UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MODULARISATION DES ONTOLOGIES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

PATRICK TCHOUANTO POOSIA

JUILLET 2014

REMERCIEMENTS

En préambule à ce mémoire, je souhaite adresser mes remerciements les plus sincères aux personnes qui m'ont apporté leur aide et qui ont contribué à l'élaboration de ce mémoire.

Je tiens à remercier sincèrement Monsieur Roger Nkambou qui, en tant que Directeur de mémoire, s'est toujours montré à l'écoute et disponible tout au long de la réalisation de ce mémoire, ainsi que pour l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Je remercie également Monsieur Petko Valtchev, le responsable du laboratoire GDAC, pour sa disponibilité et ses recommandations.

Je tiens aussi à remercier Amine Rouane-Hacène qui a su mettre à ma disposition les outils nécessaires à l'élaboration de COMET.

Mes remerciements s'adressent aussi à Madame Cynthia Lisée, pour avoir eu la gentillesse de lire et réviser la bibliographie et la qualité de la langue de ce mémoire.

Enfin, j'adresse mes plus sincères remerciements à ma mère, mon frère, ma famille et tous mes proches et amis, qui m'ont aidé et toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

TABLE DES MATIÈRES

LISTE DES FIGURES ix
LISTE DES TABLEAUX xi
LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES xiii
RÉSUMÉ xv
INTRODUCTION 1
0.1 Contexte et problématique 2
0.2 Objectifs de la recherche 5
0.3 Organisation du mémoire 6
CHAPITRE I
ÉTAT DE L'ART 7
1.1 La notion d'ontologie 7
1.1.1 L'ontologie : tentatives de définition 7
1.1.2 Les composantes d'une ontologie 9
1.1.3 Les types d'ontologies 10
1.1.4 Les rôles des ontologies 11
1.2 Le but de la modularisation des ontologies 12
1.3 Les principales approches de modularisation des ontologies 14
1.3.1 Les techniques de partitionnement des ontologies 15
1.3.2 Les techniques d'extraction de module 20
1.4 Avantages et limites des approches structurelle et sémantique 36
CHAPITRE II
APPROCHE DE MODULARISATION COMET 39
2.1 Algorithme de segmentation 39
2.2 Algorithme d'extraction 44

2.3 Implémentation du prototype COMET 45
2.3.1 Architecture de COMET 46
2.3.2 Les technologies et les langages utilisés 48
2.3.3 Présentation de cas de modularisation avec COMET 53
CHAPITRE III
LES DIFFÉRENTES APPROCHES D'ÉVALUATION DES ONTOLOGIES 59
3.1 Approches basées sur la qualité de l'ontologie 59
3.2 Méthodes quantitatives d'évaluation des ontologies 61
3.3 Approches basées sur la comparaison 62
3.3.1 Approche basée sur la comparaison par rapport à une référence 62
3.3.2 Approche basée sur la comparaison par rapport à d'autres ontologies 65
3.4 Approches basées sur l'utilité d'une ontologie 66
CHAPITRE IV
EXPÉRIMENTATIONS ET ÉVALUATION DE COMET 71
4.1 Critères d'évaluation d'un module 71
4.2 Protocole de validation 73
4.3 Analyse des résultats 77
4.3.1 Évaluation des modules en fonction des métriques d'Alani *et al.* 77
4.3.2 Évaluation des modules en fonction des métriques d'Orme *et al.* 85
4.3.3 Évaluation des modules en fonction des métriques de Staab *et al.* 88
CONCLUSION 91
APPENDICE A
OUTILS D'EXTRACTION DE MODULES 95
APPENDICE B
RÉSULTATS DE L'ÉVALUATION DES MODULES 99
APPENDICE C
SCHÉMAS DE MODULES GÉNÉRÉS PAR COMET 105
APPENDICE D

EXEMPLE DE MODULE GÉNÉRÉ PAR COMET EN OWL 109
RÉFÉRENCES 117

LISTE DES FIGURES

Figure		Page
1.1	Différents types d'ontologies. Les flèches représentent des relations de spécialisation (Guarino, 1998).	11
1.2	Les principaux rôles des ontologies.	12
1.3	Un exemple de graphe avec des dépendances proportionnelles de force (Stuckenschmidt et Schlicht, 2009).	16
1.4	Exemple de graphe pour l'attribution des nœuds restant à des modules (Stuckenschmidt et Schlicht, 2009).	17
1.5	Le processus de sélection de connaissances et son utilisation avec *Magpie* (d'Aquin *et al.*, 2006).	22
1.6	Interface utilisateur de PROMPT pour la spécification de la vue de parcours (Noy et Musen, 2004).	26
1.7	Parcours de la hiérarchie des classes à travers les liens (Seidenberg et Rector, 2005).	28
1.8	Extraction de segment avec profondeur de 2 (Seidenberg et Rector, 2006).	30
1.9	Le framework SOMET (Doran *et al.*, 2008).	35
2.1	Processus de modularisation avec COMET.	46
2.2	Diagramme de classes de COMET.	49
2.3	Processus de modularisation de l'ontologie `camera.owl` avec COMET.	54
2.4	Schéma de l'ontologie modularisée `camera.owl`.	55
2.5	Schéma du module extrait `module_camera.owl`.	55
2.6	Schéma de l'ontologie modularisée `conference.owl`.	57
2.7	Schéma du module extrait `module_conference.owl`.	58

4.1 Score des modules en fonction des métriques d'Alani *et al.* 81

4.2 Score total des modules en fonction de la variation des poids. . . . 84

4.3 Score des modules en fonction des métriques d'Orme *et al.* 87

4.4 Score du Rappel lexical et de la F'-mesure taxonomique des modules extraits. 90

A.1 Interface Web de l'outil OWL-ME. 95

A.2 Module extrait de l'ontologie `conference.owl` à l'aide de OWL-ME. 96

A.3 Processus de modularisation à l'aide de SegmentationApp. 97

A.4 Module extrait de l'ontologie `conference.owl` à l'aide de SegmentationApp. 98

C.1 Module `comet_sweto.owl` extrait de l'ontologie `sweto_simplified.owl` avec un seuil de 0,1 et une profondeur hiérarchique de 1. 105

C.2 Module `comet_sigkdd.owl` extrait de l'ontologie `sigkdd.owl` avec un seuil de 0,2 et une profondeur hiérarchique de 1. 106

C.3 Module `comet_edas.owl` extrait de l'ontologie `edas.owl` avec un seuil de 0,2 et une profondeur hiérarchique de 1. 107

LISTE DES TABLEAUX

Tableau Page

4.1 Liste d'ontologies à modulariser à l'aide des différents outils d'extraction. 75

4.2 Extraction de modules à l'aide des approches de Seidenberg *et al.*, de Cuenca *et al.* et COMET. 76

B.1 Évaluation des modules en fonction des métriques d'Alani *et al.* . 100

B.2 Calcul du score des modules en fonction des poids des métriques. 101

B.3 Évaluation des modules en fonction des métriques d'Orme *et al.* . 102

B.4 Évaluation des modules en fonction des métriques de Staab *et al.* 103

LISTES DES ABRÉVIATIONS, SIGLES ET ACRONYMES

W3C	World Wide Web Consortium
OIL	Ontology Interference Layer
XML	Extensible Markup Language
GPL	General Public License
JRE	Java Runtime Environment
API	Application Programming Interface
URI	Uniform Resource Identifier
JAR	Java ARchive
OWL	Ontology Web Language
JUNG	Java Universal Network/Graph Framework
RDF(S)	Resource Description Framework (Schema)
DAML	DARPA Agent Markup Language
COMET	Compact Ontology Module Extraction Tool
OWL-ME	OWL Module Extractor
OWL-DL	Ontology Web Language Description Logics
SPARQL	SPARQL Protocol and RDF Query Language

RÉSUMÉ

De nos jours, on assiste à un développement incessant des technologies de l'information. Ces nouvelles technologies ont permis l'élaboration de systèmes intelligents à base de connaissances. Ces systèmes sont principalement caractérisés par le fait qu'ils sont aptes à représenter la connaissance sous une forme interprétable et manipulable par des machines. Toutefois, cette amélioration devient envisageable dès lors que les ontologies du domaine sont intégrées dans les systèmes d'information. Cette approche est justifiée par le fait que les ontologies sont considérées comme étant des structures efficaces pour la representation des connaissances du domaine. Aussi, remarque-t-on que la conception des ontologies est une tâche non triviale qui peut tout simplement être ramenée à la réutilisation d'une ou plusieurs ontologies déjà existantes. Cependant, dans la mesure où un expert en ingénierie de connaissances aurait seulement besoin d'une partie de l'ontologie afin de réaliser une tâche précise, l'obtention de cette partition représentera dès lors un but de la modularisation des ontologies.

Dans ce travail, les différentes approches et outils de modularisation existants sont présentés. Le processus de modularisation des ontologies se fait suivant deux principales approches, à savoir le partitionnement de l'ontologie et l'extraction du module. Ce mémoire traite de l'élaboration de l'outil d'extraction COMET qui est basé sur une approche de modularisation hybride dans le but de tirer avantage des techniques structurelles et sémantiques existantes.

Un protocole d'évaluation basé sur l'utilisation des métriques de qualité est mis sur pied. Par ailleurs, une étude détaillée des résultats de l'évaluation de modules générés à partir de différents outils de modularisation est réalisée dans l'optique de démontrer que les performances de COMET sont supérieures à celles des outils existants.

Mots clés : ontologies, modularisation, métriques, évaluation des ontologies, critères de qualité, conceptualisation, modélisation des connaissances, OWL.

INTRODUCTION

Le développement des technologies du Web ont entrainé un intérêt croissant pour la recherche sur le partage et l'intégration des connaissances dans un environnement distribué. Le Web sémantique tend non seulement à rendre l'information accessible mais aussi lisible et utilisable par des applications de traitement de données. Celui-ci offre des outils permettant de mieux organiser l'information et ce, en extrayant, partageant et réutilisant les connaissances spécifiques à un domaine. Dans cette optique de partage et d'interopérabilité, la prolifération et la disponibilité des ontologies sont cruciales. Il va sans dire que les ontologies sont devenues un modèle incontournable pour la représentation et le raisonnement sur des connaissances du domaine. Bien qu'essentielles à la gestion des systèmes à base de connaissances, il n'en demeure pas moins que la conception, la réutilisation et l'intégration des ontologies restent des tâches complexes.

Dans la situation actuelle, on se rend assez vite compte que la modularisation serait une approche efficace qui permettrait d'apporter des réponses à bons nombres de problèmes du génie de connaissances. Nous aborderons la modularisation dans le sens d'un processus au cours duquel on identifie un ensemble de concepts pertinents et spécifiques à un domaine donné. Le module qui est aussi une ontologie sera uniquement composé de ces concepts. Toute ontologie se doit de satisfaire un certain nombre de critères afin de s'assurer qu'elle décrive de façon précise et complète les connaissances du domaine qu'on entend modéliser. En d'autres termes, on ne saurait parler de modularisation sans aborder la question de l'évaluation des ontologies. De ce fait, la mise en place d'un protocole de

validation devient un impératif dès lors qu'il est essentiel d'évaluer la qualité du module extrait.

0.1 Contexte et problématique

Les systèmes informatisés sont devenus dans la société actuelle des outils indispensables. On s'attend à ce que ceux-ci accomplissent des tâches de plus en plus diverses et complexes. Cette complexité même requiert des systèmes qu'ils soient plus «intelligents». Tout comme l'interaction des hommes avec des machines est devenu phénomène normal de nos jours, on constate également que l'interaction des systèmes informatisés entre eux l'est tout autant. C'est le cas des ordinateurs branchés en réseau via le Net qui interagissent ensemble dans le but de réaliser une multitude de tâches.

On a tous d'une façon ou d'une autre vécu l'expérience de l'utilisation d'Internet et de son vaste contenu. Clifford Stoll définit l'Internet comme étant un grand océan de données non-éditées et sans aucun semblant de complétude (Stoll, 1995). On réalise donc assez tôt qu'il est difficile d'y trouver ce dont on a besoin. Par exemple, faire une requête avec pour seul terme «jaguar» nous donnera des résultats à propos de la «jaguar» comme modèle de voiture et aussi à propos du «jaguar» comme étant un félin (animal). De cette confusion de sens dans l'interprétation des termes par les systèmes naît donc la nécessité d'évoluer vers un Web plus «intelligent» qui a une connaissance complète de ce que chaque objet représente. Ceci requiert la transition de la notion «donnée» à la notion «connaissance». Si nous voulons être à même de pouvoir séparer le concept voiture du concept félin une fois le résultat de la requête «jaguar» obtenu, il faudrait avant tout que le moteur de recherche utilisé soit capable de les distinguer.

Le passage des données à la connaissance représente une nouvelle perspec-

tive dans le domaine de l'informatique. Afin d'encoder cette connaissance dans l'optique de la rendre lisible et utilisable par les machines, on a recourt aux ontologies. Ces dernières permettent de conceptualiser les connaissances du domaine à l'aide d'un vocabulaire de termes, et ce tout en exprimant de façon explicite les différentes relations entre eux. Les ontologies doivent leur succès actuel à leur aptitude à être partageables et réutilisables. Quoiqu'avec le partage, on se retrouve avec des ontologies de plus en plus larges. La construction d'une ontologie à partir de zéro est une tâche complexe qui nécessite du temps. Ainsi, dans le but de simplifier le processus de conception, on tablera sur le principe que l'on puisse construire une ontologie à partir d'une autre déjà existante. Ceci nous amène donc à envisager la modularisation comme étant une approche qui apporterait des solutions non seulement au problème de conception, mais aussi à celui du partage et de l'intégration des ontologies.

La notion de modularisation est basée sur le principe de «diviser pour mieux régner» généralement appliqué dans le génie logiciel où il est question de développer une application dont la structure repose essentiellement sur des composants autonomes facilement concevables et réutilisables. Le module représenterait donc ainsi un composant du logiciel qui réalise une tâche précise et qui interagit avec d'autres. Dans le génie ontologique, la modularisation pourrait être perçue de deux façons. Premièrement, elle pourrait être vue comme un processus menant à la décomposition d'une ontologie large en des modules ontologiques de petite taille. Deuxièmement, elle pourrait également être perçue comme une étape du processus de construction de l'ontologie qui se réalise à travers la conception d'un ensemble de modules ontologiques indépendants les uns des autres. Modulariser revient donc à sélectionner les concepts et relations pertinents en fonction de la tâche et de l'application pour lesquelles on décide de modéliser une ontologie. Pour ce faire, on peut soit :

- Réutiliser toute l'ontologie en entier. Tous les concepts et relations de l'ontologie du domaine seront inclus dans l'ontologie qu'on tend à concevoir. Ceci entraîne un alourdissement de l'ontologie de l'application qui sera d'autant plus grande si l'ontologie du domaine est large. On se retrouvera dans ce cas avec des concepts et définitions qui ne sont pas forcément pertinents pour la tâche à réaliser.
- Construire une nouvelle ontologie. Ce qui représenterait un travail fastidieux. La conception d'une ontologie demeure une tâche complexe dans la mesure où il revient à l'expert de définir tous les concepts et relations nécessaires.
- Réutiliser une partie de l'ontologie initiale. C'est un compromis entre les deux alternatives précédentes. Dans ce cas, l'ontologie de l'application représenterait donc une partie ou un module de l'ontologie de domaine.

Dans le cadre de la production d'ontologies de meilleure qualité, le laboratoire GDAC a mis sur pied une plateforme de service de génie ontologique *INUKHUK* qui permet d'effectuer les opérations telles que le filtrage, la fusion, l'extraction de modules ontologiques, la restructuration et l'analyse de la qualité des ontologies. Le sujet de ce mémoire s'inscrit dans le cadre de l'élaboration d'un outil d'extraction de modules ontologiques, car la mise en place de ce celui-ci faciliterait largement la conception d'ontologies d'application. Ainsi, l'on serait capable de produire un module qui soit uniquement focalisé sur le domaine qui nous intéresse. Le but de cette initiative est de pouvoir disposer d'une partie très spécifique de l'ontologie du domaine suffisante pour l'usage entrepris. Un tel module se doit d'être calculé et extrait d'une ou plusieurs ontologies. Il va de soi que cette tâche n'est pas intuitive.

0.2 Objectifs de la recherche

L'objectif de notre travail est tout d'abord de proposer une approche de modularisation des ontologies. Puis, sur la base de cette dernière, il sera question de concevoir un outil capable d'extraire des modules ontologiques. Ce projet sera réalisé en trois étapes à savoir : l'élaboration d'un algorithme de segmentation, l'élaboration d'un algorithme d'extraction et la mise en place d'un protocole de validation.

L'algorithme de segmentation aura pour but de parcourir la structure hiérarchique et les relations sémantiques de l'ontologie choisie afin de repérer les concepts pertinents en fonction d'une liste de termes préalablement définie par l'utilisateur. La pertinence d'un concept dépend de deux paramètres essentiels : le seuil sémantique et la profondeur hiérarchique. Le second algorithme qui est celui de l'extraction a pour but de générer un module ontologique composé uniquement des concepts sélectionnés lors de la phase de segmentation. Le rôle de cet algorithme est de produire un module complet et indépendant qui sera exporté dans un fichier.

Le protocole de validation quant à lui repose sur une étude comparative de qualité des modules ontologiques générés à l'aide de différents outils de modularisation dont le nôtre. L'évaluation de module se fait à partir des frameworks ou API qui implémentent un bon nombre de métriques. La mise en place de ce protocole est une tâche importante dans la mesure où celui-ci nous permettra de valider notre approche de modularisation, et ce à partir des résultats retournés par les métriques de qualité utilisées pour l'évaluation.

0.3 Organisation du mémoire

Le mémoire de recherche est organisé comme suit. Le premier chapitre est la présente introduction. Le chapitre suivant présente un état de l'art sur l'ensemble des travaux qui ont été menés sur la modularisation des ontologies. Plus précisément, il sera question dans cette partie de définir brièvement la notion d'ontologie, tout en revenant plus en détails sur les principaux buts de la modularisation. Les différentes techniques et outils de modularisation seront présentés tout en mettant l'accent sur les limites actuelles de ceux-ci. Après avoir démontré la nécessité d'introduire une nouvelle approche de modularisation, dans le chapitre 2 nous parlerons de l'algorithme COMET que nous proposons, de son fonctionnement et des détails de son implémentation.

Dans le chapitre 3, il sera question de présenter les différentes approches d'évaluation d'ontologies, mais aussi les frameworks basés sur ces techniques. Dans cette partie, nous parlerons également des métriques de qualité et de l'utilité de chacune d'elles. Le chapitre 4 portera sur les expérimentations et les résultats obtenus en fonction du protocole de validation d'ontologies que nous aurons défini. Enfin, nous concluons dans la dernière partie de ce travail en présentant un bilan général de l'ensemble de nos contributions et en évoquant de nouvelles perspectives de recherche.

CHAPITRE I

ÉTAT DE L'ART

Ce chapitre introduit les fondements de la modularisation, car pour comprendre l'utilité de celle-ci, il est important de se pencher sur la notion d'ontologie tout en précisant quel est son rôle dans le domaine de l'ingénierie de connaissances. La modularisation est perçue principalement comme étant une solution aux problèmes de conception et de partage des ontologies. Les techniques de modularisation actuelles reposent sur des approches sémantiques et structurelles. Chacune de ses approches présente des particularités et limites qui seront discutées et ce, en démontrant la nécessité d'élaborer une nouvelle approche de modularisation.

1.1 La notion d'ontologie

1.1.1 L'ontologie : tentatives de définition

Au fil des années, plusieurs définitions d'ontologies sont proposées par différentes communautés scientifiques. Cependant, un nombre important de chercheurs s'accorde sur celle de Gruber, qui définit une ontologie comme étant une spécification explicite d'une conceptualisation (Gruber *et al.*, 1993). Cette définition de l'ontologie est la plus citée dans la littérature et constitue la base de plusieurs autres

définitions proposées par la suite. Borst a reformulé la définition de Gruber en présentant l'ontologie comme étant une spécification formelle d'une conceptualisation partagée (Borst, 1997). Studer, Benjamins et Fensel quant à eux fusionnent les deux définitions précédentes et considèrent l'ontologie comme étant une spécification formelle et explicite d'une conceptualisation partagée (Studer *et al.*, 1998). Au regard des différentes définitions proposées, nous sommes vite amenés à nous poser deux questions essentielles :

1. Qu'est-ce qu'une spécification ?

2. Qu'est-ce qu'une conceptualisation ?

La notion de spécification naît avec la définition de Gruber. Dans le domaine de l'informatique, la spécification serait une description formelle de la façon dont un objet devrait être défini afin de satisfaire un critère précis (Gruber, 1995). Par exemple, la forme de Backus-Naur (BNF) qui est une notation permettant de décrire les règles syntaxiques dans bon nombre de langages de programmation. Aussi, avec les ontologies il est important de pouvoir spécifier à la fois la syntaxe et la sémantique. Une spécification se doit donc d'être explicite dans le sens où tous les concepts d'une ontologie doivent être clairement définis.

La conceptualisation implique la mise en place d'un ensemble de symboles du langage des ontologies dans un domaine. Par exemple la formule propositionnelle $A(x) \rightarrow B(x)$ qui, bien qu'abstraite, pourrait s'appliquer à des éléments précis d'un domaine de la façon suivante : $homme(x) \rightarrow mortel(x)$. La conceptualisation permet d'identifier des objets qui existent dans un monde donné et les relations qui les lient les uns aux autres. La conceptualisation seule est insuffisante tant que l'interprétation des symboles utilisés pour conceptualiser une ontologie n'est pas partagée. D'où la nécessité de mettre en place un vocabulaire cohérent pour tous.

1.1.2 Les composantes d'une ontologie

Une ontologie définit formellement les termes employés pour décrire et représenter un domaine de connaissance. Cette formalisation se fait sur la base de deux principales approches : l'approche de la logique du premier ordre et l'approche de la logique de description. La première approche permet de formaliser les connaissances en fonction de cinq éléments : les classes, les relations, les fonctions, les axiomes et les instances (Gruber *et al.*, 1993).

- Les classes (C) : Elles représentent les concepts généraux d'un domaine donné.
- Les relations (R) : Elles représentent l'ensemble des associations existant entre les différents concepts du domaine. On distingue des relations de généralisation (*is a*) et des relations d'agrégation (*is a part of*). Formellement, une relation se définit comme un sous-ensemble (R) de produits de n éléments tel que : $\forall r \in R, r = (C_1 \times C_2 \times ... \times C_n)$.
- Les fonctions (F) : Ce sont des cas particuliers de relations. Une fonction représente une relation entre n-éléments dont le n-ième élément est unique pour les (n-1) éléments précédents. Formellement, elle se définit de la façon suivante : $\forall f \in F : (C_1 \times C_2 \times ... \times C_{n-1} \mapsto \times C_{n-1})$.
- Les axiomes (A) : Ils servent à représenter les assertions qui sont toujours vraies.
- Les instances (I) : Elles permettent de représenter les éléments ou les individus des classes de l'ontologie.

La deuxième approche quant à elle permet de véhiculer les connaissances, notamment à partir des trois composantes suivantes : les concepts, les attributs et les individus.

1. Les concepts : Ils représentent les abstractions utilisées pour décrire des

classes d'objets. Un concept est décrit par un terme (*un symbole*), une extension et une intention. Par exemple, pour le concept "voiture", la dimension intentionnelle qui représente la définition de l'objet pourrait être formulée comme suit : la voiture comme étant un véhicule automobile conçu et aménagé pour le transport d'un petit nombre de personnes. La dimension extensionnelle quant à elle serait une description exhaustive de tout ce qui obéit à la définition (intention) : la liste de toutes les voitures du monde

2. Les attributs : Ils permettent de décrire les relations entre concepts ainsi que leurs propriétés. Exemples : l'âge, le nom, la taille, etc...
3. Les individus : Ils représentent les instances de classes. Ce sont des éléments décrits par des classes. Exemple : la voiture de Jean.

1.1.3 Les types d'ontologies

Il existe différents types d'ontologies. Guarino propose une classification des ontologies selon leurs niveaux de généralité (Guarino, 1998). Ainsi, distingue t-on :

- Les ontologies génériques (*Upper-level ontologies*) qui décrivent des notions universelles ou des concepts généraux et abstraits tels que l'espace, le temps et la matière. Elles contiennent des concepts de haut niveau applicables à plusieurs domaines.
- Les ontologies du domaine et de tâches (*Domain and task ontologies*) qui décrivent des concepts spécifiques à une tâche précise ou à un domaine. Elles donnent une représentation formelle des concepts du domaine étudié ainsi que des différentes relations qui lient ces derniers (Gómez-Pérez *et al.*, 2002).
- Les ontologies applicatives (*Application ontologies*) qui décrivent des concepts dépendant à la fois d'un domaine et d'une tâche particulière. Il s'agit en l'occurrence de spécialisation d'ontologies relatives.

- Les ontologies de représentation (*Representation ontologies*) qui conceptualisent les primitives des langages de représentation des connaissances. C'est le cas d'une ontologie sur le formalisme des graphes conceptuels.

Les différents niveaux d'ontologies sont schématisés sur la figure 1.1. Toutes les ontologies sont des spécialisations de l'ontologie générique.

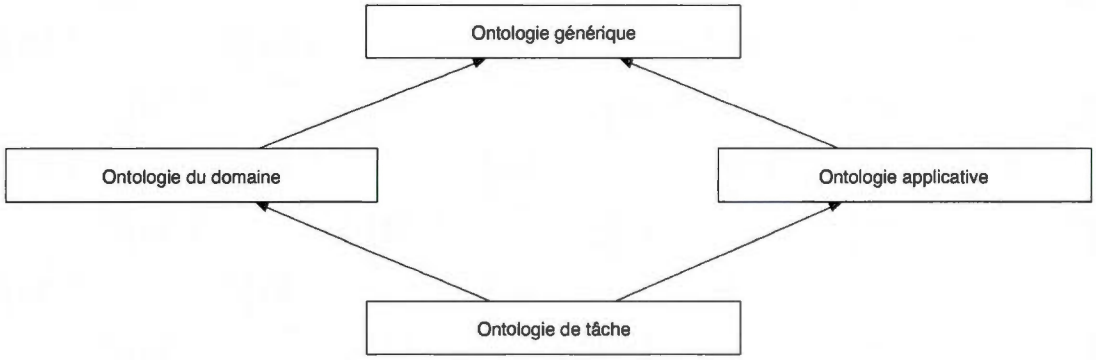

Figure 1.1: Différents types d'ontologies. Les flèches représentent des relations de spécialisation (Guarino, 1998).

1.1.4 Les rôles des ontologies

Étant donné que les ontologies sont des structures très pratiques, on remarquera qu'elles sont utilisées dans des domaines et des contextes variés. De ce fait leur rôle varie d'un cadre à l'autre. En fonction de leur domaine d'application, nous pouvons attribuer aux ontologies les principaux rôles suivants (Figure 1.2) :

- La représentation des connaissances et le raisonnement sur celles-ci.
- Le partage et la réutilisation des connaissances.
- Les communications homme-homme et homme-machine.
- La recherche d'informations.
- L'intégration sémantique des sources de données.

- L'interopérabilité entre les systèmes d'information.

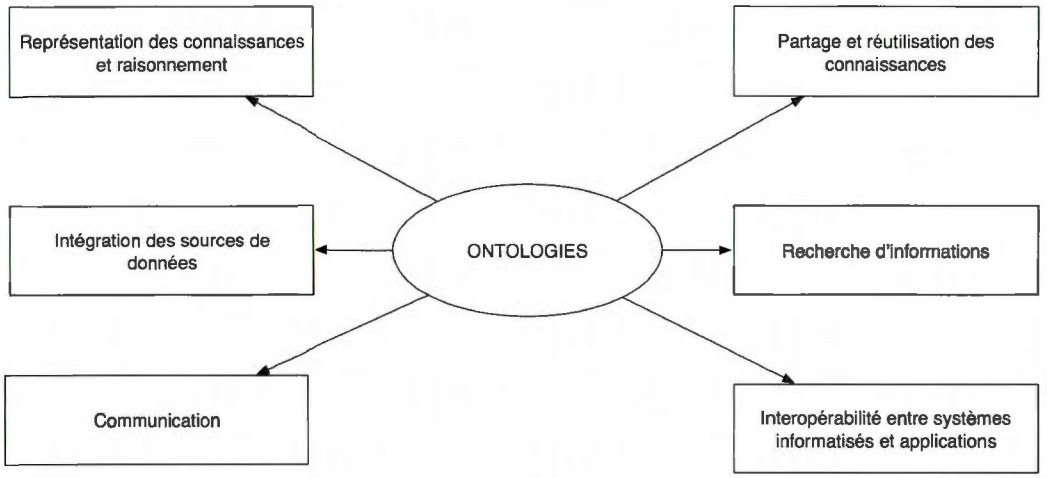

Figure 1.2: Les principaux rôles des ontologies.

1.2 Le but de la modularisation des ontologies

La modularisation est un concept qui nous renvoie simultanément à deux aspects différents de l'ontologie à savoir : «l'ontologie comme un tout» et «l'ontologie comme un ensemble de modules». Les différentes approches de la modularisation et la façon dont elles sont mises en pratique dépendent en grande partie du but visé. Ainsi, le but de la modularisation est d'apporter des solutions aux problèmes suivants (Parent et Spaccapietra, 2009) :

L'extensibilité en fonction des requêtes et du raisonnement sur les ontologies

Les raisonneurs sont plus efficaces sur des ontologies de petite taille. La performance de ceux-ci décroît au fur et à mesure que la taille de l'ontologie croît. Travailler avec des petites ontologies permet de pallier à la perte de per-

formance des moteurs d'inférences, et la modularisation apporterait une solution à ce problème, dans la mesure où elle fragmenterait une ontologie qui tendrait à gagner en taille en un ensemble d'ontologies plus petites. Ainsi, des requêtes peuvent être faites uniquement sur un module précis sans pour autant avoir à explorer l'ontologie en entier.

L'extensibilité en fonction de la maintenance et de l'évolution des ontologies

À ce niveau, le but de la modularisation serait de localiser et contenir l'impact que pourrait avoir la mise à jour d'une base de connaissances (*knowledge repository*) dans les limites des modules. L'implémentation d'une bonne distribution des connaissances nécessite, dans un premier temps, une bonne compréhension du principe de propagation de la mise à jour à l'intérieur même de l'ontologie et, en second lieu, une bonne connaissance de l'information contenue dans cette mise à jour.

La complexité dans la gestion des ontologies

L'on ne saurait garantir l'efficacité et la précision des bases de connaissances de grande taille élaborées par des humains, aussi bien en termes d'objets et de relations entre objets qu'en termes d'axiomes et de règles. Les ontologies qui contiennent des milliers de concepts ne peuvent être conçues et gérées par un seul expert. D'où la nécessité de ramener la conception des ontologies à la création de différents modules manipulables et de taille raisonnable, qui pourront par la suite être liés les uns aux autres.

La compréhensibilité du contenu des ontologies

Les utilisateurs doivent pouvoir être à même de comprendre le contenu d'une ontologie afin de pouvoir la manipuler. Que ce contenu soit représenté sous une forme graphique ou textuelle, il sera plus aisé pour des humains de le lire si l'ontologie est de petite taille. Toutefois, la compréhensibilité ne dépend pas uniquement de la taille de l'ontologie mais aussi de la façon dont les objets, relations et règles sont agencés les uns par rapport aux autres.

La réutilisation des ontologies

La réutilisation est vue comme l'une des motivations premières de la modularisation. Elle permet de mettre l'accent sur les mécanismes de construction de modules ontologiques de telle sorte que ceux-ci puissent être réutilisés par la suite. Il va de soi que le contenu de ces modules se doit d'être pertinent, nécessaire et compréhensible afin que ces modules soient sélectionnés pour réutilisation.

1.3 Les principales approches de modularisation des ontologies

Une ontologie $\mathcal{O}$ est définie par la formule suivante (Palmisano *et al.*, 2009) :

$$\mathcal{O} = (Ax(\mathcal{O}), Sig(\mathcal{O}))$$

où $Ax(\mathcal{O})$ représente l'ensemble d'axiomes composé de concepts, de relations et de fonctions (sous-classe, équivalence, instanciation, etc...). $Sig(\mathcal{O})$ est la signature de $\mathcal{O}$ qui représente l'ensemble de noms des entités qui se retrouvent dans les axiomes. En d'autres termes, il s'agit du vocabulaire de $\mathcal{O}$. La modularisation de l'ontologie $\mathcal{O}$ permet de définir un module M tel que :

$$M = (Ax(M), Sig(M))$$

où M fait partir de $\mathcal{O}$, avec $(Ax(M))^{\mathcal{I}} \subseteq (Ax(\mathcal{O}))^{\mathcal{I}}$ et $Sig(M) \subseteq Sig(\mathcal{O})$.

L'on ne saurait définir l'ontologie sans pour autant introduire la notion d'interprétation $\mathcal{I}$ qui est une base de la sémantique de la logique descriptive. On la symbolise par $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$, où $\Delta^{\mathcal{I}}$ est le domaine d'interprétation et $\bullet^{\mathcal{I}}$ la fonction d'interprétation.

Il existe deux principales approches de modularisation des ontologies. Ainsi, distingue-t-on des approches basées sur le partitionnement de l'ontologie et des approches basées sur l'extraction de module ontologique. Le partitionnement subdivise une ontologie en un ensemble de sous-structures autonomes ou dépendantes les unes des autres qu'on a appellera partitions, tandis que l'extraction de module consiste à extraire une sous-ontologie en fonction d'une signature précise (d'Aquin *et al.*, 2009).

1.3.1 Les techniques de partitionnement des ontologies

Le partitionnement est un processus au cours duquel une ontologie $\mathcal{O}$ est fractionnée en un ensemble de modules (pas forcement disjoints) $\mathcal{M}$ tel que l'union de l'interprétation de toutes les partitions ainsi crées soit équivalente à l'interprétation de l'ontologie initiale $\mathcal{O}$. Formellement, le partitionnement de l'ontologie peut être définie comme suit :

$$\mathcal{M} = \{\{M_1, M_2, ..., M_n\} | \{(Ax(M_1))^{\mathcal{I}} \cup (Ax(M_2))^{\mathcal{I}} \cup ... \cup (Ax(M_n))^{\mathcal{I}}\} = (Ax(\mathcal{O}))^{\mathcal{I}}\}$$

La technique de Stuckenschmidt et Klein

Stuckenschmidt et Klein proposent une approche pour le partitionnement automatique des ontologies basée sur la structure de la hiérarchie des classes (concepts) (Stuckenschmidt et Klein, 2004). Cette approche est basée sur l'hypothèse que les dépendances entre les concepts peuvent être dérivées de la structure même de l'ontologie. Cette dernière peut être représentée comme un graphe

pondéré $O = \langle C, D, w \rangle$ où les nœuds (C) représentent des concepts et les arêtes (D) représentent des relations entre concepts, dont le poids (w) varie en fonction de la dépendance. Le partitionnement se fait en deux étapes. Tout d'abord on extrait l'arbre de dépendance qui est en fait un sous-graphe de l'ontologie initiale. Ensuite on calcule le degré proportionnel de dépendance entre les concepts (Figure 2.3). Et pour cela, on calculera le degré proportionnel de dépendance $w(c_i, c_j)$ entre deux concepts c_i et c_j, avec a_{ij} qui est le poids assigné à la relation qui les unit :

$$w(c_i, c_j) = \frac{a_{ij} + a_{ji}}{\sum_k a_{ik} + a_{ki}}$$

où a_{ij} représente le poids assigné à la relation entre les concepts c_i et c_j. Dans le cadre de leurs expériences, Stuckenschmidt et Klein fixent la valeur de a_{ij} à 1. Ainsi, le degré proportionnel de dépendance sera égale au rapport de a_{ij} par le nombre de concepts auquel le concept c_i est connecté.

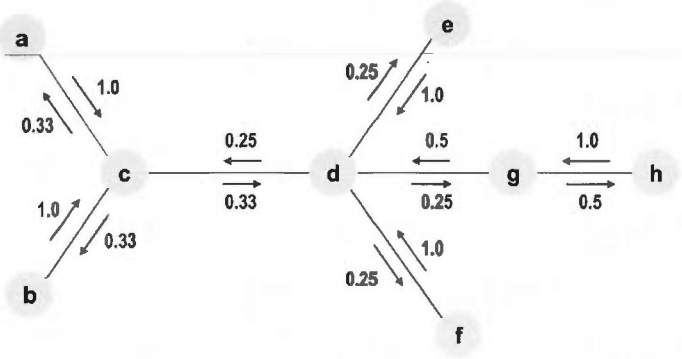

Figure 1.3: Un exemple de graphe avec des dépendances proportionnelles de force (Stuckenschmidt et Schlicht, 2009).

La figure 1.3 fait état du cas où, le nœud **d** est utilisé dans le calcul des poids à l'aide des dépendances proportionnelles. Le nœud **d** a quatre voisins directs, ce qui entraine que le degré proportionnel de dépendance de la relation entre ce nœud et ses voisins est de 0.25 (la valeur a_{ij} qui est fixée à 1 par défaut divisée

par quatre). On constate que différents niveaux de dépendances entre **d** et ses voisins proviennent de la dépendances de ces voisins là avec le nœud **d**. Toutefois, il est important de souligner que ce degré proportionnel de dépendances est non-symétrique, c'est-à-dire qu'il sera différent respectivement pour les relations $\mathbf{d} \rightarrow \mathbf{g}$ et $\mathbf{g} \rightarrow \mathbf{d}$. Les nœuds **e** et **f** n'ont aucun voisins directs autre que **d** ou bien qui dépendent de celui-ci. De ce fait, le poids de leur relation respective avec le nœud **d** demeurent égal à 1. Le degré proportionnel de dépendances entre les nœuds **g** et **d** est de 0.5, car **g** n'a que deux voisins. Quant à la dépendance entre les nœuds **b** et **d** est de 0.33 étant donne que **b** n'a que trois voisins.

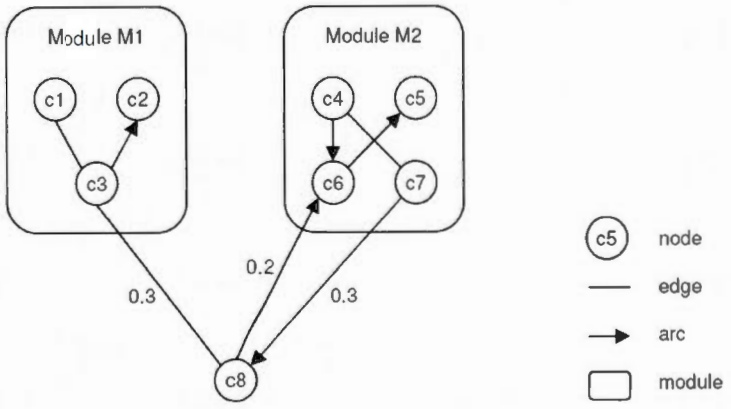

Figure 1.4: Exemple de graphe pour l'attribution des nœuds restant à des modules (Stuckenschmidt et Schlicht, 2009).

Afin de déterminer les partitions, on procède à une analyse du graphe, qui représente un réseau de dépendances proportionnelles dans lequel les nœuds fortement liés seront regroupés sous forme de "cluster" (Figure 1.4). En effet, ces derniers représentent des sous-graphes à l'intérieur desquels les arêtes sont liées plus fortement entre elles qu'à toutes autres arêtes voisines, qui sont en fait des arêtes les liant ainsi à des concepts ou sous-graphes voisins. La taille du sous-graphe est un paramètre que l'utilisateur se doit de fixer au début du processus

de partitionnement.

Stuckenschmidt et Klein estiment que les concepts dans un module se doivent d'être fortement interconnectés. Cette propriété du module permettrait d'identifier tout d'abord les concepts qui ne le sont pas afin des les exclure du cluster. Un fois les limites du potentiel module défini, chacun des concepts qui aurait été préalablement exclu sera rajouté au dit module uniquement dans la mesure où ce dernier contiendrai un ou plusieurs concepts auxquels ce concept potentiellement pertinent soit lié, et ce plus qu'à tout autre concept appartenant à un autre cluster ou potentiel module voisin.

La technique de Cuenca Grau *et al.*

Afin de résoudre le problème du partitionnement d'une ontologie, Cuenca Grau *et al.* proposent une approche basée sur les ε-connexions (Grau *et al.*, 2005). Une ε-connexion Σ est un langage de représentation de connaissances défini comme étant une combinaison de différents formalismes logiques. Les ε-connexions ont été élaborées dans le but d'accroître l'expressivité de chaque composant logique tout en préservant la décidabilité des raisonneurs (Kutz *et al.*, 2004). Ainsi, Cuenca Grau *et al.* utilisent les ε-connexions comme langage de définition et d'instanciation d'ontologies OWL.

Les ε-connexions permettent de séparer distinctement les domaines d'interprétation de n-systèmes combinés (chaque système peut être vu comme étant une base de connaissances en logique de description), où ces domaines sont liés à l'aide de n relations de type «*link relations*». Ces relations permettent de représenter les connexions entre différentes partitions de telle sorte que le raisonnement peut se faire sur chaque partition individuellement ou plutôt sur une combinaison de partitions liées entre elles.

Les partitions générées par Cuenca Grau *et al.* sont à la fois structurellement et sémantiquement compatibles (Grau *et al.*, 2005). Soient une ontologie $\mathcal{O}$ et une ε-connexion Σ ayant respectivement pour vocabulaires V et V_n. Σ est structurellement compatible avec $\mathcal{O}$ ($\Sigma \sim \mathcal{O}$) si et seulement si :

1. V_n est un vocabulaire partitionné de $\mathcal{O}$. Ce qui sous-entend que Σ contient exactement les mêmes entités (concepts, propriétés et individus) et axiomes que $\mathcal{O}$.

2. $A \in \Sigma \Leftrightarrow A \in \mathcal{O}$.

Σ est sémantiquement compatible avec $\mathcal{O}$ ($\Sigma \approx \mathcal{O}$) si et seulement si :

1. $\mathcal{O}$ est partitionnable pour V_n.

2. Si $\mathcal{I}(V_n) \leftrightarrow \mathcal{M}$, alors $\mathcal{M} \models \Sigma$ ssi $\mathcal{I}(V_n \models \mathcal{O}$.

La compatibilité structurelle a pour but de garantir qu'aucune entité ou axiome ne sera ajouté, retiré ou modifié lors du partitionnement. En d'autres termes, on tient à s'assurer que chaque axiome existant dans la ε-connexion existe aussi dans l'ontologie. La compatibilité sémantique représente ici la relation souhaitée entre l'ontologie initiale et l'ontologie résultante du processus de partitionnement. En effet, le rôle de la compatibilité sémantique est de garantir que l'interprétation de l'ontologie partitionnée est équivalente à l'interprétation de la même ontologie non-partitionnée, donnant lieu ainsi à la préservation du modèle d'interprétation. On s'assure principalement que chaque ontologie équivalente corresponde exactement au même ensemble de ε-connexions compatibles. L'approche de partitionnement de Cuenca Grau *et al.* permet ainsi d'identifier les propriétés dans $\mathcal{O}$ qui lient $\mathcal{O}$ à Σ_i ou Σ_i à $\mathcal{O}$, tout en garantissant à la fois les compatibilités structurelle et sémantique.

Analyse des techniques de partitionnement

On remarque que les deux approches présentées ci-dessus abordent le problème de partitionnement avec des perspectives complètement différentes. L'approche de Stuckenschmidt et Klein ne considère pas la sémantique de l'ontologie, ce qui rend son approche applicable à travers différents langages d'ontologie ; tandis que Cuenca Grau *et al.* s'appuient uniquement sur la logique de description, qui apporte à leur approche la garantie qu'il n'y aura aucune altération de l'information contenue dans les partitions par rapport au contenu de l'ontologie d'origine. Néanmoins, il convient de souligner que le calcul de degré de dépendance dans l'approche de Stuckenschmidt et Klein pourrait dans un certain sens être vu comme un facteur limitant l'efficacité de leur algorithme, dans la mesure où en plus de faire abstraction de la sémantique de l'ontologie, il ne considère aucunement le cadre dans lequel la partition crée entend être utilisée. À l'instar de l'approche de Stuckenschmidt et Klein, l'algorithme de partitionnement de Cuenca Grau *et al.* n'est pas paramétrable, mais son réel défaut vient du fait qu'il ne s'applique pas à toutes les ontologies, car certaines ne sont pas partitionnables à l'aide des ε-connexions, et ce dans les cas où les compatibilités structurelle et sémantique ne pourraient être garanties lors du processus de partitionnement.

1.3.2 Les techniques d'extraction de module

L'extraction de module ontologique est un processus au cours duquel un module M couvrant une signature spécifique est extrait d'une ontologie $\mathcal{O}$, telle que $Sig(M) \subseteq Sig(\mathcal{O})$. M est la partie pertinente de $\mathcal{O}$ qui couvre l'ensemble d'éléments défini par $Sig(M)$. Le module M est une ontologie en soi, et comme tel, d'autres modules peuvent être extraits à partir de lui. Formellement, l'extraction

du module peut être définie comme suit :

$$extraction(\mathcal{O}, Sig(M)) \rightarrow M | (Ax(M))^{\mathcal{I}} \subseteq (Ax(\mathcal{O}))^{\mathcal{I}}$$

L'extraction basée sur le parcours de graphe

Les techniques d'extraction basées sur une approche structurelle nécessite une représentation de l'ontologie sous forme de graphe afin de parcourir celle-ci et d'en extraire le module ontologique. Parmi ces approches on distingue :

L'approche de d'Aquin et al.

d'Aquin *et al.* décrivent l'extraction de module ontologique comme étant une étape d'un processus plus complexe nommé «la sélection des connaissances» (*Knowledge selection*) (d'Aquin *et al.*, 2006). La sélection des connaissances a pour but de récupérer les composants pertinents des ontologies disponibles en ligne afin d'annoter automatiquement la page Web retournée par le navigateur. Cette approche a été implémentée au sein d'un outil appelée *Magpie*, qui est un plugin pour navigateur. *Magpie* permet d'identifier les instances de classes d'ontologies sur une page Web en surlignant chacune d'elles avec une couleur qui lui est associée. En plus de ce mécanisme de sélection automatique d'ontologies, *Magpie* permet aussi d'extraire les parties les plus utiles et pertinentes des ontologies qui décrivent les classes des instances sur une page Web.

Le processus de sélection de connaissances se fait en trois étapes (Figure 2.4) :

1. *La sélection des ontologies pertinentes.*

   Tout d'abord, on rentre un ensemble de termes pour lesquels on recherche une ontologie. Ensuite, on identifie les ontologies ou les ensembles d'ontologies qui couvrent les termes qui ont été rentrés. Par couverture, on sous-

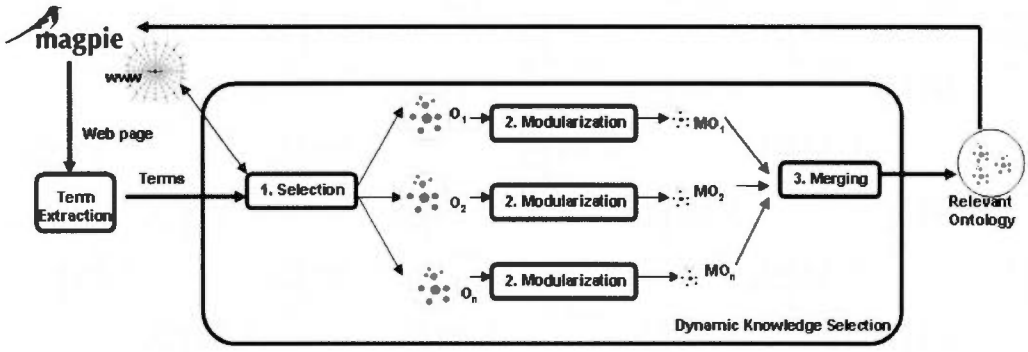

Figure 1.5: Le processus de sélection de connaissances et son utilisation avec *Magpie* (d'Aquin *et al.*, 2006).

entend que l'ensemble des ontologies contenant des concepts, propriétés, ou instances qui sont sémantiquement liés aux termes donnés doit être retourné par l'algorithme.

2. *La modularisation des ontologies sélectionnées.*

   La technique de partitionnement de Stuckenschmidt et Klein est appliquée sur les ontologies retournées lors de l'étape précédente afin d'identifier les différentes sous-ontologies qui contiennent une connaissance suffisamment pertinente pour la tâche à réaliser (Stuckenschmidt et Klein, 2004). Le principe d'extraction de cette approche est de rajouter au fur et à mesure dans les modules tous les éléments participant directement ou indirectement à la définition des entités déjà incluses dans chacun de ceux-ci. En d'autres termes, si un concept impliqué dans une définition n'est pas encore inclus dans l'un des modules, alors celui-ci y est systématiquement rajouté.

3. *La fusion des modules ontologiques pertinents.*

   Dans la mesure où plusieurs ontologies seraient retournées après la phase de sélection, les modules extraits individuellement de celles-ci sont fusionnés afin de produire une ontologie unique, résultat final du processus de sélection

de connaissances. d'Aquin *et al.* ne fournit aucun détails sur le déroulement du processus de fusion des modules.

L'approche de Doran et al.

Doran *et al.* abordent le problème d'extraction avec la perspective de l'ingénieur de connaissances qui souhaiterait pouvoir réutiliser une partie d'une ontologie existante. Cette approche ne tient pas compte du type de langage dans lequel l'ontologie est représentée, dès lors que celui-ci est transformable en un *modèle de graphe abstrait* (Doran *et al.*, 2007). Le graphe abstrait de Doran *et al.* est un graphe orienté et étiqueté G avec un alphabet donné Σ_E, tel que $G = (V, E)$ où :

- V est un ensemble fini de sommets.
- $E \subseteq V \times \Sigma_E \times V$ est une relation ternaire décrivant les arêtes.

Doran *et al.* distinguent dans ce graphe un ensemble d'arêtes à parcourir et un autre à ne pas parcourir, du moins lors de la première itération de l'algorithme. Par exemple, lorsqu'on extrait un module d'une ontologie OWL-DL, les arêtes `owl:disjointWith` ne sont pas parcourues à la première itération, elles le sont uniquement lors des itérations suivantes. Les arêtes sont étiquetées en fonction du langage de l'ontologie.

En utilisant le modèle abstrait de Doran *et al.*, il serait donc possible de définir un module ontologique G_M tel que $G_M = (V_M, E_M)$, avec $V_M \subseteq V \wedge V_M \neq \emptyset$ et $E_M \subseteq E$. Ce qui implique que $G_M \subseteq G$. Cette approche permettrait de ramener le problème d'extraction de module à un parcours de graphe avec pour point de départ un sommet x tel que $x \in V_G$. La seule condition est que les arêtes étiquetées «*disjoint*» du sommet x ne soient pas parcourues à la première itération. Deux concepts sont disjoints s'ils n'ont aucun ancêtre commun. Toutefois, Doran *et al.* supposent que si l'utilisateur souhaite inclure les concepts disjoints dans le module

lors de la première itération, alors il devra tout simplement choisir pour point de départ leur ancêtre commun.

Le module extrait est un graphe $G_M = (V_M, E_M)$, où V_M et E_M représentent respectivement les ensembles de sommets et d'arêtes. Le module ontologique généré par cette approche reste transitivement fermé et ce, même en tenant compte des différentes relations parcourues. Afin de garantir cette fermeture transitive, l'ontologie dont on veut extraire le module se doit elle aussi d'être transitivement fermée. La notion de fermeture transitive ici suggère que toute relation existante entre concepts est considérée, et ce même ci cette relation lie un concept du module à un concept intermédiaire (concept qui n'appartient pas au module mais qui est lié à un autre appartenant au module). Il convient tout de même de noter que les concepts hiérarchiquement supérieurs au concept à partir duquel le processus commence ne sont pas parcourus ; d'autant plus qu'il revient à l'utilisateur de définir le concept de départ. En considérant les concepts plus «généraux» lors du processus d'extraction, on risquerait de se retrouver avec un module dont les proportions sont équivalentes à celles de l'ontologie à modulariser (Doran *et al.*, 2007).

Le principal avantage de l'approche de Doran *et al.* est que le modèle de graphe abstrait est applicable à toutes ontologies, quelque soit leur langage. Par exemple, l'alphabet pour une ontologie OWL-DL est :

$$\Sigma_E = \{subClassOf, disjointWith, equivalentTo, subPropertyOf, property\}$$

tandis que celle d'une ontologie en RDFS sera :

$$\Sigma_E = \{subClassOf, subPropertyOf, property\}$$

Si une arête du graphe abstrait est étiqueté « *property*» alors cela signifie que le sommet de départ est le concept de départ (*domain*) et le sommet d'arrivée est le concept d'arrivée (*range*).

L'approche de Noy et Musen

Noy et Musen adaptent le concept de vue (*view*) issu du domaine des bases de données en introduisant une nouvelle notion : la vue d'ontologie (*ontology view*) (Noy et Musen, 2004). La vue (*database view*) est la portion d'instances d'une base de données qui est générée en réponse à une requête de l'utilisateur. Par analogie, en ingénierie des connaissances, un module ontologique peut être considéré comme une vue d'ontologie, dans la mesure où en se basant sur le principe d'encapsulation, celui-ci représente une sous-structure générée en réponse à une requête de l'utilisateur sur l'ontologie initiale. Cette requête est composée d'une liste de termes qui représente en fait la signature du module à obtenir. L'utilisateur se doit de fixer le point de départ du processus d'extraction en désignant un concept dont les relations seront parcourues récursivement afin d'inclure l'ensemble des entités qui lui sont liées.

Les relations à parcourir sont sélectionnées par l'utilisateur et à chacune de celles-ci il attribuera une profondeur de parcours : il s'agit de la directive de parcours (*traversal directive*) $\mathcal{TD}$. Lorsque cette profondeur est atteinte, l'algorithme arrête de parcourir la relation sélectionnée. La directive de parcours D d'une ontologie $\mathcal{O}$ est définie par la paire $\langle C_{st}, \mathcal{PT} \rangle$ où :

- C_{st} est le concept de départ du parcours.
- $\mathcal{PT}$ est un ensemble de directives de propriétés (*property directives*). Chaque directive de propriété est une paire $\langle P, n \rangle$ où P est une propriété de $\mathcal{O}$ et n est un entier non-négatif ou infini (∞) qui définit la profondeur de parcours de la propriété P. Si $n = \infty$, alors le parcours inclura aussi une couverture transitive pour P à partir de C_{st}.

Noy et Musen définissent une spécification de la vue de parcours (*traversal view specification*) T comme étant un ensemble de directives de parcours. Soit une

ontologie $\mathcal{O}$ et une spécification de la vue de parcours T constitué d'un ensemble de directives de parcours $\mathcal{TD}$. Le résultat d'une spécification de la vue de parcours $TV(\mathcal{O}, T)$ est aussi une vue de parcours qui en fait, représente l'union de tous les résultats des directives de parcours D, tel que $D \in \mathcal{TD}$. On rappellera qu'une vue de parcours contient toutes les classes et instances rencontrées tout le long du parcours.

La technique de Noy et Musen a été implémentée et incorporée dans l'outil *PROMPT*, qui est un plugin pour l'éditeur d'ontologie *Protégé* et qui donne à l'utilisateur la possibilité de gérer plusieurs ontologies et ce, en lui permettant de comparer les versions, de fusionner et d'extraire les modules ontologiques (Noy et Musen, 2003).

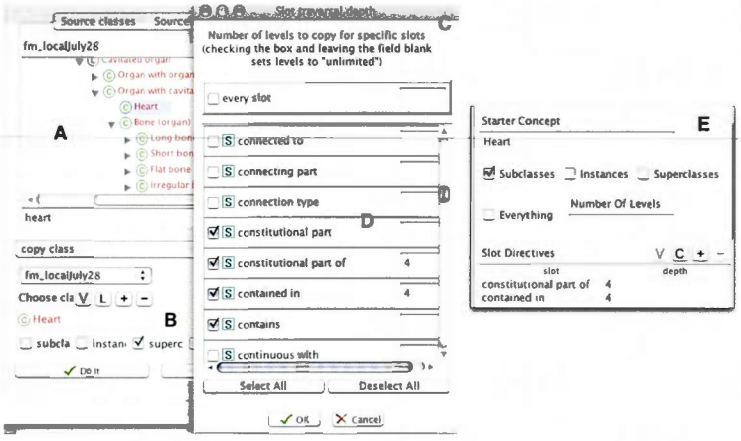

Figure 1.6: Interface utilisateur de PROMPT pour la spécification de la vue de parcours (Noy et Musen, 2004).

La figure 2.5 représente l'interface de spécification des vues de parcours. En naviguant dans la hiérarchie de l'ontologie, l'utilisateur choisit le concept de départ (A). Puis, il sélectionne les directives de propriétés (B). Si nécessaire, une fenêtre annexe offre des options supplémentaires sur les propriétés directives liées

au concept sélectionné (C), de plus l'utilisateur peut y spécifier la profondeur des propriétés qui l'intéressent (D). Le résultat obtenu est affiché sur une fenêtre supplémentaire. L'utilisateur peut une nouvelle fois spécifier les vues de parcours s'il estime que le résultat obtenu est non satisfaisant (E).

L'approche de Noy et Musen est flexible et permet à un ingénieur de connaissances de construire un module de façon itérative, mais pour ce faire, il doit avoir des connaissances approfondies sur l'ontologie à manipuler afin d'être à même de définir des directives de parcours appropriées (Noy et Musen, 2004).

L'approche de Seidenberg et Rector

Seidenberg et Rector ont développé une technique d'extraction de module ontologique à partir de GALEN qui est une ontologie médicale (Seidenberg et Rector, 2005). Toutefois, le noyau de cette technique est générique et de ce fait, l'algorithme reste applicable à d'autres ontologies. Cette approche prend en entrée une ou plusieurs ontologies et la signature du module à extraire $Sig(M)$. Les classes présentent dans la signature du module ainsi que tout autre élément participant, même indirectement, à la définition d'une classe déjà incluse dans le module sont systématiquement rajoutés à celui-ci.

Soit un concept A de l'ontologie et une signature de M tel que $Sig(M) = A$. Le processus d'extraction se fait en deux phases (Figure 2.5). Tout d'abord l'ontologie est parcourue vers le haut afin d'inclure toutes les super-classes de A. Puis, la hiérarchie de l'ontologie est parcourue vers le bas dans le but de rajouter les sous-classes de A. Il convient de souligner que les classes sœurs de A (*sibling classes*), c'est-à-dire des classes ayant un ancêtre commun et se trouvant à la même profondeur dans la hiérarchie, sont ignorées. Pour inclure ces dernières dans le module, on doit au préalable les rajoutées dans $Sig(M)$. Les restrictions,

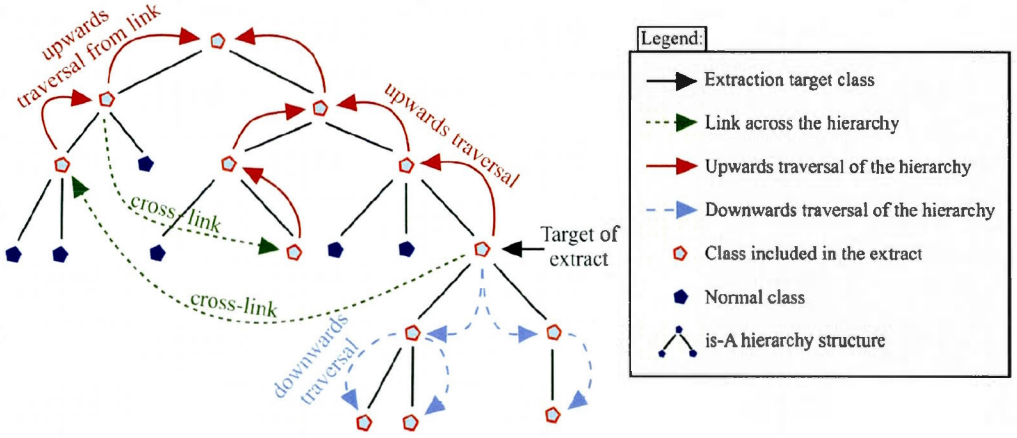

Figure 1.7: Parcours de la hiérarchie des classes à travers les liens (Seidenberg et Rector, 2005).

intersections, unions et toutes classes équivalentes de celles déjà incluses sont aussi rajoutées au module. En dernier lieu, les propriétés des classes précédemment sélectionnées sont aussi parcourues vers le haut afin de rajouter d'autres classes au module.

Seidenberg et Rector améliorent leur algorithme en introduisant deux notions essentielles qui sont le filtrage de propriétés et l'utilisation de classes frontières, afin de pouvoir contrôler les proportions du module dans le cas où on travaillerait avec une ontologie dense et large. Le processus de filtrage de propriétés consiste à supprimer les propriétés choisies par l'utilisateur (Seidenberg et Rector, 2006). Pour illustrer cette approche, considérons un cas où l'utilisateur ne serait pas intéressé par l'ensemble des maladies modélisé dans l'ontologie GALEN. Celui-ci choisira alors d'exclure toutes les propriétés locatives, c'est-à-dire uniquement des propriétés qui lient des maladies à des parties précises du corps humain. Par exemple, l'utilisateur pourrait, afin d'éliminer l'ensemble de maladies de notre modèle, juste supprimer toutes les propriétés *hasLocation* dans des relations simi-

laires à celle-ci : "IschaemicCoronaryHeartDisease *hasLocation* Heart".

Le filtrage de propriétés passe aussi par la suppression de toutes les restrictions de classes dans lesquelles ces propriétés apparaissent. Toutefois, il arrive fréquemment qu'en supprimant une restriction, la définition de la classe concernée devienne, soit impossible à distinguer, soit équivalente à une autre définition de classe similaire. On assiste ainsi à l'apparition dans l'ontologie de longues séries de classes équivalentes qui, bien que correctes sont impossibles à visualiser dans un éditeur d'ontologie tel que Protégé. Pour résoudre ce problème, Seidenberg et Rector proposent une méthode permettant de transformer les classes équivalentes en classes primitives qui conservent leur position dans la hiérarchie, devenant ainsi facile a visualiser dans les éditeurs. L'exemple suivant illustre le filtrage d'une propriété avec le retrait d'une définition :

$$SkinOfFrontalScalp \equiv (SkinOfScalp \sqcap \exists hasSpecificProximity.FrontalBone)$$

$$SkinOfFrontalScalp \equiv SkinOfScalp$$

$$SkinOfFrontalScalp \sqsubseteq SkinOfScalp$$

Lors du processus de filtrage, la restriction présente sur la classe $SkinOfScalp$ est retirée. La résultante est une classe équivalente, dont la définition est convertie par la suite en classe primitive.

La seconde méthode de délimitation du module proposée par Seidenberg et Rector se fait à l'aide de deux éléments préalablement définis par l'utilisateur à savoir la profondeur de récursion et le concept cible à partir duquel le processus de segmentation débute. Le parcours des liens à travers l'ontologie en partant du concept cible s'arrête lorsque la classe qui se trouve à la frontière est atteinte : on parlera alors de classe frontière (*boundary class*). En effet, une classe frontière est la classe sur laquelle l'algorithme de segmentation s'arrête lorsqu'une certaine

profondeur de récursion est atteinte, entraînant ainsi la suppression de tous les liens de cette classe (Figure 2.6). Il s'agit d'une méthode efficace qui permet de limiter la taille du module dès lors que les classes frontières sont considérées comme étant la condition d'arrêt de l'algorithme.

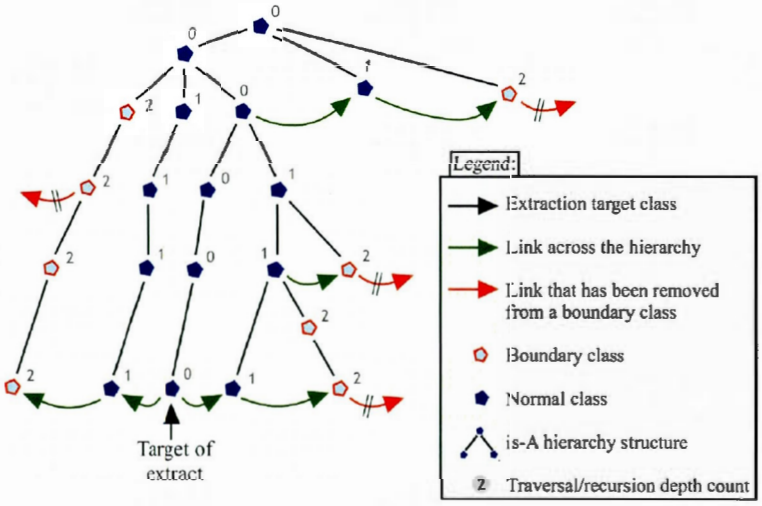

Figure 1.8: Extraction de segment avec profondeur de 2 (Seidenberg et Rector, 2006).

L'algorithme basique d'extraction de module ontologique de Seidenberg et Rector est implémenté dans l'outil *SegmentationApp*[1]. Il s'agit d'une application contenue dans un fichier `.jar` qui s'exécute à l'aide du terminal (Figure A.3). Cet outil prend en entrée deux éléments à savoir l'ontologie à modulariser et le fichier texte contenant la signature du module à extraire. Toutefois, rappelons que l'algorithme basique de Seidenberg et Rector n'est pas paramétrable. Le module ontologique ainsi généré est automatiquement exporté dans un fichier `.owl`.

1. `https://www.w3.org/2001/sw/wiki/SegmentationApp`

L'extraction basée sur la logique

À l'instar des techniques d'extraction basée sur le parcours de graphe, les techniques d'extraction basée sur la logique reposent sur la notion d'extension conservatrice (*conservative extension*). Un module est une sous-structure de l'ontologie dont il est extrait. Cette ontologie est une extension conservatrice si les implications logiques à propos du module sont comprises dans sa signature. De façon plus formelle, Lutz *et al.* définissent l'extension conservatrice comme suit (Lutz *et al.*, 2007) :

Soient $\mathcal{T}_1$ et $\mathcal{T}_2$ des T-Box formulées en logique descriptive $\mathcal{L}$, et soit $\Gamma \subseteq Sig(\mathcal{T}_1)$ une signature. Alors $\mathcal{T}_1 \cup \mathcal{T}_2$ est une Γ-extension conservatrice de $\mathcal{T}_1$, si pour tout $C_1, C_2 \subseteq \mathcal{L}(\Gamma)$, on a $\mathcal{T}_1 \models C_1 \sqsubseteq C_2$ si et seulement si $\mathcal{T}_1 \cup \mathcal{T}_2 \models C_1 \sqsubseteq C_2$.

Cette définition sous-entend que toutes les implications logiques de la signature d'un module ontologique sont les mêmes que si on fait l'union de ce module et de l'ontologie dont il a été extrait. L'inconvénient avec la définition de Lutz *et al.* est que déterminer si une ontologie $\mathcal{O}$ est une extension conservatrice est un problème indécidable en langage OWL-DL.

Afin de venir à bout des limites de l'extension conservatrice pour une logique descriptive plus expressive, Grau *et al.* proposent d'utiliser des contraintes moins strictes : on parlera de modules basés sur la localité (*locality-based modules*) (Grau *et al.*, 2008). Grau *et al.* introduisent également deux nouvelles notions : la couverture (*coverage*) et la sécurité (*safety*) qui sont des propriétés garanties par des modules basés sur la localité. Ces propriétés sont définies en des termes d'un module importé par une ontologie locale (L) comme suit (Grau *et al.*, 2007) :

- La couverture garantit que l'ensemble des termes qui se rapportent aux termes spécifiés sera extrait de l'ontologie. Un module $\mathcal{O}'$ couvre une on-

tologie $\mathcal{O}$ pour les termes d'une certaine signature Sig dans la mesure où pour toutes les classes A et B tels que $A, B \in Sig(\mathcal{O}')$, si $L \cup \mathcal{O} \models A \sqsubseteq B$ alors $L \cup \mathcal{O}' \models A \sqsubseteq B$.

- La sécurité garantit que la signification des termes extraits ne sera pas modifiée. L utiliserait les termes de la signature Sig de façon sécuritaire dans la mesure où pour toutes les classes A et B tels que $A, B \in Sig(\mathcal{O}')$, si $L \cup \mathcal{O}' \models A \sqsubseteq B$ alors $\mathcal{O}' \models A \sqsubseteq B$.

Grau *et al.* décrivent aussi deux types de localité : la localité syntaxique qui peut être calculée en temps polynomial, et la localité sémantique dont le calcul est un problème PSPACE-complet (Grau *et al.*, 2009). À la différence de la localité syntaxique qui est calculée à partir de la structure syntaxique des axiomes, la localité sémantique repose uniquement sur l'interprétation ($\mathcal{I}$) de l'axiome. Jimènez *et al.* quant à eux proposent deux conditions différentes de localité afin d'extraire les modules ontologiques (Jiménez-Ruiz *et al.*, 2008) :

- La $\bot$-localité qui extrait un module approprié pour un raffinement et celui-ci contient tous les super-concepts de la signature.
- La $\top$-localité qui extrait un module approprié pour une généralisation et qui contient tous les sous-concepts de la signature.

Afin d'illustrer ce principe, considérons la T-Box suivante composée de trois axiomes :

$$\texttt{Périodique} \sqsubseteq \texttt{Publication}$$

$$\texttt{Hebdomadaire} \sqsubseteq \texttt{Périodique}$$

$$\texttt{Journal} \sqsubseteq \texttt{Périodique}$$

Dans la mesure où l'on voudrait extraire un module à propos de `Périodique` ($Sig(\texttt{Périodique})$), la $\bot$-localité inclura uniquement le premier axiome, tandis que la $\top$-localité inclura les deux derniers axiomes.

Le raffinement et la généralisation simultanée d'un terme externe (ne se trouvant pas dans l'ontologie locale) pourrait compromettre la sécurité. L'inconvénient de la localité syntaxique est que des axiomes sémantiquement équivalents peuvent être traités différemment s'ils sont syntaxiquement différents (Borgida et Giunchiglia, 2007). Considérons deux ensembles d'axiomes $\{A \sqsubseteq (B \sqcap C)\}$ et $\{A \sqsubseteq B, A \sqsubseteq C\}$, bien que ces axiomes soient sémantiquement équivalents, la différence syntaxique affectera sans aucun doute le processus d'extraction du module. Aussi, la localité syntaxique ne peut pas gérer les tautologies, quoique des ontologies contenant des tautologies sont considérées comme des bases de connaissances mal conçues.

L'approche d'extraction basée sur la logique de Grau *et al.* est implémentée dans l'outil *OWL Module Extractor*[2]. Il s'agit d'une application Web développée par Rafael Gonçalves dans le cadre d'un projet en ingénierie de connaissances à l'université de Manchester (Royaume-Uni). OWL Module Extractor prend en entrée deux paramètres : l'URI ou le contenu de l'ontologie à modulariser et la signature qui représente la liste de termes à couvrir par le module à extraire. Avant de procéder à l'extraction, l'utilisateur devra en premier lieu définir la localité du module en question (Figure A.1). Ainsi, pour une extraction basée sur la $\top$-localité, il se doit de cocher la case «*Bottom module*». Dans le cas d'une extraction basée sur la $\bot$-localité, la case «*top module*» devra être cochée. Une fois le processus d'extraction terminé, le module généré est affiché sur une nouvelle page Web dont le contenu peut être copié et sauvegardé.

2. `https://www.w3.org/2001/sw/wiki/OWLModuleExtractor`

L'extraction basée sur SPARQL

Il existe différentes approches d'extraction de module ontologique. Toutes ces techniques ont été conçues pour différentes applications et sont également basées sur des hypothèses toutes différentes. Il devient donc impératif d'unifier toutes ces approches en un framework commun offrant ainsi la possibilité de sélectionner, adapter et combiner ces techniques.

Les approches proposées par Borgida et Giunchiglia ainsi que d'Aquin *et al.* nécessitaient que l'utilisateur se familiarise avec des formalismes non-standards (Borgida et Giunchiglia, 2007; d'Aquin *et al.*, 2007). Par contre le travail de Doran *et al.* utilise RDF et SPARQL comme base pour un framework commun d'extraction de module, car tous deux sont des standards W3C (Doran *et al.*, 2008).

Toutes les ontologies OWL peuvent être représentées en un graphe RDF et SPARQL est le langage de requête pour RDF. Doran *et al.* présentent le framework SOMET (Figure 2.8) et montrent ainsi qu'il est possible de classer les approches d'extraction de module basées sur le parcours comme étant une série de requêtes SPARQL sur un graphe RDF.

Le framework SOMET contient donc des représentations SPARQL des différentes techniques d'extraction de module par parcours de graphe. C'est un outil flexible en ce sens qu'il permet à l'utilisateur d'ajouter, de modifier ou de retirer des requêtes SPARQL de l'ensemble de requêtes qui doivent passer au moteur d'extraction (*Traversal Extraction Engine*). Étant donné que les différentes approches d'extraction sont assimilées à un ensemble de requêtes SPARQL, ces ensembles ne sont pas disjoints et leurs intersections permettent ainsi de mettre en évidence les points communs entre les différentes techniques.

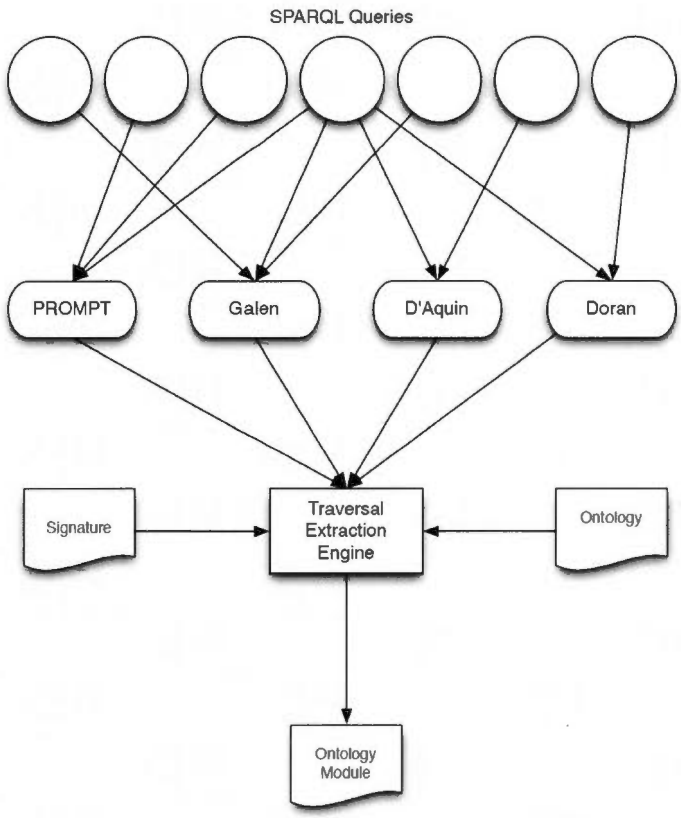

Figure 1.9: Le framework SOMET (Doran *et al.*, 2008).

SOMET utilise un algorithme de parcours de graphe dont les paramètres sont : l'ontologie dont on veut extraire le module, la signature qui décrit le module et un ensemble de requêtes SPARQL. Il s'agit d'un algorithme itératif qui, dans une premier temps, applique des requêtes sur les éléments de la signature afin de construire le module désiré, et dans un deuxième temps de nouvelles requêtes sont appliquées sur chacun des éléments retournés par les requêtes précédentes. Ainsi de nouveaux éléments sont rajoutés au module à chaque itération (Doran *et al.*, 2008).

Dans le cas de l'approche de Doran *et al.*, dont le processus d'extraction se fait à partir d'un concept unique, on distingue les requêtes SPARQL suivantes :

- `DESCRIBE` $?c$: cette requête permet de décrire le concept sur lequel elle est appliquée et ce, en lisant toutes les déclarations (*statements*) dans lesquelles le concept $?c$ apparaît comme étant le sujet.
- `CONSTRUCT` $\{?y$ `rdfs : domain` $?c\}$ `WHERE` $\{?y$ `rdfs : domain` $?c\}$: retourne toutes propriétés $?y$ dont le concept $?c$ est l'espace de départ (*domain*) de la propriété.
- `DESCRIBE` $?y$ `WHERE` $\{?y$ `rdfs : subClassOf` $?c\}$: retourne toutes les sous-classes de $?c$
- `CONSTRUCT` $\{?y$ `owl : equivalentClass` $?c\}$
  `WHERE` $\{?y$ `owl : equivalentClass` $?c\}$: retourne tous les concepts $?y$ où $?y$ est une classe équivalente à $?c$.

1.4 Avantages et limites des approches structurelle et sémantique

La façon la plus évidente pour l'Homme de visualiser une ontologie est de la représenter comme un graphe dont les sommets sont des concepts et individus et les arêtes sont des relations (hiérarchiques et sémantiques). Le problème

d'extraction de module ontologique serait dont ainsi ramené à l'extraction d'un sous-graphe contenant les concepts les plus pertinents par rapport à la liste de termes entrée par l'utilisateur (Dupont *et al.*, 2006). Mais alors, on se retrouve dans la nécessité d'évaluer la pertinence des concepts les uns par rapport aux autres. On pourrait se référer à la notion de distance en ce sens que les plus pertinents seraient les plus proches des concepts de départ dans un certain rayon prédéfini par l'utilisateur, et dont la distance inter-concepts pourrait être évaluée par le calcul du plus court chemin de *Dijkstra*. Cette approche serait réalisable, d'autant plus qu'il existe une bibliothèque Java qui s'appelle JUNG, qui permet de transformer une ontologie en graphe.

Les avantages de cette approche structurelle sont multiples. Tout d'abord, il existe déjà des algorithmes efficaces d'extraction de sous-graphes. En supposant que l'ontologie dont on veut extraire le module est de qualité satisfaisante, cette approche ne déstructure pas l'ontologie dans la mesure où le module extrait conserve la même structure interne que l'ontologie d'origine. Et pour peu que le graphe ne soit pas dense, la complexité de l'algorithme s'en trouve réduit. Par contre, JUNG considère par défaut que toutes les arêtes du graphe sont équivalentes. De ce fait, il devient impossible d'évaluer correctement les nœuds et les arcs, d'autant plus que dans une ontologie les relations entre différents concepts ne sont jamais équivalentes. De plus, cette méthode ignorerait les concepts éloignés des concepts de départ mais qui pourraient être tout aussi pertinents.

La deuxième façon de voir une ontologie serait de la considérer à travers son utilité qui est la représentation des connaissances. En d'autres termes, lorsqu'il est question de définir des concepts qui sont utiles les uns par rapport aux autres afin de comprendre le domaine que l'ontologie décrit, on parlera de sémantique du domaine. Ceci dit, en faisant totalement abstraction de la structure de l'ontologie, on serait à même de prendre des concepts deux à deux afin de voir s'ils sont

sémantiquement proches (Jiang et Conrath, 1997). À la différence de l'approche structurelle, on fait allusion ici à une distance sémantique qui nous permettrait d'évaluer la proximité sémantique des concepts de l'ontologie par rapport à ceux correspondant aux termes rentrés par l'utilisateur.

Les avantages de l'approche sémantique sont la pertinence et la précision, car ce qui fait le principal atout d'une ontologie demeure sa sémantique. Il suffit qu'un concept soit lié à un concept qui ne cadre aucunement avec le contexte pour lequel l'ontologie a été crée pour que la qualité de celle-ci s'en trouve altérée, car on se retrouverait ainsi avec une ontologie qui ne cadre plus avec la réalité. L'approche sémantique nous permet donc de mettre l'ontologie à plat et de tester la similarité sémantique de tous les couples de concepts de l'ontologie. Par contre, étant donné qu'on travaille avec des ontologies larges, calculer la distance sémantique de tous les couples de concepts semble quelque peu utopique en terme de complexité algorithmique et de temps d'exécution. Il devient donc primordial de se tourner vers d'autres alternatives de résolution qui nous permettraient de considérer tout de même la structure de l'ontologie au lieu de l'ignorer complètement.

La structure et la sémantique de l'ontologie sont des aspects dont on ne saurait faire abstraction lors de la modularisation. La structure nous permet guider le processus de recherche de concepts sémantiquement pertinents, et ce en ciblant concepts et paires de concepts dont la proximité sémantique se doit d'être évaluée. De plus, la structure initiale de l'ontologie guide le processus d'ajout de la descendance des concepts jugés pertinents au potentiel module.

CHAPITRE II

APPROCHE DE MODULARISATION COMET

Ce chapitre décrit l'approche hybride COMET qui intègre à la fois les approches structurelles et sémantiques. Le processus de modularisation avec COMET se fait en 2 étapes : la segmentation de l'ontologie et l'extraction du module. L'algorithme de segmentation a pour but de délimiter l'ontologie, tandis que l'algorithme d'extraction génère un fichier contenant la sous-ontologie retournée après la phase de segmentation. Le fonctionnement des algorithmes de COMET sera présenté plus en détails ainsi que les différentes composantes de son architecture.

2.1 Algorithme de segmentation

La segmentation est la première phase de notre approche d'extraction. Elle est basée sur l'hypothèse que lorsqu'un concept est pertinent, alors ses sous-concepts le sont également (Stuckenschmidt et Schlicht, 2009). La phase de segmentation consiste donc à déterminer les limites de notre module. Pour ce faire, l'utilisateur doit avant tout rentrer une liste de termes, ensuite définir un seuil sémantique et une profondeur hiérarchique. Une fois tous ces paramètres définis, l'algorithme va dans un premier temps chercher tous les concepts correspondants à la liste de termes et les sauvegardera dans une pile préalablement définie : il

s'agit de l'initialisation. Cette pile contenant les concepts de départ sera le noyau initial du module. Le but de cette approche est d'étendre le noyau au fur et à mesure que l'on découvre des concepts sémantiquement proches. Ainsi pour chaque concept, on vérifiera qu'il possède des relations sémantiques sortantes de type objet (Algorithm 1). Dans le cas où il en a, alors pour chacune de ces relations, on recherchera le concept auquel il est connecté à travers des propriétés du domaine et on évaluera la distance entre ces deux concepts. Dans la mesure où celle-ci est supérieure ou égale au seuil sémantique fixé, le nouveau concept ainsi découvert sera rajouté à la pile contenant la liste des concepts pertinents (Algorithm 2). Puis, en fonction de la profondeur hiérarchique définie par l'utilisateur, la descendance du concept jugé pertinent sera également rajoutée à la pile. Dans le cas où la distance sémantique serait inférieure au seuil et ne trouverait ainsi pas de concept sémantiquement lié au concept pertinent déjà présent dans la pile, l'algorithme fera appel à la hiérarchie de l'ontologie afin de vérifier si notre concept initial à des sous-concepts. Si des sous-concepts existent, alors pour chacun d'entre eux l'algorithme recommencera de nouveau à chercher les relations sémantiques. Puis, s'il en existe, il évaluera les distances sémantiques inter-concepts par rapport au seuil. Dans le cas où il n'existe pas de relations sémantiques ou si le seuil n'est toujours pas atteint, l'algorithme ira chercher encore plus en profondeur dans la hiérarchie jusqu'à ce qu'il trouve un concept ayant des relations sémantiques satisfaisantes.

Il s'agit d'un algorithme de segmentation récursif qui tend à agrandir le noyau initial à chaque itération jusqu'à ce que le seuil au-delà duquel on estimera qu'il n'existe plus de proximité sémantique soit atteint (Monjanel, 2011). L'on ne saurait dire à l'avance quelle sera la taille du module, car celle-ci dépend à la fois du seuil sémantique et de la profondeur hiérarchique fixés par l'utilisateur. Dans la situation où il n'existerait aucune relation sémantique, l'algorithme se baserait uniquement sur la profondeur hiérarchique pour générer le module, en ce

sens que la descendance des concepts initiaux sera rajoutée à la pile contenant les concepts pertinents correspondant à la signature. Dans l'approche que nous proposons, la profondeur hiérarchique représente le nombre d'arcs entre un concept et ses sous-concepts. Pour une profondeur de 2, on rajoutera systématiquement les sous-concepts des sous-concepts des concepts pertinents.

Pour résumer notre approche, on pourrait dire que l'algorithme COMET parcourt l'ontologie horizontalement (à travers les relations sémantiques) afin de trouver les concepts sémantiques proches des concepts de départ afin de former un noyau, mais l'ajout des nouveaux concepts à ce noyau initial se fait verticalement (à travers les relations hiérarchiques). Néanmoins, le choix du seuil reste assez complexe, car une question persiste : sur quel base définit-on le seuil ? Et en rajoutant les concepts à partir des relations hiérarchiques de l'ontologie, on se retrouve indubitablement à inclure dans notre module des concepts plus spécifiques dont on n'a forcément pas besoin. D'où les questions supplémentaires suivantes : quand estime-t-on qu'un concept est structurellement proche ? À quel niveau s'arrête-t-on dans la hiérarchie ? Ainsi, nous proposons que le seuil sémantique et la profondeur hiérarchique soient fixés arbitrairement par l'expert qui, au fil des tests (modules retournés par l'algorithme) pourra définir lesquels conviendraient le mieux en fonction du résultat recherché.

Le calcul de la distance sémantique se fait à l'aide de la distance de *Wu et Palmer*, qui est une mesure pratique et intuitive qui repose sur la longueur du chemin entre deux concepts d'une même hiérarchie. Elle calcule la distance séparant deux concepts dans la hiérarchie en fonction de leur position par rapport à la racine (Wu et Palmer, 1994). Il est évident que deux concepts se trouvant à la même profondeur dans la hiérarchie auront une similarité plus élevée que des concepts se trouvant à différents niveaux de la hiérarchie. La mesure de *Wu et*

Palmer se définit comme suit :

$$sim(c_1, c_2) = \frac{2 * depth(c)}{depth_c(c_1) + depth_c(c_2)}$$

où c est le subsumant commun le plus spécifique, $depth(c)$ est la longueur du chemin entre c et la racine de la hiérarchie, $depth_c(c_i)$ est le nombre d'arcs entre c_i et la racine passant par c. Cette mesure est comprise entre 0 et 1.

Algorithm 1 List concept outgoing object properties

Require: Ontology $\mathcal{O}$

Require: oc : OntClass (oc $\in \mathcal{O}$)

Ensure: outProp : ArrayList<OntProperty>

```
1: procedure GETOUTGOINGOBJECTPROPERTIES(O, oc)
2:     properties : Iterator<ObjectProperty>
3:     properties ← listObjectProperties(O)
4:     while (properties.hasNext()) do
5:         (ObjectProperty) p ← properties.next()
6:         if p.getRange() ≠ 0 then
7:             if p.getDomain() = oc then
8:                 outProp.add(p)
9:             end if
10:        end if
11:    end while
12:    return outProp
13: end procedure
```

Algorithm 2 COMET segmentation algorithm

Require: Ontology $\mathcal{O}$

Require: domain : OntClass

Ensure: relevantClasses : Vector<OntClass>

```
1: procedure COMPUTERELEVANTCLASSES(domain)
2:     relations ← GETOUTGOINGOBJECTPROPERTIES(O, domain)
3:     if (relations.size() > 0) then
4:         for each op ∈ relations do
5:             range ← op.getRange()
6:             if range ≠ 0 then
7:                 if SemanticDistance(domain, range) > threshold then
8:                     if relevantClasses.contains(range) then
9:                         continue
10:                    else
11:                        relevantClasses.add(range)
12:                    end if
13:                else
14:                    if domain.hasSubclass() then
15:                        relevantSubClasses ← domain.listSubClasses()
16:                        for each c ∈ relevantSubClasses do
17:                            COMPUTERELEVANTCLASSES(c)
18:                        end for
19:                    end if
20:                end if
21:            else
22:                relevantClasses.remove(range)
23:            end if
24:        end for
25:    else
```

Algorithm 2 COMET segmentation algorithm (continued)

```
26:         if domain.hasSubclass() then
27:             relevantSubClasses ← domain.listSubClasses()
28:             for each c ∈ relevantSubClasses do
29:                 COMPUTERELEVANTCLASSES(c)
30:             end for
31:         end if
32:     end if
33:     return relevantClasses
34: end procedure
```

2.2 Algorithme d'extraction

Après la segmentation, s'en suit la seconde phase du processus de modularisation qui est l'extraction de la sous-ontologie ainsi délimitée et stockée en mémoire. Pour ce faire, on procède à un élagage du graphe (arbre) représentant l'ontologie en parcourant celui-ci dans son ensemble afin de supprimer les concepts non-pertinents. Il est important de préciser que l'on travaille avec deux piles de données, l'une contenant l'ensemble des concepts pertinents et l'autre l'ensemble des concepts jugés non-pertinents. Lorsqu'un concept est supprimé de l'ontologie, toutes les relations sémantiques et hiérarchiques qui le lient à d'autres concepts le sont également. Une fois que l'élagage terminé, la nouvelle ontologie sera exportée dans un fichier. L'élagage s'avère de loin être la technique la plus adaptée pour la phase d'extraction, en ce sens qu'il aurait été nettement plus complexe de sauvegarder à la fois les concepts pertinents ainsi que leurs différentes propriétés dans des structures de données (du type liste) et uniquement à partir de celles-ci de générer le module ontologique.

La figure suivante illustre le processus de modularisation à partir de COMET dans son ensemble, avec pour paramètres d'entrée : une liste de termes composée de a et b, un seuil sémantique fixé à 0.4, une profondeur hiérarchique de 1. L'ontologie à modulariser est composée de 15 concepts. Au final, nous obtenons un module de 7 concepts. Le processus de modularisation commence à partir des concepts a et b. Chacun de ces concepts est pris séparément afin de détecter dans l'ontologie les concepts pertinents. d est un sous-concept de a, de ce fait, il est rajouté à la liste des concepts pertinents. d est lié à e par une relation sémantique dont le poids est égal au seuil fixé au départ. Ceci permet de rajouter les concepts e à la liste. Bien que e ait plusieurs sous-concepts, uniquement le concept g sera rajouté car la profondeur hiérarchique a été fixée à 1 (uniquement les sous-concepts directs sont rajoutés). f est lié à b par une relation sémantique dont le poids est de 0.5 ; ce qui est supérieur au seuil. En rajoutant le concept f ses sous-concepts directs g et i le sont également. Hors, g fait déjà partie dans la liste et un concept présent dans celle-ci ne pourrait y être rajoute à nouveau. De ce fait, uniquement i sera rajouté à la pile.

2.3 Implémentation du prototype COMET

Avant l'implémentation de COMET, nous avons tout d'abord modélisé son architecture et défini tous les modules qui le composent, puis nous avons sélectionné les technologies et outils adéquats qui nous permettraient de réaliser cette tâche aisément. La manipulation de ces technologies et outils a été apprise lors de l'élaboration de ce projet de recherche. Il va sans dire que les différents problèmes liés à l'incompatibilité de certains de ces outils et technologies ont fait partie des difficultés techniques à surmonter.

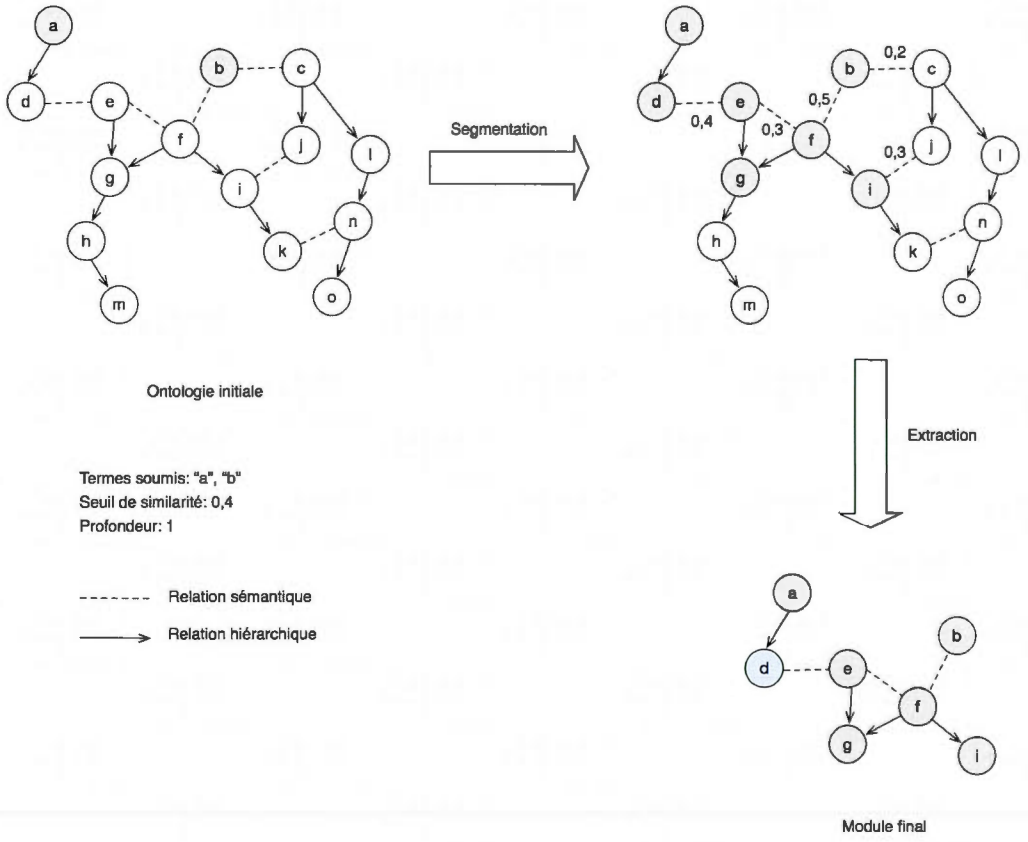

Figure 2.1: Processus de modularisation avec COMET.

2.3.1 Architecture de COMET

L'architecture de COMET repose sur quatre grandes entités (packages) que nous avons développées. Il s'agit des packages *extract*, *measure*, *util* et *test* (Figure 3.2).

Le package *extract*

Dans ce package on retrouve les classes *Modularizer* et *pruner* qui contiennent les algorithmes de segmentation et d'extraction de module. *Modularizer* a pour

but de segmenter l'ontologie $\mathcal{O}$ rentrée par l'utilisateur et ce, en fonction du seuil sémantique et de la profondeur hiérarchique qu'il aura fixé. *Modularizer* implémente l'algorithme de segmentation $computeRelevantClasses(c)$ où c est un concept pertinent à partir duquel d'autres seront découverts. Au fur et à mesure que de nouveaux concepts pertinents seront détectés, ils seront systématiquement rajoutés au vecteur $relevantClasses : Vector < OntClass >$: il s'agit d'une structure de données pouvant contenir des éléments de type *OntClass*. *Modularizer* fait appel à la méthode $getOutgoingObjectProperties(\mathcal{O}, c)$ afin de lister toutes les relations de type objet d'un concept c donné. La classe *pruner* quant à elle implémente des méthodes nous permettant de générer la liste de concepts non pertinents qui contient en effet tous les concepts n'appartenant pas à la liste de concepts pertinents retournée par la méthode $computeRelevantClasses(c)$. Ensuite avec la méthode $getModule()$, l'arbre de l'ontologie sera élagué en supprimant tout concept appartenant à la liste de concepts non pertinents. Une fois l'élagage terminé, la nouvelle ontologie sera exportée dans un fichier `owl` à l'aide de la méthode $ExportModule()$.

Le package *measure*

Ce package contient les méthodes de calcul de distances structurelle et sémantique. La méthode $getStructuralDistance()$ permet de calculer le chemin le plus court entre deux concepts. Ce calcul est basé sur l'algorithme de *Dijkstra* qui est implémenté dans la libraire JUNG. La méthode $getSemanticSimilarity()$ calcule la distance sémantique entre deux concepts sur la base de la mesure de *Wu et Palmer* qui est implémentée dans la libraire SimPack API. Ces deux méthodes sont appelées à chaque itération de la fonction $computeRelevantClasses(c)$ du package *extract* afin de générer la liste de concepts pertinents.

Le package *util*

La classe *OntologyHelper* contient un ensemble de méthodes permettant de charger une ontologie et d'y effectuer diverses opérations sur ses concepts, notamment le calcul de l'ensemble de relations sortantes de type objet d'un concept donné. Ce calcul se fait à l'aide de la méthode $getOutgoingObjectProperties(\mathcal{O}, oc)$. Cette méthode est appelée à chaque fois qu'un concept pertinent est détecté afin de parcourir les propriétés de type objet de celui-ci dans l'optique de découvrir de nouveaux concepts sémantiquement proches qui pourraient tout aussi être pertinents. La classe *Settings* permet d'indexer le chemin des fichiers avec lesquels on travaille. Elle contient aussi une méthode permettant de fixer les valeurs du seuil sémantique et de la profondeur hiérarchique par défaut.

Le package *test*

ExtractModule est la classe principale de notre application. Elle permet d'exécuter COMET. Elle fait appel aux méthodes des packages *extract* et *util*. Les méthodes qu'elle implémente permettent d'afficher en plus de la taille de l'ontologie, les relations sémantiques et les concepts parcourus lors du processus de segmentation. Une fois les listes de concepts pertinents et non-pertinents générées, elles sont affichées ainsi que le nombre de concepts que chacune d'elle contient.

2.3.2 Les technologies et les langages utilisés

JAVA

Java est à la fois un langage de programmation orienté-objet et une plateforme d'exécution JRE. Java est une technologie de plus de quinze ans qui est très utilisée dans l'industrie et la recherche. Java a pour principal avantage d'être

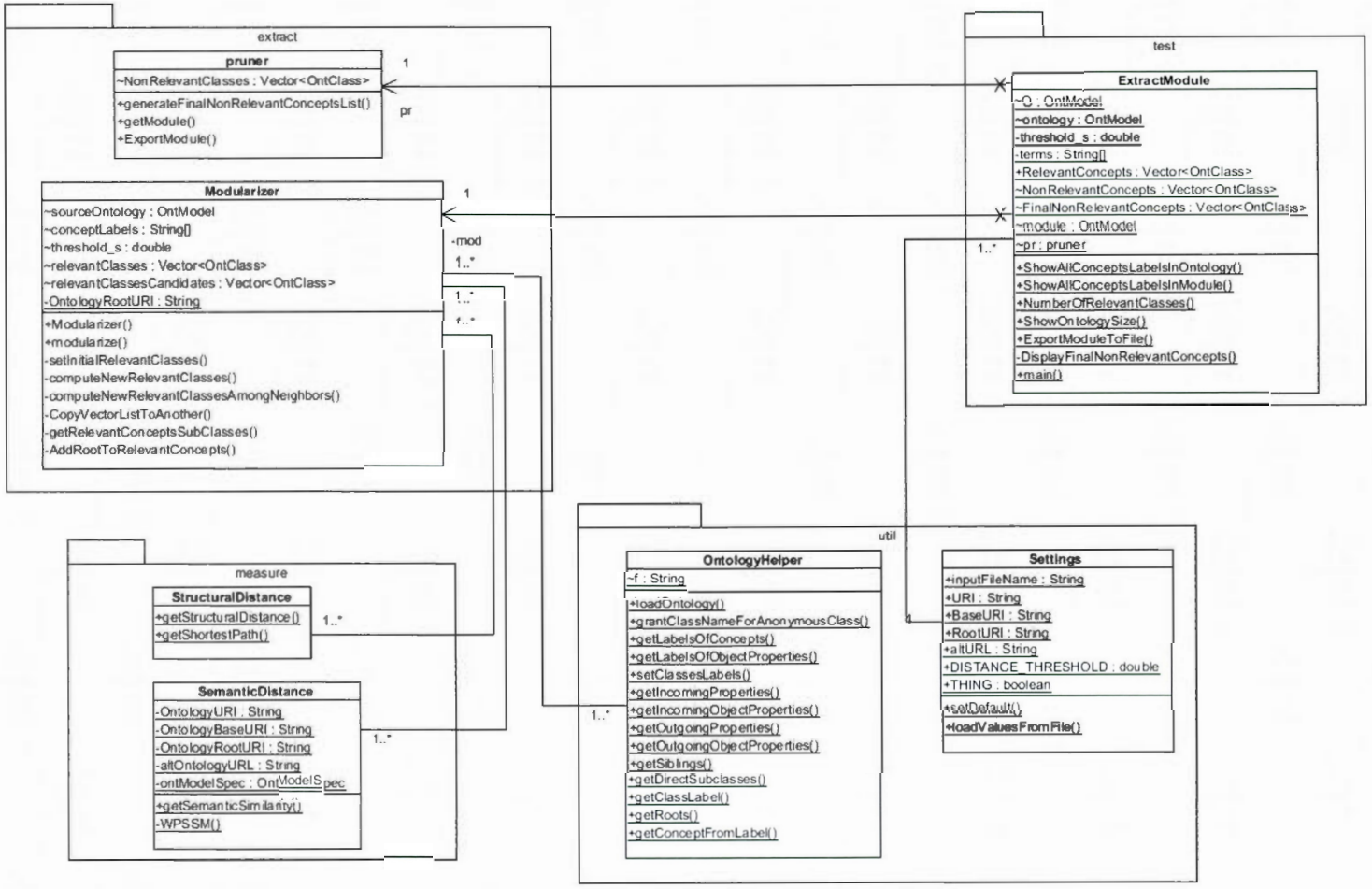

Figure 2.2: Diagramme de classes de COMET.

multi-plateforme, c'est-à-dire que toute plateforme disposant d'une machine virtuelle est susceptible de pouvoir exécuter une application conçue en Java. Depuis quelques années, il est parmi les langages les plus utilisés pour le développement d'applications de bureau, mais la nouveauté est que sont apparus récemment de nombreuses applications Web très performantes avec le cloud computing et les téléphones intelligents. Java s'est donc vu doté de nombreuses bibliothèques permettant le développement d'applications Web. Les avantages de Java pour notre application sont les suivants :

- C'est un langage de programmation objet ce qui permet de rendre les applications très modulaires et évolutives.
- C'est un langage qui permet d'utiliser des outils de gestion de version, ce qui permet de faciliter le développement en équipe.
- Il existe dans ce langage une bibliothèque reconnue par la communauté des chercheurs qui permet la manipulation des ontologies : JENA.

JENA API

JENA est une bibliothèque Java open-source et gratuite pour le Web sémantique. Elle permet les manipulations courantes des structures sémantiques écrites dans les langages spécifiés par le W3C comme standards pour ces structures : RDF, RDFS, OWL et DAML + OIL. JENA est actuellement le cadre de travail idéal pour la plupart des équipes de recherche ou des développeurs pour le Web sémantique. La raison en est qu'elle respecte le mieux les normes énoncées par le W3C quant à l'écriture des méta-données et des structures comme les ontologies. JENA bénéficie d'une équipe de développeurs dynamiques qui n'ont de cesse de l'améliorer.

SimPack API

Simpack API est un framework qui permet de calculer la similarité entre les concepts d'une ontologie ou entre plusieurs ontologies. Elle dispose d'un ensemble de mesures telles que les mesures de similarité de *Wu et Palmer*, de *Resnik*, ou celle de *Jiang et Conrath*. Ces mesures s'appuient sur la théorie des graphes, des ensembles et des vecteurs. Les méthodes de calcul de cette API sont implémentées en Java et sont génériques. Les mesures qu'elle implémente sont applicables à différentes structures.

JUNG API

JUNG est une librairie qui fournit un langage de modélisation, d'analyse et de visualisation de données qui peuvent être représentées sous forme de graphe ou de réseau. L'architecture de JUNG supporte une grande variété de représentations d'entités et des relations qui les lient, comme les graphes dirigés et non dirigés, les graphes à arcs parallèles pour n'en citer que ceux-la. De plus, JUNG implémente divers algorithmes issus de la théorie des graphes, de la fouille de données et d'analyse de réseaux. Enfin, la visualisation est accessible par cette bibliothèque de fonctions. JUNG implémente l'algorithme de *Dijkstra* qui nous permet de calculer la distance structurelle, c'est-à-dire le chemin le plus court entre deux concepts.

OWL

OWL est un langage permettant d'écrire, de publier et de partager des ontologies sur le Web. OWL est basé sur RDF(S) et est inspiré du langage DAML + OIL. En février 2004, il devient le langage standard de W3C. Si RDF(S) permet la description des classes et des propriétés, OWL apporte plus d'expressivité notamment avec la possibilité de comparer des classes et des propriétés.

Protégé

Protégé est un outil permettant d'éditer et de créer des ontologies. Il a été élaboré à l'Université de Stanford et est très populaire dans le domaine du Web sémantique ainsi qu'au niveau de la recherche académique. Il est développé en JAVA et est offert sous licence GPL et à code source libre. L'outil protégé permet de lire et de sauvegarder des ontologies dans la plupart des formats d'ontologies connus : RDF, RDF(S), et OWL.

OntoEval API

OntoEval API est un framework permettant d'évaluer la qualité des ontologies. OntoEval contient des métriques d'évaluation d'ontologies basées sur les algorithmes de précision et rappel de Dellschaft et Staab (Dellschaft et Staab, 2006). Elle prend en entrée deux ontologies définies OWL ou RDF(S). La première est une ontologie de référence (Gold Standard ontology) et la seconde est l'ontologie dont on veut évaluer la qualité. Il s'agit d'une approche d'évaluation basée sur la comparaison.

OntoMetricsAPI

OntoMetricsAPI, tout comme OntoEval, est un framework qui permet d'évaluer la qualité des ontologies. Les métriques implémentées dans OntoMetricsAPI sont celles d'Alani, d'Orme et de Tartir. Ces mesures jouent un rôle important dans notre protocole de validation. Nous verrons plus en détails comment ces métriques s'appliquent à un module ontologique dans le prochain chapitre.

2.3.3 Présentation de cas de modularisation avec COMET

Exemple 1

Dans le but d'illustrer le fonctionnement de l'outil COMET, nous avons choisi de modulariser l'ontologie `camera.owl` (Figure 3.4) qui contient 12 concepts. Dans ce cas de figure, nous souhaitons obtenir un module à partir des termes *camera* et *SLR*. Le seuil et la profondeur hiérarchique sont fixés respectivement à 0,1 et à 1. Les relations sémantiques parcourues ainsi que les concepts qu'elles indexent sont systématiquement affichés à chaque itération de l'algorithme de segmentation (Figure 3.3).

Lors de la phase d'initialisation de l'algorithme. Tous les concepts correspondants à la liste de termes rentrée sont rajoutés à la pile de concepts pertinents. Ainsi, les concepts *camera* et *SLR* sont systématiquement rajoutés. L'algorithme COMET listera ensuite tous les sous-concepts des concepts présents dans la pile, et ce en fonction de la profondeur fixée par l'utilisateur au départ. Étant donné que nous travaillons avec une profondeur hiérarchique de 1, uniquement les sous-concepts directs seront listés et rajoutés : c'est le cas des concepts *Large-Format* et *Digital* qui sont des sous-concepts de *camera*. Les concepts *Body*, *Lens*, *Range* et *Viewer* sont rajoutes par le via les relations sémantiques, dont le poids est supérieur au seuil préalablement défini. Ces relations les lient aux concepts déjà présents dans la pile. Nous rappelons toutefois que l'outil COMET prend en entrée le fichier contenant l'ontologie à modulariser en format `.owl`. Une fois le processus de modularisation terminé, COMET génère un fichier `module_camera.owl` contenant ainsi la sous-ontologie extraite (Figure 3.5).

```
<terminated> ExtractModule (1) [Java Application] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Hon
 -> The property 'viewFinder' from concept 'Camera' to 'Viewer' is added
 -> The property 'body' from concept 'Camera' to 'Body' is added
 -> The property 'lens' from concept 'Camera' to 'Lens' is added
 -> The property 'cost' from concept 'Camera' to 'Money' is added
 -> The property 'cost' from concept 'Body' to 'Money' is added
 -> The property 'shutter-speed' from concept 'Body' to 'Range' is added
 -> The property 'compatibleWith' from concept 'Lens' to 'Body' is added
 -> The property 'cost' from concept 'Lens' to 'Money' is added
Ontology size = 12
Ontology concepts name : [Viewer, Body, Money, SLR, Large-Format, Camera, Window, Lens, D
Number of Relevant Concepts = 8
Number of NonRelevant Concepts = 4
 *************************    RELEVANT CONCEPTS LIST:   *************************
[0] - SLR
[1] - Camera
[2] - Viewer
[3] - Body
[4] - Lens
[5] - Range
[6] - Digital
[7] - Large-Format
 *************************  NONRELEVANT CONCEPTS LIST:  *************************
[0] - Money
[1] - Window
[2] - BodyWithNonAdjustableShutterSpeed
[3] - PurchaseableItem
 -> The module is generated!
 Done in 50 msec.
```

Figure 2.3: Processus de modularisation de l'ontologie `camera.owl` avec COMET.

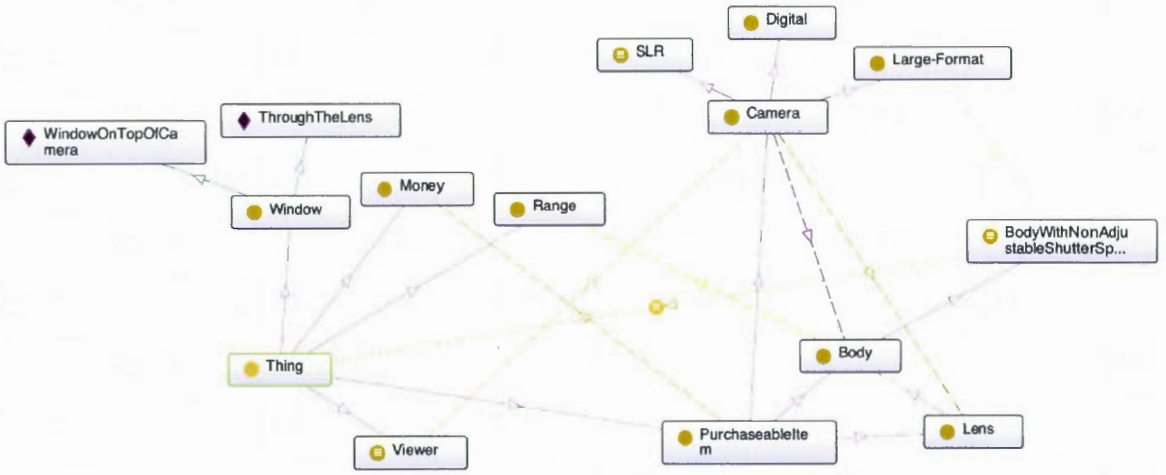

Figure 2.4: Schéma de l'ontologie modularisée `camera.owl`.

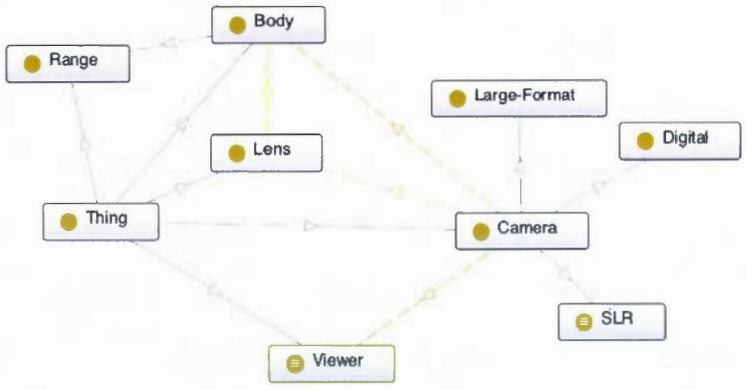

Figure 2.5: Schéma du module extrait `module_camera.owl`.

Exemple 2

Dans ce second cas, nous avons choisi de modulariser une ontologie un peu plus large que la précédente. Il s'agit de l'ontologie `conference.owl` qui contient 59 concepts (Figure 3.6). Nous voulons extraire un module à partir des termes *Submitted_contribution* et *Topic*. Le seuil sémantique est fixé à 0,2 et la profondeur hiérarchique à 1. Afin d'illustrer le processus de modularisation, nous avons de choisi de travailler avec une ontologie dense et riche en propriétés du domaine. De ce fait, la plupart des concepts présents dans le module ont été détectés via les relations sémantiques. Après analyse de la figure 3.7, on constate également que les 21 concepts qui constituent le module sont fortement interconnectés.

Dans ce chapitre, il a été question de présenter l'architecture et le fonctionnement de l'outil COMET. Afin de valider notre approche de modularisation, nous allons mettre sur pied un protocole de validation et réaliser des tests sur la base de celui-ci. Le protocole de validation des modules ontologiques que nous proposons repose sur l'usage de métriques de qualité qui permettent à un expert en génie de connaissances d'évaluer la qualité du module, soit par rapport l'ontologie d'origine, soit par rapport à une ontologie de référence. Toutefois, nous ne saurions présenter les résultats des tests menés sur COMET sans pour autant présenter les différentes approches d'évaluation qui existent. Aussi, le chapitre suivant aura pour but de faire un bref état de l'art sur les principales techniques d'évaluation qui sont le plus souvent utilisées dans le cadre de l'évaluation des ontologies.

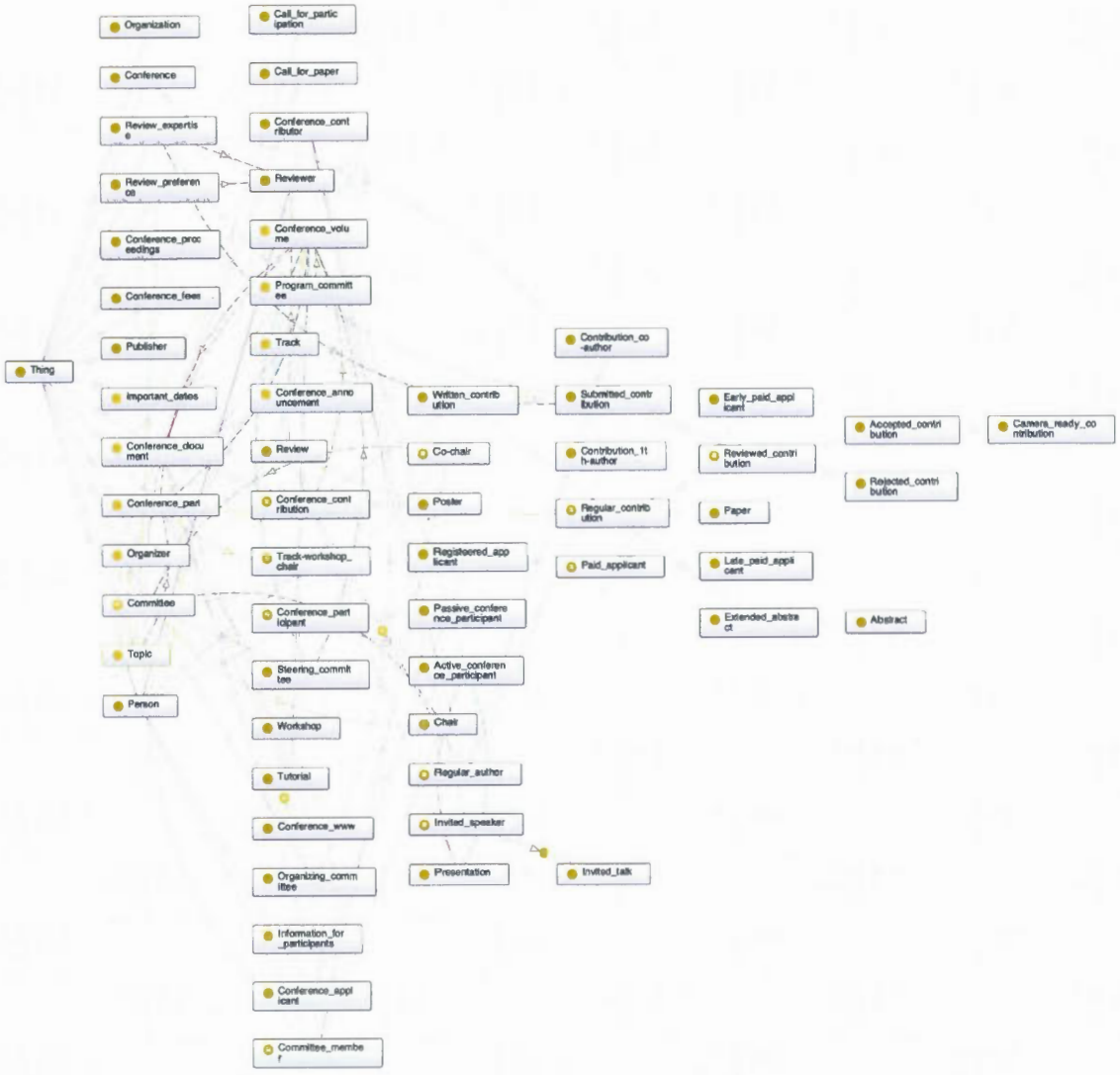

Figure 2.6: Schéma de l'ontologie modularisée `conference.owl`.

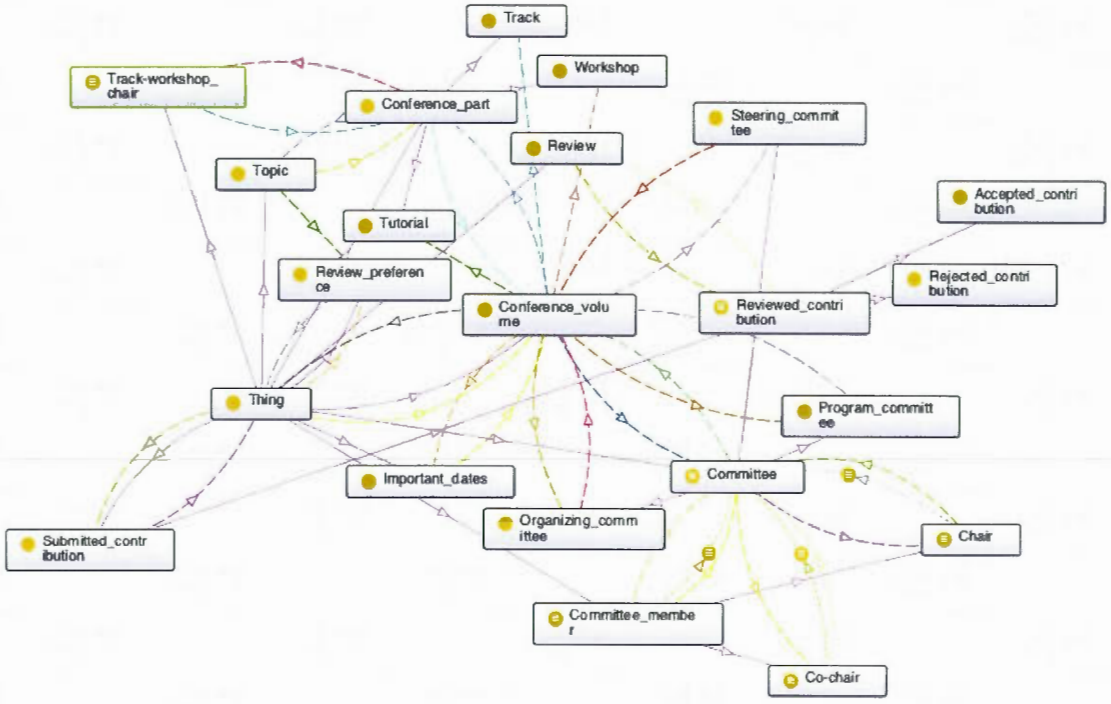

Figure 2.7: Schéma du module extrait `module_conference.owl`.

CHAPITRE III

LES DIFFÉRENTES APPROCHES D'ÉVALUATION DES ONTOLOGIES

Alors que la littérature regorge d'approches de modularisation des ontologies, peu d'efforts ont été consacrés à l'évaluation objective des modules extraits à partir des techniques présentées précédemment. Au delà de la taille qui reste l'un des principaux aspect à évaluer dans un module, il existe d'autres métriques évaluant des aspects tous aussi importants qui, au final, permettent de juger de la qualité d'une ontologie ou d'un module. En général, les métriques ont pour but de guider l'expert dans l'évaluation et l'appréciation de différents aspects d'un module et ce, en fonction de la tâche pour laquelle l'ontologie est modularisée. L'évaluation du module est un sujet assez complexe d'autant plus que dans divers travaux, on rencontre des différentes méthodes à savoir : des méthodes qualitatives, des méthodes quantitatives, et des méthodes mixtes d'évaluation d'ontologies.

3.1 Approches basées sur la qualité de l'ontologie

Le but des approches basées sur la qualité est d'établir un certain nombre de critères en fonction desquels un expert serait en mesure de dire si une ontologie est pertinente. On distingue trois principaux groupes de dimensions dans la littérature : les dimensions structurelle, fonctionnelle et applicative (Gangemi

et al., 2005). La dimension structurelle fait référence à la structure hiérarchique des ontologies représentées par des graphes. Ces approches ont pour but à ce niveau d'évaluer les relations hiérarchiques dans les différentes ontologies. Elles reposent essentiellement sur des mesures structurelles telles que la profondeur, la largeur, la densité, la modularité et la consistance. La dimension fonctionnelle est axée sur la sélection, la construction et l'exploitation d'une ontologie donnée et ses composants (concepts, relations hiérarchiques, rôles). La dimension applicative repose sur l'utilisabilité de l'ontologie et elle dépend du niveau d'annotation de celle-ci.

Les travaux menés sur des approches basées sur la qualité même de l'ontologie ont permis de dégager un certain nombre de méthodes d'évaluation d'ontologies. Ainsi, comme méthode fondée sur la qualité d'une ontologie, on pourrait tout d'abord citer O^2/oQual (Gangemi *et al.*, 2006). O^2 est une méta-ontologie sémiotique, c'est-à-dire un objet constitué par un objet d'information et une conceptualisation établie dans le cadre de transactions communicationnelles et de partage entre acteurs d'un même collectif. Le but de cette méta-ontologie est d'assurer l'utilisabilité des ontologies dans différentes applications. La méthode EvaLexon à la différence de la précédente a pour but d'évaluer les ontologies extraites à partir des textes lors de leur développement (Spyns, 2005). EvaLexon utilise la linguistique afin d'évaluer les termes les plus appropriés à une ontologie et ce, à partir des quatre métriques suivantes :

- La précision et le rappel qui permettent d'évaluer la capacité d'apprentissage d'une ontologie dans son ensemble.
- La couverture et l'exactitude qui permettent de comparer le vocabulaire de l'ontologie par rapport à la fréquence des termes du texte.

La méthode OntoClean quant à elle est fondée sur des notions de raisonnement (la rigidité, l'unité, l'identité et la dépendance) pour une évaluation formelle

des relations hiérarchiques de l'ontologie (Guarino et Welty, 2002).

3.2 Méthodes quantitatives d'évaluation des ontologies

L'évaluation quantitative est une tâche qui a une importance particulière dans les systèmes complexes de gestion de connaissances, où des erreurs peuvent apparaître à la fin d'un processus de traitement de l'information. Les méthodes quantitatives d'évaluation reposent, tout comme les méthodes qualitatives, sur les dimensions structurelle et sémantique des ontologies. Le framework OntoMetricsAPI qui a été élaboré au laboratoire GDAC(UQÀM) implémente les métriques d'Orme et *al.* qui nous permettent d'évaluer la cohésion et la complexité des ontologies. Les métriques proposées par Orme *et al.* sont principalement des mesures structurelles (Orme *et al.*, 2007) :

- Le nombre de propriétés (*Number of properties - NOP*) : c'est le nombre de propriétés définies de façon explicite dans une ontologie $\mathcal{O}$.

  $$NOP(\mathcal{O}) = \Sigma P_j$$

  pour tout $1 \leq j \leq n$, où j est le nombre de propriétés dans $\mathcal{O}$.
- Le nombre moyen des propriétés par classes (*Average Properties per Class - APC*) : c'est le rapport du nombre de propriétés par le nombre de classes de l'ontologie $\mathcal{O}$.

  $$APC(\mathcal{O}) = \frac{\Sigma P_j}{\Sigma C_k}$$

  pour tout $1 \leq j \leq m$ et $1 \leq k \leq n$, où j et k représentent respectivement le nombre de propriétés et le nombre de classes dans $\mathcal{O}$.
- La profondeur maximale de l'arbre d'héritage (*Maximum Depth of Inheritance Tree - MaxDIT*) : il s'agit du nombre total de concepts sur le chemin le plus long allant d'un concept racine à un concept feuille de l'ontologie $\mathcal{O}$. Il peut exister un ou plusieurs chemins allant d'un concept racine à un

concept branche, mais le *MaxDIT* représente le chemin qui passe par le plus de concepts.

$$MaxDIT(\mathcal{O}) = Max(D_j)$$

pour tout $1 \leq j \leq n$, où j est le nombre de chemins dans $\mathcal{O}$ et D_j la profondeur du j-ème chemin.

3.3 Approches basées sur la comparaison

L'évaluation d'une ontologie basée sur la comparaison peut se faire de deux façons. Dans le premier cas, la comparaison peut se faire par rapport à une référence et dans le deuxièmement cas, elle peut se faire par rapport à des ontologies existantes dont la qualité aurait déjà été évaluée et jugée satisfaisante.

3.3.1 Approche basée sur la comparaison par rapport à une référence

Le référentiel pourrait se définir comme étant un ensemble de termes construit manuellement par un expert afin d'établir les conditions du domaine pour lequel l'ontologie a été construite. Ainsi, le but de cette opération est de comparer un ensemble des termes retournés, suite à des requêtes faites sur une ontologie, à un ensemble des termes de référence. Afin donc de comparer l'ontologie de domaine extraite et l'ontologie de référence, la notion de similarité est introduite. La similarité se calcule à partir des métriques de similarité. Cette évaluation est faite aux niveaux lexical et conceptuel dans l'optique de trouver pour chaque concept dans une ontologie de référence, le(s) concept(s) similaire(s) dans l'ontologie à évaluer.

Le lexique renvoie aux concepts, aux instances et au vocabulaire de l'ontologie. Maedche et Staab ont mis au point une approche permettant d'évaluer

la qualité du lexique de l'ontologie en calculant la similitude entre deux termes (concepts) avec la distance d'édition *ed* (Maedche et Staab, 2002). Cette distance consiste à mesurer le nombre minimum d'insertions, de suppressions et de substitutions de caractères nécessaires pour transformer une chaîne de caractères en une autre. Soient l_i et l_j deux concepts, la similitude SM est définie telle que suit (plus SM est proche de 1, plus l_i et l_j sont similaires) :

$$SM(l_i, l_j) = \max(0, \frac{\min(|l_i|, |l_j|) - ed(l_i, l_j)}{\min(|l_i|, |l_j|)}) \in [0, 1]$$

Afin de comparer les niveaux lexicaux de deux ontologies, Maedche et Staab proposent de comparer leurs lexiques correspondants ($\mathcal{L}_1$ et $\mathcal{L}_2$) en calculant la mesure moyenne de l'appariement entre chaînes de caractères $\overline{SM}$ (Maedche et Staab, 2002) :

$$\overline{SM}(l_1, l_2) = \frac{1}{|\mathcal{L}_1|} \sum_{l_i \in \mathcal{L}_1} \max_{l_j \in \mathcal{L}_2} SM(l_i, l_j)$$

Le couche lexicale peut également être évaluée à partir de la précision lexicale LP et du rappel lexical LR.

Soient deux ontologies, $\mathcal{O}_{Ref}$ l'ontologie de référence et $\mathcal{O}_C$ l'ontologie à évaluer. Formellement, la précision et le rappel lexicaux de $\mathcal{O}_C$ par rapport à $\mathcal{O}_{Ref}$ se définissent comme suit :

$$LP(\mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{NTL(\mathcal{O}_{Ref})}{NT(\mathcal{O}_C)}(\%)$$

où $NTL(\mathcal{O}_{Ref})$ est le nombre de termes dans $\mathcal{O}_{Ref}$ et $NT(\mathcal{O}_C)$ est le nombre total d'éléments dans $\mathcal{O}_C$.

$$LR(\mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{NTL(\mathcal{O}_C)}{NT(\mathcal{O}_{Ref})}(\%)$$

où $NTL(\mathcal{O}_C)$ est le nombre de termes dans $\mathcal{O}_C$ et $NT(\mathcal{O}_{Ref})$ est le nombre d'éléments dans $\mathcal{O}_{Ref}$. Pour évaluer la couche taxonomique ou conceptuelle, Staab

et *al.* proposent de nouvelles métriques qui permettent de comparer les structures sémantiques de deux ontologies $\mathcal{O}_1$ et $\mathcal{O}_2$ (Dellschaft et Staab, 2006). Cette évaluation est basée sur une mesure moyenne de similarité *tp* entre deux taxonomies $\mathcal{H}_1^c$ et $\mathcal{H}_2^c$. La taxonomie étendue entre $\mathcal{H}_1$ et $\mathcal{H}_2$ est considérée comme l'ensemble des termes, référés par le lexique commun L. TO représente le résultat de la précision taxonomique locale *tp* si le lexique L fait parti du lexique de la deuxième taxonomie. TO' est le résultat si le lexique L ne fait pas parti du lexique de la deuxième taxonomie. Soient deux concepts c_1 et c_2, la précision taxonomique locale se calcule tel que suit :

$$tp_{ce}(c_1, c_2, \mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{|ce(c_1, \mathcal{O}_C) \cap ce(c_2, \mathcal{O}_{Ref})|}{ce(c_1, \mathcal{O}_C)}$$

avec $c_1 \in \mathcal{O}_C$, $c_2 \in \mathcal{O}_{Ref}$ et *ce* un extrait caractéristique.
La précision taxonomique TP se calcule par la formule suivante :

$$TP(\mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{1}{|\mathcal{L}_C|} \sum_{L \in \mathcal{L}_C} tp(L, \mathcal{O}_C, \mathcal{O}_{Ref})$$

avec

$$tp(L, \mathcal{O}_C, \mathcal{O}_{Ref}) = \begin{cases} TO(L, \mathcal{O}_C, \mathcal{O}_{Ref}), & \text{si } L \in \mathcal{L}_{Ref} \\ TO'(L, \mathcal{O}_C, \mathcal{O}_{Ref}), & \text{si } L \notin \mathcal{L}_{Ref} \end{cases}$$

On a :

$$TO(L, \mathcal{O}_C, \mathcal{O}_{Ref}) = tp(c, c, \mathcal{O}_C, \mathcal{O}_Ref)$$

$$TO'(L, \mathcal{O}_C, \mathcal{O}_{Ref}) = max_{c' \in \mathcal{L}_{Ref}} tp(c, c', \mathcal{O}_C, \mathcal{O}_Ref)$$

Une fois la précision taxonomique TP calculée, on peut facilement déduire le rappel taxonomique TR, car $TR(\mathcal{O}_C, \mathcal{O}_{Ref}) = TP(\mathcal{O}_{Ref}, \mathcal{O}_C)$. Les mesures taxonomiques TP et TR ont pour but d'évaluer la structure de l'ontologie $\mathcal{O}_C$ par rapport à celle de l'ontologie de référence $\mathcal{O}_{Ref}$, tout en tenant compte de leur lexique L.

Staab *et al.* et introduisent deux nouvelles métriques essentielles qui se basent sur les précédentes afin d'évaluer de façon plus générale la qualité d'une ontologie. Il s'agit des mesures taxonomiques TF et TF', qui se calculent comme suit :

$$TF(\mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{2.TP(\mathcal{O}_C, \mathcal{O}_{Ref}).TR(\mathcal{O}_C, \mathcal{O}_{Ref})}{TP(\mathcal{O}_C, \mathcal{O}_{Ref}) + TR(\mathcal{O}_C, \mathcal{O}_{Ref})}$$

$$TF'(\mathcal{O}_C, \mathcal{O}_{Ref}) = \frac{2.LR(\mathcal{O}_C, \mathcal{O}_{Ref}).TF(\mathcal{O}_C, \mathcal{O}_{Ref})}{LR(\mathcal{O}_C, \mathcal{O}_{Ref}) + TF(\mathcal{O}_C, \mathcal{O}_{Ref})}$$

Le rôle et l'importance des métriques de Staab *et al.* sont mis en exergue dans la dernière partie du chapitre suivant, notamment dans le cadre de l'évaluation des différents modules extraits avec les outils sélectionnés pour une étude comparative. Ces métriques sont implémentées dans l'outil d'évaluation d'ontologies OntoMetric Tool (Lozano-Tello et Gómez-Pérez, 2004). Le principal atout de ce système est qu'il permet de sélectionner l'ontologie la plus appropriée ou de prendre une décision au sujet de la pertinence d'une ontologie particulière en fonction d'un domaine donné. Par contre, OntoMetric Tool requiert de l'utilisateur une connaissance approfondie du schéma d'ontologies (le vocabulaire et l'ensemble des termes qui décrivent les propriétés des ontologies), car elle fait appel à un nombre élevé de caractéristiques de l'ontologie.

3.3.2 Approche basée sur la comparaison par rapport à d'autres ontologies

L'évaluation d'une ontologie peut se faire par comparaison avec des ontologies déjà existantes construites par apprentissage (*Ontology learning*). Dans cette optique, Brewster *et al.* présentent une approche qui a pour but d'évaluer la structure d'une ontologie par rapport à un corpus textuel propre à un domaine donné (Brewster *et al.*, 2004). Cette approche pourrait être assimilée à une évaluation

comparative entre des ontologies et des textes. L'évaluation se fait en comparant le nombre de termes existant entre l'ontologie et le corpus textuel. Pour ce faire, Brewster *et al.* proposent des approches probabilistes afin de réaliser cette évaluation et ce, en identifiant la meilleure ontologie $\mathcal{O}^*$ qui se calcule comme suit :

$$\mathcal{O}^* = \arg\max_{\mathcal{O}} P(\mathcal{O}|C) = \arg\max_{\mathcal{O}} \frac{P(C|\mathcal{O})P(\mathcal{O})}{P(C)}$$

où $\mathcal{O}$ est l'ontologie à évaluer, C le corpus textuel et $P(C|\mathcal{O})$ est la probabilité de l'existence du corpus C dans l'ontologie $\mathcal{O}$.

L'expression $\arg\max_{\mathcal{O}} P(\mathcal{O}|C)$ renvoie au fait que si une ontologie maximise la probabilité conditionnelle de l'ontologie $\mathcal{O}$ donnée par le corpus C, alors elle est considérée comme la meilleure ontologie $\mathcal{O}^*$.

3.4 Approches basées sur l'utilité d'une ontologie

En partant du principe qu'une ontologie se définie comme étant une base de connaissances, celle-ci peut donc être utilisée par plusieurs applications. Chacune des applications utilisant la même ontologie peut produire un résultat différent, car celui-ci dépend de la tâche à réaliser. Ainsi, évaluer une ontologie reviendrait donc à évaluer le résultat de l'application qui l'utilise en fonction d'une tâche précise. Soulignons toutefois que cette évaluation se fait à l'aide d'un ensemble de termes T. Alani *et al.* proposent plusieurs métriques afin d'évaluer et classer les différentes ontologies (Alani et Brewster, 2005). Ces métriques sont toutes des mesures structurelles :

- La centralité (CEM) : Elle permet de calculer le nombre de chemins les plus courts passant par chaque nœud du graphe (on considère que l'ontologie est représentée par un graphe). Les nœuds qui sont les plus fréquentés par les chemins les plus courts auront une centralité plus élevée.

Soient c_i et c_j deux classes de l'ontologie $\mathcal{O}$ et $C[\mathcal{O}]$ l'ensemble des classes de $\mathcal{O}$, $cem(c)$ est la centralité pour une classe quelconque c.

$$cem(c) = \sum_{c \neq c_i \neq c_j \in C[\mathcal{O}]} \frac{\sigma_{c_i c_j}(c)}{\sigma_{c_i c_j}}$$

où $\sigma_{c_i c_j}$ est le chemin le plus court de c_i à c_j, $\sigma_{c_i c_j}(c)$ est le nombre de chemins les plus courts allant de c_i à c_j et passant par c.

$$CEM(\mathcal{O}) = \frac{1}{n} \sum_{k=1}^{n} cem(c_k)$$

où $n = E(\mathcal{O}, T) + P(\mathcal{O}, T)$, $CEM(\mathcal{O})$ la moyenne de centralité pour une ontologie $\mathcal{O}$, $E(\mathcal{O}, T)$ et $P(\mathcal{O}, T)$ représentent le nombre de classes de $\mathcal{O}$ dont les noms (labels) respectifs correspondent exactement ou partiellement à une liste de termes prédéfinie.

- La similarité sémantique (SSM) : elle permet de calculer la similarité des classes qui correspondent aux termes recherchés. Elle est aussi basée sur le calcul du chemin le plus court. Soient c_i et c_j deux classes de l'ontologie $\mathcal{O}$, et $c_i \overset{p}{\rightsquigarrow} c_j$ est un chemin $p \in P$. P représente l'ensemble des chemins entre c_i et c_j.

$$ssm(c_i, c_j) = \begin{cases} \frac{1}{length(\min_{p \in P}\{c_i \overset{p}{\rightsquigarrow} c_j\})}, & \text{si } i \neq j \\ 0, & \text{si } i = j \end{cases}$$

$$SSM(\mathcal{O}) = \frac{1}{n} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} ssm(c_i, c_j)$$

où $n = E(\mathcal{O}, T) + P(\mathcal{O}, T)$ et $ssm(c_i, c_j)$ le nombre de chemins de longueur minimale entre c_i et c_j.

- La correspondance des classes (CMM) : elle permet d'évaluer la capacité d'une ontologie à couvrir un ensemble de termes donnés. Soient $C[\mathcal{O}]$ un ensemble de classes dans une ontologie $\mathcal{O}$, et T un ensemble de termes

recherchés.

$$E(\mathcal{O},T) = \sum_{c \in C[\mathcal{O}]} \sum_{t \in T} I(c,t)$$

$$I(c,t) = \begin{cases} 1, & \text{si } label(c) = t \\ 0, & \text{si } label(c) \neq t \end{cases}$$

$$P(\mathcal{O},T) = \sum_{c \in C[\mathcal{O}]} \sum_{t \in T} J(c,t)$$

$$J(c,t) = \begin{cases} 1, & \text{si } label(c) \textit{ contient } t \\ 0, & \text{si } label(c) \textit{ ne contient pas } t \end{cases}$$

où $E(\mathcal{O},T)$ et $P(\mathcal{O},T)$ représentent le nombre de classes de $\mathcal{O}$ dont les noms (labels) respectifs correspondent exactement ou partiellement aux termes recherchés t.

$$CMM(\mathcal{O},\tau) = \alpha E(\mathcal{O},T) + \beta P(\mathcal{O},T)$$

où α et β sont les facteurs de poids attribués respectivement à une correspondance exacte ou partielle.

- La densité (DEM) : elle permet d'approximer la densité de l'information contenue dans les différentes classes de l'ontologie. Soit $S = \{S_1, S_2, .., S_i, ..., S_6\}$ le nombre de relations directes et indirectes, d'instances, de super-classes, de sous-classes et de classes-sœurs d'une classe quelconque c appartenant à une ontologie $\mathcal{O}$.

$$dem(c) = \sum_{i=1}^{6} w_i |S_i|$$

$$DEM(\mathcal{O}) = \frac{1}{n} \sum_{i=1}^{n} dem(c)$$

où w_i est le facteur poids dont la valeur par défaut est de 1 et $n = E(\mathcal{O},T) + P(\mathcal{O},T)$.

- Le score total (*Score*) : il se calcule une fois les quatre métriques précédentes obtenues. Différents poids sont attribués à ces métriques en fonction de l'importance que l'expert accorde à chacune d'elles. La métrique la plus importante aura un poids plus élevé que celui des trois autres. La somme des quatre poids doit être égale à 1. Soient $M = \{M[1], ..., M[i], M[4]\} = \{CMM, CEM, DEM, SSM\}$, w_i un facteur poids et $\mathcal{O}$ un ensemble d'ontologies. Le score se calcule par la formule suivante :

$$Score[o \in \mathcal{O}] = \sum_{i=1}^{4} w_i \frac{M[i]}{max_{1 \leq j \leq |\mathcal{O}|} M[j]}$$

  Les résultats des métriques d'Alani et *al.* sont normalisés de telle sorte que les valeurs retournées soient comprises entre 0 et 1.

L'approche fondée sur l'utilité d'une ontologie d'Alani *et al.* est implémentée dans le système d'évaluation et de classement d'ontologies *AKTiveRank*. Les requêtes sont faites à l'aide des moteurs de recherche d'ontologies tels que *Swoogle* et *Watson*. L'ontologie retournée par le moteur de recherche est ensuite récupérée par *AKTiveRank* afin d'identifier dans celle-ci les concepts correspondants aux termes d'entrée de la requête de l'utilisateur (Alani *et al.*, 2006).

Les métriques d'Alani et d'Orme ont été implémentées dans un framework OntoMetricsAPI qui est intégré à l'outil de validation de la plateforme *INUKHUK*. Étant donné que COMET fera aussi partie de cette plateforme, nous avons donc choisi de faire appel à ce framework afin d'évaluer l'ensemble des modules générés par les outils d'extraction de modules basés sur les approches présentées dans le chapitre 2. Le prochain chapitre portera uniquement sur la validation de l'approche hybride COMET. Des tests seront menés sur plusieurs ontologies de tailles variables et relatives à différents domaines afin d'étudier les performances des outils sélectionnés, tout en mettant l'accent sur la particularité de chacun d'entre eux. Cette étude est réalisée dans l'optique de nous permettre de distinguer l'ou-

til de modularisation qui produit les modules satisfaisant le plus les critères de qualité et surtout de comprendre le contexte dans lequel on obtiendrait de tels résultats.

CHAPITRE IV

EXPÉRIMENTATIONS ET ÉVALUATION DE COMET

Un module est une ontologie, et en tant que tel, les approches d'évaluation de qualité des ontologies s'appliquent tout autant à lui. Bien que certaines techniques d'évaluation analysent la structure représentant l'ontologie à partir de son contenu, d'autres techniques quant à elles se focalisent essentiellement sur la comparaison du contenu même de l'ontologie, soit avec une autre ontologie existante dont la qualité aurait été jugée satisfaisante, soit avec une autre alternative de représentation du domaine de connaissances dont il est question, telle qu'un corpus de documents. Cette comparaison est faite dans l'optique de déterminer à quel point ce contenu modélise l'ensemble des aspects pertinents du domaine à décrire. En se basant sur ces principes et dans le but de définir notre protocole de validation, on s'attellera avant tout à définir nos critères d'évaluation et par la suite, nous établirons le procédé suivant lequel les différents modules extraits seront évalués.

4.1 Critères d'évaluation d'un module

Étant donné qu'il existe différentes approches de modularisation et d'outils, il est donc évident qu'il faille définir un certain nombre de critères qui permettront à l'expert de déterminer si un module est plus pertinent qu'un autre. Parmi ces

critères d'évaluation, on distingue :

- L'exactitude : c'est l'un des critères les plus importants qui doit être satisfait par le module et l'ontologie dont il est extrait. Chaque axiome présent dans un module $\mathcal{M}_i(O)$ doit également l'être dans l'ontologie modularisée $\mathcal{O}$ (d'Aquin *et al.*, 2009).
- La taille : c'est le nombre de classes, de propriétés et d'individus qu'une ontologie contient. C'est l'un des indicateurs les plus pertinents qui permette d'apprécier l'efficacité d'une technique de modularisation. Néanmoins, il est à noter que la taille d'un module a non seulement une forte influence sur la maintenance de celui, mais aussi sur l'efficacité des applications qui l'utilisent, en ce sens qu'un surplus d'information dans un module entraine indubitablement une perte de flexibilité dans son utilisation et son évolution. D'un autre côté, un module très petit ne peut couvrir suffisamment le domaine qui nous intéresse.
- La cohésion : elle sert à déterminer le degré de connectivité entre les différents éléments présents dans un même module. Il existe un ensemble de métriques, entre autres celles d'Orme *et al.* qui permettent d'évaluer la cohésion d'un module, notamment le nombre de classes racines dans une hiérarchie, le nombre de propriétés par classes, et la profondeur maximal de la hiérarchie d'héritage (Yao *et al.*, 2005; Orme *et al.*, 2007).
- La couverture du domaine : c'est un critère tout aussi important qui permet d'évaluer le degré de représentativité d'un domaine d'application par un module précis. Afin d'être en mesure de déterminer la couverture d'un domaine, il faut tout d'abord être capable de définir une représentation exacte du domaine à couvrir par le module en question (d'Aquin *et al.*, 2009).

4.2 Protocole de validation

Le protocole de validation repose essentiellement sur une étude comparative des modules obtenus par COMET à ceux générés par les outils OWL Module Extractor et SegmentationApp. Il s'agit en fait d'une évaluation du type *Gold standard* avec pour référence l'ontologie dont les différents modules sont extraits. Le but de notre protocole de validation est d'évaluer tant l'aspect lexico-sémantique que l'aspect structurel des différents modules générés. Et pour ce faire, en plus de OntoMetricsAPI nous ferons également appel à l'API OntoEval pour calculer la précision et le rappel des modules en fonction de ceux obtenus par l'ontologie de référence. Chacune de ces API prend en entrée deux paramètres : l'ontologie de référence et l'ontologie générée ou calculée qui, en fait, est dans notre cas le module extrait.

Le processus d'évaluation par les API se fait donc en trois phases. Lors de la première, l'API évalue l'ontologie source suivant les métriques d'Alani *et al.*, d'Orme *et al.* et de Staab *et al.*. Cette évaluation consiste tout d'abord à calculer les métriques de qualité de l'ontologie source, ensuite à attribuer le score maximal qui est de 100% à chaque résultat de métrique retourné. Cette attribution systématique d'un score maximal est justifié par le fait que notre ontologie source est considérée comme étant une ontologie de référence, c'est-à-dire une ontologie dont la qualité est jugée excellente.

À la seconde phase, les modules sont évalués à leur tour à l'aide de ces mêmes API. Les métriques sont calculées pour chacun des modules. Les résultats obtenus lors des calculs des métriques de qualité du module sont comparés à ceux de l'ontologie de référence. En fonction des score attribués à celles de l'ontologie de référence, les scores des métriques du module sont calculés. Afin d'illustrer cette approche d'évaluation, considérons une ontologie $\mathcal{O}$ et un module $\mathcal{M}$ dont

on voudrait calculer la densité *DEM*. À l'issu du calcul de la densité de l'ontologie initiale $\mathcal{O}$, nous obtenons un résultat $DEM_{\mathcal{O}} = 12{,}22$; tandis que le calcul de la densité du module donne un résultat $DEM_{\mathcal{M}} = 5{,}21$. En faisant le rapport de la densité du module par celle de l'ontologie de référence, on obtient le score de la densité du module. Dans notre cas, nous obtenons un score de 42,63%.

$$Score_{DEM} = \frac{DEM_{\mathcal{M}}}{DEM_{\mathcal{O}}}(\%)$$

La dernière phase de l'évaluation consiste tout simplement à comparer les scores des métriques des modules générés par COMET, OWL Module Extractor et SegmentationApp entre eux afin de déterminer lequel est le plus pertinent en fonction des métriques utilisées.

Pour résumer le protocole de validation, nous comparons chacun des modules générés à partir les différents outils d'extraction avec l'ontologie source afin d'obtenir des scores. Puis, nous comparons les scores des modules obtenus entre eux afin de déterminer le score le plus élevé. Ce sont les modules obtenant les meilleurs scores de métriques ou ayant le meilleur score total qui nous permettront d'évaluer les performances d'un outil d'extraction. Il est évident que le module qui héritera au maximum des caractéristiques de l'ontologie de référence se retrouvera avec les scores les plus élevés. Les scores des métriques dépendent des facteurs tels que le nombre de concepts, le nombre de propriétés et de la taxonomie du module ou de l'ontologie de référence pour n'en citer que ceux là.

Dans le tableau 5.1 est listé l'ensemble des ontologies[1] avec lesquelles nous travaillerons dans le cadre de la validation de COMET. La taille et le nombre de propriétés de ces ontologies sont également indiqués. Le tableau 5.2 représente l'ensemble des ontologies modularisées, la signature du module (la liste des termes

1. `http://oaei.ontologymatching.org/2007/conference/`

à couvrir), les outils d'extraction ainsi que la taille des ontologies et des différents modules extraits à partir d'elles. Pour les expérimentations, nous avons fixé la profondeur hiérarchique de COMET à 1 et le seuil sémantique S quant à lui est variable.

Tableau 4.1: Liste d'ontologies à modulariser à l'aide des différents outils d'extraction.

#	Ontologies	Taille	Propriétés de type objet	Propriétés de type donnée	Individus
1	sweto_simplified	115	13	56	0
2	cmt	29	49	10	0
3	confOf	38	13	23	0
4	iasted	140	38	3	4
5	Conference	59	46	18	0
6	edas	103	31	20	114
7	paperdyne	45	61	21	8
8	OpenConf	62	24	21	18
9	ekaw	73	33	0	0
10	sigkdd	49	17	11	0
11	travel	34	6	4	14
12	PPOntology	88	17	12	865
13	OTN	179	36	75	0

Tableau 4.2: Extraction de modules à l'aide des approches de Seidenberg *et al.*, de Cuenca *et al.* et COMET.

					COMET						CUENCA			
			SEIDENBERG		*S = 0.2*		*S = 0.1*		*S = 0.0*		*Top*		*Bottom*	
Ontologies	#C	Termes	#C	Ratio	#C	Ratio	#C	Ratio	#C	Ratio	#C	Ratio	#C	Ratio
sweto_simplified	115	`Person, Event`	4	3,48%	21	18,26%	21	18,26%	21	18,26%	13	11,30%	4	3,48%
cmt	29	`Co-author, Review`	7	24,14%	3	10,34%	24	82,76%	24	82,76%	28	96,55%	7	24,14%
confOf	38	`Participant, Conference`	10	26,32%	5	13,16%	5	13,16%	5	13,16%	38	100%	10	26,32%
iasted	140	`Delegate, Activity, Place, Fee, Deadline, Submission`	48	34,29%	56	40%	56	40%	56	40%	133	95%	54	38,57%
Conference	59	`Submitted_contribution, Topic`	10	16,95%	21	35,59%	23	38,98%	23	38,98%	40	67,80%	11	18,64%
edas	103	`Author, Paper, Workshop, Programme, SocialEvent, Call, Country, Document, ReviewRating, Rejectedpaper`	20	19,42%	46	44,66%	47	45,63%	47	45,63%	103	100%	24	23,30%
paperdyne	45	`Conference, Location`	18	40%	17	37,78%	18	40%	18	40%	45	100%	2	4,44%
OpenConf	62	`Paper_Review, People`	22	35,48%	7	11,29%	7	11,29%	29	46,77%	62	100%	58	93,55%
ekaw	73	`Review, Possible_Reviewer`	4	5,48%	15	20,55%	15	20,55%	15	20,55%	72	98,63%	4	5,48%
sigkdd	49	`Author, Award, Sponzor`	16	32,65%	20	40,82%	21	42,86%	21	42,86%	41	83,67%	16	32,65%
travel	34	`Destination, AccommodationRating`	1	2,94%	15	44,12%	15	44,12%	15	44,12%	33	97,06%	25	73,53%
PPOntology	88	`Case, Biotic_Disorder, Organism, Disorder, Mineral, Plant_Observation, Pesticide, Abnormality, Bacterium, Roots`	17	19,32%	33	37,50%	41	46,59%	41	46,59%	88	100%	87	98,86%
OTN	179	`Feature, Service, Road_Element, Manoeuvre, Temperature, Face, Tourism, Accident, Forest, Junction`	25	13,97%	59	32,96%	59	32,96%	59	32,96%	171	95,53%	52	29,05%
		Moyenne		**21,11%**		**29,77%**		**36,70%**		**39,43%**		**88,12%**		**36,31%**

4.3 Analyse des résultats

Afin de déterminer l'outil d'extraction le plus efficace, nous évaluerons chaque module extrait des ontologies listées dans le tableau 5.1 suivant trois ensembles de métriques : les métriques d'Alani *et al.*, les métriques d'Orme *et al.*, et les métriques de Staab *et al.*. Dépendamment de l'ontologie à modulariser et des modules générés, il sera donc question de calculer le score de chacune des métriques utilisées, puis de comparer les scores des modules obtenus entre eux.

Dans le cadre de la validation de COMET, nous avons choisi d'évaluer uniquement les modules ayant pour seuil sémantique $S = 0,2$ et $S = 0$ afin d'éviter toute redondance dans nos calculs, simplifiant ainsi le processus d'évaluation. Nous justifions ce choix par le fait que nous avons constaté dans plusieurs cas que les modules ayant pour seuil $S = 0,1$ et $S = 0$ étaient identiques.

En ce qui concerne les modules extraits par l'outil OWL Module Extractor (approche d'extraction de Cuenca *et al.*), nous évaluerons uniquement les modules dont l'extraction est basée sur la $\top$-localité (*bottom modules*), car la taille moyenne des modules dont l'extraction est basée sur la $\bot$-localité (*top modules*) représente 88,12% de la taille des ontologies sources. La taille d'un module est un critère important sinon le plus important et nous constatons qu'au regard des ontologies sélectionnées pour le protocole de validation, la plupart des top-modules issus de ces ontologies sont trop volumineux.

4.3.1 Évaluation des modules en fonction des métriques d'Alani *et al.*

Dans le but d'évaluer les aspects structurel et sémantique des ontologies, Alani *et al.* propose un ensemble de métriques. La combinaison des résultats de

ces métriques nous permettent de calculer un score global ou *rank* en fonction duquel la qualité globale des modules générés est évaluée. Les métriques proposées par Alani *et al.* sont : la correspondance des classes (*CMM*), la densité (*DEM*), la similarité sémantique (*SSM*) et la centralité (*CEM*)(Alani *et al.*, 2006). Chacune de ses métriques a pour but d'évaluer un aspect précis du module ontologique.

La correspondance des classes évalue la capacité d'une ontologie de couvrir un ensemble de termes donnés. Dans notre cas, elle a pour but de calculer le pourcentage du matching tant exact que partiel entre les classes du module et celles de l'ontologie de référence, afin de trouver le degré de représentativité des classes de l'ontologie initiale dans le module extrait. Il est évident que plus le module est grand, plus il est susceptible de couvrir l'ensemble de concepts (correspondant à la liste de termes recherchés) présent dans l'ontologie initiale. Après analyse du tableau B.1, nous remarquons que dans la plupart des cas d'ontologies étudiées, les modules de COMET obtiennent un meilleur score du *CMM*, ce qui sous-entend que les modules qu'il génère sont plus représentatifs de l'ontologie initiale.

La densité exprime le degré de précision d'un concept donné, c'est-à-dire la richesse de ses attributs. Cette définition implique qu'une représentation adéquate d'un concept doit être à même d'offrir un maximum d'information sur ce dernier. La densité dépend essentiellement du nombre de sous-classes, du nombre d'attributs associés à ce concept ou encore du nombre de concepts frères. Dans le cadre de nos expérimentations, on constate que les modules générés à l'aide de COMET ont un score du *DEM* nettement plus élevé que celui des modules extraits à partir des autres outils (Figure 5.1). Ceci s'explique par le fait que COMET parcourt l'ontologie horizontalement par les relations sémantiques afin de rechercher les concepts pertinents et une fois ceux-ci identifier, la hiérarchie est parcourue afin de rajouter leur descendance au module. Toutefois, on rappellera que dans le cas de l'outil COMET la profondeur hiérarchique est un paramètre qui, tout comme le

seuil sémantique, est défini par l'utilisateur. C'est à travers ce paramètre que l'expert est en mesure de déterminer la limite à laquelle l'algorithme devrait s'arrêter lors du parcours de la taxonomie. La profondeur hiérarchique va de paire avec le niveau de spécialisation, qui est un facteur important dans le calcul de la densité. Soulignons toutefois que, plus la profondeur hiérarchique de COMET est élevée, plus le score du *DEM* du module sera élevé ; car comme il a été mentionné plus haut, la densité d'une ontologie dépend de son degré de granularité. La densité ne dépend pas du nombre de concepts dans le module. Cette hypothèse est vérifiée dans les cas des ontologies 8, 11 et 12, où on remarque que, bien que la taille des top-modules extraits avec OWL Module Extractor représentent environs 75% à 95% de celle des ontologies sources, leur score, dans les cas où on a un seul de 0, reste néanmoins inférieur à celui des modules extraits avec COMET dont la moyenne de la taille reste largement inférieure à 50% (Figure 5.1).

La similarité sémantique calcule la proximité entre les classes correspondant aux termes recherchés dans l'ontologie. Aussi, les concepts correspondant à ces termes se doivent d'être liés soit par des relations hiérarchiques, soit par des relations sémantiques. Cette mesure est basée sur le calcul du plus court chemin entre les paires de concepts. On constate que dans la majorité des cas, les modules générés avec COMET ont un score du *SSM* élevé, ce qui va dans le sens de l'approche de COMET où la détection de concepts pertinents se fait à travers le parcours des propriétés de type objet. Plus il y aura des relations sémantiques dans le module, plus il y aura de chemins différents entre en les paires de concepts de celui-ci. De même, plus les concepts d'un module sont interconnectés aussi bien par des relations sémantiques que par des relations hiérarchiques, plus le score du *SSM* de ce module sera élevé.

La centralité mesure le degré de représentativité de l'ensemble des concepts correspondants aux termes recherchés dans une ontologie. Elle se base sur le calcul

du chemin le plus court passant par chaque concept de l'ontologie. Les concepts les plus sollicités lors du parcours de l'ontologie auront une centralité plus élevée que celle des autres concepts. Au regard des résultats illustrés sur la figure 5.1, on note que dans 6 des 13 cas d'ontologies étudiées, les modules générés par COMET ont un score du *CEM* nettement supérieur que celui des modules générés par OWL Module Extractor et SegmentationApp. Nous pouvons justifier ce résultat par le fait que ces deux derniers outils ont produit des modules ayant peu de relations sémantiques, réduisant ainsi le nombre de chemins entre les paires de concepts. Ce constat nous amène à conclure que le parcours de la structure lors du calcul du *CEM* des modules de OWL Module Extractor et SegmentationApp se fait principalement à travers les relations hiérarchiques. Il est important de rappeler que la densité, la centralité et la similarité sémantique sont des mesures structurelles qui reposent essentiellement sur le degré d'interconnexion des concepts dans le module.

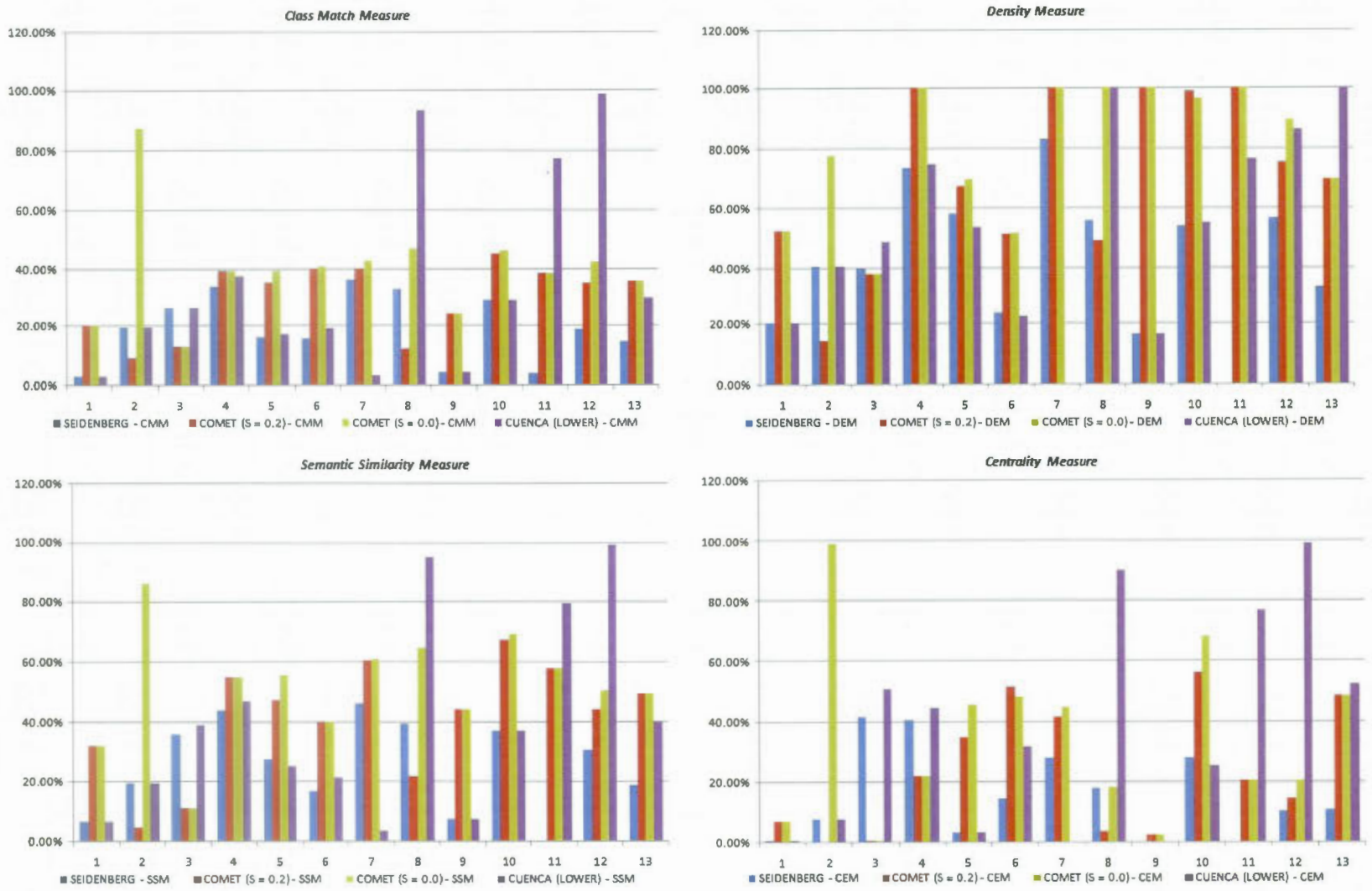

Figure 4.1: Score des modules en fonction des métriques d'Alani *et al.* .

Alani et *al.* proposent une méthode pour calculer le score total d'une ontologie qui consiste à attribuer des poids à chacune des métriques calculées précédemment (Alani et Brewster, 2005). L'attribution des poids se fait en fonction de l'importance accordée à chacune des métriques. L'expert fixera le poids le plus élevé à la métrique qu'il considère comme étant la plus importante. La somme des poids doit être égale à 1. Aussi, nous avons choisi d'étudier quatre scénarios dans lesquels nous avons fait varier les poids des métriques (Tableau B.2).

Dans le scénario 1, nous considérons que la correspondance des classes et la densité sont toutes deux prioritaires. Pour cela, nous répartissons les poids de la façon suivante : $0.4CMM, 0.4DEM, 0.2SSM, 0.0CEM$. Dans ce cas de figure, nous estimons que la centralité n'a aucun impact sur la qualité globale du module.

Dans le scénario 2, les poids sont attribués équitablement à chaque métrique, d'où la répartition suivante : $0.25CMM, 0.25DEM, 0.25SSM, 0.25CEM$. Nous estimons qu'aucune métrique n'est plus importante qu'une autre.

Dans le scénario 3, la priorité est accordée à la correspondance des classes, ensuite vient la densité, puis la similarité et au final la centralité. Les poids quant à eux sont attribués tels que suit : $0.4CMM, 0.3DEM, 0.2SSM, 0.1CEM$.

Dans le scénario 4, on considère la similarité sémantique comme étant la métrique la plus importante, tandis que la centralité reste la moins importante, d'où la répartition suivante : $0.2CMM, 0.3DEM, 0.4SSM, 0.1CEM$.

Après analyse de la figure 5.2, nous constatons que dans la majorité des ontologies étudiées, plus précisément dans 8 des 13 cas d'ontologies utilisées, COMET a produit les modules dont la qualité est la plus satisfaisante suivant les mesures d'Alani *et al.*. La profondeur hiérarchique et le seuil sémantique sont des paramètres essentiels qui nous permettent de contrôler la taille des modules.

Plus une ontologie est riche en relations de type objet, plus l'algorithme de COMET sera susceptible de retourner des modules plus larges. D'où la nécessité pour l'utilisateur de définir un seuil sémantique, afin de contrôler les proportions des modules.

Nous remarquons, après analyse des résultats retournés par les différentes métriques d'Alani *et al.*, que les valeurs les plus élevés sont celles obtenues par COMET avec un seuil de similarité $S = 0$. Ce résultat peut certes prêter à équivoque quant au bien-fondé de l'utilisation du seuil dans cette approche. Rappelons toutefois que la modularisation avec COMET est un processus assisté au cours duquel l'utilisateur se doit de définir respectivement les valeurs du seuil et de la profondeur hiérarchique. En fonction des résultats retournés lors des différents tests, ce dernier sera amené à faire varier les valeurs de ces paramètres afin de raffiner le module et obtenir au final celui qu'il jugera satisfaisant. Il est important d'insister sur le fait que chacune des métriques a pour but d'évaluer un aspect précis du module, et c'est à l'utilisateur qu'il revient la responsabilité de définir quels selon lui sont les critères les plus importants ; d'où l'élaboration des différents scénarios afin d'évaluer qualité globale de l'ontologie.

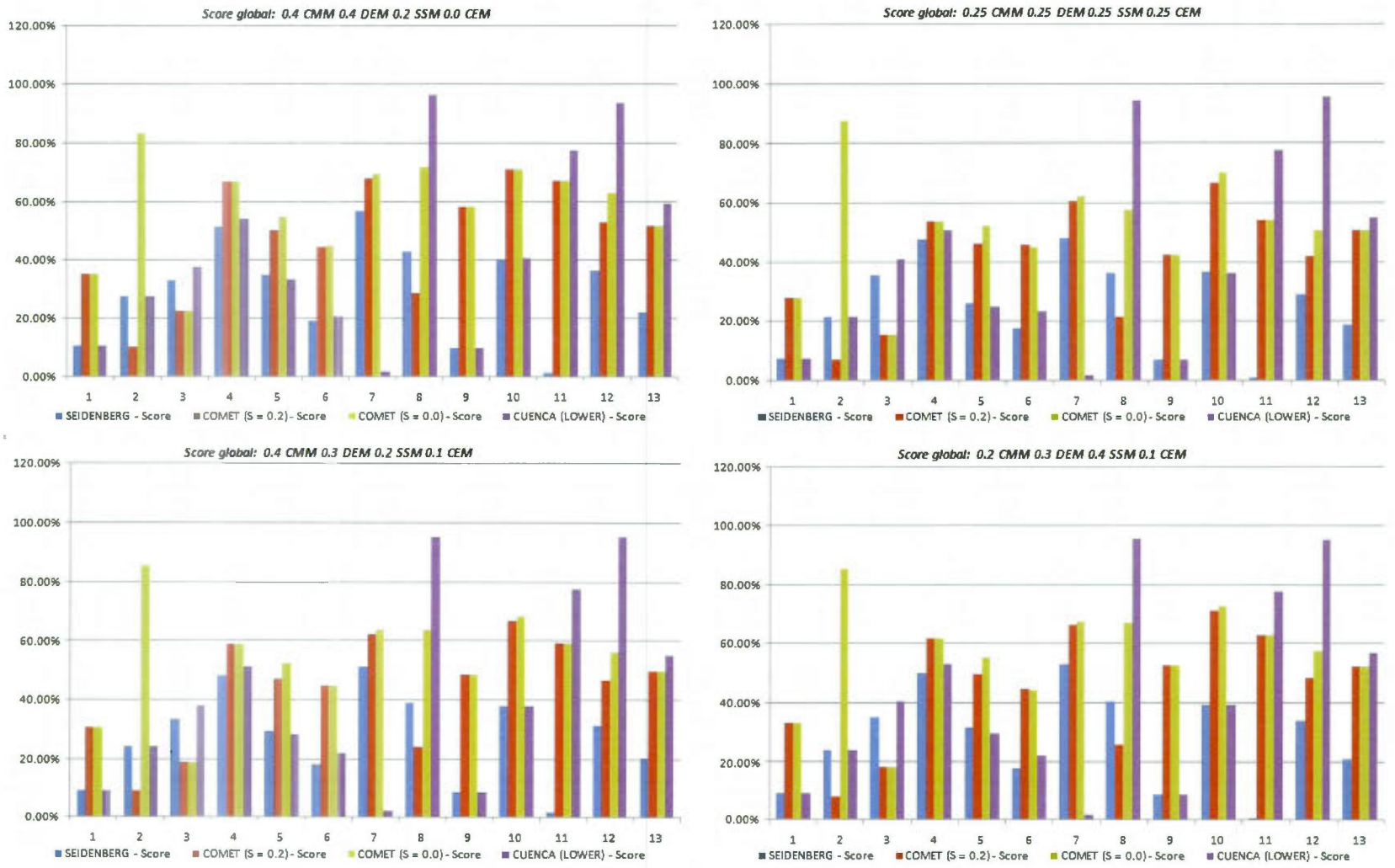

Figure 4.2: Score total des modules en fonction de la variation des poids.

4.3.2 Évaluation des modules en fonction des métriques d'Orme *et al.*

Orme *et al.* proposent un ensemble de métriques inspirées de celles utilisées dans le génie logiciel afin d'évaluer les ontologies. Parmi ces métriques, on distingue : le nombre de propriétés *NOP*, la moyenne de propriétés par classes *APC* et la profondeur maximale de l'arbre d'héritage *MaxDIT* (Orme *et al.*, 2007).

Le nombre de propriétés d'un module nous renseigne sur deux aspects importants qui sont le degré de complexité et la complétude de la définition de l'ensemble des classes du module ontologique. Une classe possédant un *NOP* élevé est une classe entièrement décrite, on peut donc déduire qu'une ontologie composée essentiellement de classes clairement définies sera une ontologie avec un *NOP* tout aussi élevé. Par propriété, on fait référence à l'ensemble des propriétés de type objet de l'ontologie. Le tableau B.3 représente les résultats des tests effectués suivant les métriques d'Orme *et al.* sur chacun des modules générés. On remarque que le score de *NOP* de COMET est de 100% dans tous les cas de figure et ce, même en considérant les cas où COMET a généré des modules de taille moindre (Figure 4.3). Ce phénomène s'explique par le fait que toutes propriétés présentent dans l'ontologie d'origine le sont également dans le module généré. Ce qui, une fois de plus, va dans le sens même de la logique de l'approche COMET, car son module est avant tout composé de concepts découverts à partir des relations sémantiques. L'algorithme de COMET recherche dans l'ontologie toutes les paires de concepts liés par une propriété de type objet en fonction du seuil. Rappelons que le seuil sémantique a un impact sur la taille du module. De ce fait, un seuil faible entraînerait systématiquement un *NOP* bas.

La moyenne de propriétés par classe permet d'évaluer à quel point chacune des classes de l'ontologie a été définie. C'est le rapport du nombre de propriétés

dans l'ontologie par le nombre de concepts de celle-ci. Un *NOP* élevé entraîne indubitablement un *APC* élevé. Dans le cadre de nos expérimentations dont les résultats sont listés dans le tableau B.3, on note que le score de l'*APC* des modules de COMET sont de 100%, car on se retrouve avec des modules dont l'*APC* est nettement supérieure à l'*APC* de l'ontologie source, entre autre, des modules dont le nombre de propriétés est égal à celui de l'ontologie de référence.

La profondeur maximale de l'arbre d'héritage nous permet d'évaluer le degré de granularité des classes de l'ontologie. Cette profondeur est en fait le nombre total de nœuds à travers lesquels passe le chemin allant du nœud racine au nœud feuille le plus bas dans la taxonomie de l'ontologie. Cette définition implique que le score du *MaxDIT* dépend donc uniquement de la profondeur du module en question. Dans notre cas, cette mesure s'avère être subjective en ce sens qu'il revient à l'utilisateur de COMET de fixer la profondeur hiérarchique afin de préciser jusqu'à quelle limite l'algorithme pourrait se rendre lors du parcours de la descendance des concepts pertinents. Plus la profondeur hiérarchique de COMET est élevée, plus le score du *MaxDIT* sera élevé. Dans le cadre de nos expériences, la profondeur hiérarchique a été fixée à 1 justifiant ainsi les scores obtenu pas COMET. Après analyse de la figure 5.3, on remarque qu'OWL Module Extractor et SegmentationApp ont obtenu des scores élevés dans la majorité des cas étudiés.

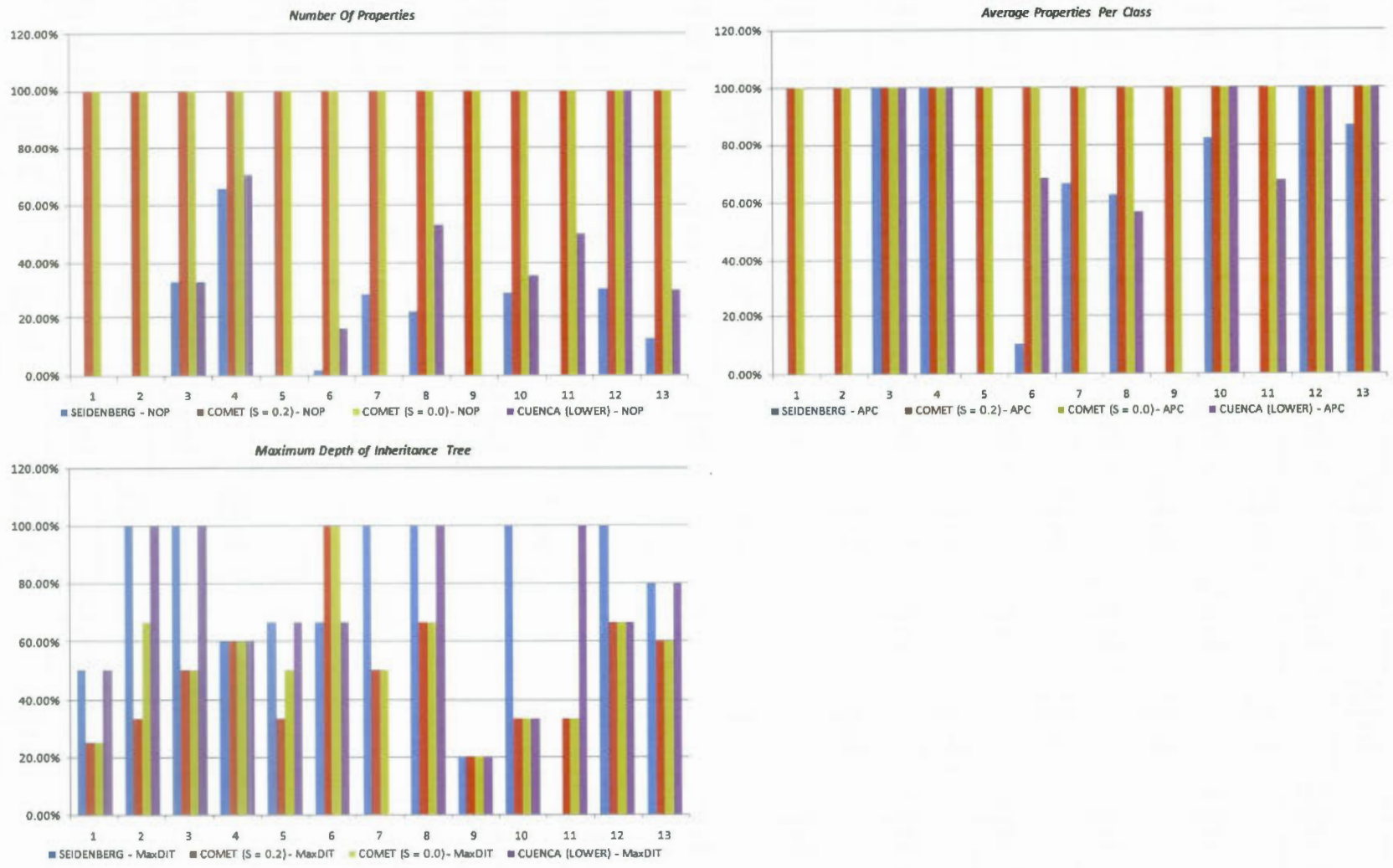

Figure 4.3: Score des modules en fonction des métriques d'Orme *et al.* .

4.3.3 Évaluation des modules en fonction des métriques de Staab *et al.*

L'approche de Staab *et al.* a pour but d'évaluer aussi bien le contenu lexical que la structure de l'ontologie. L'évaluation du contenu lexical consiste à comparer l'ensemble des termes de l'ontologie de référence à celui de l'ontologie générée afin de calculer leur correspondance (Dellschaft et Staab, 2006). L'évaluation de la structure quant à elle consiste à comparer la structure de l'ontologie calculée à celle de notre référentiel. Dans notre protocole de validation, évaluer les couches lexical et structurelle revient à calculer la précision et le rappel du module par rapport à ceux de l'ontologie source. Staab *et al.* proposent six métriques d'évaluation d'ontologies : la précision lexicale *LP*, le rappel lexical *LR*, la précision taxonomique *TP*, le rappel taxonomique *TR*, la F-mesure taxonomique *TF* et la F'-mesure taxonomique *TF'*.

La précision lexicale est le rapport du nombre total des termes lexicaux de l'ontologie de référence par le nombre d'éléments présents dans notre module. Lors de nos tests, on a constaté que ce rapport est de 100% quelque soit le module et l'ontologie. Ce résultat nous amène donc à conclure qu'il y a une correspondance lexicale parfaite entre les termes contenus dans le module et ceux de l'ontologie de référence à laquelle ce dernier est comparé. Il en est de même pour la précision taxonomique qui évalue la correspondance de la structure du module à celle de l'ontologie source. On notera que quelque soit le l'outil d'extraction utilisé dans nos expérimentations, la structure des modules générés ne s'en trouve aucunement altérée. Étant donné que nous avons obtenu les scores maximaux avec les différents outils d'extraction lors des calculs du *LP* et *TP*, nous avons estimés qu'il n'était donc pas nécessaire de les rajouter dans le tableau B.4.

Le rappel lexical est le rapport du nombre de termes lexicaux du module

par le nombre de termes lexicaux de l'ontologie de référence. On se rend très vite compte qu'en fait il s'agit du rapport du nombre de concepts du module sur celui de l'ontologie source. Après l'analyse du tableau B.4, on peut conclure que plus le module est grand, plus le rappel sera élevé. D'une manière générale, on remarque que COMET nous donne un meilleur rappel lexical *LR* dans la plupart des cas d'ontologies étudiés (Figure 5.4). Le rappel taxonomique *TR* est une mesure permettant d'évaluer la correspondance entre la position d'un concept dans la taxonomie du module à celle de ce même concept dans la taxonomie de l'ontologie de référence. Ainsi, le module qui se retrouve avec un agencement de concept différent de celui de l'ontologie de référence aura un *TR* inférieur à 100%. Cette altération de la structure est récurrente dans le cas de l'approche de Seidenberg et Rector sur les ontologies 3, 8, 12 et 13.

La F-mesure taxonomique nous permet de calculer un score sur la base de la précision et du le rappel taxonomiques. Il s'agit tout simplement d'une combinaison du *TP* et du *TR*. La F-mesure taxonomique est une métrique importante, car elle permet d'évaluer de façon plus générale à la fois les aspects structurel et lexical d'une ontologie. Toujours dans le souci de définir une mesure d'évaluation plus globale de l'ontologie calculée, Staab *et al.* introduisent une nouvelle métrique : il s'agit de la F'-mesure taxonomique qui se calcule en fonction du rappel lexical *LR* et de la F-mesure taxonomique *TF*. Cette mesure est pratique dans le sens où c'est à travers elle qu'on évalue la qualité du module ontologique dans son ensemble : elle peut être interprétée comme étant le *score global* de notre module. La figure 5.4 représente les différentes variations de la F'-mesure en fonction de l'outil d'extraction et des ontologies étudiées. Étant donné que la F'-mesure est liée au *LR*, on constatera également que les résultats de COMET dans l'ensemble demeurent tout aussi satisfaisants.

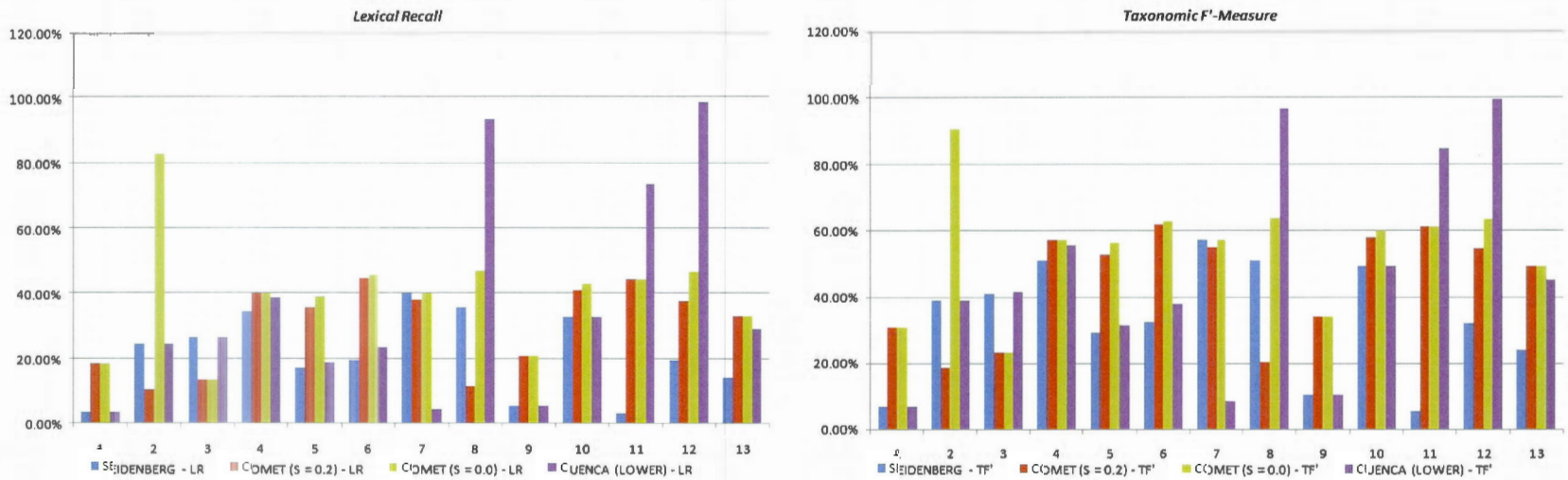

Figure 4.4: Score du Rappel lexical et de la F'-mesure taxonomique des modules extraits.

CONCLUSION

Dans ce mémoire, nous avons proposé une nouvelle technique de modularisation des ontologies, qui est une combinaison des approches structurelle et sémantique existantes. Sur la base de cette technique, nous avons implémenté un prototype d'outil d'extraction de module : COMET. Des tests ont été menés sur différents modules générés à partir d'outils d'extraction dont COMET afin de démontrer qu'on pouvait tirer avantage à la fois de la structure et de la sémantique d'une ontologie, ceci dans l'optique de produire des modules de qualité satisfaisante.

COMET est un outil qui, à partir d'une ontologie et d'une liste de termes, génère un module. Ce dernier représente en fait un segment qui est constitué uniquement de concepts jugés pertinents suivant un algorithme de segmentation. Cet algorithme se base sur deux paramètres qui sont la profondeur hiérarchique et le seuil sémantique ; des éléments essentiels permettant de réguler le parcours de la taxonomie de l'ontologie source et de contrôler les proportions du module à extraire.

Les concepts potentiellement pertinents sont découverts à travers les relations sémantiques. La hiérarchie quant à elle intervient dans deux cas, soit lorsque qu'on recherche dans la taxonomie des concepts ayant des propriétés de type objet, plus précisément des concepts liés à d'autres par une relation dont le poids est supérieur ou égal au seuil ; soit lorsqu'il faut rajouter la descendance des concepts pertinents au module. Après le processus de segmentation de l'ontologie, un second algorithme procède à l'élagage de celle-ci en supprimant tout concept jugé

non-pertinent afin de générer le module final.

Nous avons également mis sur pied un protocole de validation nous permettant, à l'aide d'un certain nombre de métriques, d'évaluer la qualité des différents modules extraits. La validation des outils repose sur une étude comparative des modules générés par rapport à une ontologie de référence, qui dans notre cas est l'ontologie source.

Fort des résultats obtenus lors de expérimentations, nous avons remarqué que la densité est une caractéristique essentielle car elle fait état du niveau de complétude d'une ontologie. Une ontologie dense est une ontologie riche en relations sémantiques, c'est-à-dire une ontologie dont les classes sont clairement définies. Au fil des tests, nous sommes arrivés au conclusion que, plus une ontologie est dense, plus le module retourné par COMET l'est également.

Nous avons pu réaliser l'ensemble des objectifs fixés dans le cadre de ce mémoire, notamment l'élaboration d'un algorithme d'extraction, l'implémentation d'un prototype sur la base de celui-ci et la mise en place du protocole de validation de l'approche ainsi proposée. Cependant, face aux limites actuelles de COMET et dans la perspective d'un développement futur, des améliorations pourraient être faites tant au niveau de l'algorithme qu'au niveau de l'implémentation de l'outil. Ainsi, les points suivants devraient être abordés :

- Choisir une meilleure distance sémantique. Il serait intéressant de se tourner vers une autre mesure telle qu'une distance sémantique basée sur WordNet, car en plus d'être une base de données répertoriant du contenu lexical et sémantique, WordNet se présente aussi comme une ontologie. Cette représentation peut être utiliser afin d'évaluer la distance sémantique entre deux concepts non pas en fonction leur position dans l'ontologie à modulariser, mais plutôt en fonction de leur position dans la taxonomie

de WordNet.

- Proposer une approche empirique permettant de fixer le seuil sémantique et la profondeur hiérarchique. L'expert se doit d'effectuer un certain nombre de tests afin de trouver le seuil idéal, d'où la nécessité d'élaborer un protocole suivant lequel ces tests devront être menés.
- S'inspirer des méthodes d'exploration de graphe basées sur des heuristiques ou sur un approfondissement incrémental lors du parcours de l'ontologie et de l'ajout de la descendance des concepts pertinents au module. En effet, il serait question d'explorer les nœuds du graphe représentant l'ontologie en fonction des poids qui leur sont associés. Dépendamment de la profondeur fixée par l'utilisateur, l'algorithme pourrait non pas rajouter systématiquement la descendance des concepts découverts, mais plutôt rajouter des concepts appartenant à cette descendance en fonction de leur poids.

Ce projet de recherche a été une expérience très enrichissante qui nous a permis d'apporter des tentatives de solution à un problème d'actualité, et de poser des problématiques intéressantes pour une étude future.

APPENDICE A

OUTILS D'EXTRACTION DE MODULES

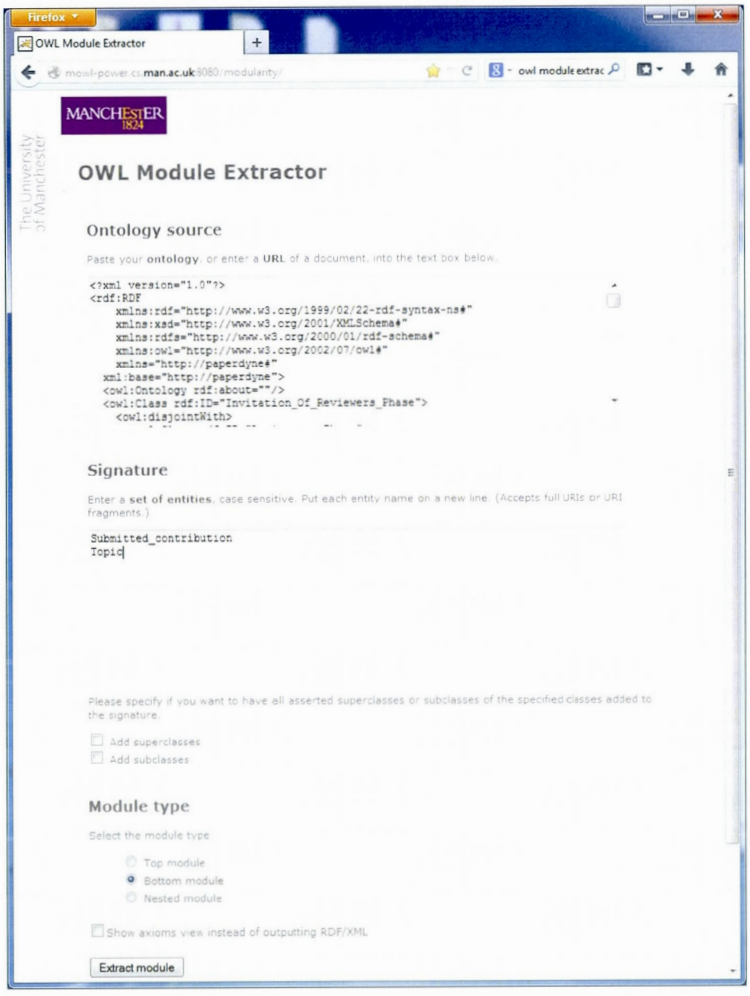

Figure A.1: Interface Web de l'outil OWL-ME.

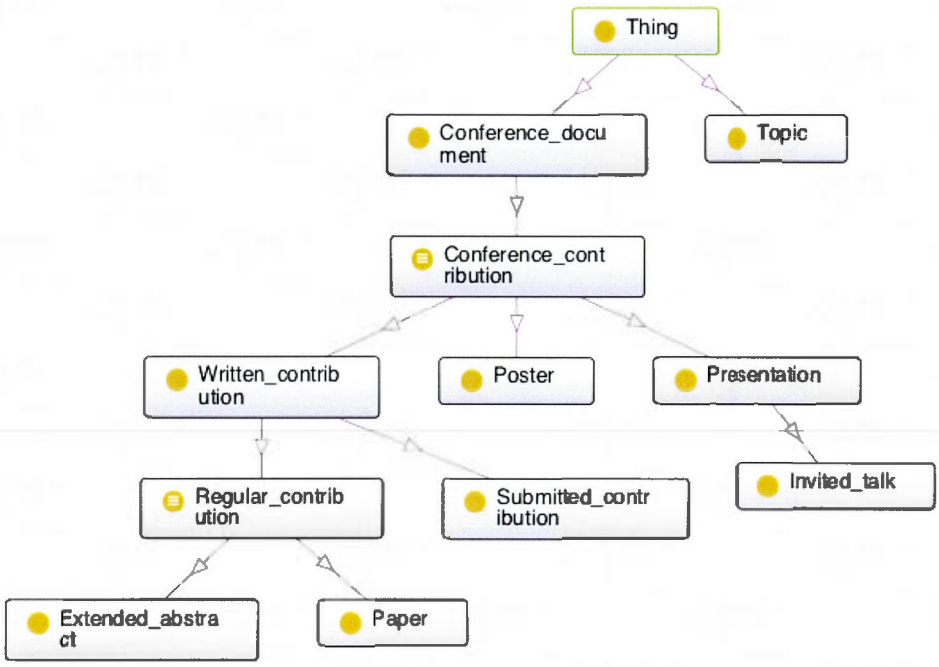

Figure A.2: Module extrait de l'ontologie `conference.owl` à l'aide de OWL-ME.

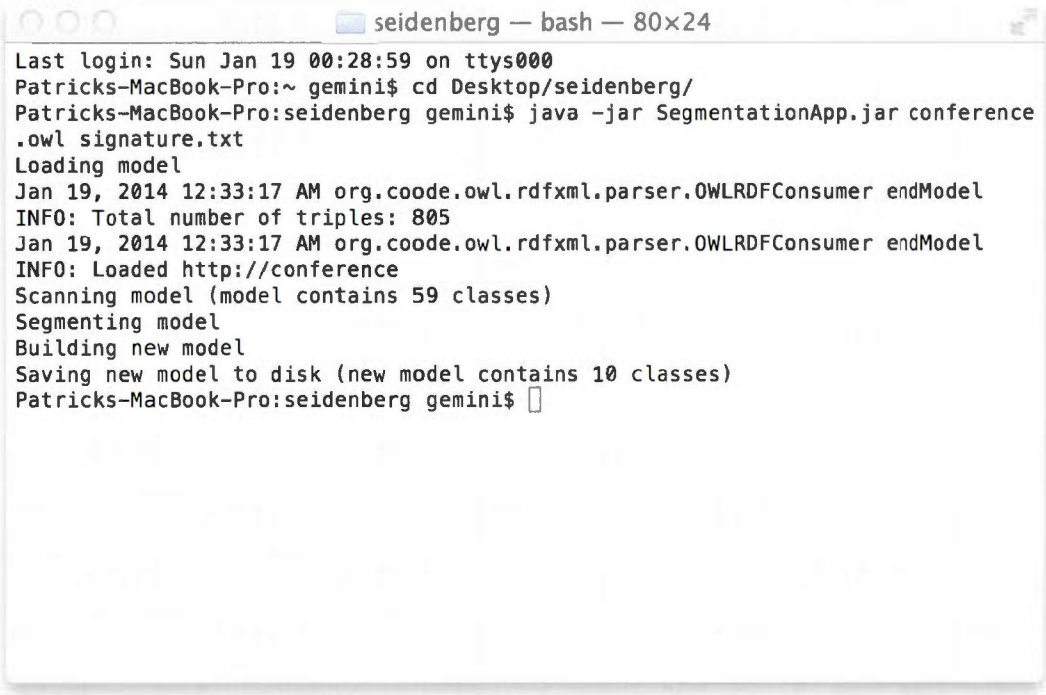

Figure A.3: Processus de modularisation à l'aide de SegmentationApp.

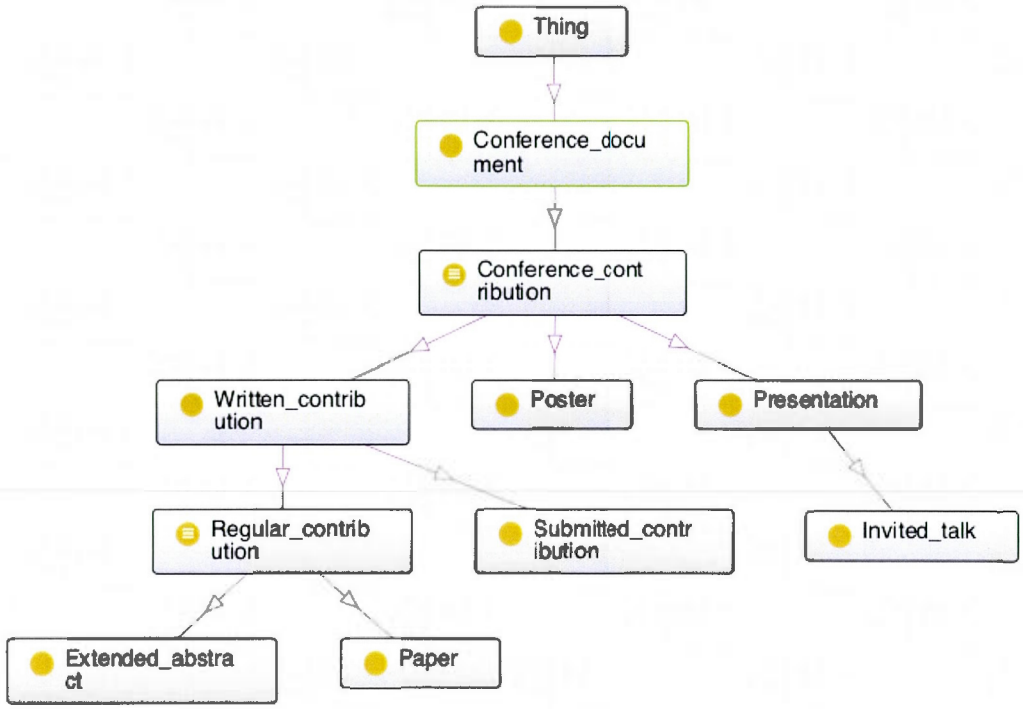

Figure A.4: Module extrait de l'ontologie `conference.owl` à l'aide de SegmentationApp.

APPENDICE B

RÉSULTATS DE L'ÉVALUATION DES MODULES

Tableau B.1: Évaluation des modules en fonction des métriques d'Alani *et al.*

				COMET		
#	Ontologies	Métriques	SEID.	$S = 0.2$	$S = 0.0$	CUENCA-B
1	sweto_simplified	CMM	3,21%	20,10%	20,10%	3,21%
		DEM	20,14%	52,58%	52,58%	20,14%
		SSM	6,36%	32,06%	32,06%	6,36%
		CEM	0,58%	7,17%	7,17%	0,58%
2	cmt	CMM	19,32%	9,24%	87,39%	19,32%
		DEM	40,34%	14,24%	77,84%	40,34%
		SSM	19,28%	4,71%	86,34%	19,28%
		CEM	7,96%	0,00%	99,47%	7,96%
3	confOf	CMM	25,86%	12,93%	12,93%	25,86%
		DEM	39,47%	37,89%	37,89%	48,94%
		SSM	35,57%	11,13%	11,13%	38,66%
		CEM	41,79%	0,65%	0,65%	51,29%
4	iasted	CMM	33,33%	39,58%	39,58%	37,50%
		DEM	73,37%	100%	100%	74,64%
		SSM	43,55%	54,72%	54,72%	46,63%
		CEM	40,81%	22,03%	22,03%	44,88%
5	Conference	CMM	15,96%	35,21%	39,90%	17,37%
		DEM	58,33%	67,40%	69,67%	53,84%
		SSM	27,20%	47,15%	55,40%	25,11%
		CEM	3,56%	34,99%	45,78%	3,28%
6	edas	CMM	15,68%	40,52%	41,17%	19,17%
		DEM	23,57%	51,38%	51,63%	22,40%
		SSM	16,83%	39,97%	39,97%	21,36%
		CEM	14,68%	51,74%	48,69%	31,81%
7	paperdyne	CMM	36,25%	40,35%	43,27%	3,50%
		DEM	82,93%	100%	100%	0,00%
		SSM	45,87%	60,40%	60,69%	3,42%
		CEM	28,31%	41,52%	45,00%	0,00%
8	OpenConf	CMM	32,11%	12,38%	47,24%	93,57%
		DEM	55,87%	49,16%	100%	100%
		SSM	38,96%	21,77%	64,72%	94,95%
		CEM	18,18%	3,94%	18,46%	90,23%
9	ekaw	CMM	4,62%	24,09%	24,09%	4,62%
		DEM	16,57%	100%	100%	16,57%
		SSM	7,30%	44,13%	44,13%	7,30%
		CEM	0,47%	2,64%	2,64%	0,47%
10	sigkdd	CMM	28,57%	45,32%	46,79%	28,57%
		DEM	54,09%	98,87%	96,88%	55,30%
		SSM	36,85%	67,12%	69,15%	36,85%
		CEM	28,07%	56,60%	68,48%	25,60%
11	travel	CMM	4,16%	39,16%	39,16%	77,50%
		DEM	0,00%	100%	100%	76,64%
		SSM	0,00%	57,73%	57,73%	79,39%
		CEM	0,00%	20,80%	20,80%	77,10%
12	PPOntology	CMM	18,78%	34,71%	42,99%	99,04%
		DEM	56,90%	75,58%	89,65%	85,91%
		SSM	30,37%	43,95%	50,63%	99,19%
		CEM	10,46%	14,52%	20,74%	99,13%
13	OTN	CMM	14,66%	35,42%	35,42%	29,32%
		DEM	32,19%	69,61%	69,61%	100%
		SSM	18,72%	49,25%	49,25%	40,02%
		CEM	10,91%	48,88%	48,88%	52,68%

Tableau B.2: Calcul du score des modules en fonction des poids des métriques.

				COMET		
#	Ontologies	Poids	SEID.	$S = 0.2$	$S = 0.0$	CUENCA-B
1	sweto_simplified	0.4, 0.4, 0.2, 0.0	10,61%	35,49%	35,49%	10,61%
		0.25, 0.25, 0.25, 0.25	7,57%	27,98%	27,98%	7,57%
		0.4, 0.3, 0.2, 0.1	8,66%	30,94%	30,94%	8,66%
		0.2, 0.3, 0.4, 0.1	9,28%	33,34%	33,34%	9,28%
2	cmt	0.4, 0.4, 0.2, 0.0	27,72%	10,33%	83,36%	27,72%
		0.25, 0.25, 0.25, 0.25	21,73%	7,04%	87,76%	21,73%
		0.4, 0.3, 0.2, 0.1	24,48%	8,91%	85,52%	24,48%
		0.2, 0.3, 0.4, 0.1	24,47%	8,00%	85,31%	24,47%
3	confOf	0.4, 0.4, 0.2, 0.0	33,24%	22,55%	22,55%	37,65%
		0.25, 0.25, 0.25, 0.25	35,67%	15,65%	15,65%	41,19%
		0.4, 0.3, 0.2, 0.1	33,48%	18,83%	18,83%	37,89%
		0.4, 0.3, 0.2, 0.1	35,42%	18,47%	18,47%	40,45%
4	iasted	0.4, 0.4, 0.2, 0.0	51,39%	66,77%	66,77%	54,18%
		0.25, 0.25, 0.25, 0.25	47,77%	54,08%	54,08%	50,91%
		0.4, 0.3, 0.2, 0.1	48,13%	58,98%	58,98%	51,21%
		0.4, 0.3, 0.2, 0.1	50,18%	62,01%	62,01%	53,03%
5	Conference	0.4, 0.4, 0.2, 0.0	35,15%	50,47%	54,91%	33,50%
		0.25, 0.25, 0.25, 0.25	26,26%	46,19%	52,52%	24,90%
		0.4, 0.3, 0.2, 0.1	29,68%	47,23%	52,52%	28,45%
		0.4, 0.3, 0.2, 0.1	31,93%	49,62%	55,62%	30,00%
6	edas	0.4, 0.4, 0.2, 0.0	19,07%	44,75%	45,12%	20,90%
		0.25, 0.25, 0.25, 0.25	17,69%	45,90%	45,37%	23,68%
		0.4, 0.3, 0.2, 0.1	18,18%	44,79%	44,82%	21,84%
		0.4, 0.3, 0.2, 0.1	18,41%	44,68%	44,58%	22,28%
7	paperdyne	0.4, 0.4, 0.2, 0.0	56,85%	68,22%	69,44%	2,08%
		0.25, 0.25, 0.25, 0.25	48,34%	60,64%	62,24%	1,73%
		0.4, 0.3, 0.2, 0.1	51,38%	62,40%	63,94%	2,08%
		0.4, 0.3, 0.2, 0.1	53,31%	66,41%	67,43%	2,07%
8	OpenConf	0.4, 0.4, 0.2, 0.0	42,98%	28,97%	71,84%	96,42%
		0.25, 0.25, 0.25, 0.25	36,28%	21,81%	57,60%	94,69%
		0.4, 0.3, 0.2, 0.1	39,21%	24,45%	63,69%	95,44%
		0.4, 0.3, 0.2, 0.1	40,58%	26,33%	67,18%	95,72%
9	ekaw	0.4, 0.4, 0.2, 0.0	9,93%	58,46%	58,46%	9,93%
		0.25, 0.25, 0.25, 0.25	7,24%	42,71%	42,71%	7,24%
		0.4, 0.3, 0.2, 0.1	8,32%	48,72%	48,72%	8,32%
		0.4, 0.3, 0.2, 0.1	8,86%	52,73%	52,73%	8,86%
10	sigkdd	0.4, 0.4, 0.2, 0.0	40,43%	71,10%	71,30%	40,92%
		0.25, 0.25, 0.25, 0.25	36,90%	66,98%	70,33%	36,58%
		0.4, 0.3, 0.2, 0.1	37,83%	66,87%	68,46%	37,95%
		0.4, 0.3, 0.2, 0.1	39,49%	71,24%	72,93%	39,60%
11	travel	0.4, 0.4, 0.2, 0.0	1,66%	67,21%	67,21%	77,53%
		0.25, 0.25, 0.25, 0.25	1,04%	54,42%	54,42%	77,66%
		0.4, 0.3, 0.2, 0.1	1,60%	59,29%	59,29%	77,58%
		0.4, 0.3, 0.2, 0.1	0,83%	63,00%	63,80%	77,96%
12	PPOntology	0.4, 0.4, 0.2, 0.0	36,35%	52,91%	63,18%	93,82%
		0.25, 0.25, 0.25, 0.25	29,13%	42,19%	51,00%	95,82%
		0.4, 0.3, 0.2, 0.1	31,70%	46,80%	56,29%	95,14%
		0.4, 0.3, 0.2, 0.1	34,02%	48,65%	57,82%	95,17%
13	OTN	0.4, 0.4, 0.2, 0.0	22,48%	51,86%	51,86%	59,73%
		0.25, 0.25, 0.25, 0.25	19,12%	50,79%	50,79%	55,07%
		0.4, 0.3, 0.2, 0.1	20,36%	49,79%	49,79%	55,00%
		0.4, 0.3, 0.2, 0.1	21,17%	52,55%	52,55%	57,14%

Tableau B.3: Évaluation des modules en fonction des métriques d'Orme *et al.*

				COMET		
#	Ontologies	Métriques	SEID.	$S = 0.2$	$S = 0.0$	CUENCA-B
1	sweto_simplified	NOP	0,00%	100%	100%	0,00%
		APC	0,00%	100%	100%	0,00%
		MaxDIT	50,00%	25,00%	25,00%	50,00%
2	cmt	NOP	0,00%	100%	100%	0,00%
		APC	0,00%	100%	100%	0,00%
		MaxDIT	100%	33,33%	66,66%	100%
3	confOf	NOP	33,33%	100%	100%	33,33%
		APC	100%	100%	100%	100%
		MaxDIT	100%	50,00%	50,00%	100%
4	iasted	NOP	65,85%	100%	100%	70,73%
		APC	100%	100%	100%	100%
		MaxDIT	60,00%	60,00%	60,00%	60,00%
5	Conference	NOP	0,00%	100%	100%	0,00%
		APC	0,00%	100%	100%	0,00%
		MaxDIT	66,66%	33,33%	50,00%	66,66%
6	edas	NOP	2,00%	100%	100%	16,00%
		APC	10,31%	100%	100%	68,66%
		MaxDIT	66,66%	100%	100%	66,66%
7	paperdyne	NOP	28,20%	100%	100%	16,00%
		APC	66,80%	100%	100%	68,66%
		MaxDIT	100%	50,00%	50,00%	66,66%
8	OpenConf	NOP	22,22%	100%	100%	53,33%
		APC	62,62%	100%	100%	57,01%
		MaxDIT	100%	100%	100%	100%
9	ekaw	NOP	0,00%	100%	100%	0,00%
		APC	0,00%	100%	100%	0,00%
		CEM	20,00%	20,00%	20,00%	20,00%
10	sigkdd	NOP	28,57%	100%	100%	35,71%
		APC	82,35%	100%	100%	100%
		MaxDIT	100%	33,33%	33,33%	33,33%
11	travel	NOP	0,00%	100%	100%	50,00%
		APC	0,00%	100%	100%	68,00%
		MaxDIT	0,00%	33,33%	33,33%	100%
12	PPOntology	NOP	30,40%	100%	100%	100%
		APC	100%	100%	100%	100%
		MaxDIT	100%	66,66%	66,66%	66,66%
13	OTN	NOP	12,61%	100%	100%	29,72%
		APC	86,83%	100%	100%	100%
		MaxDIT	80,00%	60,00%	60,00%	80,00%

Tableau B.4: Évaluation des modules en fonction des métriques de Staab *et al.*

				COMET		
#	Ontologies	Métriques	SEID.	$S = 0.2$	$S = 0.0$	CUENCA-B
1	sweto_simplified	LR	3,47%	18,26%	18,26%	3,47%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	6,72%	30,88%	30,88%	6,72%
2	cmt	LR	24,13%	10,34%	82,75%	24,13%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	38,89%	18,75%	90,56%	38,89%
3	confOf	LR	26,31%	13,15%	13,15%	26,31%
		TR	86,67%	100%	100%	100%
		TF	92,85%	100%	100%	100%
		TF'	41,00%	23,25%	23,25%	41,67%
4	iasted	LR	34,28%	40,00%	40,00%	38,57%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	51,06%	57,14%	57,14%	55,67%
5	Conference	LR	16,94%	35,59%	38,98%	18,64%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	28,98%	52,50%	56,09%	31,42%
6	edas	LR	19,41%	44,66%	45,63%	23,30%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	32,52%	61,74%	62,67%	37,79%
7	paperdyne	LR	40,00%	37,77%	40,00%	4,44%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	57,14%	54,83%	57,14%	8,51%
8	OpenConf	LR	35,48%	11,29%	46,77%	93,54%
		TR	84,21%	100%	100%	100%
		TF	91,43%	100%	100%	100%
		TF'	51,12%	20,28%	63,73%	96,66%
9	ekaw	LR	5,47%	20,54%	20,54%	5,47%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	10,38%	34,09%	34,09%	10,38%
10	sigkdd	LR	32,65%	40,81%	42,85%	32,65%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	49,23%	57,97%	60,00%	49,23%
11	travel	LR	2,94%	44,11%	44,11%	73,52%
		TR	100%	100%	100%	100%
		TF	100%	100%	100%	100%
		TF'	5,71%	61,22%	61,22%	84,74%
12	PPOntology	LR	19,31%	37,50%	46,59%	98,86%
		TR	88,23%	100%	100%	100%
		TF	93,75%	100%	100%	100%
		TF'	32,03%	54,54%	63,56%	99,42%
13	OTN	LR	13,96%	32,96%	32,96%	29,05%
		TR	70,74%	97,41%	97,41%	100%
		TF	82,86%	98,68%	98,68%	100%
		TF'	23,90%	49,41%	49,41%	45,02%

APPENDICE C

SCHÉMAS DE MODULES GÉNÉRÉS PAR COMET

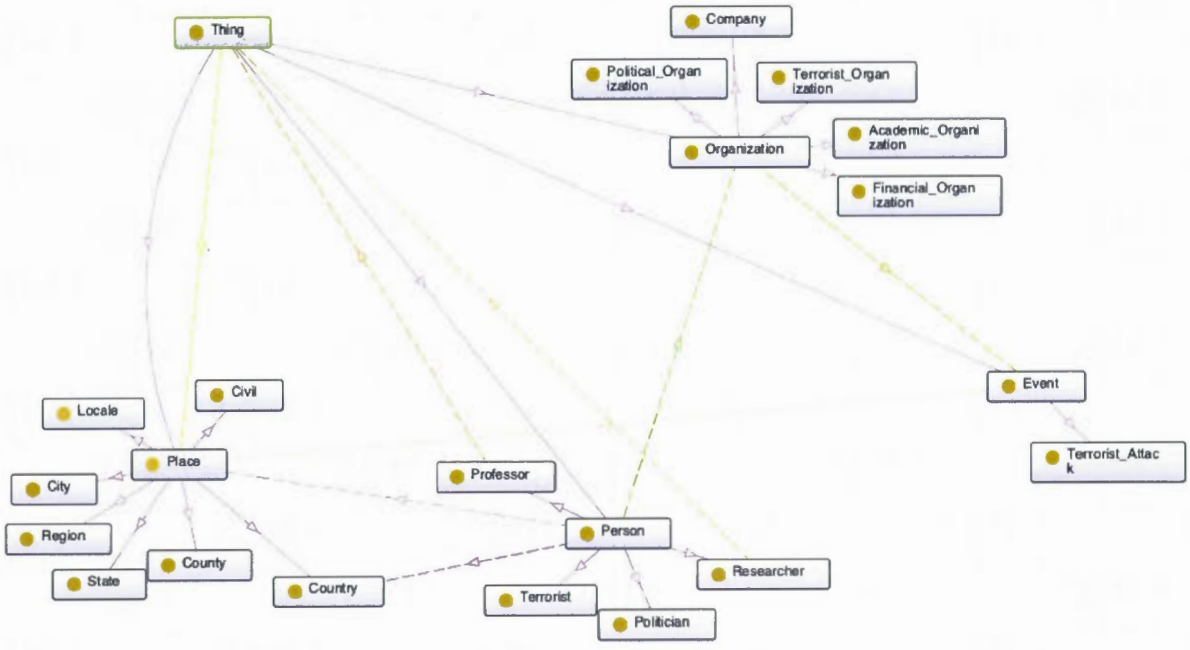

Figure C.1: Module `comet_sweto.owl` extrait de l'ontologie `sweto_simplified.owl` avec un seuil de 0,1 et une profondeur hiérarchique de 1.

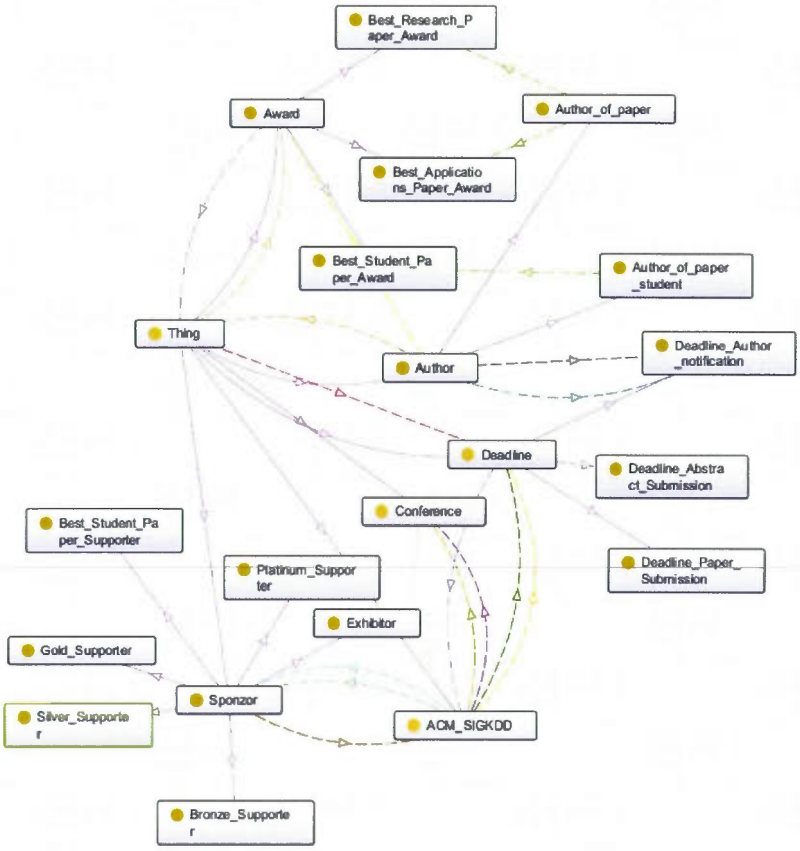

Figure C.2: Module `comet_sigkdd.owl` extrait de l'ontologie `sigkdd.owl` avec un seuil de 0,2 et une profondeur hiérarchique de 1.

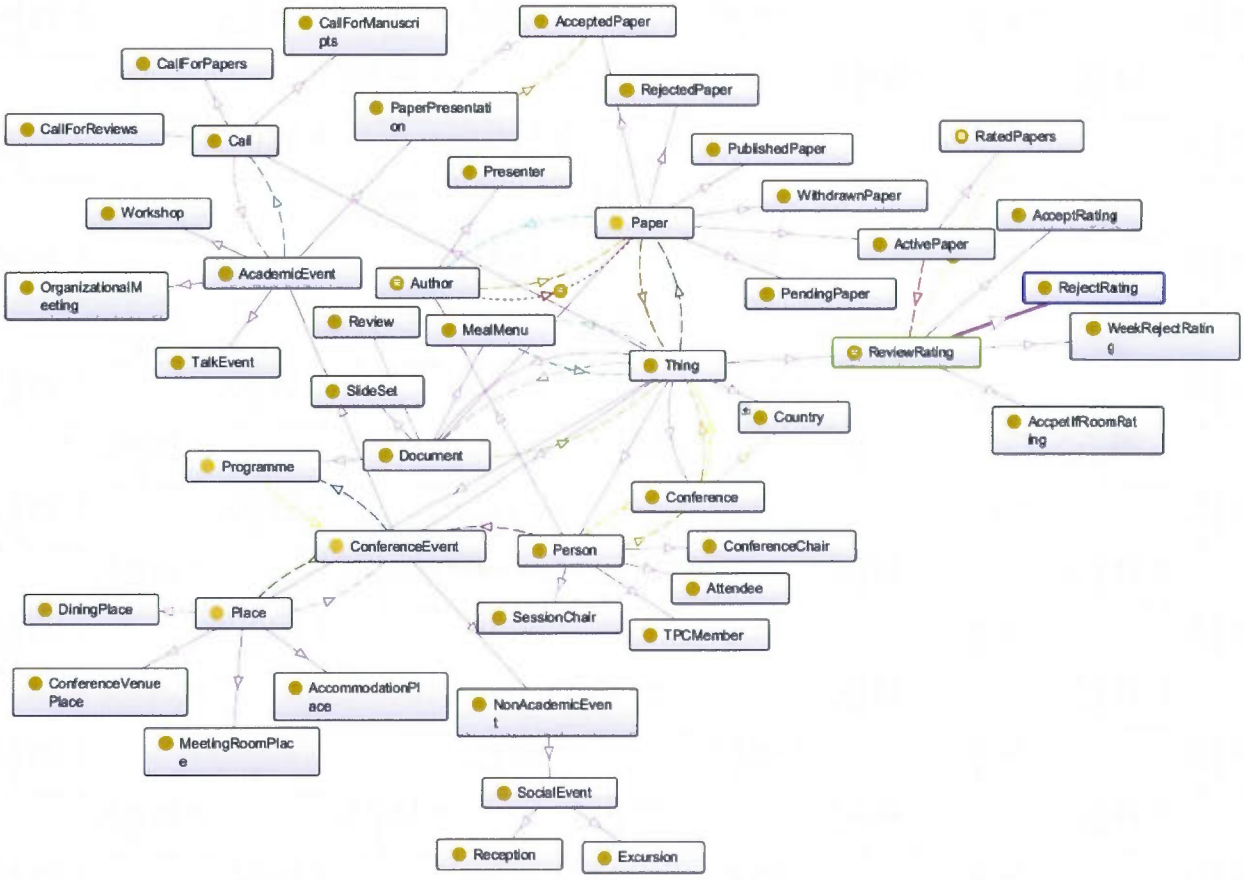

Figure C.3: Module `comet_edas.owl` extrait de l'ontologie `edas.owl` avec un seuil de 0,2 et une profondeur hiérarchique de 1.

APPENDICE D

EXEMPLE DE MODULE GÉNÉRÉ PAR COMET EN OWL

Listing D.1: comet_sigkdd.owl

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://sigkdd#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://sigkdd">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Best_Student_Paper_Supporter">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Sponzor"/>
    </rdfs:subClassOf>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Best_Student_Paper_Supporter</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Deadline_Paper_Submission">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Deadline"/>
    </rdfs:subClassOf>
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Deadline_Paper_Submission</rdfs:label>
  </owl:Class>
  <owl:Class rdf:ID="Bronze_Supporter">
```

```
29      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
30      >Bronze_Supporter</rdfs:label>
31      <rdfs:subClassOf>
32        <owl:Class rdf:about="#Sponzor"/>
33      </rdfs:subClassOf>
34    </owl:Class>
35    <owl:Class rdf:ID="Deadline_Abstract_Submission">
36      <rdfs:subClassOf>
37        <owl:Class rdf:about="#Deadline"/>
38      </rdfs:subClassOf>
39      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
40      >Deadline_Abstract_Submission</rdfs:label>
41    </owl:Class>
42    <owl:Class rdf:ID="Best_Applications_Paper_Award">
43      <rdfs:subClassOf>
44        <owl:Class rdf:ID="Award"/>
45      </rdfs:subClassOf>
46      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
47      >Best_Applications_Paper_Award</rdfs:label>
48    </owl:Class>
49    <owl:Class rdf:ID="Author_of_paper_student">
50      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
51      >Author_of_paper_student</rdfs:label>
52      <rdfs:subClassOf>
53        <owl:Restriction>
54          <owl:someValuesFrom>
55            <owl:Class rdf:ID="Best_Student_Paper_Award"/>
56          </owl:someValuesFrom>
57          <owl:onProperty>
58            <owl:ObjectProperty rdf:ID="award"/>
59          </owl:onProperty>
60        </owl:Restriction>
61      </rdfs:subClassOf>
62      <rdfs:subClassOf>
63        <owl:Class rdf:ID="Author"/>
64      </rdfs:subClassOf>
65    </owl:Class>
66    <owl:Class rdf:about="#Best_Student_Paper_Award">
67      <rdfs:subClassOf>
68        <owl:Class rdf:about="#Award"/>
69      </rdfs:subClassOf>
70      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
71      >Best_Student_Paper_Award</rdfs:label>
72    </owl:Class>
73    <owl:Class rdf:ID="Best_Research_Paper_Award">
74      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
75      >Best_Research_Paper_Award</rdfs:label>
76      <rdfs:subClassOf>
77        <owl:Class rdf:about="#Award"/>
78      </rdfs:subClassOf>
79    </owl:Class>
80    <owl:Class rdf:ID="Platinum_Supporter">
81      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
82      >Platinum_Supporter</rdfs:label>
83      <rdfs:subClassOf>
84        <owl:Class rdf:about="#Sponzor"/>
85      </rdfs:subClassOf>
86    </owl:Class>
87    <owl:Class rdf:about="#Award">
88      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
89      >Award</rdfs:label>
90    </owl:Class>
91    <owl:Class rdf:ID="Deadline_Author_notification">
92      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
93      >Deadline_Author_notification</rdfs:label>
94      <rdfs:subClassOf>
95        <owl:Class rdf:about="#Deadline"/>
96      </rdfs:subClassOf>
97    </owl:Class>
98    <owl:Class rdf:about="#Deadline">
99      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
100     >Deadline</rdfs:label>
101   </owl:Class>
102   <owl:Class rdf:ID="Conference">
103     <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
104     >Conference</rdfs:label>
105   </owl:Class>
106   <owl:Class rdf:ID="Silver_Supporter">
107     <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
108     >Silver_Supporter</rdfs:label>
109     <rdfs:subClassOf>
110       <owl:Class rdf:about="#Sponzor"/>
111     </rdfs:subClassOf>
112   </owl:Class>
```

```
<owl:Class rdf:about="#Author">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Deadline_Author_notification"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="notification_until"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Author</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Exhibitor">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Exhibitor</rdfs:label>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Sponzor"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ACM_SIGKDD">
  <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >ACM_SIGKDD</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hold"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Conference"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="search"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Sponzor"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
```

```
155         <owl:onProperty>
156           <owl:ObjectProperty rdf:ID="design"/>
157         </owl:onProperty>
158         <owl:someValuesFrom rdf:resource="#Deadline"/>
159       </owl:Restriction>
160     </rdfs:subClassOf>
161   </owl:Class>
162   <owl:Class rdf:about="#Sponzor">
163     <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
164     >Sponzor</rdfs:label>
165   </owl:Class>
166   <owl:Class rdf:ID="Author_of_paper">
167     <rdfs:subClassOf>
168       <owl:Restriction>
169         <owl:onProperty>
170           <owl:ObjectProperty rdf:about="#award"/>
171         </owl:onProperty>
172         <owl:someValuesFrom rdf:resource="#Best_Research_Paper_Award"/>
173       </owl:Restriction>
174     </rdfs:subClassOf>
175     <rdfs:subClassOf>
176       <owl:Restriction>
177         <owl:someValuesFrom rdf:resource="#Best_Applications_Paper_Award"/>
178         <owl:onProperty>
179           <owl:ObjectProperty rdf:about="#award"/>
180         </owl:onProperty>
181       </owl:Restriction>
182     </rdfs:subClassOf>
183     <rdfs:subClassOf rdf:resource="#Author"/>
184     <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
185     >Author_of_paper</rdfs:label>
186   </owl:Class>
187   <owl:Class rdf:ID="Gold_Supporter">
188     <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
189     >Gold_Supporter</rdfs:label>
190     <rdfs:subClassOf rdf:resource="#Sponzor"/>
191   </owl:Class>
192   <owl:ObjectProperty rdf:ID="pay">
193     <owl:inverseOf>
194       <owl:ObjectProperty rdf:ID="payed_by"/>
195     </owl:inverseOf>
196   </owl:ObjectProperty>
```

```
197   <owl:ObjectProperty rdf:ID="presentationed_by"/>
198   <owl:ObjectProperty rdf:ID="can_stay_in"/>
199   <owl:ObjectProperty rdf:ID="searched_by">
200     <owl:inverseOf>
201       <owl:ObjectProperty rdf:about="#search"/>
202     </owl:inverseOf>
203     <rdfs:domain rdf:resource="#Sponzor"/>
204     <rdfs:range rdf:resource="#ACM_SIGKDD"/>
205   </owl:ObjectProperty>
206   <owl:ObjectProperty rdf:ID="awarded_by">
207     <owl:inverseOf>
208       <owl:ObjectProperty rdf:about="#award"/>
209     </owl:inverseOf>
210     <rdfs:domain rdf:resource="#Award"/>
211   </owl:ObjectProperty>
212   <owl:ObjectProperty rdf:about="#design">
213     <rdfs:range rdf:resource="#Deadline"/>
214     <rdfs:domain rdf:resource="#ACM_SIGKDD"/>
215   </owl:ObjectProperty>
216   <owl:ObjectProperty rdf:about="#search">
217     <rdfs:domain rdf:resource="#ACM_SIGKDD"/>
218     <rdfs:range rdf:resource="#Sponzor"/>
219   </owl:ObjectProperty>
220   <owl:ObjectProperty rdf:about="#hold">
221     <owl:inverseOf>
222       <owl:ObjectProperty rdf:ID="holded_by"/>
223     </owl:inverseOf>
224     <rdfs:range rdf:resource="#Conference"/>
225     <rdfs:domain rdf:resource="#ACM_SIGKDD"/>
226   </owl:ObjectProperty>
227   <owl:ObjectProperty rdf:ID="presentation">
228     <owl:inverseOf rdf:resource="#presentationed_by"/>
229   </owl:ObjectProperty>
230   <owl:ObjectProperty rdf:ID="submit_until">
231     <rdfs:range rdf:resource="#Deadline"/>
232   </owl:ObjectProperty>
233   <owl:ObjectProperty rdf:about="#award">
234     <rdfs:range rdf:resource="#Award"/>
235   </owl:ObjectProperty>
236   <owl:ObjectProperty rdf:about="#notification_until">
237     <rdfs:domain rdf:resource="#Author"/>
238     <rdfs:range rdf:resource="#Deadline_Author_notification"/>
```

```
239   </owl:ObjectProperty>
240   <owl:ObjectProperty rdf:ID="submit">
241     <rdfs:domain rdf:resource="#Author"/>
242   </owl:ObjectProperty>
243   <owl:ObjectProperty rdf:ID="designed_by">
244     <owl:inverseOf rdf:resource="#design"/>
245     <rdfs:range rdf:resource="#ACM_SIGKDD"/>
246     <rdfs:domain rdf:resource="#Deadline"/>
247   </owl:ObjectProperty>
248   <owl:ObjectProperty rdf:about="#holded_by">
249     <rdfs:range rdf:resource="#ACM_SIGKDD"/>
250     <rdfs:domain rdf:resource="#Conference"/>
251   </owl:ObjectProperty>
252   <owl:ObjectProperty rdf:ID="obtain">
253     <rdfs:range rdf:resource="#Award"/>
254     <rdfs:domain rdf:resource="#Author"/>
255   </owl:ObjectProperty>
256   <owl:DatatypeProperty rdf:ID="City_of_conference">
257     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
258     <rdfs:domain rdf:resource="#Conference"/>
259   </owl:DatatypeProperty>
260   <owl:DatatypeProperty rdf:ID="E-mail">
261     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
262   </owl:DatatypeProperty>
263   <owl:DatatypeProperty rdf:ID="End_of_conference">
264     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
265     <rdfs:domain rdf:resource="#Conference"/>
266   </owl:DatatypeProperty>
267   <owl:DatatypeProperty rdf:ID="Price">
268     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
269   </owl:DatatypeProperty>
270   <owl:DatatypeProperty rdf:ID="Name_of_conference">
271     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
272     <rdfs:domain rdf:resource="#Conference"/>
273   </owl:DatatypeProperty>
274   <owl:DatatypeProperty rdf:ID="Nation">
275     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
276   </owl:DatatypeProperty>
277   <owl:DatatypeProperty rdf:ID="Name_of_sponsor">
278     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
279     <rdfs:domain rdf:resource="#Sponzor"/>
280   </owl:DatatypeProperty>
```

```
281   <owl:DatatypeProperty rdf:ID="Start_of_conference">
282     <rdfs:domain rdf:resource="#Conference"/>
283     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
284   </owl:DatatypeProperty>
285   <owl:DatatypeProperty rdf:ID="Name">
286     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
287   </owl:DatatypeProperty>
288   <owl:DatatypeProperty rdf:ID="Currency">
289     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
290   </owl:DatatypeProperty>
291   <owl:DatatypeProperty rdf:ID="Date">
292     <rdfs:domain rdf:resource="#Deadline"/>
293     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#dateTime"/>
294   </owl:DatatypeProperty>
295 </rdf:RDF>
```

RÉFÉRENCES

Alani, H. et Brewster, C. (2005). Ontology ranking based on the analysis of concept structures. Dans *Proceedings of the 3rd International Conference on Knowledge Capture*, 51–58. ACM.

Alani, H., Brewster, C. et Shadbolt, N. (2006). Ranking ontologies with aktiverank. Dans *Proceedings of the 5th International Conference on The Semantic Web (ISWC'06)*, 1–15. Springer-Verlag.

Borgida, A. et Giunchiglia, F. (2007). Importing from functional knowledge bases-a preview. Dans B. C. Grau, V. Honavar, A. Schlicht, et F. Wolter (dir.). *Proceedings of the 2nd International Workshop on Modular Ontologies, WoMO 2007*, volume 315 de *CEUR Workshop Proceedings*, Whistler, Canada.

Borst, W. (1997). *Construction of Engineering Ontologies for Knowledge Sharing and Reuse.* (Thèse de doctorat). University of Twente, Netherlands.

Brewster, C., Alani, H. et Dasmahapatra, A. (2004). Data driven ontology evaluation. Dans *International Conference on Language Resources and Evaluation (LREC'04).*

d'Aquin, M., Doran, P., Motta, E. et Tamma, V. (2007). Towards a parametric ontology modularization framework based on graph transformation. Dans B. C. Grau, V. Honavar, A. Schlicht, et F. Wolter (dir.). *Proceedings of the 2nd International Workshop on Modular Ontologies, WoMO 2007*, volume 315 de *CEUR Workshop Proceedings*, Whistler, Canada.

d'Aquin, M., Sabou, M. et Motta, E. (2006). Modularization : a key for the dynamic selection of relevant knowledge components. Dans *Proceedings of the ISWC 2006 Workshop on Modular Ontologies*, Athens, Georgia, USA.

d'Aquin, M., Schlicht, A., Stuckenschmidt, H. et Sabou, M. (2009). Criteria and evaluation for ontology modularization techniques. Dans *Modular Ontologies*, volume 5445 de *Lecture Notes in Computer Science*, 67–89. Springer-Verlag.

Dellschaft, K. et Staab, S. (2006). On how to perform a gold standard based evaluation of ontology learning. In *The Semantic Web-ISWC 2006* 228–241. Springer.

Doran, P., Palmisano, I. et Tamma, V. (2008). Somet : algorithm and tool for sparql based ontology module extraction. Dans U. Sattler et A. Tamilin (dir.). *Proceedings of the 2008 ESWC International Workshop on Ontologies : Reasoning and Modularity (WORM-08)*, volume 348 de *CEUR Workshop Proceedings*, Tenerife, Spain.

Doran, P., Tamma, V. et Iannone, L. (2007). Ontology module extraction for ontology reuse : an ontology engineering perspective. Dans *Proceedings of the 16th ACM conference on Conference on Information and Knowledge Management*, 61–70. ACM.

Dupont, P., Callut, J., Dooms, G., Monette, J. et Deville, Y. (2006). *Relevant subgraph extraction from random walks in a graph.* Rapport technique, Université catholique de Louvain, UCL/INGI.

Gangemi, A., Catenacci, C., Ciaramita, M. et Lehmann, J. (2005). Qood grid : A metaontology-based framework for ontology evaluation and selection. Dans *Proceedings of Evaluation of Ontologies for the Web, 4th International EON Workshop, located at WWW2006*, Edinburgh, UK.

Gangemi, A., Catenacci, C., Ciaramita, M. et Lehmann, J. (2006). Modelling ontology evaluation and validation. Dans *Proceedings of the 3rd European Semantic Web Conference*, volume 4011 de *Lecture Notes in Computer Science*, 140–154., Budva, Monténégro. Springer-Verlag.

Gómez-Pérez, A., Corcho, O. et Fernandez-Lopez, M. (2002). *Ontological Engineering.* Springer-Verlag, London, Berlin.

Grau, B. C., Horrocks, I., Kazakov, Y. et Sattler, U. (2007). Just the right amount : extracting modules from ontologies. Dans *Proceedings of the 16th International Conference on World Wide Web*, 717–726., Banff, Alberta, Canada. ACM.

Grau, B. C., Horrocks, I., Kazakov, Y. et Sattler, U. (2008). Modular reuse of ontologies : Theory and practice. *Journal of Artificial Intelligence Research*, *31*(1), 273–318.

Grau, B. C., Horrocks, I., Kazakov, Y. et Sattler, U. (2009). Extracting modules from ontologies : A logic-based approach. In *Modular Ontologies* 159–186. Springer.

Grau, B. C., Parsia, B., Sirin, E. et Kalyanpur, A. (2005). Automatic partitioning of owl ontologies using e-connections. Dans *Proceedings of the 2005 International Workshop on Description Logics (DL-2005).*

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing ? *International Journal of Human-Computer Studies*, *43*(5), 907–928.

Gruber, T. R. *et al.* (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220.

Guarino, N. (1998). Formal onthology in information systems. Dans *Proceedings of the First International Conference (FIOS'98)*, volume 46, Trento, Italy. IOS press.

Guarino, N. et Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, *45*(2), 61–65.

Jiang, J. et Conrath, D. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. Dans *Proceedings of 10th International Conference on Research in Computational Linguistics (ROCLING X)*, 19–33. Computing Research Repository CoRR. Récupéré de `http://www.citebase.org/abstract?id=oai:arXiv.org:cmp-lg/9709008`.

Jiménez-Ruiz, E., Grau, B. C., Sattler, U., Schneider, T. et Berlanga, R. (2008). Safe and economic re-use of ontologies : A logic-based methodology and tool support. In *The Semantic Web : Research and Applications*, volume 5021 de *LNCS* 185–199. Springer.

Kutz, O., Lutz, C., Wolter, F. et Zakharyaschev, M. (2004). E-connections of abstract description systems. *Artificial Intelligence*, *156*(1), 1–73.

Lozano-Tello, A. et Gómez-Pérez, A. (2004). Ontometric : A method to choose the appropriate ontology. *Journal of Database Management*, *2*(15), 1–18.

Lutz, C., Walther, D. et Wolter, F. (2007). Conservative extensions in expressive description logics. Dans *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, volume 7, 453–458., Hyderabad, India.

Maedche, A. et Staab, S. (2002). Measuring similarity between ontologies. In *Knowledge Engineering and Knowledge Management : Ontologies and the Semantic Web* 251–263. Springer.

Monjanel, B. (2011). *Outil compact d'extraction de module d'une ontologie.* Rapport technique, UQAM (GDAC).

Noy, N. et Musen, M. (2004). Specifying ontology views by traversal. *The Semantic Web–ISWC 2004*, 713–725.

Noy, N. F. et Musen, M. A. (2003). The prompt suite : interactive tools for

ontology merging and mapping. *International Journal of Human-Computer Studies*, *59*(6), 983–1024.

Orme, A. M., Yao, H. et Etzkorn, L. H. (2007). Indicating ontology data quality, stability, and completeness throughout ontology evolution. *Journal of Software Maintenance and Evolution : Research and Practice*, *19*(1), 49–75.

Palmisano, I., Tamma, V., Payne, T. et Doran, P. (2009). Task oriented evaluation of module extraction techniques. In *The Semantic Web-ISWC 2009* 130–145. Springer.

Parent, C. et Spaccapietra, S. (2009). An overview of modularity. Dans *Modular Ontologies*, volume 5445 de *Lecture Notes in Computer Science*, 24–55. Springer-Verlag.

Seidenberg, J. et Rector, A. (2005). Techniques for segmenting large description logic ontologies. Dans *Workshop on Ontology Management : Searching, Selection, Ranking, and Segmentation. 3rd International Conference on Knowledge Capture*, 49–56.

Seidenberg, J. et Rector, A. (2006). Web ontology segmentation : analysis, classification and use. Dans *Proceedings of the 15th International Conference on World Wide Web*, 13–22., NY, USA. ACM.

Spyns, P. (2005). *EvaLexon : Assessing triples mined from texts.* Rapport technique 09, STAR Labs.

Stoll, C. (1995). The internet ? bah ! *Newsweek*, *27*, 41.

Stuckenschmidt, H. et Klein, M. (2004). Structure-based partitioning of large concept hierarchies. In *The Semantic Web–ISWC 2004* 289–303. Springer.

Stuckenschmidt, H. et Schlicht, A. (2009). Structure-based partitioning of large ontologies. Dans *Modular Ontologies*, volume 5445 de *Lecture Notes in Computer Science*, 187–210. Springer-Verlag.

Studer, R., Benjamins, V. R. et Fensel, D. (1998). Knowledge engineering : principles and methods. *Data & Knowledge Engineering*, *25*(1), 161–197.

Wu, Z. et Palmer, M. (1994). Verbs semantics and lexical selection. Dans *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, 133–138. Association for Computational Linguistics.

Yao, H., Orme, A. M. et Etzkorn, L. (2005). Cohesion metrics for ontology design and application. *Journal of Computer Science*, *1*(1), 107.