N° d'ordre :

REPUBLIQUE ALGERIENNE DEMOCRATIQUE & POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR & DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DJILLALI LIABES
FACULTE DES SCIENCES EXACTES
SIDI BEL ABBÈS

THESE DE DOCTORAT

Présentée par

BOUOUGADA Benamar

Spécialité : Informatique
Option : Interopérabilité et intégration des systèmes d'informations dans le Web

Intitulée

La Réingénierie des applications Web vers les données liées

Soutenue le 17 octobre 2019
Devant le jury composé de :

Président :	*Sofiane BOUKLI HACENE,*	*MCA,*	*UDL-SBA*
Examinateurs :	*Ghalem BELALEM*	*Professeur*	*Université d'Oran*
	Reda ADJOUDJ,	*MCA,*	*UDL-SBA*
	Moussa ALI CHERIF	*MCA*	*UDL-SBA*
Directeur de thèse :	*Djelloul Bouchiha*	*Professeur*	*CU-Naama*
Co-directeur de thèse :	*Mimoun MALKI*	*Professeur*	*UDL-SBA*

Année universitaire 2019/2020

Je dédie ce travail
À mon père
À ma femme
À mon fils ABDELNOUR
À tous mes frères
À toutes mes sœurs
À ma belle-mère
À toute ma famille
À tous ceux qui m'ont encouragé et soutenu

Remerciements

Cette thèse n'aurait pas été possible sans l'intervention et l'aide d'un grand nombre de personnes. Nous souhaitons ici les en remercier.

Tout d'abord personne ne mérite d'être remercié avant le bon DIEU pour son aide afin de réaliser ce travail.

Je voudrais d'abord remercier très chaleureusement Mr Bouchiha Djeloul qui m'a permis de bénéficier de son encadrement. Les conseils qu'il m'a prodigués, la patience, la confiance qu'il m'a témoignée ont été déterminants dans la réalisation de ma thèse de doctorat.

Je voudrais également adresser mes sincères remerciements à Mr MALKI MIMOUN pour ses aides et ses remarques et conseils de fond tout au long de ce travail.

Mes vifs remerciements vont également au président de jury et aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner mon travail et de l'enrichir par leurs propositions.

Je voudrais remercier tous mes collègues du département de mathématiques et d'informatique du Centre universitaire de NAAMA, en particulier Mr. Bouziane Abdelghani et Mr. NAIMA Khatir.

Mes sincères remerciements à mon père, toute ma famille, en particulier mon épouse qui m'a beaucoup soutenu à accomplir ce travail.

Enfin, je tiens à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Naama, le 26 octobre 2019.

Table des matières

Table des matières **iv**

Liste des figures **vii**

Liste des tableaux **ix**

Introduction générale **1**
- 1.1 Contexte . . . 1
- 1.2 Problématique . . . 1
- 1.3 Objectif . . . 2
- 1.4 Organisation de la thèse . . . 3

I Background **4**

2 Application Web **5**
- 2.1 Introduction . . . 6
- 2.2 Définition d'une application Web . . . 6
- 2.3 Avantages d'une application Web . . . 6
  - 2.3.1 Par rapport aux internautes . . . 6
  - 2.3.2 Par rapport aux développeurs . . . 7
- 2.4 Types d'une application Web . . . 7
  - 2.4.1 Classification selon l'architecture . . . 7
  - 2.4.2 Classification selon le fonctionnement . . . 9
- 2.5 Cycle de développement d'une application Web . . . 9
- 2.6 Historique des applications Web . . . 11
- 2.7 Evolution du Web . . . 12
  - 2.7.1 Web statique . . . 13
  - 2.7.2 Web dynamique . . . 13
  - 2.7.3 Web semi-structuré . . . 15
  - 2.7.4 Web sémantique . . . 16
- 2.8 Les langages Web . . . 16
  - 2.8.1 Le langage HTML . . . 16
  - 2.8.2 Le Langage PHP . . . 18
- 2.9 Conclusion . . . 20

3 Web sémantique **21**
- 3.1 Introduction . . . 22
- 3.2 Historique . . . 22
- 3.3 Langages du Web sémantique . . . 23
  - 3.3.1 Le langage XML (Extensible Markup Language) . . . 24

3.3.2 Le schéma XML (XML schema) . . . 27
3.3.3 Conflit de nommage dans XML . . . 29
3.3.4 Espace de nommage XML (Namespace) . . . 29
3.4 Les données Liées RDF (Ressource Description Framework) . 30
3.4.1 Les concepts de base de RDF . . . 30
3.4.2 Exemple d'un graphe RDF . . . 32
3.4.3 Sérialisation . . . 32
3.5 Vocabulaire de RDF ou Schéma RDF . . . 33
3.5.1 Les classes . . . 34
3.5.2 Les propriétés . . . 34
3.5.3 Autre concepts supplémentaires . . . 35
3.6 Le Langage Web d'ontologie OWL . . . 36
3.6.1 Les types d'OWL . . . 36
3.7 Le langage SPARQL . . . 41
3.8 Conclusion . . . 42

4 Ingénierie Dirigée par les Modèles 44
4.1 Introduction . . . 45
4.2 Définition de MDA . . . 46
4.2.1 Pourquoi MDA ? . . . 46
4.3 Architecture de l'approche MDA . . . 46
4.3.1 Le modèle d'exigences CIM (Computation Independent Model) . 47
4.3.2 Le modèle abstrait d'analyse et de conception PIM (Platform Independent Model) . . . 48
4.3.3 Le modèle concret de code ou de conception PSM (Platform Specific Model) . . . 48
4.4 Transformation des modèles . . . 49
4.5 Les Modèles et les Méta-modèles . . . 49
4.5.1 Les modèles . . . 49
4.5.2 Les méta-modèles . . . 49
4.5.3 Relation entre un modèle et son méta-modèle . . . 50
4.5.4 Méta méta-modèle . . . 50
4.6 Architecture à quatre niveaux de l'IDM . . . 51
4.7 Les Langages de modélisation . . . 52
4.7.1 UML . . . 52
4.7.2 MOF 2.0 . . . 54
4.7.3 Le Framework de modélisation d'Eclipse (Eclipse Modeling Framework EMF) . . . 54
4.8 Les modèles en XML . . . 56
4.8.1 Le format XML . . . 56
4.8.2 Le standard XMI (XML Metadata Interchange) . . . 56
4.8.3 DI (Diagram Interchange) . . . 59
4.9 Les Langages de transformation des modèles dans l'IDM . . . 60
4.9.1 Le langage QVT (Query/View/Transformation) . . . 60
4.9.2 Le langage de transformation ATLAS (ATL . . . 62
4.10 Conclusion . . . 63

II Réingénierie des applications Web vers les données liées & ontologies **65**

5 Etat de l'art **66**
5.1 Introduction . . . 67
5.2 Mappage des ressources Web vers RDF . . . 67
5.3 Génération des graphes RDF à partir des données relationnelles . . . 70
5.4 Génération d'ontologie à partir de differentes sources de données . . . 74
5.5 Conclusion . . . 76

6 Approche I **77**
6.1 Introduction . . . 78
6.2 Exemple d'exécution . . . 78
6.3 Approche proposée . . . 79
6.3.1 Étape de pretraitement . . . 80
6.3.2 Étape de réingénierie basée sur l'IDM . . . 81
6.3.3 Étape de raffinement . . . 88
6.4 Evaluation . . . 90
6.4.1 L'outil implémenté . . . 90
6.4.2 Expérimentations et discussion des résultats . . . 91
6.5 Conclusion . . . 92

7 Approche II **94**
7.1 Introduction . . . 95
7.2 Approche proposée . . . 95
7.2.1 La phase de Rétro-ingénierie . . . 96
7.2.2 La Phase d'ingénierie directe . . . 98
7.3 Exemple de déroulement . . . 100
7.4 Expériences et discussion des résultats . . . 103
7.5 Conclusion . . . 105

Conclusion générale **106**
8.1 Conclusion générale . . . 106
8.2 Perspectives . . . 107

Bibliographie **108**

Liste des figures

2.1 Architecture d'une application Web statique 8
2.2 Architecture d'une application Web dynamique 8
2.3 Cycle de vie d'une application Web 10
2.4 Architecture du Web statique 13
2.5 Architecture du Web dynamique 14

3.1 La pile du Web sémantique de Tim Berners-Lee 23
3.2 Représentation graphique d'un triplet RDF 30
3.3 Exemple d'un graphe RDF 32
3.4 Triplet RDF simples 42
3.5 Graphe d'une requête SPARQL simple 42

4.1 Le méga-modèle de Favre (L. Favre,2010) 45
4.2 Architecture de l'approche MDA (Xavier,2005) 47
4.3 Relations entre un modèle (diagramme de cas d'utilisation) et son méta-modèle (Xavier, 2005) 50
4.4 Le méta méta-modèle MOF 1.4 (Xavier, 2005) 51
4.5 Architecture à quatre niveaux (Xavier, 2005) 52
4.6 Les diagrammes UML (OMG, 2004) 53
4.7 Représentation schématique du méta méta-modèle MOF2.0 (Xavier, 2005) 54
4.8 Méta méta-modèle ECORE (Xavier, 2005) 55
4.9 Alignement entre méta-modèle/modèle et DTD/document XML (Xavier, 2005) 57
4.10 XMI et la structuration de balises XML (Xavier, 2005) 57
4.11 Méta-modèle de processus (Xavier, 2005) 58
4.12 Principe de fonctionnement de DI (Xavier, 2005) 60
4.13 Relation entre les méta-modèles QVT (OMG, 2016) 60

5.1 Framework d'extraction de DBpedia (Lehmann et al., 2014) 68
5.2 Aperçu des composants (Konstantinou et al., 2014) 71
5.3 Architecture de Datalift (Scharffe et al., 2012) 72

6.1 Transformation des documents HTML en fichiers RDF 78
6.2 Exemple de transformation 79
6.3 Architecture du système 80
6.4 Prétraitement : Exemple d'extraction et d'adaptation des données . 81
6.5 Processus de réingénierie basé sur les modèles 82
6.6 Le méta-modèle source (MMS) 83
6.7 Liste des données écrites par les balises du MMS 85
6.8 Le méta-modèle cible MMC 85

6.9 Organigramme de raffinement 89
6.10 L'interface principale de l'outil *HTML2RDF* 90
6.11 Graphe d'évaluation. 92

7.1 Génération d'ontologie à partir une application Web 95
7.2 Organigramme de rétro-ingénierie 96
7.3 Mapping du diagramme de classes en Ontologie 98
7.4 Diagramme de classes généré par le processus de retro-ingénierie . 101
7.5 Interface de l'outil PHP2OWL 103
7.6 Graphe d'évaluation. 104

Liste des tableaux

2.1 Les Balises HTML. 17
2.2 Instructions PHP. 19

3.1 Les littéraux de type booléenne 31
3.2 Les constructeurs de OWL Lite 36
3.3 Les constructeurs OWL DL et Full 40

5.1 Approches pour mapper des ressources Web en données liées RDF 70
5.2 Approches de transformation des données relationnelles en RDF . 73
5.3 Résumé des travaux d'ingénierie des ontologies à partir de différentes sources de données 76

6.1 Les balises défini par MMS 84
6.2 Mappage des éléments du MMS vers des éléments du MMC . . . 88
6.3 Résultant de l'évaluation. 91

7.1 Eléments du langage de sérialisation 97
7.2 Résultats d'évaluation 103

Liste des listings

2.1 Structure d'un document HTML 17
3.1 Exemple d'un élement XML contenant d'attribut 24
3.2 Exemple d'un document XML bien formé 25
3.3 Syntaxe geénérale d'un élément complexe 28
3.4 Syntaxe d'un élément complexe contient des valeurs simple et attributs 29
3.5 RDF sérialisé en Turtle 33
3.6 RDF sérialisé en XML 33
3.7 Requête SPARQL simple 42
4.1 La DTD générée à partir du méta-modèle de processus 58
4.2 Exemple d'un modèle de processus sérialisé en document XML . . 59
6.1 La règle de transformation de l'élément Table en une ressource RDF imbriquée 86
6.2 La règle de transformation d'une liste ordonnée en une ressource RDF 86
6.3 La règle de transformation d'une liste de définition en une ressource RDF imbriquée. 87
6.4 La règle de transformation d'un lien en une ressource RDF. 87
6.5 Le fichier RDF généré par le système. 89
6.6 Le fichier RDF après le raffinement. 90
7.1 Extrait du code source PHP 100
7.2 Diagramme de classes sérialisé en XML 101
7.3 Extrait de l'ontologie générée 101
7.4 Instances du concept "Annonce" en format RDF 102

Introduction générale

1.1 Contexte

Actuellement, le Web est un ensemble de documents interconnectés contenant une quantité de données. Ces données sont souvent représentées dans des pages Web HTML sous forme de tableaux ou de listes. Selon [Cafarella *et al.* 2008], il existe des milliards de tableaux stockant des données sur le Web. Dans la plus part des cas, ces données sont générées automatiquement à partir de bases de données.
Une grande quantité de bases de données est accessible actuellement à travers le Web. Cependant, il n'y a pas un accès public direct aux bases de données elles-mêmes [Nagy *et al.* 2011]. Seuls des documents HTML liés sont construits dynamiquement et présentés à l'utilisateur.
RDF (Resource Description Framework) [1] offre un mécanisme efficace pour représenter directement des données sur le Web. On parle de Données Liées (Linked Data), discutées en tant que technologie du Web sémantique [Berners-Lee 2006]. La notion Données Liées fournit un paradigme de publication dans lequel non seulement les documents, mais aussi les données, peuvent être un citoyen de première classe dans le Web [Heath & Bizer 2011].
Un nombre important d'applications Web sont actuellement opérationnelles, permettant de manipuler une grande quantité de bases de données. Pour permettre un accès direct à ces données sur le Web sans les créer à nouveau comme Données Liées RDF, un processus de réingénierie peut être appliqué à des applications Web, afin de générer des Données Liées RDF.
Selon [Chikofsky & Cross 1990], la reconstruction d'un système sujet sous une nouvelle forme nécessite un processus de réingénierie. La réingénierie comprend généralement deux phases : la rétro-ingénierie et l'ingénierie directe. La rétro-ingénierie est le processus d'analyse d'un système sujet pour identifier ses composantes et leurs interrelations, et créer des représentations du système sous une autre forme ou à un niveau d'abstraction plus élevé. L'ingénierie directe est le processus traditionnel permettant de passer d'un niveau d'abstraction élevé avec des modèles conceptuels, à l'implémentation du système.

1.2 Problématique

Les bases de données étant situées sur des serveurs, il n'est pas possible d'y accéder directement via le Web. Par conséquent, le seul accès possible aux données se fait via des pages HTML. Cependant, en raison de leur format, il est très difficile et presque impossible aux machines de traiter les données HTML d'une façon autonome. Afin de rendre les données Web plus exploitables

1. http://www.w3.org/TR/rdf-primer/

et interprétables par une machine, nous devons les présenter sous une forme plus adéquate telle que les données liées RDF [Berners-Lee 2006]. Il s'agit donc d'une réingénierie des applications Web existant vers les données liées, ou tout simplement une translation du Web classique vers le Web sémantique.
La communauté du Web sémantique a relevé plusieurs défis de recherche tels que l'alignement des ontologies [Ardjani *et al.* 2015], la maintenance des liens RDF [Ardjani *et al.* 2017] et la réingénierie des applications Web [Mulwad *et al.* 2010, Embley *et al.* 2011, Nagy *et al.* 2011, Mulwad *et al.* 2013, Lehmann *et al.* 2014, Munoz *et al.* 2014]. Dans cette thèse, nous nous intéressons à la réingénierie des applications Web vers le Web sémantique. Les applications Web consistent en un ensemble de pages HTML et de scripts PHP. Nous nous concentrons sur la transformation des pages HTML car elles sont riches en données. Nous utilisons également les pages PHP pour extraire le schéma de données.

1.3 Objectif

En pratique, nous appliquons un processus de réingénierie pour transformer une application Web en une application Web sémantique. Par conséquent, nous autorisons la publication de données RDF sur le Web. Pour aboutir à cet objectif, nous proposons deux approches : l'une pour générer les données d'une application Web sous forme de triplets RDF, et l'autre pour générer le schéma de données sous forme OWL.
La première approche est basée sur l'ingénierie dirigée par les modèles (IDM) [Bouougada *et al.* 2018]. Elle permet de transformer une application Web en données liées RDF. Elle reçoit en entrée des pages HTML (pages clientes) et produit en sortie des triplets RDF. L'approche proposée comprend trois étapes : (1) Prétraitement, qui consiste à préparer les entrées HTML pour l'étape suivante, (2) Transformation, qui est un processus de réingénierie selon les principes de l'IDM, et (3) Raffinement, qui consiste à affiner les résultats préliminaires pour avoir des résultats RDF finaux. Cette approche est supportée par un outil appelé HTML2RDF.
La deuxième approche à pour objectif de transformer les applications Web en ontologie OWL [Bouougada *et al.* 2017]. L'ontologie est l'un des concepts de base du Web sémantique. [Gruber 1993, p. 2] définit l'ontologie comme une spécification explicite d'une conceptualisation. [Borst 1997, p. 12] affirme que la conceptualisation doit exprimer une vue partagée entre plusieurs parties et doit être exprimée dans un format lisible par machine, de sorte qu'elle définit l'ontologie comme une spécification formelle d'une conceptualisation partagée. [Studer *et al.* 1998, p. 25] définissent l'ontologie comme une spécification formelle et explicite d'une conceptualisation partagée ; cela fusionne les deux définitions précédentes [Guarino *et al.* 2009].
Dans la littérature, plusieurs travaux visent à créer des ontologies. Ce processus s'appelle également l'ingénierie des ontologies. Certaines approches sont basées sur un langage naturel contrôlé (LNC), contenus Wiki, utilisateurs et experts, telles que [Kaljurand 2008, Denaux *et al.* 2013, Kuhn 2013a, Kuhn 2013b]. D'autres s'appuient sur des bases de données relationnelles, telles que [Sequeda *et al.* 2012, Ougouti *et al.* 2015]. Bien que les travaux cités utilisent divers sources de données pour créer des ontologies, ils restent insuffisants pour définir des ontologies et la supervision par un expert de domaine est toujours

nécessaire.
Nous avons constaté qu'un nombre important d'applications Web classiques (applications Web non sémantiques) sont déjà en cours d'utilisation. Pour tirer parti de cette situation en faveur du Web sémantique, nous appliquons un processus de rétro-ingénierie (reverse engineering en anglais) pour récupérer la conception des applications Web. Notre objectif était donc de réutiliser les applications Web pour la création d'ontologies en les présentant dans un haut niveau d'abstraction, en se basant sur l'analyse du code source. Ensuite, nous appliquons un processus d'ingénierie directe pour créer l'ontologie et ses instances qui représentent le domaine de l'application en question. D'autre part, ces applications Web deviennent compatibles avec la technologie du Web sémantique.
L'approche proposée commence par un processus de rétro-ingénierie qui vise à récupérer la conception originale à partir du code source de l'application Web, en particulier les scripts HTML et PHP, en utilisant des techniques de compréhension du programme. Ensuite, un processus d'ingénierie direct pour créer une ontologie OWL à partir des diagrammes récupérés en appliquant un ensemble de règles de transformation.

1.4 Organisation de la thèse

Le reste du manuscrit est organisé en deux grandes parties. Dans la première partie, on présente toutes les notions nécessaires à la compréhension du problème traité dans cette thèse. La deuxième partie est consacrée aux deux approches proposées pour atteindre les objectifs visés.
La *première partie* consiste en trois chapitres : Le *premier chapitre* présente la notion d'application Web et ses déférents concepts de base. Le *deuxième chapitre* est consacré au Web sémantique, là où des détails sur les langages du Web sémantique notamment RDF, RDFS et OWL sont introduits. Dans le *troisième chapitre*, nous allons présenter toutes les concepts de l'ingénierie dirigée par les modèles (IDM). Nous définissons c'est quoi un modèle, méta-modèle, méta méta-modèle, les transformations entre les modèles, etc.
La *deuxième partie*, elle aussi consiste en trois chapitres : Le *quatrième chapitre* donne un état de l'art sur le problème traité. Dans le *cinquième chapitre* nous allons présenter la première approche, à travers laquelle nous pouvons transformer une application Web vers les données liées. Le *sixième chapitre* présente la deuxième approche qui permet de générer une ontologie à partir d'une application Web.
Le manuscrit est clôturé par une conclusion générale récapitulant et synthétisant tout le travail. Des perspectives ont été aussi données à ce niveau, initiant à la continuité du travail.

Première partie

Background

Application Web 2

Sommaire

2.1 Introduction . . . 6
2.2 Définition d'une application Web . . . 6
2.3 Avantages d'une application Web . . . 6
2.3.1 Par rapport aux internautes . . . 6
2.3.2 Par rapport aux développeurs . . . 7
2.4 Types d'une application Web . . . 7
2.4.1 Classification selon l'architecture . . . 7
2.4.2 Classification selon le fonctionnement . . . 9
2.5 Cycle de développement d'une application Web . . . 9
2.6 Historique des applications Web . . . 11
2.7 Evolution du Web . . . 12
2.7.1 Web statique . . . 13
2.7.2 Web dynamique . . . 13
2.7.3 Web semi-structuré . . . 15
2.7.4 Web sémantique . . . 16
2.8 Les langages Web . . . 16
2.8.1 Le langage HTML . . . 16
2.8.2 Le Langage PHP . . . 18
2.9 Conclusion . . . 20

2.1 Introduction

Actuellement, le Web n'est plus seulement un ensemble de sites Web statiques qui contiennent des pages interconnectées entre elles, mais aussi un ensemble d'applications Web pleines de données, photos, vidéos, etc. Cette technologie permet de rendre le Web plus utile et plus attrayant. Comme exemple d'applications Web on cite les messageries, les réseaux sociaux, les achats en ligne, etc. Pratiquement, les applications Web sont devenues indispensables, et elles sont utilisées par toutes les catégories d'utilisateurs.
Les applications Web pallient le problème de sites Web statiques qui ne gèrent que des contenus statiques. À travers les applications Web, les utilisateurs peuvent recevoir différentes réponses de différentes requêtes d'une façon dynamique.
Ce chapitre est consacré à la description des applications Web, des langages Web, ainsi que les différentes étapes d'évolution du Web.

2.2 Définition d'une application Web

Pour un utilisateur Web, il est difficile de percevoir la différence entre un site Web (site internet) et une application Web. Un site Web statique constitue un outil de communication, contrairement à une application Web qui facilite la réalisation d'une activité bien précise. Le mot "application" en informatique désigne : un programme ou un logiciel destiné à l'utilisateur afin de l'aider pour accomplir un traitement précis. La même définition est valable pour une application Web, sauf que, cette dernière utilise les technologies du Web tels que les navigateurs Web, les serveurs Web, le protocole HTTP, les langages Web, etc. Donc, on peut conclure qu'une application Web est un logiciel applicatif de type client/-serveur, où elle est hébergée sur un serveur Web et accessible via un navigateur Web. D'où, une application Web n'a pas besoin d'une application cliente ; il suffit d'utiliser un navigateur Web pour y accéder [Véronneau 2016].
Les technologies utilisées pour développer les applications Web sont les mêmes que celles employées dans le développement des sites Web. En plus, d'autres langages dédiés sont utilisés.

2.3 Avantages d'une application Web

Lors de l'apparition des applications Web, les développeurs et les internautes ont tous bénéficié de ses avantages. Dans cette section, nous en résumons quelques-uns [1].

2.3.1 Par rapport aux internautes

- Le budget peut être contrôlé et diminué car il n'y a plus de frais supplémentaires à payer, ni pour la mise à niveau, ni pour l'adaptation, ni pour la maintenance, ni pour l'amélioration. Tous ces frais sont pris en charge par les fournisseurs. Ils paient seulement des frais d'abonnement quand il s'agit des applications payantes.

1. Dictionnaire du digital : https://www.ideematic.com/dictionnaire-digital/application-web/

- L'échange de données entre utilisateurs est plus rapide.
- Réduction de l'intervention des informaticiens puisque l'utilisation des applications Web est plus intuitive et plus facile.
- Disponibilité des données. Les données sont presque éternelles.
- Aucune restriction d'accès aux applications Web, que ce soit spatiale ou matérielle. un utilisateur peut accéder aux applications à n'importe quel endroit et avec n'importe quel appareil.
- La sécurité est garantie par les contrôles d'accès aux données, telles que les identifications et les certificats. De plus, le stockage de données d'une façon confidentiel participe aussi à la sécurité des données.
- Toujours profiter des nouveautés des applications grâce leurs évolutions continues.
- Les remarques sont tenues en compte pour améliorer et innover.

2.3.2 Par rapport aux développeurs

- Le développement et le déploiement sont devenus très rapides, ce qui donne un gain de temps.
- Les données sont centralisées, ce qui est avantageux aux développeurs lorsqu'il y a un problème.
- Compatibilité des systèmes, car il n'y a pas d'installation des applications dédiées au niveau des clients. Il suffit que chaque client ait un navigateur pour qu'il puisse bénéficier du service.
- Excellente gestion de sécurité grâce à l'exploitation des services des infrastructures Web existantes.
- Facilité des mises à niveau des applications en les rendant transparentes et automatiques.

2.4 Types d'une application Web

Selon l'architecture des applications Web, on distingue généralement deux types : applications Web statiques et applications Web dynamiques. Cependant, on peut faire une autre classification selon leurs fonctionnements, et là on distingue les applications e-commerces, les applications portails, les applications animées, les applications de type CMS [Yeeply 2018].

2.4.1 Classification selon l'architecture

Les applications Web sont classées selon l'architecture comme suit :

Les applications Web statiques

Une application Web est dite statique si et seulement si son contenu est produit d'une façon statique. La figure 2.1 montre l'architecture d'une telle application. Elle se base sur une architecture client/serveur : le client envoie sa requête HTTP au serveur. Le serveur Web traite la requête et retourne la réponse sous forme d'un document HTML au client. Le rôle du serveur se limite à la localisation de la page demandée par le client, il ne fait aucun autre traitement.

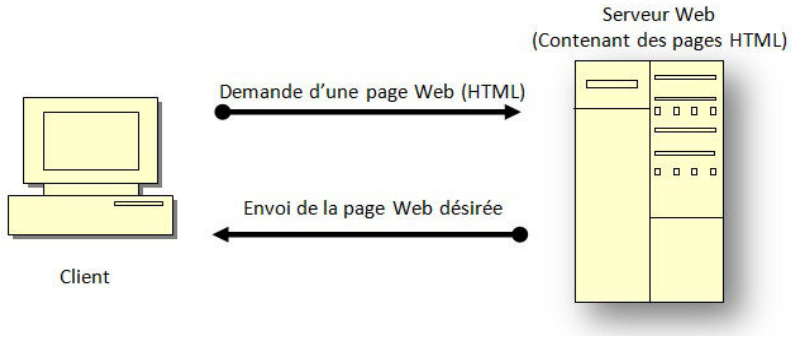

FIGURE 2.1 – *Architecture d'une application Web statique*

Ce type d'application est généralement développé en utilisant des langages Web qui s'exécutent au niveau du client, tel que HTML, CSS et Javascript. Néanmoins, le développement peut se faire aussi par Jquery et Ajax dans le cas où l'application possède des objets animés, tels que bannières, GIF, vidéos, etc. Notez que la gestion de contenu des applications statiques est difficile.

Les applications Web dynamiques

Contrairement aux applications Web statiques, le contenu des applications Web dynamiques est mis à jour automatiquement, et ce à chaque demande d'utilisateur. La figure 2.2 donne un aperçu de l'architecture des applications Web dynamiques. Elle se repose sur l'architecture n-tiers. Pour chaque requête HTTP envoyée au serveur Web, le client reçoit un document HTML.

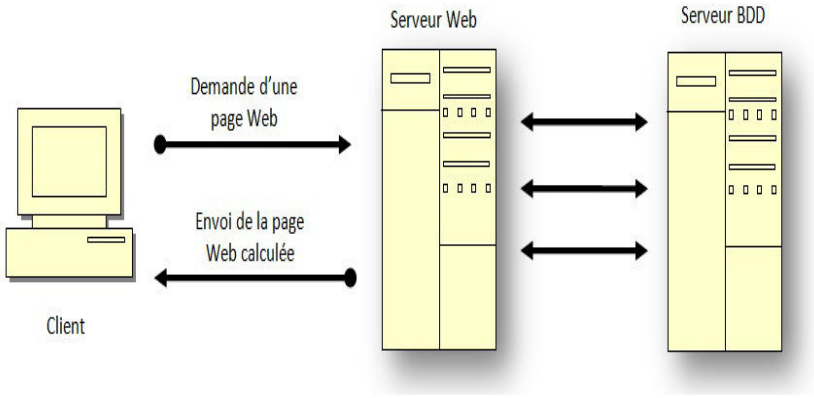

FIGURE 2.2 – *Architecture d'une application Web dynamique*

Le serveur Web résout la requête HTTP comme suit : premièrement, il localise le fichier script (PHP, ASP, etc.) stocké sur son disque dur. Deuxièmement, il délègue le logiciel approprié à l'exécution du script. Troisièmement, le résultat du traitement du fichier script est placé dans un document HTML qu'il l'envoie au client.
Dans la plupart des cas, la génération d'un document HTML par le serveur nécessite d'autres applications pour extraire des informations à partir d'une base de données. Ici, l'application de script fait appel au système de gestion de bases de données pour obtenir les informations nécessaires. Une fois ces informations

obtenues, le script les met dans un document HTML qu'il l'adresse au client à travers le serveur Web.

2.4.2 Classification selon le fonctionnement

Les applications Web sont classées selon le fonctionnement comme suit :

Application Web e-commerce

Ce sont des applications Web dynamiques caractérisés par un traitement supplémentaire dans le but d'assurer le paiement électronique soit par carte de crédit ou d'autres modes de paiement. Les développeurs doivent prendre ces exigences en compte. En plus de ça, ils doivent offrir un moyen de gestion de l'application à l'administrateur pour qu'il puisse mettre en vente des produits, les mettre à jour et gérer les commandes.

Application Web portail

Ces applications fournissent plusieurs services dans leurs pages d'accueil. Ils permettent l'accès, par exemple, à d'autres catégories d'applications, telles que les emails, les moteurs de recherche, les forums, les chats, les formulaires d'enregistrement, etc.

Application Web animée

Ces applications utilisent la technologie de FLASH pour créer et gérer des contenues animés. La technologie de FLASH rend les applications plus créatives avec un design moderne, mais elle empêche les moteurs de recherche de lire correctement les informations.

Application Web CMS

Les applications de ce type nécessitent des mises à jour régulières de leurs contenus. Les administrateurs gèrent ce type d'application par un système de gestion de contenu appelé CMS (Content Management System, CMS). Les systèmes CMS sont intuitifs et très faciles à utiliser, parmi eux : WordPress, Joomla et Drupal.

Application Web de gestion

Ce type d'application fait partie des applications Web dynamiques. Les applications de gestion sont destinées à un usage particulier selon les besoins des clients. Elles regroupent tous types de gestion tels que gestion des étudiants, gestion de la bibliothèque, gestion de paiement, etc. Elles permettent de manipulier les données des clients afin de faciliter la gestion de leurs tâches.

2.5 Cycle de développement d'une application Web

Dans la littérature, il n'existe pas un modèle précis pour présenter le cycle de vie des applications Web, mais on peut s'appuyer sur les modèles de cycles de vie des systèmes d'informations traditionnels, et les modèles de conception des

systèmes hypermédias [Garzotto *et al.* 1995]. La Figure 2.3 montre le cycle de vie d'une application Web [Fraternali 1999].

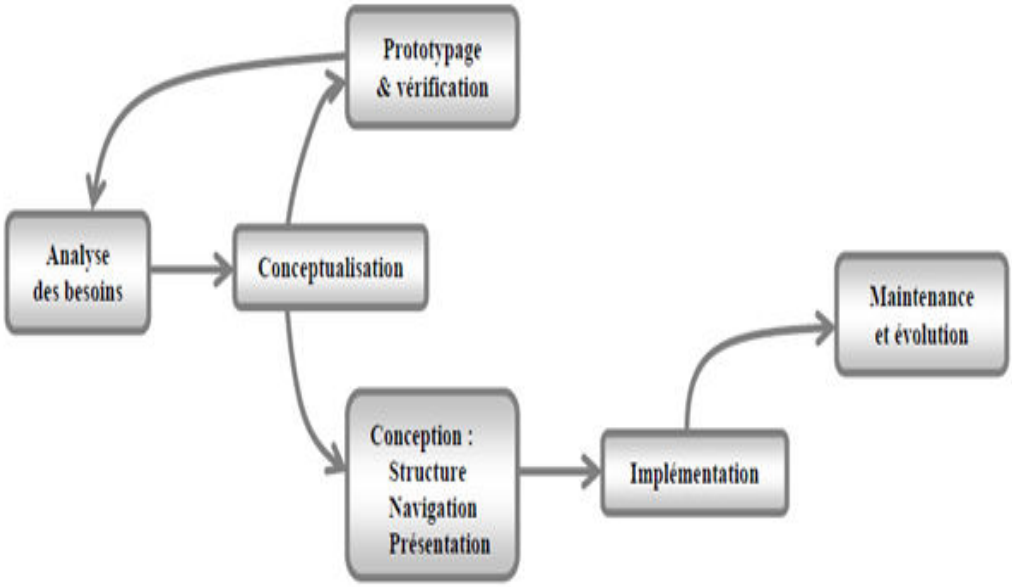

FIGURE 2.3 – *Cycle de vie d'une application Web*

Analyse des besoins : cette étape est très importante. Elle n'est pas facile comme pour les applications classiques, là où les utilisateurs sont bien connus. Il faut d'abord identifier les utilisateurs éventuels de l'application, puis déterminer leurs besoins. Parmi les besoins qu'il faut tenir en compte par les développeurs, on cite l'aspect ergonomique de l'application car l'application Web doit avoir un mode interactif approprié à chaque catégorie d'utilisateur, et pour chaque dispositif utilisé pour accéder à l'application.
Conceptualisation : dans les systèmes d'information, la phase de conception représente la représentation de l'application par un ensemble de modèles reflétant les différentes composantes clés de la solution envisagée. Par contre, la conception dans le contexte d'une application Web a une vue orientée utilisateur, c'est-à-dire, qu'il est nécessaire de se concentrer sur la manière dont les objets et leurs relations apparaissent aux utilisateurs. On peut utiliser les mêmes modèles pour la conception des applications Web et pour les applications de bases de données, comme on peut utiliser des modèles différents.
Prototypage et vérification : la complexité des interfaces et de développement requière une évaluation préalable de la structure, de la navigation, et de la présentation pour l'application (Nielsen, 1997). Cette étape permet de réaliser cette évaluation qui est très importante lors du développement d'une application Web. Elle permet de connaître les opinions et les comportements des utilisateurs avant que l'application soit construite, et ce par la mise en place d'un prototype aux utilisateurs.
Conception : c'est l'étape de transformation des schémas obtenus à partir de l'étape précédente en d'autres représentations de bas niveau, où la vue structurelle est transformée en un schéma de base de données, la vue de navigation est transformée en un ensemble de primitifs d'accès au contenu de base de données, et la vue de présentation est transformée en un ensemble de spécifications visuelles [Sano 1996]

Implémentation : c'est l'étape de réalisation de l'application Web en utilisant tous les modèles des étapes précédentes, donc c'est le passage des modèles conceptuels vers les modèles concrets (codes source). Dans cette étape, selon le type de l'application Web, les développeurs doivent choisir la plateforme de développement, les langages de développement Web et le mode de connexion entre les éléments de l'application (online ou offline).
Evolution et maintenance : la maintenance permet la correction des erreurs. Elle peut intervenir dans différentes étapes du cycle de vie de l'application Web. On distingue trois types de maintenance : (1) la maintenance corrective permet de corriger les erreurs, (2) la maintenance adaptative permet l'adaptation et l'évolution de l'application à l'apparition de nouvelles contraintes par les utilisateurs, et (3) la maintenance perfective qui a pour objectif d'optimiser les performances. Nous observons que les développeurs ont besoin d'une bonne compréhension de données et de flux de contrôle dans le développement d'une application Web par rapport à une application traditionnelle. Malheureusement, les outils de développement des applications Web actuels sont plus concentrés sur la rapidité de réalisation. Cette concentration est attribuée à l'allure rapide de leur développement. Ainsi, le besoin de la rétro-ingénierie des applications Web se fait sentir afin de faciliter leur maintenance, leur évolution et leur migration vers de nouvelles technologies (e.g. Web sémantique).

2.6 Historique des applications Web

Dans cette section, nous allons donner les profils historiques de certains langage de développement des applications Web [Véronneau 2016].
Larry Wall en 1987, a inventé le langage "Perl". Perl est l'un des premiers langages de programmation pour le développement d'applications Web avant qu'elles soient à la portée du grand public. Du coup, on peut conclure que le concept d'application Web n'est pas un nouveau.
Rasmus Lerdorf a participé au développement des applications Web, et ce en 1995 lorsqu'il a rendu le langage PHP à la disponibilité de tout le monde. Ce qui a permis de développer et de diffuser des applications rapidement et de manière significative jusqu'à nos jours. Beaucoup d'applications célèbres ont été développées par PHP y compris Facebook, Wikipedia, Google, etc.
Dans la même année, le navigateur Netscape offrirait aux développeurs un moyen pour gérer le contenu d'une page Web d'une façon dynamique grâce au langage "JavaScript". Ce langage était et est encore beaucoup plus interactif pour les usagers. Il a vraiment participé à la progression des applications Web.
En 1996, Sabeer Bhatia et Jack Smith ont lancé "Hotmail". Hotmail est un service en ligne de messagerie destiné au grand public, et c'est la première fois que le public bénéficie de ce genre de services. Il permet aux utilisateurs où qu'ils soient d'accéder, de consulter, de partager des informations à travers des courriels.
En 1997, Shockwave Flash apparait. Il est considéré comme première plateforme Flash. Il a été acquis par Macromedia puis par Adobe. Cette plateforme a aussi un grand impact sur le développement des applications Web interactives. Il sert à améliorer et ajouter des contenus interactifs aux applications Web.
En 1998, le site "The Drudge Report" diffusa un événement journalistique sur le Web avant d'être diffusé par d'autres médias, tels que la télévision et les journaux, et c'était pour la première fois dans le Web. L'événement publié a été une étincelle qui a déclenché le média en ligne, car avant cette date, l'objectif n'était

pas un média de presse.
Dans la même année, la mise en ligne d'un moteur de recherche par la compagnie "Google". Ce moteur de recherche facilita la recherche d'information sur Internet par un nouveau mécanisme d'indexation des pages Web. Aujourd'hui, la compagnie de "Google" est devenue plus productive en termes d'applications Web (Google Maps, Google Docs, Gmail, etc.).
En 2001, le projet "Nupedia" de Jimmy Wales lança un sous-projet appelé "Wikipedia" sous forme d'encyclopédie, pour laquelle Larry Sanger est devenu son éditeur en chef. Ce sous-projet utilisa une application Web intitulée "Wiki" permettant aux internautes d'ajouter et de modifier les contenus. Aujourd'hui, Wikipedia est devenue une encyclopédie universelle multilingue, contenant de millions d'articles. C'est l'un des sites les plus visités au monde.
En 2003, la mise en place de site "MySpace" qui devint quelques années plus tard le site social le plus visité, et c'était entre 2005 et 2008. Le site MySpace a eu beaucoup de modules supplémentaires, tels que YouTube, Slide.com et RockYou, qui sont devenus actuellement, des applications Web eux-mêmes.
En 2004, trois événements de grande importance ont émergé, commençons par la conférence Web 2.0 présentée par John Batelle et Tim O'Reilly, sous le concept du « Web en tant que plateforme ». Ce concept prépara le terrain aux applications Web de prendre leur place dans la venir.
Le lancement du site "Digg" fut marqué le deuxième évènement en 2004. C'est un site interactif qui offrirait une nouvelle façon aux internautes afin de générer et trouver du contenu sur le Web en faisant une promotion des nouvelles et des liens. Le troisième événement est assez important, c'est le lancement du site de média social Facebook. Au début, il était limité dans son utilisation uniquement pour les étudiants. Par contre, aujourd'hui, il est devenu le site de média social le plus utilisé et le plus visité, avec plus de 900 millions d'utilisateurs.
En 2005, le lancement officiel de l'application Web "Youtube". Aujourd'hui, elle est devenue une plateforme qui permet aux internautes de partager, de télécharger, d'uploader, de voir des vidéos en ligne, en outre, elle est un service de location des films en ligne. Youtube offre plus de 4 milliards de vidéos par jour.
En 2006, le lancement du site "Twitter", qui est un réseau social de microblogage géré par l'entreprise Twitter Inc. Il permet aux internautes d'envoyer gratuitement de brefs messages, appelés tweets, sur internet, par messagerie instantanée ou par SMS.
Depuis l'existence d'Internet, les développeurs ont essayé de lever les barrières qui séparent les applications traditionnelles des applications Web. Le progrès de la technologie les a aidés à atteindre leurs buts.
Les applications Web deviennent progressivement pleines de fonctionnalités, et faciles à être utilisées, On cite Google Docs, Office Web Apps, BitDefender QuickScan, etc.
Actuellement, les applications Web ont un impact important sur la vie des gens en leur offrant de nombreux avantages, tels que le partage d'information, la communication, la gestion des affaires en ligne, etc.

2.7 Evolution du Web

Le Web a connu un développement remarquable ces dernières années. Selon certaines statistiques, en novembre 2018, le nombre de sites atteignait environ un

milliard et 650 millions [2]. Dans cette section, nous aborderons cette évolution en termes de catégorie d'applications, par ordre chronologique, commençant par le Web statique, le Web dynamique, le Web semi-structuré jusqu'au Web sémantique.

2.7.1 Web statique

Le Web statique est le berceau du Web. Au début, le Web était statique. Il n'y avait que des données statiques dans des pages HTML interconnectées entre elles. La Figure 2.4 illustre l'architecture générale du Web statique. Nous voyons qu'il repose sur la fameuse architecture client/serveur.

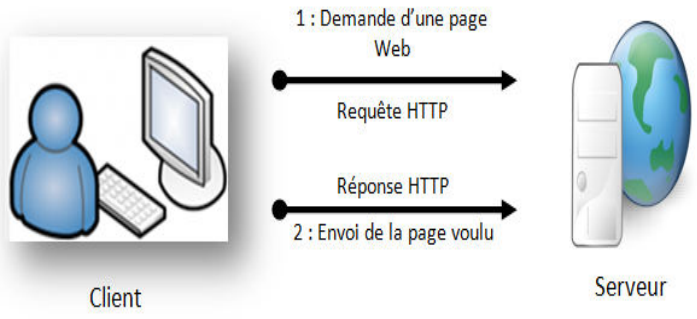

FIGURE 2.4 – *Architecture du Web statique*

Les clients (visiteurs) accèdent aux documents des serveurs par l'envoi des requêtes *HTTP* (1), et ce grâce à des logiciels dédiés appelés navigateurs. Le serveur, dès qu'il reçoit la requête, il commence le traitement. Il cherche le document demandé sur son système de fichiers local, et il le retourne au client (2).
Le Web statique représente les données dans des pages *HTML* d'une façon statique. Ces pages sont interconnectées entre elles. Cette représentation offre aux utilisateurs l'avantage de naviguer sur les pages Web sans savoir où elles se trouvent, que ce soit sur le serveur ou ailleurs. Cependant, cette représentation souffre d'un problème de mise à jour. Il est difficile de mettre à jour le contenu d'un site Web statique.
Parmi les facteurs qui ont mené les chercheurs de la communauté du Web à penser à d'autres formats de représentation, et à d'autres techniques, nous mentionnons quelques uns :

- le besoin d'une représentation dynamique des documents.
- la plupart des données vient des bases de données.
- la non-flexibilité des documents HTML.
- Le besoin de représenter la structure des données.

2.7.2 Web dynamique

La communauté du Web a développé plusieurs technologies pour créer des pages Web dynamiques. La naissance du Web dynamique était en 1993, et ce lors de la première utilisation de CGI (Common Gateway Interface).

2. https://news.netcraft.com/

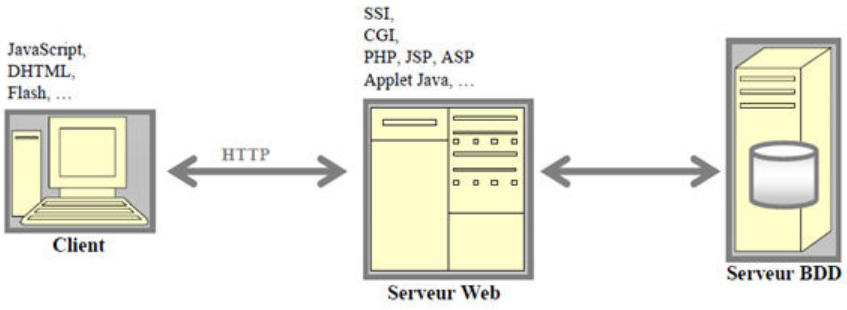

FIGURE 2.5 – *Architecture du Web dynamique*

La Figure 2.5 montre l'architecture du Web dynamique qui est basée sur l'architecture n-tiers. Le client envoie sa demande sous forme d'un fichier contenant des scripts. Le serveur traite la demande du client et exécute les scripts pour construire un document HTML, éventuellement à partir de plusieurs sources. Donc, les documents HTML sont construits à la demande contrairement au Web statique.
Aujourd'hui, le Web a énormément évolué et s'appuie sur des technologies et des langages, tels que PHP, Java, ASP.NET, Python, Ruby et bien d'autres.
Côté serveur Web : il existe quatre techniques pour conceptualiser un serveur dynamique. Nous en parlons brièvement :

- La première technique s'appelle SSI (Server Side Includes). Elle consiste à intégrer des commandes dans un document HTML. Les commandes intégrées servent à construire des documents à la demande. Le serveur reçoit le fichier HTML qui contient les commandes, il les interprète et les exécute pour construire le document demandé qui sera ensuite envoyé au client.
- La deuxième technique appelée CGI (Common Gateway Interface). CGI sont des programmes, généralement des scripts Perl à l'époque. Ces programmes permettent de générer des documents HTML lors de l'arrivée d'une requête HTTP.
- La troisième technique repose sur les langages spécifiques, tels que PHP et ASP. Ces langages fournissent des scripts qui seront insérés dans des documents portant l'extension du langage (.PHP et .ASP). Le serveur, dès qu'il reçoit la demande du client, fait l'exécution des scripts qui permettent de construire des fragments de données à partir de bases de données ou d'autres sources de données, et ensuite construire un document HTML qui est envoyé au client.
- La quatrième technique est la plus récente. Elle consiste à utiliser des applets Java dans le code HTML du document.

Côté client : concernant les trois premières techniques développées du côté serveur, le serveur dynamique interprète les scripts trouvés dans le document HTML avant de l'envoyer au navigateur. Ce qui implique qu'il n'y a pas de modification à faire au niveau du client (navigateur). Par contre, pour la dernière technique (applets Java), le navigateur se charge de l'exécution du code Java de l'applet trouvée dans le document HTML, puis insère et présente le résultat dans un document purement HTML.
Les techniques orientées client permettent d'aider le serveur, à travers l'exécution de certaines opérations au niveau client. Les opérations qui peuvent être traitées

à ce niveau sont par exemple : la validation des formulaires et la génération des diagrammes.

2.7.3 Web semi-structuré

En raison des besoins croissants en traitement de données, tels que la recherche, la présentation, l'intégration, etc. et du fait que les structures de données existantes (HTML, base de données) ne peuvent pas représenter les données facilement et clairement, les développeurs Web ont développé un nouveau format pour la représentation des données et les structures de données. D'où la naissance des modèles semi-structurés. Les données semi-structurées sont des données qui n'ont pas été organisées en référentiel spécialisé ou une méthode d'organisation complexe, comme c'est le cas dans une base de données. Cela rend possible un accès et une analyse sophistiqués de données. Le terme semi-structuré désigne des données (ou des sources de données) ayant une structure implicite, partielle ou irrégulière [Abiteboul 1997].
Les données semi-structurées ont les caractéristiques suivantes :
Implicite : la structure de données est intégrée avec les données dans un seul document, et non pas distincte comme, c'est le cas dans les bases de données. Les données semi-structurées utilisent des balises de métadonnées, qui permettent l'adressage des éléments qu'elles renferment. Les données semi-structurées sont des données auto-descriptives puisqu'elles portent le schéma qui les décrit.
Partielle : la structure de données n'est que partiellement connue.
Irrégulière : les données d'une même source ou de sources différents, peuvent avoir diverses structures. Par exemple, une personne peut être représentée par son nom, prénom, adresse et fonction, alors qu'une autre est représentée seulement par son nom et prénom.
Le schéma est descriptif et non prescriptif : cela signifie que le schéma fournit une structure qui donne une vision globale des données, mais il n'empêche pas les violations de cette structure (ajout/suppression qui entraînent la modification du schéma de données).
En 1998, XML a été défini par le W3C comme un nouveau standard de structuration et d'échange de documents semi-structurés [Bray *et al.* 1997].
Une représentation, comme celle de XML, permet de manipuler les informations irrégulières d'une façon simple et facile, et ce grâce à la représentation des données d'une façon structurée et annotée, sans se préoccuper de la définition d'un schéma fortement structuré et contraignant. Le langage XML représente les données sous forme de documents textuels annotés et structurés par des balises. La structure générée par les balises correspond à une arborescence d'éléments.
On peut créer des documents XML sans avoir défini de contraintes sur leur structure. Mais on peut également, comme dans le modèle relationnel, restreindre ou obliger un document XML à respecter une structure définie sous forme de schémas ou types de documents par une DTD (définition du type de document) ou par un Schéma XML. Une DTD ou un schéma XML sont des règles qui spécifient, pour chaque élément XML, les types de ses éléments fils, leur ordonnancement et leur fréquence et leurs attributs. De plus, des détails sur XML, DTD, schéma XML seront fournis dans le chapitre suivant.
Les modèles de données semi-structurées sont simples et flexibles. Ils permettent de représenter les données et leurs structures dans une même représentation. Cependant, ils ont des lacunes que nous résumons comme suit :

- Il est très difficile d'interroger des données dont on ignore la structure.
- Parfois, il est nécessaire de parcourir toute la base des données semi-structurée pour atteindre l'information cherchée.
- il est difficile de mettre à jour, de vérifier ou de garantir l'intégrité d'une base de données semi-structurée.

2.7.4 Web sémantique

Malgré tous ces développements dans la représentation de la structure de données, la machine ne peut pas traiter ces informations automatiquement. La communauté de W3C propose un autre mécanisme de représentation de données d'une façon sémantique. D'où, le lancement du Web sémantique. Selon Tim Berners-Lee, le Web sémantique fournit un modèle qui permet aux données d'être partagées et réutilisées entre plusieurs applications, entreprises et groupes d'utilisateurs [Berners-Lee *et al.* 2001, Herman 2008]. Des détails sur le Web sémantique seront présentés dans le chapitre du Web sémantique.

2.8 Les langages Web

Lors de la création d'un site Web, les développeurs doivent connaître plusieurs langages Web et non pas un seul. Parmi les langages Web que nous devons connaître, des langages du coté client, tels que *HTML*, *CSS*, *JavaScript*, etc. et les langages du côté serveur, tels que *ASP*, *PHP*, etc. Dans cette thèse, nous n'aborderons d'une façon brève que les langages dont nous avons besoin dans notre sujet, tels que *HTML* et *PHP*.

2.8.1 Le langage HTML

L'HyperText Markup Language (HTML) est un langage de balisage utilisé pour développer des pages hypertexte. Il est dérivé du Standard Generalized Markup Language (SGML). Le langage HTML permet de créer et de structurer des pages. Il permet aussi d'inclure des données variées, telles que les vidéos, les images, les formulaires de saisie et les programmes informatiques. Les documents créés par HTML sont interopérables avec différents équipements, de manière conforme aux exigences de l'accessibilité du Web [DavidL 2018].
Les pages HTML peuvent être générées automatiquement ou écrites avec un éditeur de texte. Souvent, le langage HTML est utilisé avec d'autres langages Web, tels que JavaScript et CSS (feuilles de style en cascade).
Depuis son apparition en 1991, HTML a vu plusieurs versions ; nous en mentionnons : HTML 2.0 de 1995, HTML 3.2 de 1997, HTML 4.01 de 1999, XHTML de 2000 et HTML5 de 2014 [3].
Un document HTML doit respecter la structure illustrée dans le Listing 2.1. Il est composé de deux parties essentielles, à savoir l'entête et le corps, qui seront définies par les balises <head>et <body> respectivement.

3. https://www.w3schools.com/html/html_intro.asp

```
<!DOCTYPE html>
<html>
<head>
<title> titre de la Page </title>
</head>
<body>
le corps de la page HTML
</body>
</html>
```

Listing 2.1 – *Structure d'un document HTML*

Les balises <*head*> et <*body*> peuvent contenir d'autres balises. Parmi les balises HTML les plus utilisées, nous en résumons quelques uns dans le tableau 2.1 [W3Schools 2018],[DavidL 2018].

TABLE 2.1 – *Les Balises HTML.*

Balise	**Déscription**
`<a href="adresse du lien" name="définir une ancre" target="la cible du lien"> le contenu </a>`	permet de créer un lien. Le contenu du lien peut être du texte, une image, etc.
`<img src="mon\_image.png" alt="texte si l'image n'est pas affichée"/>`	permet d'insérer une image dont le chemin est indiqué par l'attribut Src. Il existe d'autres attributs tels que align, title, border, heigh et width.
`<ul start="le numéro du début de numérotation" > <li>premier élément de la liste</li> <li>deuxième élément</li> <li>etc.</li> </ul>`	Définir une liste non numérotée d'éléments
`<dl> <dt>premier élément</dt> <dd>sa description</dd> <dt>deuxième élément</dt> <dd>sa description</dd> <dt>etc..</dt> <dd>...</dd> </dl>`	Définir une liste de définition ou liste descriptive

  - premier élément de la liste - deuxième élément - etc. 	Définir une liste numérotée d'éléments
<table border="1" > <caption> titre de la table </caption> <tr> <!-- ligne 1 --> <th> entête colonne 1</th> <th> entête colonne 2 </th> </tr> <tr> <!-- ligne 2 --> <td> valeur cellule 1 </td> <td> valeur cellule 2 </td> </tr> </table>	Définir un tableau. La balise <tr> définit une ligne, alors que <td> définit une colonne. En plus de l'attribut border, il existe d'autres attributs : lang, title, rules, frames, width. Il existe d'autres attributs qui s'appliquent à <tr> et <td>
<form method="post"> champ 1 : <input type="text" name="nom_champ1" id="id1" > champ 2 : <input type="text" name="nom_champ2" id="id1"> </form>	permet de définir un formulaire qui offre à l'utilisateur un moyen pour introduire les données. Ces données pourront être envoyées au serveur pour être éventuellement traitées. Un formulaire est composé d'un ou plusieurs éléments d'entrée <input>.
Insérer votre paragraphe.	La balise p sert à créer un paragraphe.

Le document HTML est organisé et mis en forme à l'aide des balises répertoriées ci-dessus. Il existe, bien entendu, d'autres balises. HTML est utilisé de manière intensive par les sites Web, de sorte que son utilisation est limitée à la structure, laissant la mise en forme à CSS, et l'animation au Javascript.
Afin de protéger le code et lui donner plus d'interactivité entre l'utilisateur et les bases de données, l'emploi du langage HTML avec PHP est nécessaire.

2.8.2 Le Langage PHP

Hypertext Preprocessor (PHP) est un langage Web conçu pour produire des pages Web dynamiques via un serveur HTTP. L'objectif principal de ce langage est de construire des pages HTML d'une façon rapide et dynamique. Sa syntaxe

est inspirée d'autres langages, tels que C, Java et Perl. Il est largement utilisé, où il s'est classé quatrième en 2018 comme langage Web le plus utilisé [4]. Les scripts PHP sont exécutés sur le serveur, et le résultat est retourné au navigateur (client) sous forme de page HTML.
Un fichier PHP contient non seulement des scripts PHP, mais également d'autres codes, tels que HTML et Javascript. PHP génère non seulement des pages HTML, mais aussi des images, des fichiers PDF, des films Flash, XHTML et XML. Les développeurs optent pour le PHP pour plusieurs raisons. Nous en citons :

- il est facile à apprendre.
- il est multiplateforme, il s'exécute sur Windows, Linux, Unix, Mac OS X, etc.
- il est compatible avec plusieurs serveurs Web, tels que : Apache, IIS, etc.
- il supporte plusieurs types de base de données.
- il est open source et gratuit et disponible sur *(www.php.net)*.

Pour que le fichier PHP soit exécuté par le serveur, il doit remplir les conditions suivantes :

- Le fichier doit posséder une extension .php.
- Le code PHP doit être placé entre deux balises <*?php votre code. ?*>, pour être interprété par le serveur. Le Tableau 2.2 donne quelques codes PHP [5].

TABLE 2.2 – *Instructions PHP.*

Instruction	**Exemple**
`Echo : Afficher le texte entre guillemets`	`<?php echo "Bon jour"; ?>`
`$ suivi par un texte permet de déclarer une variable`	`<?php \$age = 17; ?>`
`Array : permet de créer un vecteur d'éléments`	`<?php \$prenoms = array ('Mohammed', 'Ali','Mustapha', 'Amina', 'Fatima'); ?>`
`If (condition) {traitement} : Permet de définir un traitement conditionnel`	`<?php \$age = 8; if (\$age <= 12) { echo " Tu es mineur !";}?>`

4. https://www.developpez.com/actu/217533/
5. https://www.w3schools.com/php/php_intro.asp

While (condition) traitement : Permet de définir un traitement répétitif	<?php while (\$continuer_boucle == true) {//instructions à exécuter dans la boucle }?>

Historique de PHP

Dans cette section on va présenter un peu de l'historique sur PHP, notamment sa création et ses versions [Luc 2004].
EN 1994, Rasmus Lerdorf, a mis au point le langage PHP. Sa première utilisation était dans l'application Web de gestion des CV. Cette application permet à l'utilisateur de gérer son CV en stockant ses informations dans une base de données. Les informations sont accessibles par l'intermédiaire des requêtes SQL. En raison de la demande croissante de cette application par les internautes, Rasmus Lerdorf a publié la première version de cette application sous le nom "Personal Sommaire Page Tools", puis "Personal Home Page v1.0", et c'était en 1995.
En raison du succès de PHP, Rasmus Lerdorf a publié une deuxième version. Cette version était plus riche et plus améliorée. Elle a intégré les instructions de base de la programmation impérative, telles que les boucles et les structures conditionnelles. Elle a inclus aussi d'autres fonctionnalités comme la gestion des formulaires et MySQL.
En 1997, Zeev Suraski et Andi Gurmans rejoignirent Rasmus afin de constituer un groupe de développeurs pour lancer la troisième version de PHP. Ensuite, ils les ont rejoints : Stig Bakken, Shane Caraveo et Jim Winstead. Et puis, la version 3 fut apparue en 1998, la version 4 en 1999 et la version 5 en 2012. Ensuite, les versions sont apparues l'une après l'autre jusqu'à la dernière version 7.3 en 6 décembre 2018.

2.9 Conclusion

Ce chapitre était comme une introduction détaillée d'un élément important dans notre sujet de thèse, à savoir l'application Web. Nous avons fourni beaucoup de notions sur ce concept, notamment sa définition, ses avantages, son cycle de développement, son architecture, ses différents types et son historique. En plus, nous avons abordé aussi comment le Web est évolué du Web statique vers le Web dynamique puis vers le Web semi-structuré jusqu'au Web sémantique. Ensuite, nous avons donné un aperçu sur les langages Web pertinents à notre sujet, tels que HTML et PHP.

Web sémantique

3

Sommaire

3.1 Introduction . . . 22
3.2 Historique . . . 22
3.3 Langages du Web sémantique . . . 23
3.3.1 Le langage XML (Extensible Markup Language) . . . 24
3.3.2 Le schéma XML (XML schema) . . . 27
3.3.3 Conflit de nommage dans XML . . . 29
3.3.4 Espace de nommage XML (Namespace) . . . 29
3.4 Les données Liées RDF (Ressource Description Framework) . . . 30
3.4.1 Les concepts de base de RDF . . . 30
3.4.2 Exemple d'un graphe RDF . . . 32
3.4.3 Sérialisation . . . 32
3.5 Vocabulaire de RDF ou Schéma RDF . . . 33
3.5.1 Les classes . . . 34
3.5.2 Les propriétés . . . 34
3.5.3 Autre concepts supplémentaires . . . 35
3.6 Le Langage Web d'ontologie OWL . . . 36
3.6.1 Les types d'OWL . . . 36
3.7 Le langage SPARQL . . . 41
3.8 Conclusion . . . 42

3.1 Introduction

Le Web sémantique est une nouvelle génération dont l'objectif est de représenter les données de manière à pouvoir les utiliser non seulement à des fins d'affichage, mais aussi pour l'automatisation, l'intégration et la réutilisation [Boley *et al.* 2001]. Le Web sémantique devient un axe de recherche et de développement dans ces dernières années. Le Web sémantique est une activité importante du W3C [Herman 2008].
Dans ce chapitre on va jeter la lumière sur cette nouvelle génération du Web, notamment sa pile de langages.

3.2 Historique

Le Web sémantique consiste à rendre le Web classique compréhensible par les machines [Heflin & Hendler 2001]. Il s'agit de permettre aux agents intelligents d'exécuter des actions complexes sur le Web via une infrastructure appropriée [Hendler 2001]. Il permet l'utilisation complète de l'infrastructure par les agents, et l'intégration des agents dans le Web d'une façon transparente permettre à ces agents de récupérer et manipuler des informations pertinentes [Cost *et al.* 2002]. En outre, le Web sémantique offre de nombreux avantages aux applications Web. Il permet ainsi de déclarer explicitement les connaissances intégrées, intégrer les informations de manière intelligente, fournir un accès sémantique à Internet et extraire des informations de textes [Corcho *et al.* 2002]. Il explique aussi comment établir une interopérabilité fiable aux services Web [McIlraith *et al.* 2001, Devedzic 2004].
Le problème du Web classique réside dans la quantité d'information qu'il contient, et la manière de l'exploiter pour servir vraiment le besoin d'utilisateur. Le Web classique fournit des informations à l'utilisateur d'une façon non intelligente. La tâche d'interprétation et de compréhension des informations est laissée à l'utilisateur dans pratiquement toutes les applications Web. En outre, les informations présentes dans le Web sont en langage naturel, ce qui les rend non compréhensibles et non interprétables par la machine. Seuls les utilisateurs peuvent le faire. Malgré ça, l'homme ne peut pas traiter qu'une infime fraction d'information existante sur le Web. Par contre, s'ils pouvaient rendre la machine intelligente, ils bénéficieraient énormément des contenus Web [Dragan *et al.* 2009].
Malheureusement, le Web est conçu pour les humains et non pas pour les machines, quoique tout ce qui se trouve sur le Web est lisible par machine, mais n'est pas compréhensible [Lassila 1998].
Le besoin du Web sémantique est accru avec la croissance du contenu Web. Dans le Web sémantique, les données sont exprimées sous une forme précise et interprétable par machine. Même les agents logiciels peuvent traiter, partager et réutiliser ses données. Le Web sémantique permet aux applications Web d'interopérer au niveau syntaxique et au niveau sémantique de données [Dragan *et al.* 2009].
Le fondateur du Web sémantique Tim Berners-Lee a envisagé q'un site Web sémantique offre un accès à l'information d'une façon automatique en se basant sur la sémantique des données traitée par une machine et heuristiques utilisant ces métadonnées [Berners-Lee *et al.* 1999, Berners-Lee *et al.* 2001]. Le Web sémantique offre un niveau de service d'une haute qualité grâce à la représentation explicite de la sémantique des données.
L'organisme de standardisation W3C a développé des techniques et des lan-

gages pour décrire les données sur le Web. de plus, les gens de l'IA ont déjà développé des applications et des outils utiles pour le Web sémantique [Noy *et al.* 2001, Cost *et al.* 2002]. Il existe plusieurs problèmes cruciaux liés au Web sémantique qui peuvent être attribués à l'une des catégories suivantes : les langages du Web sémantique, les ontologies, l'annotation sémantique des pages du Web, Framework de développement, le raisonnement, et les services fournis par le Web sémantique [Dragan *et al.* 2009].
La Web sémantique fournit un Framework intitulé *Ressources Description Framework (RDF)*. Ce Framework permet le partage et la réutilisation des données à travers les applications, des entreprises et des communautés [Dragan *et al.* 2009].
Le Web sémantique est le Web de données, c.à.d. que les données peuvent être utilisées par différentes applications contrairement au Web classique où chaque application contrôle ses propres donnés [Dragan *et al.* 2009].
En résumé, le Web sémantique concerne deux choses : les formats communs d'intégration et de combinaison de données provenant de diverses sources, et le langage pour enregistrer les relations entre les données et les objets du monde réel [Dragan *et al.* 2009].

3.3 Langages du Web sémantique

La Figure 3.1 présente la pile du Web sémantique. Elle est constituée de plusieurs couches [Berners-Lee *et al.* 1999, Berners-Lee *et al.* 2001]. Le langage de la couche supérieure utilise la syntaxe et la sémantique des langages des couches inférieures. Tous les langages de la pile reposent sur la syntaxe XML. En fait, XML est un métalangage pour représenter d'autres langages du Web sémantique. Par exemple, XML Schéma utilise XML pour définir une classe de documents XML. RDF représente les métadonnées sur les ressources Web en utilisant XML. Même les autres langages comme RDF Schéma et OWL utilisent également la syntaxe XML [Dragan *et al.* 2009]. Notez que la pile du Web sémantique comporte plusieurs versions selon l'évolution des techniques du Web sémantique.

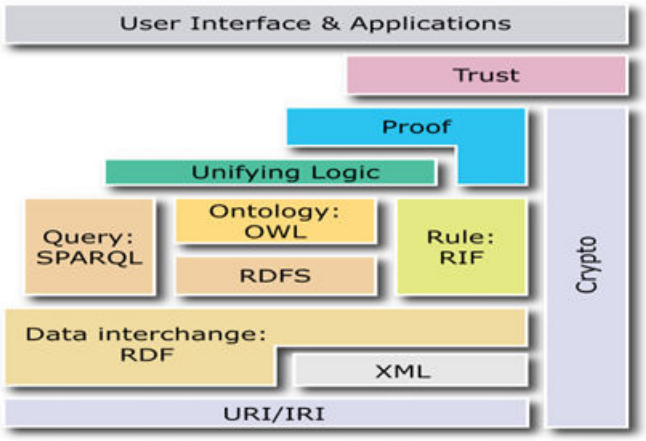

Figure 3.1 – *La pile du Web sémantique de Tim Berners-Lee*

Les principaux techniques du Web sémantique s'inscrivent dans un ensemble de spécifications en couches. Les composants actuels sont le modèle RDF (Res-

source description Framework), le langage de RDF schéma, le langage d'ontologie OWL, et le langage de requête standardisé SPARQL qui permet de «joindre» des collections de données RDF [Dragan *et al.* 2009].
Les sous-sections suivantes discutent en détail la pile du Web sémantique et décrivent les langages les langages les plus pertinents à notre problème, notamment XML, XML Schéma, RDF, RDF Schéma, OWL et SPARQL.

3.3.1 Le langage XML (Extensible Markup Language)

Le XML (Extensible Markup Language) est langage qui permet de représenter des données sous une forme structurée à l'aide des balises. Il a été dérivé du standard SGML (ISO 8879). Le XML joue un rôle très important dans l'échange de données sur le Web. Il représente aussi un langage de base dans la pile du Web sémantique [Bray *et al.* 2006]. Dans cette partie nous allons introduire les concepts importants du standard XML, tels que la syntaxe, la notion de document bien formé, document valide, DTD, etc.

Syntaxe du langage XML

Dans cette partie on ne va pas donner touts les détails sur la syntaxe du langage XML, mais quelques notions fondamentales pour établir un document XML.

a) Les balises : constituent les éléments de base d'un document XML. Une balise porte un nom, qui doit être une chaîne de caractères alphanumériques, commençant obligatoirement par une lettre et ne contenant pas d'espaces. Une balise commence par un < et se termine par un >. Par exemple, <*Livre*>. Il existe deux types de balise notamment balise par paires et balise unique [Bray *et al.* 2006].

b) Les balises par paires et uniques : les balises par paires sont composées de deux balises, une s'appelle ouvrantes et l'autre fermantes, qui doivent être identiques. Entre ces deux balises, on peut mettre un contenu qui peut être une chaîne de caractères ou une balise. Par exemple : <*Livre*> c'est un livre de BDD </*Livre*>. Les balises uniques sont des balises par paires qui ne possèdent pas un contenu [Bray *et al.* 2006].

c) Les attributs : c'est une information supplémentaire qui est ajoutée dans une balise afin de donner des renseignements supplémentaires sur son contenu. Un attribut doit respecter les mêmes conditions de nommage que les balises, et doit être déclaré selon la syntaxe suivante : <*nom_de_balise nom_attribut= "valeur"*> [Bray *et al.* 2006]. Par exemple :(voir le listing 3.1)

```
1 <livre langue= "Fr ">
2 <description>c'est un livre de BDD. </description>
3 <livre/>
```

Listing 3.1 – *Exemple d'un élement XML contenant d'attribut*

d) Les commentaires : c'est un texte qui permet d'expliquer ce que l'on fait. Il vous permet d'annoter des parties d'un fichier (code source). En XML, on exprime les commentaires sous forme de balise unique selon la syntaxe suivante <*!–Votre commentaire ici–*>. [Bray *et al.* 2006].

e) La structure d'un document XML : un document XML est composé de deux parties : le prologue et le corps. Le prologue est la première ligne. Il représente l'entête d'un document XML. Il donne des informations de traitement, notamment la version d'XML, le codage à utiliser et l'autonomie qui permet d'indiquer si ce document est rattaché à un autre document ou non. La syntaxe de prologue est : <*?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?*> Le corps consiste d'un ensemble de balises qui décrivent les données. Ces balises doivent être placées dans une balise paire appelée l'élément racine du corps [Bray *et al.* 2006].

Documents XML bien formé

Un document XML est dit bien formé si et seulement, s'il respecte les conditions suivantes [Bray *et al.* 2006] :

- Il doit utiliser la version 1.1 du XML. Ceci doit être renseigné dans son prologue.
- Il doit possèder une seule balise racine.
- Il doit respecter la syntaxe décrite dans la section précédente, concernant la définition des balises et des attributs.
- Toute balise ouvrante doit être fermée.
- Les balises ne doivent pas chevaucher. Par exemple, il n'est pas possible d'ouvrir la balise A puis la balise B et de fermer la balise A avant de fermer la balise B.

Le Listing 3.2 montre un exemple d'un document XML bien formé.

```
1 <?xml version="1.1" encoding="UTF-8" standalone="yes"?>
2 <!--Description d un livre --!>
3 <Livre titre="Base de données">
4     <Auteur prenom="Mohammed" nom="Ben ali"/>
5     <Chapitre numero="1" titre="SGBD">
6   Les Système de gestion de base de données  (SGBD) sont des
     systèmes permettent de faciliter la gestion d une BDD
7     </Chapitre>
8 </Livre>
```

Listing 3.2 – *Exemple d'un document XML bien formé*

Les documents valides

Un document valide doit respecter une hiérarchie ou une structuration de balises préétablie. En tenant l'exemple précédent, la balise <*Chapitre*> doit être obligatoirement inclue dans la balise <*Livre*>. Tout document respectant cette structure sera considéré comme un document valide. Notez qu'un document bien formé n'est forcément un document valide. Pour garantir la structuration des balises d'un document XML, on doit utiliser une DTD (Document Type Definition) ou XML Schéma [Xavier 2005].
Une DTD est le premier moyen pour spécifier la structure des balises. Elle est représentée sous forme d'un document textuel contenant un ensemble de règles que l'on impose au document XML. Ces règles permettent de décrire la façon dont le document XML doit être construit, c'est-à-dire sa structuration. Les règles peuvent être utilisées pour [Xavier 2005] :

- Définir une balise ou un attribut
- Imposer le type d'une donnée des attributs.
- Imposer l'ordre d'apparition des balises dans le document.

Une DTD est écrite soit dans le même document XML et là on parle d'une DTD interne, ou bien elle sera mise dans un document séparé (DTD externe) et invoquée dans le document XML via l'élément DOCTYPE [Bray *et al.* 2006].

La structure d'une DTD

Une DTD structure les balises à travers les mots-clés suivants : *ELEMENT*, *ATTLIST* et *ENTITY* [Bray *et al.* 2006].

ELEMENT : permet de définir les règles portant sur une balise, notamment le nom de la baise et son contenu. Le contenu peut être une liste de balises séparées par des virgules, ou bien une valeur simple (une chaîne de caractères, un entier, un nombre décimal, un caractère, etc.). On utilisera pour cela le mot-clé *#PCDATA*. la syntaxe est la suivante : *<!ELEMENT nom_balise (Contenu) >* [Bray *et al.* 2006].

ATTLIST : permet de définir un ensemble d'attributs qui seraient appliqués à une balise. La syntaxe suivante montre comment utiliser cet élément : *<!ATTLIST nom_balise attributs type mode>* [Bray *et al.* 2006].

- **Nom_balise :** indique la balise auquel sera attaché l'attribut.
- **Attribut :** pour définir un attribut ou une liste d'attributs.
- **Type :** pour indiquer le type de données de l'attribut qui pourra être :
  - le mot-clé *CDATA* pour désigner une chaîne de caractères, un entier, un nombre décimal, un caractère, etc.,
  - liste de valeurs sous la forme : *(valeur1 | valeur2 | valeur2 | ...)*
  - le mot-clé *ID* pour indiquer que la valeur de l'attribut est unique.
- **Le mode :** pour mentionner une information supplémentaire sur l'attribut, telle qu'une indication sur son obligation ou sa valeur par défaut.
  - Le mot-clé *#REQUIRED* permet de spécifier que l'attribut soit obligatoirement renseigné. Et le mot-clé *#IMPLIED* pour dire que l'attribut est optionnel.
  - Le mot-clé *#FIXED* permet de définir une valeur constante pour l'attribut.
  - Pour définir une valeur par défaut sur l'attribut, il suffit de renseigner cette valeur.

ENTITY : les entités sont des alias qui permettent de réutiliser des informations au sein du document XML ou d'une DTD. Il existe trois types d'entité, à savoir : les entités générales, les entités paramètres et les entités externes. Au moment de l'interprétation du document XML, les références aux entités seront remplacées par leurs valeurs respectives [Bray *et al.* 2006].

- **Entités générales :** possède une syntaxe simple : *<!ENTITY nom 'valeur'>*. Elle permet d'associer un alias à une information *(valeur)* afin de l'utiliser dans un document XML, et ce par la syntaxe suivante : *&nom*.
- **Entités paramètres :** ce type d'entités est utilisé seulement dans les définitions DTD. Elles permettent d'associer un alias à une partie de la déclaration de la DTD. Elles respectent la syntaxe suivante : *<!ENTITY % nom 'valeur'>*.
- **Entités externes :** il existe deux types d'entités externes, à savoir analysées et non analysées. Les entités externes analysées ont le même rôle

que les entités générales. Sauf que les informations sont stockées dans un fichier séparé. Leurs syntaxe est : *<!ENTITY nom SYSTEM 'URI'>*. URI permet d'indiquer le nom du fichier contenant l'information.

Les DTD offrent un mécanisme très simple pour définir la structuration des balises. Cependant, les DTD souffrent de plusieurs lacunes, parmi lesquelles on cite :

- ils se basent sur un format textuel et non pas XML, ce qui les rend inexploitables.
- ils ne permettent pas de typer les données. Ils permettent seulement d'indiquer qu'une balise contient des données sans préciser que c'est un nombre entier, un nombre décimal, une date, une chaîne de caractères, etc.
- ils possèdent une expressivité très limitée. Il n'est pas possible de réutiliser une structuration existante afin de l'enrichir, ni de définir des ordres particuliers d'inclusion.

En raison de toutes ces faiblesses, le W3C a mis en place le standard XML Schéma, qui permet de donner davantage de possibilités afin de spécifier mieux des structurations de balises XML.

3.3.2 Le schéma XML (XML schema)

Le schéma XML est standardisé par le W3C. Le schéma XML substitue la DTD grâce à leur souplesse de définition des structurations de balises XML. Les schémas XML apportent plusieurs avantages, en voici quelques-uns [Shudi *et al.* 2012] :

- Ils sont définis sous forme de documents XML. Donc, il est possible de les manipuler et de l'exploiter.
- Ils permettent de définir des types de structuration. Il est possible de réutiliser ces types afin de les étendre ou de les restreindre.
- Ils utilisent un système de typage beaucoup plus complet.
- Ils sont plus expressifs que les DTD lors de l'écriture des différentes contraintes qui régissent un document XML.

Structuration d'un schéma XML

Bien que les documents de schéma XML soient écrits en langage XML, ils ne portent pas l'extension *".xml"*. Un document écrit avec un schéma XML porte l'extension *".xsd"*. Comme pour les DTD, on sépare les données XML du schéma XML. Un document de schéma XML est composé de deux parties : le prologue et le corps, comme c'était le cas pour les documents XML [Shudi *et al.* 2012].

- Le prologue est définit comme suit :
  <?xml version='1.0' encoding='UTF-8'?>

- Le corps est définit par l'élément racine *"xsd :schema"* comme suit :
  <xsd :schema xmlns :xsd= "http ://www.W3.org/2001 :xmlschema"> ...</xsd :schema >

Pour utiliser ou référencer un document de schéma XML dans un document XML, il suffit de renseigner les deux attributs *"xmlns :xsi"* et *"xsi :schemaLocation"* dans l'élément racine du document XML, comme suit [Shudi *et al.* 2012] :

- **xmlns :** *xsi= "http ://www.w3.org/2001/XMLSchema".*
- **xsi :** *schemaLocation="chemin_vers_le_documents_xsd".*

Syntaxe du schéma XML

Comme la DTD, le schéma XML permet de spécifier la structure des balises d'un document XML à travers plusieurs constructeurs, tels que les éléments simples, les attributs, les éléments complexes, etc [Shudi *et al.* 2012].

Les éléments simples : permettent de déclarer la structure des balises simples. Les balises simples ne contiennent ni des balises imbriquées, ni des attributs. Ils contiennent seulement des valeurs de types simples, par exemple un chiffre, une chaîne de caractères ou une date. La syntaxe pour spécifier les éléments simples est la suivante [Shudi *et al.* 2012] :
<xsd :element name="nom_balise" type="xsd :type" default="valeur" fixed="constant"/>

- **Name :** permet de spécifier le nom de la balise XML
- **Type :** pour indiquer le type de valeur d'une balise qui peut être string, int, etc.
- **Default :** pour donner à une balise une valeur par défaut.
- **Fixed :** pour imposer une valeur inchangeable à une balise.

Les attributs : ils ont la même syntaxe que les éléments simples. Ils sont déclarés via le mot-clé *"attribut"* [Shudi *et al.* 2012].
<xsd :attribut name="nom_balise" type="xsd :type" default="valeur" fixed="constant" use="required"/>

- **Use :** permet de spécifier si l'attribut est obligatoire ou non. L'absence de cette propriété signifie que l'attribut n'est pas obligatoire. Le reste des propriétés (*name*, *type*, *default*) ont les mêmes définitions (Shudi et al., 2012).

Les éléments complexes : permettent de spécifier la structure des balises complexes dans un document XML. Un élément complexe contient des attributs et des balises imbriquées. La syntaxe pour définir ces éléments est donnée dans le listing 3.3 [Shudi *et al.* 2012] :

```
1 <xsd :element name="nom_balise" type="xsd :type"
     default="valeur" fixed="constant">
2  <xsd :complexType>
3           < !--contenu-->
4  </xsd :complexType>
5 </xsd :element>
```

Listing 3.3 – *Syntaxe geénérale d'un élément complexe*

Le contenu peut être simple, standard ou mixte.

- **Le contenu simple :** il n'est formé que d'attributs et de valeur de type simple. La syntaxe de listing 3.4 permet de déclarer un élément complexe qui ne contient que de valeur simple et des attributs [Shudi *et al.* 2012].

```
<xsd:element name="mon_balise">
<xsd:complexType>
<xsd:simpleContent>
<xsd:extension base="type_de_valeur">
<xsd:attribute name="nom_attribut_1" type="
    type_attribut_1">
</xsd:attribute>
<xsd:attribute name="nom_attribut_2" type="
    type_attribut_2">
</xsd:attribute>
                .....
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
```

Listing 3.4 – *Syntaxe d'un élément complexe contient des valeurs simple et attributs*

- **Le contenu standard :** dans ce type de contenu on peut représenter des éléments complexes qui contiennent des éléments imbriqués (simple ou complexes) ou uniquement des attributs [Shudi *et al.* 2012].
- **Le contenu mixte :** est un mélange entre les types précédents (les éléments simples & les attributs). Un élément complexe peut contenir des attributs et des éléments imbriqués que ce soit simples ou complexes [Shudi *et al.* 2012].

Notez qu'il est possible d'exprimer d'autres spécifications au sein du Schéma XML. Par exemple, on peut [Shudi *et al.* 2012] :

- limiter le nombre d'occurrences d'un élément via les mots-clefs *minOccurs* et *maxOccurs*.
- référencer un élément dans un document Schéma XML afin de le réutiliser plusieurs fois.
- définir une relation d'héritage entre les éléments d'un Schéma XML.

Nous avons vu que la définition d'une structuration de balises à laide du Schéma XML est plus complexe qu'elle ne l'est en DTD. C'est pour cette raison qu'il n'existe que peu de structurations en schéma XML.

3.3.3 Conflit de nommage dans XML

On peut trouver dans des documents XML différents, des balises possèdant des noms identiques, mais ayant des significations différentes. Cette problématique nous engendre l'incertitude de l'information lorsqu'on veut utiliser ces balises dans un même document XML. Il serait impossible d'interpréter un tel document. Dans l'ingénierie dirigée par les modèles (IDM), ce problème du conflit de nommage est très intéressant. Pour cela, le W3C a défini la notion de l'espace de nommage (namespace) [Xavier 2005].

3.3.4 Espace de nommage XML (Namespace)

L'espace de nommage est défini par une URL qui permet de regrouper différentes balises ayant une signification commune, des DTD ou des schémas XML. Pour utiliser un espace de nommage au sein d'un document XML, il suffit de

définir un alias vers l'URL de l'espace de nommage, et cela se fait à l'aide de la propriété *"xmlns"* dans n'importe quelle balise ouvrante. Pour référencer une balise appartenant à l'espace de nommage, il suffit de préfixer la balise par l'alias [Xavier 2005].

3.4 Les données Liées RDF (Ressource Description Framework)

RDF est l'acronyme de Ressource Description Framework. C'est est un Framework qui permet de représenter des informations sur le Web manière flexible et peu contraignante. Dans cette section, on va projeter la lumière sur ce Framework en abordant sa syntaxe abstraite, concrète et sa sémantique formelle. RDF peut être utilisé dans des applications indépendantes, où il offre une présentation directe et facile à comprendre. Mais généralement, la valeur de RDF est considérable lorsqu'il y a un partage d'information, car la valeur de l'information augmente lorsqu'elle est accessible. RDF est conçu pour les utilisations suivantes [Graham & Jeremy 2004] :

- Métadonnées : fournir des informations sur les ressources Web et les systèmes qui les utilisent.
- Applications nécessitant des modèles d'informations ouvertes, par exemple, planification et description des processus organisationnels.
- Pour rendre l'information traitable automatiquement par ordinateur.
- Pour combiner les données de différentes applications afin d'extraire ou déduire une nouvelle information.
- Pour automatiser le traitement des informations Web par les agents logiciels.

3.4.1 Les concepts de base de RDF

Les concepts de base de RDF sont comme suit : le modèle graphique, les IRIs, les types de données, les littéraux, sérialisation. Les sous-sections suivantes détaillent toutes ces notions.

Modèle de données graphique

Une expression RDF est exprimée sous forme de triplets, où chaque triplet est constitué d'un sujet, un prédicat et un objet. La fusion d'un ensemble de triplets forme un graphe RDF, dont les nœuds sont des sujets ou des objets et les arcs orientés représentent les prédicats. La Figure 3.2 représente un triplet sous forme d'un lien nœud-arc-nœud. Un triplet représente une déclaration d'une relation entre les éléments désignés par les nœuds (sujet et objet). La direction de l'arc est significative : il pointe toujours vers l'objet [Graham & Jeremy 2004].

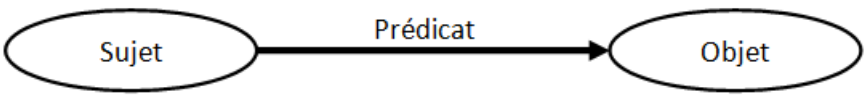

Figure 3.2 – *Représentation graphique d'un triplet RDF*

Les IRIs

Un IRI (Internationalized Resource Identifier) est une chaîne de caractères Unicode qui doit être conforme à la syntaxe définie dans la RFC 3987. Dans un triplet RDF, les adresses IRI doivent être absolues et peuvent contenir un identificateur de fragments. Une IRI est une généralisation d'une URI. Elle permet une large présentation de caractères Unicode. Chaque adresse URI et URL absolue est une adresse IRI, mais une IRI n'est pas forcément une URI [Richard *et al.* 2014].
Un nœud (sujet ou objet) d'un triplet peut être :

- un URI avec un identificateur de fragments facultatifs (référence URI ou UR Iref)
- un littéral
- un nœud blanc qui n'est pas une référence d'URI ou un littéral. Un nœud vide est simplement un nœud unique pouvant être utilisé dans une ou plusieurs instructions RDF.

Un prédicat (propriété) est une référence d'URI qui identifie une relation entre les éléments représentés par les nœuds auxquels elle se connecte. Une référence d'URI de prédicat peut figurer comme un nœud dans un graphe RDF. Notez que les URI relatifs ne sont pas utilisés dans un graphe RDF [Richard *et al.* 2014].

Types de données

Les types de données sont utilisés par RDF pour représenter des valeurs, telles que les entiers, les nombres à virgule flottante et les dates. Les types de données utilisées dans RDF sont celles de XML Schéma. En plus, RDF définit deux types de données non normatifs, à savoir *rdf :HTML* et *rdf :XML literal* [Graham & Jeremy 2004, Richard *et al.* 2014].
Un type de données consiste en un espace lexical, un espace de valeurs et un Mapping entre les deux espaces (lexical et valeur). L'espace lexical est un ensemble de chaines Unicode. Le mappage entre l'espace lexical et l'espace valeur d'un type de donnée est un ensemble de paires, dont le premier élément appartient à l'espace lexical, et le second à l'espace de valeurs [Richard *et al.* 2014].
Par exemple, le type de données XML Schéma *(xsd :boolean)*, dans lequel chaque membre de l'espace de valeur comporte deux représentations lexicales, est défini comme suit [Richard *et al.* 2014] :

- Espace lexical : *{Vrai, faux, 1, 0}*
- Espace de valeur : *{vrai, faux}*
- Mapping : *{<"True", true>, <"false", false>, <"1", true>, <"0", false>,}*.

Le tableau 3.1 représente les littéraux qui peuvent être définis à l'aide de ce type de donnée.

TABLE 3.1 – *Les littéraux de type booléenne*

Littérale	Valeur
<"true", xsd :boolean>	True
<"false", xsd :boolean>	False
<"1", xsd :boolean>	True
<"0", xsd :boolean>	False

Les littéraux

Les littéraux sont utilisés pour identifier des valeurs, telles que les nombres, les dates et les chaînes de caractères. Un littéral pourrait être représenté par un URI, mais il est plus pratique d'utiliser des littéraux. Un littéral ne peut être qu'un objet d'une déclaration RDF.

3.4.2 Exemple d'un graphe RDF

RDF permet de représenter des faits simples indiquant une relation entre deux choses. Un tel fait peut être représenté par un triplet RDF dans lequel le prédicat désigne la relation, et le sujet et l'objet désignent les deux choses. Par exemple, un document en langue arabe identifié par son URI et qui a comme auteur *"Mohamed"*, est représenté comme suit :

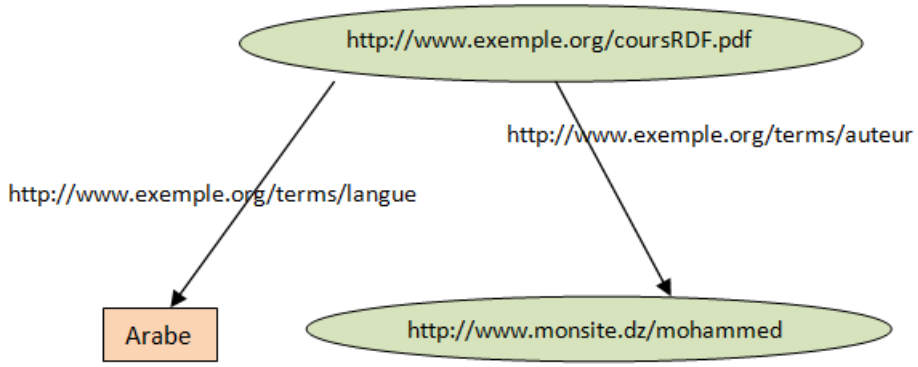

FIGURE 3.3 – *Exemple d'un graphe RDF*

La Figure 3.3 représente un graphe RDF qui contient une ressource dont le sujet est IRI du document et les objets sont *"Mohamed"* et *"Arabe"* qui pourraient être des ressources, et les prédicats sont *"Langue"* et *"Auteur"*.

3.4.3 Sérialisation

Pour échanger les données d'un graphe RDF il faut le sérialiser. Il existe de nombreux formats de sérialisation, tels que N-Triplets, Turtle et XML/RDF.

N-Triples

L'écriture la plus simple d'un triplet est une séquence de termes (sujet, prédicat, objet), séparés par des espaces et terminés par un point (.) après chaque triplet, selon la syntaxe suivante [David 2014] :

<IRI du sujet> <IRI du prédicat> <IRI de l'objet ou littéral>.

Par exemple :

```
<http://example.org/ben#moi> <http://schema.org/DateDeNaissance>
"1985-01-31"^^<http://www.w3.org/2001/XMLSchema#date>.
```

Turtle

La syntaxe Turtle semble de N-triples à la différence que Turtle est plus lisible que N-Triples. Listing 3.5 montre un exemple d'un document RDF exprimé en format Turtle. Des détails peuvent être trouvés dans [David *et al.* 2014].

```
@BASE <http://example.org/>
@PREFIX foaf: <http://xmlns.com/foaf/0.1/>
@PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
@PREFIX schema: <http://schema.org/>
<ben#me>
    a foaf:Person ;
  foaf:name "Benamar bouougada"
    schema:DatedeNaissance "1985-01-31"^^xsd:date ;
```

Listing 3.5 – *RDF sérialisé en Turtle*

Sérialisation en XML

La norme RDF/XML utilise la syntaxe XML pour représenter un graphe RDF. Une ressource RDF est décrite à l'aide de l'élément *"Description"*, dont l'attribut *"about"* représente le sujet, et les propriétés sont des sous-éléments qui peuvent être simplifiés en attributs. D'autres éléments, tels que les listes, les ressources imbriquées, etc. peuvent être trouvés dans [Fabien & Guus 2014]. Le listing 3.6 montre un exemple de sérialisation en utilisant XML.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:schema="http://schema.org/">
<rdf:Description rdf:about="http://example.org/ben#moi">
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
<schema: DateDeNaissance rdf:datatype="http://www.w3.org/2001/
   XMLSchema#date"> 1985-01-31</schema:DateDeNaissance>
<foaf:knows rdf:resource="http://example.org/benamar#moi"/>
</rdf:Description>
</rdf:RDF>
```

Listing 3.6 – *RDF sérialisé en XML*

3.5 Vocabulaire de RDF ou Schéma RDF

RDF Schéma (RDFS) fournit un vocabulaire de modélisation de données pour le RDF. RDFS est une extension sémantique du vocabulaire RDF. Il fournit des mécanismes pour décrire des groupes de ressources connexes et les relations entre eux.
Un système RDFS (classes et propriétés) est similaire à un système qui utilise un langage de programmation orienté objet, tel que Java. Il se diffère dans le sens de définition d'une classe. Au lieu de définir une classe comme les langages de

programmation orientés objet, il décrit les propriétés en fonction des classes de ressources auxquelles elles s'appliquent, et ce à l'aide de mécanisme *"domaine" & "range"* qui sera décrit par la suite [Dan & Guha 2014].
Par exemple, on définit une propriété *"exp :auteur"* pour un domaine *"exp :document"* et un range *"exp :personne"*, alors qu'un système classique orienté objet définit une classe *"exp :Livre"* avec un attribut *"auteur"* de type *"personne"*. Donc l'un des avantages de l'approche centrée sur les propriétés de RDF est qu'elle permet à n'importe qui d'étendre la description des ressources existantes [Dan & Guha 2014].
Le vocabulaire de base est défini dans l'espace de nommage *"rdfs"* identifié par l'IRI suivante : *"http ://www.w3.org/2000/01/rdf-schema#"*.

3.5.1 Les classes

Les ressources sont divisées en classes. Les membres d'une classe constituent les instances de cette classe. Une classe est identifiée par une IRI et décrite par les propriétés RDF. La propriété "rdf :type" indique qu'une ressource est une instance d'une classe. Notez qu'une classe est aussi une ressource [Dan & Guha 2014].
RDF différencie entre la classe et ses instances. Il associe à chaque classe son extension (ensemble de distances). Deux classes différentes peuvent avoir la même extension. Par exemple, une banque peut définir une classe de personnes ayant des comptes courants ouverts dans la même adresse d'habitation. La même banque peut aussi définir une autre classe de personnes ayant des comptes de crédits ouverts dans la même adresse d'habitation. Il est donc possible d'avoir les mêmes instances, mais ayant des propriétés différentes, et ce selon leurs propriétés définies dans sa propre classe [Dan & Guha 2014].
Une classe peut être une instance d'elle-même. Un groupe de ressources des classes RDFS représente une classe appelée *"rdfs :Class"*. La propriété *"rdf :subClassOf"* est utilisée pour indiquer qu'une classe est une sous-classe d'une autre. RDFS définit un ensemble de classes, telles que [Dan & Guha 2014] :

- ***rdfs :Ressource :*** est la classe mère de toutes les autres classes. *rdfs :Resource* est une instance de *rdfs :Class*. Chaque déclaration décrite par RDF est appelée ressources qui est une instance de la classe *"rdfs :Resource"*.
- ***rdfs :Class :*** c'est la classe des ressources qui sont des classes RDF. rdfs :Class est une instance d'elle-même.
- ***rdfs :Litteral :*** c'est la classe des valeurs littérales, telles que les chaînes de caractères et les entiers etc. *"rdfs :Literal"* est une instance de *"rdfs :Class"*. *"rdfs :Literal"* est une sous-classe de *"rdfs :Resource"*.
- ***rdfs :Datatype :*** c'est la classe des types de données. Toutes les instances de *"rdfs :Datatype"* correspondent aux types de données RDF. *"rdfs :Datatype"* est une instance et une sous-classe de *"rdfs :Class"*. Chaque instance de *"rdfs :Datatype"* est une sous-classe de *"rdfs :Literal"*.

3.5.2 Les propriétés

RDF décrit le concept de propriété *"rdf :Property"* comme une relation entre le sujet et l'objet. *"rdf :Property"* est une classe des propriétés RDF, et est une instance de *"rdfs :Class"*. RDFS définit un nouveau concept appelé sous-propriété *"rdfs :subPropertyOf"* pour indiquer qu'une propriété est une sous propriété d'une

autre [Dan & Guha 2014]. La classe *"rdf :Property"* possède un ensemble d'instances parmi lesquelles on cite [Dan & Guha 2014] :

- ***rdfs :range :*** utilisée pour indiquer que les valeurs d'une propriété sont des instances d'une ou plusieurs classes.
- ***rdfs :domain :*** utilisée pour indiquer que toute ressource ayant une propriété donnée est une instance d'une ou de plusieurs classes.
- ***rdf :type :*** utilisée pour indiquer qu'une ressource est une instance d'une classe.
- ***rdfs :subClassOf :*** utilisée pour indiquer que toutes les instances d'une classe sont des instances d'une autre.
- ***rdfs :subPropertyOf :*** utilisée pour indiquer que toutes les ressources associées à une propriété sont également liées par une autre.
- ***rdfs :label :*** pouvant être utilisée pour fournir une lisibilité à l'utilisateur via l'attribution d'un nom à une ressource.
- ***rdfs :comment :*** un commentaire pouvant être utilisé pour fournir une description lisible à l'utilisateur d'une ressource.

3.5.3 Autre concepts supplémentaires

RDFS définit d'autres classes et propriétés supplémentaires pour représenter les collections et les déclarations RDF, et pour déployer des descriptions de vocabulaire RDF sur le Web [Dan & Guha 2014].

Les conteneurs

Les conteneurs sont des ressources utilisées pour représenter les collections de ressources. Une ressource peut apparaître dans une collection plusieurs fois. RDF définie trois types de conteneur [Dan & Guha 2014] :

- ***"rdf :Bag" :*** l'ordre des ressources n'est pas important.
- ***"rdf :Seq" :*** contrairement au "rdf :Bag", l'ordre des ressources est significatif.
- ***"rdf :Alt" :*** le traitement des ressources consiste à sélectionner un des membres.

En outre RDFS définit un ensemble de classes et propriétés parmi lesquelles on mentionne :

- ***rdfs :Container :*** c'est une super classe des classes de conteneurs RDF (rdf :Bag, rdf :Seq, rdf :Alt).
- ***rdfs :ContainerMembershipProperty :*** c'est une classe qui a pour instances les propriétés *"rdf :_1"*, *"rdf :_2"*, *"rdf :_3"*, etc. Ces instances sont utilisées pour indiquer qu'une ressource est un membre d'un conteneur. *"rdfs :ContainerMembershipProperty"* est une sous-classe de *"rdf :Property"*.
- ***rdfs :membre :*** est une instance de *"rdf :Property"* qui est une super-propriété de toutes les propriétés d'appartenance au conteneur.

Les collections RDF

Une collection RDF représente une liste d'éléments. RDF de base ne définit aucun mécanisme pour indiquer qu'un conteneur n'a plus de membres. pour décrire une collection fermée, c'est-à-dire qu'elle ne peut avoir plus de membres, on utilise le vocabulaire de collection RDF suivant : *"rdf :list"*, *"rdf :first"*, *"rdf :rest"* et *"rdf :nil :"* [Dan & Guha 2014].

3.6 Le Langage Web d'ontologie OWL

Les machines besoin d'une tâche de raisonnement pour exploiter les données. Le langage qui supporte cette tâche doit dépasser la sémantique de base de RDFS. Cela nécessite un langage plus expressif tel que OWL. La couche juste au-dessus de RDF et RDFS requis pour le Web sémantique est bien la couche d'ontologie OWL [Deborah & v. Harmelen 2004].
OWL est l'acronyme de « *Ontology Web Language* ». C'est un langage sémantique pour représenter les connaissances dans le Web. Il fournit un mécanisme aux programmes informatiques pour exploiter les connaissances exprimées en OWL, et ce à travers des déductions logiques [OWL-Working-Group 2012].
OWL a été développé par le groupe *"W3C Web Ontology Working Group"* en 2004. En 2009, ils ont publiée une autre version appelée OWL2. Cette version est aussi étendue en deuxième édition en 2012 [OWL-Working-Group 2012].
OWL fournit plus de vocabulaire que RDFS pour mieux décrire les classes et les propriétés, par exemple la notion de disjonction entre les classes, les classes énumérées et la symétrie des propriétés. Dans cette partie, on va aborder quelques notions d'OWL utiles pour notre problème.

3.6.1 Les types d'OWL

Le langage OWL offre trois types de langage, de plus en plus expressif, à savoir *OWL Lite*, *OWL DL* et *OWL Full*.

OWL Lite

OWL Lite permet d'exprimer des concepts simples, des hiérarchies de classification et des contraintes simples. Il est plus facile d'utiliser OWL Lite que d'utiliser ses parents les plus expressifs (OWL DL et OWL Full). Cependant, il a des limites d'utilisation des fonctionnalités par rapport à ses parents, parmi lesquelles on cite [Deborah & v. Harmelen 2004] :

- Les classes ne peuvent être définies qu'en termes de super-classes nommées, et seuls certains types de restrictions de classe peuvent être utilisés.
- OWL Lite utilisent uniquement les classes nommées.
- L'équivalence n'est pas autorisée qu'entre les classes nommées.
- Les seules cardinalités permises sont 0 ou 1.

En outre le vocabulaire de RDFS, OWL Lite définit son propre vocabulaire. Le Tableau 3.2 donne un synopsis sur le vocabulaire d'OWL Lite.

Table 3.2 – *Les constructeurs de OWL Lite*

Type	Constructeur	Déscription
Information d'entête	Ontology	Pour définir une ontologie
	Imports	fait référence à une autre ontologie OWL contenant des définitions

Egalité et Inégalité	equivalentClass	permet d'indiquer que deux propriétés sont équivalentes, ou pour créer un synonyme d'une propriété.
	equivalentProperty	permet d'indiquer que deux propriétés sont équivalentes, ou pour créer un synonyme d'une propriété.
	sameAs	pour déclarer que deux individus sont identiques.
	differentFrom	pour indiquer qu'un individu est différent aux autres individus.
	AllDifferent	pour indiquer que tous les membres d'une liste sont distincts.
	distinctMembers	pour déclarer que deux individus sont identiques.
Caractéristiques d'une propriété	ObjectProperty	pour indiquer les relations entre les individus.
	DatatypeProperty	pour indiquer les relations entre les individus et les valeurs de données.
	inverseOf	pour indiquer que deux propriétés sont inverses ; si P1 inversOf P2 et x P2 y alors y P1 x.
	TransitiveProperty	des propriétés peuvent être déclarées transitives. pour garantir la décidabilité des langages OWL Lite et OWL DL, il faut que la cardinalité maximale des propriétés transitives et leurs super-propriétés ne puissent pas avoir une restriction de cardinalité maximale égale à 1.
	SymmetricProperty	pour déclarer que deux propriétés sont symétriques.

	FunctionalProperty	une propriété peut avoir une valeur unique, et est appelée aussi propriété unique. Elle signifie que sa cardinalité minimale est égale à zéro et sa cardinalité maximale est égale à 1.
	InverseFunctional-Property	Les propriétés peuvent être définies comme étant inverses. Une telle propriété déclarée comme InverseFunctionalProperty, son inverse est une propriété fonctionnelle. Cette caractéristique est appelée propriété sans ambiguité.
Restriction sur la propriété	Restriction	permet de restreindre l'utilisation des propriétés par les instances d'une classe.
	onProperty	pour indiquer la propriété restreinte.
	allValuesFrom	pour déclarer une restriction sur une propriété par rapport à une classe. Cela signifie que cette propriété sur cette classe est associée à une restriction de plage locale. allValuesFrom nécessite que les valeurs de la propriété de chaque instance de la classe (la classe qui possède la restriction allValuesFrom), soient des membres de la classe indiquée dans "owl :allValuesFrom".

	someValuesFrom	pour déclarer une restriction sur une propriété par rapport à une classe. Contrairement à allValuesFrom et someValuesFrom, elle ne limite pas toutes les valeurs de la propriété à des instances de la même classe. Il peut y avoir des valeurs déférentes aux instances de la classe indiquée dans "owl :someValuesFrom".
Restriction sur la cardinalité	minCardinality (only 0 or 1)	pour mentionner la cardinalité minimale d'une propriété (égale à 0 ou 1).
	maxCardinality (only 0 or 1)	pour mentionner la cardinalité maximale d'une propriété (égale à 0 ou 1).
	cardinality (only 0 or 1)	cette propriété est utilisée lorsqu'on a une cardinalité minimale = la cardinalité maximale= 0 ou 1.
Intersection de classe	intersectionOf	pour indiquer qu'une classe est une intersection de deux classes.
Annotation de propriété	rdfs :label	Pour associer une étiquette à une ontologie.
	rdfs :comment	fait référence à une autre ontologie OWL contenant des définitions.
	Il existe d'autres annotations telles que : rdfs :seeAlso, rdfs :isDefinedBy AnnotationProperty et OntologyProperty.	

OWL DL

OWL DL fournit une expressivité maximale tout en conservant les caractéristiques suivantes :

- La complétude de calcul : toutes les conclusions doivent d'être calculées.
- La décidabilité : tous les calculs doivent être terminés.

OWL DL inclut tous les constructeurs qui ont été définis par le langage OWL Lite. OWL DL se base sur les logiques de description [Deborah & v. Harmelen 2004]. OWL DL met en place certaines contraintes sur l'utilisation du langage OWL. Ces contraintes sont basées sur la logique de description (DL) pour fournir un mécanisme de raisonnement aux utilisateurs d'ontologie. Parmi ces contraintes on cite [Bechhofer *et al.* 2004] :

- OWL DL nécessite une séparation entre ses constructeurs : les classes, les types de données, les propriétés de type de données, les propriétés des objets, les propriétés des annotations, les propriétés des versions, les individus, les valeurs et le vocabulaire intégré. Par exemple, une classe ne peut pas être à la fois un individu et une classe. Une propriété ne peut pas être un individu ou une classe.
- Les propriétés d'objet et les propriétés du type de données doivent être disjointes. Cela implique que les propriétés suivantes : *"inverse of"*, *"inverse functional"*, *"symmetric"*, et *"transitive"*, ne peuvent jamais être utilisées avec les propriétés de type de données.
- OWL DL n'exige aucune contrainte de cardinalité sur les propriétés transitives, inverses et super-propriétés.
- Les annotations sont restreintes.
- La plupart du vocabulaire RDFS ne peut pas être utilisé dans OWL DL.
- Les axiomes doivent être bien formés, et forment une arborescente.
- Les axiomes d'égalité d'individu doivent être appliqués aux individus nommés.

OWL Full

OWL Full et OWL DL utilisent tous les deux le même vocabulaire, sachant qu'OWL Full offre une liberté syntaxique mais sans garantie de calcul, d'où OWL Full est destiné aux utilisateurs voulant une expressivité maximale. Par exemple, on peut traiter une classe comme une collection d'individus et comme un seul individu simultanément. OWL Full permet à une ontologie d'augmenter la sémantique du vocabulaire RDFS ou OWL. Il est presque impossible qu'un logiciel de raisonnement prend en charge un raisonnement complet pour toutes les fonctionnalités du langage OWL full [Deborah & v. Harmelen 2004]. Les langages OWL DL et OWL full ajoutent un ensemble de constructions au langage OWL Lite. Le Tableau 3.3 donne un synopsis sur ces constructeurs [Deborah & v. Harmelen 2004] :

TABLE 3.3 – *Les constructeurs OWL DL et Full*

Type	**Constructeur**	**Déscription**
Les axiomes de classe	`oneOf, dataRange`	`pour créer une classe énumérée. Les classes peuvent être décrites par énumération des individus qui composent la classe. Les instances de la classe sont l'ensemble des individus énumérés, ni plus ni moins.`
	`disjointWith`	`les classes peuvent être déclarées disjointes les unes des autres.`
	`equivalentClass`	`pour indiquer qu'une classe est équivalente à une autre classe.`

	rdfs :subClassOf	pour indiquer qu'une classe est une sous-classe d'une autre classe.
Combinaison de classes	unionOf	pour construire une classe à partir d'autres classes. Il contient comme individus l'union des individus des autres classes.
	complementOf	pour définir le complément d'une classe.
	intersectionOf	pour définir une classe qui contient comme individus l'intersection d'autres classes.
Cardinalité arbitraire	minCardinality	pour indiquer la cardinalité minimale d'une propriété.
	maxCardinality	pour indiquer la cardinalité maximale d'une propriété.
	cardinality	pour indiquer une cardinalité complète d'une propriété.
Filtrage d'information	hasValue	une propriété peut avoir besoin d'un certain individu en tant que valeur (également appelée parfois valeurs de propriété).

3.7 Le langage SPARQL

SPARQL est un langage de requête pour les données en format RDF (Prud'hommeaux et Seaborne 2008). Pour une meilleure compréhension du langage SPARQL, il est indispensable de représenter les ressources RDF sous forme de triplets. SPARQL peut être utilisé pour [Dragan *et al.* 2009] :

- Extraire des informations sous forme d'URI, nœuds blancs et littéraux à partir d'un graphe RDF.
- Extraction des sous-graphes.
- Construction des nouveaux graphes RDF à partir des graphiques interrogés.

Une requête SPARQL consiste à évaluer ou faire une correspondance entre un pattern graphique et un graphe RDF de la requête. Un pattern est vu comme un graphe RDF qui peut contenir des variables dans certains nœuds (ressources) ou prédicats. Le modèle graphique le plus simple ressemble à un triplet RDF unique (ressource propriété valeur) [Dragan *et al.* 2009].
La Figure 3.4 représente deux triplets RDF qui correspondent tous les deux au modèle de triplet de la Figure 3.5. Une liaison est une correspondance entre une

variable de pattern (requête) et les termes de graphe RDF. Chaque triplet de la Figure 3.4 est une solution pour le pattern de la Figure 3.5. Le résultat de la requête SPARQL est un ensemble de patterns [Dragan *et al.* 2009]. Le résultat de la requête représentée dans la Figure 3.5 est la solution du modèle de la Figure 3.4.

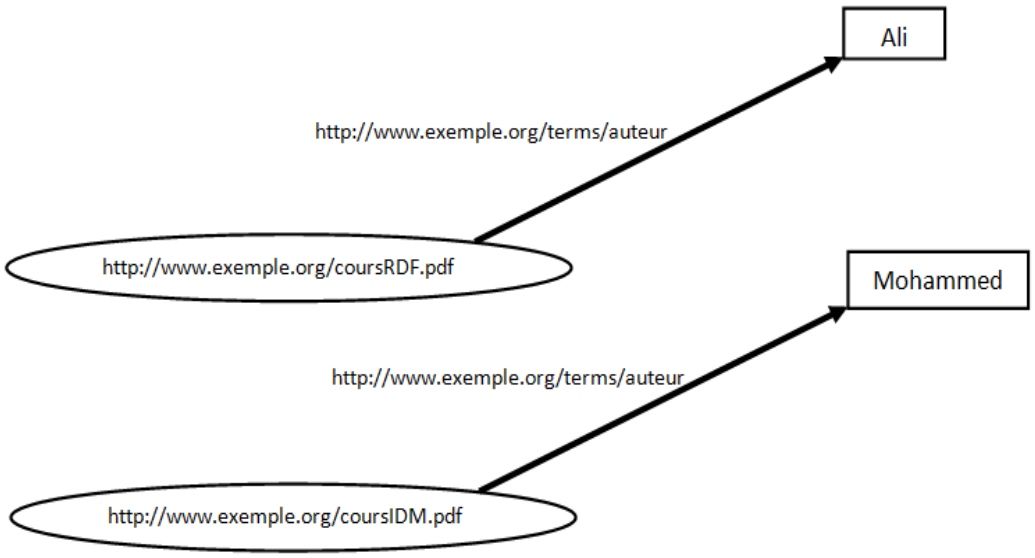

Figure 3.4 – *Triplet RDF simples*

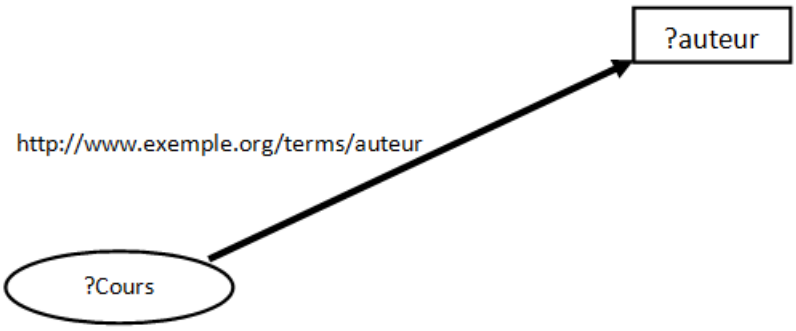

Figure 3.5 – *Graphe d'une requête SPARQL simple*

La requête SPARQL correspond au modèle de la Figure 3.5 est donné par le Listing 3.7.

```
1 PREFIX aut: <http://www.exemple.org/terms/>
2 SELECT ?Cours, ?auteur
3 WHERE {?Cours aut:auteur ?auteur }
```

Listing 3.7 – *Requête SPARQL simple*

3.8 Conclusion

Nous avons vu dans ce chapitre que le Web sémantique est une extension du Web classique. A travers le Web sémantique, on peut présenter les connaissances

d'une façon sémantique à l'aide des langages appropriés qui sont définis dans la pile du Web sémantique. Parmi ces langages on cite RDF, RDFS et OWL. L'un des avantages du Web sémantique est qu'il permet aux machines de communiquer entre elles.
Dans le Web sémantique, les données sont présentées à l'aide de RDF sous forme de triplets. Un triplet est composé d'un sujet, d'une propriété et d'un objet. Ces composantes sont représentées par des URI, ce qui rend les données, liées entre elles.

Ingénierie Dirigée par les Modèles

4

Sommaire

4.1 Introduction . . . 45
4.2 Définition de MDA . . . 46
4.2.1 Pourquoi MDA ? . . . 46
4.3 Architecture de l'approche MDA . . . 46
4.3.1 Le modèle d'exigences CIM (Computation Independent Model) . . . 47
4.3.2 Le modèle abstrait d'analyse et de conception PIM (Platform Independent Model) . . . 48
4.3.3 Le modèle concret de code ou de conception PSM (Platform Specific Model) . . . 48
4.4 Transformation des modèles . . . 49
4.5 Les Modèles et les Méta-modèles . . . 49
4.5.1 Les modèles . . . 49
4.5.2 Les méta-modèles . . . 49
4.5.3 Relation entre un modèle et son méta-modèle . . . 50
4.5.4 Méta méta-modèle . . . 50
4.6 Architecture à quatre niveaux de l'IDM . . . 51
4.7 Les Langages de modélisation . . . 52
4.7.1 UML . . . 52
4.7.2 MOF 2.0 . . . 54
4.7.3 Le Framework de modélisation d'Eclipse (Eclipse Modeling Framework EMF) . . . 54
4.8 Les modèles en XML . . . 56
4.8.1 Le format XML . . . 56
4.8.2 Le standard XMI (XML Metadata Interchange) . . . 56
4.8.3 DI (Diagram Interchange) . . . 59
4.9 Les Langages de transformation des modèles dans l'IDM . . . 60
4.9.1 Le langage QVT (Query/View/Transformation) . . . 60
4.9.2 Le langage de transformation ATLAS (ATL . . . 62
4.10 Conclusion . . . 63

4.1 Introduction

Dans ce chapitre, nous allons présenter les concepts de base de l'ingénierie dirigée par les modèles (IDM). IDM fournit plusieurs avantages aux différents éléments d'élaboration d'un logiciel. Parmi ces bénéfices on cite la transformation, la synchronisation et la gestion. L'intérêt de l'IDM est de partager la complexité sur différents niveaux d'abstraction allant des modèles conceptuels de haut niveau jusqu'aux bas niveaux.
Dans la littérature, l'IDM comporte deux parties : la première sert à la définition des méta-modèles pour décrire la structure et la sémantique du langage, et les modèles qui représentent des éléments du monde réel. La deuxième partie concerne les transformations entre les modèles ou bien les langages.
Schmidt [Schmidt 2006], affirme que la technologie d'IDM devrait utiliser à la fois des langages spécifiques à un domaine bien précis et des transformations. En outre, d'autres auteurs [Kent 2002] voient que l'IDM a besoin de l'introduction d'un processus associé à l'étape de conception des langages et des transformations.
Selon Miller et Mukerji [Miller & Mukerji 2003], l'architecture dirigée par les modèles (ADM) est une instance de l'IDM, et elle est basée sur le standard MOF (Meta-Object Facility) de l'OMG qui s'intéresse à la représentation du modèle et du méta-modèle, et leurs manipulations. Souvent, il inclut le langage UML pour modéliser les modèles et les méta-modèles, et les modèles de transformation et de gestion communes. Un modèle indépendant de la plateforme (PIM) est transformé en un modèle spécifique de plateforme (PSM) [Kleppe *et al.* 2002].
Favre et Nguyen [Favre & Nguyen 2005] ont proposé un méga-modèle qui permet de décrire les différents concepts de base de l'IDM à savoir le modèle, le méta-modèle, le langage de modélisation et la transformation de modèle. La Figure 4.1 récapitule ces concepts.

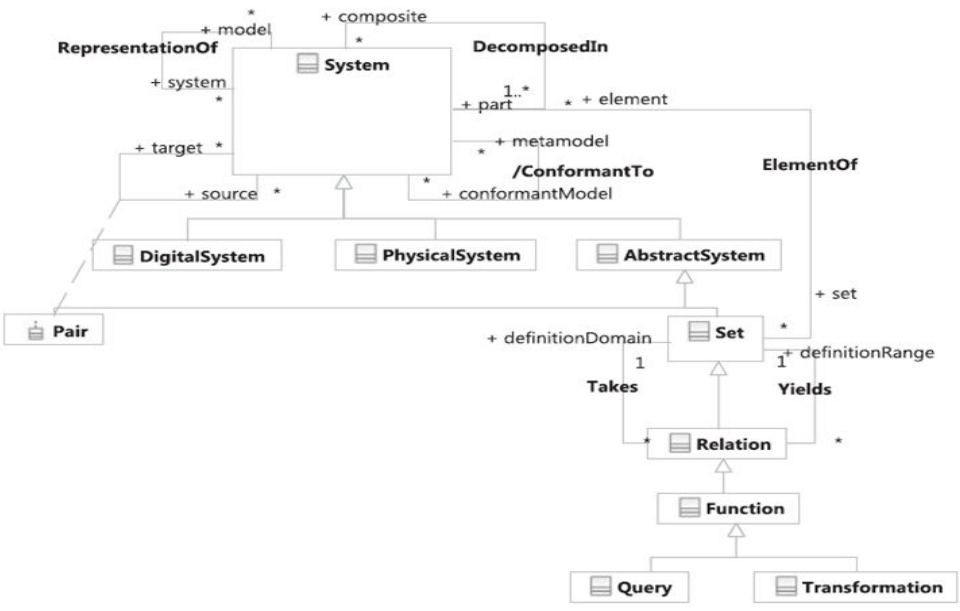

Figure 4.1 – *Le méga-modèle de Favre (L. Favre,2010)*

4.2 Définition de MDA

Architecture dirigée par les modèles (en anglais Model Driven Architecture -MDA-) est un Framework créé par OMG (Object Mangement Group) afin d'améliorer la portabilité, l'interopérabilité et la réutilisabilité des systèmes [OMG 2003a]. MDA est basé sur différentes normes d'OMG, telles que le langage UML (Unified Modeling Language), XMI [OMG 2007] et CORBA [OMG 2012]. MDA utilise les modèles dans toutes les différentes phases de construction d'un système, notamment la spécification des exigences, la description de l'architecture, la description de la conception et l'écriture du code. Elle offre une approche pour décrire une spécification d'un système indépendamment des plateformes, et par la suite cette spécification est transformée en une implémentation correspondant à une plateforme désirée.

4.2.1 Pourquoi MDA ?

OMG définie MDA pour plusieurs raisons parmi lesquelles on cite [Favre 2010] :

- Assurer l'interopérabilité des applications.
- Intégration d'applications d'entreprise.
- Intégration de différents middlewares sur Internet.
- Amélioration de la productivité.
- Amélioration de qualité du code.
- Amélioration des processus et des coûts de maintenance des logiciels.

4.3 Architecture de l'approche MDA

La Figure 4.2 montre bien l'architecture générale de l'approche MDA. On voit que cette architecture est constituée de plusieurs niveaux. Chacun est formé d'un ensemble de modèles. Le premier niveau contient un ou plusieurs modèles d'exigences (CIM), puisque chaque application doit commencer par l'élaboration des besoins de client. Dans le deuxième niveau, on trouve les modèles de conception et d'analyse (PIM). Théoriquement, ces modèles doivent être générés à partir des modèles CIM pour assurer les liens de traçabilité entre les modèles. Le troisième niveau contient des modèles qui permettent de réaliser l'application. Ces modèles sont obtenus par une transformation des modèles PIM en y ajoutant les informations techniques relatives aux plateformes d'exécution (PSM). Le dernier niveau représente le code de l'application qui est généré à partir des modèles PSM. L'approche MDA ne considère pas la génération de code comme une transformation, mais plutôt une traduction des modèles PSM dans un formalisme textuel [Xavier 2005].

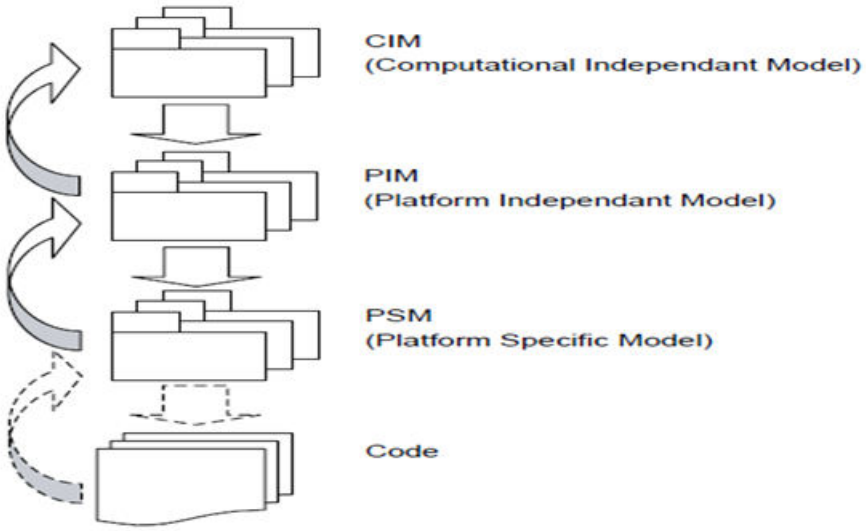

FIGURE 4.2 – *Architecture de l'approche MDA (Xavier,2005)*

D'après la Figure 4.2, on peut noter que la vocation de l'approche MDA ne se limite pas à la création de nouvelles applications, mais aussi elle procure d'autres fonctionnalités, à savoir la rétro-conception d'applications existantes, et ceci est effectué par des transformations inverses entre les modèles [Xavier 2005].
L'objectif implicite de MDA est de gérer les transformations entre les différents modèles du système. Par exemple, produire des modèles et des codes pour des plateformes spécifiques (PSM) à partir des modèles d'abstraction plus élevés qui sont développés indépendamment d'une plateforme particulière (PIM), c.à.d., la transformation d'un PIM vers PSM, et par la suite, dériver le code directement à partir de ce dernier ; ce processus est le cœur de MDA et il est connu sous le nom de l'ingénierie directe dirigée par les modèles IDM (en anglais MDE). Il comporte les étapes suivantes : élaborer un CIM, transformer le CIM en un PIM, puis transformer le PIM en un ou plusieurs PSM, et finalement générer le code directement à partir des PSM [Favre 2010].
OMG offre le standard MOF (Meta Object Facility), qui est un Framework qui permet de représenter ou définir les méta-modèles. Il fournit une gamme de services aux développeurs afin de faciliter le développement et l'interopérabilité des systèmes dirigés par les modèles[OMG 2007].
Un méta-modèle est un langage abstrait pour représenter différents types de modèles et de données. Le Framework de méta-modélisation est basé sur des architectures à quatre niveaux, à savoir le modèle objet, le modèle, le méta-modèle et le méta-méta-modèle [Favre 2010].

4.3.1 Le modèle d'exigences CIM (Computation Independent Model)

Le modèle d'exigences représente la première phase dans le cycle de développement d'une application. Il permet de décrire les demandes du client. Il peut même être considéré comme une référence lorsqu'on voudra s'assurer qu'une application est conforme aux exigences du client. Il représente aussi l'application dans son environnement afin d'identifier les services offerts par l'application. L'utilisation d'un tel modèle a une importance capitale. Cela est illustré dans les liens de traçabilité avec les modèles qui seront construits dans les autres phases

dans le cycle de développement de l'application.
Notons que dans le vocabulaire de l'ingénierie dirigée par les modèles, ce modèle est appelé le modèle indépendant de la programmation (en anglais, Computation Independent Model (CIM)). Cela signifie que les modèles d'exigences ne contiennent aucune information, ni sur la réalisation, ni sur les traitements.
Dans le langage UML, le modèle d'exigences est réalisé grâce à un diagramme de cas d'utilisation. Car ce diagramme permet de représenter les fonctionnalités fournies par l'application à l'aide des cas d'utilisation, ainsi que les entités qui interagissent avec elles à travers les acteurs. Cela est fait sans apporter de l'information sur le fonctionnement de l'application [Xavier 2005].

4.3.2 Le modèle abstrait d'analyse et de conception PIM (Platform Independent Model)

Après le modèle d'exigences, vient une phase cruciale qui est l'analyse et la conception. Cette phase est réalisée par plusieurs méthodes, telles que les méthodes classiques : Merise et Coad/Yourdon, et les méthodes objet : Schlear et Mellor, OMT, OOSE et Booch, et UML. Chaque méthode représente le système par ses propres modèles.
Parfois, on utilise des patterns ou modèles de conception (Design Patterns) du GoF (Gang of Four) dans la conception, afin de structurer l'application en modules et sous modules. Ces modèles sont dédiés à certaines plateformes. Par contre, dans les modèles PIM (Platform Independent model), nous ne considérons que la conception abstraite, c'est-à-dire, l'utilisation des modèles qui sont réalisés sans aucune connaissance des techniques d'implémentation.
L'approche MDA a préconisé d'utiliser le langage UML pour établir les modèles d'analyse et de conception, puisqu'elle est indépendante de toutes les plateformes d'implémentation (J2EE, .Net, PHP, etc.). Les détails d'implémentation sont intégrés très tard dans le cycle de développement de l'application. En outre, l'approche MDA ne donne aucune interdiction d'utilisation d'autres langages, ni du nombre de modèles, ni des indications sur la façon d'élaboration de ces modèles. Le rôle des modèles PIM se résume à l'établissement d'une liaison entre les modèles d'exigences et le code de l'application, quelque soit le langage utilisé pour le développent. Ces modèles PIM doivent être précis et doivent contenir suffisamment d'information afin d'assurer la génération de automatique du code [Xavier 2005].

4.3.3 Le modèle concret de code ou de conception PSM (Platform Specific Model)

Dans le vocabulaire MDA, les PSM (Platform Specific Model) sont des modèles de code spécifique à une plateforme d'exécution. Ces modèles de code servent à faciliter la génération de code d'une application à partir d'un modèle d'analyse et de conception (PIM). Ils contiennent des informations d'exploitation d'une plateforme d'exécution. Ils sont productifs, mais ne sont pas forcément pérennes.
L'approche MDA propose l'utilisation des profils UML pour construire des modèles de code. Un profil UML est une adaptation du langage UML à un domaine particulier, comme le profil UML d'EJB (concerne le domaine EJB).

En outre la génération de code, les modèles de code (PSM) nous fournissent d'autres caractéristique, et ils font le lien avec les plateformes d'exécution [Xavier 2005].

4.4 Transformation des modèles

Nous avons mentionné précédemment qu'il important de bien établir des liens de traçabilité entre les modèles. En effet ces liens sont automatiquement établis grâce à l'exécution des transformations entre les modèles.
Les transformations, qui sont définies dans MDA, sont essentiellement les CIM vers PIM et PIM vers PSM, ou des transformations inverses (rétro-ingénierie), telles que code source vers PSM, PSM vers PIM et PIM vers CIM.
Une transformation de modèle est considérée comme une application. Selon l'approche MDA, les différentes étapes d'une transformation de modèle peuvent être modélisées. Cette modélisation nous a permis de définir un ensemble de modèles tels que, les modèles d'exigences, d'analyses, de conceptions et de codes [Xavier 2005].

4.5 Les Modèles et les Méta-modèles

L'approche MDA préconise d'utiliser les modèles durant les différentes phases de cycle de vie d'une application. Ces modèles sont créés et structurés par d'autres modèles appelants les méta-modèles. A travers ces méta-modèles, la pérennité des modèles est assurée. Nous avons vu qu'il existe différents types de modèles, tels que CIM, PIM, PSM, etc. Chacun d'eux est élaboré par un formalisme bien défini. L'OMG définit le standard MOF (Meta Objet Facility) pour définir les méta-modèles. donc le standard MOF apporte un support de définition des formalismes de modélisation [Xavier 2005].

4.5.1 Les modèles

Un modèle est une représentation d'un système et son environnement. En MDA, un modèle est défini par le biais d'un langage de modélisation bien précis. MDA distingue plusieurs types de modèle selon les niveaux d'abstraction, notamment les CIM, PIM, PSM et ISM (code) [Xavier 2005].

4.5.2 Les méta-modèles

Les méta-modèles représentent le cœur de MDA. Ils sont représentés sous forme de modèles, ce qui rend la structure des modèles pérennes. Un méta-modèle donne la structure d'un modèle par la définition de ses entités, ainsi que les propriétés de leur connexion et de ses règles de cohérence. Donc, tout modèle doit être conforme à la structure définie par son méta-modèle [Favre 2010].
L'OMG définit le standard MOF (Meta Objet Facility) pour définir les méta-modèles. Le standard MOF apporte un support de définition de formalismes de modélisation sous forme de diagrammes de classes [Xavier 2005].
MOF structure un méta-modèle par un ensemble de méta-classes avec des méta-associations, etc. cette structuration propose un mécanisme générique, tel que XMI (XML Metadata Interchange). Afin de stocker les modèles en format XML,

le XMI construit des grammaires XML à partir du méta-modèle qui correspond à ces modèles.

4.5.3 Relation entre un modèle et son méta-modèle

La Figure 4.3 (a) représente un méta-modèle d'un diagramme de cas d'utilisation selon la définition suivante : un diagramme de cas d'utilisation contient un système qui est intitulé par un nom. Ce système contient un ensemble d'acteurs et de cas d'utilisation. Un acteur est identifié par un nom. Il peut hériter un autre acteur. Les acteurs sont reliés avec des cas d'utilisation. Un cas d'utilisation est identifié par un nom. Il peut étendre ou inclure d'autres cas d'utilisation.

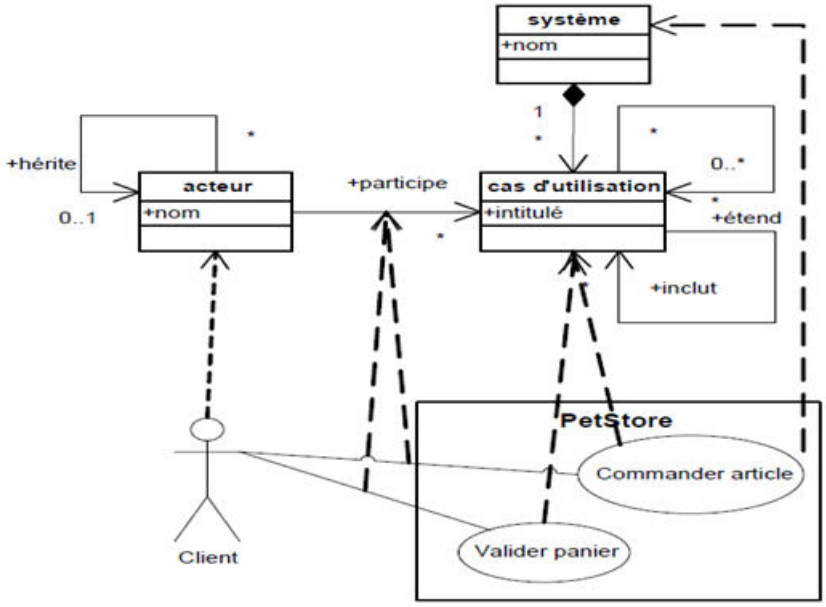

FIGURE 4.3 – *Relations entre un modèle (diagramme de cas d'utilisation) et son méta-modèle (Xavier, 2005)*

La Figure 4.3 montre la relation qui existe entre un modèle côté (b) et son méta-modèle côté (a). L'instanciation de ce méta-modèle nous donne le modèle de la Figure 4.3 côté (b). Ce modèle représente un diagramme de cas d'utilisation. Les flèches pointillées représentent les relations existantes entre les éléments du méta-modèle et les éléments du diagramme de cas d'utilisation (modèle). Nous voyons que l'acteur "client" est une instance de la classe "acteur". Les cas d'utilisation "commander article" et "voir panier" sont aussi des instances de la classe "cas d'utilisation". Les relations existantes entre le "client" et les cas d'utilisation sont des instances de la relation "participe". Donc, toutes ces instances doivent être conformes à leur méta-modèle. D'où, on peut considérer un modèle comme une instance de son méta-modèle, et qu'il doit être conforme ce méta-modèle [Xavier 2005].

4.5.4 Méta méta-modèle

OMG définit le standard MOF (Meta Objet Facility) en tant que méta-méta-modèle. Le MOF peut représenter n'importe quel méta-modèle grâce aux moyens

de structuration qu'il offre. La relation qui existe entre un modèle et son méta-modèle est la même que celle qui existe entre le standard MOF et les méta-modèles. La Figure 4.4 illustre une partie de MOF sous forme d'un diagramme de classes. Les concepts de MOF, tels que la méta-classe, le méta-attribut, la méta-opération, le type de données et le package, sont représentés par des classes, tandis que les relations entre les concepts de MOF sont représentées par des associations. Donc, ce diagramme de classes est considéré comme un méta-modèle des méta-modèles, autrement dit un méta-méta-modèle.

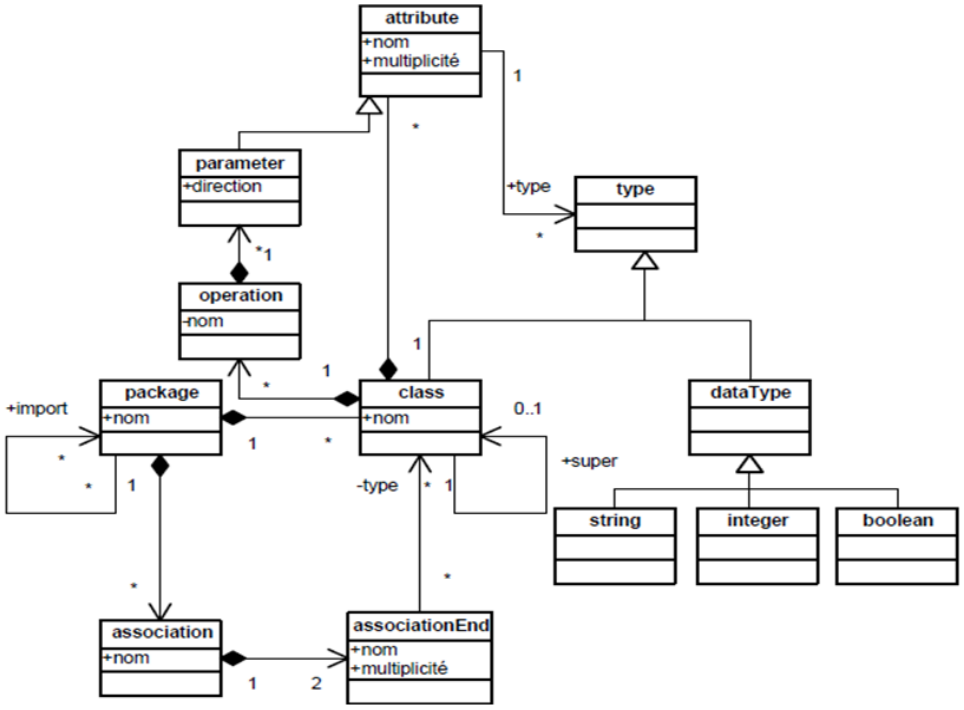

FIGURE 4.4 – *Le méta méta-modèle MOF 1.4 (Xavier, 2005)*

La question qui se pose : le niveau méta méta-modèle nécessite-t-il l'existence d'un autre niveau élevé que lui ? c.à.d. un niveau méta méta-méta-modèle !!
pour répondre à cette question, nous supposons qu'il existe un niveau supérieur est le niveau méta méta-méta-modèle, qu'il peut décrire la structure d'un méta méta-modèle. Maintenant si on veut modéliser ce niveau (méta méta-méta-modèle) sous forme de diagramme de classes, nous aboutirions à dessiner le même diagramme de la figure 4.4. de ce fait, on dit que le méta méta-modèle s'auto définit. Donc, il n'y a que le niveau-modèle, le niveau méta-modèle et le niveau méta méta-modèle (MOF) [Xavier 2005].

4.6 Architecture à quatre niveaux de l'IDM

À partir de ces définitions, nous pouvons représenter la fameuse architecture à quatre niveaux (couches) de MDA. La Figure 4.5 illustre cette architecture. On voit que cette architecture est formée d'un ensemble de niveaux (couches) M0, M1, M2 et M3.

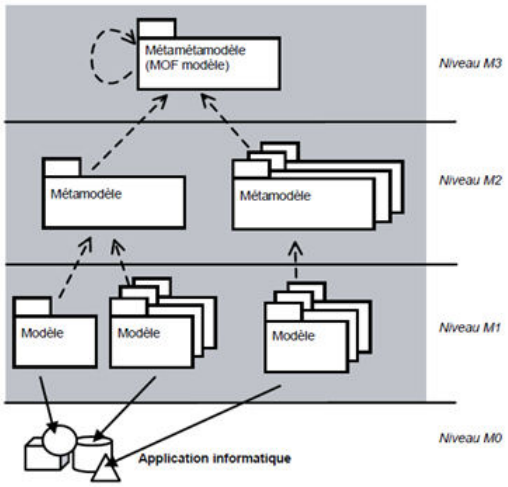

FIGURE 4.5 – *Architecture à quatre niveaux (Xavier, 2005)*

La couche la plus basse Mo est la couche d'information ou la couche du monde réel. Elle contient les entités à modéliser qui sont définies dans la couche M1.
La couche M1 s'appelle aussi la couche des modèles. Elle définit les langages pour représenter les données provenant de la couche Mo. Les modèles de cette couche doivent être conformes au méta-modèle décrit dans la couche M2.
La couche M2 est la couche des méta-modèles. Les méta-modèles de cette couche offrent un langage pour décrire les modèles représentées au niveau M1. Un modèle est une instance d'un méta-modèle. Le méta-modèle UML est un exemple de méta-modèle qui est conforme à MOF.
La couche M3 est la couche méta-méta-modèle, par exemple, le MOF. Le MOF est un Framework pour la spécification, la construction et la gestion des méta-modèles. A travers MOF, on définit les langages de modélisation, tels que l'UML ou le MOF lui-même. Donc, Un méta-méta-modèle nous permet de définir un ensemble de méta-modèles. Notons qu'un méta-méta-modèle est conforme à lui-même [Favre 2010].

4.7 Les Langages de modélisation

4.7.1 UML

Le langage UML (Unified Modeling Language) a été d'abord défini dans trois versions préliminaires officieuses [Booch & Rumbaugh 1995, Booch *et al.* 1996, Booch *et al.* 1997]. La première version officielle 1.1 definit UML comme suit : « The Unified Modeling Language (UML) is a general purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of a software system » [OMG 1997].
Il existe d'autres versions, telles que la version 1.5 [OMG 2003b] qui est la plus « officielle », et la version 2.0 qui est adoptée [OMG 2003c, OMG 2003d, OMG 2004].
La définition d'UML dans la version 2.0 est : « The Unified Modeling Language is a visual language for specifying, constructing and documenting the artifacts of systems. It is a general purpose modeling language that can be used with all

major object and component methods, and that can be applied to all application domains (e.g. health, finance, telecom, aerospace) and implementation platforms (e.g. *J2EE], .NET*).» [OMG 2003c].
D'après cette définition, UML est devenu un langage de modélisation à usage général, et ce par l'apparition de trois termes : component, domaine et platform. Ces termes signifient que le développement sera basé sur les composants, les aspects métier et les serveurs d'application.

Les diagrammes UML

UML 2.0 [OMG 2004] se compose d'un ensemble de diagrammes qui sont répartis en trois catégories, comme illustré dans la Figure 4.6 :

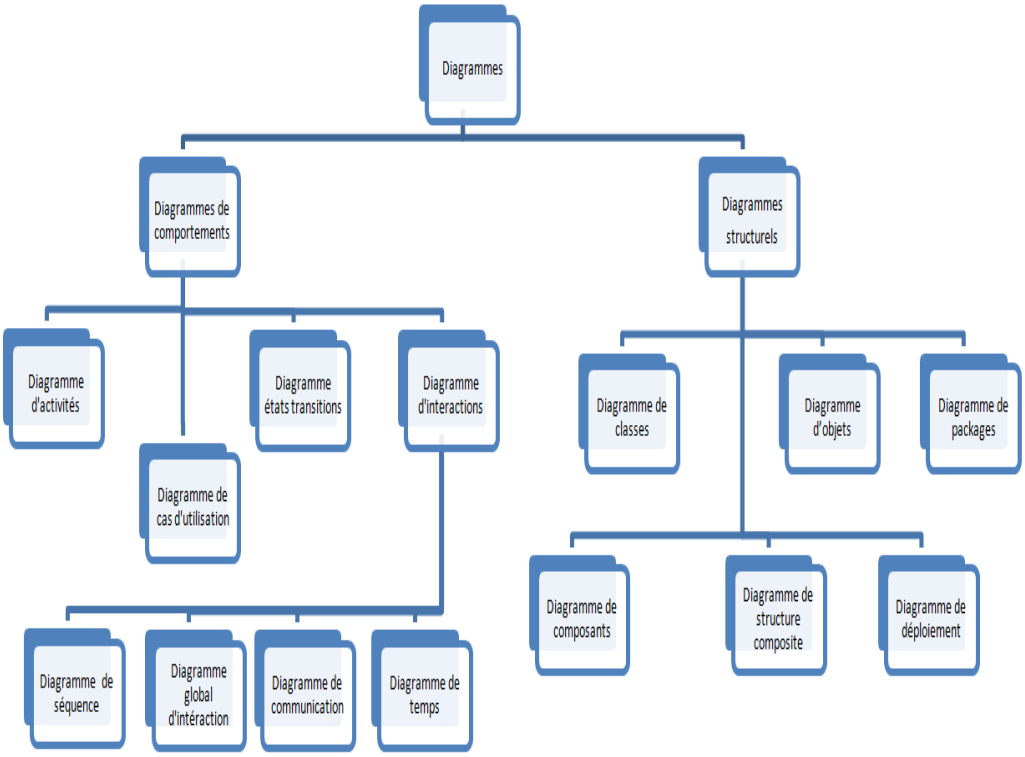

FIGURE 4.6 – *Les diagrammes UML (OMG, 2004)*

- **Les diagrammes structurels ou statiques :** comme le diagramme de classes, le diagramme d'objets, le diagramme de composant, le diagramme de déploiement, le diagramme des paquets, le diagramme de structure composite et le diagramme de profils (depuis UML 2.2).

- **Les diagrammes de comportement :** tels que le diagramme d'activité, le diagramme de séquence et le diagramme d'état transition.
- **Les diagrammes d'interaction ou dynamiques :** tels que le diagramme de séquence, le diagramme de communication, le diagramme global d'interaction et le diagramme de temps (depuis UML 2.3).

Dans l'IDM, on s'intéresse au diagramme de classes, car on représente tous les modèles et les méta-modèles par des diagrammes de classes.

4.7.2 MOF 2.0

La Figure 4.7 montre que le méta méta-modèle *MOF2.0* est composé de deux parties : *EMOF (Essential MOF)* et *CMOF (Complete MOF)*. *EMOF* est utilisé pour élaborer des méta-modèles sans association, tandis que *CMOF* est conçu pour les méta-modèles avec association. En outre, *MOF2.0* intègre le méta-modèle Infrastructure de l'*UML2.0*, *EMOF* intègre le package Basic et *CMOF* le package Constructs. Notons que ces relations d'intégration sont assez complexes [Xavier 2005].

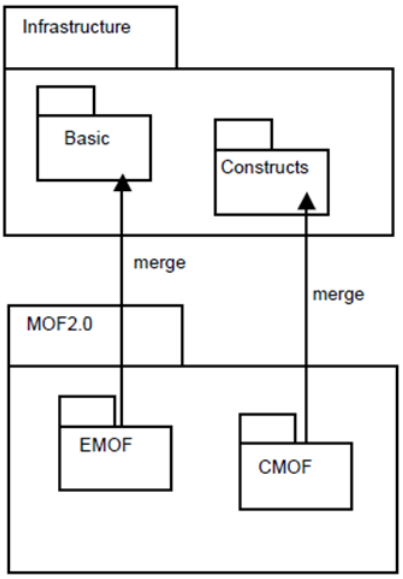

FIGURE 4.7 – *Représentation schématique du méta méta-modèle MOF2.0 (Xavier, 2005)*

4.7.3 Le Framework de modélisation d'Eclipse (Eclipse Modeling Framework EMF)

Le projet EMF est un Framework Open Source. Il est intégré dans la plateforme Éclipse. EMF offre des outils pour faire une modélisation et une méta-modélisation, et pour générer de codes sources (des classes Java), afin de développer des outils ou des applications basées sur des modèles de données structurées en XMI [Steinberg *et al.* 2009].
EMF est constitué de trois éléments, à savoir le noyau EMF (Ecore), les interfaces réflectives et les fonctionnalités de EMF telles que la génération de code EMF [Steinberg *et al.* 2009].

Les interfaces réflectives d'EMF

Le Framework EMF offre des interfaces réflectives pour manipuler les modèles. Ces interfaces réflectives sont spécifiées dans le méta méta-modèle Ecore. On cite parmi ces interfaces *EObject*, *Eclass*, *Epackage* et *EFactory* [Xavier 2005].

EObject : C'est l'interface la plus importante. Elle permet de représenter les éléments du modèle ou du méta-modèle. Elle comporte plusieurs opérations, telles que *eClass()*, *eGet()* et *eSet()*. L'opération *eClass()* permet d'obtenir la classe d'un élément du modèle. Les deux opérations, *eGet()* et *eSet()*, nous

permettent d'accéder aux différentes attributs et références de l'élément [Xavier 2005].

EClass : Cette interface est utilisée pour représenter une méta-classe d'un méta-modèle. Elle fournit des opérations *getEAttributes()* et *getEReferences()*, qui permettent d'obtenir toutes les propriétés contenues dans une méta-classe [Xavier 2005].

EPackage : Cette interface regroupe toutes les interfaces *EClass*. Elle offre une opération *getEClassifier(String qname)* pour accéder à une classe d'un méta-modèle [Xavier 2005].

EFactory : C'est l'interface qui nous permet d'instancier le méta-modèle à l'aide de l'opération *create()* [Xavier 2005].

Le méta méta-modèle ECORE

Le Framework EMF propose son propre méta-méta-modèle Écore. Il se base sur le standard *MOF2.0*. Il ne supporte que la notion de méta-classe. Donc, il ressemble à EMOF. Il est intégré dans la plateforme Éclipse. La Figure 4.8 illustre le méta méta-modèle *Ecore*. Il est clair que les méta-modèles conformes à *Ecore* sont composés seulement *d'EClass* contenant des *EAttribute* et des *EReference*. Il n'a y a pas de méta-association [Xavier 2005].

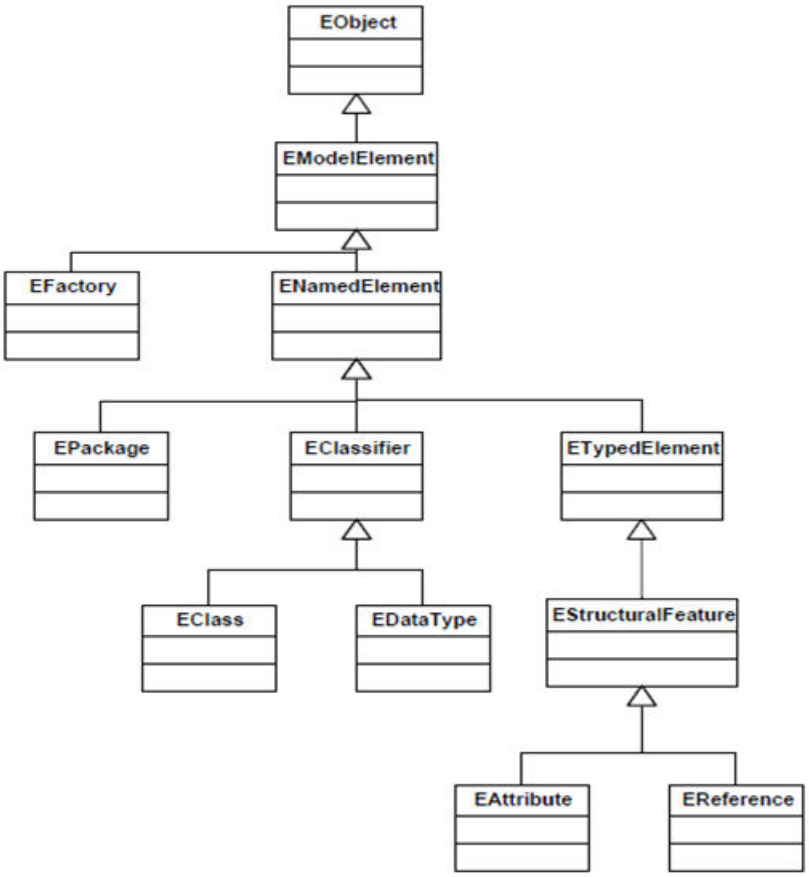

FIGURE 4.8 – *Méta méta-modèle ECORE (Xavier, 2005)*

Fonctionnalités du Framework EMF

Pour une meilleure exploitation et manipulation des méta-modèles et ses modèles instances dans la plateforme Eclipse, EMF offre plusieurs fonctionnalités. La fonctionnalité la plus intéressante est la génération de code. Elle permet de créer un éditeur graphique à partir d'un méta-modèle. L'éditeur permet de représenter les modèles sous forme arborescente où chaque nœud représente une instance d'une méta-classe [Steinberg *et al.* 2009].
L'utilisation d'une telle fonctionnalité est très simple. Il suffit de générer automatiquement les classes Java correspondant à un méta-modèle, puis lancer l'exécution des ces classes pour obtenir l'éditeur graphique des modèles. Notons que EMF propose d'autres fonctionnalités [Xavier 2005].

4.8 Les modèles en XML

Jusqu'à ici, nous avons vu que les modèles sont présentés avec leurs méta-modèles sous forme d'entités abstraites. Dans ce cas, il est presque impossible de parler sur la pérennité des modèles, car les modèles ne peuvent pas être enregistrés dans un support de stockage. Ils restent toujours volatils.
OMG propose les standards *XMI (XML Metadata Interchange)* et *DI (Diagram Interchange)* permettant d'enregistrer les modèles dans des supports de stockage, et par conséquent les rendre pérennes. Ces standards sont basés sur le fameux format international d'échange de données, à savoir XML, et ce grâce à son succès le plus distingué [Xavier 2005].

4.8.1 Le format XML

Le format XML est utilisé pour représenter les modèles. Cette section a été déjà détaillée dans le chapitre précédent (Web sémantique).

4.8.2 Le standard XMI (XML Metadata Interchange)

OMG a proposé un standard XMI pour donner une représentation concrète des modèles sous forme de documents XML, et cela est le but principal de l'XMI. XMI fournit les balises nécessaires et suffisantes à la représentation des modèles en format XML en se basant sur la définition de la structure des balises : XML DTD et XML Schéma.
La figure 4.9 montre comment XMI définit la structuration des balises. XMI est basé sur l'alignement existant entre les modèles et leur méta-modèle, d'un côté, et les documents XML et leur structuration, de l'autre côté. Les méta-modèles et les structurations des balises sont identiques en termes de rôle qu'ils jouent. Tous les deux définissent une structure : le premier définit une structure des modèles, alors que, la seconde définit la structure d'un document XML [Xavier 2005].

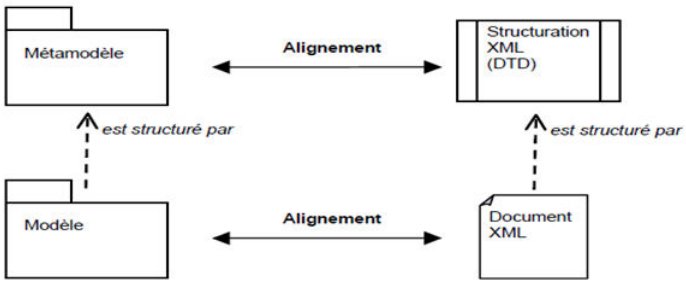

FIGURE 4.9 – *Alignement entre méta-modèle/modèle et DTD/document XML (Xavier, 2005)*

Un document XML est structuré par son DTD, comme c'est le cas pour un modèle qui doit être conforme à son méta-modèle. Les modèles sont représentés sous forme de documents XML en utilisant la structuration des balises (DTD) qui est basée sur les méta-modèles.

Processus de génération des balises XML

La Figure 4.10 montre comment XMI génère automatiquement les balises XML. Cette génération est effectuée en deux processus, à savoir la génération et la sérialisation. Le premier processus est illustré dans la partie (1) de la Figure 4.10. Il permet de définir un ensemble de règles permettant de générer automatiquement une DTD (une structuration de balises XML) à partir d'un méta-modèle. Ces règles de génération sont assez naturelles, et sont expliquées dans la sous-section suivante.
Le deuxième processus est illustré dans la partie (2) de la Figure 4.10 Il permet de représenter un modèle en format *XML* en utilisant la *DTD* définie dans le premier processus [Xavier 2005].

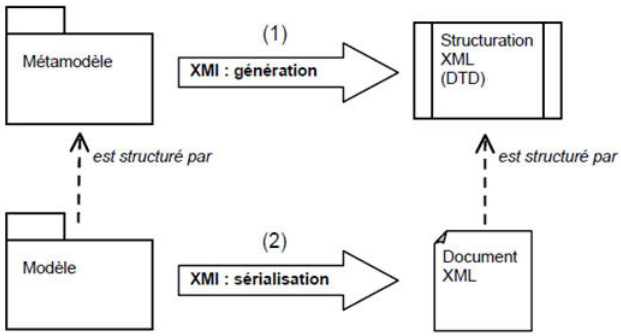

FIGURE 4.10 – *XMI et la structuration de balises XML (Xavier, 2005)*

Règles XMI de génération d'une DTD à partir d'un méta-modèle MOF1.4

Ces règles permettent de produire une DTD à partir d'un méta-modèle MOF1.4 [Xavier 2005] :
Règle 1 : Toute méta-classe produit une balise XML qui possède :

- Un nom identique au nom de la méta-classe.
- Un attribut id pour identifier chaque instance.

Règle 2 : Tout méta-attribut d'une méta-classe produit une sous-balise qui doit être placée à l'intérieur de la balise de la méta-classe. Cette sous-balise possède :

- Un nom identique au nom du méta-attribut.
- Un contenu égal à la valeur du méta-attribut.

Règle 3 : Toute méta-référence d'une méta-classe produit une sous-balise qui doit être contenue dans la balise correspondant à la méta-classe. Cette sous balise possède :

- Un nom identique au nom de la méta-référence.
- Un contenu qui égal à un id afin d'identifier l'instance référenciée.

Règle 4 : Toute méta-association entre deux méta-classes produit une balise possèdant :

- Un nom identique au nom de la méta-association.
- Un contenu qui représente les instances des méta-classes qui sont en relation (ou bien leurs id)

Si la méta-association est navigable alors la balise qui correspondant à la méta-association doit être placée dans la balise correspondant à la méta-classe source de l'association [Xavier 2005].

Exemple de génération d'une DTD et d'un document XML à partir d'un méta-modèle

La Figure 4.11 représente un méta-modèle qui permet de modéliser des processus simples. Chaque processus est défini par un ensemble d'étapes et de transitions, d'où nous avons trois méta-classes dans ce méta-modèle. Supposons que ces méta-classes se trouvent dans un méta-package nommé MMProcessus.

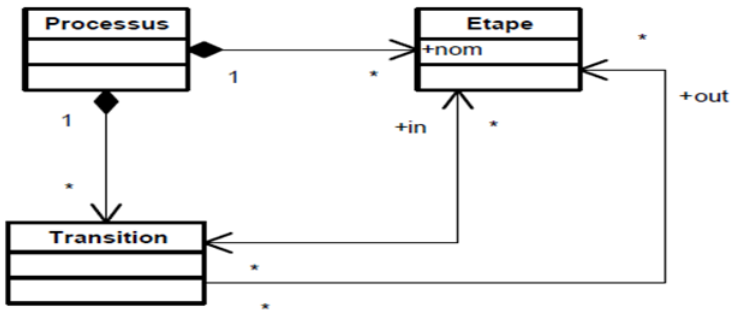

FIGURE 4.11 – *Méta-modèle de processus (Xavier, 2005)*

Par l'application des règles citées précédemment, on obtient la DTD du Listing 4.1.

```
1 <!ELEMENT Processus (ProcessusToTransition , ProcessusToEtape)
    >
2 <!ATTLIST Processus id ID> /* règle 1 et règle 4*/
3 <!ELEMENT Etape (nom) >
4 <!ATTLIST Etape id ID> /* règle 1 */
5 <!ELEMENT Transition (in, out) >
6 <!ATTLIST Transition id ID> /* règle 1 et règle 4*/
7 <!ELEMENT ProcessusToTransition #PCDATA> /* règle 4*/
8 <!ELEMENT ProcessusToEtape #PCDATA> /* règle 4*/
9 <!ELEMENT nom #PCDATA> /* règle 2 */
```

```
<!ELEMENT in #PCDATA> /* règle 4 */
<!ELEMENT out #PCDATA> /* règle 4 */
```

Listing 4.1 – *La DTD générée à partir du méta-modèle de processus*

La règle 1 permet de générer les balises *"Processus"*, *"Etape"* et *"Transition"*. La règle 2 donne la définition de la sous-balise *"nom"* qui doit être placée dans la balise *"Etape"*. La règle 4 permet de définir les balises *"in"*, *"out"*, *"Processus-ToTransition"* et *"ProcessusToEtape"*.
En se basant sur cette DTD, nous pouvons représenter des modèles de processus qui sont conforme à leur méta-modèle sous forme d'un document *XML*. Listing 4.2 représente un exemple de processus sérialisé en document *XML*. Ce processus est composé de deux étapes nommées *"début"* et *"fin"* reliées par une transition [Xavier 2005].

```
<Processus id="p1">
    <ProcessusToEtape>
     <Etape id="e1">
      <nom>début</nom>
     </Etape>
     <Etape id="e2">
      <nom>fin</nom>
     </Etape>
    </ProcessusToEtape>
    <ProcessusToTransition>
     <Transition>
      <in idref="e1"/>
      <out idref="e2"/>
     </Transition>
    </ProcessusToTransition>
</Processus>
```

Listing 4.2 – *Exemple d'un modèle de processus sérialisé en document XML*

4.8.3 DI (Diagram Interchange)

Nous avons vu qu'avec le standard *XMI* on peut représenter les modèles dans des documents *XML*. Réellement, il ne représente que les informations dont la structure est définie dans un méta-modèle, et ce conformément aux règles *d'XMI* [Xavier 2005].
Dans UML, la représentation graphique des modèles est ignorée lors de l'élaboration d'un document *XML*, car cette partie est absente dans le méta-modèle *UML*. Par exemple, il est impossible de redessiner un diagramme d'états ou de séquence. De plus, on ne peut pas connaitre le nombre de diagrammes à dessiner [Xavier 2005].
OMG a proposé un standard intitulé *DI (Diagram Interchange)* dont le but est de pallier le problème de représentation graphique des modèles UML.

Principe de fonctionnement de DI

L'idée générale de *DI* est de représenter en format *XML* les parties graphiques des modèles *UML*, et ce par, une définition d'un nouveau méta-modèle contenant des méta-classes représentant les éléments graphiques nécessaires à la reconstruction de toutes les parties graphiques d'UML. La Figure 4.12 montre ce nou-

veau méta-modèle (méta-modèle graphique), et qui doit être lié au méta-modèle *UML*. Par l'application de *XMI* a ce méta-modèle on obtient une structuration des balises *XML* (*DTD* graphique) qui nous permet de représenter les différentes parties graphiques en document *XML* [Xavier 2005].
De plus, *DI* définit une transformation de documents *XML* vers des documents *SVG*. Cette transformation est importante pour visualiser les modèles *UML* et leurs parties graphiques dans des outils supportant le format *SVG*.

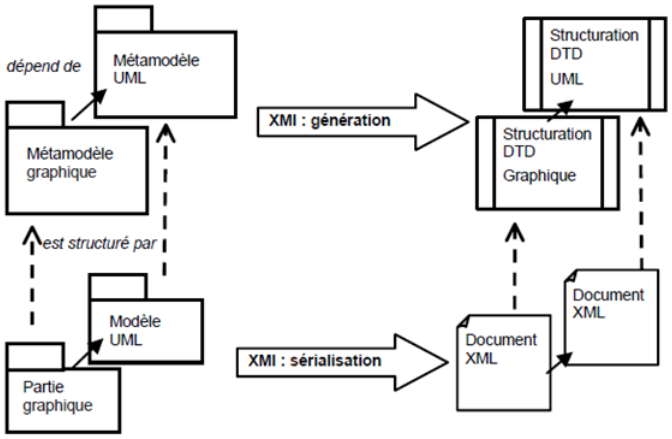

FIGURE 4.12 – *Principe de fonctionnement de DI (Xavier, 2005)*

4.9 Les Langages de transformation des modèles dans l'IDM

Dans cette section nous abordons deux langages de transformation des modèles notamement le langage QVT (Query/View/Transformation) et le langage ATL (Atlas transformation language).

4.9.1 Le langage QVT (Query/View/Transformation)

La notion la plus substantielle dans l'ingénierie dirigée par les modèles est la transformation des modèles. OMG a défini un standard qui permet d'assurer une telle transformation. Ce standard est le langage QVT. Il est composé de deux parties, à savoir la partie déclarative et la partie impérative. La partie déclarative constitue le Framework de la sémantique d'exécution de la partie impérative. Elle est divisée en une architecture à deux niveaux selon la figure 4.13 [OMG 2016].

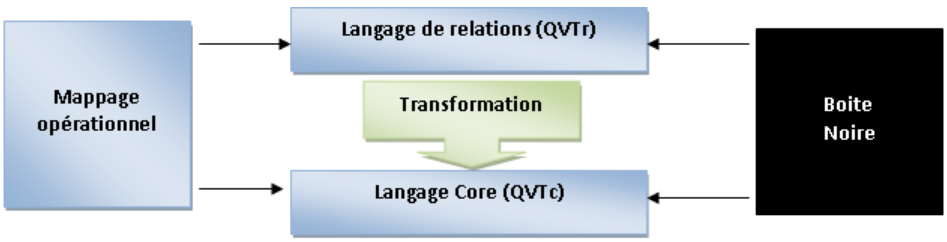

FIGURE 4.13 – *Relation entre les méta-modèles QVT (OMG, 2016)*

Les parties déclaratives

Les parties déclaratives sont découpées en une architecture à deux niveaux. Le premier niveau représente un langage de relations *(QVTr)* et son méta-modèle. Il permet de faire la correspondance de modèles d'objets complexes et la création de gabarit d'objet. Alors que le deuxième niveau représente le langage Core *(QVTc)* et son méta-modèle. Ces derniers utilisent des extensions minimales de *EMOF* et *OCL* [OMG 2016].

a) QVTr - Langage des relations : Le *QVTr* est un langage déclaratif de relations entre les modèles MOF. Il permet de faire une correspondance de motifs d'objet complexe. Lors de l'exécution d'une transformation le *QVTr*, il crée implicitement des classes de trace et leurs instances [OMG 2016].

b) QVTc - Langage Core : C'est un langage léger qui permet la correspondance d'un modèle avec un ensemble de variables en évaluant les conditions sur ces variables par rapport à un ensemble de modèles. Il traite tous les éléments du modèle source, cible et de trace, de manière symétrique.
Bien que les descriptions de transformation décrites à l'aide de Core soient plus détaillées, le langage QVTc est puissant en raison de sa simplicité, sa simple sémantique et ses modèles de trace qui doivent être définis explicitement [OMG 2016].

Implémentations impératives

En plus du langage de relations et le langue Core, il existe deux mécanismes permettant d'invoquer des implémentations impératives de transformation de relations ou de Core, à savoir une langue standardisée de mappages opérationnels, et la boite noire non standardisée [OMG 2016].

a) Langage de Mappage opérationnel : Le langage de mappage opérationnel permet de fournir des implémentations impératives. Ces implémentations remplissent les mêmes modèles de trace qui sont établis par le langage de relations. Il fournit également un style procédural, et une syntaxe concrète familière aux programmeurs impératifs en utilisant des extensions *OCL*. Lorsqu'il est difficile de fournir une spécification purement déclarative, on utilise les opérations de mappage pour implémenter une ou plusieurs relations à partir d'une spécification de relations. Les opérations de mappage peuvent appeler d'autres opérations, et ce implique une relation, dans le but de créer implicitement une trace entre les éléments du modèle. Une transformation écrite à l'aide d'opérations de mappage est appelée une transformation opérationnelle [OMG 2016].

b) Implémentations de Boite noire : Les boîtes noires implémentent explicitement les opérations. Ces opérations conservent les traces entre les éléments du modèles pertinents à l'implémentation. Les opérations MOF peuvent être dérivées à partir des relations. Cela permet un "plug-in" de toute implémentation d'une opération MOF avec la même signature. Ces "plug-in" sont avantageux pour plusieurs raisons [OMG 2016] :

- Ils permettent de coder des algorithmes complexes en n'importe quel langage de programmation avec une liaison MOF.
- Lors de calculs des attributs du modèle, ils facilitent l'utilisation des bibliothèques spécifiques à un domaine, lorsqu'ils seront difficiles ou impossibles à être utilisés avec OCL.

- ils permettent d'opacifier certaines parties d'une transformation.

L'inconvenant de l'implémentation des "plug-in" est qu'ils ont un accès aux objets du modèle [OMG 2016].

4.9.2 Le langage de transformation ATLAS (ATL

ATL est un langage de transformation de modèle. Il a été proposé par le groupe de recherche AtlanMod (Inria, Ecole des Mines de Nantes et LINA) afin de répondre au langage QVT de l'OMG. Dans l'ingénierie dirigée par les modèles, ATL est spécifié en tant qu'un méta-modèle et une syntaxe textuelle concrète. Il est intégré dans la plateforme Eclipse sous forme d'un plug-in. Il offre aux développeurs un moyen pour spécifier la manière de produire les éléments d'un modèle cible.
ATL comme QVT, est un langage de programmation hybride entre deux styles de programmation, à savoir la programmation déclarative et la programmation impérative. À travers le style déclaratif, ATL nous permet d'exprimer les correspondances entre les éléments du modèle source et cible. Dans le cas où il est difficile ou impossible de les faire de manière déclarative, ATL offre des constructions impératives pour faciliter la spécification des correspondances.
La correspondance et l'initialisation des éléments du modèle source se font sous forme d'un programme ATL qui se compose d'un ensemble de règles. En outre les modules de transformation de modèle, ATL permet de définir deux autres unités, à savoir les requêtes et les bibliothèques [Dennis 2015].

Les modules de transformation ATL

Le module ATL correspond à une transformation de modèle vers modèle. Il permet aux développeurs ATL de spécifier la manière de produire un ensemble de modèles cibles à partir d'un ensemble de modèles sources. Les modèles sources et cibles d'un module ATL doivent être "typés" par leurs méta-modèles respectifs. De plus, un module ATL accepte un nombre fixe de modèles en entrée, et renvoie un nombre fixe de modèles en sortie [William 2014].
a) La structure d'un module ATL : Un module ATL est composé des éléments suivants :

- Une section d'en-tête définissant certains attributs relatifs au module de transformation.
- Une section d'importation facultative permettant d'importer certaines bibliothèques ATL existantes.
- Un ensemble de helpers pouvant être considérés comme des méthodes en Java.
- Un ensemble de règles qui définit la manière dont les modèles cibles sont générés à partir des modèles source. On distingue trois types de règles : les règles de correspondance, les règles d'appel et les règles des fainéants.

Les helpers et les règles n'appartiennent pas à des sections spécifiques dans une transformation ATL. Ils peuvent être déclarés dans n'importe quel ordre pour certaines conditions [William 2014].
b) Module d'exécution : Le moteur d'exécution ATL définit deux modes d'exécution différents pour les modules ATL, à savoir le mode d'exécution par défaut et le mode d'exécution de raffinement. Dans le mode d'exécution par défaut, la transformation vise à créer un modèle cible en copiant le modèle source avec

quelques modifications. La conception d'une telle transformation nécessite d'une part, de spécifier les règles qui génèreront les éléments de modèle modifiés, et d'autre part toutes les règles qui ne feront que copier les éléments du modèle source. Par contre, dans le mode d'exécution de raffinage, on spécifie uniquement les règles qui portent les modifications à effectuer entre les modèles sources et cibles [William 2014].
c) Module d'exécution sémantique : Les modes d'exécution sémantiques disponibles sont : le mode normal ou par défaut et le mode de raffinement. Quoique la conception des transformations ATL ne nécessite aucune connaissance sur la sémantique de son exécution, la compréhension d'exécution d'une transformation ATL par le moteur ATL peut s'avérer utile dans certains cas, notamment lors de débogage d'une transformation [William 2014].
Le mode d'exécution sémantique par défaut est organisé en trois phases successives :

- une phase d'initialisation du module,
- une phase d'appariement ou correspondance des éléments du modèle source,
- une phase d'initialisation des éléments du modèle cible.

Dans le mode d'exécution sémantique de raffinement, aucune copie ne se produit, et les modifications sont directement appliquées au modèle source.

Les requêtes ATL

Une requête ATL est considérée comme une transformation de modèle vers une valeur primitive, telle que la chaîne de caractères, les valeurs numériques et booléennes. Elle peut être vue comme une opération qui permet de retourner une valeur primitive à partir d'un ensemble de modèles sources. L'utilisation la plus simple des requêtes ATL est la génération d'une sortie textuelle à partir d'un ensemble de modèles sources [William 2014].

Les bibliothèques ATL

Une bibliothèque (Librairie) ATL est formée d'une section d'importation facultative et un ensemble de "helpers". Elle peut être appelée de n'importe quelle unité ATL. Seul le type de "helper d'attributs" est permis dans les bibliothèques ATL, car une bibliothèque n'est associée à aucune étape d'initialisation au moment de l'exécution. Contrairement aux autres unités ATL (les modules de transformations, les requêtes), où l'utilisation de deux types d'helper sont permises.
Dans une bibliothèque, il est possible de déclarer des "helpers" dans le contexte par défaut du module. Lors de l'utilisation d'un tel "helper", il est obligatoire de l'associer à un contexte bien déterminer [William 2014].

4.10 Conclusion

Nous avons vu dans ce chapitre la notion de l'ingénierie dirigée par les modèles et ses concepts de base, notamment le modèle, méta-modèle, le méta méta-modèle et la relation qui existe entre ces différents concepts, comme il était présenté dans l'architecture à quatre niveaux d'IDM. Nous avons vu aussi les langages de modélisation, tels que l'UML et son impact sur les modèles de tous les niveaux de l'IDM, et le MOF et Écore pour la représentation et la gestion

des méta-modèles et méta-méta-modèles. A travers ces définitions, nous avons vu l'utilité de l'IDM qui se résume dans la transformation entre les modèles. Ces transformations sont mises en œJuvre à l'aide des langages de transformation, tels que les standards QVT de l'OMG et ATL d'Inria.

Deuxième partie

Réingénierie des applications Web vers les données liées & ontologies

Etat de l'art

5

Sommaire

5.1 Introduction 67
5.2 Mappage des ressources Web vers RDF 67
5.3 Génération des graphes RDF à partir des données relationnelles 70
5.4 Génération d'ontologie à partir de differentes sources de données 74
5.5 Conclusion 76

5.1 Introduction

Dans ce chapitre nous allons citer quelques travaux connexes à notre domaine d'intérêt, à savoir la réingénierie des applications Web vers les données liées. Plusieurs travaux peuvent être trouvés dans la littérature ; nous les classons en trois catégories. Nous présentons dans la première catégorie les travaux qui permettent de générer des données liées RDF à partir des ressources Web, telles que les tables HTML. Dans la deuxième catégorie on cite les travaux qui construisent des graphes RDF à partir des données relationnelles. La dernière catégorie regroupe quelques travaux qui permettent de construire des ontologies à partir de déférentes sources de données, telles que les bases de données et les pages Web.

5.2 Mappage des ressources Web vers RDF

Lehmann et al. [Lehmann *et al.* 2014] ont présenté un projet baptisé *DBpedia*. Le projet *DBpeida* n'est qu'une transformation de données structurées qui se trouvent dans des pages *Wikipedia* en une base de connaissances multilingue. Cette caractéristique multilingue rend la base de connaissances *DBpedia* utile dans plusieurs domaines d'application, tels que l'intégration de données, la reconnaissance des entités nommées, la détection de sujets et le classement de documents. On note aussi que de nombreuses applications ont utilisé la base de connaissances *DBpedia* comme une banque d'essai.

DBpedia représente l'exemple concret des données liées sur le Web. Elle définit des liens RDF pointant vers des ressources de données externes. En plus, *DBpedia* fait correspondre plusieurs structures de représentation des informations *Wikipedia* avec son ontologie. Ces structures peuvent être *des Infoboxes*, *des informations de catégorisation*, *des images*, *des coordonnées géographiques*, *des liens vers des pages Web externes*, *des pages de désambiguisation*, etc. *DBpedia* est devenu une plateforme d'interconnexion dans le Web des données liées, et a participé au succès du *LOD (Linked Open Data)*. La gestion de la structure de *DBpedia* est assurée par la communauté de ses utilisateurs [Lehmann *et al.* 2014].

La Figure 5.1 montre le Framework d'extraction de DBpedia. Il permet de construire une base de connaissances riches à partir des informations structurées de Wikipedia. Il est structuré en quatre phases : [Lehmann *et al.* 2014]

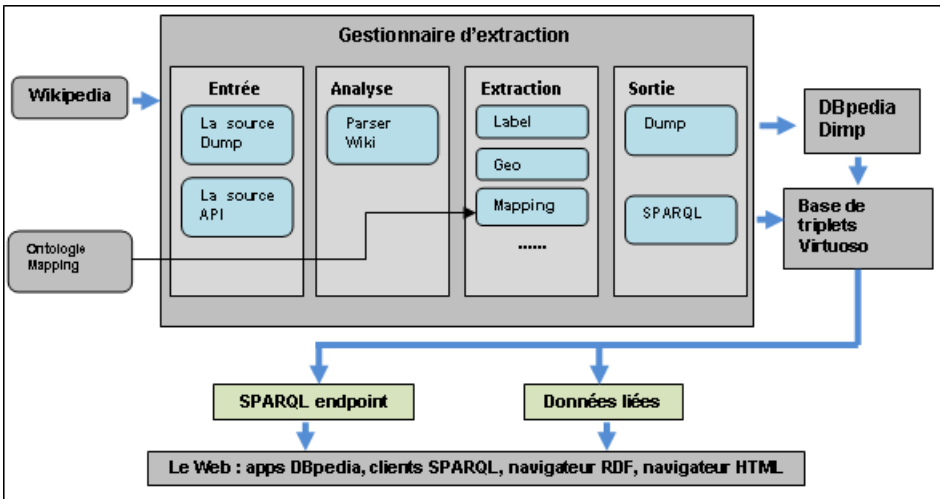

FIGURE 5.1 – *Framework d'extraction de DBpedia (Lehmann et al., 2014)*

La phase d'entrée : le Framework lit les pages *Wikipedia* à partir d'un *dump* de *Wikipedia* ou directement extraites à l'aide de l'API *MediaWiki*.

La phase d'analyse : l'analyseur de *Wiki* construit un arbre syntaxique abstrait en transformant le code source des pages lues précédemment.

La phase d'extraction : c'est la phase de construction des triplets RDF. *DBpedia* propose de divers extracteurs correspondant à chaque structure de données, telles que *les étiquettes*, *les résumés* ou *les coordonnées géographiques*. Chaque extracteur est utilisé pour produire un ensemble de triplets RDF.

La phase de sortie : c'est la phase de sérialisation des triplets RDF collectés à partir de la phase d'extraction. Plusieurs formats sont pises en charge, tels que *N-Triples* et *XML/RDF*.

Notez que *DBpedia* est dédiée uniquement aux pages *Wikipedia*, et ne traite pas d'autres applications Web [Lehmann *et al.* 2014].

Munoz et al. [Munoz *et al.* 2014] ont proposé des méthodes dans le but d'extraire des faits de toutes les tables de Wikipedia. Ces méthodes sont inspirées de celles des tables Web. Ils ont utilisé la base de connaissances des données liées DBpedia comme une référence lors du traitement des tables. Ils commencent à transformer les entités des tables en entités de base de connaissances DBpedia en utilisant des URI pour chaque sujet d'un article Wikipedia et ces relations. Par la suite, ils interrogent la base de connaissances sur des faits existants impliquant ces entités. Puis, ils construisent une image incomplète de la sémantique de la table à l'aide de l'image des entités qui ont des relations antérieures. Aussi, ils ont tenté d'extrapoler cela au reste de la table [Munoz *et al.* 2014].

Plusieurs points de vue montrent la signification et la portée de la proposition des auteurs. Premièrement, ils ont interprété la sémantique des tables Web en se basant sur l'utilisation d'une base de connaissances existante comme référence, telle que DBpedia. Deuxièmement, ils ont ajouté de nouvelles données à la base de connaissances. Ces données sont extraites à partir des tables Web. Finalement, ils ont servi les requêtes des utilisateurs par le bai des connaissances cumulatives des tables disponibles [Munoz *et al.* 2014].

En résumé, Munoz et al. [Munoz *et al.* 2014] ont utilisé dans leur approche une base de connaissances de données liées *DBpedia* pour trouver des relations entre les entités dans les tables de *Wikipedia*, suggérant les mêmes relations comme propriétés pour d'autres entités dans les mêmes colonnes sur différentes lignes. Cette approche à une précision de 40% seulement.

Dans un premier travail, Nagy et al. [Nagy *et al.* 2011] ont proposé une méthode pour transformer les tables HTML vers des tables relationnelles en se basant sur la technique « Header-Paths ». Dans un deuxième travail, Embley et al. [Embley *et al.* 2011] ont proposé de transformer les tables relationnelles en triplets RDF. Avec ces deux approches, les auteurs ont pu transformer les tables HTML en triplets RDF. Les règles de transformation pour avoir des triplets RDF dépendent du processus de factoring appliqué sur des tables HTML. Nous notons également que l'approche proposée dans [Nagy *et al.* 2011] ne suivent pas les règles standards du W3C présentées dans [Arenas *et al.* 2012]. En outre, la détection de la clé primaire dépend d'une étape préliminaire (représentation canonique de table basée sur Header-Paths), et ne considère pas directement les en-têtes des tables HTML.

Trias et al. [Trias *et al.* 2015] ont présenté une méthode de migration basée sur les principes de *ADM (Architecture-Driven Modernization)* pour transformer automatiquement les applications Web traditionnelles (classiques) en des applications Web basées sur la plateforme *CMS (Content Management Systems)* . Cette méthode de migration est basée sur *ADM*. Elle est définie comme un processus de réingénierie qui se compose de trois étape notamment l'étape de rétro-ingénierie, l'étape de structuration et l'étape d'ingénierie direct.
La première étape, consiste en trois phases :

1. La construction d'un modèle appelé *ASTM_PHP* à partir d'un source code PHP, et ce par l'utilisation d'un *DSL (Domain Specific Language)* propre au langage PHP. Le modèle obtenu représente la syntaxe et la sémantique du code PHP dans le niveau *PSM (Plateforme Spécific Model)*.
2. La construction d'un modèle *KDM (Knowledge Discovery Metamodel)* à partir du modèle précédent. Ce modèle représente la syntaxe et la sémantique du code PHP dans le niveau *PIM (Plateforme Independent Model)*. Notez que ce modèle est obtenu par une transformation *M2M (Model to Model)*.
3. La génération d'un modèle *CMS* à partir du modèle *KDM*, et la aussi par l'utilisation d'une transformation *M2M*.

La deuxième étape est la structuration des modèles obtenus (*les modèles CMS*), et cela se fait manuellement par les développeurs en tenant compte des spécifications de la plateforme *CMS* utilisée.
La dernière étape est l'ingénierie directe. Elle est composée en trois sous-phases :

1. La génération des modèles *KDM* à partir des modèles *CMS* structurés, ces modèles permettent de représenter l'implémentation de l'application Web (basée sur *CMS*) à un niveau *PIM*.
2. La génération des modèles *ASTM_PHP* à partir des modèles KDM. Ces modèles permettent de représenter l'implémentation de l'application Web à un niveau *PSM*.
3. La dernière phase est la génération du code. Elle consiste à générer des

artefacts de l'application à partir des modèles *ASTM_PHP*. Ces artefacts représentent l'application Web basée sur la plateforme *CMS*.

Notez que cette méthode est également supportée par un outil.

Le Tableau 5.1 résume les approches présentées ci-dessus comme suit : la colonne *"Approche"* donne l'approche en question, la colonne *"Entrée"* présente les entrées de l'approche, la colonne *"Elément clé"* donne les principaux éléments à analyser dans l'entrée, la colonne *"Sortie"* montre ce que l'approche génère en sortie, et enfin la colonne *"Outil"* indique si l'approche est ou non prise en charge par un outil.

TABLE 5.1 – *Approches pour mapper des ressources Web en données liées RDF*

Approche	Entrée	Elément clé	Sortie	Outil
Lehmann et al., 2014	Wikipedia (HTML)	Infoboxes	RDF	Oui
Munoz et al., 2014	Wikipedia (HTML)	Tables	Triplets RDF	/
Nagy et al., 2011	HTML Tables HTML	Tables	relationnelles	/
Embley et al., 2011	Tables relationnelles	Attributs et données	Triplets RDF	/
Trias et al., 2015	Application Web traditionnelle	code source PHP	Application Web à base CMS	Oui

5.3 Génération des graphes RDF à partir des données relationnelles

Konstantinou et al. [Konstantinou *et al.* 2014] ont proposé une approche modulaire pour générer un graphe RDF à partir des métadonnées stockées dans des bases de données relationnelles des systèmes de bibliothèques numériques, en utilisant un moteur de transformation de données relationnelles vers RDF. Ce moteur utilise un mappage R2RML défini manuellement (RDB vers RDF Mapping Language). *R2RML* est un langage de mappage pour la définition des correspondances entre une base de données relationnelle et un ou plusieurs graphes RDF. *R2RML* est une recommandation du *W3C*. L'adoption de tel langage garantit l'interopérabilité et la pertinente d'utilisation des mappages relationnels à RDF dans différentes implémentations de moteurs, et différents produits de système de base de données.

Le moteur de transformation *R2RML* est implémenté comme un projet Java modulaire qui se compose en deux parties, à savoir l'analyseur et le navigateur d'ontologies. L'analyseur utilise une base de données relationnelle à travers un document de description *R2RML*. Ce document contient un ensemble d'instructions décrivant comment les triplets seront générés en fonction des requêtes SQL posées sur la base de données relationnelle [Konstantinou *et al.* 2014].

La Figure 5.2 montre un aperçu de plus haut niveau de l'architecture du sys-

tème. Elle se compose en trois parties. La partie gauche représente un référentiel institutionnel sous forme d'un système en couche qui peut être une application de base de données relationnelle. L'application peut également avoir des fichiers binaires qui résident dans le système de fichiers. La seconde partie (au milieu) contient un document de définition de mappage qui permet à l'utilisateur de générer des triplets RDF. La troisième partie (à droite) montre comment les résultats de mappage seront stockés dans un magasin de triples, et comment seront exploités par une couche de présentation (*SPARQL ENDPOINT*) en utilisant le navigateur d'ontologies [Konstantinou *et al.* 2014].

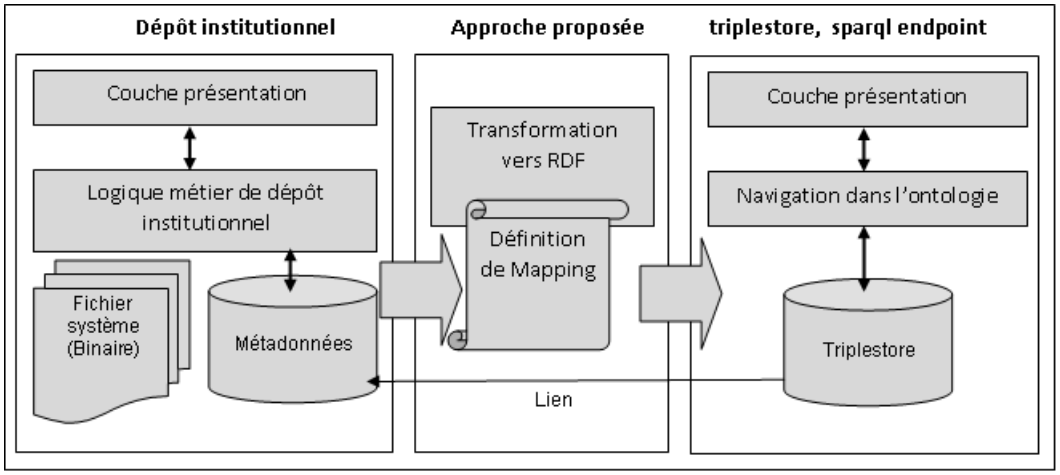

FIGURE 5.2 – *Aperçu des composants (Konstantinou et al., 2014)*

Cependant, cette approche ne permet pas de requêtes intelligentes, et en plus, elle a besoin d'une implémentation supplémentaire sur le système d'information.

Arenas et al. [Arenas *et al.* 2012] ont proposé une approche qui permet de mapper directement une base de données relationnelle vers des données liées RDF. L'approche proposée prend en entrée une base de données relationnelle complète, i.e., schémas et données, et produit comme sortie un graphe RDF appelé le graphe direct. Les algorithmes proposés composent un graphe des IRIs relatives qui doivent être résolues dans une base IRI pour former un graphe RDF. Les clés primaires et étrangères sont prises en compte dans le processus de transformation. Dans une base de données, les clés étrangères jouent un rôle très important dans le processus de transformation. Elles forment une référence à partir de n'importe quelle ligne d'une table vers une seule ligne dans une table. Le graphe direct utilise ces références, ainsi que chaque valeur de la ligne. Cela provoque un problème, tandis que les clés primaires ou étrangères sont explicites dans les bases de données relationnelles, dans le cas des tables HTML, ces clés sont implicites et cachées dans les colonnes de la table.

Scharffe et al. [Scharffe *et al.* 2012] ont présenté un projet sous le nom *Datalift*. Il s'agit d'un Framework et une plate-forme pour publier un ensemble de données de déférents formats, tels que *CSV* et *XML*, ou une base de données relationnelle sur le Web sous forme de données liées RDF. Les utilisateurs du système Datalift doivent être des experts, car ils ont besoin de connaissances im-

portantes sur le Web sémantique pour pouvoir manipuler cet outil.
L'objectif de Datalift est de fournir un chemin complet depuis les données brutes, jusqu'aux données entièrement interconnectées, identifiées et qualifiées, à savoir RDF. Les auteurs ont développé une plateforme qui tien en compte les quatre processus suivants : [Scharffe *et al.* 2012]

1. La sélection d'ontologie appropriée ou adéquate pour la publication des données.
2. La conversion des données en format RDF à l'aide de l'ontologie sélectionnée.
3. L'interconnexion des données avec d'autres sources de données.
4. La publication des données liées.

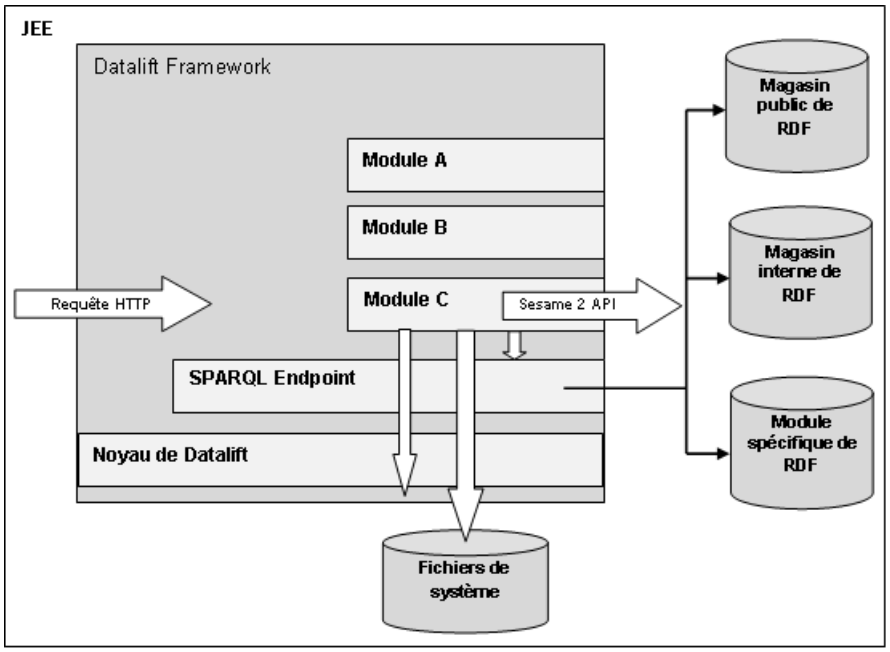

FIGURE 5.3 – *Architecture de Datalift (Scharffe et al., 2012)*

La Figure 5.3 montre l'architecture du Framework *Datalift*. Il s'agit d'un environnement d'exécution et de développement pour la mise en œuvre des de services Web REST basés sur RDF. Il se compose de : [Scharffe *et al.* 2012]

Le Framework Datalift : un ensemble d'API Java. Ces API sont utilisées dans le développement de diverses composantes du Datalift.

Le noyau Datalift : c'est une application Web *JEE* qui représente le Framework *Datalift*.

Les magasins RDF (triple stores) : pour l'enregistrement des données RDF en utilisant l'API *OpenRDF Sesame 2*.

Les modules : pour spécifier des fonctionnalités générales ou spécifiques d'un client.

Le Framework Datalift facilite le développement des services Web basés sur RDF, en automatisant l'accès aux magasins RDF, quelque soit le moteur RDF utilisé. Le *Datalift* prend en charge le déploiement dans un environnement de stockage distribué (réseau). Une fois que les données RDF sont produites, la plateforme

Dataliftet sauvegarde ces données dans des magasins RDF sous forme de ressources Web afin de les rendre accessibles directement. L'accès se fait à l'aide de *SPARQL Endpoint* qui comprend des fonctionnalités, telles que le routage des requêtes entre les magasins RDF disponibles, le contrôle d'accès, le filtrage de données et l'enrichissement des données [Scharffe *et al.* 2012].

Mulwad et al. [Mulwad *et al.* 2010] ont proposé un Framework *T2LD* pour représenter le contenu des tableaux en RDF. Le Framework commence par une attribution des étiquettes de classe à chaque colonne de tableau en interrogeant la base de connaissances *Wikitology*. Par la suite, un algorithme de liaison est utilisé pour faire une liaison entre la cellule du tableau et une entité, et cela par une nouvelle interrogation de la base de connaissances. Le résultat de la requête comprend les N premières entités possibles. L'algorithme construit un ensemble de vecteurs d'entité correspondant à chaque entité possible. Ces vecteurs sont classés et mis à jour à l'aide d'un algorithme *SVM-Rank*. Sur la base de ces nouveaux vecteurs, un autre algorithme de classification à base SVM décide de lier la cellule du tableau à cette entité de rang le plus élevé. Dans le cas échant (pas de liaison), il s'agit d'une nouvelle entité non présente dans la base de connaissances [Mulwad *et al.* 2010].

Les auteurs ont présenté une autre approche pour identifier les relations entre les colonnes de la table, en utilisant un algorithme de détection des relations entre deux colonnes. La relation ayant le score le plus élevé est choisie comme relation entre ces deux colonnes. Le résultat final du Framework est sérialisé à l'aide d'un modèle proposé par les auteurs en utilisant la notion *N3*. Ce modèle permet de représenter les tableaux sous forme *LOD*. L'approche a été évaluée sur 15 tables relationnelles. Elle n'a atteint que 25% pour l'identification des relations.
Dans un autre travail, les auteurs ont implémenté un module d'inférence pour conclure la sémantique des entêtes du tableau, les relations et les valeurs des cellules. Ce module d'inférence utilise les connaissances des données liées ouvertes (LOD). L'évaluation est effectuée sur des tables Web et de Wikipedia [Mulwad *et al.* 2013].
Le tableau 5.2 résume les approches présentées dans cette section. Il comporte cinq colonnes. La première représente l'approche en question. La deuxième colonne indique l'élément clé dans le processus de traitement de chaque approche. La troisième colonne présente la sortie de l'approche. La dernière colonne mentionne si l'approche est concrétisée par un outil ou non.

TABLE 5.2 – *Approches de transformation des données relationnelles en RDF*

Approche	Entrée	Elément clé	Sortie	Outil
Konstantinou et al., 2014	Métadonnée stockée sur BDD		Graphe RDF	/
Arenas et al., 2012	Base de données relationnelle	Schémas et données	Graphe RDF	/
Scharffe et al., 2012	CSV, XML, BDDR		RDF	Oui Datalift

Mulwad et al., 2010	Contenu des tables	Entités et relations	RDF	/
Mulwad et al., 2013	Contenu des tables+ sens		RDF	/

5.4 Génération d'ontologie à partir de differentes sources de données

Dans cette section, nous présentons quelques travaux relatifs à la création d'ontologies (ingénierie des ontologies).

Kaljurand [Kaljurand 2007] a proposé une approche qui permet de transformer un texte écrit en langage naturel contrôlé vers une ontologie OWL d'une part, et d'autre part, l'approche permet de verbaliser OWL en des fragments de texte. Les textes utilisés sont générés par *ACE (Attempto Controlled English)*. *ACE* permet de représenter des séquences de phrases déclaratives simples ou composites en anglais. Kaljurand [Kaljurand 2008] a développé un outil baptisé *ACEView* pour l'édition et la visualisation de l'ontologie OWL en utilisant le format d'échange *SWRL (Semantic Web Rule Language)* afin d'étendre les axiomes OWL. L'outil est utilisé comme un plugin dans Protégé, afin d'enrichir son interface basée sur *ACE CNL (Controlled Natural Languages)*. *ACEView* permet aux utilisateurs d'interagir avec l'ontologie à l'aide de questions *ACE (Attempto Controlled English)*. Cette interaction se fait en mappant le texte *ACE* en *OWL/SWRL* et vice-versa.

Kuhn [Kuhn 2009] a présenté un *Wiki* sémantique, appelé *AceWiki*, pour représenter les connaissances, et il a utilisé *ACE* pour impliquer les utilisateurs ordinaires pour étendre le contenu du *Wiki*. Kuhn [Kuhn 2013a] a présenté un travail étendu qui permet de transformer le contenu *d'AceWiki* en OWL en utilisant les avantages de la grammaire de notation de l'éditeur prédictif *ACE (codeco)*. Kuhn [Kuhn 2013b] a fourni une extension *d'AceWiki*, connue sous le nom *d'AcEWiki-GF*, pour prendre en charge le multilinguisme en ajoutant un cadre grammatical (GF). Par conséquent, il a donné aux utilisateurs tous les avantages *d'AceWiki*.

Dimitrova et al. [Dimitrova *et al.* 2008] ont développé un outil appelé ontologie *Rabbit to OWL (ROO)* pour l'édition d'ontologies. Ils visaient à couvrir tout le processus d'ingénierie d'ontologie. Cet outil nécessite un expert de domaine pour la création d'ontologies, en particulier dans la première étape du processus et dans la phase de conception.

Denaux et al. [Denaux *et al.* 2012] ont présenté un outil pour améliorer et augmenter la fonctionnalité de l'outil de création d'ontologies *(ROO)* par l'interaction avec les utilisateurs. Le Framework comprend et analyse les actions de l'utilisateur, et renvoie un retour sémantique pour contrôler les erreurs de modélisation lors de l'insertion d'un nouvel axiome dans l'ontologie. Les auteurs ont proposé dans [Denaux *et al.* 2013] un nouvel outil qui intègre une interface

de dialogue. L'outil est considéré comme une extension de *Rabbit*.

Ougouti et al. [Ougouti *et al.* 2015] ont présenté un outil appelé *MedPeer*, qui est un système de gestion de données *"hétérogènes et distribuées"* qui fonctionne dans un environnement *peer-to-peer*. La fonction principale de ce système est de générer une ontologie OWL à partir d'une base de données relationnelle.

Sequeda et al. [Sequeda *et al.* 2012] ont proposé une approche permettant de transformer une base de données relationnelle en un graphe RDF avec un vocabulaire sous forme d'une ontologie OWL. Le travail proposé est basé sur la norme W3C Direct Mapping. Les auteurs ont pris en compte deux propriétés fondamentales pour réaliser une cartographie directe. Ces propriétés sont la préservation des informations et la préservation des requêtes. En outre, ils ont étudié deux autres propriétés : la conservation de la monotonie et la conservation de la sémantique.

Astrova et al. [Astrova & Stantic 2005] ont proposé une approche pour créer une ontologie et ses instances. Cette approche comprend trois étapes : la conceptualisation, la transformation de schéma et la migration des données. La première étape est le résultat d'un processus de rétro-ingénierie qui consiste à créer un modèle représentant les données d'un formulaire. Le modèle est élaboré à partir de l'analyse des formulaires HTML. La deuxième étape est un processus d'ingénierie avancée, qui consiste à créer une ontologie à partir du résultat de la première étape. Enfin, des instances sont créées à partir de la base de données relationnelle lors de l'étape de migration des données.

Benslimane et al. [Benslimane *et al.* 2008] ont proposé une approche pour créer une ontologie et ses instances à partir des sites Web à forte utilisation de données. Cette approche comprend trois phases : conceptualisation, formalisation et migration. La conceptualisation est le résultat d'un processus de rétro-ingénierie qui consiste à extraire les sous-schémas relationnels des formulaires HTML et leurs dépendances à partir du schéma XML. La phase de formalisation est un processus d'ingénierie avancée, qui consiste à créer un schéma conceptuel basé sur le modèle UML à partir du résultat de la phase précédente. Ensuite, ces modèles UML sont traduits en ontologie OWL. Enfin, les instances sont créées à partir des tuples relationnels pendant la phase de migration.

Le Tableau 5.3 résume les travaux précédemment cités. Il se compose de quatre colonnes : la première contient l'approche décrite. Dans la deuxième colonne, nous donnons les connaissances en entrée qui seront transformées en ontologie OWL. La troisième colonne donne la sortie du système. Enfin, la dernière colonne affiche le nom de l'outil qui a été implémenté pour supporter le travail proposé.

TABLE 5.3 – *Résumé des travaux d'ingénierie des ontologies à partir de différentes sources de données*

Approche	**Entrée**	**Sortie**	**Outil**
Kaljurand 2007	Langage naturel contrôlé (Anglais)	OWL	Attempt ACE
Kaljurand 2008	Texte ACE- OWL/SWRL	OWL/SWRL- texte ACE	ACEView
Kuhn 2009, Kuhn 2013a	Contenu Wiki (Anglais)	OWL	AceWiki
Kuhn 2013b	Contenu Wiki multilingue	OWL	AceWiki-GF
Dimitrova et al., 2008	Expert de domaine ou utilisateur	OWL	Rabbit to OWL ontology (ROO)
Denaux et al., 2012, Denaux et al., 2013	Expert de domaine ou utilisateur et feedback.	OWL	Framework + ROO
Ougouti et al., 2015	Base de données relationnelle	OWL	MedPeer
Sequeda et al., 2012	Base de données relationnelle	OWL + RDF	/
Astrova et al., 2005	Formulaires HTML et base de données relationnelle	Ontologie	/
Benslimane et al., 2008	Formulaires HTML et base de données relationnelle + tuplets	Ontologie OWL	Prototype

5.5 CONCLUSION

La plupart des travaux présentées ci-dessus génèrent soit des données liées à partir des données relationnelles ou à partir des données Web (tables HTML), soit ils génèrent une ontologie à partir des connaissances d'un expert du domaine, d'un langage contrôlé, de bases de données relationnelles ou de formulaires HTML.
Les travaux, basés sur les bases de données relationnelles, ne prennent en compte que la partie du langage de définition de données pour créer les instances d'ontologie et les formulaires HTML pour créer les concepts d'ontologie.
Dans les deux chapitres suivants, on présente deux approches. La première approche permet de générer des données liées à partir des données HTML, telles que les tables, les listes et même les liens hypertexte. Cette approche est basée sur *l'IDM*. La seconde approche permet de construire une ontologie à partir d'une application Web en se basant sur leur le code source.

Approche I

6

Sommaire

6.1 Introduction 78
6.2 Exemple d'exécution 78
6.3 Approche proposée 79
6.3.1 Étape de pretraitement 80
6.3.2 Étape de réingénierie basée sur l'IDM 81
6.3.3 Étape de raffinement 88
6.4 Evaluation 90
6.4.1 L'outil implémenté 90
6.4.2 Expérimentations et discussion des résultats 91
6.5 Conclusion 92

6.1 Introduction

Dans ce chapitre nous allons présenter les détails de la première approche proposée pour résoudre le problème de la réingénierie des applications Web vers les données liées [Bouougada *et al.* 2018]. A travers ce que nous avons vu dans la première partie et exactement dans le chapitre des applications Web, nous avons défini une application Web comme un ensemble de fichiers classés en deux catégories : des documents HTML et des scripts PHP. Les fichiers PHP ne sont pas pris en compte dans ce premier système, puisqu'ils ne contiennent pas explicitement des données, contrairement aux documents HTML. D'où l'idée vient, de transformer les données existantes dans les pages HTML en données liées. Donc, l'approche proposée prend les documents HTML, et elle les transforme en fichiers RDF en se basant sur l'ingénierie dirigée par les modèles. L'approche proposée est évaluée sur des pages HTML de Wikipédia, précisément les Infoboxes. Wikipédia est une encyclopédie gratuite et libre. Une infobox est une table située dans le coin supérieur droit d'une page Wikipédia ; elle résume les informations utiles sur un sujet.

6.2 Exemple d'exécution

Dans cette section, nous présentons un exemple montrant comment transformer une application Web en une application Web sémantique. En particulier, mapper des documents HTML vers des triplets RDF. Comme le montre la Figure 6.1, l'objectif est de générer pour chaque document HTML un fichier RDF, dont les triplets sont construits à partir des éléments de données HTML, tels que *les listes*, *les tableaux*, etc.

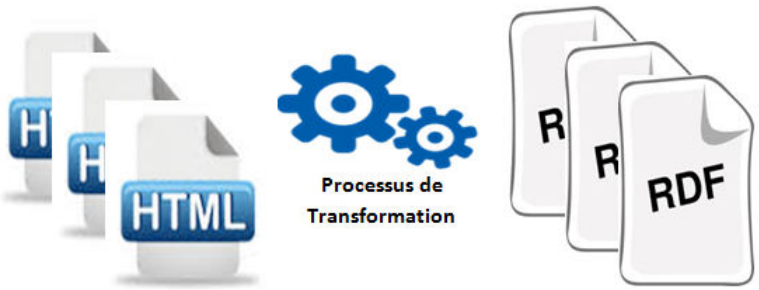

Figure 6.1 – *Transformation des documents HTML en fichiers RDF*

La Figure 6.2 montre la transformation d'une liste HTML en RDF. Le côté *(a)* de la figure représente un aperçu du navigateur internet contenant une liste de données. Le côté *(b)* montre le triplet RDF correspondant à cette liste (*côté (a)*) que nous visons à obtenir automatiquement par un processus de transformation.

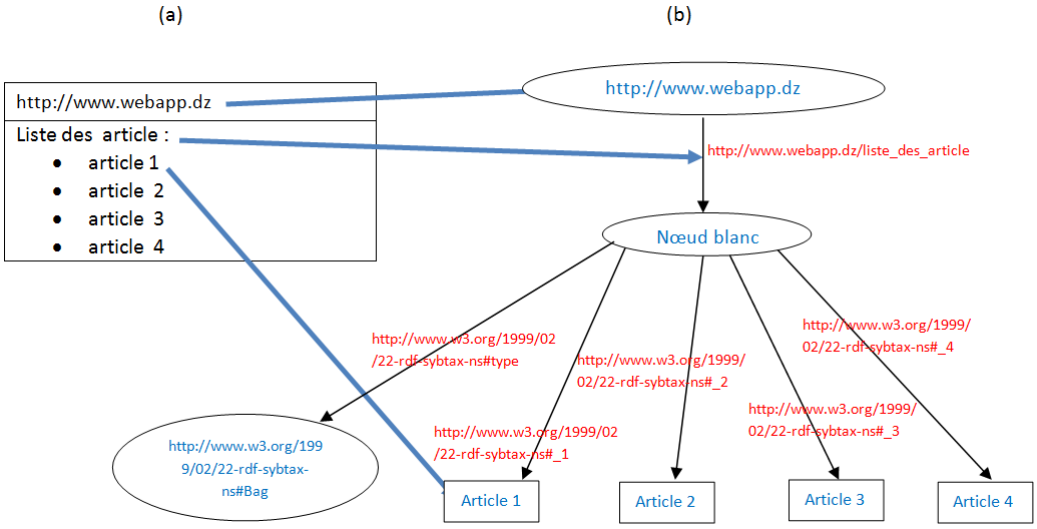

FIGURE 6.2 – *Exemple de transformation*

Notons que dans cet exemple, les éléments HTML, notamment les listes, représentent des modèles portant des données. Ces données seront représentées en format RDF. Il est clair que chaque fois qu'une transformation de données HTML en RDF est souhaitée, les mêmes règles de transformation sont appliquées, quelles que soient les données à transformer, que ce soit des tables, des listes ou des liens.
Donc, nous avons comme modèle d'entrée (listes, tableaux et liens) qui subira à un processus de transformation pour obtenir un modèle de sortie (RDF). Cette idée est similaire au paradigme IDM (Ingénierie Dirigé par les Modèles). Il repose sur la définition des méta-modèles source et cible, et sur un ensemble de règles de transformation. Dans notre cas, il faut généraliser nos modèles d'entrée et de sortie pour construire deux méta-modèles : un méta-modèle source pour représenter les modèles d'entrée HTML, et un méta-modèle cible pour les modèles de sortie RDF.
Dans la suite du document, ces méta-modèles et ces règles de mappage (transformation) sont détaillés, et reposent tous sur des paramètres d'ingénierie pilotée ou dirigée par les modèles (IDM).

6.3 Approche proposée

Nous avons proposé un processus composé de trois étapes, comme illustré dans la Figure 6.3. La première étape est dite prétraitement ou extraction de données. Cette étape a pour objectif de préparer les données candidates à l'étape suivante. Ces données sont extraites et représentées dans un format approprié. La deuxième étape constitue le noyau du système qui est le processus de transformation basé sur les modèles. La dernière étape est le raffinement du résultat obtenu par l'étape précédente. Dans la suite, nous allons détailler le rôle et le contenu de chaque étape.

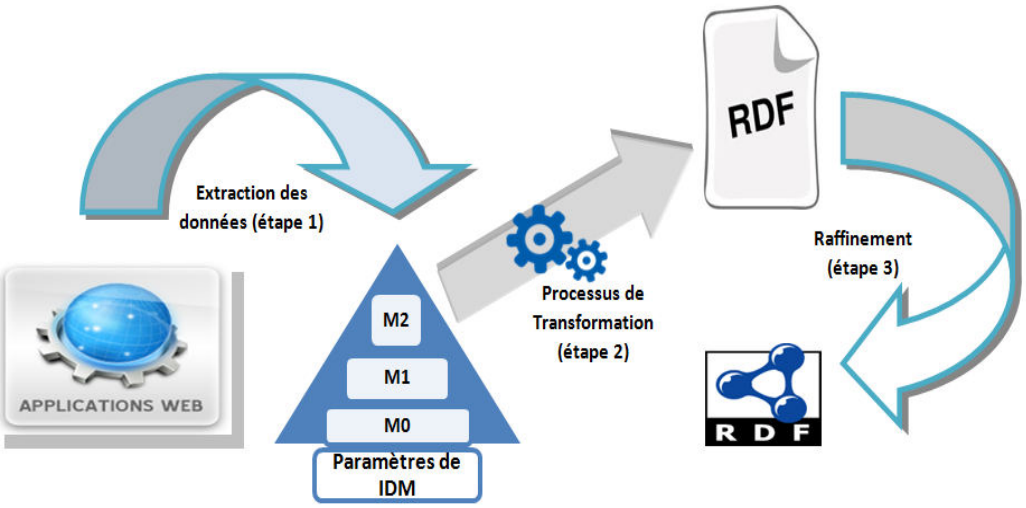

FIGURE 6.3 – *Architecture du système*

6.3.1 Étape de pretraitement

Dans cette étape, notre système prend en compte les pages HTML. Cette étape est composée de deux sous-étapes, à savoir l'extraction et l'adaptation. Le prétraitement des données vise à préparer les données pour alimenter l'étape suivante du processus.
Nous pouvons résumer cette tâche en trois mots : extraire et adapter pour construire!!! Alors on extrait quoi? Nous adaptons pour construire quoi? La réponse à ces questions est plus facile à ce que vous pensez.

- L'extraction signifie que nous ne prenons que des éléments (balises) HTML contenant des données, comme <*table*> .... </*table*>, <*ol*> ... </*ol*>, <*ul*> ... </*ul*> et <*dd* > ... </*dd*>, car ces éléments contiennent des données provenant souvent de bases de données cachées. Les autres éléments comme <*img*>, <*frame*>, <*span*> etc. sont ignorés et supprimés des pages HTML.

- Adapter pour construire signifie que le résultat de l'étape d'extraction est ajusté pour être conforme au méta-modèle source qui sera défini à l'étape suivante. Par exemple, l'élément <*ol*> .... </*ol*> est transformé en <*list*> ... </*list*>.
  Nous prenons maintenant l'exemple présenté dans la section précédente (section 2), et nous expliquons comment l'étape d'extraction et d'adaptation des données est effectuée. La Figure 6.4 affiche l'étape de prétraitement.

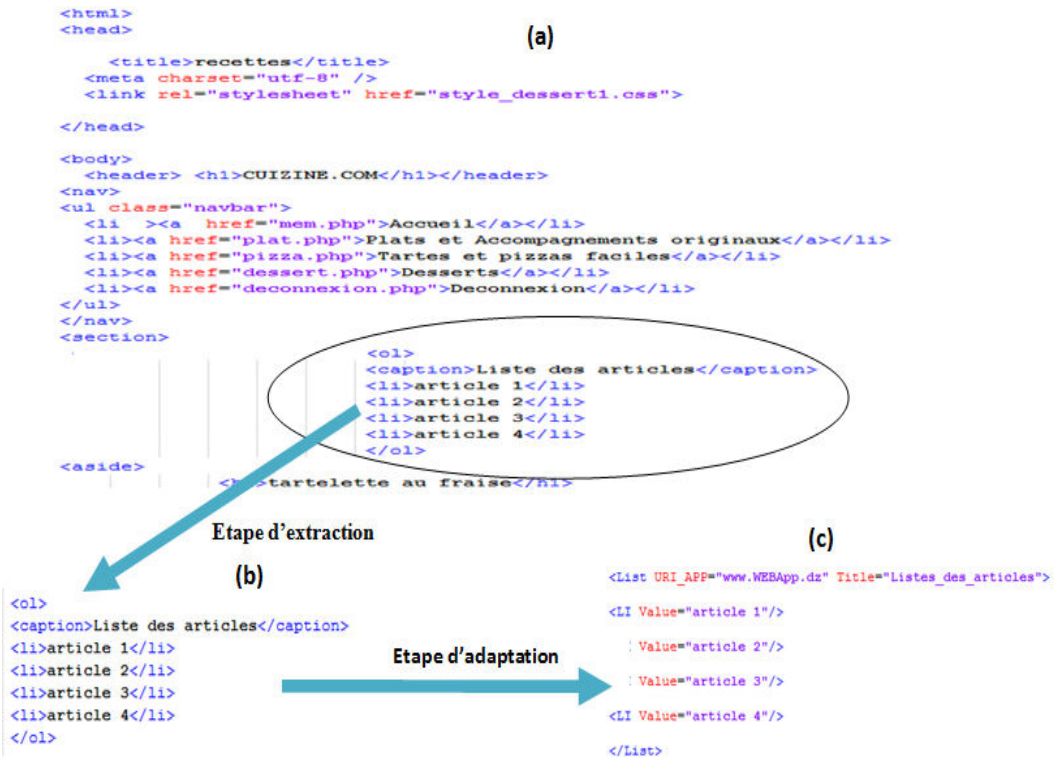

FIGURE 6.4 – *Prétraitement : Exemple d'extraction et d'adaptation des données*

Comme le montre la Figure 6.4 la partie supérieure *(a)* affiche le code HTML de l'exemple de déroulement, sur lequel nous appliquons le processus de prétraitement, à savoir l'extraction et l'adaptation des données. Le résultat est montré dans la partie *(b)* et *(c)* de la Figure 6.4. Le langage *(tags)* utilisé pour l'adaptation est défini par le méta-modèle source qui sera détaillé dans la section suivante. Le résultat de l'étape de prétraitement sera automatiquement transmis à l'étape suivante.

6.3.2 Étape de réingénierie basée sur l'IDM

Cette deuxième étape constitue le noyau du système. Elle est basée sur l'ingénierie dirigée par les modèles (IDM), et se compose de trois couches définies selon les exigences de l'IDM (Figure 6.5). La couche *Mo* contient le fichier d'entrée reçu de l'étape précédente (prétraitement) et le fichier de sortie (fichier RDF) que nous voulons créer par ce système. La couche *M1* comprend trois fichiers, notamment deux méta-modèles (MM), l'un pour définir le MM source et l'autre pour le MM cible, et le troisième fichier représente le moteur de transformation. Enfin, la dernière couche *(M2)* est constituée du méta-méta-modèle de tous les fichiers définis dans la couche précédente *(M1)*. Nous allons discuter et décrire en détail ces couches dans les sous-sections suivantes.

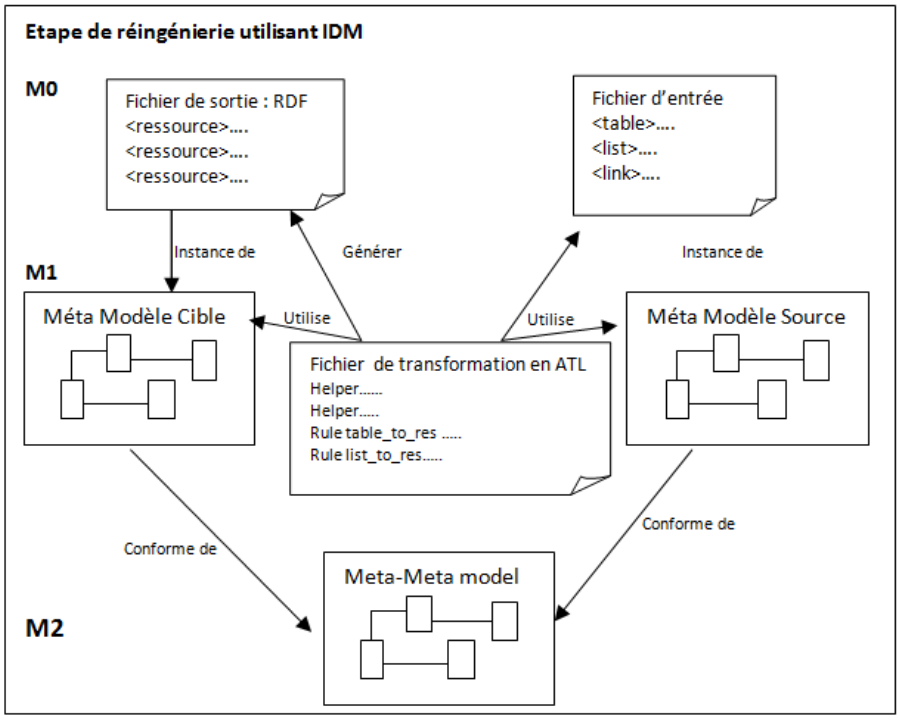

FIGURE 6.5 – *Processus de réingénierie basé sur les modèles*

La couche Mo

Dans la couche *Mo* correspondant au monde réel, il y a deux fichiers : le fichier d'entrée et le fichier de sortie (fichier résultant) :

- Le premier fichier (fichier d'entrée) représente le résultat obtenu de l'étape de prétraitement. Il contient des données que nous souhaitons transformer en données sémantiques. Ce fichier est un ensemble de balises définies dans le méta-modèle source ; il constitue donc son instance. Ces balises transportent les données à transformer en triplets RDF.
- Le second fichier (RDF) est celui qui sera obtenu en appliquant les règles de transformation sur le fichier d'entrée. Les deux contiennent les mêmes données, mais dans des représentations différentes. Avec le format RDF, les données deviennent accessibles et compréhensibles, et donc exploitables par machine.

La couche M1

C'est la couche des méta-modèles. Dans notre système, cette couche contient deux méta-modèles : source et cible, et le fichier qui contient les règles de transformation.
Les méta-modèles sont écrits en langage *Ecore* [Jouault 2008], qui est un langage de méta-modélisation basé sur des notations assez similaires à celles du diagramme de classes UML. En fait, Ecore est une implémentation d'Eclipse du standard *MOF (Meta Object Facility)*. Notez que Ecore est défini par lui-même.
Ecore est le noyau de *l'EMF (Eclipse Modeling Framework)* [Steinberg *et al.* 2003] qui permet aux utilisateurs de créer facilement des langages de conception.

Le fichier de règles de transformation est écrit en langage *ATL (Atlas Transformation Language)* [Jouault 2008]. *ATL* est un langage de transformation des modèles basé sur une syntaxe textuelle concrète. *ATL* fournit les moyens pour correspondre chaque élément du méta-modèle source avec un élément du méta-modèle cible.

A) **Le méta-modèle source (MMS)**
La Figure 6.6 montre notre méta-modèle source, à travers lequel nous allons défini un ensemble de balises (Tableau 6.1) permettant d'écrire le fichier d'entrée. Ce méta-modèle représente la conception d'éléments HTML contenant des données. Nous avons conceptualisé tous les éléments HTML nécessaires, en particulier les listes, les tableaux et les liens, car ils contiennent souvent plus de données que les autres éléments. Comme le montre la Figure 6.6, pour une table HTML, on peut voir que l'élément <*Table*> contient un ensemble d'éléments <*TR*>, dont chacun contenant un ensemble d'éléments <*TD*>. Notez que MMS est écrit en langage *Ecore*.

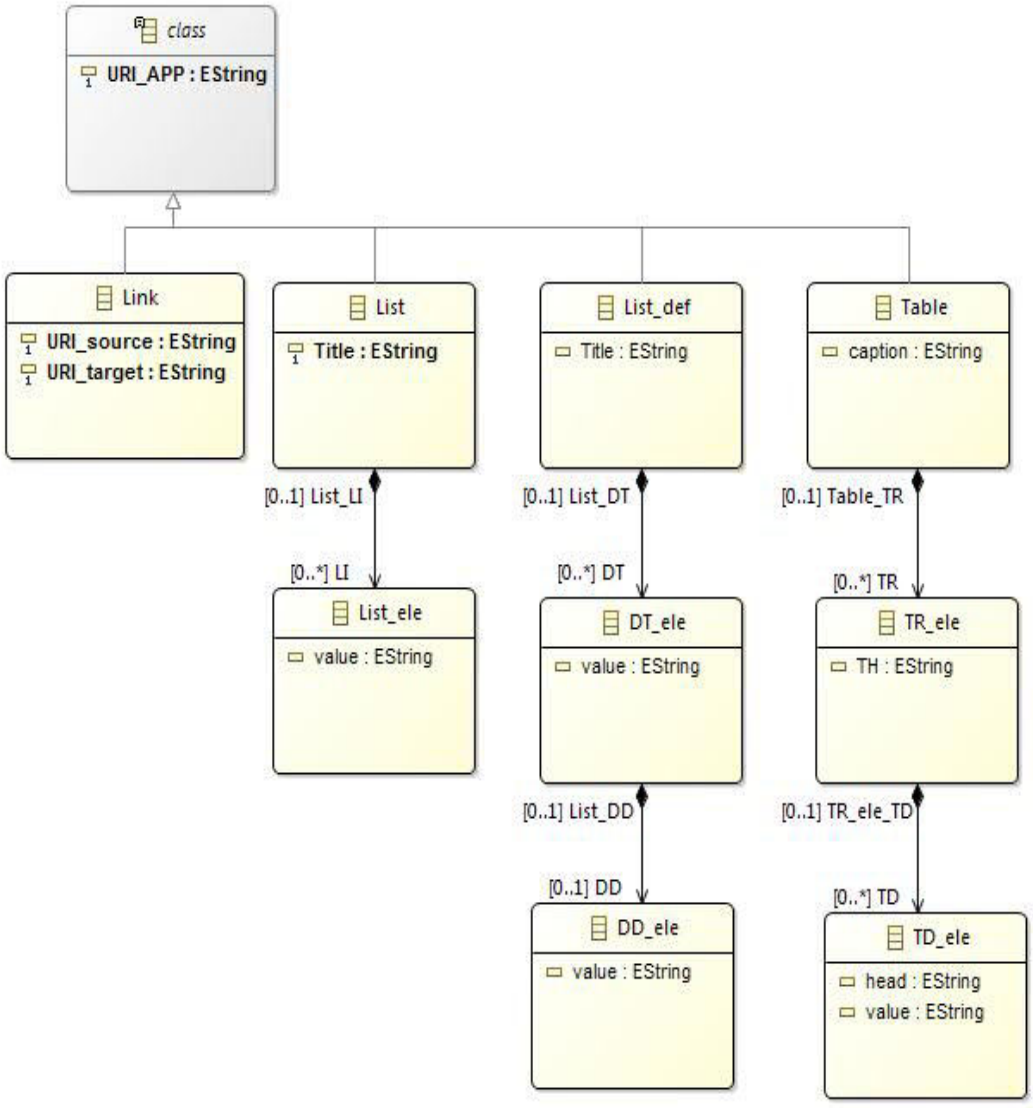

FIGURE 6.6 – *Le méta-modèle source (MMS)*

TABLE 6.1 – *Les balises défini par MMS*

Elément		Description	Exemple
Balise	**Attribut**		
List	Title	Permet de représenter une liste numérotée	`<list title= 'liste des articles'> <Li value='article 1'> <Li value='article 2'> </list>`
LI	Value	Pour citer les éléments d'une liste ordonnée	
Table	Caption	Pour construire un tableau	`<table caption='table des produits'> <TR TH='nom'> <TD value='stylo'/> </TR> <TR TH='couleur'> <TD value='rouge'/> </TR> </table>`
TR	TH	Pour former une colonne	
TD	Head, value	Pour former une cellule	
List_def	Title	Pour construire une liste de définition	`<list_def title='liste des articles'> <DT value='article 1'> <DD value='c est un article de qualité'/> </DT> <DT value='article 2'> <DD value='c'est un article de qualité moyenne'/> </DT> </list_def>`
DT	Value	Pour citer les éléments d'une liste de définition	
DD	Value	Pour donner la définition de l'élément en cours	
Link	URI_source, URI_target	Pour former un lien	`<link URI_source= 'http://www.monsite.dz' URI_target= 'http://www.google.fr'/>`

En référence à l'exemple de déroulement, la Figure 6.7 montre comment écrire une liste de données en utilisant les balises définies par le méta-modèle source, en écrivant précisément le fichier d'entrée dans l'étape de prétraitement (adaptation).

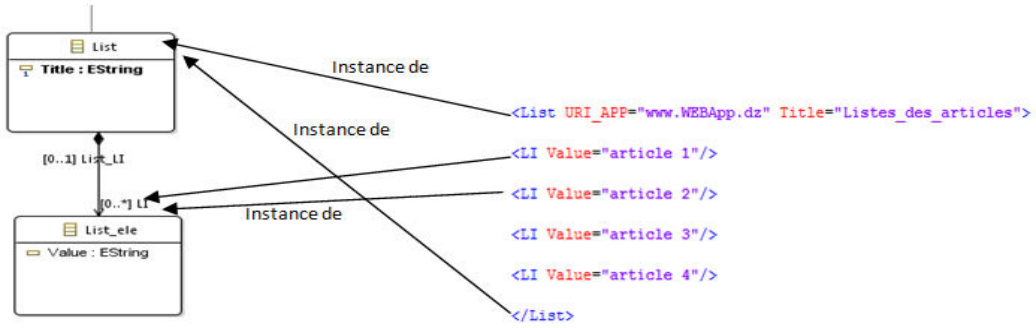

FIGURE 6.7 – *Liste des données écrites par les balises du MMS*

B) **Le méta-modèle cible (MMC)**
Dans ce méta-modèle, nous avons défini uniquement les éléments RDF dont nous aurons besoin. Comme le montre la Figure 6.8, nous voyons que MMC fournit un ensemble d'éléments RDF. Notons que ce MMC est extensible pour couvrir d'autres éléments RDF. L'utilisation de ce méta-modèle en sortie nous permet de créer un fichier RDF à partir des données d'entrée. Le MMC est écrit en langage *Ecore*.

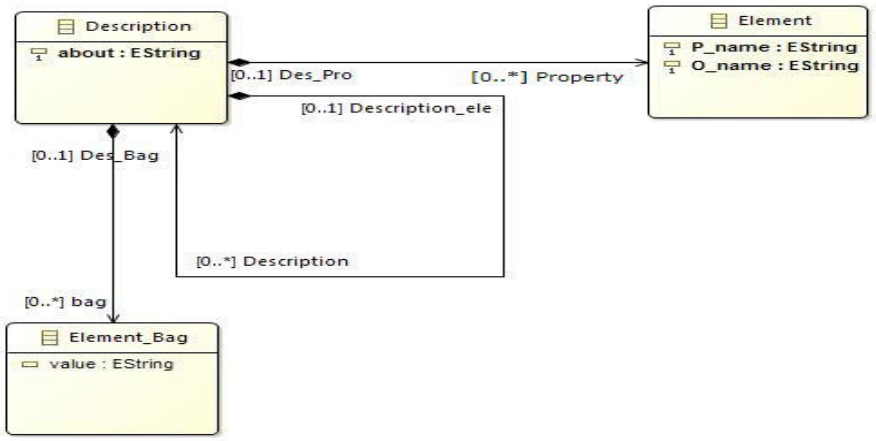

FIGURE 6.8 – *Le méta-modèle cible MMC*

C) **Les règles de transformation**
Les règles de transformation sont exprimées en langage ATL [Jouault 2008]. Donc, ils vont former un fichier ATL. Ces règles permettent de faire correspondre les éléments du MMS aux éléments du MMC. Puisque nous souhaitons transformer uniquement les éléments HTML (tables, liens, listes ordonnées et listes de définition) contenant des données, nous devons définir quatre règles correspondant aux éléments HTML.
Le processus de transformation prend en entrée trois fichiers MMS, MMC et le fichier d'entrée résultant de la phase de prétraitement (et conforme à SMM) pour générer le fichier RDF cible.

Règle 1 : Le listing 6.1 représente le code source en langage ATL de la règle qui permet de transformer une table en une ressource RDF imbriquée. L'élément <*Table*> devient une ressource imbriquée dont le sujet est la valeur de l'attribut « *Table.URI_APP* », et le prédicat est formé par la concaténation de l'attribut « *Table.URI_APP* » avec l'attribut « *Table.Caption* ». Les objets sont des ressources, construites comme suit :

- Les sujets sont la concaténation de « *Table.URI_APP* » avec « *Table.Caption* » et l'attribut « *TH* ».
- Les prédicats sont la concaténation de « *Table.URI_APP* » avec « *TD.head* ».
- Les objets sont des valeurs de l'attribut « *TD.Value* ».

```
rule Table2Description {
 from  s: MMS!Table
 to    t: MMC!Description (
            about <- s.URI_APP,
            Property<-Set{p},
            Description<-Set{q}),
       p: MMC!Element(
            P_name<-s.predicat_Table,
            O_name<-'nested ressource'),
       q: distinct MMC!Description foreach(i in (s.
    getTH()))(
          about<-s.predicat_Table+'/'+i,
          Property<-f),
       f: distinct MMC!Element foreach (n in s.
    getTDValues()) (
          P_name<-s.getTDtete().toString(),
          O_name<-n.toString())
                    }
```

Listing 6.1 – *La règle de transformation de l'élément Table en une ressource RDF imbriquée*

Règle 2 : Le listing 6.2 représente le code source en langage ATL de la règle qui permet de transformer une liste ordonnée en une ressource RDF. L'élément <*List*> devient une ressource qui a :

- Sujet est « *List.URI_APP* ».
- Prédicat est la concaténation de « *List.URI_AP* » avec « *List.Title* ».
- Object est un sac RDF de la liste de valeurs obtenue à partir de « *LI.Value* ».

```
rule List2Description {
 from   s: MMS!List
 to   t: MMC!Description (
          about <- s.URI_APP,
          Property<-Set {p},
          bag<-Set{q}),
       p: MMC!Element(  P_name<-s.predicat_list) ,
             q: distinct MMC!Element_Bag foreach(p in (
    s.getLiValue()))  (value<-p)
             }
```

Listing 6.2 – *La règle de transformation d'une liste ordonnée en une ressource RDF*

Règle 3 : Le listing 6.3 montre le code source en ATL de la règle qui permet de transformer une liste de définition en une ressource RDF imbriquée. L'élément « *List_Def* » devient une ressource qui a :

- Sujet est *List_Def.URI_APP*.
- Prédicat est la concaténation de « *List_Def.URI_APP* » avec « *List_Def.Title* ».
- L'objet est un nœud vide ayant :
  — Les prédicats sont la concaténation de « *List_Def.URI_APP* », « *List_Def.Title* » et « *DT.value* ».
  — Les objets sont les valeurs de « *DD.Value* ».

```
rule ListDef2Description {
 from s: MMS!List_def
 to   t: MMC!Description (
         about <- s.URI_APP,
         Property<-Set{p},
         Description<-Set{q}),
       p: MMC!Element(
         P_name<-s.predicat_list_def,
         O_name<-'NB'),
       q: MMC!Description (
         about<-'NB',
         Property<-f),
       f: distinct MMC!Element foreach(n in (s.
    getDTValues()))(
         P_name<-s.predicat_list_def+'/'+n,
         O_name<-s.getDDValue)
  }
```

Listing 6.3 – *La règle de transformation d'une liste de définition en une ressource RDF imbriquée.*

Règle 4 : Le listing 6.4 montre le code source en langage ATL de la règle qui permet de transformer un lien en ressource RDF. L'élément « *Link* » devient une ressource qui possède :

- Sujet est la valeur de l'élément « *Link.URI_APP* ».
- Prédicat est la concaténation des valeurs des éléments « *Link.URI_APP* » et « *URI_source* ».
- Object est la valeur de l'attribut « *URI_target* ».

```
rule LinK2Description {
 from s: MMS!LinK
 to   t: MMC!Description (
         about <- s.URI_APP,
         Property<-Set {p}),
       p: MMC!Element(
         P_name<-s.predicat_link,
         O_name<-s.URI_target)
              }
```

Listing 6.4 – *La règle de transformation d'un lien en une ressource RDF.*

Le Tableau 6.2 résume les règles de transformation présentées ci-dessus. Il comporte quatre colonnes principales : *numéro de règle, éléments source, éléments cibles* et *le Matching*. La première colonne indique le numéro de règle selon leur ordre ci-dessus. La deuxième et la troisième colonne sont divisées

en deux sous colonnes : *Elément* et *Attribut* ; *Elément* pour présenter le nom de l'élément du méta-modèle source ou cible, et *Attribut* pour répertorier les attributs de l'élément en cours. La dernière colonne *(Matching)* montre la correspondance entre les éléments du méta-modèle source et cible comme elles sont expliquées dans les règles ci-dessus.

TABLE 6.2 – *Mappage des éléments du MMS vers des éléments du MMC*

La règle	Elément source		Elément cible		Matching
	Elément	**Attribut**	**Elément**	**Attribut**	
1	Table	URI_APP	Description	About	About=URI_APP
		Caption	Property	P_name	P_name=URI_APP+caption
				O_name	O_name="nested resources"
	TR	TH	Description	About	About=URI_APP+ caption+ TH
	TD	Head	Property	P_name	P_name=URI_APP+ head
		Value		O_name	O_name=value
2	List	URI_APP	Description	About	About=URI_APP
		Title	Property	P_name	P_name=URI_APP+Title
	LI	Value		O_name	O_name=null
			Bag	Value	Bag.Value=LI.Value
3	List_def	URI_APP	Description	About	About=URI_APP
		Title	Property	P_name	P_name=URI_APP+Title
				O_name	O_name="NB"
	DT	Value	Description	About	About="NB"
			Property	P_name	P_name=URI_APP+Title+DT.value
	DD	Value		O_name	O_name=DD.Value
4	Link	URI_APP	Description	About	About=URI_APP
		URI_source	Property	P_name	P_name=URI_APP+URI_source
		URI_target		O_name	O_name=URI_target

La couche M2

M2 est la couche du méta méta-modèle. Dans notre cas, il s'agit du langage méta-méta-modèle Ecore. Il est défini dans la structure EMF [Steinberg *et al.* 2003]. A ce niveau, il ne reste plus qu'à lancer le processus de transformation pour avoir le fichier RDF de sortie.

6.3.3 Étape de raffinement

Cette tâche, illustrée dans l'organigramme de la Figure 6.9, vise à affiner automatiquement le fichier RDF obtenu à l'étape précédente. On le récrit dans un autre fichier RDF affiné, qui est conforme à la syntaxe RDF définie par le consortium W3C.

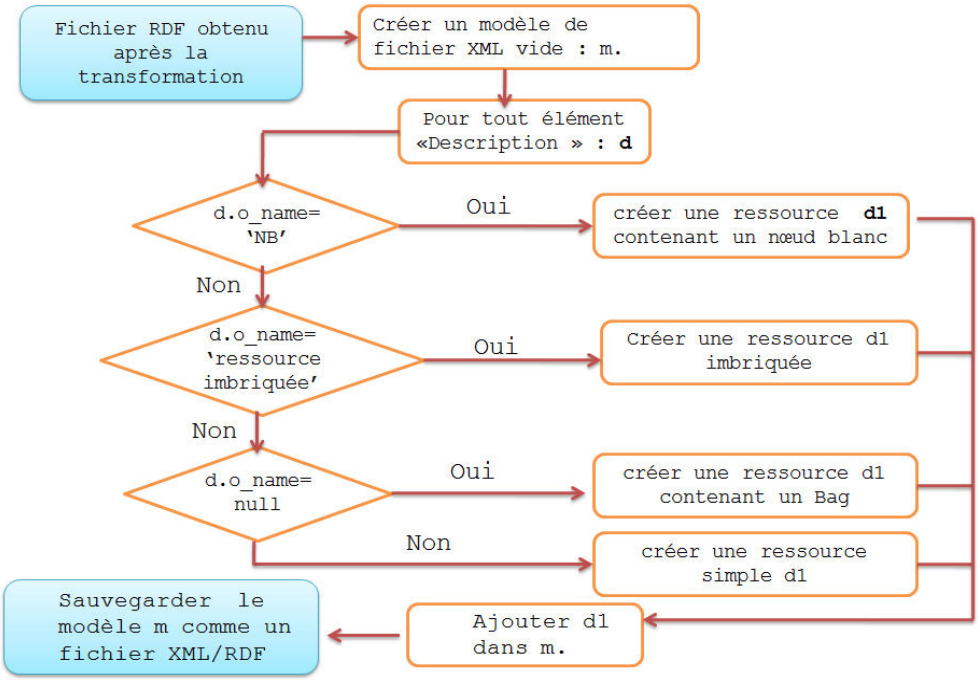

FIGURE 6.9 – *Organigramme de raffinement*

Le résultat de l'étape de raffinement est validé par un validateur RDF (nous avons utilisé l'outil online *"Validator RDF"* [1] fourni par le W3C). Le validateur RDF de W3C nous fournit un graphe RDF, et nous donne la mention suivante *«Votre document RDF est validé avec succès. »*
Reprenons l'exemple de déroulement précédent. Le fichier de sortie obtenu à la fin du processus de transformation est donné dans le Listing 6.5. Ce fichier sera affiné afin de construire un fichier RDF conforme à toutes les règles du W3C concernant le langage RDF.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI
       " xmlns="">
3  <Description xmi:version="2.0" xmlns:xmi="http://www.omg.org
       /XMI" xmlns="" about="www.WEBApp.dz">
4   <Property  P_name="www.WEBApp.dz/Items Liste"/>
5    <bag  value="Article 1"/>
6      <bag  value="Article 2"/>
7      <bag  value="Article 3"/>
8      <bag  value="Article 4"/>
9  </Description>
10 </xmi:XMI>
```

Listing 6.5 – *Le fichier RDF généré par le système.*

Le Listing 6.6 affiche le fichier RDF raffiné de l'exemple de déroulement validé par *W3C Validator*.

1. https://www.w3.org/RDF/Validator/

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:j.0="http://www.WEBApp.dz/Items ">
<rdf:Description rdf:about="http://www.WEBApp.dz">
 <j.0:Liste>
  <rdf:Bag>
   <rdf:li>Article 1</rdf:li>
   <rdf:li>Article 2</rdf:li>
   <rdf:li>Article 3</rdf:li>
   <rdf:li>Article 4</rdf:li>
  </rdf:Bag>
 </j.0:Liste>
</rdf:Description>
</rdf:RDF>
```

Listing 6.6 – *Le fichier RDF après le raffinement.*

6.4 Evaluation

6.4.1 L'outil implémenté

Pour nos expériences, nous avons implémenté dans l'environnement éclipse *(Eclipse-Foundation)* un outil appelé HTML2RDF (voir la Figure 6.10). Nous utilisons *Eclipse Modeling Framework (EMF)* [Steinberg *et al.* 2003] pour décrire les méta-modèles, et le langage de transformation *(ATL)* pour écrire les règles de transformation. Le processus de raffinement est également implémenté en langage Java.

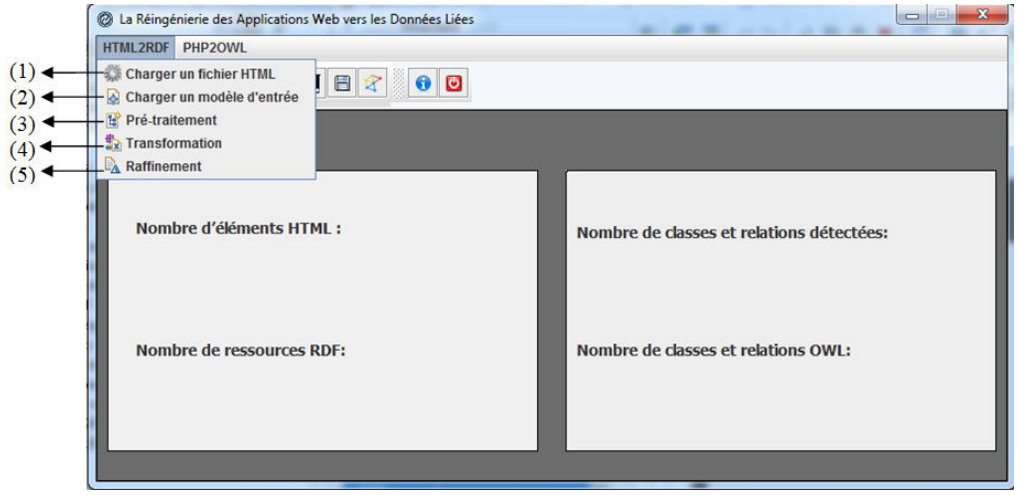

Figure 6.10 – *L'interface principale de l'outil* HTML2RDF

La Figure 6.10 représente l'interface principale de l'outil *HTML2RDF*. Elle fournit cinq fonctionnalités : (1) Chargement d'un fichier HTML, (2) Permet de charger un modèle d'entrée (un fichier HTML prétraité manuellement), (3) Lancement du processus de prétraitement afin de construire un modèle d'entrée, (4) Lancement de la transformation basée sur IDM et (5) Raffinement du modèle RDF récupéré de l'étape précédente.

Pour évaluer l'efficacité de notre outil, nous avons choisi des échantillons d'Infoboxes de Wikipédia, et nous avons créé pour chaque échantillon son document RDF qui représente des données de l'infobox dans une collection de triplets. Ensuite, nous avons généré un graphe RDF, et nous avons validé le résultat obtenu à l'aide du validateur RDF du W3C (Prud'hommeaux, 2006). Nous avons obtenu pour chaque échantillon le même message *"Votre document RDF est validé avec succès"*.
Les Infoboxes sont extraites sous forme de tableaux HTML. Nous appliquons un traitement préalable particulier qui supprime les balises indésirables comme <*sup*>, <*abbr*>, <*time*>, ... et nous ne conservons que les balises utiles, telles que <*table*>, <*td*>, <*tr*> et <*a*>, i.e., les balises contenant des données. Notons que tous les éléments <*TD*> contenant des valeurs multiples sont transformés en une liste. Les multi-valeurs sont séparées par une virgule ou une balise <*br*>. Les éléments <*TD*> restants (contenant une seule valeur) forment la table.

6.4.2 Expérimentations et discussion des résultats

Les échantillons (Exp) choisis de Wikipédia sont *l'Afrique*, *l'Algérie*, *l'Europe*, *Tim Berners-Lee*, *Edgar Frank Codd*, *MalcomX* et *Microsoft*. Le Tableau 6.3 montre, pour chaque échantillon, le résultat d'évaluation avec les métriques *Rappel*, *Précision* et *F-mesure* [David & Powers 2011].
Il existe trois valeurs *All*, *Correct* et *Expert* utilisées pour définir les valeurs d'évaluation. *All* correspond au nombre de triplets obtenus par le validateur RDF W3C. *Correct* est le nombre de triplets contenant des informations après la suppression des triplets contenant un nœud vide (blanc). *Expert* est le nombre de triplets qui doivent être renvoyés par notre outil (calculé à partir de la page Wikipédia).

TABLE 6.3 – *Résultant de l'évaluation.*

Exp / Triplets RDF	Afrique	Algérie	Europe	Tim Berners Lee	E. Codd	Malcom X	Microsoft
All	63	60	64	20	27	38	70
Correct	56	42	59	12	17	29	48
Expert	56	43	59	14	17	29	49
Rappel	1,00	0,98	1,00	0,86	1,00	1,00	0,98
Précision	0,89	0,70	0,92	0,60	0,63	0,76	0,69
F-mesure	0,94	0,82	0,96	0,71	0,77	0,87	0,81

Les expressions du Rappel et Précision sont données par les équations suivantes (6.1, 6.2) :

$$precision = \frac{le\ nombre\ de\ triplets\ corrects}{le\ nombre\ de\ tous\ les\ triplets\ obtenus} \tag{6.1}$$

$$rappel = \frac{le\ nombre\ de\ triplets\ corrects}{le\ nombre\ de\ triplets\ qui\ doivent\ etre\ retournes\ par\ le\ systeme\ (donne\ par\ l'EXPERT)} \tag{6.2}$$

La F-mesure (6.3) combine *la Précision* et *le rappel* dans une seule métrique harmonique. *La F-mesure* est exprimée comme suit :

$$F - mesure = 2 * \frac{prcision * rappel}{prcision + rappel} \tag{6.3}$$

La Figure 6.11 illustre les trois mesures : Rappel, Précision et F-mesure sous forme d'histogrammes.

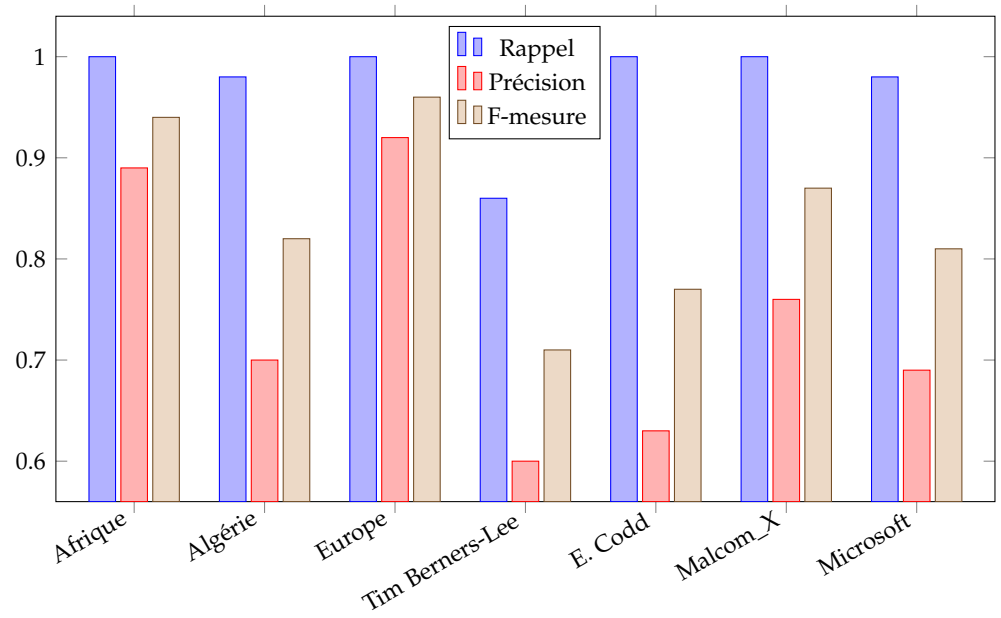

FIGURE 6.11 – *Graphe d'évaluation.*

La meilleure valeur pour toutes les mesures est 1 ; cela signifie que les triplets obtenus par notre outil sont les mêmes que ceux donnés par l'expert. Nous ne considérons donc que le résultat qui dépasse 0,8 pour la F-mesure comme bon résultat. Nous constatons que nous avons eu un résultat qui dépasse 0,8 pour tous les échantillons, à l'exception des deux échantillons : Tim Berners-Lee et Edgar Frank Codd. Avec une moyenne (pour tous les échantillons) égale à 0,84. Nous pouvons dire que notre système donne des résultats encourageants.
Pour certains échantillons tels que *Tim Berners-Lee* et *Edgar Frank Codd*, les valeurs étaient inférieures à 0,8. Ces faibles valeurs sont liées à l'existence de nombreux nœuds blancs dans le résultat, qui ont été comptabilisés comme des ressources, et c'est pourquoi nous avons un nombre élevé de ressources (en valeur *ALL*) par rapport au nombre de ressources *Correctes* (on ne compte que les triplets d'information, c.à.d. sans les nœuds blancs). Par conséquent, nous avons obtenu des valeurs faibles de *Précision* et de *F-mesure*.
L'outil *HTML2RDF* a deux objectifs : i) Transformer les données stockées dans les pages HTML (provenant de la base de données) en format RDF. ii) Transformer les applications Web classiques en applications sémantiques.

6.5 Conclusion

Dans le Web sémantique, les données sont accessibles par les utilisateurs et par les programmes. Par conséquent, les programmes sont devenus capables d'effectuer des opérations bien définies sur les données, et de déduire de nouvelles informations à partir des données existantes. Le Web sémantique offre un

langage RDF pour exprimer les données de manière à les rendre accessibles, et les donner un sens. Cependant, sur le Web traditionnel, les données se trouvent soit dans des bases de données, soit dans des documents HTML ; dans les deux cas, ils sont moins exploitables par les machines et n'ont pas de sens. Pour les rendre dans un format sémantique, nous devons transformer les applications Web en applications sémantiques.
Nous avons proposé une approche qui pallie le problème cité au-dessus. Cette approche est basée sur l'ingénierie dirigée par les modèles (IDM). Elle consiste en trois phases. La première phase sert à préparer un modèle contenant les données de l'application Web que nous voulons transformer. La deuxième phase forme le noyau de l'approche ; elle permet de transformer le modèle d'entrée en un fichier RDF. La dernière phase a pour but de raffiner le modèle de sortie (le fichier RDF) obtenu précédemment.
L'approche est concrétisée par un outil appelé *HTML2RDF*. L'efficacité de l'outil est prouvée par un ensemble d'expériences. Ces expériences sont illustrées par une transformation de certaines Infoboxes des pages Wikipédia en documents RDF. Les résultats obtenus (documents RDF) sont validés par le validateur de W3C, et évalués en utilisant la métrique *F-mesure* qui nous a donné un résultat encourageant.

Approche II

7

Sommaire

7.1 Introduction 95
7.2 Approche proposée 95
7.2.1 La phase de Rétro-ingénierie 96
7.2.2 La Phase d'ingénierie directe 98
7.3 Exemple de déroulement 100
7.4 Expériences et discussion des résultats 103
7.5 Conclusion 105

7.1 Introduction

Dans ce chapitre, nous allons présenter notre approche pour transformer les applications Web en ontologie OWL [Bouougada *et al.* 2017]. L'approche proposée commence par un processus de rétro-ingénierie qui vise à récupérer la conception d'origine à partir du code source de l'application Web, en particulier, à partir des scripts HTML et PHP, en utilisant des techniques de compréhension de programme. Ensuite, un processus d'ingénierie directe crée une ontologie OWL à partir des diagrammes récupérés en appliquant un ensemble de règles de transformation.
L'approche proposée sera détaillée dans les sections suivantes, et sera expliquée à travers un exemple de déroulement, et évaluée par des expériences sur quelques applications Web. Le résultat obtenu est discuté et interprété par des métriques d'évaluation.

7.2 Approche proposée

La Figure 7.1 montre l'architecture globale de l'approche proposée. Elle se compose principalement de deux phases, rétro-ingénierie et ingénierie directe. La première phase vise à créer un diagramme de classes à partir du code source de l'application Web (un ensemble de fichiers PHP et HTML) en utilisant des techniques de compréhension de programme, notamment la technique des patterns (motifs). Ce processus s'appelle également ingénierie inverse. La deuxième phase comprend trois étapes : transformation, enrichissement et instanciation. La première étape permet de mapper le diagramme de classes précédemment obtenu vers une ontologie RDFS légère ou OWL. La deuxième étape consiste à enrichir l'ontologie résultante avec certaines relations basées sur une ontologie de domaine. La troisième étape vise à générer des instances en utilisant l'ontologie et les pages HTML précédemment générées. Les instances sont données en format RDF.

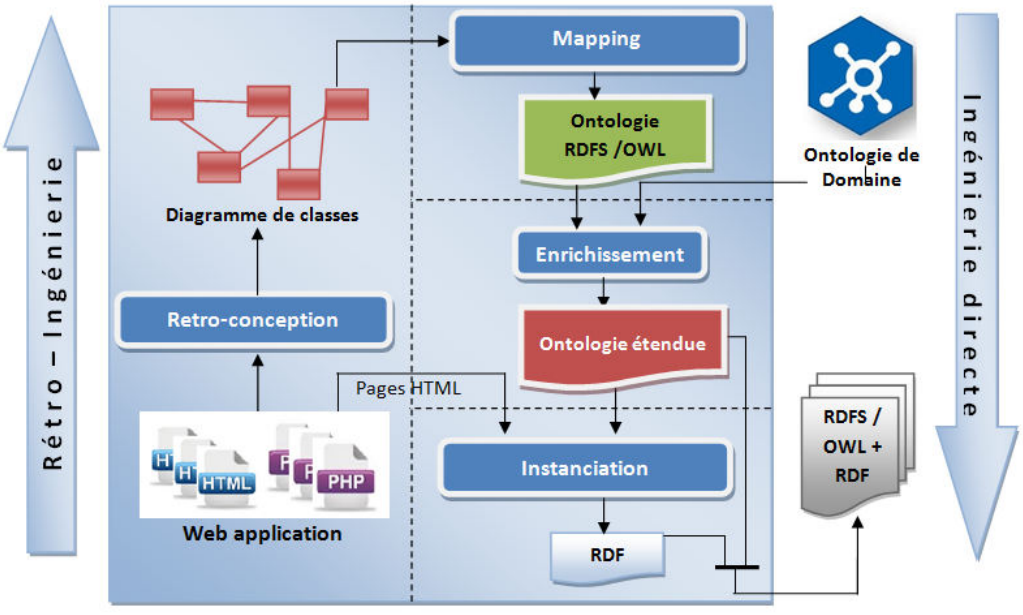

Figure 7.1 – *Génération d'ontologie à partir une application Web*

7.2.1 La phase de Rétro-ingénierie

C'est une étape cruciale de notre système. Elle permet de représenter l'application Web à un niveau d'abstraction plus élevé avec des diagrammes de classes. Pour réaliser ce processus de rétro-ingénierie, nous avons proposé une solution basée sur des techniques de compréhension de programme, notamment la technique de correspondance de patterns. Après avoir défini les patterns, un ensemble de règles est appliqué pour générer un diagramme de classes.
La Figure 7.2 montre un organigramme expliquant comment transformer le code source (PHP + HTML) en diagramme de classes.

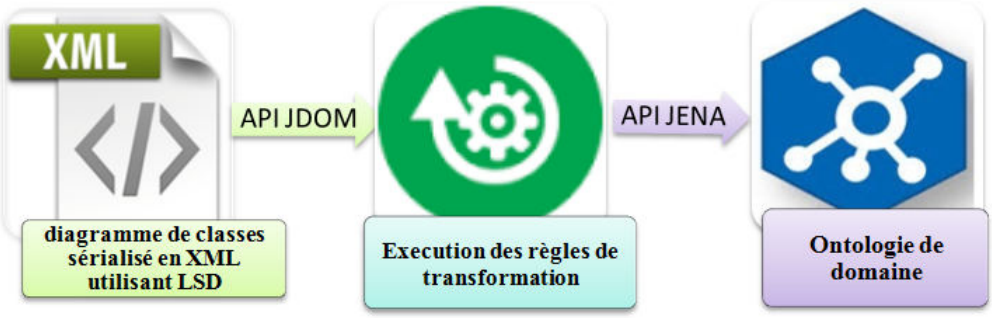

FIGURE 7.2 – *Organigramme de rétro-ingénierie*

L'analyse du code source se concentre sur les patterns de base de données présents dans les fichiers PHP constituant l'application Web. Ces patterns divisés en deux catégories : l'une contient *"insert into"*, *"update"* et *"delete"*, et l'autre contient le pattern *"select"*. Avec la première catégorie, nous pouvons extraire les concepts et leurs caractéristiques. A travers la seconde catégorie et les caractéristiques communes (attributs) entre les concepts, nous établissons les relations entre eux.

Pattern "insert into ... (...) values (...)" : permet de générer des concepts (classes). Dans ce cas, on distingue deux situations : avec un schéma et sans schéma. Dans le premier cas nous appliquons les règles 1 et 2, le second cas on applique les règles 1, 2 et 3.

Règle 1 : permet de générer une classe et ses attributs dans le cas "avec un schéma". C'est le moyen le plus simple, car nous pouvons construire un concept complet avec ses attributs. Le nom d'une classe (concept) est le mot après le modèle "insert into", et les attributs sont les mots qui suivent le nom du concept ; ces mots sont mis entre deux parenthèses.

Règle 2 : détermine les types des attributs générés par l'application de la règle 1. Le type de chaque attribut sera déterminé à l'aide d'un formulaire HTML correspondant à chaque pattern "insert into". Chaque attribut correspond à une balise HTML "input". cette balise contient un attribut nommé "type" et qui prend l'une des valeurs : "texte", "nombre", etc. grâce à ces valeurs, nous pouvons déterminer le type de chaque attribut.

Règle 3 : permet de générer une classe et ses attributs dans le cas "sans schéma". Le nom de la classe est déterminé comme dans la règle 1. Les attributs et leurs types sont extraits à partir des formulaires HTML qui

se trouvent dans le fichier PHP à traiter. Les attributs sont les valeurs de l'attribut "name" s'ils existent. Sinon, les attributs sont les mots qui se trouvent avant le mot clé "input". Les types d'attributs sont déterminés comme dans la règle 2.

Pattern "update... set... where" et "delete from... where" : nous utilisons ces patterns pour générer des classes si elles ne sont pas déjà générées par le pattern "insert into". La règle 4 explique comment faire ça.

Règle 4 : le nom de la classe est le mot trouvé après "update" ou "delete from". Les attributs sont les mots trouvés après le mot clé "set" et "where". Les types d'attributs sont extraits comme mentionné dans la règle 3 en utilisant les formulaires HTML correspondants.

Pattern "select ... from... where..." : le pattern "select" est utilisé pour affiner les classes générées comme décrit dans la règle 5.

Règle 5 : cette règle permet d'extraire les relations entre les classes détectées par les règles précédentes. Les classes trouvées après le mot clé "from" sont en relation. De plus, elle permet d'ajouter des attributs s'ils n'ont pas déjà été définis. ces attributs se trouvent après le pattern "select".

Règle 6 : cette règle vise à affiner le diagramme de classes en ajoutant la cardinalité pour chaque relation. Ainsi, nous recherchons les attributs communs entre les classes, à savoir les clés étrangères. On distingue alors deux cas : le premier représente la relation pères-fils (1- *) et l'autre représente la relation multiple (* - *).

Cas 1 : si un attribut A apparaît dans les classes C1 et C2 alors :

- Il existe une relation entre les classes C1 et C2.
- La cardinalité est de type pères-fils, ce qui implique une cardinalité de 1 .. *
  - pour la classe C1 mincard = 0 et maxcard = 1
  - pour la classe C2 mincard = 0 et maxcard = *

Cas 2 : si un attribut A de la classe C1 et un attribut B de la classe C2 apparaissent dans la classe C, alors : La classe C devient une classe d'association entre C1 et C2.

- La cardinalité est égale à *.
- La mincard = 0 et maxcard = * pour les classes C1 et C2.

Dans cette phase (Rétro-ingénierie), nous définissons un langage spécifique au domaine (LSD ou en anglais DSL Domain Specific language) qui permet de décrire un diagramme de classes en XML en respectant toutes les recommandations d'un diagramme de classes défini par OMG [OMG 2017]. Ce langage nous permet de décrire une classe en détail et ses interactions avec les autres classes. Cette phase de sérialisation est effectuée après l'extraction de toutes les classes. Le tableau 7.1 résume les éléments du langage de sérialisation.

TABLE 7.1 – *Eléments du langage de sérialisation*

Elément	Description E.	Attribut	Description A.
CD	définit un diagramme de classes	Cdname	définit le nom du diagramme de classes
Class	définit une classe	Name	définit le nom de la classe
		Mincard	indique la cardinalité minimale gauche

		Maxcard	indique la cardinalité maximale gauche
Attribute	définit les attributs de la classe ou de l'association	Name	définit le nom de l'attribut
		Type	définit le type d'attribut
Subclass	définit une sous-classe	Name	définit le nom de la sous-classe
Association	définit une classe d'association	AssName	définit le nom de l'association de classe
		Domain	spécifie le nom de la classe de domaine
		Range	spécifie le nom de la classe de plage

La colonne *"Elément"* correspond à l'élément considéré. Colonne *"Description E."* montre son rôle. La colonne *"Attribut"* cite les attributs pouvant être associés à l'élément. La colonne *"Description A."* décrit le rôle de l'attribut.

7.2.2 La Phase d'ingénierie directe

Elle vise à générer l'ontologie à partir du diagramme de classes obtenu précédemment en trois étapes :

Mapping

Dans un premier temps, un ensemble de règles de transformation (Mapping) est appliqué pour atteindre le but ultime. La figure suivante (7.3) montre une vue d'ensemble de ce processus.

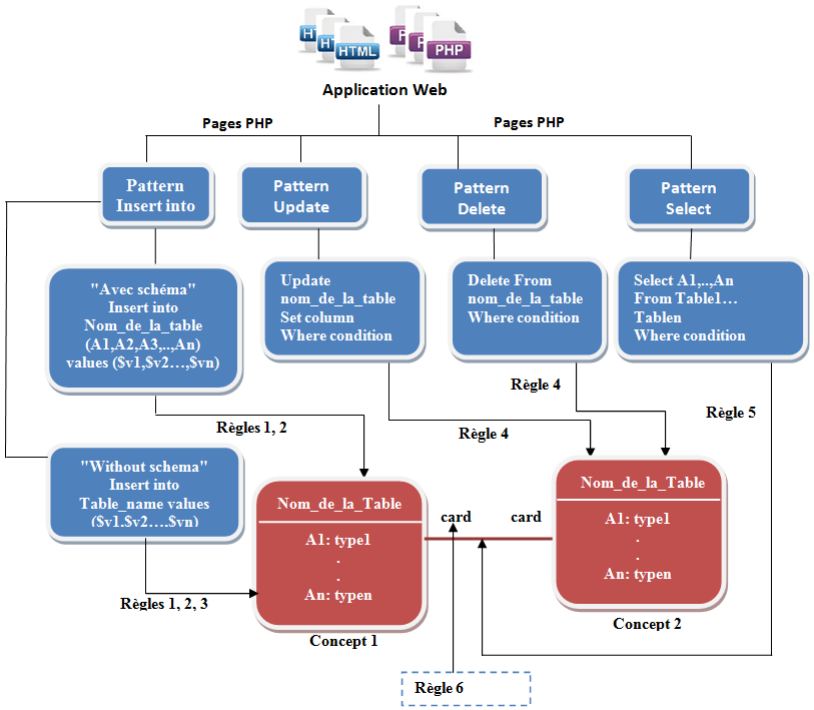

FIGURE 7.3 – *Mapping du diagramme de classes en Ontologie*

La Figure 7.3 montre le processus de Mapping du diagramme de classes en ontologie. Le fichier XML est exploré avec l'API *JDOM* [1] dans la première phase. En utilisant les règles de Mapping, chaque élément du fichier XML sera converti en son équivalent dans l'ontologie. L'API *JENA* [2] est utilisée pour écrire l'ontologie en format XML. Les règles de Mapping sont les suivantes :

Règle 1 : mappe l'élément *"Class"* du diagramme de classes en élément *"owl :class"* :
- Le nom de *"owl :class"* sera l'attribut "name" de l'élément *"Class"*.

Règle 2 : permet de mapper les attributs de classe en propriétés de type de données comme suit :
- L'élément *"rdf :DataTypeProperty"* prend comme valeur le nom de l'élément *"Attribute"*.
- L'élément *"rdfs :range"* prend comme valeur le type de l'élément *"Attribute"*.
- L'élément *"rdfs :domain"* prend comme valeur le "nom" de l'élément *"Class"*.

Règle 3 : permet de mapper les relations entre les classes en des propriétés d'objet comme suit :
- L'élément *"rdf :ObjectProperty"* prend comme valeur le nom de l'élément *"Relation"*.
- L'élément *"rdfs :domain"* prend comme valeur le nom de la classe source de l'élément *"Relation"*.
- L'élément *"rdfs :range"* prend comme valeur le nom de la classe cible de l'élément *"Relation"*.

Règle 4 : permet de créer des contraintes de cardinalité sur les propriétés d'objet comme suit :
- L'attribut *"min_card"* devient *"owl :minCardinality"*.
- L'attribut *"max_card"* devient *"owl :maxCardinality"*.

Règle 5 : vise à créer une sous-classe d'ontologie à partir d'un élément *"Subclass"* du diagramme de classes, en faisant correspondre le nom de l'élément *"Subclass"* avec l'attribut *"rdf :resource"* de l'élément *"rdfs :subClassOf"*.

Enrichissement

Cette deuxième étape du processus d'ingénierie directe vise à supprimer toutes les ambiguïtés de certaines relations détectées. En particulier, les relations d'héritage et les relations *1-** peuvent être représentées de la même manière. Deuxièmement, il vise à ajouter des relations entre les concepts pour étendre l'ontologie obtenue, comme la relation *"is-a"*, qui donne l'aspect hiérarchique entre les concepts. Ce second objectif est atteint en consultant une ontologie de domaine.

Instanciation

Jusqu'à présent, nous ne prenons pas en compte la génération de données depuis l'application Web. Nous avons seulement construit un schéma de données sous forme d'une ontologie. Pour transformer les données en instances

1. http://www.jdom.org/downloads/index.html
2. http://jena.apache.org

(ressources RDF), nous utilisons d'un côté les pages HTML et de l'autre côté l'ontologie générée pour vérifier les instances.
Dans ce contexte, des instances de l'ontologie sont créées à partir de certains éléments HTML, tels que les tableaux et les listes [Bouougada *et al.* 2015]. Les instances sont enregistrées en tant que triplets RDF. L'approche présentée dans [Bouougada *et al.* 2018] est basée sur l'ingénierie dirigée par les modèles (IDM). Elle permet de transformer les données de l'application Web (données HTML) en triplets RDF. Les détails de l'approche sont donnés dans le chapitre précédent.

7.3 Exemple de déroulement

Dans cette section, nous souhaitons illustrer toutes les étapes de l'approche proposée à travers d'un exemple de déroulement.
Listing 7.1 représente un extrait du code source PHP d'une application Web qui offre des annonces de véhicules afin de faciliter la vente.

```
1  <?php
2  $sql = mysql_query("SELECT * FROM  annonce as gr INNER JOIN
       véhicule as em
3  ON gr.id_ann= em.id_ann  WHERE gr.id_ann  = '{$id_ann}' ");
4  $date = date("F,d Y");
5  $part = $_FILES['file']['name'];
6  $sql = mysql_query("INSERT INTO annonce (id, location,
       nbr_phone, email, time_limit, prix, photo, date) VALUES (
       $id_clt,'{$_POST['location']}','{$_POST['nbr_phone']}','{
       $_POST['email']}','{$_POST['time_limit']}','{$_POST['prix
       ']}','$part','$date')"); ?>
7  <td><input class="email" type="text" name="nbr_phone" required=
       "" pattern=".{10,}" /></td>
8  <td><select class="form-control" name="location" style="width
       :150px;">
9  <input class="email" type="text" placeholder="Email" name="
       email" required=""
10 pattern="([\w-\.]+@([\w-]+\.)+[\w-]{2,4})?" title="Enter a
       valid email" /></td>
11 <select class="form-control" name="time_limit" style="width:150
       px;">
12 <input class="email" type="text" placeholder="Prix" name="prix"
        required=""  />
```

Listing 7.1 – *Extrait du code source PHP*

Maintenant, nous appliquons les règles de la phase de rétro-ingénierie sur le code PHP présenté ci-dessus. *La règle 1* génère le concept *"Annonce"*. En outre, les autres règles sont utilisées, que ce soit pour générer d'autres concepts ou pour affiner les concepts existants. Par exemple, *la règle 5* génère le concept *"Véhicule"* et *la règle 6* détermine la cardinalité. La Figure 7.4 représente le résultat de ce processus en tant que diagramme de classes. La sérialisation de diagramme de classes obtenu est donné dans le Listing 7.2.

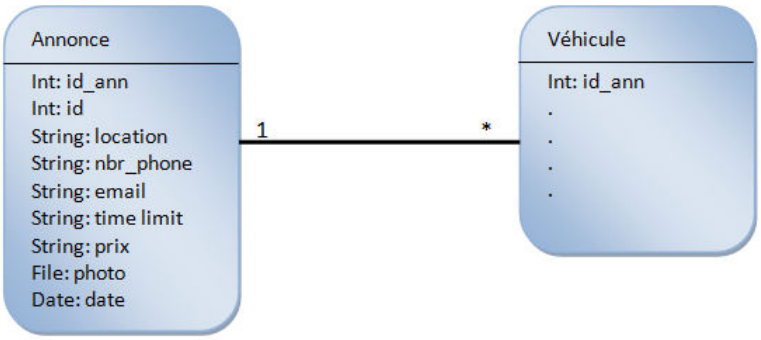

FIGURE 7.4 – *Diagramme de classes généré par le processus de retro-ingénierie*

```
<CD dcname="notice" mincard=0 maxcard=1>
 <class name="annonce" mincard="1" maxcard="n">
  <Attribute name="id_ann" type="int"/>
  <Attribute name="id" type="int"/>
  <Attribute name="location" type="string"/>
  <Attribute name="nbr_phone" type="string"/>
  <Attribute name="email" type="string"/>
  <Attribute name="time limit" type="string"/>
  <Attribute name="prix" type="string"/>
  <Attribute name="photo" type="File"/>
  <Attribute name="date" type="Date"/>
 </class>
 <class name="véhicule" mincard=0 maxcard=-1>
   <Attribut name="id_ann" type="int"/>
 </class>
<Association AssName="annonce_véhicule"
    domain="annonce" range="Vehicule"   />
</CD>
```

Listing 7.2 – *Diagramme de classes sérialisé en XML*

De la même manière, nous appliquons les règles définies dans la phase d'ingénierie directe sur le modèle obtenu du processus précédent (diagramme de classes). En effet, les classes *"owl :annonce"* et *"owl :vehicule"* et leurs attributs sont créés selon *les règles 1 et 2*. Le *"owl :ObjectProperty"* est construit en appliquant *la règle 3*. Le résultat de cette étape est illustré dans Listing 7.3. Cet extrait représente une ontologie qui correspond au diagramme de classes de la Figure 7.4.

```
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:owl="http://www.w3.org/2002/07/owl#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
<owl:Ontology rdf:about="http://www.cunivnaama.dz/Onto_annonce#
   "/>
 <owl:Class rdf:about="http://www.cunivnaama.dz/Onto_annonce#
   véhicule"/>
 <owl:Class rdf:about="http://www.cunivnaama.dz/Onto_annonce#
   annonce"/>
 <owl:ObjectProperty rdf:about="http://www.cunivnaama.dz/
   Onto_annonce#annonce_véhicule">
```

```
10    <rdfs:range rdf:resource="http://www.cunivnaama.dz/
       Onto_annonce#véhicule"/>
11    <rdfs:domain rdf:resource="http://www.cunivnaama.dz/
       Onto_annonce#annonce"/>
12    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Class"/
       >
13   </owl:ObjectProperty>
14   <owl:DatatypeProperty rdf:about="http://www.cunivnaama.dz/
       Onto_annonce#email">
15       <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#
       string"/>
16       <rdfs:domain rdf:resource="http://www.cunivnaama.dz/
       Onto_annonce#annonce"/>
17  </owl:DatatypeProperty>
```

Listing 7.3 – *Extrait de l'ontologie générée*

Après avoir enrichi l'ontologie en ajoutant des relations entre les concepts, bien sûr, s'ils existent dans l'ontologie du domaine, nous créons des instances à partir des éléments HTML de l'application Web correspondante. Cela se fait selon le travail présenté dans [Bouougada *et al.* 2015, Bouougada *et al.* 2018], qui permet de transformer des éléments HTML tels que les listes et les tableaux en données liées RDF. Listing 7.4 donne quelques instances d'une *"Annonce"* en format RDF.

```
1       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
2       xmlns:j.0="http://www.WEBApp.dz/"
3       xmlns:j.1="http://www.WEBApp.dz/ad_on_1/">
4    <rdf:Description rdf:about="http://www.WEBApp.dz">
5     <j.0:ad_on_2 rdf:parseType="Resource">
6     <j.1:date>06/03/2017</j.1:date>
7     <j.1:photo>www.WEBApp.dz/img/im5.jpg</j.1:photo>
8     <j.1:price>25 000</j.1:prix>
9     <j.1:time_limit>2 month</j.1:time_limit>
10    <j.1:email>ab5@webapp.dz</j.1:email>
11    <j.1:nbr_phone>213611141600</j.1:nbr_phone>
12    <j.1:location>mascara</j.1:location>
13    <j.1:id>5</j.1:id>
14    <j.1:id_ann>2</j.1:id_ann>
15    </j.0:ad_on_2>
16  </rdf:Description>
17       <j.0:ad_on_1 rdf:parseType="Resource">
18    <j.1:date>12/12/2017</j.1:date>
19    <j.1:photo>www.WEBApp.dz/img/im12.jpg</j.1:photo>
20    <j.1:price>12 000</j.1:prix>
21    <j.1:time_limit>2 weeks</j.1:time_limit>
22    <j.1:email>AB12@webapp.dz</j.1:email>
23    <j.1:nbr_phone>213611111100</j.1:nbr_phone>
24    <j.1:location>NAAMA</j.1:location>
25    <j.1:id>12</j.1:id>
26    <j.1:id_ann>1</j.1:id_ann>
27    </j.0:ad_on_1>
28  </rdf:Description>
29  </rdf:RDF>
```

Listing 7.4 – *Instances du concept "Annonce" en format RDF*

7.4 Expériences et discussion des résultats

La Figure 7.5 représente l'interface de l'outil *PHP2OWL*. Elle est composée de quatre fonctions, dont (1) permet de charger un (des) fichier(s) PHP, (2) permet de lancer l'étape de réingénierie du diagramme de classes à partir du (des) fichier(s) chargé(s),(3) permet d'enregistrer le diagramme de classes obtenu, et (4) permet de générer une ontologie OWL à partir du diagramme de classes obtenu précédemment.

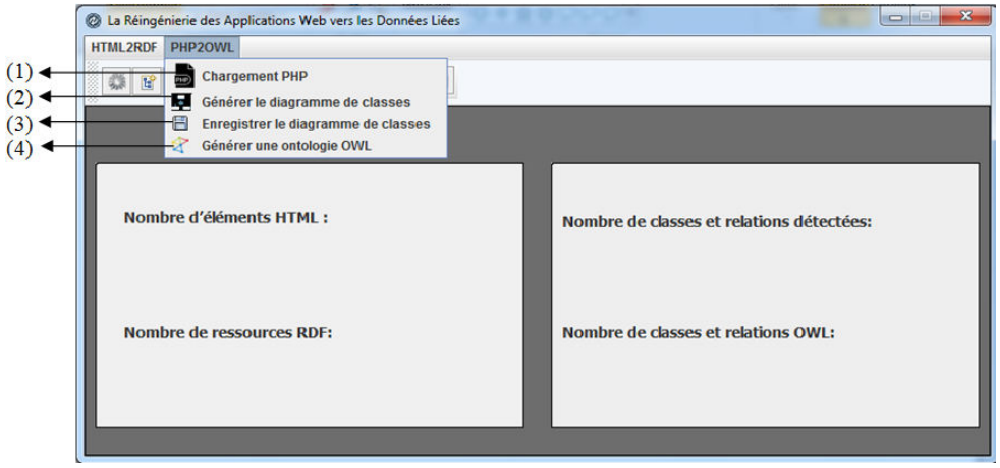

Figure 7.5 – *Interface de l'outil PHP2OWL*

Afin d'évaluer la précision et l'efficacité de l'outil *PHP2OWL*, nous utilisons différentes applications Web pour vérifier notre système. Tableau 7.2 montre les résultats empiriques obtenus après avoir effectué des tests sur un ensemble d'applications Web.

Table 7.2 – *Résultats d'évaluation*

		Application Web			
		Gestion de produits	**Commerce électronique**	**Gestion de la bibliothèque**	**Gestion de la réservation hôtelière**
Elément	**Concepts récupérés**	7	8	3	7
	Relations récupérées	6	7	2	5
	Concepts pertinents	7	8	3	7
	Relations pertinentes	6	7	2	5
	Concepts existants	8	8	3	7
	Relations existantes	6	8	2	6
Métrique	**Précision**	1	1	1	1
	Rappel	0,93	0,94	1	0,92
	F-mesure	0,96	0,96	1	0,96

Le Tableau 7.2 montre, pour chaque application Web, les résultats d'évaluation en termes de *Précision*, *Rappel* et *F-mesure*. Plusieurs applications Web ont été prises en compte dans nos expériences, notamment la gestion des produits, le site Web du commerce électronique (e-Commerce), la gestion de la bibliothèque et gestion de la réservation hôtelière. Notez que toutes ces applications sont des applications prototypes implémentées pour un objectif d'expérimentation.
Le tableau 7.2 comprend deux lignes principales et quatre colonnes. Les colonnes représentent les applications Web sur lesquelles nous souhaitons évaluer notre outil *PHP2OWL*. Les lignes contiennent des valeurs nécessaires pour l'évaluation. la première ligne comporte six sous lignes dont chacune représente une valeur d'évaluation. la deuxième ligne est composée de trois sous ligne dont chacune représente une métrique dévaluation.
Donc, pour chaque application Web, nous avons mentionné le nombre de concepts et de relations récupérés par notre outil, nous avons également mentionné les concepts et les relations pertinents récupérés par notre outil. Enfin, nous mettons dans la table le nombre de concepts et de relations qui existent réellement dans les applications web.
Dans ce contexte, les comparaisons sont effectuées en termes de *Précision* 7.1, *Rappel* 7.2 et *F-mesure* 7.3 [David & Powers 2011], comme indiqué dans les équations suivantes :

$$precision = \frac{Nombre\ de\ concepts\ et\ de\ relations\ pertinents}{Nombre\ de\ tous\ les\ concepts\ et\ relations\ recuperes} \tag{7.1}$$

$$rappel = \frac{Nombre\ de\ concepts\ et\ de\ relations\ pertinents}{Nombre\ de\ concepts\ et\ de\ relations\ existant\ dans\ l'application\ Web} \tag{7.2}$$

$$F - mesure = 2 * \frac{prcision * rappel}{prcision + rappel} \tag{7.3}$$

La Figure 7.6 exprime les résultats expérimentaux du Tableau 7.2 sous forme d'histogrammes pour une meilleure visibilité. Comme on le voit à la Figure 7.6, l'outil de rétro-ingénierie donne une *F-mesure* de moyenne de 0,97 au lieu de 1 en raison du manque de concepts ou de relations dans les fichiers PHP analysés des différentes applications Web.

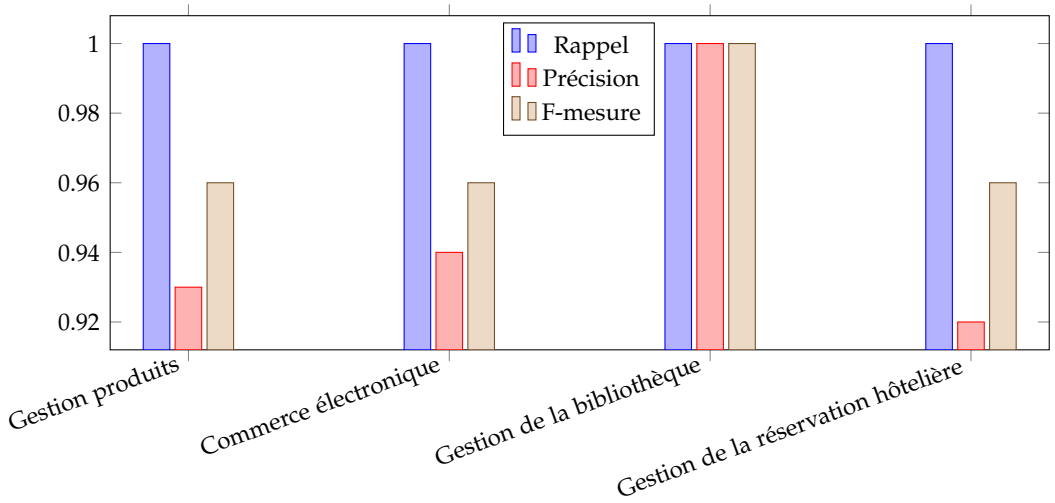

FIGURE 7.6 – *Graphe d'évaluation.*

En outre, il convient de mentionner que notre outil donne de bons résultats, en particulier lors de l'amélioration de la phase de raffinement, ce qui permet d'ajouter de nouvelles relations entre les concepts existants.
Notez que notre objectif principal était de créer une ontologie à partir d'une application Web. Dans le cas de différentes ontologies générées à partir de différentes applications Web, nous devons gérer un problème d'alignement d'ontologies. Ainsi, certaines techniques d'alignement d'ontologies peuvent être utilisées pour garantir l'aspect consensuel de l'ontologie générée globalement.

7.5 Conclusion

Nous avons proposé une approche permet de générer une ontologie depuis une application Web. Elle comprend deux processus à savoir le processus de la rétro-ingénierie et le processus de l'ingénierie directe.

- Le processus de rétro-ingénierie vise à présenter l'application Web à un niveau d'abstraction plus élevé. Ce processus nous permet d'extraire le diagramme de classes en effectuant une analyse du code source de l'application Web. l'analyse est basée sur la technique de correspondance de patterns.
- Le processus d'ingénierie directe représente la génération d'une ontologie OWL à partir des résultats récupérés (diagramme de classes) dans le premier processus. L'opération de génération consiste d'un ensemble de règles de transformation.

Les gains de l'approche proposée se résument en deux points. Le premier point consiste à tirer parti de l'ancienne expérience des applications Web héritées. Le deuxième point consiste à convertir l'application Web en une application Web sémantique. Notez que les instances sont construites en appliquant la première approche (voir chapitre précédent).
L'approche proposée a été concrétisée par un outil appelé *PHP2OWL*. A fin d'évaluer notre travail, nous avons transformé un ensemble d'applications Web vers des ontologies OWL. Les résultats encourageants obtenus lors des expériences menées ont démontré l'efficacité de l'approche proposée.

Conclusion générale et perspectives

8.1 Conclusion générale

Le Web sémantique permet aussi bien aux utilisateurs qu'aux machines (programmes) d'accéder et raisonner sur les données. Par conséquent, les programmes sont devenus capables d'effectuer des opérations bien définies sur les données, et de déduire de nouvelles informations à partir des données existantes. Le Web sémantique offre plusieurs langages pour présenter les connaissances d'une façon sémantique. Parmi ces langages, on cite :

- Le langage RDF pour exprimer les données de manière à les rendre accessibles et les donner un sens. Cependant, pour le Web traditionnel classique, les données se trouvent soit dans des bases de données, soit dans des documents HTML ; dans les deux cas, ils sont moins exploitables par les machines et n'ont pas de sens. Pour les rendre dans un format sémantique, nous devons transformer les applications Web en applications sémantiques.
- Le langage OWL pour exprimer les connaissances sous forme d'une ontologie. Afin de résoudre le problème de transformation des applications Web en applications sémantiques, nous avons proposé deux approches.

La première approche est basée sur l'ingénierie dirigée par les modèles (IDM). L'approche proposée comprend trois étapes : La première est la préparation des données (prétraitement), qui vise à créer un fichier d'entrée conforme au méta-modèle source. La deuxième étape est la transformation, dans laquelle nous lançons le processus de transformation pour obtenir un fichier de sortie RDF conforme à leur méta-modèle cible. La dernière étape est l'affinement du fichier de sortie afin d'obtenir des fichiers RDF bien formés. Les fichiers RDF obtenus sont vérifiés via le validateur du W3C. Notez bien que notre approche est prise en charge par l'outil *HTML2RDF*.
Pour montrer l'efficacité de l'approche proposée, nous avons transformé certaines infoboxes des pages Wikipedia en documents RDF en utilisant l'outil HTML2RDF. Ces documents RDF sont vérifiés via le validateur du W3C, et évalués en utilisant la métrique F-mesure qui donne une moyenne de 0,84, ce qui est un résultat encourageant.
La deuxième approche permet de créer une ontologie OWL à partir d'applications Web. L'approche proposée commence par un processus de rétro-ingénierie qui vise à récupérer le diagramme de classes du code source de l'application Web en utilisant la technique de correspondance de modèle. Ensuite, un processus d'ingénierie directe crée une ontologie OWL à partir du diagramme de classes récupéré en appliquant un ensemble de règles de transformation. Par conséquent, nous avons bénéficié de l'expertise ancienne de l'application Web héritée

et, d'autre part, nous l'avons transformée en application Web sémantique.
L'approche proposée a été implémentée en tant qu'outil *PHP2OWL*. Des expériences ont été effectuées sur un ensemble d'applications Web. Des résultats encourageants ont montré l'efficacité de l'approche proposée.

8.2 Perspectives

Concernant la première approche, nous allons dans un futur travail traiter et transformer les données non structurées, comme les textes, en données liées RDF, en appliquant des techniques de traitement automatique du langage naturel.
Pour ce qui est de la deuxième approche, nous essayerons d'étendre notre outil pour prendre en compte non seulement le code source de l'application Web, mais aussi les bases de données relationnelles, si bien sûr, elles sont disponibles. Nous pensons également à étendre notre approche aux applications basées sur *NoSQL*, notamment les applications *Bigdata*.

Bibliographie

[Abiteboul 1997] S. Abiteboul. *Querying semi-structured data.* In International Conference on Database Theory ICDT 1997 : Database Theory - ICDT '97, volume 1186, pages 1–18. Springer-Verlag., 1997.

[Ardjani *et al.* 2015] Fatima Ardjani, Djelloul Bouchiha et Mimoun Malki. *Ontology-Alignment Techniques : Survey and Analysis*. International Journal Modern Education and Computer Science, vol. 7, no. 11, pages 67–78, 2015.

[Ardjani *et al.* 2017] Fatima Ardjani, Djelloul Bouchiha et Mimoun Malki. *An approach for discovering and maintaining links in rdf linked data*. International Journal of Modern Education and Computer Science(IJMECS), vol. 9, no. 3, pages 56–63, 2017.

[Arenas *et al.* 2012] M. Arenas, A. Bertails, E. Prud'hommeaux et J. Sequeda. *A direct mapping of relational data to rdf*, 2012.

[Astrova & Stantic 2005] Irina Astrova et Bela Stantic. *An HTML Forms Driven Approach to Reverse Engineering of Relational Databases to Ontologies*. In Proceedings of the 23rd IASTED International Conference on Databases and Applications (DBA), pages 246–251, 2005.

[Bechhofer *et al.* 2004] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, L. D. McGuinness, F. P. Schneider et L. A. Stein. *OWL Web Ontology Language Reference W3C Recommendation*, 2004.

[Benslimane *et al.* 2008] Sidi Benslimane, Mimoun Malki, Mustapha Rahmouni et Adellatif Rahmoun. *Towards Ontology Extraction from Data-Intensive Web Sites : An HTML Forms-Based Reverse Engineering Approach*. The International Arab Journal of Information Technology, vol. 5, no. 1, pages 34–44, 2008.

[Berners-Lee *et al.* 1999] T. Berners-Lee, M. Fischetti et T.M. Dertouzos. Weaving the web : The original design and ultimate destiny of the world wide web by its inventor,. 1999.

[Berners-Lee *et al.* 2001] T. Berners-Lee, J. Hendler et O. Lassila. *The semantic web*. Scientific American, vol. 284, no. 5, pages 34–43, 2001.

[Berners-Lee 2006] Tim Berners-Lee. *Linked data : design issues*, 2006.

[Boley *et al.* 2001] H. Boley, S. Tabet et G. Wagner. *Design rationale of RuleML : A markup language for Semantic Web rules*. In Proceedings of the 1st Semantic Web Working Symposium, pages 381–401, 2001.

[Booch & Rumbaugh 1995] G. Booch et J. Rumbaugh. *Unified Method, technical report, version 0.8*. Report, 1995.

[Booch *et al.* 1996] G. Booch, J. Rumbaugh et I. Jacobson. *The Unified Modeling Language for Object-Oriented Development, technical report, version 0.9*. Report, 1996.

[Booch *et al.* 1997] G. Booch, J. Rumbaugh et I. Jacobson. *Unified Modeling Language, technical report, version 1.0*. Report, 1997.

[Borst 1997] Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. Thesis, 1997.

[Bouougada *et al.* 2015] Benamar Bouougada, Djelloul Bouchiha et Mimoun Malki. *A Framework for Reengineering Web applications into Linked Data based on MDA*. In Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication IPAC '15, pages 1–5. ACM New York, NY, USA, 2015.

[Bouougada *et al.* 2017] Benamar Bouougada, Djelloul Bouchiha, Abdelghani Bouziane et Mimoun Malki. *Ontology Authoring and Linked Data Generation from Web Applications*. International Journal of Strategic Information Technology and Applications, vol. 8, no. 4, pages 52–66, 2017.

[Bouougada *et al.* 2018] Benamar Bouougada, Djelloul Bouchiha, Yassine Ouhammou et Mimoun Malki. *Re-engineering Web Application towards Linked Data : a Model-Based Approach*. International Arab Journal of e-Technology, vol. 5, no. 2, pages 58–69, 2018.

[Bray *et al.* 1997] T. Bray, J. Paoli et C.M. Sperberg-McQueen. *Extensible markup language (xml).* World Wide Web Journal, vol. 2, no. 4, pages 27–66, 1997.

[Bray *et al.* 2006] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, FranÃ§ois Yergeau et John Cowan. *Extensible Markup Language (XML) 1.1 W3C Recommendation*, 2006.

[Cafarella *et al.* 2008] M.J. Cafarella, A.Y. Halevy, D.Z. Wang, E. Wu et Y. Zhang. *Web tables : exploring the power of tables on the Web*. volume 1, pages 538–549, 2008.

[Chikofsky & Cross 1990] E.J. Chikofsky et J.H. Cross. *Reverse engineering and design recovery : a taxonomy*. IEEE Xplore Digital Library, vol. 7, no. 1, pages 13–17, 1990.

[Corcho *et al.* 2002] O. Corcho, M. Fernandez-Lopez et A. Gomez-Pérez. *Methodologies, tools and languages for building ontologies. Where is their meeting point ?* Data and Knowledge Engineering, vol. 46, no. 1, pages 41–64, 2002.

[Cost *et al.* 2002] R. Scott Cost, T. Finin, A. Joshi, Y. Peng, C. Nicholas, I. Soboroff, H. Chen, L. Kagal, F. Perich, Y. Zou et S.Tolia. *ITtalks : A case study in the Semantic Web and DAML+OIL.* IEEE Intelligent Systems, vol. 17, no. 1, pages 40–47, 2002.

[Dan & Guha 2014] B. Dan et R.V. Guha. *RDF Schema 1.1 W3C Recommendation*, 2014.

[David & Powers 2011] M. David et W Powers. *Evaluation : From precision, recall and f-factor to roc informedness, markedness and correlation*. Journal of Machine Learning Technologies, vol. 2, no. 1, pages 37–63, 2011.

[David *et al.* 2014] B. David, Tim Berners-Lee, E. P. hommeaux et C. Gavin. *RDF 1.1 Turtle Terse RDF Triple Language*, 2014.

[David 2014] B. David. *RDF 1.1 N-Triples A line-based syntax for an RDF graph*, 2014.

[DavidL 2018] DavidL. *Le langage HTML*, 2018.

[Deborah & v. Harmelen 2004] L. Deborah et F. v. Harmelen. *OWL Web Ontology Language -Overview*, 2004.

[Denaux *et al.* 2012] R Denaux, D Thakker, V Dimitrova et A G Cohn. *Interactive semantic feedback for intuitive ontology authoring*. In Proceedings of the 7th International Conference on Formal Ontology in Information Systems(FOIS 2012), pages 160–173. IOS Press, 2012.

[Denaux *et al.* 2013] R. Denaux, V. Dimitrova et A. G. Cohn. *Interacting with ontologies and linked data through controlled natural languages and dialogues*. In Enabling Domain Experts to Use Formalised Reasoning - AISB Convention 2013., pages 18–20. Society for the study of artificial intelligence, 2013.

[Dennis 2015] W. Dennis. *ATL/User Guide-Introduction*, 2015.

[Devedzic 2004] V. Devedzic. *Education and the semantic web.* International Journal of Artificial Intelligence in Education, vol. 14, no. 39-65, 2004.

[Dimitrova *et al.* 2008] V. Dimitrova, R. Denaux, G. Hart, C. Dolbear, I. Holt et A. G. Cohn. *Involving domain experts in authoring OWL ontologies*. In Proceedings of the 7th International Semantic Web Conference (ISWC 2008), volume 5318, pages 1–16. Springer, Berlin, Heidelberg, 2008.

[Dragan *et al.* 2009] G. Dragan, D. Dragan et D. Vladan. Model driven engineering and ontology development. Springer, Dordrecht Heidelberg London New York, 2009.

[Embley *et al.* 2011] D.W. Embley, M. Krishnamoorthy, G. Nagy et S. Seth. *Factoring Web tables*. volume 6703, pages 253–263, 2011.

[Fabien & Guus 2014] G. Fabien et S. Guus. *RDF 1.1 XML Syntax*, 2014.

[Favre & Nguyen 2005] Jean Marie Favre et Tam Nguyen. *Towards a Megamodel to Model Software Evolution Through Transformations .* Electr. Notes Theor. Comput. Sci, vol. 127, no. 3, pages 59 – 74, 2005.

[Favre 2010] L. Favre. Model driven architecture for reverse engineering technologies : Strategic directions and system evolution. Engineering Science Reference (an imprint of IGI Global), United States of America, 2010.

[Fraternali 1999] P. Fraternali. *Tools and Approaches for Developing Data-Intensive Web Applications : A Survey.* ACM Computing Surveys (CSUR)., vol. 31, no. 3, pages 227–263, 1999.

[Garzotto *et al.* 1995] F. Garzotto, L. Mainetti et P. Paolini. *Hypermedia design, analysis, and evaluation issues.* Communications of the ACM, vol. 38, no. 8, pages 74–86, 1995.

[Graham & Jeremy 2004] K. Graham et J. Jeremy. *Resource Description Framework (RDF) : Concepts and Abstract Syntax W3C Recommendation*, 2004.

[Gruber 1993] Thomas R Gruber. *A Translation Approach to Portable Ontologies Specifications*. Knowledge Acquisition, vol. 5, no. 2, pages 199–220, 1993.

[Guarino *et al.* 2009] Nicola Guarino, Daniel Oberle et Steffen Staab. What is an ontology ?, pages 1–17. International Handbooks on Information Systems. Springer Publishing Company Incorporated, Verlag Berlin Heidelberg, 2 édition, 2009.

[Heath & Bizer 2011] Tom Heath et Christian Bizer. Linked data : Evolving the web into a global data space (1st edition)., volume 1 of *Synthesis Lectures on the Semantic Web : Theory and Technology*. 2011.

[Heflin & Hendler 2001] J. Heflin et J. Hendler. *A portrait of the Semantic Web in action.* IEEE Intelligent Systems, vol. 16, no. 2, pages 54–59, 2001.

[Hendler 2001] J. Hendler. *Agents and the semantic web*. IEEE Intelligent Systems, vol. 16, no. 2, pages 30–37, 2001.

[Herman 2008] Ivan Herman. *Semantic Web Activity Statement*, 2008.

[Jouault 2008] F. Jouault. *Atl : A model transformation too*. Science of Computer Programming, vol. 72, no. 1-2, pages 31–39, 2008.

[Kaljurand 2007] Kaarel Kaljurand. *Attempto Controlled English as a Semantic Web Language*. Thesis, 2007.

[Kaljurand 2008] Kaarel Kaljurand. *ACE view-An ontology and rule editor based on attempto controlled English*. In Proceedings of the 5th OWL Experiences and Directions Workshop (OWLED 2008), pages 1–12, 2008.

[Kent 2002] Stuart Kent. *Model Driven Engineering*. In Proceedings of the Third International Conference on Integrated Formal Methods (IFM 2002), volume 2335, pages 286 – 298. Springer, 2002.

[Kleppe *et al.* 2002] A. G. Kleppe, J. B. Warmer et W. Bast. Mda explained, the model driven architecture : Practice and promise. Boston, MA, USA, 2002.

[Konstantinou *et al.* 2014] N. Konstantinou, D. Spanos, N. Houssos et N. Mitrou. *Exposing scholarly information as linked open data : Rdfizing dspace contents*. The Electronic Library, vol. 32, no. 6, pages 834–851, 2014.

[Kuhn 2009] Tobias Kuhn. *AceWiki : A natural and expressive semantic Wiki*. In Proceedings of Semantic Web User Interaction at CHI 2008 : Exploring HCI Challenges, CEUR Workshop Proceedings, volume 534, pages 1–8. arXiv, 2009.

[Kuhn 2013a] Tobias Kuhn. *A principled approach to grammars for controlled natural languages and predictive editors*. Journal of Logic, Language and Information, vol. 22, no. 1, pages 33–70, 2013.

[Kuhn 2013b] Tobias Kuhn. *The understandability of OWL statements in controlled English*. Journal of Semantic Web - Linked Data for science and education, vol. 4, no. 1, pages 101–115, 2013.

[Lassila 1998] O. Lassila. *Web metadata : A matter of semantics*. IEEE Internet Computing, vol. 2, no. 4, pages 30–70, 1998.

[Lehmann *et al.* 2014] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer et C. Bizer. *Dbpedia a large scale, multilingual knowledge base extracted from Wikipedia*. Semantic Web Journal, vol. 6, no. 2, pages 167–195, 2014.

[Luc 2004] Van Lancker Luc. *PHP-MYSQL*, 2004.

[McIlraith *et al.* 2001] S.A. McIlraith, T.C. Son et H. Zeng. *(2001), Semantic Web services*. IEEE Intelligent Systems, vol. 16, no. 2, pages 46–53, 2001.

[Miller & Mukerji 2003] J. Miller et J. Mukerji. *MDA Guide Version 1.0.1*. Report, 2003.

[Mulwad *et al.* 2010] V. Mulwad, T. Finin, Z. Syed et A. Joshi. *T2LD : Interpreting and Representing Tables as Linked Data*. volume 658, pages 25–28, 2010.

[Mulwad *et al.* 2013] V. Mulwad, T. Finin et A. Joshi. *Semantic Message Passing for Generating Linked Data from Tables*. volume 8218, pages 363–378. Springer, 2013.

[Munoz *et al.* 2014] E. Munoz, A. Hogan et A. Mileo. *Using linked data to mine rdf from wikipedia tables*. In proceedings of the 7th ACM international conference on Web search and data mining (WSDM 14), pages 533–542, 2014.

[Nagy *et al.* 2011] G. Nagy, D. W. Embley, S. Machado, S. Seth, D. Jin et M. Krishnamoorthy. *Data extraction from Web tables : the devil is in the details*. In ICDAR '11 Proceedings of the 2011 International Conference on Document Analysis and Recognition, pages 242–246. IEEE, 2011.

[Noy *et al.* 2001] N.F. Noy, M. Sintek, S. Decker, M. Crubézy, R.W. Fergerson et M.A. Musen. *Creating Semantic Web contents with Protégé-2000*. IEEE Intelligent Systems, vol. 16, no. 2, pages 60–71, 2001.

[OMG 1997] OMG. *UML Summary, Semantics and Notation Guide, version 1.1*. Report, 1997.

[OMG 2003a] OMG. *MDA Guide Version 1.0.1*. Rapport technique Document omg/2003-06-01, 2003.

[OMG 2003b] OMG. *OMG Unified Modeling Language Specification, version 1.5 (2003)*. Rapport technique, 2003.

[OMG 2003c] OMG. *UML 2.0 Infrastructure Specification*. Rapport technique, 2003.

[OMG 2003d] OMG. *UML 2.0 Superstructure Specification*. Rapport technique, 2003.

[OMG 2004] OMG. *UML 2.0 Superstructure Specification*. Rapport technique formal/05-07-04, 2004.

[OMG 2007] OMG. *MOF 2.0 / XMI Mapping, Version 2.1.1*. Rapport technique formal/2007-12-01, 2007.

[OMG 2012] OMG. *Common Object Request Broker Architecture (CORBA) Specification, Version 3.3*. Rapport technique formal/2012-11-12, 2012.

[OMG 2016] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification*. Rapport technique formal/2016-06-03, 2016.

[OMG 2017] OMG. *Unified Modeling Language*. Report, 2017.

[Ougouti *et al.* 2015] Naima S. Ougouti, H. Belbachir et Y. Amghar. *A new owl2 based approach for relational database description*. International Journal of Information Technology and Computer Science, vol. 7, no. 1, pages 48–53, 2015.

[OWL-Working-Group 2012] OWL-Working-Group. *Web Ontology Language (OWL)*, 2012.

[Richard *et al.* 2014] C. Richard, W. David et L. Markus. *RDF 1.1 Concepts and Abstract Syntax W3C Recommendation 25 February 2014*, 2014.

[Sano 1996] D. Sano. Designing large-scale web sites : a visual design methodology. John Wiley & Sons, Inc, New York, NY, USA., 1996.

[Scharffe *et al.* 2012] F. Scharffe, G. Atemezing, R. Troncy, F. Gandon, S. Villata, B. Bucher, F. Hamdi, L. Bihanic, G. Képéklian, F. Cotton, J. Euzenat, Z. Fan, P. Vandenbussche et B. Vatant. *Enabling linked data publication with the datalift platform*. In proceedings of AAAI workshop on semantic cities, pages 25–30, 2012.

[Schmidt 2006] D C Schmidt. *Guest Editor's Introduction : Model - Driven Engineering*. Computer, vol. 39, no. 2, pages 25–31, 2006.

[Sequeda *et al.* 2012] F Juan Sequeda, Marcelo Arenas et P Daniel Miranker. *On Directly Mapping Relational Databases to RDF and OWL*. In Proceeding WWW '12 Proceedings of the 21st international conference on World Wide Web, pages 649–658. ACM New York, NY, USA, 2012.

[Shudi *et al.* 2012] Shudi, C. M. Sperberg-McQueen, S. Henry, S. Henry, Noah Mendelsohn, David Beech et Murray Maloney. *W3C XML Schema Definition Language (XSD) 1.1 Part 1 : Structures*, 2012.

[Steinberg *et al.* 2003] D. Steinberg, F. Budinsky, M. Paternostro et E.Merks. Eclipse modeling framework : A developer's guid. The Eclipse Series. Addison-Wesley Professional, 2003.

[Steinberg *et al.* 2009] Dave Steinberg, Frank Budinsky et Marcelo Paternostro. Eclipse modeling framework (emf). Addison-Wesley Professional, 2009.

[Studer *et al.* 1998] R. Studer, R. Benjamins et D. Fensel. *Knowledge engineering : Principles and methods.* Data & Knowledge Engineering, vol. 25, no. 1-2, pages 161–198, 1998.

[Trias *et al.* 2015] F. Trias, Valeria de Castro, M. Lopez-Sanzand et E. Marcos. *Migrating traditional Web applications to cms-based Web applications*. Electronic Notes in Theoretical Computer Science, vol. vol 314, pages 23–44, 2015.

[Véronneau 2016] Martin Véronneau. *L'histoire du développemnt d'applications Web*, 2016.

[W3Schools 2018] W3Schools. *HTML Introduction*, 16/12/2018.

[William 2014] P. William. *ATL/User Guide - Overview of the ATL Language*, 2014.

[Xavier 2005] B. Xavier. Mda en action ingénierie logicielle guidée par les modÃ¨les. EDITIONS EYROLLES, 61, bd Saint-Germain, 75240 Paris Cedex 05, 2005.

[Yeeply 2018] Yeeply. *Types de développement d'applications web*, 2018.

العنوان : إعادة تصميم تطبيقات الويب للبيانات المرتبطة.

ملخص : يخزن الويب الحالي كمية كبيرة من المعلومات غالبًا ما تكون مخصصة للاستخدام العام. تطور الويب مع ظهور تقنيات جديدة ، مثل الويب الدلالي، خدمات الويب، وتطبيقات الويب. تسمح هذه التقنيات للمستخدمين بالمعالجة غير المباشرة للبيانات المتوفرة بأشكال متعددة. في الوقت نفسه، تتزايد أيضًا الرغبة في الوصول إلى البيانات مباشرة على الويب. فقط البيانات المرتبطة، التي تعتبر تقنية من تقنيات الويب الدلالي، يمكنها تلبية هذه الرغبة. هنالك عدد كبير من تطبيقات الويب قيد التشغيل حاليًا، مما يجعل من الممكن التعامل مع كمية كبيرة من قواعد البيانات. للسماح بالوصول المباشر إلى هذه البيانات على الويب دون إنشائها مرة أخرى كبيانات مرتبطة، نقترح في هذه الأطروحة مقاربتين لتحويل تطبيقات الويب إلى بيانات مرتبطة وذالك باستخدام هندسة مبنية على النماذج. النهجين المقترحين كلا بتطبيقين هما HTML2RDF و PHP2OWL. تم تجربة التطبيقين على تطبيقات ويب مختلفة، والنتائج المتحصل عليها كانت جد مشجعة و أظهرت فعالية النهجين المقترحين.

الكلمات المفتاحية : البيانات المرتبطة RDF، هندسة مبنية على النماذج (MDE) ، إعادة الهندسة، تطبيقات الويب، الويب الدلالي، هندسة الانطولوجيا، OWL.

Titre : La Réingénierie des applications Web vers les données liées.

Résumé : Le Web actuel stocke une grande quantité d'informations, souvent destinées à des usages publics. Le Web se développe de plus en plus avec l'émergence de nouvelles technologies, telles que le Web sémantique, les services Web, les applications mobiles, etc. Ces techniques permettent aux utilisateurs de manipuler les données sous plusieurs formats, et d'une manière indirecte. Parallèlement, le désir d'accéder directement aux données sur le Web augmente. Seules les Données Liées RDF, qui sont une technologie du Web sémantique, peuvent répondre à ce besoin. Un nombre important d'applications Web sont actuellement opérationnelles, permettant ainsi de manipuler une grande quantité de bases de données. Pour permettre un accès direct à ces données sur le Web sans les créer à nouveau sous forme de Données Liées, nous proposons dans cette thèse deux approches permettant de transformer les applications Web en données liées et ontologies en se basant sur l'ingénierie dirigée par les modèles. Les approches proposées sont implémentées par deux outils : HTML2RDF et PHP2OWL. Elles ont été évaluées sur différentes applications Web. Les résultats obtenus ont été encourageants et ont montré l'efficacité des approches proposées.

Mots-clés : Données liées RDF, Ingénierie dirigée par les modèles (IDM), Réingénierie, Application Web, Web Sémantique, Ingénierie des ontologies, OWL.

Title: Reverse engineering of Web applications towards linked data

Abstract: The current Web stores a large amount of information often intended for public use. The Web is developing with the emergence of new technologies, such as semantic Web, Web services, Ubiquitous Computing, etc. These techniques manipulate data in multiple formats in a hidden or unusable way for users. At the same time, the desire to access data directly on the Web is also increasing. Only Linked Data, which is Semantic Web technology, can meet this need. A large number of Web applications are currently operational, making it possible to handle a large amount of databases. To allow direct access to this data on the Web without creating it from scratch as Linked Data, we propose in this dissertation two approaches that transform Web applications into Linked Data and ontology based on model-driven engineering. The proposed approaches are concerted by HTML2RDF and PHP2OWL tools. They are evaluated on different Web applications. The obtained results were encouraging and shown the effectiveness of the proposed approaches.

Keywords: RDF Linked Data, Model-Driven Engineering (MDE), Reengineering, Web Application, Semantic Web, Ontology Engineering, OWL.