, les besoins et attentes des développeurs web ont été analysés et pris en compte pour la création d'outils qui facilitent le développement d'applications web. Les premiers utilisateurs du framework nous ont permis de cerner les points positifs et les aspects à améliorer. Le framework Sydonie, logiciel libre, est désormais prêt à être diffusé, et la construction d'une communauté autour de Sydonie sera l'un des enjeux de ses évolutions. Les étudiants qui travailleront en projet avec le framework Sydonie offrent un bon moyen de commencer la diffusion et l'adoption du framework.

Nous réfléchissons à l'utilisation de la logique métier sans intégration forte pour rendre le framework plus attractif pour des développements d'applications. Cette réflexion se doit d'éviter les écueils des *plugins* utilisés avec les CMS (conflits entre plugins, adaptation difficile, etc.).

L'utilisation d'une architecture orientée services (SOA[76]) est une des solutions envisagées pour cette intégration, comme le propose [Hinton 11]. Un service web en mode Rest est d'ores et déjà utilisé par l'application Mémoire des catastrophes. Développé par C&Féditions, le service web permet la recherche de villes de France. Cette expérience est une des pistes de réflexion pour l'intégration de la logique métier dans les applications utilisant Sydonie.

## Modèle de données RDF et stockage

Le modèle de données interne à Sydonie est très largement basé sur les relations entre objets du système sous la forme de triplets (sujet, prédicat, objet). Les contenus des documents sont gérés sous la forme d'objets AttributeType liés au document. Les relations entre objets sont gérés *via* les Statements directement inspirés du modèle RDF. De plus, tous les autres objets gérés par le framework Sydonie sont basés sur ce même fonctionnement.

Nous allons utiliser le framework Sydonie pour permettre aux applications web d'intégrer le web de données. Pour cela, les données d'une application sont rendues publiques sous la forme de triplets RDF. Il faut pouvoir interroger le système ainsi réalisé. Un moteur de requêtes SPARQL [Prud'hommeaux & Seaborne 08] doit être développé, pour la mise à disposition d'un *Sparql endpoint* pour la mise à disposition de données. L'intégration des applications gérées par Sydonie dans le Web de Données[77] est d'ailleurs un des objectifs prioritaires dans les évolutions du framework.

L'utilisation d'un stockage RDF pour la persistance des données gérées par Sydonie est une expérimentation qui va être menée en complément de la mise à disposition des données en RDF. L'utilisation d'un système de base de données relationnelle reste, cependant, le meilleur moyen

76. Service Oriented Architecture
77. http://www.w3.org/standards/semanticweb/data

de garantir la possibilité d'utiliser Sydonie avec un hébergement web standard. Néanmoins, la possibilité de stocker des triplets RDF dans une base de données relationnelle est aussi possible.

### Sydonie et documents composites

Le modèle de document proposé pour Sydonie a été utilisé avec succès dans plusieurs applications. Le modèle basé sur FRBR et ses avantages pour gérer diverses Expressions et Manifestations ont été mis à l'épreuve. Nous présentons ici des aspects du document composite qui seront approfondis dans la suite de nos travaux.

### Document composite ou composition de documents ?

Les documents composites gérés par Sydonie ont été, selon les applications, gérés de façons différentes. Pour les applications où la mise en page des différents éléments est fixée par le graphiste, l'interface de création d'un document pourra proposer, par exemple, la saisie de texte et l'ajout des images associées de façon distincte. Le texte saisi sera alors enregistré dans un document. Chaque image est associée à ce document *via* un Statement. Le document ne spécifie pas la mise en forme des différentes parties auxquelles il est associé. Le template est celui qui spécifie cette mise en forme. Ce dernier précise la façon dont l'ensemble sera recomposé.

Peut-on alors considérer ce document comme un document composite ? En effet, la Manifestation HTML ne contient en aucun cas les images associées. Dans cette situation, le document composite est en fait matérialisé par l'intégration des différents composants dans le template d'affichage. Sans celui-ci, le système peut connaître la liste des composants mais ne peut pas produire une vue de l'ensemble.

Une autre façon d'appréhender la composition peut être la suivante, par exemple. Dans une application où l'utilisateur peut saisir un texte libre en utilisant une syntaxe de type Wiki, il peut insérer des références à d'autres documents. Par exemple, en insérant le code `[[x32b, small]]`, l'image d'identifiant `x32b` sera affichée dans un format défini par le mot-clé `small` à l'endroit où le code est inséré. Le système traite alors le texte saisi pour ajouter les Statements nécessaires. La Manifestation du document saisi contient dans ce cas la mise en page des documents inclus.

Ces deux exemples montrent l'ambivalence du statut des *templates* dans le système actuel. Il illustre aussi la variété des cas rencontrés lors de développement d'applications. L'utilisateur peut-il créer un document avec une mise en forme libre, ou est-il soumis à une présentation donnée ? Dans le cas où des contraintes de présentation sont présentes, une réflexion complémentaire doit être menée pour déterminer comment ces contraintes sont exprimées. Les pistes de XML, avec PRISM ou TEI par exemple, sont bien entendu envisagées mais les problématiques d'éditeur HTML ou XML reviennent une fois de plus pour l'interface utilisateur. L'utilisation

d'autres outils basés sur XML, comme l'éditeur AXEL [Quint *et al.* 10] par exemple, doit être étudiée pour envisager d'autres modes de saisie.

#### Relations entre parties de documents

Les relations entre documents du système se limitent à la composition de documents complets. En effet, les relations `EST_PARTIE_DE` ou `A_POUR_PARTIE` sont utilisables entre entités de type Work. Cependant, d'autres types de relations sont possibles et définies dans les FRBR.

Premier exemple, les relations ne sont pas encore utilisées pour spécifier des relations entre documents textuels. Par exemple, si un Work A est composé de chapitres, chaque chapitre peut être lui-même un Work C1, C2, etc.

Lorsque la Manifestation en HTML du Work A doit être affichée, le système devra présenter une table des matières avec des liens vers les parties C1, C2, etc. En revanche, si la Manifestation au format ePUB du Work A est demandée, alors cette Manifestation devra intégrer tous les contenus des parties C1, C2, etc.

Autre exemple, les relations entre Expressions sont uniquement des relations entre versions linguistiques. Or, FRBR définit d'autres types de relations entre Expressions. Par exemple, une table des matières, un index ou un glossaire peut être considéré comme une Expression qui est une partie d'une autre Expression.

Les travaux de Pierre-Yves Buard ne manqueront pas d'apporter des éclaircissements sur ces situations. Il a en effet choisi d'utiliser Sydonie dans le cadre de sa thèse car il bénéficie ainsi de la gestion des documents inspirée de FRBR.

### Documents et multilinguisme

Dans le domaine du multilinguisme, le modèle de document proposé par Sydonie fournit une base de travail pour réaliser des travaux dans le domaine des traitements du multilinguisme.

Le modèle de document de Sydonie intègre les versions linguistiques d'un même document, en créant une Expression pour chaque nouvelle version d'un document. Nous allons travailler en collaboration avec d'autres membres de l'équipe Document, Langues et Usages du GREYC pour intégrer des services de traduction automatique. De tels services permettraient de fournir une version linguistique de base pour un traducteur. Cependant, dans la cas où une traduction est mise en ligne sans amélioration, les internautes consultant les documents doivent être informés de la qualité de la version linguistique qu'ils consultent. De plus, le cas de l'existence de plusieurs versions dans une même langue devra être traité de façon systématique : parmi ces versions, doit-on en proposer une par défaut à la consultation ? Ou doit-on proposer la liste, mais, dans ce cas, quelles aides proposer au choix pour l'utilisateur ? Enfin, comment « classer » ces traductions en

fonction de leur qualité de traduction ? Nous souhaitons expérimenter l'utilisation de relations pondérées entre les Expressions. La pondération pourrait exprimer la fiabilité de la traduction.

Les travaux en collaboration avec des membres de l'équipe DLU, spécialiste du traitement automatique des langues, permettront d'approfondir ces questions. De plus, les problématiques d'alignement de textes entre diverses traductions recouvre les problématiques des documents composites évoquées précédemment. En effet, dans ce cadre, un document sera composé des diverses Expressions de la même entité Work ou d'entités Work différentes. On retrouve la problématique des documents et chapitres présentée ci-avant.

### Sydonie et le Web

Les applications web d'aujourd'hui doivent pouvoir être interconnectées. Les formats Prism et newsML sont utilisés par les professionnels de la presse et des magazines. La TEI est utilisée dans le domaine des Sciences Humaines et Sociales. Nous allons réaliser des mécanismes d'import/export pour améliorer les fonctionnalités du framework dans le domaine de l'édition.

Nous avons utilisé les FRBR comme modèle pour les documents. Cette métaphore opérationnelle fonctionne bien. Le modèle de document utilisé par Sydonie apporte de réels avantages pour la gestion des documents. D'autres rapports publiés par l'IFLA [Patton 08] [Zeng *et al.* 10] viennent compléter les FRBR pour constituer la *FRBR family*.

Pour aborder la gestion des personnes et des outils documentaires dans le framework, nous allons regarder comment cette *FRBR family* modélise les relations et entités. Les relations avec les travaux du W3C [78], ou les systèmes organisés de gestion des connaissances, seront étudiées pour établir un modèle pour le framework Sydonie.

Une étude plus approfondie de ces travaux et une réflexion sur leurs relations avec les technologies du web devrait faire évoluer le modèle de Sydonie et permettre l'émergence de processus pour la gestion des personnes et des sujets. Nous aimerions étudier la façon dont les modèles élaborés par les bibliothécaires peuvent s'intégrer dans le web interconnecté, le web de données, ou les systèmes organisés de gestion des connaissances de façon plus générale. Plus généralement, les relations des modèles définis par la *FRBR family* avec le web mérite des études approfondies.

## Conclusion

Dans le domaine de l'industrie du développement web, la diffusion du framework en logiciel libre nous fera partager ces modèles avec une communauté de développement. Le partenariat avec C&Féditions se poursuit et les projets d'applications utilisant Sydonie ne manquent pas.

Dans le domaine de la recherche, les travaux de thèse de Pierre-Yves Buard feront avancer

78. Par exemple, SKOS Simple Knowledge Organization System, http://www.w3.org/TR/skos-reference/

les modèles utilisés par Sydonie et apporteront de nouvelles problématiques pour l'édition et la gestion de documents. Les projets menés avec l'Imec et le Craham permettront eux-aussi d'apporter de nouvelles idées pour le futur du projet. L'obtention des fonds FEDER de la Communauté Européenne permettra l'embauche d'un ingénieur pour nous aider à poursuivre nos recherches et réaliser des développements.

Enfin, l'expérience acquise montre que la conception et la réalisation d'un framework est un bon support pour l'enseignement de l'ingénierie du web. Cette ingénierie doit être articulée autour de la réflexion sur les métiers et les processus associés pour réaliser ensuite la définition des modèles. C'est ce que nous avons essayé de réaliser et de montrer dans ce mémoire.

# Bibliographie


[DC 95] « Dublin Core Metadata Initiative », 1995. {http://dublincore.org/}

[OPF 07] « Open Packaging Format (OPF) 2.0 », Rapport technique, International Digital Publishing Forum, Septembre 2007.

[OPG 10] « The Open Graph Protocol », Rapport technique, facebook, 2010.

[ECM 11] « ECMAScript Language Specification », Rapport technique, Ecma international, Juin 2011.

[Adida & Birbeck 08] B. Adida & M. Birbeck, « RDFa Primer, Bridging the Human and Data Webs », Rapport technique, W3C, Octobre 2008.

[Adida *et al.* 11] B. Adida, M. Birbeck, S. McCarron & I. Herman, « RDFa Core 1.1 », Rapport technique, W3C, Mars 2011.

[Adrian *et al.* 10] B. Adrian, J. Hees, I. Herman, M. Sintek, A. Dengel, P. Cimiano & H. Pinto, « Epiphany : Adaptable RDFa Generation Linking the Web of Documents to the Web of Data », *in* P. Cimiano & H. S. Pinto (dir.), *Proceedings of EKAW 2010 - Knowledge Engineering and Knowledge Management by the Masses*, vol. 6317 de *Lecture Notes in Computer Science*, pages 178–192, Berlin, Heidelberg, 2010, Springer Berlin Heidelberg.

[Aryeh 11] G. Aryeh, « HTML Editing APIs », Rapport technique, W3C, August 2011.

[Baker 00] T. Baker, « A Grammar of Dublin Core », *D-Lib Magazine*, vol. 6, n° 10, Octobre 2000.

[Barker 09] S. Barker, « The next 700 access control models or a unifying meta-model ? », *in Proceedings of the 14th ACM symposium on Access control models and technologies*, SACMAT '09, pages 187–196, New York, NY, USA, 2009, ACM.

[Barker & Rosenthal 01] S. Barker & A. Rosenthal, « Flexible Security Policies in SQL », *in In Proceedings of the 15th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, pages 15–18, 2001.

[Beged-Dov *et al.* 00] G. Beged-Dov, D. Brickley, R. Dornfest, I. Davis, L. Dodds, J. Eisenzopf, D. Galbraith, R. Guha, K. MacLeod, E. Miller, A. Swartz &

E. van der Vlist, « RDF Site Summary (RSS) 1.0 », Rapport technique, RSS-DEV Working Group, 2000.

[Bizer *et al.* 08] C. Bizer, T. Heath, K. Idehen & T. Berners-Lee, « Linked Data on the Web (LDOW2008) », *in Proceedings of WWW2008*, 2008.

[Bruce & Hillmann 04] T. R. Bruce & D. Hillmann, *The continuum of metadata quality : defining, expressing, exploiting*, chap. 15, pages 238–256, ALA Editions, 2004.

[Butuc 09] M.-G. Butuc, « Semantically Enriching Content Using OpenCalais », *Interface*, vol. 9, n° Figure 2, 77–80, 2009.

[Christensen *et al.* 01] E. Christensen, F. Curbera, G. Meredith & S. Weerawarana, « Web Services Description Language (WSDL) 1.1 », Rapport technique, W3C, Mars 2001.

[Committee 08] X. T. Committee, « XLIFF Version 1.2 », Rapport technique, Organization for the Advancement of Structured Information Standards (OASIS), février 2008.

[Conboy *et al.* 11] G. Conboy, M. Garrish, M. Gylling, W. McCoy, M. Makoto & D. Weck, « EPUB 3 OVerview », Rapport technique, International Digital Publishing Forum (IDPF), Septembre 2011.

[Connell 08] R. S. Connell, « Survey of Web Developers in Academic Libraries », *The Journal of Academic Librarianship*, vol. 34, n° 2, 121–129, mars 2008.

[Connolly 07] D. Connolly, « Gleaning Resource Descriptions from Dialects of Languages (GRDDL) », Rapport technique, W3C, Septembre 2007.

[Corlosquet 11] S. Corlosquet, « The future of structured data in HTML : RDFa, Microdata and microformats », Juin 2011. {http://groups.drupal.org/node/157784}

[Corlosquet *et al.* 09] S. Corlosquet, R. Cyganiak, A. Polleres & S. Decker, « RDFa in Drupal : Bringing cheese to the web of data », *in Proc. of 5th Workshop on Scripting and Development for the Semantic Web at ESWC*, vol. 2009. Citeseer, 2009.

[Cubrilovic 11] N. Cubrilovic, « Logging out of Facebook is not enough », Septembre 2011. {http://nikcub-cache.appspot.com/logging-out-of-facebook-is-not-enough}

[Cutter 76] C. A. Cutter, *Rules for a Dictionary Catalog*, Washington, D.C. : Government Printing Office, 1876.

[DCMI 08] DCMI, « Metadata Terms », Rapport technique, DCMI, January 2008.

[Doctorow 01] C. Doctorow, « Metacrap : Putting the torch to the seven straw-men of the meta-utopia », 2001. {http://www.well.com/~doctorow/metacrap.htm}

[Dornier & Buard 08] C. Dornier & P.-Y. Buard, « Editer un cahier de travail de Montesquieu : les apports du numérique et de la TEI », *Recherches & Travaux*, n° n ° 72, p. 139–156., septembre 2008.

[Fielding 00] R. Fielding, « Architectural Styles and the Design of Network based Software Architectures », Rapport technique, Univerity of California, Irvine, 2000.

[Franzon 11] E. Franzon, « Schema.org Workshop – A Path Forward », Septembre 2011. {http://semanticweb.com/schema-org-workshop-a-path-forward_b23387}

[Gilliland 08] A. J. Gilliland, « Setting the Stage », *in* M. Baca (dir.), *Introduction to Metadata Version 3.0*, vol. 10, chap. 1, pages 1–19, Getty Research Institute, 2008.

[Ginige & Murugesan 01] A. Ginige & S. Murugesan, « Web engineering : An introduction », *Multimedia, IEEE*, vol. 8, n° 1, 14–18, 2001.

[Haslhofer & Klas 10] B. Haslhofer & W. Klas, « A survey of techniques for achieving metadata interoperability », *ACM Computing Surveys*, vol. 42, n° 2, 1–37, February 2010.

[Herman 11] I. Herman, « Proposing two new SW Interest Group Task Forces », Septembre 2011. {http://www.w3.org/QA/2011/09/proposing_two_new_sw_interest.html}

[Hickson 10] I. Hickson, « Web SQL Database W3C Working Draft », Rapport technique, W3C, Novembre 2010.

[Hickson 11a] I. Hickson, « HTML Microdata », Rapport technique, W3C, Mai 2011.

[Hickson 11b] I. Hickson, « HTML5 W3C Working Draft », Rapport technique, W3C, May 2011.

[Hickson 11c] I. Hickson, « Web messaging API Specification Working Draft », Rapport technique, W3C, Mars 2011.

[Hickson 11d] I. Hickson, « Web Storage W3C Working Draft », Rapport technique, W3C, Septembre 2011.

[Hickson 11e] I. Hickson, « Web workers API Specification Working Draft », Rapport technique, W3C, Septembre 2011.

[Hinton 11] E. Hinton, « The Twilight of the CMS », 2011. {http://labs.talkingpointsmemo.com/2011/07/the-twilight-of-the-cms.php}

[Hodgins & Duval 02] W. Hodgins & E. Duval, « IEEE Standard for Learning Object Metadata », Rapport technique, IEEE, 2002.

[IDEAlliance 09] IDEAlliance, « PRISM : Publishing Requirements for Industry Standard Metadata », Rapport technique, International Digital Enterprise Alliance, Inc., 2009. {http://www.prismstandard.org/}

[IETF-RFC2426 98] IETF-RFC2426, « vCard MIME Directory Profile », Rapport technique, IETF, 1998.

[IETF-RFC2445 98] IETF-RFC2445, « Internet Calendaring and Scheduling Core Object Specification (iCalendar) », Rapport technique, IETF, 1998.

[IPTC 11a] IPTC, « Object Reuse and Exchange Primer », Rapport technique, IPTC, Février 2011.

[IPTC 11b] IPTC, « schema.org adopts IPTC's rNews for news markup », Septembre 2011.

[Johnson 09] C. Johnson, « Content Management Systems just don't work », 2009. {http://sunlightlabs.com/blog/2009/content-management-systems-just-dont-work/}

[Johnston & Powell 08] P. Johnston & A. Powell, « Expressing Dublin Core metadata using HTML/XHTML meta and link elements », Rapport technique, DCMI, Aout 2008.

[Kautz & Nø rbjerg 03] K. Kautz & J. Nø rbjerg, « Persistent problems in information systems development : the case of the World Wide Web », *in Proceedings of the 11th European Conference on Information Systems (ECIS)*. Citeseer, 2003.

[Klyne & Carroll 04] G. Klyne & J. J. Carroll, « Resource Description Framework (RDF) : Concepts and Abstract Syntax », Rapport technique, W3C, 2004.

[Landesman 11] B. Landesman, « Seeing Standards : A Visualization of the Metadata Universe », *Technical Services Quarterly*, vol. 28, n° 4, 459–460, 2011.

[Le Bœuf 03] P. Le Bœuf, « FRBR : un p'tit coin d'paradigme... », *in First IFLA Meeting of Experts on an International Cataloguing Code*, 2003.

[Le Crosnier & Lecarpentier 10] H. Le Crosnier & J.-M. Lecarpentier, « Webdesign, normalisation et stratégies des firmes », *in* K. Zreik (dir.), *actes de la conférence CIDE 13 (13e Colloque International sur le Document Electronique)*, Paris, France, Décembre 2010, Europia éditions.

[Le Hors *et al.* 04] A. Le Hors, P. Le Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion & S. Byrne, « Document Object Model (DOM) Level 3 Core Specification », Rapport technique, W3C, Avril 2004.

[Le Hégaret 10] P. Le Hégaret, « ISSUE-7 : codec support and the ¡video¿ element », Mai 2010. {http://www.w3.org/QA/2010/05/html5_video.html}

[Lecarpentier *et al.* 10] J.-M. Lecarpentier, C. Bazin & H. Le Crosnier, « Multilingual composite document management framework for the internet : an FRBR approach », *in Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*, page 13, New York, New York, USA, septembre 2010, ACM Press.

[Lecarpentier *et al.* 08] J.-M. Lecarpentier, H. Le Crosnier & J. Madelaine, « Evolutions de l'architecture du web et des documents numériques », *in* E. Broudoux & G. Chartron (dir.), *actes de la deuxième conférence (DocSoc '08)*, pages 13–30, Paris, France, Novembre 2008, ADBS éditions.

[Madison 98] O. Madison, *Functional Requirements for Bibliographic Records*, K. G. Saur, München, Germany, 1998.

[Madison 01] O. Madison, *Spécifications Fonctionnelles des Notices Bibliographiques*, BnF, Paris, France, 2001.

[Manola & Miller 04] F. Manola & E. Miller, « RDF Primer », Rapport technique February, W3C, 2004.

[Mehta *et al.* 11] N. Mehta, J. Sicking, E. Graff, A. Popescu & J. Orlow, « Indexed Database API W3C Working Draft », Rapport technique, W3C, Avril 2011.

[Mendes *et al.* 06] E. Mendes, N. Mosley & S. Counsell, « The Need for Web Engineering : An Introduction », *Web Engineering*, E. Mendes & N. Mosley (dir.), pages 1–27, 2006.

[Mitra & Lafon 07] N. Mitra & Y. Lafon, « SOAP Version 1.2 Primer (Second Edition) », Rapport technique, W3C, Avril 2007.

[Murugesan 08] S. Murugesan, « Web Application Development : Challenges and the Role of Web Engineering », *in* G. Rossi, O. Pastor, D. Schwabe & L. Olsina (dir.), *Web Engineering : Modelling and Implementing Web Applications*, Human-Computer Interaction Series, chap. 2, pages 7–32, Springer London, London, 2008.

[Nelson & Warner 08] M. Nelson & S. Warner, « The Open Archives Initiative Protocol for Metadata Harvesting », Rapport technique, Open Archives Initiative, Décembre 2008.

[NISO 04] NISO, « Understanding Metadata », Rapport technique, NISO Press, 2004.

[NISO 07] NISO, « A Framework of Guidance for Building Good Digital Collections - 3rd edition », Rapport technique Decembre, National Information Standards Organization (NISO), Baltimore, MD, 2007.

[Nottingham & Sayre 05] M. Nottingham & R. Sayre, « The Atom Syndication Format », Rapport technique, The Internet Society, 2005.

[Opera 07] Opera, « Opera Press Releases, Opera files antitrust complaint with the EU. », http ://www.opera.com/press/releases/2007/12/13/, Décembre 2007. {http://www.opera.com/press/releases/2007/12/13/}

[OSCAR 05] OSCAR, « TMX 1.4b Specification », Rapport technique, LISA, avril 2005.

[Patton 08] G. E. Patton, *Functional Requirements for Authority Data*, K. G. Saur, München, Germany, 2008.

[Pemberton 02] S. Pemberton, « XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition) », Rapport technique, W3C, August 2002.

[Pemberton *et al.* 06] S. Pemberton, M. Birbeck, M. Gylling & S. McCarron, « XHTML™ 2.0 W3C Working Draft », Rapport technique, W3C, Juillet 2006.

[Popescu 10] A. Popescu, « Geolocation API Specification Working Draft », Rapport technique, W3C, Septembre 2010.

[Prud'hommeaux & Seaborne 08] E. Prud'hommeaux & A. Seaborne, « SPARQL Query Language for RDF », Rapport technique January, W3C, 2008.

[Pédauque 06] R. T. Pédauque, *Le document à la lumière du numérique*, C&F éditions, 2006.

[Quint *et al.* 10] V. Quint, C. Roisin, S. Sire & C. Vanoirbeek, « From Templates to Schemas : Bridging the Gap Between Free Editing and Safe Data Processing », *in Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*, page 61, New York, New York, USA, septembre 2010, ACM Press.

[Riley & Becker 09] J. Riley & D. Becker, « Seeing Standards : A Visualization of the Metadata Universe », 2009. {http://www.dlib.indiana.edu/~jenlrile/metadatamap/}

[Rode & Rosson 06] J. Rode & M. M. B. Rosson, « End user development of web applications », *in End User Development*, pages 161–182, Springer, 2006.

[Rosson *et al.* 05a] M. Rosson, J. Ballin & J. Rode, « Who, What, and How : A Survey of Informal and Professional Web Developers », *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 199–206, 2005.

[Rosson *et al.* 05b] M. B. Rosson, J. F. Ballin, J. Rode, B. Toward, D. Lowe & M. Gaedke, « Designing for the Web " Revisited : A Survey of Informal and Experienced Web Developers », *ICWE 2005*, D. Lowe & M. Gaedke (dir.), vol. 3579, 522–532, 2005.

[Roszkiewicz 08] R. Roszkiewicz, « XMP Primer », Rapport technique, IDEAlliance, 2008.

[SAA Photo Metadata Project] SAA Photo Metadata Project, « Are you metasmart ? », {http://photometadata.org/}

[Salven 07] E. G. Salven, « Basic Metadata : A Photographer's Best Friend », *ASPM Bulletin*, n° Fall, 20–23, 2007.

[Sandhu *et al.* 96] R. S. Sandhu, E. J. Coyne, H. L. Feinstein & C. E. Youman, « Role-Based Access Control Models », *IEEE Computer*, vol. 29, n° 2, 38–47, 1996.

[Sauermann *et al.* 08] L. Sauermann, R. Cyganiak, D. Ayers & M. Völkel, « Cool URIs for the Semantic Web », Rapport technique, W3C, Decembre 2008.

[Singel 11a] R. Singel, « Google Explores Re-Ranking Search Results Using +1 Button Data », Aout 2011. {http://www.wired.com/epicenter/2011/08/google-studying-re-ranking-search-results-using-1-button-data-but-its-touchy/}

[Singel 11b] R. Singel, « Google Plus vs. Facebook on privacy : Plus ahead on points - for now », Juin 2011. {http://www.wired.co.uk/news/archive/2011-06/29/google-facebook-privacy}

[Sporny & McCarron 11] M. Sporny & S. McCarron, « HTML+RDFa 1.1 », Rapport technique, W3C, Mai 2011.

[Stevens 10] K. Stevens, « The ALA 2010 Web Design Survey », *A List Apart*, 2010.

[Stonebraker 75] M. Stonebraker, « Implementation of integrity constraints and views by query modification », *in Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, SIGMOD '75, pages 65–78, New York, NY, USA, 1975, ACM.

[TEI 07] TEI, « Text Encoding Initiative », 2007. {http://www.tei-c.org/}

[Tennison 11] J. Tennison, « Using Multiple Vocabularies in Microdata », Juillet 2011. {http://www.jenitennison.com/blog/node/161}

[Thomas 97] R. K. Thomas, « Team-based access control (TMAC) », *in Proceedings of the second ACM workshop on Role-based access control - RBAC '97*, pages 13–19, New York, New York, USA, novembre 1997, ACM Press.

[Tillett 01] B. Tillett, « Bibliographic Relationships », *in* C. A. Bean & R. Green (dir.), *Relationships in the Organization of Knowledge*, page 23, Kluwer Academic Publishers, Boston, 2001.

[Tillett 03] B. Tillett, « What is FRBR ? A conceptual model for the Bibliographic Universe », *Technicalities*, vol. 25, n° 5, 2003.

[Tolone *et al.* 05] W. Tolone, G.-J. Ahn, T. Pai & S.-P. Hong, « Access control in collaborative systems », *ACM Comput. Surv.*, vol. 37, 29–41, March 2005.

[Van Kesteren 10] A. Van Kesteren, « Access Control API Specification Working Draft », Rapport technique, W3C, Juillet 2010.

[Vora 98] P. R. Vora, « Design/Methods & Tools : Designing for the Web : a survey », *interactions*, vol. 5, 13–30, May 1998.

[Zeng *et al.* 10] M. L. Zeng, M. Žumer & A. Salaba, *Functional Requirements for Subject Authority Data*, K. G. Saur, München, Germany, 2010.