Estimation et propagation des effets de changement résultant de l'évolution d'une ontologie

Mouhamadou Gaye

UNIVERSITÉ GASTON BERGER DE SAINT-LOUIS

École Doctorale des Sciences et des Technologies

T H È S E

pour l'obtention du grade de

Docteur de l'Université Gaston Berger de Saint-Louis

Spécialité : Informatique

Présentée par

Mouhamadou GAYE

Estimation et propagation des effets de changement résultant de l'évolution d'une ontologie

Thèse soutenue le 08 juin 2017

Jury :

Président :	Pr. Ousmane Thiaré	-	Professeur Titulaire Université Gaston Berger de Saint-Louis
Rapporteurs :	Pr. Henri Basson	-	Professeur Titulaire Université du Littoral Côte d'Opale
	Pr. Roger Nkambou	-	Professeur Titulaire Université du Québec à Montréal
	Dr. Mouhamadou Thiam	-	Maître de Conférences, Université de Thiès
Examinateur :	Dr. Mamadou Bousso	-	Maître Assistant, Université de Thiès
Directeur :	Pr. Moussa Lo	-	Professeur Titulaire Université Gaston Berger de Saint-Louis
Co-directeur :	Dr. Ousmane Sall	-	Maître de Conférences, Université de Thiès

Remerciements

Je remercie Monsieur Ousmane Thiaré de m'avoir fait l'honneur de présider le jury de soutenance de cette thèse.

J'adresse mes sincèrement remerciements à mes directeurs de thèse Monsieur Moussa Lo et Monsieur Ousmane Sall pour leur confiance et leur assistance durant toute cette période. Vous ne m'avez pas juste donné l'opportunité d'obtenir ce prestigieux diplôme, vous avez participé au développement personnel de l'homme que je suis grâce à vos conseils et leçons de vie.

Mes remerciements vont à l'endroit de Monsieur Henri Basson, Monsieur Roger Nkambou et Monsieur Mouhamadou Thiam pour avoir accepté d'être les rapporteurs de cette thèse. Vos remarques et suggestions ont été d'un grand apport à ce travail.

J'adresse mes remerciements à Monsieur Mamadou Bousso, l'examinateur de cette thèse. J'aurais voulu être dans des circonstances plus appropriées pour vous témoigner toute ma reconnaissance pour l'entière confiance que vous m'accordez, mais aussi et surtout l'assistance que vous m'avez toujours apportée.

Je remercies tous les membres du Laboratoire d'Analyse Numérique et d'Informatique de l'Université Gaston Berger, tous mes camarades de promotions, particulièrement ma camarade binôme et amie.

Je remercie mes amis, étudiantes et étudiants de l'Université Gaston Berger, les membres du Dahira Mafaatihul Bichri de l'UGB, mes amis, étudiantes et étudiants de l'Université de Thiès, les PER et PATS des UFRs Sciences Et Technologies et Sciences Economiques et Sociales de l'Université de Thiès. J'associe à ces remerciements mes amis depuis l'Ecole Diamaguene 2, le CEM ETAB et le Lycée Demba Diop de Mbour.

Je remercie tous les membres de ma famille, mes sœurs, frères, tantes, oncles, cousines, cousins, nièces, neveux.

Je ne terminerai pas sans remercier très sincèrement des amis de mon défunt père (paix à son âme), Monsieur Djibril Ndong qui a convaincu mon père de m'inscrire à l'école française, Monsieur Loucar Mar qui a beaucoup contribué à ma réussite scolaire et Monsieur Saliou Ndiaye. A mon ami Saliou Gueye, mon frère et confident Mbacké, mon frère cadet et assistant Serigne Cheikh, mon frère et ami Serigne Fallou, mon défunt frère Assane Diop, Serigne Lamine Mbacké, Serigne Moustapha, Serigne Mbacké Ba, et à tous mes amis, je dis merci.

Résumé

L'évolution de l'ontologie se réfère au processus de modification d'une ontologie en réponse à un changement dans le domaine ou sa conceptualisation. Elle se déroule en plusieurs phases dont la représentation des changements, la sémantique, la propagation et la validation constituent les plus importantes. L'analyse des changements à opérer dans l'ontologie est nécessaire pour identifier les conséquences potentielles sur celle-ci et sur les objets dépendants. En effet, une modification d'une entité ontologique peut engendrer des impacts rendant le système inconsistant. Ainsi, l'estimation du degré d'inconsistances d'un changement d'un composant de l'ontologie et de la propagation de ses impacts est essentielle pour maintenir le système en équilibre. Cependant, cette gestion des changements et de leurs effets ne constitue pas une tâche simple ; elle est d'autant plus difficile si l'ontologie concernée est très volumineuse.

Nous proposons une approche de mesure des inconsistances résultant d'une opération de modification sur une ontologie et une approche de suivi de propagation des effets de changement. La mesure d'inconsistances donne une vue sur la sévérité de l'opération de modification et permet de prévoir un scénario de résolution des inconsistances. Notre approche de mesure décompose l'ontologie en communautés et évalue le désordre dans la communauté concernée par le changement, puis dans l'ontologie globale. Cette phase de mesure précède la phase de propagation pour laquelle l'approche proposée permet de quantifier la répartition quantitative des impacts de changement sur les entités ontologiques mais aussi d'estimer le degré de vulnérabilité des composants de l'ontologie aux modifications devant s'opérer.

Mots clés : Ontologie, Évolution d'ontologies, Mesure d'inconsistances, Changements ontologiques, Propagation d'impacts

Abstract

Ontology evolution refers to the process of an ontology modification in response to change in the domain or its conceptualization. It takes in several phases which the most important are the changes representation, semantics of changes, propagation and validation. The analysis of the changes to be made in the ontology is necessary to identify the potential consequences on the ontology and on the dependent objects. Indeed, a modification of an ontological entity can generate impacts making the system inconsistent. Thus, the estimation of the degree of a change inconsistencies on ontology and its impacts propagation are important to keep the system stable. However, managing changes and their effects is not a simple task ; it is more difficult if the ontology concerned is very voluminous.

We propose an approach to measure inconsistencies of a modification operation on ontology and a propagation follow-up approach of the change effects. The inconsistencies measurement gives a view on the severity of the modification operation and allows predicting a scenario for resolving the inconsistencies. Our measurement approach decomposes ontology into communities and estimates disorder in the community concerned by change and then in the full ontology. This phase of measurement precedes the propagation phase for which the proposed approach makes it possible to quantify the quantitative distribution of the impacts on the ontological entities but also to estimate the degree of vulnerability of the ontological components to the modifications that must be accomplished.

Keywords : Ontology, Ontology evolution, Inconsistencies measurement, Ontological changes, impacts propagation

Table des figures

Liste des tableaux

Table des matières

Chapitre 1

Introduction générale

Sommaire

1.1 Contexte général

La vallée du fleuve Sénégal est une zone propice aux activités agricoles et agro-industrielles. Elle s'élargit sur 10 à 25 kilomètres, après une zone de transition appelée Haute vallée, de Kayes à Bakel [DAB+93]. Avec son fort potentiel économique, la vallée du Sénégal fait intervenir, dans le cadre de sa mise en valeur, plusieurs organismes (Ministère de l'Agriculture, OMVS-Organisations pour la Mise en Valeur du fleuve Sénégal, ISRA- Institut Supérieur de Recherche Agricole, SAED- Société d'Aménagement et d'Exploitation des terres du Delta, OMS- Organisation Mondiale de la Santé, etc.) de différents domaines de compétence [Sal10]. Ces organismes produisent et exploitent de grands volumes de données dans le cadre de leur intervention dans cette zone du fleuve Sénégal. L'interdépendance des activités qu'ils mènent implique le besoin d'échanger des informations. La SAED par exemple intégrera dans sa politique d'aménagement des rizières, des données provenant de l'ISRA sur le rendement des espèces de riz. L'information relative au niveau de l'eau sur le fleuve au cours de l'année fournie par l'OMVS intéresserait l'OMS dans la cadre sa politique de prévention des maladies liées à l'eau telles que la bilharziose. Cependant ces données ne sont pas structurées ou stockées sous un même format, posant ainsi un problème pour leur partage entre les organismes les exploitant. Il est alors nécessaire de disposer des outils pour l'intégration sémantique et structurelle de ces données. C'est dans ce sillage qu'est né en 2004 le projet SIC-Sénégal [BDI04] à l'Université Gaston Berger de Saint-Louis. L'objectif de ce projet est de mettre en place une plateforme logicielle à la disposition des producteurs de données (experts, organismes)

appelés partenaires et des consommateurs de données (décideurs, bailleurs) pour faciliter l'intégration, l'organisation, la gestion et l'exploitation des données produites dans la cadre de la mise en valeur du fleuve Sénégal. Plusieurs thèses de doctorat ont contribué à la réalisation de ce projet [Fay07], [Sal10], [Nia13].

Dans le cadre de ses travaux dans [Lo02], Lo propose une approche d'entrepôt de données et apporte des solutions aux problèmes de migration de systèmes d'information vers le web. L'approche repose sur la notion de dataweb qui permet d'intégrer dans un format XML des données provenant de sources structurées ou semi-structurées. Les travaux issus de [Sal10], portant sur la modélisation de données multi sources de type dataweb basé sur XML, ont donné une approche de système d'intégration permettant l'intégration structurelle et sémantique des données d'un partenaire au moyen d'un entrepôt de documents XML et d'une base de connaissances induite. Le modèle consiste à extraire une ontologie à partir des sources d'un partenaire et à mettre en place un système de médiation globale des ontologies des partenaires pour le partage de données entre les différents composants du système. L'utilisation des ontologies est justifiée par le fait que ces dernières fournissent une formalisation d'une compréhension partagée, facilitent l'interopérabilité entre les systèmes partenaires, et ainsi améliorent la qualité de service.

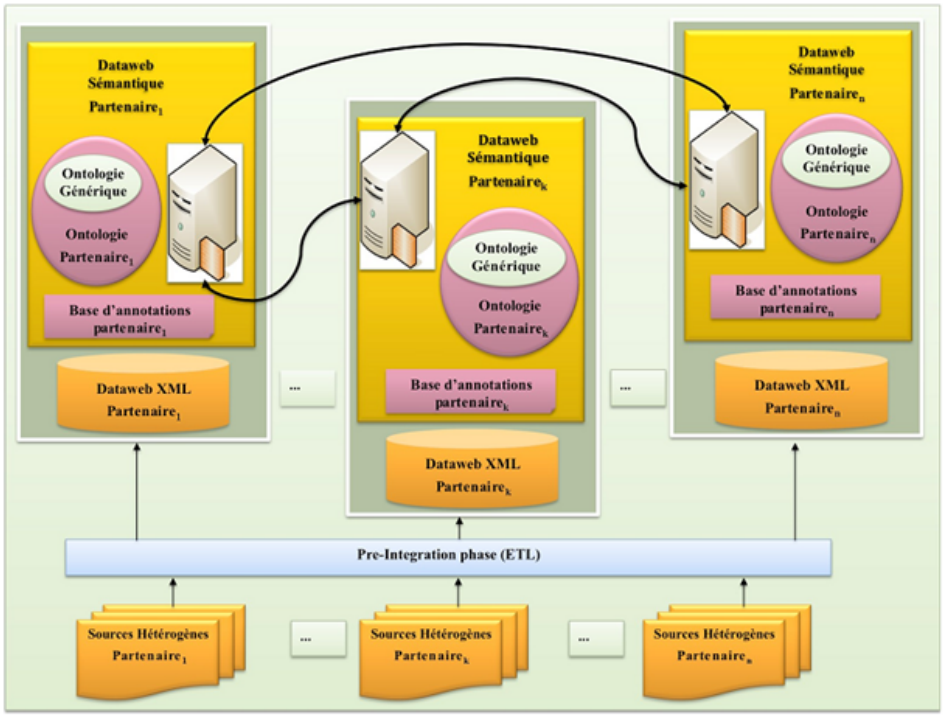

FIGURE 1.1 – Architecture du système d'intégration proposé dans [Sal10]

Toutefois, les sources de données des partenaires sont autonomes dans le sens où elles peuvent évoluer en tout moment. L'ISRA par exemple a introduit depuis 2011

huit nouvelles variétés de mil, haricot et sorgho. D'autre part, les travaux de [Nia13] aboutissant à la mise en place d'un système d'information et de connaissances pour l'échange de données agricoles (SIC-Agri), un sous-projet de SIC-Sénégal, ont permis l'intégration de l'ontologie de référence AGROVOC [1] dans la construction automatique de l'ontologie globale du système de médiation. AGROVOC est un thésaurus multilingue développé par l'équipe informatique de la FAO (Food and Agriculture Organization). Il contient plus de 32 000 concepts disponibles en 23 langues et est maintenu à jour par un groupe d'experts du fait de son évolution continuelle. Par conséquent, les changements intervenus sur ce schéma de concepts auront des répercutions sur les ontologies développées dans le cadre du projet SIC-Sénégal. Selon les auteurs des travaux dans [SMSS03], le domaine d'intérêt d'un partenaire peut changer ou même s'il reste statique, le point de vue sur lequel le domaine est considéré peut évoluer [NK04] (c'est le cas de l'ISRA sur les variétés de culture), ou un défaut de conception dans la définition originale du domaine peut être constaté [PDT05], donnant ainsi lieu à l'évolution de l'ontologie.

L'évolution des ontologies développées pour la plateforme de partage de données doit donc effectivement être prise en charge. En effet, un changement intervenu dans une ontologie partenaire peut provoquer un déséquilibre logique ou architectural dans le système d'intégration global.

1.2 Contexte particulier

L'intégration de données est définie dans [SLG+09] comme un processus guidé par le besoin de recueillir des données ou des documents présentant une hétérogénéité à deux niveaux : structurel et sémantique. Un système d'intégration basé sur les ontologies permet le partage de données d'origine hétérogène à travers des ontologies modélisant ces données. L'utilisation des ontologies dans l'intégration de données s'explique par les avantages qu'elles confèrent au système [BCB05]. En effet, les ontologies permettent de décrire sémantiquement les données sources mais aussi peuvent être utilisées en tant que modèles de requêtes. Cela correspond mieux aux appréciations des utilisateurs dans le domaine concerné. Dans un cadre plus général, les ontologies permettent d'analyser le savoir sur un domaine, d'expliciter ce qui est implicite, de permettre la réutilisation du savoir et son partage entre les personnes ou les applications.

Il faudra cependant noter que les ontologies évoluent, cela pour plusieurs raisons. Une ontologie peut changer parce que le point de vue sur le monde (domaine) change, la connaissance sur certains sujets évolue ou le domaine lui-même change [Kle04]. Selon Klein et Fensel dans [KF01], les changements dans le domaine, les changements dans la conceptualisation partagée ou les changements dans la spécification sont les causes de l'évolution d'une ontologie. Les conséquences d'un changement, toujours dans cet article, peuvent toucher les données qui se conforment à l'ontologie modifiée, les ontologies et les applications l'utilisant. Toutefois, un changement dans une on-

1. http ://aims.fao.org/aos/agrovoc/

tologie doit, comme défini dans [STLB11], [STBL12], respecter certaines conditions afin de garder le système globalement consistant. Selon les travaux dans [SSM+04], une ontologie simple (une ontologie qui n'inclut pas d'autres ontologies dans sa définition) est dite consistante et respecte son modèle si et seulement si, elle préserve les contraintes définies pour le modèle fondamental de celle-ci. Stojanovic [Sto04] définit le modèle de l'ontologie comme une unité autonome d'informations structurées qui peut être réutilisée. Ces contraintes représentent les invariants du modèle, les contraintes souples (pouvant être temporairement invalidées) et les contraintes définies par les utilisateurs. La gestion de l'évolution d'ontologies devient ainsi un domaine de recherche où la maitrise des contraintes constitue un aspect primordial. L'évolution de l'ontologie se définit dans [Sto04] comme l'adaptation dans le temps aux changements produits et la propagation consistante de ces changements dans les artefacts dépendants. C'est une réponse à une modification dans la conceptualisation du domaine ou sa définition. Cette réponse doit, comme mentionné dans [SMMS02], (1) permettre de résoudre les changements apportés à l'ontologie, garantir la consistance de l'ontologie concernée et de tous les objets dépendants, (2) être supervisée permettant aux utilisateurs de traiter plus facilement les changements et (3) offrir aux utilisateurs des conseils pour un raffinement continuel de l'ontologie.

En ce sens, plusieurs approches d'évolution d'ontologies sont proposées dans la littérature. Stojanovic [Sto04] met en place un modèle multidimensionnel pour l'évolution de l'ontologie qui prend en compte le nombre d'ontologies et leur distribution physique. Il propose des méthodes pour la découverte des changements en analysant les comportements des utilisateurs finaux. Les travaux dans [DA09] aboutissent à une méthodologie de gestion de changements appelée Onto-Evoal (Ontology Evolution-Evaluation) qui s'appuie sur une modélisation à l'aide de patrons. Ces patrons spécifient des classes de changements, des classes d'incohérences et des classes d'alternatives de résolution. Les propositions issues de [STL11] et [STLB11] permettent de découvrir les impacts de modification dans les ontologies et la maîtrise des changements provoqués par la détection des chemins de propagation d'impacts. Elles sont basées sur la restructuration de chaque version de l'ontologie selon un modèle lexical, l'analyse préalable des relations et les sens de conductivité d'impacts selon le modèle d'établissement des règles de propagation. La méthodologie de gestion de l'évolution de [RPR04] s'inscrit dans un environnement dynamique, multi acteurs et distribué. Dans ce même sillage, d'autres études ([KPLL10], [KN03], [PDT05], [NCLM06], [DAA07]) proposent des approches d'évolution d'ontologies qui, néanmoins, ne traitent pas l'estimation du degré d'impacts des changements pour une ontologie de très grande taille. L'estimation du degré d'impacts ou la mesure de l'inconsistance consiste à quantifier la contribution de chaque axiome ou assertion d'une base de connaissances dans l'ensemble des inconsistances. Cette mesure permet entre autre de prévoir un scénario de résolution des inconsistances découlant d'une opération de changement. La prise en charge de cette estimation apporte plus de précision dans la gestion de l'évolution des ontologies. En effet, elle donne une vue sur la sévérité de l'inconsistance dans le système et permet de savoir si elle doit être traitée ou ignorée.

1.3 Problématique

L'analyse d'un changement à opérer dans l'ontologie d'un partenaire est nécessaire pour identifier les conséquences potentielles sur l'ontologie en question et par la suite sur le système d'intégration de la plateforme. Comprendre la signification, la structure et les relations entre les items du changement est une tâche à accomplir en amont. La problématique posée à ce niveau consiste à trouver des réponses aux questions suivantes :

- Comment analyser un processus d'évolution d'une ontologie ?
- Qu'est-ce qu'une opération de changement ontologique et comment peut-on la définir ?
- Quels sont les éléments mis en jeu pour la réalisation d'un changement ontologique ?

Les réponses à ces questions serviront à avoir un vocabulaire uniformisé des changements ontologiques. Cela est nécessaire pour l'implémentation de plateformes de gestion de l'évolution d'ontologies. Dans [PDTC07], les auteurs soulignent que la compréhension de l'évolution est fondamental car en fonction de cette dernière, les décisions sur les changements éventuels sont prises.

La seconde problématique de cette thèse consiste à déterminer le degré d'inconsistances d'une modification d'un composant sur l'ontologie. Cette estimation du degré de désordre est essentielle pour maintenir le système en équilibre. En effet, la mesure d'inconsistances qu'un changement ontologique peut engendrer permet de décider si ce changement doit être opéré ou s'il faut recourir à d'autres alternatives. Cette quantification est plus problématique si l'ontologie concernée a une très grande taille [ERDN10]. En effet, la maintenance des ontologies volumineuses est d'autant plus complexe [AMB15] qu'elle est souvent confiée à un groupe d'experts qui ne prennent en charge que la partie de l'ontologie qu'ils ont créée. C'est le cas de NCI-Thesaurus [CSG+05] et de Gene Ontology [PTPD09]. Étant donné que dans l'ingénierie de connaissances les structures modulaires sont moins complexes à exploiter [PJC09], décomposer une ontologie volumineuse en composantes modulaires rendra son exploitation plus simple. Cependant, cette décomposition doit être bien menée d'une manière à ne pas altérer la structure d'origine de l'ontologie. Il convient ainsi de définir les caractéristiques d'un bon partitionnement et de trouver un procédé efficace pour la décomposition mais aussi la mesure d'inconsistances sachant qu'il n'y a pas de méthode universelle pour le partitionnement d'une ontologie et le choix d'une approche est guidé par les besoins du scénario considéré comme le soulignent les auteurs de [dSSS09].

Enfin, le dernier point à traiter dans cette thèse concerne le suivi des effets d'onde d'une opération de modification sur les entités de l'ontologie changeante et les objets dépendants. Il s'agit entre autre de définir une probabilité d'impact d'une modification ontologique sur chaque élément susceptible d'être touché. Cette valeur permet à l'ingénieur de maintenance de l'ontologie d'avoir une vue précise sur la sévérité d'une opération de modification mais aussi d'élaborer un plan de résolution des inconsistances causées. Les travaux jusque là ne traitent pas cet aspect non

moins important dans la résolutions des effets de changements ontologiques.

Notre objectif dans cette thèse est de proposer des approches apportant des solutions à l'ensemble des questions posées. En plus d'être des contributions scientifiques à la gestion de l'évolution des ontologies, ces réponses assurent une bonne maintenance de la plateforme de partage de données du projet SIC-Sénégal, cadre applicatif de la thèse. Un prototype est développé pour valider les différentes propositions.

1.4 Cadre applicatif

1.4.1 Description et objectifs du projet

Avant la mise en place du projet SIC-Sénégal, le système de partage de données des organismes intervenant au niveau de la vallée du fleuve Sénégal était basé sur un serveur central installé à Dakar. Les données échangées à l'époque n'étaient que des bases de données relationnelles. Depuis sa mise en place, le projet SIC-Sénégal s'intéresse aux problématiques liées à l'intégration de données notamment la gestion des données manquantes et l'exploitation des données fédérées. Son but principal est de fournir aux différents intervenants de cette zone agro-industrielle une plateforme permettant le partage et l'exploitation des données produites qui concourent à sa mise en valeur.

1.4.2 Ressources disponibles

Lo a proposé à l'issue de ses travaux dans [Lo02], un dataweb qui permet d'intégrer les sources de données d'un partenaire. Il définit le dataweb comme un entrepôt de document XML construit à partir de données issues de sources hétérogènes (structurées et/ou semi-structurées). L'approche dans [SL07] construit à partir de chaque document XML de cet entrepôt, une ontologie OWL en se basant aussi sur la réutilisation de l'ontologie AGROVOC de la FAO. A son tour, Faye [Fay07] propose SenPeer, un nouveau système Pair-à-Pair de gestion de données distribuées permettant le partage décentralisé et flexible de données relatives à la mise en valeur du fleuve Sénégal. Ce système de type Super-Pair repose sur une organisation des pairs en domaines sémantiques dans lesquels les pairs peuvent publier des bases de données relationnelles, objets ou des documents XML. Un modèle fédérateur structurel et sémantique a été proposé [Sal10] et permis une exploitation intelligente de l'ensemble des données produites dans ou pour cette zone de la vallée. Chaque partenaire dispose d'un dataweb qui constitue la base de connaissances sur laquelle nous travaillons. Elle est construite à partir de l'ontologie partenaire, de l'ontologie générique et d'une base d'annotations. Les travaux de [Nia13] s'orientent dans le même sillage en développant un prototype de construction semi-automatique d'un système de médiation sémantique. Son approche assure la maintenance du schéma global suite à un ajout ou une suppression de source de données et le traitement des requêtes par un processus de réécriture.

Notre contribution dans ce projet concerne la gestion de l'évolution des ontologies

des différents partenaires dont l'ontologie de référence AGROVOC utilisée dans la définition de certains schémas. Nous devons proposer un outil d'évaluation d'impacts de changement d'une ontologie partenaire sur l'ontologie elle-même et sur le système global. Cette évaluation concerne la mesure des inconsistances éventuelles résultant d'opérations de modification et leur propagation sur les artéfacts dépendants de l'ontologie.

1.5 Contributions

Dans cette thèse, nous apportons les contributions suivantes :

- **Analyse des changements ontologiques** : formalise les différents changements simples ou complexes pouvant être opérés dans une ontologie. En plus de la modélisation, l'approche définit les assertions qu'une opération de modification ontologique doit vérifier avant et après l'enclenchement de sa phase d'exécution. Il s'agit de pré-conditions, d'invariants et de post-conditions. Les pré-conditions représentent les assertions que l'opération de modification doit vérifiée avant que l'exécution ne puisse être effective tandis que les post-conditions, si elles sont vérifiées après l'exécution, permettent de valider l'opération. Les assertions à vérifier aussi bien avant et après l'opération de changement constituent les invariants. La non vérification d'un invariant après l'exécution d'un changement déclenche un processus de propagation d'impacts sur l'ontologie changeante et sur les ontologies dépendantes, d'où l'intérêt des autres contributions.
- **Mesure d'inconsistances basée sur le degré de connectivité des nœuds du graphe ontologique** : dans cette approche, il s'agit d'abord de définir une méthode permettant de transformer une ontologie spécifiée dans un langage quelconque en un graphe simple et d'attribuer un poids à chaque type de dépendance. La proposition utilise les flux d'impacts de changement pour définir les différents types de dépendance entre les entités ontologiques. La mesure proposée repose sur ce poids de dépendance entre les nœuds du graphe ontologique et les assertions portant sur une opération de modification notamment les pré-conditions, post-conditions et invariant.
- **Mesure d'inconsistances dans une ontologie de grande taille** : l'approche consiste dans un premier temps à définir une méthode de partitionnement d'une ontologie volumineuse. La décomposition se base sur un algorithme appelé "Island Line" prenant en entrée un graphe pondéré. Après cette étape, nous avons attribué à chaque paire de nœuds du graphe un poids selon le sens et le type de relations auxquelles ils sont noués. Ce poids permet de définir un degré de dépendance entre les différentes entités de l'ontologie. Comme pour l'approche précédente, la mesure se base sur les assertions de l'opération de modification et le poids de dépendance entre les entités présentes dans la négation de l'invariant de l'opération.
- **Propagation quantitative des inconsistances résultant d'une modifi-**

cation sur une ontologie de grande taille : un algorithme de propagation des effets de changements est d'abord proposé. Il détermine toutes les entités de l'ontologie touchées par un changement opéré sur une des leurs. Après cette étape, nous proposons une répartition quantitative des effets de changements sur les éléments du graphe ontologique pour lequel nous déterminons enfin un degré de vulnérabilité de chaque nœud face aux modifications dans l'ontologie.

1.6 Organisation du manuscrit

Ce document s'articule autour de sept chapitres dont les cinq concernent l'état de l'art, les contributions et la validation.

Le premier chapitre fait une introduction générale en commençant d'abord par les contextes général et particulier de la thèse. La problématique des travaux de recherche, le cadre applicatif et les différentes contributions apportées sont relatés dans ce chapitre qui s'achève par l'annonce du plan du manuscrit.

Le chapitre 2 fait une revue littéraire des ontologies et de leur évolution. Dans ce chapitre une panoplie de définitions du terme ontologie est donnée, définitions qui varient selon le contexte et l'usage. Le chapitre relate aussi l'évolution des ontologies, des besoins qui motive l'évolution aux différentes phases d'exécution, avant de terminer par un exposé sur des approches dans cette thématique et des outils de développement d'ontologies intégrant la gestion de l'évolution.

Dans le chapitre 3, un état de l'art sur la gestion des impacts de changements d'une ontologie sur les artéfacts dépendants y est consacré. Il s'agit d'abord de définir un changement ontologique en termes d'atomicité ou de complexité et de disserter ensuite sur des techniques utilisées pour analyser les impacts résultant d'un changement. Le chapitre traite de la prise en charge des mesures d'inconsistances destinées à estimer l'impact d'une opération de modification d'une ontologie sur les objets dépendants. Il fait aussi état des approches d'estimation d'inconsistances résultant pour la plupart de mesures d'impacts classiques dont une liste est présentée, chacune étant définie et caractérisée par des propriétés.

Le chapitre 4 propose deux types de mesure du degré d'impact d'un changement ontologique sur les autres entités de l'ontologie. La première mesure est applicable à une ontologie légère de taille moyenne. Elle repose sur le poids des nœuds du graphe ontologique et le sens du flux de propagation d'impacts. La seconde mesure permet d'estimer l'inconsistance produite par un changement dans une ontologie de grande taille. Pour cette approche, il s'agit d'abord de définir une méthodologie de partitionnement d'ontologie volumineuse dont le chapitre fait état, ensuite de l'établissement la mesure permettant de quantifier l'inconsistance induite par un changement sur une large ontologie.

Le chapitre 5 présente l'approche de propagation et de répartition quantitative des impacts de changements sur le reste de l'ontologie. Il commence par l'approche de marquage des entités affectées par une modification intervenue dans l'ontologie, puis développe la méthode de détermination des probabilités de vulnérabilité et de

la répartition quantitative des inconsistances sur le graphe ontologique.

Le chapitre 6 présente l'implémentation des différentes approches proposées dans cette thèse. Il s'agit du prototype SIC-Evonto, une plateforme de gestion des changements ontologiques destinée à être intégrée dans le système du projet SIC-Sénégal.

Le dernier chapitre de ce document est consacré à la conclusion faisant un résumé des travaux menés dans cette thèse et aux perspectives que ces travaux ouvrent. Il fait aussi état des contributions scientifiques de la thèse.

1.7 Publications

Conférences internationales indexées

1. Mouhamadou Gaye, Mamadou Bousso, Ousmane Sall and Moussa Lo. "**Estimation and propagation of change effects in ontology based on graph properties**". Accepted for publication by the 2nd IEEE International Conference on Agents (IEEE/ICA 2017) to be held in Beijing, China, July 6-9, 2017.
2. Mouhamadou Gaye, Ousmane Sall, Mamadou Bousso and Moussa Lo. "**Measuring inconsistencies propagation from change operation based on ontology partitioning**". In Proceedings of the IEEE - International Conference on Signal Image Technology & Internet based Systems (IEEE/SITIS 2015), pages 178-184, Bangkok, Thailand, November 2015. IEEE DOI : 10.1109/SITIS.2015.18
3. Mouhamadou Gaye, Mamadou Bousso, Ousmane Sall and Moussa Lo. "**Measuring Inconsistency Ontological Changes based on Communities in Ontology**". In Proceedings of the 12th ACS/IEEE International Conference on Computer Systems and Applications (ACS-IEEE/AICCSA 2015), pages 1-4, Marrakech, Morocco, November 2015. IEEE DOI : 10.1109/AICCSA.2015.7507172
4. Mamadou Bousso, Mouhamadou Gaye, Ousmane Sall, Mouhamadou Thiam and Moussa Lo. "**Measuring Inconsistencies on ontology change estimate based on cooperative game and Shannon entropy**". In Proceedings of the 3rd IEEE International Conférence on Control, Engineering & Information Technology (IEEE/CEIT 2015), pages 1-7, Tiemcen-Algeria, May 2015. IEEE DOI : 10.1109/CEIT.2015.7233130

Conférence nationale

1. Mouhamadou Gaye, Ousmane Sall, Mamadou Bousso et Moussa Lo. "**Mesure de l'Inconsistance d'une Ontologie basée sur le Degré de connectivité des Nœuds**". Dans les actes du Colloque National sur la Recherche en Informatique et ses Applications (CNRIA'2015), Ecole Polytechnique de Thiès, Octobre 2015.

Chapitre 2

Évolution des ontologies

Sommaire

2.1 Introduction

Le web sémantique, attribué à Tim Berners-Lee [BLHL+01] du W3C [1](World Wide Web Consortium) , est une infrastructure dont le but est de permettre à des agents logiciels d'aider plus efficacement différents types d'utilisateurs dans leur accès aux ressources sur le web [LRC02]. Le terme "sémantique" désigne la volonté de faire émerger le sens du contenu de la ressource dans sa description/référencement afin d'améliorer le processus de recherche d'informations, sa compréhension, sa réutilisation par les humains, mais aussi par les agents logiciels du web. Cette mission du web sémantique s'accomplit grâce à l'utilisation des ontologies dont le rôle premier est de fournir un vocabulaire partagé pour un domaine de connaissances donné.

Dans ce chapitre, nous faisons une revue littéraire sur les ontologies et leur évolution. L'objectif est de comprendre les concepts, outils et approches concernant la définition, le développement et la maintenance des ontologies. Cette compréhension permettra un traitement adéquat des changements ontologiques et les conséquences éventuelles.

1. http ://www.w3.org

Nous commençons le chapitre par la problématique de la définition du terme ontologie qui varie selon le contexte et l'usage. Nous parlons ensuite des langages de représentation et de spécification des ontologies et des outils permettant de les développer, après un passage sur les différents types d'ontologies. La seconde partie du chapitre est consacrée à l'évolution des ontologies traitée dans la littérature. Le chapitre se termine par une synthèse de l'état de l'art sur l'évolution des ontologies et une conclusion annonçant d'autres points relatés dans ce manuscrit.

2.2 Ontologies du web sémantique

2.2.1 Définitions d'une ontologie

Le terme ontologie n'est pas consensuellement définie dans la littérature. La définition s'adapte selon le contexte d'utilisation de l'ontologie.

Une des premières définitions donnée dans [NFF+91] atteste qu'"*une ontologie définit les termes et relations basiques comprenant aussi bien le vocabulaire d'un sujet d'un domaine que les règles pour combiner les termes et relations afin de définir des extensions de ce vocabulaire*". Gruber dans [G+93], définissait l'ontologie comme "*une spécification explicite d'une conceptualisation*". Cette définition est étendue dans [Bor97] où on parle d'une "*spécification formelle d'une conceptualisation partagée*". Considérant le contexte des sciences de l'information, Gruber [2] donne une nouvelle définition en affirmant qu'"*une ontologie définit un ensemble de primitives représentationnelles avec lesquelles on modélise un domaine de connaissances ou de discours. Les primitives représentationnelles sont typiquement des classes (ou ensembles), attributs (ou propriétés) et des relations (ou des relations sur les classes membres). Les définitions de ces primitives représentationnelles incluent des informations sur leur signification et les contraintes sur la consistance logique de leur application*". Cette définition s'approprie dans le cadre de la représentation de connaissances.

D'autre part, "*un système conceptuel informel*", "*une explication sémantique formelle*", "*une spécification d'une conceptualisation*", "*une représentation d'un système conceptuel par une théorie logique*", "*un vocabulaire utilisé par une théorie logique*" ou encore "*une spécification d'une théorie logique*" constituent une liste de définitions que l'on retrouve dans [GG95] et dont chacune s'adapte selon le contexte d'utilisation de l'ontologie. Dans [FF99], l'on considère une ontologie comme "*des théories de domaine qui spécifient un vocabulaire d'un domaine spécifique d'entités, de classes, de propriétés, de prédicats et de fonctions, et un ensemble de relations qui se tiennent nécessairement sur ce vocabulaire de termes*". La définition de Guarino [Gua98] que l'on peut adopté dans le domaine de l'intelligence artificielle stipule qu'"*une ontologie représente un artefact d'ingénierie constitué d'un vocabulaire utilisé pour construire une réalité, accompagnée d'un ensemble d'hypothèses implicites concernant la signification du vocabulaire des mots*". Pour Uschold [Usc98], "*une on-*

2. http ://tomgruber.org/writing/ontology-definition-2007.htm

tologie peut prendre une variété de formes, mais elle doit nécessairement inclure un vocabulaire de termes, et une certaine spécification de leur signification. Cela inclut des définitions et une indication de comment les concepts sont reliés entre eux, ce qui impose correctement une structure sur le domaine et contraint l'interprétation possible des termes". Ici on insiste sur la structure formelle de l'ontologie et la non ambiguïté de ses termes, ce qui est d'ailleurs une caractéristique d'une définition explicite.

Cependant, la définition la plus partagée dans la littérature, que l'on trouve aussi dans [SBF98], considère une ontologie comme : "*une représentation spécifique et formelle d'une conceptualisation partagée d'un domaine*". La représentation est spécifique parce qu'elle spécifie clairement les concepts, relations, instances et axiomes du domaine, et formelle parce qu'étant interprétable et compréhensible par la machine. C'est une conceptualisation dans le sens où elle est un modèle abstrait d'un domaine, et conceptualisation partagée signifie que son contenu est un consensus des membres de la communauté. Dans [GP99], cette conceptualisation permet de représenter les :

- concepts organisés taxonomiquement à l'aide des relations de subsomption ;
- relations représentant des types d'interactions entre les concepts ;
- axiomes explicitant des énoncés conceptuels ;
- instances, utilisées pour représenter des entités concrètes du domaine.

Nous adoptons dans ce document cette définition qui s'approprie au contexte de l'estimation et de la propagation des effets de changements des ontologies. En effet, la définition caractérise le mieux les aspects fondamentaux d'une ontologie, ontologies que l'on peut classer en différentes catégories.

2.2.2 Types d'ontologies

Nous distinguons différents types d'ontologies selon qu'elles soient caractérisées par leur granularité, formalité, généralité ou calculabilité [HCD+09].

Une ontologie peut être classifiée en granularité importante si elle facilite la conceptualisation du domaine au niveau macro et est représentée dans un langage d'une expressivité minimale, ou en granularité fine permettant la conceptualisation du domaine au niveau micro et représentée dans ce cas à l'aide d'un langage d'une expressivité signifiante [HCD+09].

En termes de formalité, la classification se fait en quatre catégories [UG96] :

- ontologies **fortement informelles** : exprimées librement dans un langage naturel ;
- ontologies **semi-informelles** : exprimées dans une forme restreinte et structurée d'un langage naturel, réduisant l'ambiguïté ;
- ontologies **semi-formelles** : exprimées dans un langage formellement artificiel ;
- ontologies **rigoureusement formelles** : composées de termes méticuleusement définis avec des sémantiques formelles, des théorèmes et preuves de certaines propriétés comme la stabilité et la complétude.

Selon la généralité, les ontologies se classent généralement en ontologies de haut niveau, ontologies de niveau intermédiaire, ontologies de tâche ou de méthode, ontologies de domaine et ontologies d'application.

- Les **ontologies de haut niveau** ou ontologies fondamentales sont des ontologies indépendantes du domaine. La principale utilisation de ces types d'ontologies est pour supporter l'interopérabilité sémantique des ontologies de domaines spécifiques même si elles peuvent aussi être utilisées pour l'alignement d'ontologies. Nous citons dans cette catégorie DOLCE [3] (Descriptive Ontology for Linguistic and Cognitive Engineering), une ontologie développée pour la linguistique et les sciences cognitives et SUMO [4] (Suggested Upper Merged Ontology) qui nait d'une fusion d'autres ontologies de haut niveau.
- Les **ontologies de niveau intermédiaire** ou ontologies utilitaires servent de pont entre les ontologies de domaine et les ontologies de haut niveau.
- Les **ontologies de domaine** spécifient les concepts et les relations interconcepts d'un domaine particulier (domaine médical, agricole, etc.). Nous pouvons citer Gene Ontology [5] et AGROVOC comme exemples d'ontologies de domaine respectivement pour la génétique et l'agriculture.
- Les **ontologies de tâche ou de méthode** sont des ontologies développées pour des tâches ou méthodes spécifiques. Une ontologie peut être par exemple développée pour une tâche d'apprentissage de langues.
- Les **ontologies d'application** sont développées pour des applications spécifiques et utilisent les ontologies de tâche et les ontologies de domaine. C'est le cas des ontologies développées pour la plateforme de partage de données issue du projet SIC-Sénégal.

Nous retrouvons dans [SBF98] une catégorisation des ontologies en ontologies de domaine, ontologies génériques qui sont valides pour plusieurs domaines, ontologies d'applications et ontologies de représentation qui fournissent des entités représentationnelles sans indiquer ce qui doit être représenté. Une autre classification selon le niveau d'abstraction peut être tirée de [Cam13] où il s'agit d'ontologies de fondement, d'ontologies noyaux et d'ontologies de domaine. Les ontologies de fondement représentent un niveau d'abstraction des connaissances indépendant de tout domaine d'application. Elles sont construites à partir des théories issues de la métaphysique, de la logique philosophique, des sciences cognitives et de la linguistique. Elles sont conçues surtout pour pallier à d'éventuels problèmes d'interopérabilité liés à la réingénierie d'ontologies [dOBdAFG11], [Gui06] en servant de modèle de référence pour la construction d'ontologie de domaine. Les ontologies de noyaux (exemple LKIF-Core Ontology, Organization Ontology) modélisent les concepts généraux et communs à plusieurs sous-domaines d'un même domaine et les relations qui les lient. Elles fournissent un haut niveau de conceptualisation et d'axiomatisation [SSFS11] mais ne couvrent pas tous les domaines de connaissances. En fin les ontologies de domaine modélisent des aspects spécifiques d'un domaine. Elles fournissent à la fois

3. http ://www.ontology4.us/Ontologies/Upper-Ontologies/Dolce%20Ontology/index.html
4. http ://www.adampease.org/OP/
5. http ://geneontology.org/page/go-database

une terminologie facilitant la communication entre les acteurs du domaine et une spécification explicite et formelle des connaissances du domaine. Cette classification rejoint la spécification des ontologies en termes de généralité illustrée par la figure 2.1.

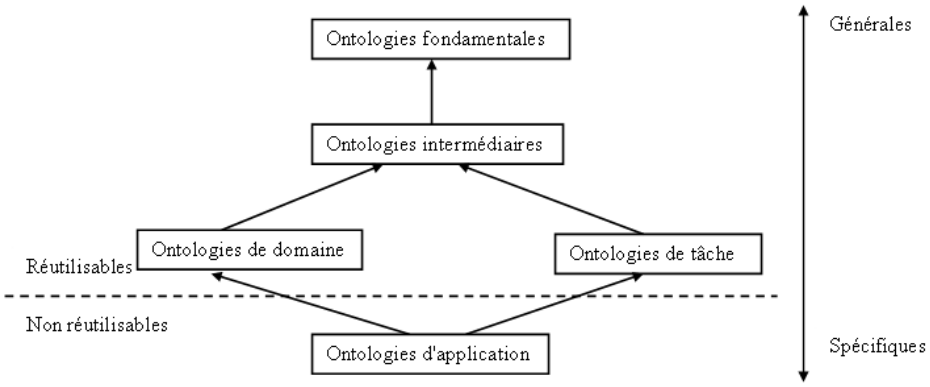

FIGURE 2.1 – Niveaux d'ontologies selon la généralité

En termes de capacité de calcul, les ontologies sont classées en ontologies lourdes et ontologies légères. Les ontologies lourdes contiennent des axiomes et des contraintes permettant d'inférer sur les connaissances qu'elles représentent, alors que les ontologies légères n'ont pas cette capacité.

En définitive, les ontologies sont représentées à l'aide de langages, et le choix d'un langage dépend du type d'ontologie à spécifier.

2.2.3 Langages de représentation et de spécification d'ontologies

Dans la phase de construction d'une ontologie, il peut être usage de plusieurs langages, adaptés aux différentes étapes : le langage naturel ou langage de modélisation informel pour acquérir les connaissances ontologiques, ou des langages de représentation formels et/ou exécutables si l'ontologie doit devenir un composant d'une application. Dans cette partie, nous parlons des langages de représentation et de spécification d'ontologies, illustrés au besoin par des exemples sur l'ontologie de la figure 2.2.

2.2.3.1 DefOnto

DefOnto est un langage développé par l'équipe Ingénierie des Connaissances du LaRIA (Laboratoire de Recherche en Intelligence Artificielle) de l'Université de Picardie Jules Verne. Il se définit comme un sous-langage du langage d'opérationnalisation de modèles de résolution de problèmes Def* [KBA00]. L'objectif visé était de doter le langage de mécanismes de compilation modulaire pour faciliter le développement et la maintenance d'ontologies formelles. Le langage DefOnto permet de représenter des concepts génériques, des relations et des concepts individuels. A

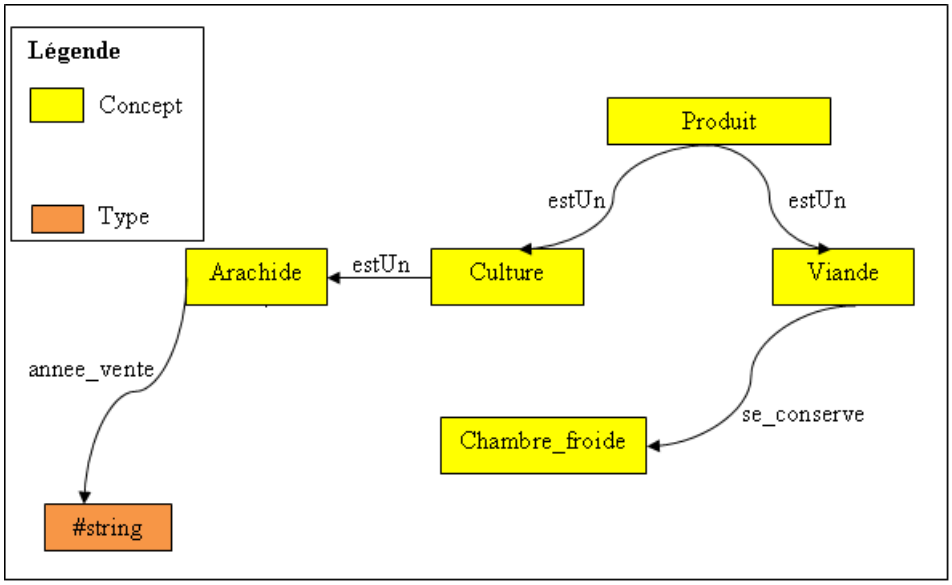

FIGURE 2.2 – Ontologie illustratrice

chaque type d'entité conceptuelle correspond une construction du langage particulière, définie au moyen d'un des "constructeurs" suivants :

- **DefGenConcept** pour définir des concepts génériques ;
- **DefRelation** pour définir des relations ;
- **DefIndConcept** pour définir des concepts individuels.

Exemple 2.1 : Représentation de l'ontologie de la figure 2.2 en DefOnto.

DefGenConcept Produit
DefGenConcept Viande
IsA Produit
DefGenConcept Culture
IsA Produit
DefGenConcept Arachide
IsA Culture
DefRelation se_conserve
RelationProperties
domain Viande
range Chambre_froide
DefRelation annee_vente
RelationProperties
domain Arachide
range #string

2.2.3.2 RDF

1. **Description**

   RDF (Resource Description Framework) [LS+98], [KC06] est un modèle de données pour décrire des ressources sur le web. Une ressource définit une page web, une partie d'une page, un ensemble de pages ou toute entité à décrire qui n'est pas nécessairement accessible sur le web. Le rôle de RDF est de permettre que les informations sur les ressources soient manipulables par des applications. Sa flexibilité et son extensibilité lui confèrent une capacité simplifiée de mettre à jour une ressource. La sémantique d'un document RDF s'exprime en théorie des modèles [HM04] et l'objectif vise à donner des contraintes sur les objets décrits. Le modèle de base de RDF contient les ressources, les propriétés et les états. Une propriété est un aspect spécifique, une caractéristique, un attribut ou une relation qui décrit une ressource. Un état représente une ressource spécifique avec une propriété nommée et une valeur de la propriété pour cette ressource. Un modèle RDF ne définit pas à priori les sémantiques d'un domaine d'application ou met les assomptions par rapport à un domaine d'application en particulier. Il fournit juste un mécanisme de domaine neutre pour décrire les métadonnées.

2. **Syntaxe**

   RDF est un modèle de données sous forme de graphe. La structure sous-jacente de toute expression en RDF représente un ensemble de triplets, composés chacun d'un sujet, un prédicat et un objet (Figure 2.3). L'ensemble de ces triplets est appelé graphe RDF. Il existe cependant plusieurs syntaxes

FIGURE 2.3 – Graphe RDF

   pour représenter une description RDF parmi lesquelles la syntaxe RDF/XML reste la plus connue.

   – **Syntaxe N-Triples**
   La notation N-Triples se qualifie comme la plus simple des notations. Un graphe RDF est représenté, selon cette notation, par une collection de triplets de la forme :

   Sujet Prédicat Objet.

   Si le sujet, le prédicat ou l'objet est une URI (Universal Resource Identifiers) [BL98a], il est représenté en mettant entre crochets <> la forme non

abrégée de cette URI. Dans le cas où le sujet ou l'objet est un noeud vide, la forme _ :nom est utilisé, où nom est un identificateur unique pour ce noeud vide.

- **Syntaxe Notation 3**
  La syntaxe Notation 3 (abrégée N3) [BL98b] représente un graphe RDF de manière flexible et lisible. A la base, elle utilise une notation similaire à la syntaxe N-Triples, sauf qu'elle permet de définir et d'utiliser des préfixes. Lorsqu'une ressource est désignée par une URI dans une forme non abrégée, elle est mise entre crochets <>. Si un préfixe est utilisé, les crochets disparaissent. Cette notation représente un noeud vide par [ ].

- **Syntaxe RDF/XML**
  Dans la syntaxe RDF/XML, un noeud vide est représenté par une balise rdf :Description ne contenant aucun attribut rdf :about (l'attribut rdf :about permet d'identifier une ressource.), ce qui revient à considérer qu'il s'agit d'une description qui ne précise pas la ressource décrite. Tout comme dans la notation N-Triples, un noeud vide peut avoir un identificateur. Il suffit pour cela d'ajouter l'attribut rdf :nodeID dans la balise Description et d'y associer un identificateur unique. A tout endroit dans le même document où on retrouve dans une balise rdf :Description l'attribut rdf :nodeID avec le même identificateur, il s'agira toujours d'une référence au même noeud vide.
  Un littéral dans un graphe RDF contient un ou deux composants nommés. Un littéral simple est représenté en l'insérant entre les balises qui désignent le prédicat ayant ce littéral comme valeur. Si la spécification de la langue utilisée doit être donnée, il suffit de mettre l'attribut xml :lang dans la balise du prédicat et de spécifier la langue en question.
  Le langage RDF présente cependant des limites liées à la définition des classes et la caractérisation des propriétés. Il ne permet non plus d'exprimer la négation.

2.2.3.3 RDFS

Le RDF Schema [BG99], [BG04] est un langage de déclaration et de description légère (typage des ressources et de leurs relations). Il fournit des sémantiques permettant de définir le vocabulaire, la structure et les contraintes pour exprimer les métadonnées des ressources du web. Le langage est adapté pour des ontologies légères mais ne fournit pas néanmoins des sémantiques exactes.

Les primitives de base de RDFS sont les définitions de classes (rdfs :Class) et de sous classes (rdfs :subClassOf), les définitions de propriétés (rdf :Property) et de sous propriétés (rdfs :subPropertyOf), les domaines (rdf :domain) et co-domaines (rdf :range) pour restreindre les combinaisons possibles des classes et propriétés et les types (rdf :type) pour déclarer une ressource comme une instance d'une classe spécifique.

En résumé, RDFS fournit des outils qui permettent de définir des classes et leurs instances, d'établir des propriétés binaires entre objets, d'organiser les classes et propriétés en hiérarchies et d'associer des types aux propriétés. Par ailleurs ce langage présente des limites dont l'une des plus signifiantes est l'impossibilité d'exprimer l'équivalence entre deux instances.

2.2.3.4 OWL

Le groupe W3C ayant constaté un ensemble de caractéristiques que les ontologies du web requièrent et dont les langages RDF et RDFS ne permettent d'exprimer, a initié des recherches pour des langages de définition plus expressifs. A cela s'ajoute l'impossibilité de mener des raisonnements automatisés sur les modèles de connaissances établis à l'aide de RDF/S. Ainsi, après une première tentative qui a donné le langage DAML+OIL, le groupe W3C a développé le langage OWL (Ontology Web Language) [MVH+04]. OWL est un langage fondé sur la syntaxe XML/RDF qui offre une plus grande capacité d'interprétation des ressources du web, grâce à un vocabulaire plus large et une sémantique plus formelle. De part son caractère complexe, OWL est divisé en trois sous-langages ayant des capacités d'expression croissantes.

1. **OWL Lite**
   OWL Lite prend essentiellement en charge les besoins des utilisateurs en hiérarchie de classification simple. Il considère aussi les contraintes de cardinalité, mais ne permet seulement que des valeurs de cardinalité 0 ou 1. Adapté aux migrations rapides pour des thésaurus et d'autres taxonomies, OWL Lite permet d'exprimer les restrictions de domaine et de co-domaine des propriétés, l'équivalence entre classes ou propriétés et l'égalité ou la différence entre instances.

2. **OWL DL**
   OWL DL est adapté aux besoins d'une expressivité maximale garantissant la complétude de calcul et la décidabilité des systèmes de raisonnement. OWL DL est construit avec des restrictions telles que la séparation de type (une classe ne peut pas être un individu, une propriété ne peut pas être un individu ou une classe). Il doit son nom en raison de sa proximité avec la Logique Descriptive [BN03] et offre des expressions algébriques de Bool.

3. **OWL Full**
   OWL Full est le langage OWL qui a le plus haut niveau d'expressivité. Il est destiné aux situations d'une expressivité maximale même si la complétude des raisonnements et leur décidabilité ne sont pas garanties. OWL Full permet d'augmenter le sens du vocabulaire prédéfini. Schreiber [Sch02] affirme que c'est le langage le plus adapté pour l'intégration d'ontologies. Cependant,

toutes ses fonctionnalités ne sont pas supportées par tous les logiciels de raisonnement.

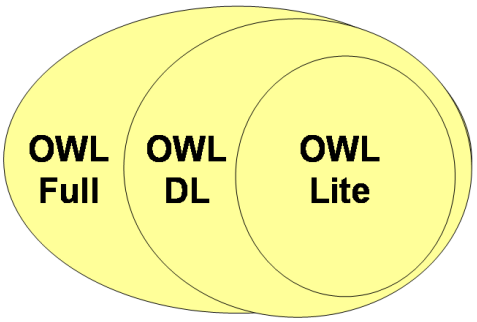

FIGURE 2.4 – Variantes de OWL

Toutefois, le langage OWL a beaucoup évolué et aujourd'hui la version OWL 2 est utilisée avec ses profils d'application.

2.2.3.5 OWL 2

OWL 2 [MPSP+09] est un langage de définition d'ontologie qui ajoute plusieurs fonctionnalités à la norme OWL de 2004 (OWL 1) qu'il remplace. Ces fonctionnalités sont syntaxiques comme l'union des classes et les classes disjointes ou offrent une nouvelle expressivité comme les clés (qui indiquent que chaque instance de la classe est identifiée de manière unique par les propriétés objets et/ou de données), les plages de données, les propriétés asymétriques. Avec OWL 2, une entité peut être interprétée soit comme une classe soit comme un individu en fonction du contexte et une collection de propriétés peut être attribuée comme clé d'une expression de classe. En plus du OWL 2 Full et du OWL 2 DL qui utilise les fragments de la logique du premier ordre et rend efficient le raisonnement pour les bases de connaissances réelles, OWL 2 spécifie trois profils, OWL 2 EL, OWL 2 RL et OWL 2 QL, conçus pour être des sous-ensembles faciles à traiter pour une variété d'applications.

1. **OWL 2 EL**
   OWL 2 EL, assez similaire OWL 2 DL, est conçu pour de grandes ontologies telles que SNOMED-CT [6] , NCI-Thesaurus [7] et GALEN [8]. Ces ontologies comprennent des descriptions structurelles complexes , un grand nombre de classes et ont besoin d'une expressivité terminologique. OWL 2 EL offre des propriétés de domaine, des hiérarchies de classes et de propriétés, des intersections de classes, des disjonctions de classes et de propriétés mais aussi des

6. http ://www.snomed.org/snomed-ct
7. https ://ncit.nci.nih.gov/ncitbrowser/
8. https ://bioportal.bioontology.org/ontologies/GALEN

propriétés de chaînes. Cependant, il ne permet pas d'exprimer des propriétés symétriques, la non disjonction ou la négation ; son utilisation n'est pas non plus limitée au domaine biomédicale.

2. **OWL 2 RL**
   Le profil OWL 2 RL s'adresse aux applications nécessitant un raisonnement évolutif sans sacrifier trop d'énergie pour l'expressivité. Il est conçu pour enrichir des données RDF, surtout lorsque les données doivent être massées. OWL 2 RL interdit les déclarations où l'existence d'un individu impose l'existence d'un autre individu. Par exemple, la déclaration "*chaque personne a un parent*" n'est pas exprimable en OWL RL.
3. **OWL 2 QL**
   OWL 2 QL est lié à la technologie de bases de données relationnelles standard. Cela signifie qu'il peut être étroitement intégré aux SGBDR et bénéficier de ses implémentations robustes et de ses fonctions multi-utilisateurs. OWL 2 QL a également de nombreuses fonctionnalités couramment utilisées dans RDFS et ses extensions telles que la propriété inverse et les hiérarchies de sous-propriété. OWL 2 QL interdit cependant la quantification existentielle de rôles. Par exemple on peut affirmer que "*chaque personne a un parent" mais pas que "chaque personne a une parent femme*".

Le choix du profil à utiliser dépend du niveau d'expressivité que requiert l'ontologie, de la priorité de raisonnement sur les classes ou sur les données, de la taille des données et de leur scalabilité.

2.2.3.6 G-OWL

G-OWL (Graphic OWL) [HN13] est un langage de modélisation graphique pour la construction d'ontologies. Le langage définit un vocabulaire graphique de deux niveaux (abstrait et factuel) constitué d'entités et de relations typées. Le vocabulaire au niveau abstrait comprend quatre entités graphiques : *gowl :Concept*, *gowl :Restriction*, *gowl :Collection* et *gowl :Attribut*. Le vocabulaire au niveau factuel comprend deux entités graphiques : *gowl :Fait* (l'instance d'un concept) et *gowl :Donnee* pour représenter une donnée de type entier, réel ou chaîne de caractères. La figure 2.5 tirée de [HN13] montre une représentation d'une ontologie en G-OWL en comparaison avec d'autres représentations.

2.2.3.7 VOWL

VOWL (Visual Notation for OWL) [NL13], [LNHE14] est un langage de visualisation d'ontologies OWL suivant trois niveaux (TBox, ABox et le niveau combinant les TBox et ABox). La notation en VOWL utilise quatre symboles :

- des cercles pour représenter les concepts ;
- des sections dans les cercles pour représenter les instances ;
- des lignes et arcs pour représenter les propriétés ;
- des rectangles pour représenter les littéraux, les types de données et les valeurs.

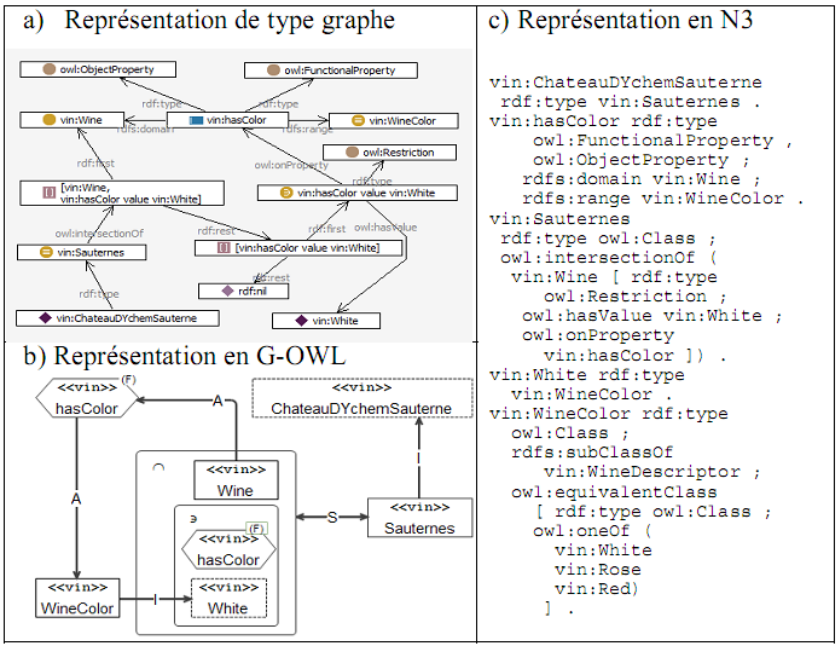

FIGURE 2.5 – Ontologie partielle du Sauterne Château d'Yquem en représentation graph, G-OWL et N3 [HN13]

2.2.3.8 Langages logiques

Les Logiques Descriptives (DLs) [CDGL99], [BHS05] font partie d'une famille de langages de représentation de connaissances pouvant être utilisés pour représenter un domaine d'application de manière formelle et structurée. Les DLs représentent la base de connaissances (ontologie dans notre cas) avec des formalismes terminologiques (TBox) et assertionnels (ABox). Les TBox peuvent être utilisés pour représenter les connaissances non contingentes (les classes) et les ABox pour exprimer les propriétés des instances. Le tableau 2.2 donne un exemple de TBox tiré de [Nia13]. Il existe plusieurs familles de DLs.

1. **Le langage de base $\mathcal{AL}$**
   $\mathcal{AL}$ (Attributive Language) [Gag07] est un langage où les axiomes sont construits à partir d'un ensemble de concepts. Une définition est un énoncé de la forme C≡D tel que C est concept atomique et pour déclarer qu'un concept C est un D on utilise l'axiome C⊑D. Le tableau 2.1 donne les descriptions possibles dans le langage $\mathcal{AL}$. Dans ce tableau, A désigne un concept atomique, C et D sont des concepts atomiques ou complexes.

Table 2.1 – Descriptions dans AL

Syntaxe	Signification
A	Concept atomique
$\top$	Concept universel
$\bot$	Concept impossible
$\neg$A	Négation atomique
C$\sqcap$D	Intersection de concepts
$\forall$R.C	Restriction de valeur
$\exists$R.C	Quantification existentielle limitée

Le langage $\mathcal{AL}$ n'est pas assez expressif pour représenter les ontologies. Ainsi, des constructeurs désignés chacun par un symbole sont ajoutés à ce langage pour le rendre plus expressif.

2. **Les constructeurs de la famille $\mathcal{AL}$**

   – Le **constructeur d'union $\mathcal{U}$** permet de représenter des individus qui appartiennent soit de la classe C, soit de la classe D s'écrivant sous la forme :

   $$C \sqcup D$$

   – La **quantification existentielle complète $\mathcal{E}$** permet de spécifier la classe dont une entité doit avoir au moins une relation avec un objet. Par exemple, pour spécifier la classe des arachides qui contiennent au moins une graine on écrit :

   $$\exists \text{contient.Graine}$$

   – Le **constructeur de fonction $\mathcal{F}$** donne la possibilité de spécifier une relation telle qu'aucune entité ne peut être liée à plus d'une autre entité par cette relation. Pour dire qu'une relation R est une fonction, on écrit *Fun(R)*.
   – Le **constructeur d'inversion $\mathcal{I}$** permet de définir qu'une relation est l'inverse d'une autre.
   – La **restriction de cardinalité $\mathcal{C}$** : les restrictions de cardinalité de la forme $\leq nR$ et $\geq nR$ permettent d'exprimer des concepts comme l'ensemble de ceux qui ont au moins n entités par rapport à une relation ou ceux qui ont au plus n entités par rapport à une relation. Une arachide qui a exactement 3 variétés s'exprime par :

   $$\text{Culture} \sqcap \geq 3\text{aVariete} \sqcap \leq 3\text{aVariete}$$

   – La **restriction de cardinalité qualifiée $\mathcal{Q}$** permet d'exprimer d'imposer le nombre de minimum ou maximum d'entités d'une classe par rapport à une relation que la restriction de cardinalité précédente ne peut pas représenter.

Chacun de ces constructeurs donne un nouveau langage. L'ajout de l'union de concept donne le langage $\mathcal{ALU}$, l'ajout de la quantification existentielle

complète et de la restriction de cardinalité donne le langage $\mathcal{ALEN}$. Si la restriction de cardinalité se limite aux valeurs 0 ou 1, nous obtenons le langage $\mathcal{ALCF}$

3. **La famille $\mathcal{SH}$**
   Les langages logiques de la famille $\mathcal{AL}$ permettent des descriptions complexes sur les concepts mais peinent sur les relations lorsque celles-ci doivent exprimer des restrictions. La famille de langages $\mathcal{SH}$ ajoute une possibilité de hiérarchie et de transivité de relation au langage $\mathcal{ALC}$. Elle comprend essentiellement les langages $\mathcal{SHIF}$ qui correspond à OWL-Lite, $\mathcal{SHOIN}$ correspondant à OWL-DL et $\mathcal{SHIQ}$.

TABLE 2.2 – Un exemple de TBox

$\exists$porteSur$\sqsubseteq$StatAgricoles	Oignon$\sqsubseteq\neg$Sorgho
$\exists$porteSur$^{-}\sqsubseteq$Recolte	StatAgricoles$\sqsubseteq\delta$(annee)
$\exists$concerne$\sqsubseteq$Recolte	Varietes_culture$\sqsubseteq\delta$(prix)
$\exists$concerne$^{-}\sqsubseteq$Varietes_culture	Varietes_culture$\sqsubseteq\delta$(origine)
Niambi$\sqsubseteq$Varietes_culture	ρ(annee)$\sqsubseteq$xsd :string
Oignon$\sqsubseteq$Varietes_culture	ρ(prix)$\sqsubseteq$xsd :string
Sorgho$\sqsubseteq$Varietes_culture	ρ(origine)$\sqsubseteq$xsd :string
Niambi$\sqsubseteq\neg$Oignon	

Les langages logiques offrent des possibilités d'inférence pour déduire des connaissances implicites des représentations explicites.

Dans cette thèse, nous adoptons le langage OWL pour représenter les ontologies utilisées dans l'implémentation des approches proposées. Ce choix s'explique par le fait que OWL offre, en plus de ses possibilités de raisonnement, une capacité de description et d'interprétation de haut niveau. Il s'y ajoute que les ontologies déjà développées dans le cadre du projet SIC-Sénégal sont exprimées en OWL.

2.2.4 Outils de développement d'ontologies

Les outils de développement d'ontologies offrent des interfaces graphiques de construction ou de réutilisation d'ontologies. Ils se différencient essentiellement par leur capacité d'inférence et le nombre de langages supportés.

2.2.4.1 OntoEdit

OntoEdit [SEA+02] est un environnement de développement d'ontologies open source développé à l'Université de Karlsruhe. L'outil se focalise sur trois étapes importantes de développement d'ontologie : les besoins de spécification, le raffinement et l'évaluation. La phase de spécification avec le plug-in OntoKick donne comme sortie une description semi-formelle de l'ontologie qui sera étendue et complètement

formalisée avec un langage spécifique dans la phase de raffinement, grâce aux capacités de raisonnement d'OntoEdit. Le résultat de cette deuxième phase est une ontologie mature que l'outil évalue selon des critères d'évaluation formels.

2.2.4.2 OilEd

L'outil open source OilEd [BHGS01] supporte la construction d'ontologies basées sur OIL. Le composant principal utilisé par OilEd est l'interface de description ("frame description"). Elle représente une collection de super classes avec une liste de contraintes. Avec son design influencé par les outils comme Protégé et OntoEdit, OilEd supporte la maintenance d'ontologies grâce aux raisonnements. L'outil peut être aussi utilisé dans l'inférence pour vérifier la consistance de classes et inférer sur les relations de subsomption. Cependant, OilEd ne fournit pas de support pour le développement d'ontologies de grande échelle, le versioning ou le raisonnement sur des ontologies multiples.

2.2.4.3 Protégé

L'outil Protégé [SCM00], développé à la l'Université de Medecine de Stanford, est l'une des meilleures plateformes open source de développement d'ontologies avec une grande communauté d'utilisateurs. Il implémente une palette de structures et de fonctionnalités qui permettent la création, la visualisation et la manipulation d'ontologies dans de nombreux formats de représentation. Protégé donne la possibilité de définir des classes, des hiérarchies de classes, des valeurs de restriction, des relations entre classes et propriétés. Il peut être étendu avec des plug-ins et des API Java.

2.2.4.4 Apollo

Apollo [LSH+02] est une application de modélisation des connaissances conviviale et gratuite qui permet à un utilisateur de modéliser des primitives de base d'une ontologie, telles que les classes, les instances, les fonctions, les relations. Le modèle interne est un système de trame basé sur le protocole d'OKBC [CFF+98]. La base de connaissances d'Apollo se compose d'une organisation hiérarchique des ontologies. Ces dernières peuvent être héritées par d'autres ontologies et utilisées comme si elles étaient leurs propres ontologies. Chaque ontologie comprend toutes les classes primitives, chaque classe peut créer un certain nombre de cas, et une instance hérite de toutes les caractéristiques de la classe.

2.2.4.5 OntoStudio

Outil de développement d'ontologies open source basé sur le framework Eclipse IBM, OntoStudio [Wei09] a une architecture client/serveur, où les ontologies sont gérées dans un serveur central et les divers clients peuvent y accéder et modifier ces ontologies. Il prend en charge le développement multilingue et collaboratif

d'ontologies. L'outil permet à l'utilisateur d'éditer une hiérarchie de concepts ou de classes. Le modèle de données de représentation interne peut être exporté vers DAML + OIL, Logic, RDF/S ou XML. En outre, les ontologies peuvent être exportées vers des bases de données relationnelles via JDBC.

A la différence des outils ci-dessus cités, TopBraid Composer [9] est un outil de développement d'ontologies payant. Une ontologie développée grâce à une de ces plateformes ou à d'autres outils subit parfois des modifications durant son cycle de vie, et on parle dans ce cas d'évolution de l'ontologie.

2.3 Évolution d'ontologies

L'évolution de l'ontologie se réfère au processus de modification d'une ontologie en réponse à un certain changement dans le domaine ou dans sa conceptualisation [FPA06]. Pour Stojanovic et collaborateurs [SMMS02], l'évolution de l'ontologie est "*la modification appropriée de l'ontologie et la propagation consistante des changements dans les artéfacts dépendants, c'est-à-dire les objets référencés, les ontologies dépendantes et les applications logicielles utilisant l'ontologie*". Selon toujours les auteurs de cet article, une évolution doit (1) permettre de résoudre les changements apportés à l'ontologie, (2) garantir la consistance de l'ontologie concernée et de tous les objets dépendants, (3) être supervisée permettant aux utilisateurs de traiter plus facilement les changements et (4) offrir aux utilisateurs des conseils pour un raffinement continuel de l'ontologie. L'étude dans [Sto04] fait une distinction entre l'évolution d'ontologie, la gestion d'ontologie, la modification d'ontologie et le versioning d'ontologie (création et gestion de versions de l'ontologie) en donnant les définitions ci-dessous.

- La gestion d'ontologies est l'ensemble des méthodes et techniques nécessaires pour une utilisation efficiente de multiples variantes de l'ontologie provenant peut être de sources différentes et destinées à différentes tâches. Ainsi, un système de gestion d'ontologies est une plateforme de création, de modification, de gestion de versions, d'interrogation et de stockage d'ontologies.
- La modification d'ontologies est adaptée lorsque le système de gestion d'ontologies permet d'effectuer des changements dans l'ontologie sans considération de la consistance.
- Le versioning d'ontologies s'adapte à la situation où le système de gestion d'ontologies permet le traitement des changements de l'ontologie par la création et la gestion de différentes versions de celle-ci.
- L'évolution d'ontologies quant à elle, est appropriée lorsque le système de gestion d'ontologies facilite la modification d'une ontologie tout en préservant sa consistance.

Dans tous les cas, les changements apportés à une ontologie s'expliquent par l'évolution de l'environnement pour lequel elle a été développée.

9. http ://www.topquadrant.com/tools/ide-topbraid-composer-maestro-edition/

2.3.1 Besoins d'évolution

La maintenance d'un logiciel est estimée à plus de cinquante pour cent (50%) du coût total de son développement [TM94]. Selon OCLC [10](Online Computer Library Center), 50% des pages web deviennent introuvables chaque année, parce qu'elles sont dans un environnement distribué qui évolue constamment. Les besoins de changements peuvent s'expliquer par l'environnement, les utilisateurs ou par les processus internes des applications utilisant l'ontologie modifiée. L'environnement s'agit du domaine pour lequel l'ontologie est définie et le point de vue sur cette dernière peut changer [Kle04]. Selon les auteurs de [KF01], les changements dans le domaine, les changements dans la conceptualisation partagée ou les changements dans la spécification sont les causes de l'évolution d'une ontologie. Concernant les utilisateurs, leurs besoins évoluent de façon fréquente. L'ajout de nouvelles compétences dans une organisation nécessitera certainement l'ajout de nouveaux concepts dans l'ontologie modélisant cette organisation. En fin les processus internes d'une application sont constamment réadaptés pour réaliser plus de performances. L'évolution permet ainsi d'augmenter l'historique de la connaissance dans l'ontologie de domaine existant pour mieux classifier la description des objets extraits [CFM06]. Cependant, les conséquences d'un changement peuvent impacter les données qui se conforment à l'ontologie, les ontologies et les applications utilisant l'ontologie modifiée [KF01].

2.3.2 Phases d'évolution

L'évolution de l'ontologie se déroule en plusieurs phases. Dans [SMMS02] cinq phases de l'évolution sont pris en compte : la représentation des changements qui identifie et représente les changements dans un format approprié, la sémantique des changements, l'implémentation, la propagation et la validation. La tâche de la phase "sémantique des changements" est de permettre la résolution des changements induits de manière systématique, assurant la consistance de toute l'ontologie. Le processus d'évolution s'organise en six phases en ajoutant la phase de détection des changements aux cinq phases citées (figure 2.6).

- La détection des changements identifie les changements à apporter à l'ontologie. Selon [KFKO02b], deux problèmes surviennent dans la détection des changements : le niveau d'abstraction et l'implication conceptuelle. Dans ce sens, l'étude de [PFF+09] propose un langage spécifique qui permet la formalisation des changements des ontologies RDF/S en considérant les opérations dans les données et le schéma, et un algorithme de détection des changements respectant le langage proposé.
- La représentation des changements définit les caractéristiques des changements sous un format adéquat. Cette phase revêt d'une grande importance parce qu'elle permet d'avoir un vocabulaire uniforme pour les changements.
- La sémantique des changements résout les effets de changements de manière

10. http ://wcp.oclc.org

sémantique afin de conserver la consistance de l'ontologie. Les inconsistances peuvent être sémantiques c'est-à-dire dues à une modification du sens des entités ontologiques ou syntaxiques, liées à un changement dans le modèle de l'ontologie.
- L'implémentation des changements consiste à appliquer les changements approuvés par l'utilisateur dans le système.
- La phase de propagation des changements détermine les conséquences induites par les changements dans les artéfacts dépendants de l'ontologie.
- La validation des changements exécute définitivement les changements intervenus. L'exécution de ces changements peut déclencher l'apparition de nouvelles modifications, ce qui fait dérouler les six phases en boucle.

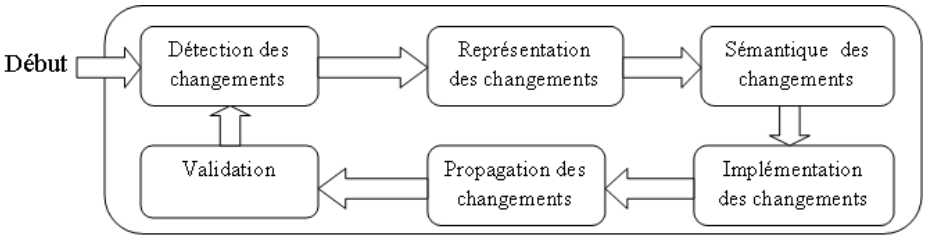

FIGURE 2.6 – Phases d'évolution d'une ontologie

2.3.3 Approches d'évolution d'ontologies

Il existe de nombreux travaux sur l'évolution d'ontologies basées sur les logs de versions. Klein et Fensel dans [KF01], discutent du problème des versions d'ontologies notamment sur les aspects identification et spécification des changements. Ils soulignent qu'il y a trois méthodes pour lier les versions d'ontologies avec les sources de données : en utilisant (1) l'ontologie avec la source de données, (2) une version de l'ontologie plus récente et (3) une version plus ancienne. La méthode de [KFKO02a] permet de comparer les versions de l'ontologie et de spécifier leurs relations conceptuelles. Pour visualiser les différences, les auteurs de cette approche utilisent un mécanisme basé sur des règles et classifient les changements des ontologies RDF. L'étude souligne qu'il n'est pas possible de déterminer automatiquement si un changement est un changement conceptuel ou un changement explicatif. L'approche pose aussi un certain nombre de problèmes :
- Il n'y a pas de support pour trouver les mappings entre les ontologies arbitraires.
- Il n'est pas possible de définir la portée du mapping, c'est-à-dire sur combien de versions il porte.
- Il n'est pas non plus possible de dériver automatiquement toutes les implications conceptuelles des changements.

Dans cette même famille d'approches d'évolution basées sur les logs de versions, les travaux issus de [Rog08] donnent une formalisation des changements élémentaires

et complexes, apportés à une version V_n d'une ontologie pour obtenir une autre version V_{n+1}. Le formalisme est semi-compatible avec OWL et rend interopérable et partageable l'historique des changements. La méthode de [TB06] décrit les possibilités d'une identification automatique des changements entre ontologies en utilisant des heuristiques. Ces dernières sont aussi utilisées dans l'algorithme PromptDiff [NM+02] pour comparer des versions d'ontologie. Les travaux de [KN03] décrivent à travers des techniques et heuristiques, les différentes manières de représenter les informations de changements. Cependant la définition des heuristiques pour avoir des changements complexes à partir des changements simples n'est pas rigoureusement établie. Dans [PDT05], une approche d'évolution de l'ontologie basée sur les traces des différentes versions de l'ontologie est proposée. L'approche combine le "top-down" et le "bottom-up" et supporte la détection implicite des changements complexes. La méthode de [ADB09] permet de détecter et de représenter des relations entre versions d'ontologie en trois étapes : la détection préliminaire de mapping, le raffinement et le nettoyage du mapping et le calcul du mapping inverse. L'évolution des logs est aussi utilisée dans [CM08] pour tracer les lignes de vie des axiomes et annotations et ce, en associant chaque axiome ou annotation à un ensemble de métadonnées. L'idée est d'étudier l'évolution individuelle des axiomes ou annotations au lieu de l'évolution de l'ontologie en entier. Les approches basées sur les logs de versions, largement traitées dans la littérature, posent néanmoins des limites résidant sur le langage utilisé, le non support des dépendances, l'absence d'un environnement de développement collaborative, la non couverture de tous les aspects de l'évolution [AA11].

Certaines études traitent l'évolution des ontologies avec des méthodologies de l'évolution des bases de données [LTZT12], [FGM00]. Noy et Klein dans [NK04] font une étude sur la différence entre l'évolution des ontologies et l'évolution des schémas de bases de données. Pour eux, la traditionnelle distinction entre évolution et versioning n'est pas applicable aux ontologies. En effet, les opérations de changements pour les ontologies sont différentes de celles des bases de données et la sémantique est plus explicite avec les ontologies. Néanmoins, une méthode de stockage de multiples versions d'ontologie basée sur les bases de données relationnelles, proposée dans [LTZT12], permet à travers des requêtes SQL de faire des comparaisons structurelles des ontologies. Le modèle ne supporte pas cependant les expressions booléennes et les définitions d'ontologies sont décomposées et converties en tables relationnelles suivant les métadonnées de l'ontologie. Dans ce même sillage, l'étude de [FGM00] présente une approche pour la spécification et la gestion de l'évolution des schémas de bases de données. Le concept de stratégie d'évolution encapsulant une politique d'évolution qui respecte les besoins des utilisateurs est introduit dans [SMMS02]. Elle consiste à définir un ensemble de points de résolution particuliers. Pour chaque changement élémentaire, un algorithme contenant les points de résolution rencontrés durant la résolution des changements est donné.

Des approches basées sur les patrons de changements existent dans [DA09] et [DA10]. Les patrons correspondent aux dimensions changement, incohérence et alternative de résolution. Sur la base de ces patrons et des liens entre eux, l'approche

définit un processus automatisé permettant de conduire l'application des changements. Les patrons de changements catégorisent les changements, définissent formellement leur signification, leur portée et leurs implications potentielles. L'on note aussi des approches d'évolution d'ontologie utilisant dans leur modèle la logique descriptive. C'est le cas de [KLGE07] qui définit une métrique logique temporelle avec des modularités et un opérateur de satisfaction hybride pour l'analyse de l'évolution. Cependant des opérations de restriction ou d'extension de domaine ou de co-domaine ne sont pas prises en compte. D'autres modèles font recours aux grammaires de graphe pour formaliser et gérer l'évolution des ontologies. Les auteurs de [MTFH13] proposent une approche de gestion des changements ontologiques basée sur la grammaire de graphes. Une grammaire de graphes [Din11] est une paire composée d'un graphe initial et d'un ensemble de règle de production (réécriture). Leur approche définit une règle par un sous-graphe gauche représentant la pré-condition et un sous-graphe droite représentant la post-condition de la règle. Le modèle de [FT06] repose aussi sur un environnement de graphes et permet de raisonner sur des ontologies lourdes. La méthode de [LJL+13] permet l'analyse des caractéristiques des changements, la comparaison des caractéristiques et similarités des croyances et les changements ontologiques. Elle se base sur la révision des croyances [S+76], un cadre général de représentation des incertitudes, englobant la théorie des probabilités comme cas particulier. Dans [JB08], une contribution à l'étude formelle des ontologies basée sur les concepts topologiques est donnée. Le modèle facilite la maintenance de l'ontologie en évitant la restructuration. Cependant la définition de l'échelle de typicalité reste à améliorer.

Nous notons l'existence d'outils de développement d'ontologies intégrant la gestion de l'évolution. Dans la partie qui suit, nous présentons certains d'entre eux.

2.3.4 Outils intégrant l'évolution d'ontologies

2.3.4.1 KAON

KAON (KArlsruhe ONtology) [GSV04] est une infrastructure de gestion d'ontologies Open Source. Il fournit un environnement de travail graphique pour les applications basées sur l'ontologie. Il est constitué d'un ensemble de modules permettant la création, le stockage, la récupération, la maintenance d'ontologies. KAON repose principalement sur les composants suivants :

- OI-Modeler : un éditeur graphique d'ontologie inclus dans la composante KAON Workbench. Son utilité est de permettre l'édition d'ontologies de grande scalabilité, leur gestion et leur maintenance. Il offre différentes vues de l'ontologie créée ainsi que l'inspection de ses composantes.
- KAON API : un ensemble d'interfaces qui donnent accès aux ontologies KAON en fournissant des classes d'implémentation.
- KAON Engineering Server : le serveur d'ingénierie de KAON, un mécanisme de stockage des ontologies KAON sous forme de bases de données relationnelles. Il offre l'accès concurrentiel, transactionnel et scalable aux informations de

l'ontologie.

L'outil KAON donne à ses utilisateurs la possibilité de gérer l'évolution de leurs ontologies en leur permettant de définir des stratégies d'évolution. L'option "Set-up Evolution Parameters" de l'onglet "Procedures" du menu utilisateur permet de définir, comme le montre la figure 2.7, la stratégie avec laquelle, l'OI-Modeler traite les changements survenus à l'ontologie. Ces changements sont exécutés en assemblant les changements élémentaires et composés en une séquence basée sur la stratégie d'évolution adéquate. L'outil assure la création de fichiers logs enregistrant les informations sur les changements intervenus à l'ontologie.

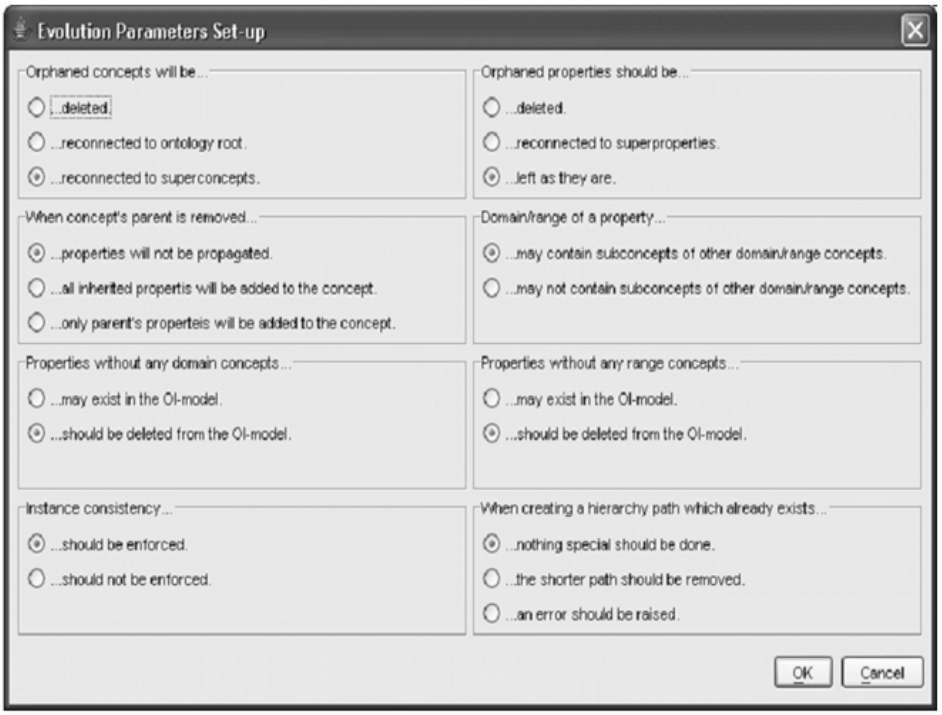

FIGURE 2.7 – Interface KAON de paramétrage pour une stratégie d'évolution

2.3.4.2 OntoView

OntoView [KFKO02a] est un système de gestion de l'évolution d'ontologies basé sur l'interprétation des liens entre différentes versions de l'ontologie. Le système décrit les métadonnées des changements (quels changements, où et quand), les relations conceptuelles entre les différentes versions et les transformations entre les spécifications de l'ontologie. Son fonctionnement se fait à travers les quatre processus : la lecture des changements et des ontologies, l'identification par l'utilisation des espaces de nommage, l'analyse des effets de changements et l'exportation des changements qui pourront être utilisés comme adaptateurs pour d'autres ontologies ou sources de données.

2.3.4.3 PromptDiff

PromptDiff [NM+02] est un algorithme intégrant différents heuristiques de correspondance pour comparer des versions d'une ontologie. Cette comparaison permet de détecter les changements intervenus dans une ontologie. L'algorithme PromptDiff comprend deux parties : un ensemble extensible de coupleurs heuristiques et un algorithme qui combine les résultats des coupleurs pour produire un diff structurel entre les versions de l'ontologie. Chaque coupleur utilise un certain nombre de propriétés structurelles de l'ontologie pour faire ressortir des appariements. PromptDiff est implémenté comme un plug-in dans Protégé 2000.

Tous les outils et approches montrent la prise en charge de l'évolution d'ontologies dans ces phases de détection, sémantique, représentation des changements. Certaines études traitent la résolution des effets de changements même si c'est de façon partielle, généralement.

2.4 Synthèse

Une ontologie définit un ensemble de primitives de représentation avec lesquelles on modélise un domaine. Les ontologies facilitent l'interopérabilité entre les applications en reproduisant une compréhension partagée d'un problème d'un domaine. Elles fournissent une formalisation d'une compréhension partagée qui les rend exécutables par une machine. La représentation explicite des sémantiques des données à travers les ontologies permet aussi aux applications de fournir qualitativement un nouveau type de services. Les ontologies sont représentées à l'aide de langages dont le choix varie selon le niveau de spécification et de formalisation. L'apparition d'ontologies de très grande taille est notée avec l'avènement de grosses données en agronomie ou en médecine nécessitant de nouvelles approches pour leur maintenance. En effet, une ontologie évolue dans le temps en raison de plusieurs facteurs.

L'évolution se réfère au processus de modification en réponse à un certain changement dans le domaine ou sa conceptualisation. Une évolution doit, selon Stojanovic et collaborateurs, [SMMS02] permettre de résoudre les changements apportés à l'ontologie, garantir la consistance de l'ontologie concernée et de tous les objets dépendants, être supervisée permettant aux utilisateurs de traiter plus facilement les changements et offrir aux utilisateurs des conseils pour un raffinement continuel de l'ontologie. Les approches sur l'évolution des ontologies se basent pour la plupart sur les logs de versions, les méthodes de gestions des schémas de bases de données ou les patrons de changements. Il existe cependant des outils de développement et de gestion d'ontologies donnant la possibilité aux utilisateurs de gérer certains aspects de leur évolution.

2.5 Conclusion

Dans ce chapitre, nous avons fait un état de l'art sur les ontologies et leur évolution. Nous avons commencé par un exposé de définitions du terme ontologie.

Les définitions varient selon les besoins et l'usage de l'ontologie. La diversité notée dans la signification du mot ontologie est aussi présente dans le choix de ses langages de représentation et de spécification. L'évolution des ontologies, abordée dans la deuxième partie de ce chapitre, n'est pas non plus exempte de cette pluralité. En effet, plusieurs approches de gestion d'évolution d'ontologies sont proposées dans la littérature. Pour la plupart des approches le processus est assimilé à la gestion des différentes versions de l'ontologie. D'autres approches utilisent la logique descriptive ou sont orientées utilisateurs.

Nous avons constaté cependant que l'évolution des ontologies de grande taille n'est pas spécifiquement prise en charge par les approches ci-dessus citées. Dans tous les cas, la gestion de l'évolution d'une ontologie doit être compréhensible, et comme mentionné dans le cycle d'évolution, inclure une phase d'analyse d'impacts de changements et de propagation des inconsistances qui devront faire l'objet de mesures. Le chapitre suivant fait un état de l'art sur ces principaux aspects.

Chapitre 3

Gestion des changements ontologiques

Sommaire

3.1 Introduction

L'évolution représente une phase importante dans le cycle de vie d'une ontologie. En effet, un changement d'un des composants de l'ontologie peut rendre inconsistants les autres composants taxonomiquement et sémantiquement, et propager ses impacts sur ces éléments. Étant donné que la consistance de l'ontologie doit demeurer vérifiée, il est alors important de quantifier le degré d'inconsistances d'une modification d'une entité donnée sur l'ontologie, afin de pouvoir définir les actions à mener pour rendre celle-ci consistante. On parle ainsi d'analyse et de mesure d'impacts de changements. L'analyse d'impacts étudie l'impact d'une opération de modification d'une entité ontologique et prédit les conséquences éventuelles de cette dernière alors que la mesure d'inconsistances quantifie la contribution de chaque axiome ou assertion de l'ontologie dans l'ensemble des inconsistances produites.

Dans ce chapitre, nous faisons un état de l'art sur la gestion des impacts de changements d'une ontologie sur les artéfacts dépendants. Il s'agit d'abord de définir un changement ontologique en termes d'atomicité ou de complexité et de disserter

ensuite sur des techniques utilisées pour analyser les impacts résultant d'un changement. Nous parlons aussi dans ce chapitre des mesures d'inconsistances destinées à estimer l'impact d'une opération de modification d'une ontologie sur les objets dépendants. Le chapitre fait état des approches de mesures reposant, pour la plupart, sur des mesures d'inconsistances classiques dont les plus courantes seront présentées, chacune étant définie et caractérisée par des propriétés.

Ce chapitre organisé en trois parties commence par une revue littéraire sur l'analyse d'impacts de changements incluant les types de changements ontologiques et la propagation d'impacts de changements. Cette partie est suivie par un exposé sur les mesures d'inconsistances. La troisième partie du chapitre relate les approches de résolution d'inconsistances, chapitre que nous terminons par une synthèse sur l'état de l'art et une conclusion annonçant les principales contributions de cette thèse.

3.2 Gestion des impacts de changements

La gestion des impacts de changements est l'activité destinée à analyser une opération de changement ontologique et à prévoir ses impacts sur les artéfacts dépendants de l'ontologie modifiée. Dans cette partie, nous parlons des types de changements ontologiques et de la prise en charge de l'analyse mais aussi de la propagation de leurs impacts.

3.2.1 Types de changements ontologiques

Selon le critère de taxonomie considéré, les changements ontologiques sont différemment classifiés dans la littérature. Klein et ses collaborateurs [KFKO02b] énumèrent deux types de changements [PDBcS97] qui affectent la conceptualisation de l'ontologie : les changements conceptuels et les changements explicatifs qui sont des changements dans la spécification de la conceptualisation et non dans la conceptualisation. Klein ajoute un troisième type de changement dans [Kle04] et parle désormais de changement dans la conceptualisation, changement dans la spécification et changement dans la représentation. Du point de vue de la détection, les changements ontologiques peuvent être de deux catégories : les changements "top-down" et les changements "bottom-down" [SMMS02]. Les changements "top-down" sont des changements explicites, introduits, par exemple, par un utilisateur "top-manager" qui veut adapter le système aux nouvelles exigences et peuvent être facilement réalisés par le système d'évolution d'ontologie. Les changements "bottom-up" quant à eux sont extraits de l'ensemble des instances de l'ontologie. Pour Stojanovic [Sto04], les changements ontologiques forment deux familles : les changements élémentaires appelés aussi changements simples ou atomiques et les changements complexes qui ne représentent pas une prédominance dans les résultats de recherche proposés.

3.2.1.1 Changements simples

Un changement simple est un changement de l'ontologie qui modifie (ajoute ou supprime) une seule entité du modèle ontologique [Sto04]. Cette entité peut être un concept, une relation, une instance, un type ou une caractéristique d'une relation. Le tableau 3.1 donne une liste de changements ontologiques simples tirés des travaux de [Sto04], [Kle04], [SMMS02], [STB+12] et [SJ07].

Exemple 3.1

Nous opérons dans l'ontologie de la figure 2.2 les changements élémentaires suivants :

- AjouterConcept "Animal" ;
- AjouterSupConcept "Produit" à "Animal" ;
- SupprimerSupConcept "Produit" de "Viande" ;
- AjouterSupConcept "Animal" à "Viande"

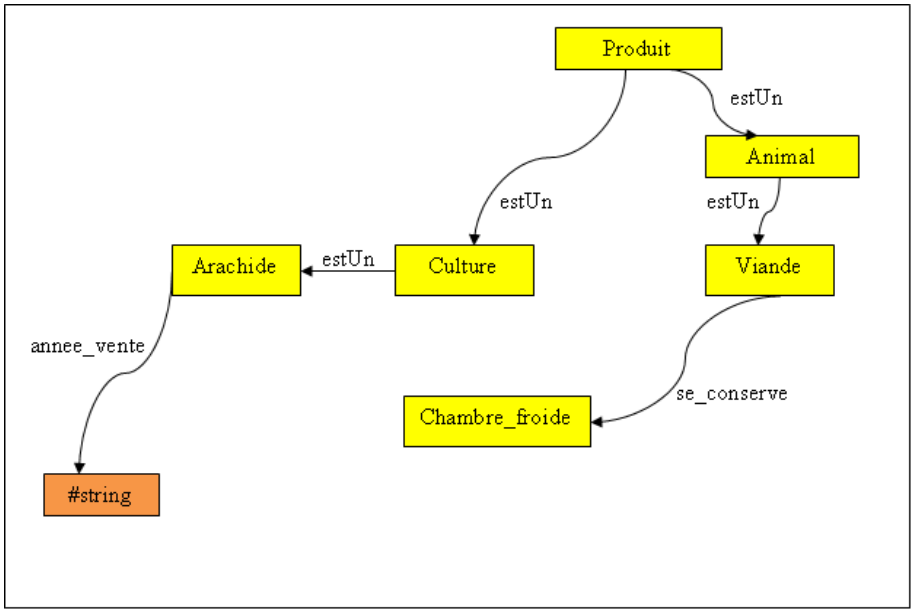

FIGURE 3.1 – Ontologie illustratrice modifiée par des changements simples

3.2.1.2 Changements complexes

Un changement complexe est une collection ordonnée de changements élémentaires [Rog08]. Pour beaucoup d'auteurs, il n'existe pas de différence entre changement complexe et changement composé. Stojanovic fait néanmoins cette différence et définit un changement composé comme un changement ontologique qui modifie (crée, renomme, change) le voisinage d'une entité ontologique et un changement complexe comme un changement qui peut être décomposé en une combinaison d'au moins de deux changements élémentaires ou composés [Sto04].

TABLE 3.1 – Changements simples

Identifiant	Changement	Signification
01	AjouterConcept	Ajouter un concept
02	AjouterSubConcept	Ajouter un super concept
03	RenommerConcept	Renommer un concept
04	RenommerSubConcept	Renommer un super concept
05	SupprimerConcept	Supprimer un concept
06	SupprimerSubConcept	Supprimer un super concept
07	CopierConcept	Copier un concept
08	AjouterPropriete	Ajouter une propriété (ou relation)
09	AjouterSubPropriete	Ajouter une super propriété
10	AjouterDomainePropriete	Ajouter un domaine à une propriété
11	AjouterRangePropriete	Ajouter un co-domaine à une propriété
12	AjouterRelationSymetrique	Ajouter une relation symétrique
13	AjouterRelationTransitive	Ajouter une relation transitive
14	AjouterRelationInverse	Ajouter une relation inverse
15	SupprimerPropriete	Supprimer une propriété
16	SupprimerSubPropriete	Supprimer une super propriété
17	SupprimerDomainePropriete	Supprimer un domaine d'une propriété
18	SupprimerRangePropriete	Supprimer un co-domaine d'une propriété
19	SupprimerRelationSymetrique	Supprimer une relation symétrique
20	SupprimerRelationTransitive	Supprimer une relation transitive
21	SupprimerRelationInverse	Supprimer une relation inverse
22	AjouterType	Ajouter un type
23	SupprimerType	Supprimer un type
24	AjouterCardinaliteMax	Ajouter une cardinalité maximale
25	SupprimerCardinaliteMax	Supprimer une cardinalité maximale
26	AjouterCardinaliteMin	Ajouter une cardinalité minimale
27	SupprimerCardinaliteMin	Supprimer une cardinalité minimale

Les changements complexes offrent plusieurs avantages comparés aux changements simples [KN03]. Ils permettent de formuler les requêtes de changements au plus haut niveau d'abstraction et facilitent la validation de l'évolution d'une ontologie. Ils sont plus explicites sémantiquement et permettent une analyse plus précise sur la conceptualisation de l'ontologie et sur les effets de référencement sémantiques. Par contre, la difficulté avec les changements complexes réside dans leur présence dans les logs, leur vocabulaire qui n'est pas uniformisé et leur nombre d'opérations illimité [PDT05]. Le tableau 3.2 donne une liste de changements complexes tirés de [Sto04], [Kle04], [SMMS02] et [Rog08].

Exemple 3.2
Effectuons une opération de changement complexe sur l'ontologie de la figure 3.1. Il s'agit d'éclater le concept "Culture" en sous concepts "Cereale" et "Vegetal". Ainsi nous obtenons l'ontologie donnée par la figure 3.2.

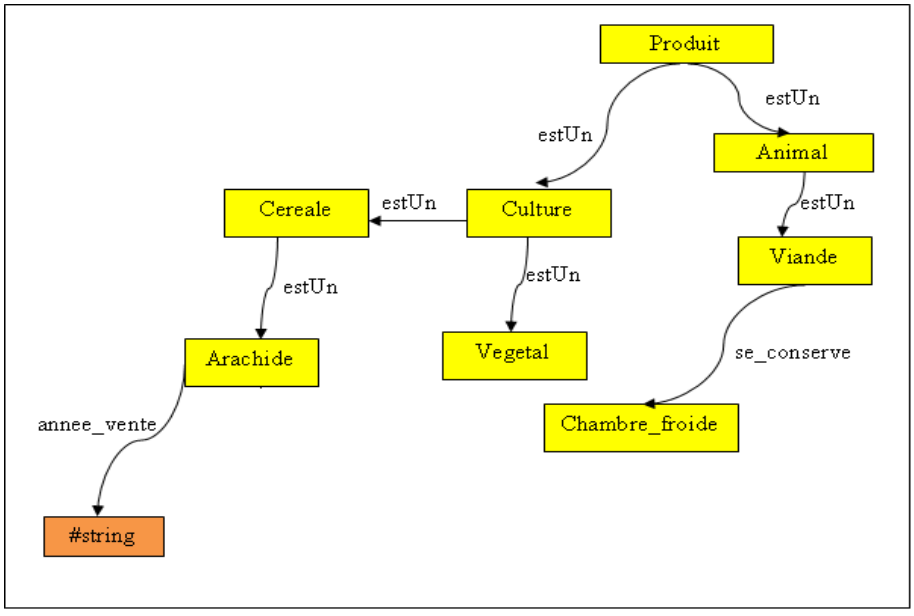

FIGURE 3.2 – Ontologie illustratrice modifiée par un changement complexe

L'identification du type de changement à opérer revêt d'une grande importance. En effet, les conséquences sur l'ontologie ne sont pas prises en charge de la même manière qu'il s'agisse d'un changement élémentaire ou d'un changement complexe. La section qui suit se consacre à l'étude des impacts de changement sur les entités de l'ontologie modifiée.

Table 3.2 – Changements complexes

Identifiant	Changement	Signification
01	FusionnerConcepts	Fusionner des concepts en un seul concept avec agrégation de toutes les instances
02	ExtraireSuperConcept	Créer un super concept commun à un ensemble de concepts non liés et transférer leurs propriétés communes à ce nouveau concept
03	AjouterSuperConcept	Ajouter un concept entre un concept x et tous ses sous-concepts
04	EclaterConcept	Eclater un concept en sous concepts et distribuer ses propriétés aux sous concepts
05	DupliquerConcept	Dupliquer un concept avec toutes ses propriétés
06	DeplacerHautConcept	Déplacer un concept vers le haut de la hiérarchie
07	DeplacerBasConcept	Déplacer un concept vers le bas de la hiérarchie
08	DeplacerProprietes	Déplacer des propriétés d'un concept vers un autre
09	FusionnerProprietesMemeConcept	Fusionner des propriétés liant les mêmes concepts
10	FusionnerProprietes	Fusionner des propriétés en une seule
11	EclaterPropriete	Eclater une propriété en plusieurs propriétés
12	DeplacerHautPropriete	Déplacer une propriété vers le haut de la hiérarchie
13	DeplacerBasPropriete	Déplacer une propriété vers le bas de la hiérarchie
14	DupliquerPropriete	Dupliquer une propriété avec le même domaine et le même co-domaine
15	DeplacerInstance	Déplacer une instance d'un concept à un autre

3.2.2 Analyse d'impacts de changements

3.2.2.1 Définitions

Un impact est l'effet ou l'impression d'une chose sur une autre (dictionnaire Larousse). Dans [AB93], un impact définit une partie destinée à être affectée, et ainsi exposée à l'inspection. L'analyse d'impact quant à lui peut se définir en tant qu'activité pour identifier ce qui doit être modifié afin d'accomplir un changement ou d'identifier les potentielles conséquences d'un changement. L'expression analyse d'impact trouve plusieurs définitions dans la littérature. Pfleeger et Bohner [PB90] la définissent en ces termes : "*l'évaluation des nombreux risques associés au changement, y compris les estimations des effets sur les ressources, les efforts et le calendrier.*" Pour Turver et Munro [TM94], l'analyse d'impact de changement est "*l'évaluation d'un changement, à l'aide du code source, d'un module sur les autres modules du système*". Elle détermine la portée d'un changement et fournit une mesure de sa complexité. Dans [AB93] on définit l'analyse d'impacts en identifiant les conséquences potentielles d'un changement, ou l'estimation de ce qui doit être modifié pour réaliser un changement ; on met l'accent sur l'estimation des impacts. Selon cette étude [Aji95], l'analyse d'impacts est "*l'activité consistant à identifier les objets à modifier pour accomplir un changement, et cela en mesurant les conséquences potentielles de l'exécution d'un changement*". Cette analyse peut représenter, comme indiqué dans [BB05], une mesure du coût d'un changement. Elle est aussi nécessaire pour comprendre la signification et les relations entre les éléments du changement d'une part et la structure de l'ontologie d'autre part, d'où l'existence de plusieurs approches en ce sens.

3.2.2.2 Approches d'analyse d'impacts

Les techniques d'analyse d'impacts se classent en deux catégories : l'analyse d'impacts à priori et l'analyse d'impacts à posteriori [Has09]. La première technique consiste à simuler des modifications sur un système et à étudier ses éventuelles conséquences. La seconde technique effectue les opérations de modification directement sur le système concerné et observe les impacts réels à l'issue de ces opérations.

De nombreuses approches sur l'évolution de l'ontologie se focalisent sur l'analyse d'impacts de changements à priori. Les techniques utilisées sont souvent tirées de l'analyse d'impacts d'un logiciel [Han72], [Aji95], [LO96]. Ajila dans [Aji95] présente un système d'analyse d'impacts et de propagation de changements génériques d'un logiciel. Selon cette approche, le problème de changement des objets d'un logiciel réside dans l'identification des victimes du changement, des types de changements qui peuvent s'opérer sur chaque objet, l'évaluation des effets d'un changement et l'acquisition de connaissances sur comment l'effectuer. L'étude dans [LO96] propose un algorithme automatique d'analyse d'impacts de changements dans un logiciel orienté-objet. L'approche permet d'estimer la taille d'un changement, de déterminer les relations susceptibles de propager des impacts et de les mesurer. Dans [MLM12], la méthode d'analyse d'impacts proposée identifie les impacts structurels et séman-

tiques. Cette méthode ascendante se déroule en deux phases : une analyse d'impacts des opérations de changement atomiques et une analyse d'impacts des changements composés qui inclue la suppression, le rééquilibrage et la transformation d'impacts dus à l'implémentation de deux ou plusieurs changements atomiques. L'approche d'analyse proposée dans [AJP12] se déroule aussi en deux phases : une première phase d'analyse d'impacts de changements atomiques et une seconde phase d'analyse concernant les changements complexes et incluant l'annulation, la neutralisation et la transformation d'impacts. Pour Goknil et collaborateurs [GKvdB08], l'analyse d'impacts peut se baser sur la traçabilité des besoins de changements. Ces spécifications sont utilisées comme des liens pour déterminer l'impact. Cependant, le manque de sémantique dans le traçage des liens peut rendre imprécis les résultats de cette analyse. Les propositions issues de [STL11] et de [STLB11] permettent de découvrir les impacts de modification d'ontologies et la maitrise des changements provoqués par la détection des chemins de propagation d'impacts. Seuls les changements atomiques sont pris en compte par cette approche. Dans [STB+12], une approche de modélisation des changements basée sur la sémantique axiomatique de Hoare est proposée. Elle permet d'étudier la satisfiabilité d'une opération de modification ontologique. Cependant, l'évaluation de l'impact sur les entités en dépendance n'est pas prise en compte.

En définitive, l'analyse d'impacts de changements représente une tâche assez difficile. En effet, effectuée de façon manuelle ou automatique, la ramification des changements rend souvent l'analyse d'impacts inappropriée car les changements ne sont pas toujours bien spécifiés. L'implémentation des modifications ontologiques validées déclenche souvent une phase de propagation d'impacts sur les artéfacts dépendants.

3.2.3 Propagation d'impacts de changements

La propagation d'impacts de changements est un processus qui peut affecter les composants de l'ontologie changeante mais aussi les ontologies dépendantes. Le terme "ripple effect" (effet d'onde) a été premièrement utilisé dans [Han72] pour décrire la manière qu'un changement dans un module doit nécessiter un autre changement dans d'autres modules. Un effet d'onde est causé par l'exécution d'un petit changement dans un système qui impacte plusieurs autres parties.

Dans [Tic95], Ticky présente un scénario de propagation des changements dans un logiciel consistant à extraire des informations de l'architecture du logiciel pour construire une version de base du programme. La méthode proposée sous le nom de "recompilation intelligente" utilise les dépendances entre les composants du système pour suggérer des composants qui dépendent d'un composant précédemment modifié. La recompilation intelligente fournit, d'après [Raj00], un algorithme de propagation de changement hautement spécialisé pour une propagation de bas en haut d'une entité E_j à une entité E_i. Une méthode de propagation d'impacts est décrite dans [KEEC05]. La technique permet de calculer le risque qu'un changement dans un composant affecte d'autres composants en analysant les changements indirects.

Cependant, cette technique, comme celles offrant une représentation visuelle des impacts, ne permettent pas d'explorer de manière efficiente les connexions directes et indirectes des composants. Dans [Raj97], la propagation est modélisée comme une séquence de clichés et chaque cliché représente un moment particulier du processus. Deux processus sont présentés dans cette proposition : le "change-and-fix" et le "top-down propagation". Le processus "change-and-fix" définit d'abord les types de dépendances entre les composants d'un programme en dépendances entrantes et dépendances sortantes. L'algorithme consiste ensuite à marquer les entités en dépendance de l'entité modifiée c'est-à-dire les entités situées dans le voisinage de l'entité modifiée. Le processus "top-down propagation" quant à lui est une version améliorée du premier processus qui assure l'inexistence de dépendances cycliques. La propostion dans [STBL12] procède par dérivation des changements pour analyser et suivre les impacts de modiciations sur l'ontologie. L'algorithme de propagation proposé se base sur la pré-condition, l'invariant et la post-condition de l'opération de changement. Selon cette approche :

- si la pré-condition et l'invariant sont vérifiés, alors l'opération peut s'exécuter sans propagation d'impacts ;
- si la pré-condition n'est pas vérifiée, alors l'opération n'est pas exécutée et il n'y a pas d'impact sur les composants ontologiques ;
- si la pré-condition est vérifiée et que l'invariant ne l'est pas alors l'opération est exécutée et il y a un processus de propagation d'impacts.

Ce processus est illustré par la figure 3.3 qui montre la propagation des impacts suite à la suppression du concept "Culture" de l'ontologie de la figure 3.2.

Nous remarquons à travers cet exemple que la suppression d'un concept d'une ontologie peut impacter plusieurs autres composants de la même ontologie voire d'autres ontologies dépendantes. C'est pour cette raison qu'une opération de modification doit être au préalable évaluée pour quantifier les inconsistances éventuelles qu'elle est susceptible de produire.

3.3 Mesures d'inconsistances

Dans cette section, nous parlons des mesures d'inconsistances dans leurs aspects définitions, caractéristiques et approches. Nous commençons par différencier les types d'inconsistances avant de voir comment elles peuvent être quantifiées.

3.3.1 Types d'inconsistances

La consistance est classée dans [HS05], en trois catégories :

- la consistance structurelle selon laquelle l'ontologie doit obéir aux contraintes du langage de définition de celle-ci ;
- la consistance logique qui exige que l'ontologie ne puisse contenir des informations contradictoires ;
- la consistance définie par les utilisateurs qui se conforme aux conditions explicitement définies par les utilisateurs et dépendant du contexte d'application.

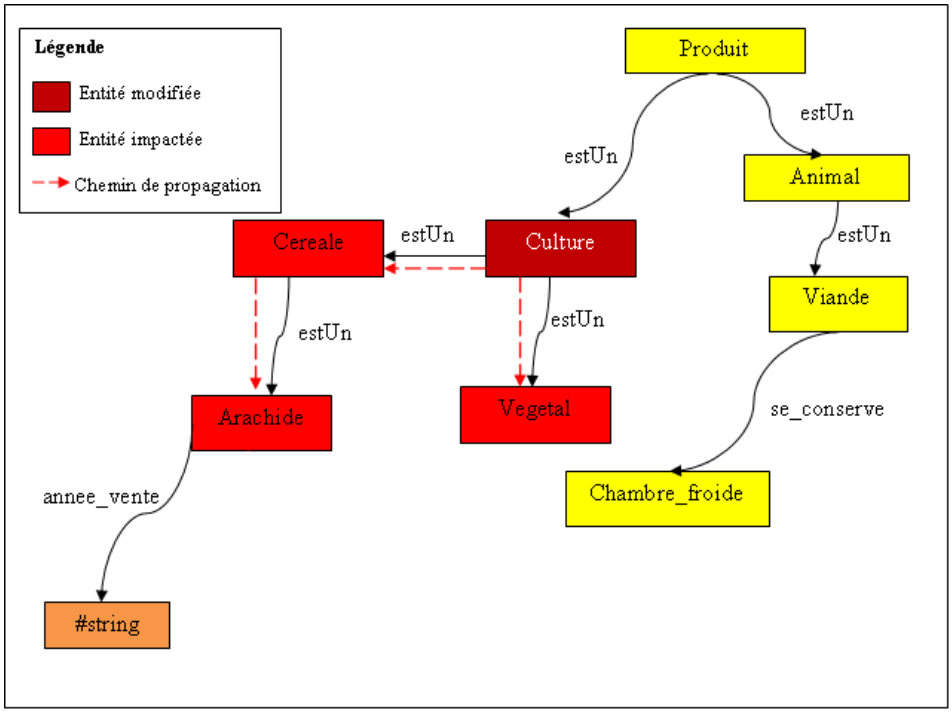

FIGURE 3.3 – Propagation d'impacts de changement sur l'ontologie illustratrice

Maedche et collaborateurs [MMS+02] mentionnent deux types d'inconsistances, les plus relatés dans la littérature : l'inconsistance syntaxique et l'inconsistance sémantique. L'inconsistance structurelle ou syntaxique survient quand une des contraintes du modèle ontologique est invalide. Alors que l'inconsistance sémantique apparait lorsque la signification d'une entité ontologique est changée suite aux modifications apportées à l'ontologie [Sto04]. Ma et Hitzter [MH10] différencient l'inconsistance de l'incohérence. Selon eux, une inconsistance indique qu'il y a des contradictions logiques de telle sorte que l'ontologie devienne triviale parce que toute conclusion suit cette contradiction. Une incohérence, par contre, suggère des erreurs de construction de l'ontologie car certains concepts sont nommés mais ne sont jamais instanciés. Cette différence est nuancée dans [Hun02] où l'incohérence dans une ontologie correspond à l'inconsistance dans une base de connaissances formée d'un ensemble fini de formules logiques classiques. En logique classique, la définition de l'inconsistance peut prendre plusieurs formes [HK05] :

- inconsistance comme raisonnement explosif qui permet une dérivation de chaque formule du langage ;
- inconsistance comme inférences en conflit : la base est inconsistante s'il existe deux inférences contradictoires pour une même formule ;
- inconsistance comme inférence d'une formule contradictoire, si une formule contradictoire est le modèle d'une base alors la base est inconsistante.

– inconsistance comme raisonnement trivial, si α est une inférence triviale[1] d'une base de connaissances alors cette dernière est inconsistante ;
– inconsistance comme absence d'un modèle : si une base de connaissances n'a pas de modèle alors elle est inconsistante.

Dans cette thèse, les approches de mesures d'inconsistances relatées concernent plus les inconsistances structurelles que celles sémantiques.

3.3.2 Définition d'une mesure d'inconsistances

Une mesure d'inconsistances quantifie la contribution de chaque axiome ou assertion d'une base de connaissances dans l'ensemble des inconsistances. Elle permet entre autre de prévoir un scénario de résolution des inconsistances découlant d'une opération de changement. Une mesure d'inconsistances donne aussi une vue sur la sévérité de l'inconsistance dans le système, renseignant si l'inconsistance doit être traitée ou ignorée. Dans [GH11], la mesure d'inconsistances proposée sert d'heuristiques pour sélectionner les formules adéquates à modifier afin de rendre un système consistant. Pour une base de connaissances, une mesure d'inconsistances peut être de trois types selon l'étude [QH07]. Pour les mesures qui prennent en compte le nombre de formules de l'inconsistance, moins de formules dans une inconsistance signifie un grand degré d'inconsistances. Les mesures prenant en compte la proportion d'atomes affectée par l'inconsistance attestent que plus qu'il y a de formules impliquées dans l'inconstance, plus le degré d'inconsistances est important. En fin pour les mesures considérant le nombre de conflits auxquels chaque formule est impliquée, si une formule est présente dans plusieurs conflits, il se produira alors un grand degré d'inconsistances.

Les mesures d'inconsistances qui calculent le pourcentage d'axiomes impliquées dans l'inconsistance sont appelées des mesures basées sur la syntaxe, et celles qui évaluent le pourcentage d'assertions atomiques impliquées dans l'inconsistance sous certains modèles para-consistants représentent les mesures basées sur la sémantique [MH10].

Toutefois, une mesure doit obéir à un certain nombre de principes pour pouvoir être définie comme une mesure d'inconsistances [HK10], [Thi13] et [Thi14]. Ces principes sont donnés dans la sous section suivante.

3.3.3 Caractéristiques d'une mesure d'inconsistances

Les principes d'une mesure d'inconsistances sont des propriétés que doit vérifier la mesure pour être considérée comme outil de quantification d'impacts de changements. Ces propriétés sont la consistance, la monotonicité, la super-additivité, la faible indépendance, l'indépendance, la pénalité, l'impertinence de syntaxe, la MI-séparabilité, la normalisation et la dépendance. Il n'est pas fréquent d'établir une mesure qui vérifie toutes ces propriétés, mais la plupart d'entre elles doivent être

1. Une inférence triviale est une inférence α d'une base Δ telle que α n'est pas une tautologie et $\text{Atoms}(\Delta) \cap \text{Atoms}(\alpha) \neq \varnothing$

supportées pour que la mesure soit valide.
Soient K et K' deux bases de connaissances, α et β deux formules de K et I une mesure d'inconsistances. Les propriétés de la mesure I se définissent ainsi :

- **Consistance** : K est consistant si et seulement si $I(K) = 0$.
- **Monotonicité** : $I(K) \leq I(K \cup \{\alpha\})$.
- **Super-additivité** : Si $K \cap K' = \varnothing$ alors $I(K \cup K') \geq I(K) + I(K')$.
- **Faible indépendance** : Si $\alpha \in K$ est une formule sûre (voir définition 1) alors $I(K) = I(K \setminus \{\alpha\})$.
- **Indépendance** : Si $\alpha \in K$ est une formule libre (voir définition 3) alors $I(K) = I(K \setminus \{\alpha\})$.
- **Pénalité** : Si $\alpha \in K$ n'est pas libre alors $I(K) > I(K \setminus \{\alpha\})$.
- **Impertinence de syntaxe** : Si $K \equiv K'$ alors $I(K) = I(K')$.
- **MI-séparabilité** : Si $MI(K \cup K') = MI(K) \cup MI(K')$ et $MI(K) \cap MI(K') = \varnothing$ alors $I(K \cup K') = I(K) + I(K')$ (voir définition 2 pour MI).
- **Normalisation** : $I(K) \in [0\,;\,1]$.
- **Dominance** : Si $\alpha \models \beta$ et $\alpha \nvdash \bot$ alors $I(K \cup \{\alpha\}) \geq I(K \cup \{\beta\})$.

Une mesure est appelée mesure d'inconsistances basique si elle vérifie la consistance, la monotonicité, l'indépendance et la dominance [HK+08]. D'après les travaux [GH13] et [Thi14], la consistance, la monotonicité et l'indépendance suffisent pour caractériser une mesure d'inconsistances. Thimm montre dans [Thi13] que la super-additivité implique la monotonicité, l'indépendance implique la faible indépendance et la MI-séparabilité implique l'indépendance.

Définition 1 (Formule sûre) *Soit α une formule de K.*
α est sûr dans K si et seulement si $At(\alpha) \cap At(K \setminus \{\alpha\}) = \varnothing$
avec At la signature d'une proposition atomique.

Définition 2 (Ensemble minimum inconsistant (MI)) .
M est un ensemble minimum inconsistant si M est inconsistant et tout $M' \subseteq M$ est consistant.

Définition 3 (Formule libre) .
Une formule $\alpha \in K$ est libre dans K si et seulement si $\alpha \notin M$, $\forall\ M \in MI(K)$.
$MI(K)$ est l'ensemble des sous ensembles minimums inconsistants de K.

3.3.4 Mesures d'inconsistances classiques

Dans cette partie, nous présentons les mesures d'inconsistances classiques [GH13], [JR13], [Thi16a] sur lesquelles on se base souvent pour exprimer des approches de mesures. Tous les exemples que nous citons s'appliquent aux bases de connaissances K_1 et K_2 sur l'ensemble E définis comme suit :

$K_1 = \{\neg a \vee c,\ a \wedge b,\ a \wedge \neg b,\ \neg c\}$

$K_2 = \{a, \neg a, b, \neg b\}$

E = {a, b, c}, avec :

a = Culture $\sqsubseteq$ Produit

b = $\exists$ se_conserve $\sqsubseteq$ Viande

c = ρ(annee_vente) $\sqsubseteq$ xsd :string

3.3.4.1 Mesure d'inconsistances drastique

Définition 4 (Mesure d'inconsistances drastique) *Soit I_d : $\mathbb{K} \rightarrow [0, \infty)$ une fonction définie par :*

$$I_d(K) = \begin{cases} 0 & \text{si } K \text{ est consistant} \\ 1 & \text{si } K \text{ est inconsistant} \end{cases}$$

pour $K \in \mathbb{K}$.
La fonction I_d définit une mesure appelée mesure d'inconsistances drastique.

La mesure d'inconsistances drastique [HK+08] ne permet pas cependant d'estimer le niveau de sévérité des inconsistances car elle ne rend qu'une décision binaire : consistant et inconsistant. Elle sert souvent pour exprimer d'autres mesures d'inconsistances.

Exemple 3.3
Sur les bases de connaissances K_1 et K_2, la mesure d'inconsistances drastique donne :

$I_d(K_1) = I_d(\{\neg a \vee c, a \wedge b, a \wedge \neg b, \neg c\}) = 1$

$I_d(K_2) = I_d(\{a, \neg a, b, \neg b\}) = 1$

car K_1 et K_2 ont des formules en conflit ($a \wedge b$ et $a \wedge \neg b$ pour K_1 et b et $\neg b$ ou a et $\neg a$ pour K_2).

Propriétés de la mesure
La mesure d'inconsistances drastique satisfait la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance, la faible indépendance et la normalisation [Thi12]. Elle ne satisfait pas par contre la MI-séparabilité, la super-additivité, la pénalité et la dépendance. La fonction I_d n'est pas non plus continue sur son domaine d'application.

3.3.4.2 Mesure d'inconsistances MI

Définition 5 (Mesure d'inconsistances MI) *Soit* $I_{MI} : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$$I_{MI}(K) = |MI(K)|$$

pour $K \in \mathbb{K}$.
La fonction I_{MI} *définit une mesure appelée mesure d'inconsistances MI.*

La mesure d'inconsistances MI [HK+08], [Thi12] se base sur les sous-ensembles minimums inconsistants et est motivée par le fait que ces derniers contiennent par définition les inconsistances d'une base de connaissances.

Exemple 3.4
Sur les base de connaissances K_1 et K_2, nous avons :

$MI(K_1) = \{\{a \wedge b, a \wedge \neg b\}\}$

$MI(K_2) = \{\{a, \neg a\}, \{b, \neg b\}\}$

Alors cette mesure donne :

$I_{MI}(K_1) = 1$

$I_{MI}(K_2) = 2$

Propriétés de la mesure
La mesure d'inconsistances MI satisfait la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance, la faible indépendance, la MI-séparabilité, la super-additivité, la pénalité [Thi12]. Par contre, la normalisation, et la continuité ne sont pas vérifiées par la fonction I_{MI}.

3.3.4.3 Mesure d'inconsistances MI^C

Définition 6 (Mesure d'inconsistances MI^C) *Soit* ${I_{MI}}^C : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$${I_{MI}}^C(K) = \sum_{\mathcal{M} \in MI(K)} \frac{1}{|\mathcal{M}|}$$

pour $K \in \mathbb{K}$.
La fonction ${I_{MI}}^C$ *définit une mesure appelée mesure d'inconsistances* MI^C.

La mesure d'inconsistances MI^C [HK+08] est née de la remarque que plus un ensemble minimal inconsistant est grand moins elle contribue à la valeur de l'inconsistance. C'est pour cela qu'elle s'exprime par l'inverse du cardinal de tous les sous-ensembles minimums inconsistants.

Exemple 3.5
Rappelons que :

MI(K_1) = {{a∧b, a∧¬b}}

MI(K_2) = {{a, ¬a}, {b, ¬b}}

Alors appliquée sur K_1 et K_2, la mesure donne :

$I_{MI}{}^{C}(K_1) = 1/(|\{a\wedge b, a\wedge\neg b\}|) = 1/2$

$I_{MI}{}^{C}(K_2) = 1/(\{a,\neg a\}) + 1/(\{b,\neg b\}) = 1/2 + 1/2 = 1$

Propriétés de la mesure
La mesure d'inconsistances MI^C a les mêmes propriétés que la mesure d'inconsistances MI [Thi12]. Elle satisfait donc la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance, la faible indépendance, la MI-séparabilité, la superadditivité et la pénalité.

3.3.4.4 Mesure Df-inconsistances

La mesure d'inconsistances MI^C a été étendue par Mu et son équipe [MLJ11] pour prendre en compte la distributivité de l'inconsistance dans la base de connaissances en introduisant dans la mesure le ratio d'ensembles minimums inconsistants d'une taille donnée sur le nombre de sous-ensembles minimums inconsistants et de sous-ensembles minimums consistants de cette même taille, défini comme suit :

$$R_i(K) = \begin{cases} 0 & \text{si } |MI^{(i)}(K)| + |CN^{(i)}(K)| = 0 \\ \frac{|MI^{(i)}(K)|}{|MI^{(i)}(K)|+|CN^{(i)}(K)|} & \text{sinon} \end{cases}$$

pour i = 1, ..., |K|
avec
$MI^{(i)}(K) = \{M \in MI(K) : |M| = i\}$ (l'ensemble des sous-ensembles inconsistants de taille i)
et
$CN^{(i)}(K) = \{C \in K : |C| = i \wedge C \nvdash \bot\}$ (l'ensemble des sous-ensembles consistants de taille i)

Définition 7 (Mesure Df-inconsistances) *Soit* $I_{Df} : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$$I_{Df}(K) = 1 - \prod_{i=1}^{|K|}(1 - \frac{R_i(K)}{i})$$

pour $K \in \mathbb{K}$.
La fonction $I_{Df}(K)$ définit une mesure appelée mesure Df-inconsistances.

Pour chaque valeur de i, la mesure Df-inconsistances définit une instance d'une série de mesures proposée dans [MLJ11].

Exemple 3.6
Nous avons toujours :

$\mathrm{MI}(\mathrm{K}_1) = \{\{a \wedge b,\ a \wedge \neg b\}\}$

$\mathrm{MI}(\mathrm{K}_2) = \{\{a,\ \neg a\},\ \{b,\ \neg b\}\}$

Donc :

$\mathrm{MI}^1(\mathrm{K}_1) = \varnothing$

$\mathrm{MI}^1(\mathrm{K}_2) = \varnothing$

$\mathrm{MI}^2(\mathrm{K}_1) = \{\{a \wedge b,\ a \wedge \neg b\}\}$

$\mathrm{MI}^2(\mathrm{K}_2) = \{\{a,\ \neg a\},\ \{b,\ \neg b\}\}$

$\mathrm{MI}^3(\mathrm{K}_1) = \varnothing$

$\mathrm{MI}^3(\mathrm{K}_2) = \varnothing$

$\mathrm{MI}^4(\mathrm{K}_1) = \varnothing$

$\mathrm{MI}^4(\mathrm{K}_2) = \varnothing$

Nous avons aussi :

$\mathrm{CN}^1(\mathrm{K}_1) = \{\{\neg a \vee c\},\ \{a \wedge b\},\ \{a \wedge \neg b\},\ \{\neg c\}\}$

$\mathrm{CN}^1(\mathrm{K}_2) = \{\{a\},\ \{\neg a\},\ \{b\},\ \{\neg b\}\}$

$\mathrm{CN}^2(\mathrm{K}_1) = \{\{\neg a \vee c,\ a \wedge b\},\ \{\neg a \vee c,\ a \wedge \neg b\},\ \{\neg a \vee c,\ \neg c\},\ \{a \wedge b,\ \neg c\},\ \{a \wedge \neg b, \neg c\}\}$

$\mathrm{CN}^2(\mathrm{K}_2) = \{\{a,\ b\},\ \{\neg a,\ b\},\ \{\neg b,\ a\},\ \{\neg a,\ \neg b\}\}$

$\mathrm{CN}^3(\mathrm{K}_1) = \{\{\neg a \vee c,\ a \wedge b,\ \neg c\},\ \{\neg a \vee c,\ a \wedge \neg b,\ \neg c\}\}$

$\mathrm{CN}^3(\mathrm{K}_2) = \varnothing$

$CN^4(K_1) = \varnothing$

$CN^4(K_1) = \varnothing$

Le calcul des ratios pour les bases de connaissances K_1 et K_2 donne ainsi :

$R_1(K_1) = 0$

$R_2(K_1) = 1/(1+5) = 1/6$

$R_3(K_1) = 0$

$R_4(K_1) = 0$

$R_1(K_2) = 0$

$R_2(K_2) = 2/(2+4) = 1/3$

$R_3(K_2) = 0$

$R_4(K_2) = 0$

En définitive, la mesure Df-inconsistances sur les bases K_1 et K_2 donne comme résultats :

$I_{Df}(K_1) = 1 - (1 - (R_1(K_1))/1)\ (1 - (R_2(K_1))/2)\ (1 - (R_3(K_1))/3)\ (1 - (R_4(K_1))/4)$

$I_{Df}(K_1) = 1 - (1\text{-}0)(1\text{-}\ 1/12)(1\text{-}0)(1\text{-}0)$

$I_{Df}(K1) = 1 - 11/12 = 1/12$

$I_{Df}(K_2) = 1 - (1 - (R_1(K_2))/1)\ (1 - (R_2(K_2))/2)\ (1 - (R_3(K_2)\ )/3)\ (1 - (R_4(K_2))/4)$

$I_{Df}(K_2) = 1 - (1\text{-}0)(1\text{-}\ 1/6)(1\text{-}0)(1\text{-}0)$

$I_{Df}(K_2) = 1 - 5/6 = 1/6$

Propriétés de la mesure

La mesure Df-inconsistances [Thi16b] vérifie la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance, la faible indépendance, la dominance, la MI-séparabilité, la super-additivité et la pénalité.

3.3.4.5 Mesure d'inconsistances problématique

Définition 8 (Mesure d'inconsistances problématique) *Soit* $I_p : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$$I_p(K) = \left| \bigcup_{M \in MI(K)} M \right|$$

pour $K \in \mathbb{K}$.
La fonction I_p *définit une mesure appelée mesure d'inconsistances problématique.*

La mesure d'inconsistances problématique [GH11] utilise le nombre de formules présentes dans les sous-ensembles minimums inconsistants appelées formules problématiques comme mesure de l'inconsistance.

Exemple 3.7
Etant donné que les sous ensembles minimums inconsistants de K_1 et K_2 sont les suivants :

$MI(K_1) = \{\{a \wedge b, a \wedge \neg b\}\}$

$MI(K_2) = \{\{a, \neg a\}, \{b, \neg b\}\}$
alors la mesure d'inconsistances problématique sur K_1 et K_2 donne :

$I_p(K_1) = 2$

$I_p(K_2) = 4$

Propriétés de la mesure
La mesure d'inconsistances problématique est une mesure qui satisfait la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance et la faible indépendance.

3.3.4.6 Mesure d'inconsistances MC

Définition 9 (mesure d'inconsistances MC) *Soit* $I_{MC} : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$$I_{MC}(K) = |MC(K)| + |SC(K)| - 1$$

pour $K \in \mathbb{K}$.
La fonction I_{MC} *définit une mesure appelée mesure d'inconsistances MC.*

La mesure d'inconsistances MC [GH11] se base sur le sous-ensemble maximum consistant MC et l'ensemble des formules contradictoires SC de K définis ci-dessous.

$MC(K) = \{K' \subseteq K \mid K' \nvDash \perp \wedge \forall K'' \supseteq K : K'' \models \perp\}$ et

$SC(K) = \{\phi \in K \mid \phi \models \bot\}$

Exemple 3.8
Nous avons :

$MC(K_1) = \{\{\neg a \vee c, a \wedge b, \neg c\}, \{\neg a \vee c, a \wedge \neg b, \neg c\}\}$

$SC(K_1) = \varnothing$

$MC(K_2) = \{\{a, b\}, \{\neg a, b\}, \{\neg b, a\}, \{\neg a, \neg b\}\}$

$SC(K_2) = \varnothing$
Ainsi la mesure d'inconsistances MC sur K_1 et K_2 donne :

$I_{MC}(K_1) = 2 - 1 = 1$

$I_{MC}(K_2) = 4 - 1 = 3$

Propriétés de la mesure La mesure d'inconsistances MC vérifie la consistance, l'impertinence de syntaxe, la monotonicité, la MI-séparabilité.

3.3.4.7 Mesure nc-inconsistances

Définition 10 (mesure nc-inconsistances) *Soit $I_{nc} : \mathbb{K} \to [0, \infty)$ une fonction définie par :*

$$I_{nc}(K) = |K| - max\{n \mid \forall K' \subseteq K : |K'| = n \Rightarrow K' \not\models \bot\}$$

pour $K \in \mathbb{K}$.
La fonction I_{nc} définit une mesure appelée mesure nc-inconsistances.

La mesure nc-inconsistances [Thi16b] se définit comme le cardinal de K auquel on retranche le maximum n des cardinaux des sous-ensembles consistants de K de taille n.

Exemple 3.9
Sur K_1 et K_2, cette mesure donne les résultats suivants :

$I_{nc}(K_1) = 4 - 1 = 3$

$I_{nc}(K_2) = 4 - 1 = 3$
car :

$MI^1(K_1) = \varnothing$

$MI^1(K_2) = \varnothing$

$MI^2(K_1) = \{\{a \wedge b, a \wedge \neg b\}\}$

$MI^2(K_2) = \{\{a, \neg a\}, \{b, \neg b\}\}$

$MI^3(K_1) = \varnothing$

$MI^3(K_2) = \varnothing$

$MI^4(K_1) = \varnothing$

$MI^4(K_2) = \varnothing$

Propriétés de la mesure
La mesure nc-inconsistances vérifie la consistance, l'impertinence de syntaxe, la monotonicité et l'indépendance.

3.3.4.8 Mesure η-inconsistances

Définition 11 (Mesure η-inconsistances) *Soit $I_\eta : \mathbb{K} \rightarrow [0, \infty)$ une fonction définie par :*

$$I_\eta(K) = 1 - max\{\eta \mid \exists \hat{P} \in \mathcal{F}^2 : \forall c \in K : \hat{P}(c) \geq \eta\}$$

pour $K \in \mathbb{K}$.
La fonction I_η définit une mesure appelée mesure η-inconsistances.

avec $\hat{P} : \mathcal{F}(\text{At}) \rightarrow [0, 1]$ est une fonction de probabilité définie par :

$$\hat{P}(c) = \sum_{P \in \mathcal{F}(At), P \models c} \hat{P}(P)$$

où $\mathcal{F}(\text{At})$ représente l'ensemble des fonctions de probabilité sur At (un ensemble de propositions).
La mesure η-inconsistances [Kni02] se base sur le nombre minimal de formules de la base de connaissances auxquelles elle maximise leur probabilité.

Exemple 3.10
Soit $P \in \mathbb{P}(\{a, b, c\})$ une fonction de probabilité définie par :

$P(abc) = P(\overline{a}\overline{b}\overline{c}) = 0.5$
Alors nous avons les probabilités suivantes :

$P(\neg a \vee c) = P(a \wedge b) = P(a \wedge \neg b) = P(\neg c) = 0.5$

P(a) = P(¬a) = P(b) = P(¬b) = 0.5
Remarquons que :

P(θ) $\geq$ 0.5 $\forall\theta \in$ $K_1 \cup K_2$
Ainsi la mesure η-inconsistances sur K_1 et K_2 donne :

$I_\eta(K_1) = I_\eta(K_2)$ = 1 - 0.5 = 0.5

Propriétés de la mesure
La mesure η-inconsistances [Kni02] vérifie la consistance, l'impertinence de syntaxe, la monotonicité, l'indépendance, la faible indépendance et la normalisation. Elle ne satisfait pas par contre la MI-séparabilité, la super-additivité, la pénalité et la dépendance et n'est pas continue.

3.3.4.9 Mesure d'inconsistances mv

Définition 12 (Mesure d'inconsistances mv) *Soit* $I_{mv} : \mathbb{K} \rightarrow [0, \infty)$ *une fonction définie par :*

$$I_{mv}(K) = \frac{| \cup_{M \in MI(K)} At(M) |}{| At(K) |}$$

pour $K \in \mathbb{K}$.
La fonction I_{mv} *définit une mesure appelée mesure d'inconsistances mv.*

La mesure d'inconsistances mv [XM12] se base sur le nombre de propositions de l'ensemble des propositions At qui participent à l'inconsistance. La mesure est égale au rapport du nombre de propositions des sous-ensembles minimums inconsistants sur le nombre total de propositions.

Exemple 3.11
Considérons toujours les bases de connaissances K_1 et K_2 telles que :

$K_1 = \{\neg a \vee c, a \wedge b, a \wedge \neg b, \neg c\}$

$K_2 = \{a, \neg a, b, \neg b\}$
Les ensembles minimums inconsistants de ces deux bases sont les suivantes :

$MI(K_1) = \{\{a \wedge b, a \wedge \neg b\}\}$

$MI(K_2) = \{\{a, \neg a\}, \{b, \neg b\}\}$
Ainsi, la mesure d'inconsistances mv sur K_1 et K_2 est donnée par les valeurs ci-dessous :

$I_{mv}(K_1) = (|a,b|)/(|a,b,c|) = 2/3$

$I_{mv}(K_2) = (|a,b|)/(|a,b|) = 2/2 = 1$

Propriétés de la mesure
La mesure d'inconsistances mv vérifie la consistance, l'impertinence de syntaxe, la monotonicité et l'indépendance.

Ces mesures d'inconsistances classiques ne permettent pas souvent d'avoir une estimation adéquate de la sévérité d'une opération de modification ontologique. En effet, elles ne prennent pas en compte certains paramètres importants à la mesure telle que la spécificité de la base de connaissances sur laquelle se produit l'inconsistance. Elles sont utilisées, dans plusieurs cas, pour définir d'autres approches de mesures.

3.3.5 Approches de mesures d'inconsistances

La plupart des mesures proposées dans la littérature traitent l'inconsistance dans la logique propositionnelle [HK10], [MLJ11], [GH13]. Pour [HK10], l'étude concerne deux types d'approches de mesures : les approches qui comptent le nombre de formules nécessaires pour produire une inconsistance et les approches s'intéressant à la proportion du langage touchée par l'inconsistance [Hun02], [KLM03], [GH06]. Pour la première famille de mesures, plus qu'il y a de formules moins il y a d'inconsistance [Kni02]. Le problème pour ces méthodes réside dans l'impossibilité d'avoir une inspection fine du contenu des formules. Concernant la seconde famille, la distribution de la contradiction au sein de la formule n'est pas tenue en compte. Dans [HK05], cette classification d'approches de mesures se fait en termes d'approches basées sur la base de connaissances et d'approches basées sur les formules de la base de connaissances. Pour ces dernières, les mesures sont en général définies en utilisant les sous-ensembles minimums inconsistants et les modèles de coalition de jeu ou Valeur de Shapley [Win02]. En ce sens, Hunter et Konieczny proposent [HK+08] une approche qui se base sur les sous-ensembles minimums inconsistants et la valeur de Shapley. La mesure est égale à la valeur d'inconsistances minimale, cependant l'approche n'est pas implémentée. La valeur de Shapley se retrouve dans l'approche [DHS07] où la mesure concerne plutôt les clauses que les axiomes. Les auteurs de ces travaux justifient cela par le fait que les clauses ont une granularité plus fine que les axiomes.

L'approche proposée dans [ZHQ+09] permet de mesurer l'inconsistance d'une ontologie DL-Lite. La mesure utilise les "three-value semantics" dans sa définition et l'algorithme est de complexité PTime. Dans cette famille de la logique descriptive, Ma et Hitzler [MH10] définissent une mesure d'inconsistances et d'incohérences dans une ontologie DL basée sur la distance. L'idée est de calculer la distance entre une ontologie DL et ses interprétations. La valeur trouvée permet de définir un degré de déviation qui indique qu'il y a inconsistances sémantiques/incohérences terminologiques ou pas. Toutefois, l'approche est drastique dans le sens où elle ne rend qu'une décision binaire : consistant/inconsistant. La proposition de [GH13] représente aussi

une estimation basée sur la distance. Elle mesure la distance entre deux modèles d'une base de connaissances logique et dilate les points représentant ces modèles de sorte que leur intersection ne soit plus vide. Le travail de [XM12] aboutit à deux mesures d'inconsistances qui combinent des approches sémantiques et syntaxiques. La première mesure se définit avec le nombre de variables des sous-ensembles minimums inconsistants tandis que la deuxième mesure utilise les sous-ensembles qui, extraits de la base de connaissances, donnent un ensemble minimum satisfiable. L'étude montre que ces deux mesures sont équivalentes et ont une complexité polynomiale. Deux approches de mesure d'incohérence d'une ontologie DL indépendantes des langages logiques spécifiques sont proposées dans [QH07]. Les mesures se basent sur la notion de fonction de score servant à ordonner les concepts insatisfiables et les axiomes terminologiques. Les approches ne permettent pas cependant de gérer des ontologies avec un nombre important d'axiomes terminologiques en conflit.

D'autres types de mesures investissent la théorie de Dempster-shafer telle que l'approche [DYH+14] qui calcule l'inconsistance en se basant sur la fonction de croyance et la fonction de plausibilité. La mesure du gain de consistance et de la perte d'informations est utilisée dans [GH11] pour une résolution graduelle des inconsistances de la base de connaissances. L'approche de résolution consiste à définir d'abord une fonction de résolution et à mesurer la perte d'informations après une étape de résolution correspondant à une suppression de formule, un remplacement de formule ou un éclatement de formule. Thimm [Thi14], [Thi16b] propose une mesure d'inconsistances basée sur la notion de "hitting set" applicable aux bases de connaissances de grande taille. Le "hitting set" est défini comme un ensemble minimum d'interprétation classique telle que chaque formule de la base est satisfaite par au moins un élément de l'ensemble. La mesure est une approximation de la mesure η-inconsistances.

Toutes ces approches de mesures ont pour rôle de donner une idée sur la sévérité des inconsistances produites par une opération de modification ontologique et ainsi envisager une stratégie de résolution de ces inconsistances. La section qui suit présente quelques approches de résolution d'inconsistances.

3.4 Résolution d'inconsistances

Les résultats de recherche concernant la résolution d'inconsistances ne sont pas très nombreux. La résolution d'effets de changements reste un domaine de recherche à explorer. Nous pouvons citer cependant l'étude de [HS05] qui présente une méthode de détection et de résolution d'inconsistances utilisant une stratégie. La résolution se réalise en deux étapes : d'abord la détection des changements puis la génération de changements additionnels pour garantir la consistance. Pour résoudre les inconsistances logiques, les auteurs déterminent un ensemble d'axiomes à supprimer pour obtenir la consistance logique avec un impact minimal. L'approche est semi-automatique et concerne le sous-langage OWL Lite. La définition des conditions de

consistance est arbitraire, ce qui constitue une limite pour cette approche. La proposition dans [Liu06] est un algorithme de résolution d'inconsistances dans une base de connaissances stratifiée en logique descriptive. Un opérateur de révision est défini à la fois pour la syntaxe et pour la sémantique afin d'affaiblir le déséquilibre de la base. Silva et Chniti [BDSC13] proposent deux approches : une approche syntaxique (Model-Detect-Repair) et une approche sémantique basée sur une description formelle des règles de production et une règle de programmation d'inconsistances pour résoudre les impacts de changements ontologiques. Quatre approches de traitement des incohérences dans les ontologies DL : l'évolution cohérente de l'ontologie, la répartition des inconsistances, le raisonnement en présence d'inconsistances et le raisonnement en multi-version sont proposées dans [HS05]. Le modèle de résolution consiste à supprimer de façon itérative des axiomes jusqu'à obtenir une ontologie consistante. Ce travail est réalisé avec une fonction de sélection. Bell [BQL07] fait une étude sur les approches de résolution d'inconsistances et analyse leur utilisabilité. Il souligne que les raisonneurs DL, RACER [HM01] et FaCT [Hor98] détectent des inconsistances logiques mais seulement en fournissant des listes de classes insatisfiables. Dans [LJL+13], on fait recours à la révision des croyances pour résoudre les inconsistances. En effet, lorsqu'une nouvelle croyance est ajoutée, une inconsistance se produit entre la nouvelle croyance ou ses résultats logiques et les croyances de l'ensemble de départ. Pour Plessers et De Troyer [PDT06], un algorithme de sélection d'axiomes d'une ontologie causant des inconsistances et un ensemble de règles peuvent permettre à l'ingénieur de l'ontologie de résoudre les inconsistances détectées. L'approche [MTFH13] parvient à traiter les incohérences telles que la redondance des données, les nœuds isolés découlant des changements dérivés, les individus orphelins et les axiomes contradictoires.

Malgré l'existence de ces différentes approches, la résolution des inconsistances reste partiellement traitée : les changements complexes ne sont pas pris en compte de la même façon que les inconsistances dans les ontologies volumineuses.

3.5 Synthèse

L'évolution d'une ontologie que Stojanovic et collaborateurs définissent comme la modification appropriée de l'ontologie et la propagation consistante des changements dans les artéfacts dépendants représente une phase importante dans le cycle de vie d'une ontologie. La gestion de cette évolution consiste d'abord à définir les changements à opérer sur l'ontologie, changements qui peuvent être élémentaires ou complexes. Cette phase est suivie de la phase d'analyse d'impact dont le rôle est d'identifier ce qui doit être modifié pour accomplir un changement ou pour identifier les potentielles conséquences. La propagation des effets d'onde, en troisième phase, est un processus qui peut affecter les composants de l'ontologie changeante mais aussi les ontologies dépendantes.

Gérer l'évolution d'une ontologie consiste aussi à mesurer l'inconsistance susceptible d'être produite par une opération de modification ontologique. Une mesure

d'inconsistances quantifie la contribution de chaque axiome ou assertion d'une base de connaissances dans l'ensemble des inconsistances. Elle permet entre autre de prévoir un scénario de résolution des inconsistances découlant d'un changement. Cependant, pour être classée comme outil d'estimation d'inconsistances, la mesure doit vérifier certaines des propriétés que sont la consistance, la monotonicité, la super-additivité, la faible indépendance, l'indépendance, la pénalité, l'impertinence de syntaxe, la MI-séparabilité, la normalisation et la dépendance. Même si la plupart des approches de mesures traitent l'inconsistance dans la logique propositionnelle, elles utilisent souvent dans leur définition les mesures d'inconsistances classiques telles que mesure la d'inconsistances MI ou la mesure η-inconsistances.

S'il faut noter que certaines incohérences telles que la redondance des données, les nœuds isolés découlant des changements dérivés, les individus orphelins et les axiomes contradictoires sont prises en charge, il faudra cependant signaler que la résolution des inconsistances reste encore partielle.

3.6 Conclusion

Dans ce chapitre, nous avons fait un état de l'art sur la gestion des impacts de changements d'une ontologie sur les artéfacts dépendants. Nous avons d'abord donné une classification des changements en changements simples et changements complexes puis une liste des modifications ontologiques les plus usuelles. Nous avons ensuite parlé de l'analyse d'impacts de changement avant de donner quelques approches dans ce sens. La deuxième partie de ce chapitre a été consacrée aux mesures d'inconsistances destinées à estimer l'impact d'une opération de modification d'une ontologie. Ces mesures sont caractérisées par des propriétés que nous avons données, avant d'aborder les approches de mesures, puis terminer par des méthodes de résolution d'inconsistances.

De la revue littéraire que nous avons effectuée, nous avons remarqué que la gestion de l'évolution d'ontologies en termes d'analyse d'impacts est prise en charge. L'analyse d'impacts concerne toujours une base de connaissances quelconque ou une ontologie de taille moyenne. Nous avons remarqué une absence d'études qui traitent la gestion de l'évolution d'une ontologie de grande taille de la dimension de Gene Ontology par exemple. Alors que les ontologies volumineuses sont très présentes dans les systèmes de partage de données, une méthodologie de mesure et de gestion des impacts de changements intervenant à ces types d'ontologies devient une nécessité. Comment faut-il procéder alors pour gérer l'évolution des ontologies de taille importante ? Faut-il partitionner l'ontologie en communautés de taille raisonnable et gérer l'évolution au niveau de chaque communauté ? S'il faut partitionner l'ontologie, quels critères de décomposition utiliser ? Comment ensuite mesurer l'inconsistance produite dans une partition de l'ontologie et suivre sa propagation dans toute l'ontologie ? Ce sont des questions qui ne trouvent pas encore de réponses pertinentes, questions auxquelles nos différentes contributions dans cette thèse apportent des éclaircis.

Chapitre 4

Mesures d'inconsistances résultant de changements ontologiques

Sommaire

4.1 Introduction

Une mesure d'inconsistances est comme nous l'avons définie dans le chapitre 3, une estimation de l'implication des axiomes dans l'inconsistance produite par un changement dans une base de connaissances. Pour une ontologie, elle permet de quantifier l'impact causé par une opération de modification d'une entité sur les autres entités ou les objets dépendants de l'ontologie. Elle fournit à l'ingénieur de l'ontologie des informations permettant de décider si l'inconsistance doit être traitée ou ignorée. Une mesure d'inconsistances ou de degré d'impacts doit, quelque soit

la formule utilisée pour la définir, vérifier au moins la consistance, la monotonicité, l'indépendance et être applicable à la base de connaissances pour laquelle on la définit, et ce quelque soit sa taille.

Dans ce chapitre, nous proposons deux approches de mesure du degré d'impacts d'un changement ontologique sur les autres entités de l'ontologie. La première mesure est applicable à une ontologie légère de taille moyenne. Elle repose sur le poids de dépendance entre les nœuds du graphe ontologique et le sens du flux de propagation d'impacts. La mesure d'inconsistances utilise dans sa définition les assertions portant sur une opération de modification notamment les pré-condition, post-condition et invariant et le poids de connexion entre les entités. La seconde approche de mesures permet d'estimer l'inconsistance produite par un changement dans une ontologie volumineuse. Pour cette approche, nous avons d'abord proposé une méthode de partitionnement d'une ontologie de grande taille. La décomposition se base sur un algorithme appelé "Island Line" prenant en entrée un graphe ontologique pondéré. Nous attribuons ensuite à chaque nœud du graphe un poids selon le sens et le type de relations auxquelles il est noué. Ce poids est utilisé pour définir un degré de dépendance entre les différentes entités de l'ontologie. La mesure utilise, dans sa définition, les assertions de l'opération de modification et le poids de dépendance entre les entités présentes dans la négation de l'invariant de l'opération de modification.

La suite de ce chapitre est organisée en deux grandes parties. Nous présentons dans la première partie les éléments de base d'une ontologie. Nous commençons, dans cette section, par donner les formalismes d'une ontologie notamment le modèle formel et sa représentation graphique. Nous déterminons ensuite les dépendances liant les éléments d'une ontologie, puis formalisons les opérations de changements. La seconde partie est consacrée à la définition des mesures et à leur évaluation. Pour chaque mesure, nous donnons les préliminaires sur lesquels elle se fonde avant de la définir et l'illustrer par des exemples. Nous terminons par une synthèse et conclusion résumant ce chapitre et annonçant les points à traiter dans la suite du document.

4.2 Formalisme d'une ontologie

Une ontologie définit un ensemble de primitives représentationnelles avec lesquelles on modélise un domaine de connaissances ou de discours. Les primitives représentationnelles sont typiquement des classes (ou ensembles), attributs (ou propriétés) et des relations (ou des relations sur les classes membres). La représentation de ces primitives renvoie à l'utilisation d'un langage de représentation qui spécifie clairement les concepts, les relations, les instances et les axiomes, d'où l'importance de la définition d'un modèle formel de l'ontologie.

4.2.1 Modèle d'une ontologie légère

Une ontologie légère est une ontologie qui n'inclut pas dans sa définition l'utilisation d'axiomes, par opposition à une ontologie lourde. Il s'agit d'un ensemble de

concepts et d'attributs liés par des relations de subsomption, d'objet et de type de données. Nous représentons donc une ontologie légère par une structure O, dérivée du modèle établi dans [Sal10] et définie comme suit :

$$O = \{C, R, A, T, CAR_R, H^C, H^R, \sigma_R, \sigma_{CARR}, \sigma_A\}$$

avec :

- C qui représente l'ensemble des concepts de l'ontologie ;
- A, l'ensemble des attributs ;
- T, l'ensemble des types des attributs ;
- $R \subseteq (C \times C)$ qui définit l'ensemble des relations associatives entre concepts ;
- $H^C \subseteq (C \times C)$ définissant la relation de subsomption entre deux concepts ;
- $H^R \subseteq (R \times R)$ qui définit la subsomption entre deux relations associatives ;
- CAR_R représentant la caractéristique de la relation associative R qui peut prendre comme valeur symétrique, transitive, ou réflexive ;
- $\sigma_R : R \rightarrow C \times C$ qui définit la signature de la relation associative R ;
- $\sigma_{CARR} : R \rightarrow CAR_R$ la signature de la relation spécifiant la caractéristique de la relation R ;
- $\sigma_A : A \rightarrow C \times T$ la signature de la relation d'attribut entre un concept et un attribut.

Exemple 4.1

Reprenons l'ontologie de la figure 2.2 et représentons-la avec le formalisme défini ci-dessus. Nous obtenons alors la structure suivante :

$O_1 = \{C, R, A, T, CAR_R, H^C, H^R, \sigma_R, \sigma_{CARR}, \sigma_A\}$

avec :

$C = \{\text{Produit, Culture, Viande, Arachide, Chambre\_froide}\}$;

$R = \{\text{se\_conserve}\}$;

$A = \{\text{annee\_vente}\}$;

$T = \{\#\text{string}\}$;

$CAR_R = \{\text{symétrique, transitive, réflexive}\}$;

$H^C(\text{Culture, Produit}), H^C(\text{Arachide, Culture}) H^C(\text{Viande, Produit})$;

$\sigma_R(\text{Viande, se\_conserve, Chambre\_froide})$;

$\sigma_A(\text{Arachide, annee\_vente, \#string})$;

Le modèle formel d'une ontologie n'est pas toujours facile à exploiter surtout lorsque l'ontologie est constituée d'un très grand nombre d'entités. Ainsi, on recourt parfois à la représentation graphique, plus adéquate à certaines situations d'autant plus qu'elle permet une interprétation simple de l'ontologie.

4.2.2 Représentation graphique d'une ontologie

La représentation d'une ontologie sous forme de graphe dérive des travaux de Quillian [Qui68], le premier scientifique à introduire les réseaux sémantiques pour représenter la mémoire humaine. Un réseau sémantique dénote un modèle qui montre comment l'information peut être représentée en mémoire et comment on pourrait y accéder. Il est composé de nœuds modélisant différents types d'information et de pointeurs reliant les nœuds dont l'étiquette associée indique le type de relation entre les nœuds liés. Une ontologie est un réseau sémantique modélisée par un graphe orienté étiqueté que nous appelons graphe ontologique.

Définition 13 (Graphe ontologique) *Un graphe ontologique est un couple* $G = (E, \Gamma)$ *où* E *est un ensemble d'entités qui peuvent être des concepts ou des types et* Γ *une application de* E *vers l'ensemble* $P(E)$ *des parties de* E *telle que :*

$$\Gamma \in \{H^C, \sigma_R, \sigma_A\}$$

Un couple (e_i, e_j) *tel que* $e_j \in \Gamma(e_i)$ *dénote un arc étiqueté entre* e_i *et* e_j *où* e_i *est le nœud source et* e_j *le nœud terminal.*

Exemple 4.2 Considérons l'ontologie O_1 dont le modèle formel est illustré par l'exemple 4.1. Le graphe ontologique de O_1 est donné par la figure 4.1.

La représentation graphique d'une ontologie permet aux utilisateurs de comprendre, créer ou modifier directement des objets, de façon beaucoup plus simple. L'insuffisance était due par le fait que le graphe n'avait pas de sémantique formelle, les arcs peuvant représenter différents genres de relations entre les sommets, une équivoque que notre définition a levée.

4.3 Types de dépendances dans une ontologie

Une dépendance représente une relation, un lien, une association entre deux objets ou deux groupes d'objets. Les éléments d'une ontologie constituent des entités liées par des relations de dépendance de différents types.

Définition 14 (Dépendance entre concepts) *Deux concepts* C_i *et* C_j *sont en dépendance directe si* C_i *et* C_j *sont reliés par une relation de subsomption ou une relation associative. Autrement dit :*

dependance(C_i, C_j) *si* $\exists\ H^C \mid H^C(C_i, C_j)$ *ou si* $\exists\ \sigma_R \mid \sigma_R(C_i, R_k, C_j)$, R_k *étant une relation associative.*

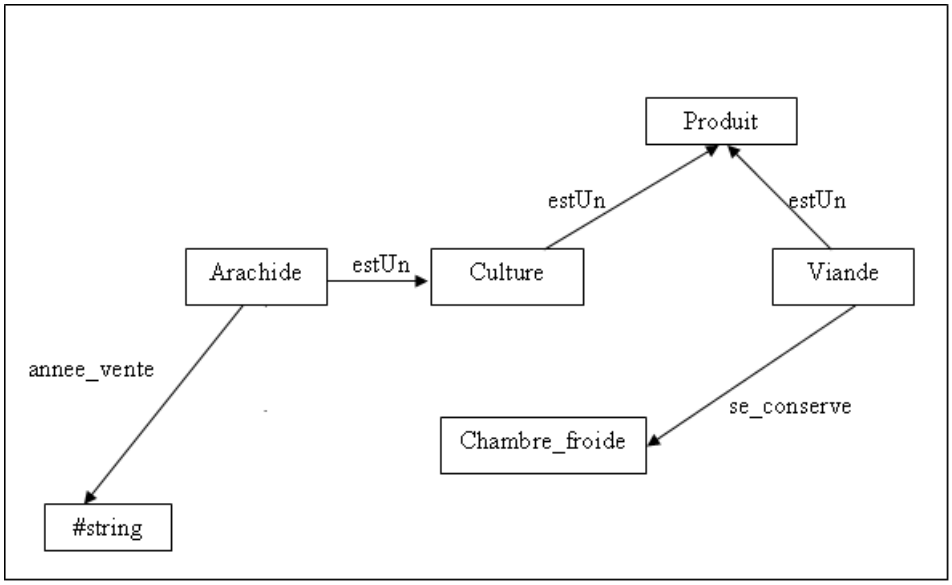

FIGURE 4.1 – Graphe ontologique

Définition 15 (Dépendance entre concept et type) *Un concept C_i et un type T_j sont en dépendance directe si C_i et T_j sont reliés par une relation d'attribut, c'est-à-dire :*

$$dependance(C_i,\ T_j)\ si\ \exists\ \sigma_A\ /\ \sigma_A(C_i,\ A_k,\ T_j),\ A_k\ étant\ une\ relation\ d'attribut.$$

Définition 16 (Dépendance entre relations) *Deux relations R_i et R_j sont en dépendance directe si R_i et R_j sont reliés par une relation de subsomption, autrement dit :*

$$dependance(R_i,\ R_j)\ si\ \exists\ H^R\ /\ H^R(R_i,\ R_j)$$

L'établissement de ces différents types de dépendances entre les entités d'une ontologie est nécessaire pour déterminer les flux d'impact utilisés dans la définition des mesures d'inconsistances des opérations de changement. Nous formalisons dans la section qui suit les opérations de changement sur lesquelles portent les estimations.

4.4 Formalisme des opérations de changements ontologiques

Le formalisme utilise une série d'assertions aidant à la compréhension des opérations de modification. Le tableau qui suit donne une liste d'assertions issues de travaux dans [STL11] et [STLB11].

TABLE 4.1 – Assertions de base

Assertion	Signification
$+\mathrm{C}_i$	$\exists(\mathrm{C}_i \in \mathrm{C})$
$+\mathrm{R}_i$	$\exists(\mathrm{R}_i \in \mathrm{R})$
$+\mathrm{A}_i$	$\exists(\mathrm{A}_i \in \mathrm{A})$
$+\mathrm{T}_i$	$\exists(\mathrm{T}_i \in \mathrm{T})$
$+\mathrm{CAR}_{Ri}$	$\exists(\mathrm{CAR}_{Ri} \in \mathrm{CAR}_R)$
$+\mathrm{H}^C(\mathrm{C}_i, \mathrm{C}_j)$	$\exists\ (\mathrm{C}_i, \mathrm{C}_j \in \mathrm{C}) : \mathrm{H}^C(\mathrm{C}_i, \mathrm{C}_j)$
$+\sigma_R(\mathrm{C}_i, \mathrm{R}_k, \mathrm{C}_j)$	$\exists(\mathrm{C}_i, \mathrm{C}_j \in \mathrm{C} \wedge \mathrm{R}_k \in \mathrm{R}) : \sigma_R(\mathrm{C}_i, \mathrm{R}_k, \mathrm{C}_j)$
$+\sigma_{CARR}(\mathrm{C}_i, \mathrm{CAR}_{Ri})$	$\exists(\mathrm{C}_i \in \mathrm{C} \wedge \mathrm{CAR}_{Ri} \in \mathrm{CAR}_R) : \sigma_{CARR}(\mathrm{C}_i, \mathrm{CAR}_{Ri})$
$+\sigma_A(\mathrm{C}_i, \mathrm{A}_j, \mathrm{T}_k)$	$\exists(\mathrm{C}_i \in \mathrm{C} \wedge \mathrm{A}_j \in \mathrm{A} \wedge \mathrm{T}_k \in \mathrm{T}) : \sigma_A(\mathrm{C}_i, \mathrm{A}_j, \mathrm{T}_k)$
$-\mathrm{C}_i$	$\neg(\mathrm{C}_i \in \mathrm{C})$
$-\mathrm{R}_i$	$\neg(\mathrm{R}_i \in \mathrm{R})$
$-\mathrm{A}_i$	$\neg(\mathrm{A}_i \in \mathrm{A})$
$-\mathrm{T}_i$	$\neg(\mathrm{T}_i \in \mathrm{T})$
$-\mathrm{CAR}_{Ri}$	$\neg(\mathrm{CAR}_{Ri} \in \mathrm{CARR})$
$-\mathrm{H}^C(\mathrm{C}_i, \mathrm{C}_j)$	$\forall(\mathrm{C}_i \in \mathrm{C}) \wedge \forall(\mathrm{C}_j \in \mathrm{C}) : \neg\mathrm{H}^C(\mathrm{C}_i, \mathrm{C}_j)$
$-\sigma_R(\mathrm{C}_i, \mathrm{R}_k, \mathrm{C}_j)$	$\forall(\mathrm{C}_i \in \mathrm{C}) \wedge \forall(\mathrm{C}_j \in \mathrm{C}) \wedge \forall(\mathrm{R}_k \in \mathrm{R}) : \neg\sigma_R(\mathrm{C}_i, \mathrm{R}_k, \mathrm{C}_j)$
$-\sigma_{CARR}(\mathrm{C}_i, \mathrm{CAR}_{Ri})$	$\forall(\mathrm{C}_i \in \mathrm{C}) \wedge \forall(\mathrm{CAR}_{Ri} \in \mathrm{CARR}) : \neg\sigma_{CARR}(\mathrm{C}_i, \mathrm{CAR}_{Ri})$
$-\sigma_A(\mathrm{C}_i, \mathrm{A}_j, \mathrm{T}_k)$	$\forall(\mathrm{C}_i \in \mathrm{C}) \wedge \forall(\mathrm{A}_j \in \mathrm{A}) \wedge \forall(\mathrm{T}_k \in \mathrm{T}) : \neg\sigma_A(\mathrm{C}_i, \mathrm{A}_j, \mathrm{T}_k)$
$-\mathrm{H}^C(*, \mathrm{C}_i)$	$\forall(\mathrm{C}_k \in \mathrm{C}) : \neg\mathrm{H}^C(\mathrm{C}_k, \mathrm{C}_i)$
$-\mathrm{H}^C(\mathrm{C}_i, *)$	$\forall(\mathrm{C}_k \in \mathrm{C}) : \neg\mathrm{H}^C(\mathrm{C}_i, \mathrm{C}_k)$
$-\sigma_{CARR}(\mathrm{R}_i, *)$	$\forall(\mathrm{CAR}_{Ri} \in \mathrm{CARR}) : \neg\sigma_{CARR}(\mathrm{R}_i, \mathrm{CAR}_{Ri})$
$-\sigma_A(*, \mathrm{A}_i, \mathrm{T}_j)$	$\forall(\mathrm{C}_k \in \mathrm{C}) : \neg\sigma_A(\mathrm{C}_k, \mathrm{A}_i, \mathrm{T}_j)$
$-\sigma_A(*, \mathrm{A}_i, *)$	$\forall(\mathrm{C}_k \in \mathrm{C}) \wedge \forall(\mathrm{T}_j \in \mathrm{T}) : \neg\sigma_A(\mathrm{C}_k, \mathrm{A}_i, \mathrm{T}_j)$
$-\sigma_A(\mathrm{C}_k, *, *)$	$\forall(\mathrm{A}_i \in \mathrm{A}) \wedge \forall(\mathrm{T}_j \in \mathrm{T}) : \neg\sigma_A(\mathrm{C}_k, \mathrm{A}_i, \mathrm{T}_j)$
$-\sigma_A(*, *, \mathrm{T}_j)$	$\forall(\mathrm{C}_k \in \mathrm{C}) \wedge \forall(\mathrm{A}_j \in \mathrm{A}) : \neg\sigma_A(\mathrm{C}_k, \mathrm{A}_i, \mathrm{T}_j)$
$-\sigma_R(\mathrm{C}_i, *, *)$	$\forall(\mathrm{C}_k \in \mathrm{C}) \wedge \forall(\mathrm{R}_j \in \mathrm{R}) : \neg\sigma_R(\mathrm{C}_i, \mathrm{R}_j, \mathrm{C}_k)$
$-\sigma_R(*, \mathrm{R}_k, *)$	$\forall(\mathrm{C}_i \in \mathrm{C}) \wedge \forall(\mathrm{C}_j \in \mathrm{C}) : \neg\sigma_R(\mathrm{C}_i, \mathrm{R}_k, \mathrm{C}_j)$

Une opération de modification d'une entité ontologique, appelée aussi opération de changement ontologique, se modélise par un triplet :

$$\Delta = (\text{Operation, Arguments, Assertions})$$

où Operation représente l'opération de modification, Arguments désigne les arguments de l'opération et Assertion = (Pre-conditions, Invariant, Post-conditions) représente les assertions de l'opération que sont les pre-conditions, l'invariant et les post-conditions telles que définies dans le tableau 4.2 pour des changements élémentaires.

– Si les assertions Pre-conditions et Invariant sont vérifiés, alors l'opération peut s'exécuter sans propagation d'impacts ;
– Si l'assertion Pre-conditions n'est pas vérifiée, alors l'opération n'est pas exécutée ;
– Si l'assertion Pre-conditions est vérifiée et que l'assertion Invariant n'est pas vérifiée alors l'opération est exécutée et il y a un processus de propagation d'inconsistances qu'il faudra mesurer.

TABLE 4.2 – Opérations de changement simples

Opération	**Pré-conditions**	**Invariant**	**Post-conditions**
AjouterConcept(C_i)	-C_i	-H^C(*, C_i) -σ_R(*, *, C_i)	+C_i
SupprimerConcept(C_i)	+C_i	-H^C(*, C_i) -σ_R(*, *, C_i)	-C_i
AjouterSubConcept(C_i, C_j)	-H^C(C_i, C_j)	+C_i +C_j	+H^C(C_i, C_j)
SupprimerSubConcept(C_i, C_j)	+H^C(C_i, C_j)	+C_i +C_j	-H^C(C_i, C_j)
CopierConcept(C_i, C_j)	-C_j +C_i	-H^C(*, C_j) -σ_R(*, *, C_j)	+C_i +C_j
AjouterType(T_i)	-T_i	-σ_A(*, *, T_i)	+T_i
SupprimerType(T_i)	+T_i	-σ_A(*, *, T_i)	-T_i
AjouterAttribut(A_i)	-A_i	-σ_A(*, A_i, *)	+A_i
SupprimerAttribut(A_i)	+A_i	-σ_A(*, A_i, *)	-A_i
AjouterRelationAssociative(R_i)	-Ri -σ_R(*, R_i, *)	-σ_{CARR}(R_i, *)	+ R_i +σ_{CARR}(R_i, CAR_{Ri})
SupprimerRelationAssociative(R_i)	+R_i +σ_R(*, R_i, *)	-σ_{CARR}(R_i, *)	-R_i -σ_{CARR}(R_i, CAR_{Ri})
AjouterSubRelationAss(R_i, R_j)	-H^R(R_i, R_j)	+R_i +R_j	+H^R(R_i, R_j)
SupprimerSubRelationAss(R_i, R_j)	+H^R(R_i, R_j)	-R_i +R_j	+H^R(R_i, R_j)

Le formalisme des opérations de modification n'a pas concerné les changements complexes. En effet, Stojanovic montre dans [Sto04] que tout changement complexe peut être transformé en une séquence de changements élémentaires. Le tableau 4.3 montre quelques exemples de transformation de changements complexes en changements simples.

TABLE 4.3 – Exemples de décomposition de changements complexes

Changements complexes	Décomposition en changements simples
GeneraliserConcept(C_i, C_{nouv})	AjouterConcept(C_{nouv}) AjouterRelationSub(C_i, C_{nouv}) AjouterRelationSub(C_{nouv}, Parent(C_i))
SpecialiserConcept(C_i, C_{nouv})	AjouterConcept(C_{nouv}) AjouterRelationSub(C_{nouv}, C_i) AjouterRelationSub(Fils(C_i), C_{nouv})
GeneraliserDomaine(R_l, C_i, C_j, C_k)	AjouterRelationAssociative(R_l ,C_j ,C_k) SupprimerRelationAssociative(R_l ,C_i ,C_k)
SpecialiserDomaine(R_l, C_i, C_j, C_k)	AjouterRelationAssociative(R_l ,C_j ,C_k) SupprimerRelationAssociative(R_l ,C_i ,C_k)
GeneraliserCoDomaine(R_l, C_i, C_j, C_k)	AjouterRelationAssociative(R_l ,C_j ,C_k) SupprimerRelationAssociative(R_l ,C_i ,C_k)
SpecialiserCoDomaine(R_l, C_i, C_j, C_k)	AjouterRelationAssociative(R_l ,C_j ,C_k) SupprimerRelationAssociative(R_l ,C_i ,C_k)
FusionnerAttributs(A_i, A_j, A_{nouv}, C_l)	CreerAttribut(A_{nouv}, T_k) AjouterRelationAttribut(A_{nouv}, C_l, T_k) SupprimerRelationAttribut(A_i, C_l, T_k) SupprimerRelationAttribut(A_j, C_l, T_k)

4.5 Mesure d'inconsistances d'une opération de changement dans une ontologie

4.5.1 Préliminaires

Nous avons défini dans les sections précédentes un graphe ontologique et les types de dépendances entre entités ontologiques. Nous utilisons ces définitions pour déterminer le poids de connexion entre les nœuds du graphe et sur lequel repose la mesure d'inconsistances.

Définition 17 (Poids de connexion) *Soit $G = (E, \Gamma)$ un graphe ontologique, E_i et E_j deux entités de E. Le poids de connexion entre E_i et E_j se définit comme suit :*

$$p(E_i, E_j) = \begin{cases} 1 & si\ dependance(E_i,\ E_j) \\ 0 & si\ dependance(E_j,\ E_i) \end{cases} \tag{4.1}$$

4.5.2 Définition de la mesure

Soit Op une opération de modification d'une entité E_i sur une ontologie O de la forme AjouterConcept(E_i), AjouterType(E_i), AjouterAttribut(E_i),

SupprimerConcept(E_i), SupprimerType(E_i) ou SupprimerAttribut(E_i) et Invariant l'invariant de l'opération Op. Soit K un ensemble contenant toutes les entités E_j qui rendent l'ensemble Invariant inconsistant.

Définition 18 (Mesure d'inconsistances I_e) *La mesure d'inconsistances I_e d'une opération de modification Op d'une entité E_i sur une ontologie légère O est donnée par la formule suivante :*

$$I_e(Op(E_i)) = I_e(K) = \frac{\sum_{E_j \in K} p(E_i, E_j)}{\sum_{E_k, E_l \in O} p(E_k, E_l)} \tag{4.2}$$

Proposition 4.1
I_e définit une mesure d'inconsistances sur K.

Preuve
Nous devons montrer que I_e satisfait les assertions ci-dessous :

1. Consistance : K est consistant si et seulement si $I_e(K) = 0$.
2. Monotonicité : $I_e(K) \leqslant I_e(K \cup \{\alpha\})$.
3. Indépendance : Si $\alpha \in K$ est une formule libre alors $I_e(K) = I_e(K \setminus \{\alpha\})$.

Soit K un ensemble contenant les entités de l'ontologie O qui rendent inconsistant l'invariant d'une opération de modification Op d'une entité E_i de O.

1. $I_e(Op(E_i)) = I_e(K) = 0 \Leftrightarrow \forall\, E_j \in K\ p(E_i, E_j) = 0 \Leftrightarrow$ K est vide ou E_i n'est en dépendance avec aucun élément de K $\Leftrightarrow$ K donc consistant.
2. Soit α tel que $MI(K \cup \{\alpha\}) = MI(K) \cup MI(\{\alpha\})$ et $MI(K) \cap MI(\{\alpha\}) = \varnothing$
   $I_e(K \cup \{\alpha\}) = I_e(K) + I_e(\{\alpha\}) \geqslant I_e(K)$ car $I_e(K) \geqslant 0\ \forall$ K.
3. Soit $\alpha \in K$ est une formule libre.
   α libre $\Rightarrow I_e(\{\alpha\}) = 0$.
   Or $I_e(K) = I_e(K \setminus \{\alpha\}) + I_e(\{\alpha\})$ car $MI((K \setminus \{\alpha\}) \cup \{\alpha\}) = MI(K \setminus \{\alpha\}) \cup MI(\{\alpha\})$ et $MI(K \setminus \{\alpha\}) \cap MI(\{\alpha\}) = \varnothing$.
   Donc $I_e(K) = I_e(K \setminus \{\alpha\}) + I_e(\{\alpha\}) = I_e(K \setminus \{\alpha\})$.

Montrons maintenant que :
$MI(K \cup \{\alpha\}) = MI(K) \cup MI(\{\alpha\})$ et $MI(K) \cap MI(\{\alpha\}) = \varnothing$.

α étant une formule, alors $MI(\{\alpha\}) = \varnothing$, ce qui veut dire que :
$MI(K) \cup MI(\{\alpha\}) = MI(K)$ et $MI(K) \cap MI(\{\alpha\}) = \varnothing$.
Supposons que K contient une formule β en conflit avec α, alors $MI(K \cup \{\alpha\}) = MI(K \setminus \{\beta\}) \cup MI(\{\alpha, \beta\})$ car si MI(K) existe, $MI(K \setminus \{\beta\}) = \varnothing$ du fait que $\beta \in K$.
Donc $MI(K \cup \{\alpha\}) = MI(\{\alpha, \beta\}) \Rightarrow K = \{\beta\}$ et $MI(K) = \varnothing$ ce qui est absurde.
Ainsi donc, K ne contient aucune formule en conflit avec α, d'où
$MI(K \cup \{\alpha\}) = MI(K) \cup MI(\{\alpha\})$ et $MI(K) \cap MI(\{\alpha\}) = \varnothing$.

4.5.3 Illustrations

Exemple 4.4
Appliquons une modification sur l'ontologie de la figure 4.1 consistant à supprimer le concept **Culture**. Cette opération (SupprimerConcept(Culture)) donne les assertions suivantes :

- Pre-conditions = {+Culture}
- Invariant = {-H^C(*, Culture), -σ_R(*, *, Culture)}
- Post-conditions = {-Culture}

Nous remarquons que la pré-condition Pre-conditions est vérifiée mais l'invariant Invariant ne l'est pas puisque nous avons cette relation de subsomption H^C(Arachide, Culture). L'exécution de la suppression du concept Culture produira alors une inconsistance que nous allons mesurer. Soit K l'ensemble des éléments qui rendent l'ensemble Invariant inconstant. Alors nous avons :
K = {Arachide}

$$I_e(\text{Ajouter(Culture)}) = I_e(K) = \frac{p(Arachide, Culture)}{\sum_{E_j, E_k \in O} p(E_j, E_k)}$$

$$I_e(\text{Ajouter(Culture)}) = \frac{1}{7} = 0.14$$

Exemple 4.5
Si nous appliquons une opération d'ajout d'un nouveau type **entier** à l'ontologie, nous obtenons :

- Pre-conditions = {-entier}
- Invariant = {-σ_A(*, *, entier)}
- Post-conditions = {+entier}

Nous remarquons que la pré-condition et l'invariant sont vérifiés. Donc l'opération AjouterType(entier) s'exécute sans induire d'inconsistances. Cela est prouvé par le résultant de notre mesure. En effet, soit K l'ensemble des éléments qui rendent l'ensemble Invariant inconstant. K est alors vide, car il n'existe pas encore d'éléments dans l'ontologie en relation avec le type entier.

$$I_e(\text{Ajouter(entier)}) = I_e(K) = 0$$

Propriétés de la mesure I_e
En plus de la consistance, la monotonicité et l'indépendance, la mesure I_e vérifie la normalisation.

4.6 Mesures d'inconsistances d'une opération de changement dans une ontologie de grande taille

Une ontologie de grande taille est une ontologie pouvant compter jusqu'à des dizaines de millions de concepts, instances et types. Le volume très important de

ce type d'ontologie rend complexe la gestion des impacts de changements. Dans cette section, nous présentons une approche de mesures d'inconsistances résultant d'une opération de modification sur une telle ontologie. L'approche consiste d'abord à partitionner l'ontologie en communautés appelées aussi modules, puis à mesurer l'inconsistance dans la communauté hôte de la modification, et enfin à la mesurer sur l'ontologie globale.

4.6.1 Méthodologie de partitionnement d'ontologies volumineuses

La détection de communautés ou partitionnement est le fait de décomposer un ensemble E en sous-ensembles E_1, E_2, ..., E_n de sorte que les éléments d'un sous-ensemble E_i partagent plus de propriétés à l'intérieur de E_i qu'en dehors. En théorie des graphes [BM76], le partitionnement a pour but d'indentifier les communautés et leur organisation hiérarchique en utilisant les informations issues de la topologie du graphe. L'identification des communautés et de leurs limites permet de classifier les sommets du graphe, selon leur position structurelle dans les modules [For10]. Le partitionnement pose cependant des problèmes d'ordre conceptuel et technique. Du point de vue conceptuel, la définition ce qui doit être considéré comme communauté pose problème. Quand est-ce qu'un sous-graphe doit être considéré comme un groupe significatif, c'est-dire-dire une communauté ? La réponse de cette question n'est pas toujours évidente. Le problème technique demeure la complexité des algorithmes de partitionnement qui suivent une fonction exponentielle pour la plupart des cas.

Le partitionnement trouve beaucoup d'applications dans le domaine de l'économie, de la biologie, des réseaux sociaux, des réseaux informatiques ou de télécommunication, du web sémantique, etc.

4.6.1.1 Partitionnement d'ontologies

L'apparition d'ontologies de très grande taille est notée dans le domaine de la médecine et de l'agronomie avec des ontologies comme AGROVOC, NALT [1], NCI [2], YAGO [3], Gene Ontology [4] . L'ontologie YAGO (Yet Another Great Ontology) constitue une énorme base de connaissances, développée à l'Institut d'informatique Max Planck de Sarrebruck, décrivant plus de 2 millions d'entités de personnes, d'organisations, de villes et contenant plus de 20 millions de faits sur ces entités. Ses données proviennent de Wikipedia [5] et sont structurées à l'aide de WordNet [6]. Gene Ontology a pour objectif [ABB+00] de fournir un vocabulaire contrôlé qui peut être appliqué à tous les organismes, comme une base de connaissances de gènes et de rôles de protéines dans les cellules. L'objectif au départ était de permettre

1. http ://agclass.nal.usda.gov/agt.shtml
2. https ://bioportal.bioontology.org/ontologies/NCIT
3. http ://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/downloads/
4. geneontology.org
5. https ://fr.wikipedia.org/
6. https ://wordnet.princeton.edu/

aux chercheurs d'interroger les bases de données connexes et d'obtenir des protéines apparentées et des produits génétiques. L'ontologie compte plusieurs dizaines de milliers de concepts.

Toutefois, la maintenance de ces ontologies volumineuses constitue une tâche difficile. Si une mise à jour s'opère sur une entité de l'ontologie, le suivi de ses impacts sur toute l'ontologie nécessite des techniques assez complexes. Une des techniques fait appel à la décomposition de l'ontologie en sous ontologies grâce aux méthodologies de partitionnement.

Le partitionnement d'ontologies peut être vu comme le problème de regrouper les entités d'une ontologie (concepts, instances et types) en clusters ou communautés de sorte que les entités dans une communauté partagent plus de propriétés à l'intérieur de la communauté qu'en dehors. Il était surtout utilisé pour l'alignement d'ontologies [HSZR09], mais il est aussi pratiqué dans le domaine des réseaux sociaux, les réseaux biologiques, les réseaux d'information, les réseaux linguistiques, etc.

Grahne et Räihä, dans le cadre de leur travail sur la décomposition de schéma de base de données [GR93], attestent qu'un bon partitionnement doit respecter les principes que sont la :

- **redondance minimale** : le partitionnement doit représenter les objets significatifs ;
- **représentation** : il doit être possible de récupérer la relation universelle des relations composantes ;
- **séparation** : lorsqu'une relation est mise à jour, il doit être possible de s'assurer que les dépendances dans la relation universelle tiennent toujours après la mise à jour, sans construire réellement la relation universelle.

Dans le cadre d'un partitionnement d'ontologies, ces principes trouvent aussi leur pertinence. Une bonne décomposition d'une ontologie doit représenter toutes les entités significatives de l'ontologie, recouvrir l'ontologie (l'union de toutes les partitions doit donner l'ontologie initiale) et conserver les dépendances initiales.

Dans la littérature, nous avons des approches de décomposition dont la plupart sont destinées à l'alignement d'ontologies. La méthodologie de partitionnement de [HSZR09] est guidée par l'existence de deux ontologies à aligner. L'outil TaxoPart ainsi développé utilise un algorithme de classification hiérarchique [HZQ06] qui regroupe itérativement dans un même bloc, les concepts proches suivant une mesure de similarité ne s'appuyant que sur la position relative des concepts dans l'ontologie. L'outil décompose la première ontologie la plus structurée et force la deuxième à suivre cette décomposition. Dans notre contexte, cette méthodologie ne convient pas car nous disposons d'une seule ontologie que nous cherchons à diviser. Les travaux dans [LPLTS07] présentent deux approches de décomposition basées sur les algorithmes de partitionnement de graphe. La première approche consiste à déterminer une ligne séparatrice minimale du graphe en deux sous-ensembles disjoints. La seconde approche s'inspire de la méthodologie de segmentation d'image. [SK04] propose une méthode de partitionnement basé sur la hiérarchie des concepts. Elle consiste à créer un graphe de dépendance pondéré à partir de la structure de l'ontologie, puis à déterminer les modules en utilisant l'algorithme "Island Line" [Bat93].

Notre méthode de partitionnement suit ce même procédé mais diffère par la définition des pondérations et des types de dépendances entre les différentes entités de l'ontologie.

4.6.1.2 Méthodes de partitionnement d'ontologies

Il existe différentes méthodes de partition d'ontologies allant des méthodes de partitionnement de graphes aux méthodes de segmentation d'image.

- **Méthode de bissection spectrale** [Fie73] : elle consiste à calculer le vecteur propre correspondant à la plus petite valeur propre non nulle de la matrice Laplacienne du graphe. Le graphe est séparé en deux parties en fonction du signe de leurs composants selon ce vecteur propre.
- **Méthode de Kernighan et Lin** [KL70] : c'est un algorithme de bissection visant à trouver la coupe du graphe minimisant le nombre d'arêtes tombant entre les deux groupes.
- **Méthodes de clustering hiérarchique** [NC04] : au début, chaque sommet représente une petite communauté. On itère en calculant les distances entre communautés et en fusionnant les deux communautés les plus proches en une nouvelle communauté.
- **G-décomposition** [LP08] : cette décomposition est représentée par un graphe d'intersection dont chaque sommet constitue un sous-graphe et les arêtes représentent les connexions de chaque paire de sommets. La décomposition ne considère que les axiomes de la TBox.
- **Lignes de séparateur minimal** [LPLTS07] : cette méthode cherche une paire de connexion du graphe en calculant le séparateur minimal, puis décompose le graphe suivant ce séparateur.
- **Méthode de partitionnement pour l'alignement d'ontologies** [HSZR09] : cette approche est guidée par l'existence de deux ontologies à aligner.
- **Méthode s'inspirant de la segmentation d'image** [JF07] : elle utilise les coupes normalisées qui mesurent aussi bien la similarité au sein d'un groupe que la dissemblance entre les groupes.

4.6.2 Algorithme de partitionnement d'ontologie

La méthodologie de partitionnement que nous proposons dans cette thèse s'inspire de l'approche proposée dans [SK04]. Elle consiste à créer un graphe de dépendance pondéré à partir de la structure de l'ontologie, puis à déterminer les communautés en utilisant l'algorithme "Island Line".

4.6.2.1 Préliminaires

Une ontologie est un ensemble d'entités constituées de concepts, d'instances et types reliées par des relations de subsomption, des relations associatives et des relations d'attributs. Pour chaque type de relation entre deux entités E_i et E_j de

l'ontologie, nous attribuons une valeur P_{ij} en fonction du sens de propagation des flux d'impacts.

- Si $H^C(E_i, E_j)$ alors $P_{ji} = 1$ et $P_{ij} = 0$;
- Si $\sigma_R(E_i, R_k, E_j)$ alors $P_{ji} = 1$ et $P_{ij} = 0$;
- Si $\sigma_A(E_i, A_k, T_j)$ alors $P_{ji} = 1$ et $P_{ij} = 0$.

Cette valeur est utilisée pour définir le poids de dépendance entre deux entités du graphe ontologie.

Définition 19 (Poids de dépendance) *Soit $G = (E, \Gamma)$ un graphe ontologique, E_i et E_j deux entités de E. Nous définissons le poids de dépendance entre les entités E_i et E_j comme suit :*

$$w(E_i, E_j) = \frac{P_{ij} + P_{ji}}{\sum_{k=1}^{N}(P_{ik} + P_{ki})} \tag{4.3}$$

avec N le nombre d'entités dans G auxquelles E_i est lié.

Définition 20 (Graphe ontologique de dépendance) *Un graphe ontologique de dépendance est un triplet $G_d = (E, \Gamma, W)$ où E est un ensemble d'entités qui peuvent être des concepts ou des types, Γ une application de E vers l'ensemble P(E) des parties de E et W une fonction de dépendance entre deux entités de E.*

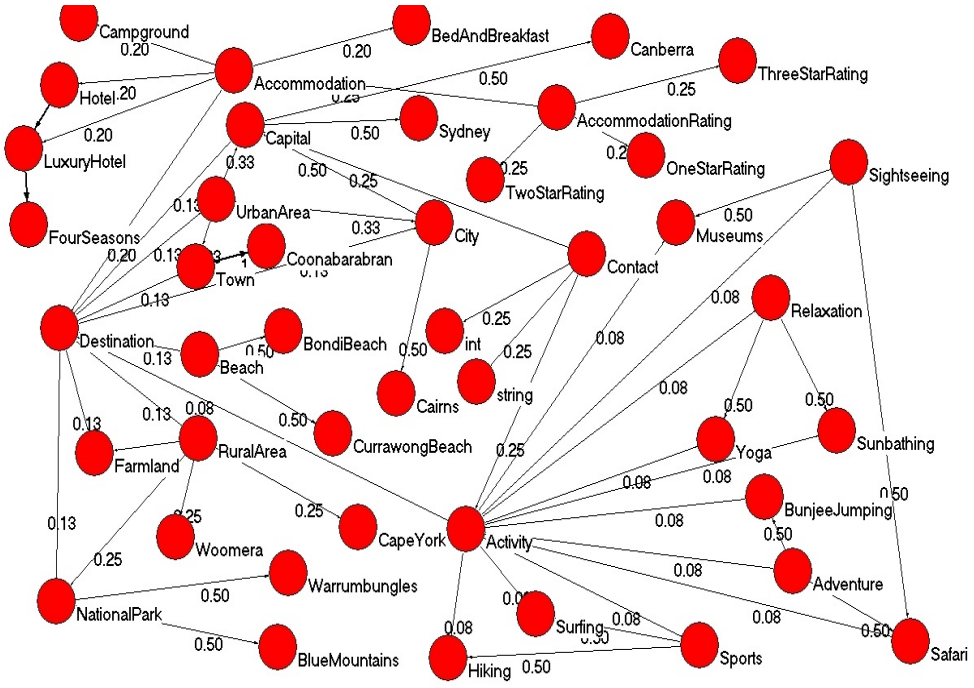

FIGURE 4.2 – Graphe de dépendance de l'ontologie donnée en annexe B.1

Il n'est pas facile de trouver une définition quantitative d'une communauté dans la littérature. Toutes les définitions se basent sur les propriétés ou les comportements

des membres du groupe considéré. Nous donnons une définition basée sur la structure de l'ontologie.

Définition 21 (Communauté ontologique) *Soit un G = (E, Γ, W) graphe ontologique de dépendance. Une communauté C⊆G est un ensemble de concepts, d'instances ou et de types telles que deux entités de C partagent plus de propriétés à l'intérieur de C qu'en dehors.*

L'approche de détermination des communautés d'une ontologie utilise l'algorithme de la Ligne de séparation ("Island Line") pour partitionner l'ontologie. L'algorithme détermine les lignes de séparateur minimum comme critère de décomposition et le nombre de lignes de séparation est variable et dépend de la taille de l'ontologie.

Définition 22 (Ligne de séparation) *Un ensemble de nœuds V d'un graphe est une ligne de séparation s'il est :*

- *un singleton, ou ;*
- *un sous-graphe correspondant à un graphe connecté tel que :*

$$\max_{E_i \in V \wedge E_j \notin V} w(E_i, E_j) \leq \min_{E_k, E_l \in V} w(E_k, E_l) \tag{4.4}$$

Une arête V⊆G est une arête de séparation régulière si :

$$\max_{E_i \in V \wedge E_j \notin V} w(E_i, E_j) < \min_{E_k, E_l \in V} w(E_k, E_l) \tag{4.5}$$

4.6.2.2 Principe de l'algorithme

L'algorithme de partitionnement de l'ontologie commence par une division du graphe ontologique en sous-ensembles appelés îles. Chaque île est déterminée par son port désignant le point le plus proche de l'île. Cette étape de l'algorithme crée donc un ensemble d'îles regroupées sous un nouveau nom de sous île. La deuxième étape consiste à choisir parmi les îles créées celles dont le nombre d'éléments (nombres de sommets) est compris entre le minimum et le maximum (fixés au départ) comme candidats au partitionnement. Les îles dont le nombre de sommets est inférieur au minimum fixé sont supprimées, celles dont le nombre est supérieur au maximum fixé sont subdivisées en îlots qui seront intégrés à l'ensemble des îles. La troisième et dernière étape détermine les partitions en prenant comme entrées tous les candidats. Chaque candidat est d'abord étendu avec les sommets isolés les plus proches sur la base de leur poids avant d'être décomposé.

Nous considérons qu'une entité appartient à une et une seule communauté. Si une entité doit suivre deux communautés par l'affinité des propriétés partagées, on la rattache à celle dont elle est la plus proche.

Algorithme 1 Partitionnement d'ontologie
Entrées : G = (E, Γ, W), un graphe ontologique de dépendance et maximumCtes, le nombre maximum de communautés à obtenir.
Sorties : nombreCtes, le nombre de communautés obtenues.

```
1 min = 1
2 max = |E| - 1
3 iles = {{v} : v ∈ E}
4 sousIles = ∅
5 Ordonner E en ordre décroissant suivant le poids de dépendance w
6 Pour (u, v) ∈ G faire
7       i1 = ile ∈ iles : u ∈ ile
8       i2 = ile ∈ iles : v ∈ ile
9       Si (i1 ≠ i2) alors
10              ile = creerIle()
11              ile.port = u
12              ile.sousIle1 = i1
13              ile.sousIle2 = i2
14              sousIles = sousIles ∪ {ile} \ {i1, i2}
15      FinSi
16 FinPour
17 candidats = ∅
18 Tantque (sousIles ≠ ∅) faire
19      Sélectionner ile ∈ sousIles
20      sousIles = sousIles \ {ile}
21      Si (|ile| < min) alors
22              Supprimer ile
23      Sinon Si (|ile| > max) alors
24                      sousIles = sousIles ∪ {ile.sousIle1, ile.sousIle2}
25                      Supprimer ile
26              Sinon
27                      candidats = candidats ∪ {ile}
28              FinSi
29      FinSi
30 FinTantque
31 Pour communaute ∈ candidats faire
32      etendre(communaute, |E|)
33      partitionnner(maximumCtes, communaute, nombreCtes)
34 FinPour
```

Algorithme 2 Etendre communauté
Entrées : une communauté C, l'ensemble des entités E et l'ensemble des candidats.
Sorites : la communauté C éventuellement enrichi

```
1 Pour v ∈ E : v est isolé faire
2       Si (w(v, C.port) < min_{C_i ∈ candidats} w(v, C_i.port) et |C| < |E|) alors
3             C = C ∪ {v}
4       FinSi
5 FinPour
```

Algorithme 3 Partitionner communauté
Entrées : une communauté C, le nombre maximum de communautés maximumCtes et l'ensemble des candidats
Sorites : nombreCtes, le nombre de communautés obtenues et l'ensemble des candidats

```
1 Si(nombreCtes < maximumCtes) alors
2       C_1 = {u} : u∈C
3       C_2 = {v} : v∈C
4       Pour E_i ∈ C faire
5             Si (w(E_i, C_1.port) < w(E_i, C_2.port)) alors
6                   C_1 = C_1 ∪ {E_i}
7             Sinon
8                   C_2 = C_2 ∪ {E_i}
9             FinSi
10       FinPour
11       Supprimer C
12       nombreCtes = nombreCtes + 1
13       candidats = candidats ∪ {C_1, C_2}
14 FinSi
```

4.6.3 Complexité de l'algorithme

L'algorithme de partitionnement de l'ontologie proposé fait un parcours en profondeur d'un graphe ontologique et prend tous les couples de sommets possibles pour déterminer les candidats. Donc avec un graphe de n sommets, nous aurons C_n^2 combinaisons, ce qui donne :

$$C_n^2 = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{4} = \frac{n^2-n}{4}$$

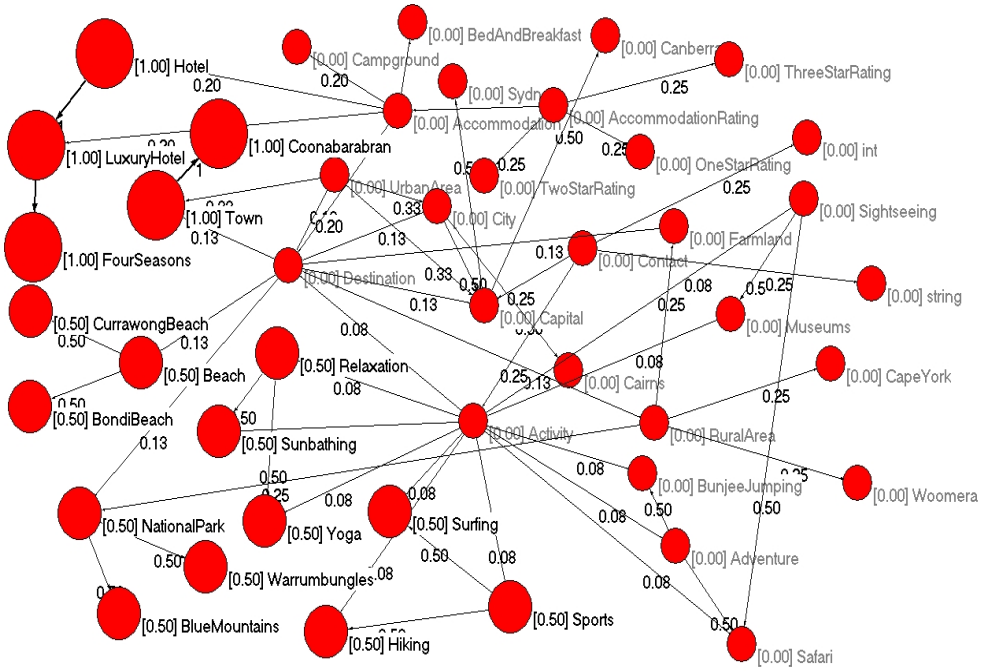

FIGURE 4.3 – Une partition en trois communautés de l'ontologie donnée en B.1

Ainsi, la complexité de l'algorithme s'exprime en $\Theta(n^2)$. Elle est quadratique alors qu'on montre dans [SK07] que les algorithmes de raisonnement sur les ontologies modulaires sont de complexité exponentielle.

4.6.4 Mesure d'inconsistances dans une communauté de l'ontologie

La mesure d'inconsistances dans une communauté trouve sa pertinence dans le cas d'une ontologie volumineuse. Suivre l'impact d'une modification dans une telle ontologie est une tâche difficile et couteuse alors qu'elle peut devenir moins complexe dans le voisinage de l'entité modifiée que nous appelons communauté.

Définition 23 *Soit Op une opération de modification d'une entité E_i sur une ontologie O de la forme AjouterConcept(E_i), AjouterType(E_i), SupprimerConcept(E_i) ou SupprimerType(E_i) et Invariant l'invariant de l'opération Op. Soit K un ensemble contenant toutes les entités E_j qui rendent l'ensemble Invariant inconsistant. La mesure d'inconsistances de l'opération Op dans une communauté C de l'ontologie O est définie comme suit :*

$$I_C^d(Op(E_i)) = I_C^d(K) = \frac{\sum_{E_j \in K} p(E_i, E_j)}{\sum_{E_k, E_l \in C} p(E_k, E_l)} \tag{4.6}$$

avec p(u, v) le poids de connexion entre deux nœuds u et v.

Proposition 4.2
I_C^d définit une mesure d'inconsistances sur K.
Preuve
Cette proposition se démontre de la même manière que celle de la proposition 4.1 de la section 4.5. En effet :

1. $I_C^d(Op(E_i)) = I_C^d(K) = 0 \Leftrightarrow \forall\ E_j \in K\ p(E_i, E_j) = 0 \Leftrightarrow$ K est vide ou E_i n'est en dépendance avec aucun élément de K $\Leftrightarrow$ K donc consistant.
2. Soit α tel que $MI(K \cup \{\alpha\}) = MI(K) \cup MI(\{\alpha\})$ et $MI(K) \cap MI(\{\alpha\}) = \varnothing$
   $I_C^d(K \cup \{\alpha\}) = I_C^d(K) + I_C^d(\{\alpha\}) \geqslant I_C^d(K)$ car $I_C^d(K) \geqslant 0\ \forall$ K.
3. Soit $\alpha \in$ K est une formule libre.
   α libre $\Rightarrow I_C^d(\{\alpha\}) = 0$.
   Or $I_C^d(K) = I_C^d(K \backslash \{\alpha\}) + I_C^d(\{\alpha\})$ car $MI((K \backslash \{\alpha\}) \cup \{\alpha\}) = MI(K \backslash \{\alpha\}) \cup MI(\{\alpha\})$ et $MI(K \backslash \{\alpha\}) \cap MI(\{\alpha\}) = \varnothing$.
   Donc $I_C^d(K) = I_C^d(K \backslash \{\alpha\}) + I_C^d(\{\alpha\}) = I_C^d(K \backslash \{\alpha\})$.

Example 4.6
Supposons une opération de suppression du concept **Activity** de l'ontologie donnée en B.1. Alors nous avons les assertions suivantes :

- Pre-condition = {+Activity}
- Post-condition = {-Activity}
- Invariant ={$\neg H^C$(Sightseeing, Activity), $\neg H^C$(Adventure, Activity), $\neg H^C$(Safari, Activity), $\neg H^C$(BunjeeJumping, Activity), $\neg H^C$(Museums, Activity), $\neg \sigma_R$(Destination, hasActivity, Activity)}

L'ontologie concernée est décomposée en 3 communautés (voir figure 4.3) et le concept Activity fait partie de la communauté marquée [0.00]. Soit C_1 cette communauté, alors la composition de C_1 est donnée ci-dessous :
C_1 = {UrbanArea, City, Campground, Sightseeing, RuralArea, BedAndBreakfast, BudgetHotelDestination, Destination, AccommodationRating, Capital, BudgetAccommodation, BackpackersDestination, RetireeDestination, Museums, Accommodation, Farmland, FamilyDestination, Activity, QuietDestination, BunjeeJumping, Contact, Adventure, Cairns, TwoStarRating, CapeYork, Sydney, Canberra, Woomera, ThreeStarRating, OneStarRating, string, int}

Soit K l'ensemble des entités de C_1 qui rendent inconsistant l'invariant de l'opération de suppression du concept Activity. Alors nous avons :

K = {Sightseeing, Adventure, Safari, BunjeeJumping, Museums, Destination}

Les poids de dépendance non nuls entre les entités de C_1 sont donnés par le tableau suivant.

Entité E_i	Entité E_j	p(E_i, E_j)
UrbanArea	City	1
UrbanArea	Capital	1
City	Capital	1
City	Cairns	1
Sightseeing	Safari	1
Sightseeing	Museums	1
RuralArea	Farmland	1
RuralArea	CapeYork	1
RuralArea	Woomera	1
Destination	UrbanArea	1
Destination	RuralArea	1
Destination	Farmland	1
Destination	City	1
Destination	Capital	1
Destination	Activity	1
AccommodationRating	Accommodation	1
AccommodationRating	TwoStarRating	1
AccommodationRating	ThreeStarRating	1
AccommodationRating	OneStarRating	1
Capital	Canberra	1
Capital	Sydney	1
Accommodation	Campground	1
Accommodation	BedAndBreakfast	1
Accommodation	Destination	1
Activity	Sightseeing	1
Activity	Adventure	1
Activity	Safari	1
Activity	BunjeeJumping	1
Activity	Museums	1
Activity	Destination	1
Contact	Activity	1
Adventure	BunjeeJumping	1
Adventure	Safari	1
	$\sum$p(E_i, E_j)	33

Ainsi, nous obtenons :

$$\mathrm{I}^d_{C_1}(\text{SupprimerConcept(Activity)}) = \mathrm{I}^d_{C_1}(\mathrm{K}) = \frac{\sum_{E_i \in K} p(Activity, E_i)}{\sum_{E_j, E_k \in C_1} p(E_j, E_k)}$$

$$\mathrm{I}^d_{C_1}(\text{SupprimerConcept(Activity)}) = \frac{p(Activity, Sightseeing) + p(Activity, Adventure)}{\sum_{E_j,E_k \in C_1} p(E_j, E_k)}$$
$$\frac{+p(Activity, Safari) + p(Activity, BunjeeJumping)}{\sum_{E_j,E_k \in C_1} p(E_j, E_k)}$$
$$\frac{+p(Activity, Museums) + p(Activity, Destination)}{\sum_{E_j,E_k \in C_1} p(E_j, E_k)}$$
$$\mathrm{I}^d_{C_1}(\text{SupprimerConcept(Activity)}) = \mathrm{I}^d_{C_1}(\mathrm{K}) = \frac{1+1+1+1+1+1+1}{33}$$
$$\mathrm{I}^d_{C_1}(\text{SupprimerConcept(Activity)}) = 0.18$$

Propriétés de la mesure $\mathbf{I}^d_C$

En plus de la monotonicité, de l'indépendance et de la consistance, I^d_C vérifie la normalisation. La normalisation se justifie par le fait que :

$\sum_{E_j \in K} \mathrm{p}(\mathrm{E}_i, \mathrm{E}_j) \leq \sum_{E_k,E_l \in C} \mathrm{p}(\mathrm{E}_k, \mathrm{E}_l$ car $\mathrm{K} \subseteq \mathrm{C}$ et $\mathrm{p}(\mathrm{E}_i, \mathrm{E}_j) \geq 0 \; \forall \; \mathrm{E}_i, \mathrm{E}_j$.

4.6.5 Mesure d'inconsistances dans l'ontologie globale

Nous considérons une opération de modification simple dans une ontologie volumineuse.

Définition 24 (Relation inter-communautés) *Deux communautés C et C' sont connectés s'il existe une entité $E_i \in C$ et une entité $E_j \in C'$ telles que $w(E_i, E_j) \neq 0$.*

Définition 25 *Soit Op une opération de modification d'une entité E_i sur une ontologie O de la forme AjouterConcept(E_i), AjouterType(E_i), SupprimerConcept(E_i) ou SupprimerType(E_i) et Invariant l'invariant de l'opération Op soit K un ensemble contenant tous les entités E_j qui rendent l'ensemble Invariant inconsistant. La mesure d'inconsistances de l'opération Op dans l'ontologie O est définie comme suit :*

$$I_O(Op(E_i)) = I_O(K) = \frac{I^d_C(K) * NbRel(C)}{N} \tag{4.7}$$

où C la communauté d'appartenance de l'entité E_i, I^d_C mesure d'inconsistances dans la communauté C, NbRel(C) le nombre de communautés auxquelles C est connecté et N le nombre total de communautés dans l'ontologie O.

Example 4.6

Reprenons la même opération de suppression du concept **Activity** de l'ontologie donnée en B.1. Alors nous avons :

En effet, Activity appartient à la communauté $\mathrm{C}_{[0.00]} = \mathrm{C}_1$ qui est connectée aux deux autres communautés de l'ontologie O (voir figure 4.3).

$$\mathrm{I}_O(\mathrm{Op}(\mathrm{SupprimerConcept}(\mathrm{Activity})) = \mathrm{I}_O(\mathrm{K}) = \frac{I^d_{C_1}(K) * 2}{3}$$

$$\mathrm{I}_O(\mathrm{Op}(\mathrm{SupprimerConcept}(\mathrm{Activity})) = \frac{0.18 * 2}{3} = 0.12$$

4.7 Évaluation des mesures

Dans cette section, nous comparons notre mesure d'inconsistances I_O à d'autres mesures. Nous avons choisi les mesures les plus usuelles telles la mesure d'inconsistances MI Shapley [HK10] utilisant la valeur de Shapley et la mesure η-inconsistances [Thi16a], une approche probabiliste. Ce choix s'explique par le fait qu'il existe principalement deux grandes familles d'approches de mesures d'inconsistances : les approches probabilistes et les approches utilisant la logique propositionnelle classique.

La mesure d'inconsistances MI Shapley

La valeur de Shapley, utilisée dans la théorie de jeux, est une fonction qui mesure les chances d'un joueur dans une partie de jeu en coalition. La sortie représente un vecteur où le i-ème composant désigne les chances ou forces du i-ème joueur dans la partie. La valeur de Shapley est donnée par l'expression suivante :

$$S_i(v) = \sum_{C \subseteq N} \frac{(c-1)!(n-c)!}{n!}(v(C) - v(C \setminus \{i\})) \tag{4.8}$$

où c désigne la cardinalité d'une coalition C de joueurs, n le nombre total de joueurs et N l'ensemble des joueurs.

La mesure d'inconsistances basée sur la valeur de Shapley est une fonction qui essaie de déterminer la part d'une inconsistance de chaque formule dans l'incohérence produite dans une base de connaissances. Hunter et Konieczny utilisent cette valeur de Shapley pour définir une mesure d'inconsistances drastique appliquée à l'ensemble minimum inconsistant MI (voir définition 2). La formule suivante donne l'expression de la mesure :

$$\widehat{S}^{I_{MI}}(K) = \max_{\alpha \in K} S^{I_{MI}}_{\alpha}(K) \tag{4.9}$$

avec

$\mathrm{S}^{I_{MI}}_{\alpha}(\mathrm{K}) = (\mathrm{S}^{I_{MI}}_{\alpha_1}(\mathrm{K}), ..., \mathrm{S}^{I_{MI}}_{\alpha_n}(\mathrm{K}))$,

$\mathrm{S}^{I_{MI}}_{\alpha_i}(\mathrm{K}) = \sum_{C \subseteq N} \frac{(c-1)!(n-c)!}{n!}(\mathrm{I}_{MI}(\mathrm{C}) - I_{MI}(\mathrm{C} \setminus \{\alpha_i\}))$

et $\mathrm{K} = (\alpha_1, ..., \alpha_n)$

La mesure η-inconsistances

La mesure η-inconsistances (définition 9) maximise la probabilité de l'implication du nombre minimal de formules dans l'inconsistance produite dans une base de connaissances. Sa formule est donnée par :

$$I_\eta(K) = 1 - \max\{\eta \mid \exists \hat{P} \in \mathcal{F}^2 : \forall c \in K : \hat{P}(c) \geq \eta\} \tag{4.10}$$

La comparaison des deux mesures d'inconsistances avec la mesure que nous avons proposée dans cette thèse s'applique sur l'ontologie O_1 donnée en annexe B.1. Nous devons d'abord exprimer cette ontologie en langage logique pour pouvoir appliquer les mesures I_η et $\widehat{S}^{Id}$.

O_1 = {UrbanArea, City, LuxuryHotel, Hotel, Campground, Town, Sightseeing, RuralArea, NationalPark, BedAndBreakfast, BudgetHotelDestination, Destination, AccommodationRating, Capital, BudgetAccommodation, BackpackersDestination, Beach, RetireeDestination, Hiking, Sunbathing, Museums, Accommodation, Yoga, Relaxation, Farmland, FamilyDestination, Surfing, Activity, QuietDestination, BunjeeJumping, Contact, Adventure, Safari, Sports, Coonabarabran, Cairns, FourSeasons, TwoStarRating, CapeYork, BlueMountains, Sydney, BondiBeach, CurrawongBeach, Warrumbungles, Canberra, Woomera, ThreeStarRating, OneStarRating, string, int}
Nous considérons une opération de suppression du concept **Acitivty** que nous matérialisons par ¬Acitivty selon les tableaux 4.1 et 4.2.

Nous déterminons d'abord l'ensemble minimum inconsistant de l'invariant de l'opération de suppression.
Invariant = {¬H^C(Sightseeing, Activity), ¬H^C(Adventure, Activity), ¬H^C(Relaxation, Activity), ¬H^C(Sports, Activity), ¬H^C(Safari, Activity), ¬H^C(BunjeeJumping, Activity), ¬H^C(Hiking, Activity), ¬H^C(Surfing, Activity), ¬H^C(Museums, Activity), ¬H^C(Yoga, Activity), ¬H^C(Sunbathing, Activity), ¬σ_R(Destination, hasActivity, Activity)}

Posons K = ¬Invariant = {¬Acitivty, H^C(Sightseeing, Activity), H^C(Adventure, Activity), H^C(Relaxation, Activity), H^C(Sports, Activity), H^C(Safari, Activity), H^C(BunjeeJumping, Activity), H^C(Hiking, Activity), H^C(Surfing, Activity), H^C(Museums, Activity), H^C(Yoga, Activity), H^C(Sunbathing, Activity), σ_R(Destination, hasActivity, Activity)}

– **Pour mesure d'inconsistances MI Shapley**, nous avons :

MI(K) = {{¬Activity, H^C(Sightseeing, Activity)}, {¬Activity, H^C(Adventure, Activity)}, {¬Activity, H^C(Relaxation, Activity)}, {¬Activity, H^C(Sports, Activity)}, {¬Activity, H^C(Safari, Activity)}, {¬Activity, H^C(BunjeeJumping,

Activity)}, {¬Activity, H^C(Hiking, Activity)}, {¬Activity, H^C(Surfing, Activity)}, {¬Activity, H^C(Museums, Activity)}, {¬Activity, H^C(Yoga, Activity)}, {¬Activity, H^C(Sunbathing, Activity)}, {¬Activity, σ_R(Destination, hasActivity, Activity)}}
Or

$S^{Id}_{\{\neg Activity, H^C(Sightseeing, Activity)\}}(MI(K)) = \sum_{C \subseteq N} \frac{(c-1)!(n-c)!}{n!}(I_{MI}(C) - I_{MI}(C \setminus \{\neg Activity, H^C(Sightseeing, Activity)\}))$

$S^{Id}_{\{\neg Activity), H^C(Sightseeing, Activity)\}}(MI(K)) = \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\varnothing)) + \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Adventure, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Relaxation, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Sports, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Safari, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(BunjeeJumping, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Hiking, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Surfing, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Museums, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Yoga, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(\sigma_R(Destination, hasActivity, Activity)\}))$
$+ \frac{(2-1)!(12-2)!}{12!}(I_{MI}(\{\neg Activity, H^C(Sightseeing, Activity)\}) - I_{MI}(\{H^C(Sightseeing, Activity), H^C(Sunbathing, Activity)\}))$

$S^{Id}_{\{\neg Activity), H^C(Sightseeing, Activity)\}}(MI(K)) = \frac{1}{12*11}(2 - 0) + \frac{1}{12*11}(2 - 0) +$

$\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0)+ $\frac{1}{12*11}$(2 - 0) = $\frac{12*2}{12*11}$ = 0.18

De la même manière

$S^{Id}_{\{\neg Activity), H^C(Adventure, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity), H^C(Adventure, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Relaxation, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Relaxation, Activity)}) - $I_{MI}(\varnothing)$) = 0.18)

$S^{Id}_{\{\neg Activity, H^C(Sports, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Sports, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Safari, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Safari, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(BunjeeJumping, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(BunjeeJumping, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Hiking, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Hiking, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Surfing, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Surfing, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Museums, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Museums, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Yoga, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Yoga, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, H^C(Sunbathing, Activity)\}}$(MI(K)) = $\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, H^C(Sunbathing, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

$S^{Id}_{\{\neg Activity, \sigma_R(Destination, hasActivity, Activity)\}}$(MI(K)) =

$\frac{(2-1)!(2-2)!}{2!}$(I$_{MI}$({¬Activity, σ_R(Destination, hasActivity, Activity)}) - $I_{MI}(\varnothing)$) = 0.18

Ainsi donc :

$$\widehat{S}^{I_{MI}}(\mathrm{K}) = \max_{\alpha \in K} \mathrm{S}_{\alpha}^{I_{MI}}(\mathrm{K}) = 0.18$$

– **Pour la mesure η-inconsistances**, nous devons d'abord définir une fonction de probabilités dans la base K.
Soit P $\in$ $\mathbb{P}$({¬Acitivty, H^C(Sightseeing, Activity), H^C(Adventure, Activity), H^C(Relaxation, Activity), H^C(Sports, Activity), H^C(Safari, Activity), H^C(BunjeeJumping, Activity), H^C(Hiking, Activity), H^C(Surfing, Activity), H^C(Museums, Activity), H^C(Yoga, Activity), H^C(Sunbathing, Activity), σ_R(Destination, hasActivity, Activity)}) une fonction de probabilité définie comme suit :

P(¬Acitivty) = P(H^C(Sightseeing, Activity)) = P(H^C(Adventure, Activity)) = P(H^C(Relaxation, Activity)) = P(H^C(Sports, Activity)) = P(H^C(Safari, Activity)) = P(H^C(BunjeeJumping, Activity)) = P(H^C(Hiking, Activity)) = P(H^C(Surfing, Activity)) = P(H^C(Museums, Activity)) = P(H^C(Yoga, Activity)) = P(H^C(Sunbathing, Activity)) = P(σ_R(Destination, hasActivity, Activity)) = 1/12

Alors nous avons :

$$\forall\ \theta \in \mathrm{K},\ \mathrm{P}(\theta) \geqslant 1/12$$

Donc :

$$\mathrm{I}_{\eta}(\mathrm{K}) = 1 - 1/12 = 0.91$$

– **Pour notre approche**, nous avons calculé et trouvé que :

$$\mathrm{I}_O(\mathrm{K}) = \frac{0.18 * 2}{3} = 0.12$$

Discussion

Lorsque nous faisons une étude comparative sur le procédé et le calcul de ces trois mesures, nous relevons trois niveaux d'appréciation.

1. **La base de connaissances**
   Les deux mesures I$_{\eta}$ et $\widehat{S}^{I_{MI}}$, comme pour la plupart des mesures d'inconsistances s'appliquent dans une base de connaissances exprimée en termes de formules logiques. Cela implique la traduction de l'ontologie et son évaluation en langage logique (TBox, ABox par exemple) si cette dernière est exprimée en RDF ou OWL. La difficulté à ce niveau survient lorsque l'ontologie à traduire est très volumineuse.
2. **L'expression de la mesure**
   Concernant l'expression de la mesure, nous remarquons que la mesure $\widehat{S}^{I_{MI}}$ a une formulation complexe. Même si pour I$_{\eta}$, nous ne notons pas cette complexité, nous avons par contre l'établissement aléatoire des probabilités des

différentes formules de la base de connaissances.

3. **L'interprétation de la valeur trouvée**
   La différence majeure entre ces mesures d'inconsistances et celle que nous porposons reste l'interprétation de la valeur donnée par le calcul. La valeur résultant de notre approche est un ratio qui donne le nombre d'entités directement touchées par l'opération de modification alors que la mesure I_η nous renseignement seulement qu'il y a un fort taux d'inconsistances.

4.8 Synthèse

Une mesure d'inconsistances quantifie l'implication des axiomes dans l'inconsistance produite par un changement dans une base de connaissances. Pour une ontologie, elle permet de mesurer le degré d'impact d'une opération de modification sur les entités de l'ontologie et les objets dépendants. L'ontologie concernée peut être représentée, sous forme de structure lexicale, par un n-uplet :

$$\mathrm{O} = \{\mathrm{C},\ \mathrm{R},\ \mathrm{A},\ \mathrm{T},\ \mathrm{CAR}_R,\ \mathrm{H}^C,\ \mathrm{H}^R,\ \sigma_R,\ \sigma_{CARR},\ \sigma_A\}$$

Elle peut aussi être un réseau sémantique modélisée par un graphe orienté étiqueté appelé graphe ontologique. Ses éléments sont des entités liées par des relations de dépendance de différents types susceptibles de subir des opérations de changements Δ tel que :

$$\Delta = <\text{Operation, Arguments, Assertions}>$$

où Operation représente l'opération de modification, Arguments désigne les arguments de l'opération et Assertion = (Pre-conditions, Invariant, Post-conditions) les conditions rendant l'opération exécutable. Une opération de modification Δ d'une entité E_i sur une ontologie simple O produirait une inconsistance que la mesure I_e permet de quantifier par la formule ci-dessous :

$$\mathrm{I}_e(\mathrm{Op}(\mathrm{E}_i)) = \mathrm{I}_e(\mathrm{K}) = \frac{\sum_{E_j \in K} p(E_i, E_j)}{\sum_{E_k, E_l \in O} p(E_k, E_l)}$$

Cependant, pour une ontologie de grande taille, il est nécessaire de la partitionner en communautés afin de pouvoir mesurer les impacts produits par les opérations de changement. Le partitionnement consiste à regrouper les composants d'une ontologie en communautés de sorte que les entités dans une communauté partagent plus de propriétés à l'intérieur de la communauté qu'en dehors. Ainsi, la mesure de l'inconsistance produite par une opération de modification Op d'une entité E_i dans une ontologie O de grande taille est donnée par la formule suivante :

$$\mathrm{I}_O(\mathrm{Op}(\mathrm{E}_i)) = \mathrm{I}_O(\mathrm{K}) = \frac{I_C(K) * NbRel(C)}{N}$$

où C la communauté d'appartenance de l'entité E_i, NbRel(C) le nombre de communautés auxquelles C est connecté, N le nombre total de communautés dans l'ontologie O et I_C l'inconsistance générée dans la communauté C calculée avec la formule suivante :

$$\mathrm{I}^d_C(\mathrm{Op}(\mathrm{E}_i)) = \mathrm{I}^d_C(\mathrm{K}) = \frac{\sum_{E_j \in K} p(E_i, E_j)}{\sum_{E_k, E_l \in C} p(E_k, E_l)}$$

4.9 Conclusion

Dans ce chapitre nous avons présenté deux approches de mesure d'inconsistances résultant d'une opération de modification dans une ontologie. La première mesure s'applique à une ontologie simple alors que la deuxième approche permet de quantifier l'inconsistance pour une ontologie de grande taille telle que Gene Ontology. La mesure que nous avons proposée pour les larges ontologies s'opére sur des communautés créées par un processus de partitionnement. L'algorithme de décomposition utilise les poids de dépendance entre les entités du graphe ontologique afin de déterminer les lignes de séparation des différentes communautés. Pour chaque mesure, nous avons d'abord donné les préliminaires sur lesquels elle se fonde avant de la définir et l'illustrer par des exemples d'application.

L'estimation d'impacts de changements est nécessaire pour décider de l'exécution ou non de l'opération de changement. La suivie de la propagation des impacts sur les entités de l'ontologie permet d'avoir une vue globaliste sur la sévérité du changement apporté et d'envisager une stratégie de résolution. Il est alors important de pouvoir suivre la répartition quantitative des effets de changements sur le graphe ontologique. Pour une ontologie volumineuse, le partitionnement jouera un rôle important dans le processus de propagation. Le prochain chapitre de ce manuscrit traite ces différents points.

Chapitre 5

Propagation des flux d'impacts de changements

Sommaire

5.1 Introduction

La propagation des impacts de changements était utilisée en génie logiciel pour décrire la manière qu'un changement dans un module doit nécessiter un autre changement dans d'autres modules. En ingénierie de gestion d'ontologies, elle décrit la manière dont les modifications apportées à une entité ou un groupe d'entités d'une ontologie se propagent sur les autres éléments de celle-ci. Rajlich la modélise dans [Raj97] comme une séquence de clichés et chaque cliché représente un moment particulier du processus.

Dans ce chapitre, nous proposons une approche de suivie de propagation d'impacts de changements ontologiques sur les entités de l'ontologie. L'approche marque dans un premier temps toutes les entités susceptibles d'être touchées par une modification, puis détermine le taux d'impact de cette modification sur tout élément de l'ontologie. Le calcul du taux d'impact s'effectue entre entités d'une même communauté et entre communautés de l'ontologie. Nous déterminons ensuite une méthode de calcul du degré de vulnérabilité d'une entité donnée face aux modifications éventuelles sur l'ontologie. Ces valeurs permettent à l'ingénieur de maintenance de

l'ontologie d'avoir une vue globale sur la sévérité d'une opération de modification mais aussi d'élaborer un plan de résolution des inconsistances causées.

Le chapitre comprend trois parties. La première partie concerne le processus de marquage des entités touchées par une modification dans une communauté de l'ontologie. Dans la deuxième partie, nous déterminons les formules de calcul des taux d'impacts et de vulnérabilité dans une communauté et dans la troisième partie le taux d'impacts entre communautés de l'ontologie. Une synthèse des travaux et une conclusion terminent le chapitre.

5.2 Processus de marquage des entités impactées

Dans cette section, nous déterminons les différents flux d'impacts de changements existant dans une ontologie avant de procéder au marquage des entités affectées par une opération de modification.

5.2.1 Flux de propagation d'impacts de changements ontologiques

Une ontologie représente un ensemble d'entités constituées de concepts et de types reliés par des relations de subsomption, des relations associatives et des relations d'attributs. Pour chaque type de relation entre deux entités E_i et E_j, il existe un flux d'impacts des effets de changements.

– Si C_i et C_j sont deux concepts liés par une relation de subsomption H^C telle que $H^C(C_i, C_j)$ alors une modification de C_j entraine un impact sur C_i.

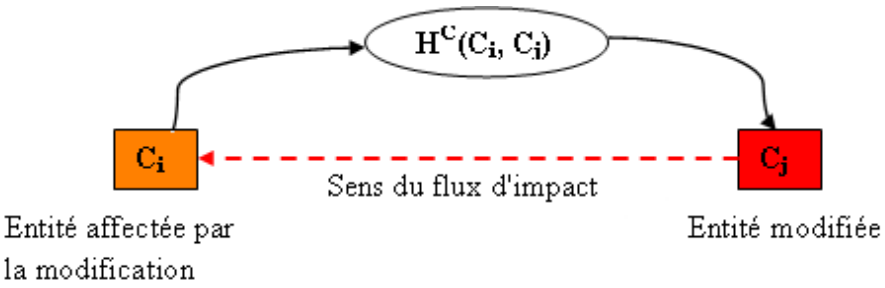

FIGURE 5.1 – Flux d'impact d'une relation d'héritage

– Si E_i et E_j sont deux entités liées par une relation associative σ_R telle que $\sigma_R(E_i, R_k, E_j)$ alors une modification de E_j entraine un impact sur E_i.

– Si E_i est une entité et T_j un type liés par une relation d'attribut σ_A telle que $\sigma_A(E_i, A_k, T_j)$ alors une modification du type T_j entraine un impact sur l'entité E_i.

La définition des sens de flux d'impacts est importante pour suivre les effets d'onde d'un changement dans l'ontologie. Le processus de propagation consiste au marquage

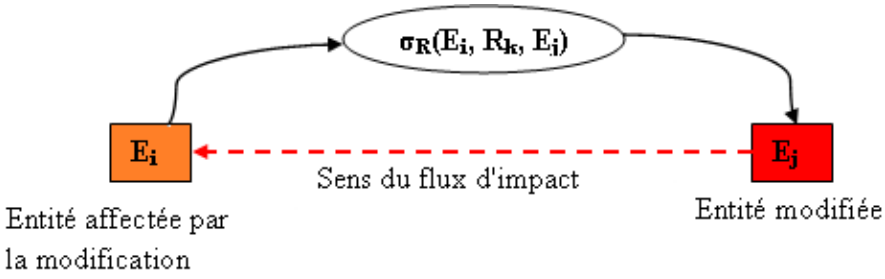

FIGURE 5.2 – Flux d'impact d'une relation associative

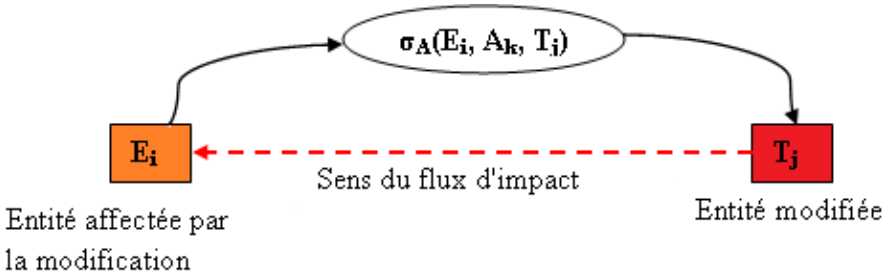

FIGURE 5.3 – Flux d'impact d'une relation d'attribut

de tous les éléments touchés par une opération de modification. Dans la section qui suit, nous proposons un algorithme de marquage des entités impactées au sein d'une partition de l'ontologie.

5.2.2 Marquage des entités impactées dans une communauté

L'algorithme de marquage permet d'identifier toutes les entités affectées par une opération de changement opérée dans une communauté de l'ontologie. Il s'appuie sur les travaux [Raj97], [Der01] sur la propagation des impacts de changements dans un logiciel et sur les travaux dans [STB+12] abordant la propagation d'impacts de changements ontologiques.

Principe de l'algorithme de marquage

L'algorithme de marquage des entités touchées par une modification dans une communauté de l'ontologie consiste d'abord à récupérer l'invariant de l'opération si ce dernier n'est pas vide. Ensuite, toute assertion de l'invariant non vérifiée est marquée. Le marquage d'une assertion donne comme résultat des entités marquées et de nouvelles assertions à marquer éventuellement et qui seront ajoutées à l'invariant. Marquer une assertion consiste à identifier toutes les entités et relations affectées par la modification et les marquer à tour de rôle. Le marquage d'une entité E_i consiste au marquage de E_i et de toutes les relations $Relation_i \in \{\sigma_R, H^C, \sigma_{CARR}, \sigma_A\}$ qui ont E_i comme source. Marquer une relation $Relation_i$ se fait en marquant les entitéss cibles de cette relation s'ils ne sont pas déjà marquées.

TABLE 5.1 – Assertions de marquage

Id	Assertions	Signification
01	marquerConcept(C_i)	1) $\forall(C_j \in C)$ si $H^C(C_j, C_i)$ alors marquerRelation($H^C(C_j, C_i)$) 2) $\forall(C_j \in C)$ et $\forall(R_k \in R)$ si $\sigma_R(C_j, R_k, C_i)$ alors marquerRelation($\sigma_R(C_j, R_k, C_i)$)
02	marquerRelation(R_k)	$\forall(C_i \in C)$ et $\forall(C_j \in C)$ si $\sigma_R(C_j, R_k, C_i)$ alors marquerRelation($\sigma_R(C_j, R_k, C_i)$)
03	marquerAttribut(A_k)	$\forall(C_i \in C)$ et $\forall(T_j \in T)$ si $\sigma_A(C_i\ A_k, T_j)$ alors marquerRelation($\sigma_A(C_i\ A_k, T_j)$)
04	marquerType(T_k)	$\forall(A_i \in A)$ et $\forall(C_j \in C)$ si $\sigma_A(C_j\ A_i, T_k)$ alors marquerRelation($\sigma_A(C_j\ A_i, T_k)$)
05	marquerCarac(CAR_{Rk})	$\forall(R_i \in R)$ si $\sigma_{CARR}(R_i, CAR_{Rk})$ alors marquerRelation($\sigma_{CARR}(R_i, CAR_{Rk})$)
06	marquerRelation($H^C(C_j, C_i)$)	Si (C_j non marqué) alors marquerConcept(C_j)
07	marquerRelation($\sigma_R(C_j, R_k, C_i)$)	Si (C_j non marqué) alors marquerConcept(C_j)
08	marquerRelation($\sigma_A(C_i, A_k, T_j)$)	Si (C_i non marqué) alors marquerConcept(C_i)
09	marquerRelation($\sigma_{CARR}(R_i, CAR_{Rk})$)	Si (R_i non marquee) alors marquerRelation(R_i)
10	marquerRelation($H^C(*, C_i)$)	$\forall(C_k \in C)$ si($H^C(C_k, C_i)$) et (C_k non marqué) alors marquerConcept(C_k)
11	marquerRelation($\sigma_{CARR}(*, CAR_{Ri})$)	$\forall(R_k \in R)$ si ($\sigma_{CARR}(R_k, CAR_{Ri})$ et (R_k non marqué) alors marquerRelation(R_k)
12	marquerRelation($\sigma_A(*, A_i, T_j)$)	$\forall(C_k \in C)$, si ($\sigma_A(C_k, A_i, T_j)$) et (C_k non marqué) alors marquerConcept(C_k)
13	marquerRelation($\sigma_A(*, *, T_j)$)	$\forall(C_k \in C)$, $\forall(A_i \in A)$ si ($\sigma_A(C_k, A_i, T_j)$) alors si (C_k non marqué) alors marquerRelation($\sigma_A(C_k, A_i, T_j)$) si (A_i non marqué) alors marquerAttribut(A_i))
14	marquerRelation($\sigma_R(*, *, C_k)$)	$\forall(R_j \in C)$, $\forall(C_i \in C)$ si ($\sigma_R(C_i, R_j, C_k)$) alors si (C_i non marqué) alors marquerConcept(C_i)
15	marquerRelation($\sigma_R(*, R_k, *)$))	$\forall(C_i \in C)$, $\forall(C_j \in C)$ si ($\sigma_R(C_i, R_k, C_j)$) alors si (C_i non marqué) alors marquerConcept(C_i)

Algorithme 4 : Marquage des entités impactées par une changement
Entrées : une communauté C et une opération de modification Op
Sorties : l'ensemble P des entités de la communauté touchées par la modification

```
executerOperation(Op)
Inv = Op.Invariant
Si (Inv ≠ ∅) alors
    P = ∅
    Pour Assertion_i ∈ Inv faire
        Si (¬Assertion_i et Assertion_i non marqué) alors
            marquer(Assertion_i)
            P = P ∪ {E_i : E_i marqué}
            Inv = Inv ∪ {Assertion_j : Assertion_j est à marquer}
        FinSi
    FinPour
FinSi
```

Exemple 5.1
La suppression du concept **Activity** (en rouge) de l'ontologie donnée en annexe B.1 déclenche un processus de marquage des entités affectées par cette modification comme le montre la figure 5.4.

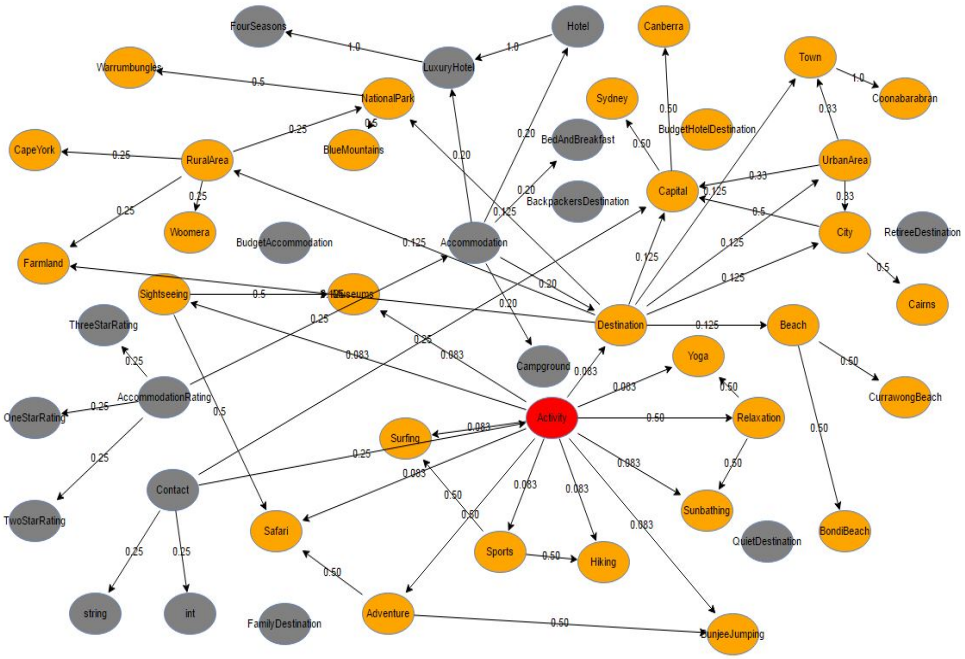

FIGURE 5.4 – Marquage des entités impactées par la suppression du concept Activity de B.1

Les entités de couleur orange représentent les entités impactées et les entités de couleur grise celles qui ne sont pas affectées.
Dans les sections qui suivent, nous déterminons une quantification de cette répartition d'impacts à travers le graphe ontologique.

5.3 Propagation d'impacts intra-communauté

Nous avons proposé dans la section précédente un algorithme de marquage des entités ontologiques impactées par une modification d'une des leurs. La sortie de cet algorithme est un ensemble de sommets du graphe ontologique affectés par un changement. Dans cette partie, nous proposons une approche de calcul du taux de propagation des impacts de changement d'une entité ontologique sur une autre entité appartenant à la même communauté et du taux de vulnérabilité d'une entité dans la communauté. Le taux de vulnérabilité, comme nous le définirons plus loin, désigne le risque pour une entité E_i d'une partition donnée d'être impactée par une modification d'une entité E_j de la même partition. Il permet de déterminer les entités les plus exposées aux changements dans l'ontologie.

5.3.1 Taux d'impacts dans une communauté

Il s'agit dans cette sous section de calculer la répartition des inconsistances résultant de changement d'une entité ontologique sur une autre entité quelcoque appartenant à la même communauté.

Rappelons que nous avons défini un graphe ontologique comme un couple G = (E, Γ) où E est un ensemble d'entités qui peuvent être des concepts ou des types et Γ une application de E vers l'ensemble P(E) des parties de E telle que :

$$\Gamma \in \{H^C, \sigma_R, \sigma_A\}$$

Un couple (e_i, e_j) tel que $e_j \in \Gamma(e_i)$ dénote un arc de e_i à e_j. Nous notons par $l(e_i, e_j)$ la longueur de l'arc (e_i, e_j).

Définition 26 (Longueur d'un chemin) *Soit $\varphi = (e_1, e_2, ..., e_n)$ un chemin de $G = (E, \Gamma)$. La longueur du chemin notée $l(\varphi)$ est égale à la somme des longueurs des arcs de φ, autrement dit :*

$$l(\varphi) = \sum_{i=1}^{n-1} l(e_i, e_{i+1}) \tag{5.1}$$

Pour notre graphe G, tous les arcs (e_i, e_{i+1}) ont la même longueur égale à 1. Donc la longueur du chemin φ est égale au nombre d'arcs de φ.

Algorithme 5 Longueur du chemin entre deux entités
Entrées : une communauté C et deux entités E_i, E_j de C
Sorites : La longueur l du chemin (E_i, E_j)

```
1 l = 0
2 s_i = E_i
3 Tantque (s_i ≠ E_j) faire
4       P = Fils(C, s_i) // donne les sous-graphes de C ayant pour racine le sommet s_i
5       Pour P_k ∈ P faire
6           Si (E_j ∈ P_k) alors
7               l = l + 1
8               s_i = s_{i+1} : s_{i+1} ∈ P_k
9           FinSi
10      FinPour
11 FinTantque
```

Définition 27 (Taux d'impacts dans une communauté) *Soient E_i et E_j deux entités appartenant à la communauté C du graphe ontologique G = (E, Γ). Le taux d'impacts d'une modification de E_j sur le sommet E_i noté t(E_j, E_i) est défini comme suit :*

$$t(E_j, E_i) = \begin{cases} \dfrac{I_C^d(Op(E_j))}{l(E_j, E_i)} & \textit{si le chemin } (E_j, E_i) \textit{ existe} \\ 0 & \textit{sinon} \end{cases} \tag{5.2}$$

avec $I_C^d(Op(E_j))$ l'inconsistance engendrée par la modification de l'entité E_j dans la communauté C.

Le taux de propagation des effets de changement sur les éléments de l'ontologie est inversement proportionnel à la longueur du chemin de propagation. En effet, plus une entité s'éloigne de l'entité modifiée, moins elle est affectée par la modification. Ce taux d'impact définit alors la répartition des inconsistances résultant de la modification d'un nœud vers un autre nœud de la même communauté. Il permet ainsi de définir la vulnérabilité d'une entité ontologique aux changements.

Exemple 5.2
Nous considérons toujours l'opération de suppression du concept **Activity** de l'ontologie donnée en annexe B.1. Nous déterminons alors le taux d'impact sur des entités appartenant à la même communauté.
Nous avions calculé et trouvé :

I_C^d(SupprimerConcept(Activity)) = 0.18

$$\text{t(Activity, City)} = \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, City)}$$

$$\text{t(Activity, City)} = \frac{0.18}{2} = 0.09$$

$$\text{t(Activity, Carns)} = \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Carns)}$$

$$\text{t(Activity, Carns)} = \frac{0.18}{3} = 0.06$$

$$\text{t(Activity, Canberra)} = \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Canberra)}$$

$$\text{t(Activity, BondiBeach)} = \frac{0.18}{4} = 0.045$$

$$\text{t(Activity, Contact)} = 0$$

car il n'existe pas de chemin dans le graphe ontologique du concept Activity au concept Contact. La suppression de Activity ne peut pas impacter le concept Contact. Cela est d'autant plus normal que l'ensemble Invariant de l'opération de suppression du concept Activity ne contient pas le concept Contact.

5.3.2 Taux de vulnérabilité d'une entité dans une communauté

Dans cette sous partie, nous déterminons le taux de vulnérabilité d'une entité ontologique à une modification dans la communauté à laquelle elle appartient. Nous utilisons des propriétés de la Propagation de Croyances (Belief Propagation) [YFW03] pour calculer cette vulnérabilité. En effet, le problème que nous avons peut être assimilé à la détermination de la croyance d'un nœud dans un graphe ou un Réseau bayésien. En Propagation de Croyances, la croyance d'un nœud désigne la probabilité que ce nœud soit crédible face à une opinion exprimée.

Exemple, considérons le jeu de lancer de dé à six faces numérotés de 1 à 6. Les chiffres possibles pouvant être tirés par le joueur sont appelés hypothèses et forment l'ensemble Ω. Les hypothèses sont exclusives, ce qui signifie que deux hypothèses ne peuvent pas être vraies simultanément.

La face tirée à l'aide du dé est en partie cachée et seul le point central est visible. Le joueur émet alors une opinion pondérée sur le chiffre du dé. Cette opinion représente une croyance sur l'événement considéré, qui peut être le tirage de F3 ou F5.

5.3.2.1 Formalisme du problème

Nous considérons deux hypothèses pouvant être vérifiées sur les entités ontologiques : $\Omega = \{\text{affecté, indemne}\}$. L'objectif est de pouvoir déterminer l'état $\omega_0 \in \Omega$ d'une entité E_i au sein d'une communauté C de l'ontologie. L'observation de l'état d'une entité est assurée par des agents qui fournissent une opinion pondérée sur

l'état réel $\omega_0 \in \Omega$ de l'entité. Ainsi, le taux de vulnérabilité que nous voulons déterminer renseigne sur l'appartenance de l'entité concernée à l'état affecté ou indemne selon que la valeur calculée soit importante ou faible.

5.3.2.2 Calcul du taux de vulnérabilité

Définition 28 (Taux de vulnérabilité) *Soit E_i une entité appartenant à la communauté C du graphe ontologique G = (E, Γ). Le taux de vulnérabilité v(E_i) de E_i dans la communauté C est défini comme suit :*

$$v(E_i) = k\phi_i(E_i) \sum_{E_j \in C} t(E_j, E_i) \tag{5.3}$$

avec $\phi_i(E_i)$ une fonction qui désigne l'existence d'un chemin entre E_j et E_i (elle est égale à 1 si ce chemin existe 0 sinon) et k une constante de normalisation.

La constante de normalisation est nécessaire car nous devons avoir :
$\sum_{E_i \in C} \mathrm{v}(\mathrm{E}_i) = 1$

Exemple 5.3
Pour l'ontologie donnée en annexe B.1, le taux de vulnérabilité du concept **Destination** dans sa communauté, pour une opération de suppression est de :

v(Destination) = k*(t(Accommodation, Destination) + t(Activity, Destination))

$$\mathrm{v(Destination)} = \mathrm{k}\left(\frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, Destination)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Destination)}\right)$$

$$\mathrm{v(Destination)} = \mathrm{k}\left(\frac{0.09}{1} + \frac{0.18}{1}\right) = 0.27\mathrm{*k}$$

5.4 Propagation d'impacts inter-communautés

Dans la section précédente, nous avons défini le taux de propagation d'impacts d'une entité vers une autre de la même communauté. Avec cette valeur, nous avons déterminé le degré de vulnérabilité de chaque entité de la partition face aux changements éventuellement produits au sein de cette partition. Dans cette section, nous étendons ce travail aux communautés formant le graphe ontologique. Nous déterminons le taux de propagation d'impacts d'une partition vers une autre, puis pour une partition donnée, nous estimons son degré de vulnérabilité aux modifications apportées à l'ontologie.

5.4.1 Taux d'impacts inter-communautés

Définition 29 (Connexion inter-communautés) *Deux communautés C et C' sont connectées s'il existe au moins une entité $E_i \in C$ et une entité $E_j \in$ telles que E_i et E_j soient connectés.*

Cette connexion sera utilisée dans la définition du taux d'impacts entre communautés.

Définition 30 (Taux d'impacts inter-communautés) *Soient C et C' deux communautés du graphe ontologique G = (E, Γ). Le taux d'impact d'une modification dans C sur la communauté C' est défini comme suit :*

$$t(C, C') = \sum_{E_j \in C, E_i \in C'} t(E_j, E_i) \tag{5.4}$$

La formule de calcul du taux d'impacts inter-communautés se justifie par le fait que chaque connexion peut être source de propagation d'impacts entre les deux communautés.

Exemple 5.4

Nous avons partitionné l'ontologie donnée en annexe B.1 en trois communautés ([0.00], [0.50] et [1.00] voir figure 5.5). Déterminons la propagation des impacts entre les communautés $C_{[1.00]}$, $C_{[0.50]}$ et $C_{[0.00]}$. L'opération de modification concernée est une suppression de concept.

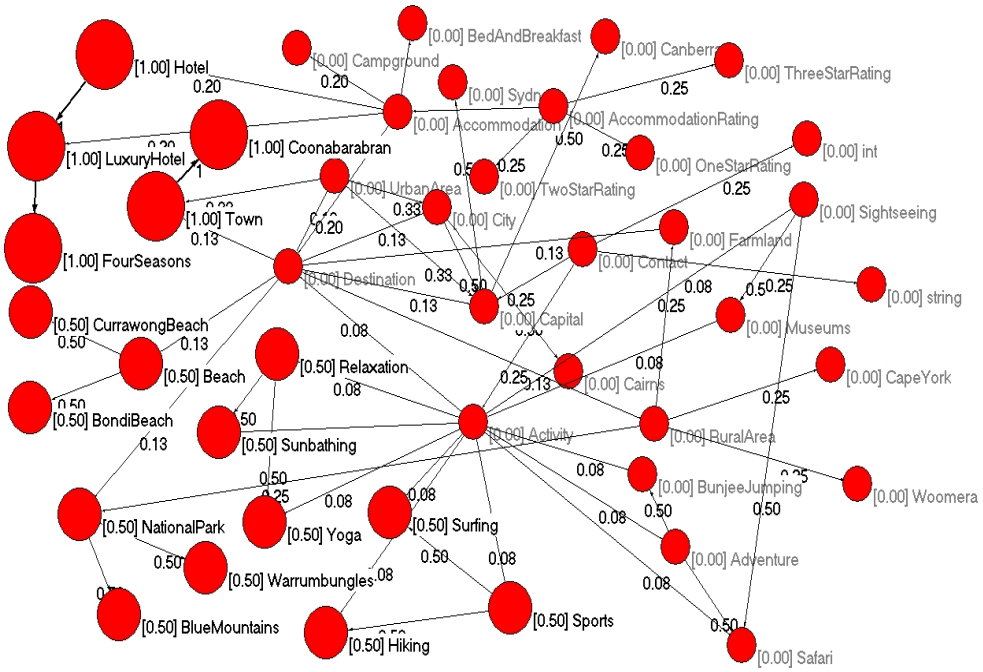

FIGURE 5.5 – Partition de l'ontologie donnée en B.1

Nous avons :

$C_{[0.00]}$ = {UrbanArea, City, Campground, Sightseeing, RuralArea, BedAndBreakfast, BudgetHotelDestination, Destination, AccommodationRating, Capital, BudgetAccommodation, BackpackersDestination, RetireeDestination, Museums, Accommodation, Farmland, FamilyDestination, Activity, QuietDestination, BunjeeJumping, Contact, Adventure, Cairns, TwoStarRating, CapeYork, Sydney, Canberra, Woomera, ThreeStarRating, OneStarRating, string, int}

$C_{[0.50]}$ = {Beach, Relaxation, CurrawongBeach, BondiBeach, Sunbathing, NationalPark, Yoga, Surfing, Hiking, Sports, BlueMountains, Warrumbungles}

$C_{[1.00]}$ = {Coonabarabran, Town, FourSeasons, Hotel, LuxuryHotel}

Après avoir déterminé tous les chemins entre les éléments de $C_{[1.00]}$ et ceux de $C_{[0.00]}$, puis entre les éléments $C_{[1.00]}$ et ceux de $C_{[0.50]}$, nous obtenons.

$t(C_{[1.00]}, C_{[0.00]}) = 0$ car il n'existe pas de chemin de $C_{[1.00]}$ vers $C_{[0.00]}$

$t(C_{[1.00]}, C_{[0.50]}) = 0$ car il n'existe de chemin de $C_{[1.00]}$ vers $C_{[0.50]}$

$t(C_{[0.50]}, C_{[0.00]}) = 0$ car il n'existe de chemin de $C_{[0.50]}$ vers $C_{[0.00]}$

$t(C_{[0.50]}, C_{[1.00]}) = 0$ car il n'existe de chemin de $C_{[0.50]}$ vers $C_{[1.00]}$

$t(C_{[0.00]}, C_{[1.00]})$ = t(Accomodation, Hotel)+t(Accomodation, LuxuryHotel)+t(Destination, Town)+t(UrbanArea, Town)+t(Accomodation, Town)+t(Activity, Town)+t(Contact, Town)+t(UrbanArea, Coonabarabran)+t(Destination, Coonabarabran)+t(Accomodation, Coonabarabran)+t(Activity, Coonabarabran)+t(Contact, Coonabarabran)+t(Accomodation, FourSeasons)

$$t(C_{[0.00]}, C_{[1.00]}) = \frac{I_C^d(SupprimerConcept(Accomodation))}{l(Accomodation, Hotel)}$$
$$+\frac{I_C^d(SupprimerConcept(Accomodation))}{l(Accommodation, LuxuryHotel)} +\frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, Town)}$$
$$+\frac{I_C^d(SupprimerConcept(UrbanArea))}{l(UrbanArea, Town)} +\frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, Town)}$$
$$+\frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Town)} +\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact, Town)}$$
$$+\frac{I_C^d(SupprimerConcept(UrbanArea))}{l(UrbanArea, Coonabarabran)} +\frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, Coonabarabran)}$$
$$+\frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, Coonabarabran)} +\frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Coonabarabran)}$$

$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact, Coonabarabran)} + \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, FourSeasons)}$$

$$t(C_{[0.00]}, C_{[1.00]}) = \frac{0.09}{1}+\frac{0.09}{2}+\frac{0.18}{1}+\frac{0.06}{1}+\frac{0.09}{2}+\frac{0.18}{2}+\frac{0.03}{3}+\frac{0.06}{2}+\frac{0.18}{2}+\frac{0.09}{3}$$
$$+\frac{0.18}{3}+\frac{0.03}{4}+\frac{0.09}{3}$$

$$t(C_{[0.00]}, {}_{[1.00]}) = 0.81$$

$t(C_{[0.00]}, C_{[0.50]})$ = t(Destination, Beach) + t(Destination, BondiBeach) + t(Destination, CurrawongBeach) + t(Destination, NationalPark) + t(Destination, BlueMountains) + t(Destination, Warrumbungles)+ t(Accommodation, Beach)+t(Accommodation, BondiBeach) + t(Accommodation, Currawong-Beach)+ t(Accommodation, NationalPark) + t(Accommodation, BlueMountains) + t(Accommodation, Warrumbungles) + t(Activity, Relaxation) + t(Activity, Sunbathing) + t(Activity, Yoga) + t(Activity, Sports) + t(Activity, Hiking) + t(Activity, Surfing) + t(Activity, NationalPark) + t(Activity, BlueMountains) + t(Activity, Warrumbungles) + t(Activity, Beach) + t(Activity, CurrawongBeach) + t(Activity, BondiBeach) + t(Contact, Relaxation) + t(Contact, Sunbathing) + t(Contact, Yoga) + t(Contact, Sports) + t(Contact, Hiking) + t(Contact, Surfing) + t(Contact, NationalPark) + t(Contact, BlueMountains) + t(Contact, Warrumbungles) + t(Contact, Beach) + t(Contact, CurrawongBeach) + t(Contact, BondiBeach)

$$t(C_{[0.00]}, C_{[0.50]}) = \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, Beach)} + \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, BondiBeach)}$$
$$+ \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, CurrawongBeach)} + \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, NationalPark)}$$
$$+ \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, BlueMountains)} + \frac{I_C^d(SupprimerConcept(Destination))}{l(Destination, Warrumbungles)}$$
$$+ \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, Beach)} + \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, BondiBeach)}$$
$$+ \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, CurrawongBeach)} + \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, NationalPark)}$$
$$+ \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, BlueMountains)} + \frac{I_C^d(SupprimerConcept(Accommodation))}{l(Accommodation, Warrumbungles)}$$
$$+ \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Relaxation)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Sunbathing)}$$
$$+ \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Yoga)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Sports)}$$
$$+ \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Hiking)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Surfing)}$$
$$+ \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, NationalPark)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, BlueMountains)}$$
$$+ \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Warrumbungles)} + \frac{I_C^d(SupprimerConcept(Activity))}{l(Activity, Beach)}$$

$$+\frac{I_C^d(SupprimerConcept(Activity))}{l(Activity,CurrawongBeach)}+\frac{I_C^d(SupprimerConcept(Activity))}{l(Activity,BondiBeach)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Relaxation)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Sunbathing)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Yoga)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Sports)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Hiking)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Surfing)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,NationalPark)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,BlueMountains)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Warrumbungles)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,Beach)}$$
$$+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,CurrawongBeach)}+\frac{I_C^d(SupprimerConcept(Contact))}{l(Contact,BondiBeach)}$$

$$\mathrm{t(C_{[0.00]},\,C_{[0.50]})}=\frac{0.18}{1}+\frac{0.18}{2}+\frac{0.18}{2}+\frac{0.18}{1}+\frac{0.18}{2}+\frac{0.18}{2}+\frac{0.09}{2}+\frac{0.09}{3}+\frac{0.09}{3}+\frac{0.09}{2}$$
$$+\frac{0.09}{3}+\frac{0.09}{3}+\frac{0.06}{1}+\frac{0.06}{1}+\frac{0.06}{1}+\frac{0.06}{1}+\frac{0.06}{1}+\frac{0.06}{1}+\frac{0.06}{2}+\frac{0.06}{3}+\frac{0.06}{3}+\frac{0.06}{2}+\frac{0.06}{3}$$
$$+\frac{0.06}{3}+\frac{0.03}{2}+\frac{0.03}{2}+\frac{0.03}{2}+\frac{0.03}{2}+\frac{0.03}{2}+\frac{0.03}{2}+\frac{0.03}{3}+\frac{0.03}{4}+\frac{0.03}{4}+\frac{0.03}{3}+\frac{0.03}{4}+\frac{0.03}{4}$$

$$\mathrm{t(C_{[0.00]},\,C_{[0.50]})}=1.84$$

5.4.2 Vulnérabilité d'une communauté dans l'ontologie

La vulnérabilité d'une communauté désigne la probabilité que cette communauté soit impactée par une modification intervenue dans une quelconque communauté de l'ontologie. Elle permet ainsi d'évaluer les risques d'un changement devant s'opérer.

Définition 31 (Taux de vulnérabilité d'une communauté) *Soit C une communauté d'un graphe ontologique G = (E, Γ). Le taux de vulnérabilité v(C) de C dans une ontologie O est défini comme suit :*

$$v(C)=k\sum_{C'\in O}t(C',C) \tag{5.5}$$

avec k une constante de normalisation.

Exemple 5.5

Avec les trois communautés de l'ontologie de l'annexe B.1 et avec k = 1/2.65, nous avons :

$$\mathrm{v(C_{[0.00]})=k(t(C_{[1.00]},\,C_{[0.00]})+t(C_{[0.50]},\,C_{[0.00]}))}=\frac{1}{2.65}(0+0)=0$$

$$\mathrm{v(C_{[0.50]})=k(t(C_{[1.00]},\,C_{[0.50]})+t(C_{[0.00]},\,C_{[0.50]}))}=\frac{1}{2.65}(0+1.84)=0.70$$

$$\mathrm{v}(\mathrm{C}_{[1.00]}) = \mathrm{k}(\mathrm{t}(\mathrm{C}_{[0.00]}, \mathrm{C}_{[1.00]}) + \mathrm{t}(\mathrm{C}_{[0.50]}, \mathrm{C}_{[1.00]}))\ \frac{1}{2.65}(0.81 + 0) = 0.30$$

La communauté $\mathrm{C}_{[0.50]}$ est alors plus exposée aux impacts de changements que la communauté $\mathrm{C}_{[1.00]}$.

5.5 Synthèse

Une ontologie est un ensemble d'entités constituées de concepts et de types reliés par des relations de subsomption, des relations associatives et des relations d'attributs. Pour chaque type de relation entre deux entités E_i et E_j de l'ontologie, il existe un flux d'impacts des effets de changement.

L'algorithme de marquage des entités touchées par une modification dans une partition de l'ontologie consiste d'abord à récupérer l'invariant de l'opération de modification si ce dernier n'est pas vide. Ensuite, toute condition non vérifiée de l'invariant est marquée et ajoutée à un ensemble recueillant les entités marquées.

Le taux d'impact d'une modification de E_j sur l'entité E_i appartenant à la même communauté est défini comme suit :

$$\mathrm{t}(\mathrm{E}_j, \mathrm{E}_i) = \begin{cases} \dfrac{I_C^d(Op(E_j))}{l(E_j, E_i)} & \text{si le chemin } (\mathrm{E}_j, \mathrm{E}_i) \text{ existe} \\ 0 & \text{sinon} \end{cases}$$

Ce taux d'impacts permet de déterminer le degré de vulnérabilité de l'entité face aux modifications éventuelles dans sa communauté calculé ainsi :

$$\mathrm{v}(\mathrm{E}_i) = \mathrm{k}\phi_i(\mathrm{E}_i)\textstyle\sum_{E_j \in C}\ \mathrm{t}(\mathrm{E}_j, \mathrm{E}_i)$$

Pour une communauté, le degré de vulnérabilité s'exprime par :

$$\mathrm{v}(\mathrm{C}) = \mathrm{k}\textstyle\sum_{C' \in O}\mathrm{t}(\mathrm{C'}, \mathrm{C})$$

où t(C', C) désigne le taux d'impact d'une modification dans C' sur la communauté C calculé comme suit :

$$\mathrm{t}(\mathrm{C'}, \mathrm{C}) = \textstyle\sum_{E_j \in C', E_i \in C}\mathrm{t}(\mathrm{E}_j, \mathrm{E}_i)$$

5.6 Conclusion

Dans ce chapitre, nous avons proposé une approche de propagation d'impacts de changements d'éléments ontologiques sur le reste de l'ontologie. L'approche marque dans un premier temps tous les éléments susceptibles d'être touchés par une modification, puis dans un deuxième temps détermine le taux d'impact de cette modification sur un élément quelconque de l'ontologie afin d'évaluer son degré de vulnérabilité. Le calcul du taux d'impact s'effectue entre les entités d'une même communauté

et entre les communautés. Pour déterminer le taux de vulnérabilité, nous avons recouru aux propriétés de la propagation de croyances qui permettent entre autre de définir une probabilité d'une opinion.

Les approches proposées dans ce chapitre et celles du chapitre précédent sont toutes illustrées par des exemples. Cependant, l'utilisation d'exemples pour illustrer une approche ne suffit pas pour la valider. Il faudra alors implémenter le système et procéder à une série de tests, ce afin de s'assurer de la validité des propositions sur des données réelles. L'architecture structurelle et fonctionnelle du système doit donc prendre en charge tous les aspects des approches. Nous consacrons le chapitre suivant à l'implémentation et la validation des différentes contributions de cette thèse.

Chapitre 6

Implémentation et validation

Sommaire

6.1 Introduction

La validation représente la phase pendant laquelle on effectue un ensemble de tests afin de vérifier les propriétés exigées par la spécification du système ou de détecter les exceptions engendrées. Elle se déroule après une implémentation des différents processus et algorithmes dont l'ensemble constitue le système. Les imperfections ou améliorations décelées durant cette phase, nécessitent alors des corrections à l'implémentation et/ou à la conception du système.

Dans ce chapitre, nous présentons le prototype SIC-Evonto développé pour valider les différentes approches proposées dans cette thèse. SIC-Evonto est une une plateforme de gestion des impacts de changements ontologiques capable d'analyser une opération de modification, de mesurer ses conséquences éventuelles et de suivre la propagation de ses impacts sur l'ontologie concernée. SIC-Evonto s'incorpore dans la plateforme SIC-Sénégal où elle prend en charge l'évolution des ontologies des partenaires et des ontologies fondamentales utilisées dans l'intégration des données produites pour la mise en valeur du fleuve Sénégal.

Le chapitre comprend trois parties. La première partie consacrée à l'architecture de la plateforme montre les composants des différentes couches et leurs interactions. La deuxième partie fait une présentation des deux outils supplémentaires Jena et

mxGraph essentiels pour la manipulation et la visualisation des ontologies du système. La troisième partie du chapitre présente les interfaces les plus saillantes de la plateforme, et est suivie d'une synthèse et d'une conclusion.

6.2 Architecture générale

L'architecture de la plateforme que nous proposons pour la gestion des changements ontologiques et de leurs effets repose sur celle mise en place dans le cadre du projet SIC-Sénégal. Cette dernière proposée dans [GLN08] est formée de hubs ou serveurs pairs dont l'architecture reste la même pour tous les partenaires. Un hub contient :

- une interface utilisateur constitué d'un serveur web proposant des applications de gestions des données ;
- une interface de programmes formée de services web permettant un accès distant aux applications du hub par d'autres applications.

La figure 6.1 donne l'architecture d'un hub selon le modèle du projet.

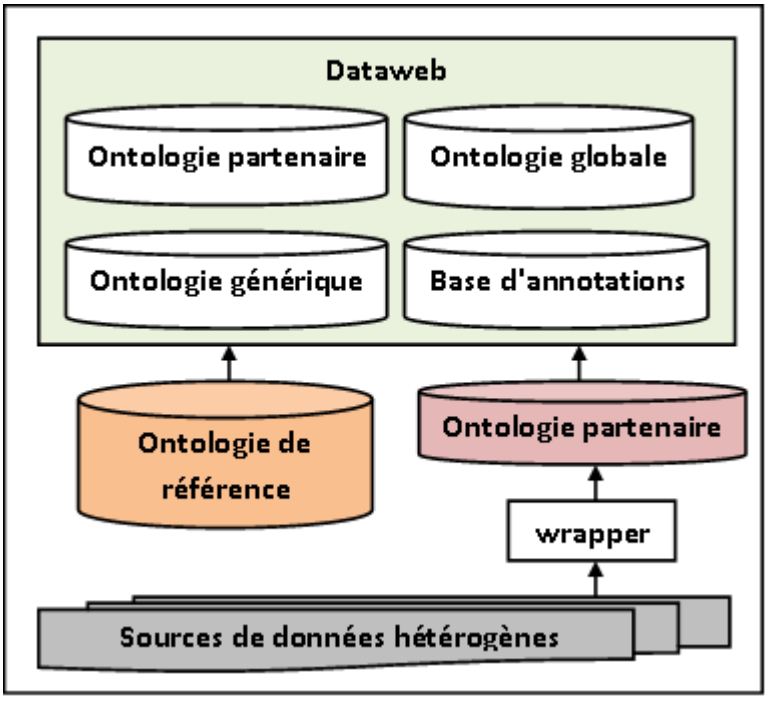

FIGURE 6.1 – Architecture d'un hub

L'architecture de notre plateforme, comme le montre la figure 6.2, ajoute une couche d'applications interagissant avec les ontologies de la base de connaissances d'un hub. La couche d'applications comprend le serveur web, le serveur d'applications intégrant les frameworks Jena et mxGraph. Elle intègre plusieurs modules dont le module de partitionnement et celui des changements ontologiques.

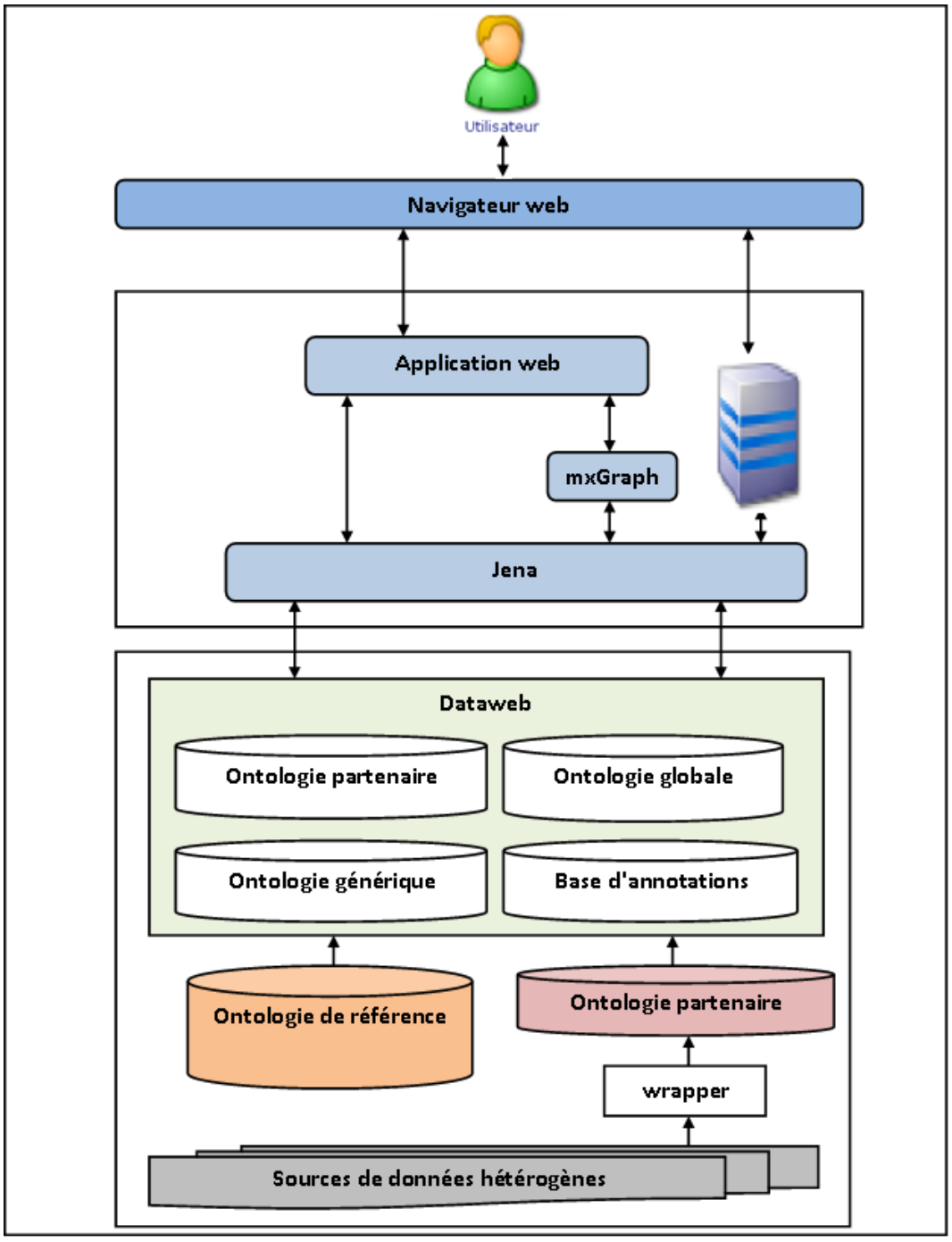

FIGURE 6.2 – Architecture générale de la plateforme

6.2.1 Architecture fonctionnelle du module de partitionnement

Le module de décomposition d'ontologies dont la figure 6.3 représente l'architecture comprend un composant d'extraction de l'ontologie à partitionner (extraite de la base de connaissances ou le dataweb). L'extraction de cette ontologie crée une copie et la charge en mémoire grâce à la fonctionne createDefaultModel de Jena (voir la section 6.3). L'ontologie est alors décomposée en communautés suivant les paramètres entrés par l'utilisateur via le navigateur. Le composant de décomposition implémente l'algorithme de partitionnement donné dans le chapitre 4. Les communautés créées ne sont pas intégrées dans le dataweb, elles sont stockées en mémoire et peuvent être visualisées sous forme de graphes dynamiques par le dernier composant qui implémente des fonctionnalités de mxGraph (voir la section 6.3).

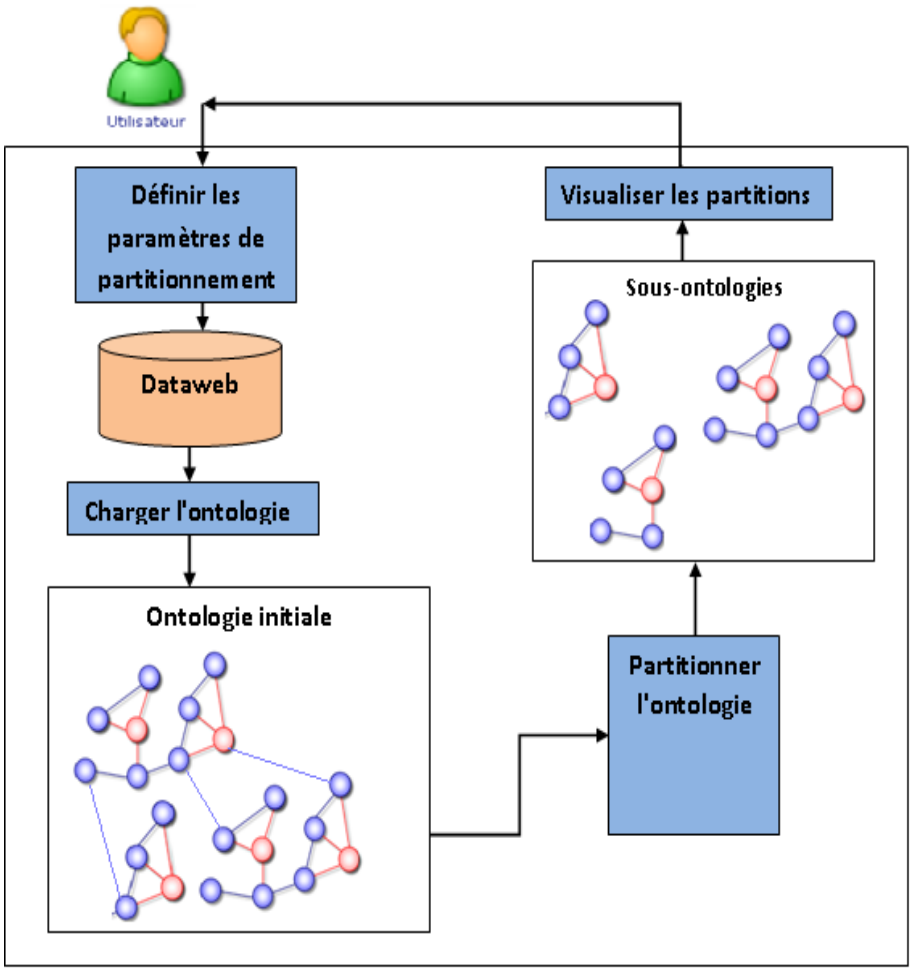

FIGURE 6.3 – Architecture du module de partitionnement

6.2.2 Architecture fonctionnelle du module de gestion des changements

Le module de gestion des changements comprend principalement deux fonctionnalités : la mesure d'inconsistances engendrées par une opération de modification et le suivi de la propagation de ses impacts. Comme le montre l'architecture à la figure 6.4, la mesure d'inconsistances commence par la spécification des paramètres de l'opération de changement (type d'opération et arguments). Un composant de vérification des assertions de cette opération implémenté sous forme de fonctions booléennes autorise le passage à la phase de mesure d'impacts. Le suivi des impacts

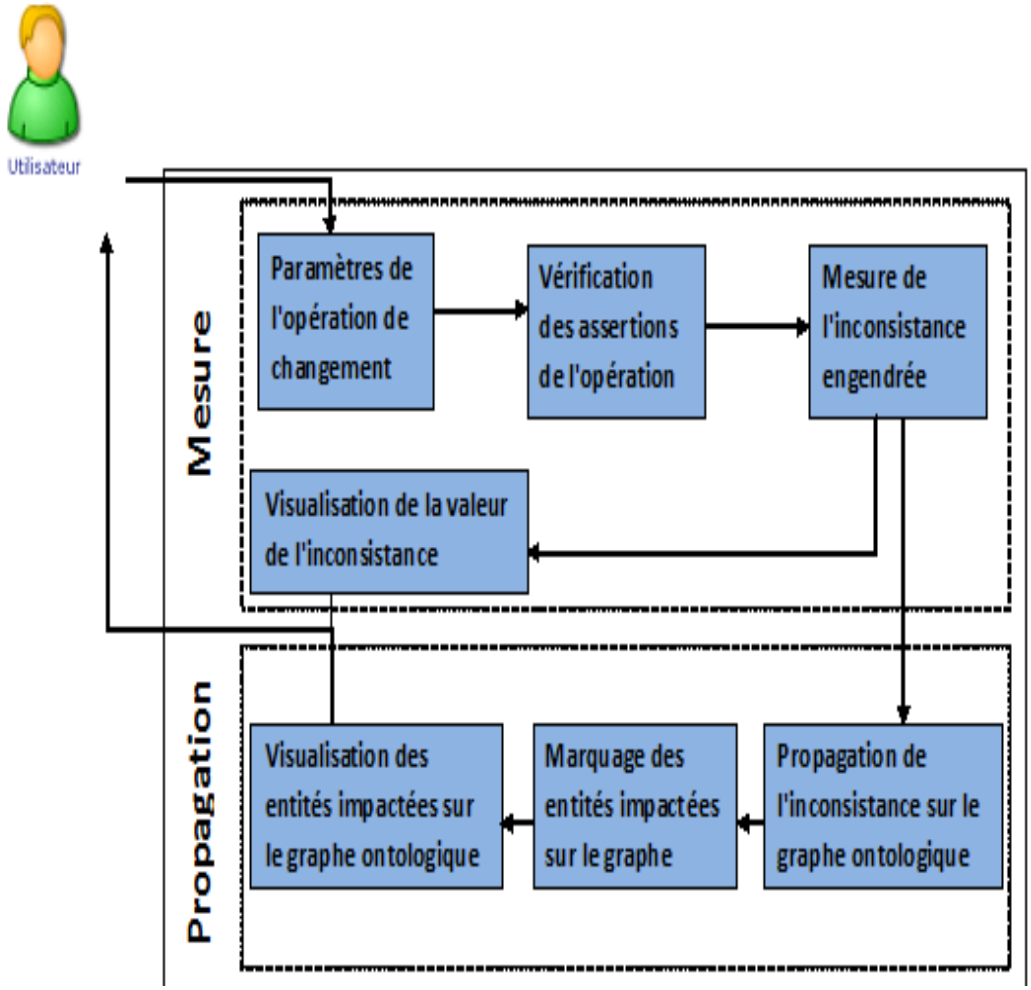

FIGURE 6.4 – Architecture du module de gestion des changements

de changement est assuré par le composant de marquage des entités affectées et le composant de propagation quantifie l'impact sur chaque entité touchée par l'exécution de l'opération. La visualisation des entités impactées sous forme de nœuds marqués dans le graphe ontologique est assuré par le composant qui prend en en-

trée l'ensemble des entités touchées par le changement. Ce composant implémente les fonctionnalités de mxGraph permettant de dessiner le graphe ontologique et d'y matérialiser les marquages.

6.3 Outils supplémentaires installés

L'implémentation de SIC-Evonto repose sur la plateforme Java Entreprise Edition (JEE) dont la modularité de l'architecture a permis l'ajout de composants tels que Jena et mxGraph pour la manipulation et la visualisation des ontologies.

6.3.1 Jena

Jena[1] est une plateforme Java pour des applications de web sémantique. Il présente une palette de bibliothèques permettant de manipuler des ontologies dans différents langages :

- une API RDF pour lire et écrire une ontologie en RDF/XML, N3, et N-Triples ;
- une API OWL
- un langage de requêtes RDF : RDFQL (RDF Query Language) ;
- une persistance des données avec surtout les SGBD Oracle, MySQL, PostgreSQL ;
- une inférence sur l'ontologie.

6.3.1.1 Des méthodes spécifiques

Le framework Jena fournit des interfaces pour représenter des graphes RDF (Model), des ressources (Resource), des propriétés (Porperty) et des littéraux (Literal).

- L'interface **Model** contient des méthodes telles que **createDefaultModel** pour créer un graphe RDF en mémoire, **createFileModelMaker** pour un graphe sur disque, ou encore **createOntologyModel** pour une ontologie et OWL, RDFS, N3, etc. L'interface fournit aussi des méthodes pour créer des ressources, des propriétés et des littéraux et les expressions qui les relient, pour ajouter des instances à une ontologie, les supprimer ou pour interroger l'ontologie.
- L'interface **Statement** permet avec ses méthodes d'accéder au sujet par la méthode **getSubject**, au prédicat par la méthode **getPredicate** et à l'objet par la méthode **getObject**.
- L'interface **Resource** fournit des méthodes pour créer des ressources(**createOntResource**). Les ressources créées peuvent être associées à un modèle spécifique accessible et modifiable avec des méthodes telles que **getProperty** et **addProperty**.
- L'inferace **Property** et ses implémentations **OntProperty**, **DatatypeProperty**, **ObjectProperty** encapsulent une propriété dans une ontologie et

1. https ://jena.apache.org/

contiennent une collection de méthodes pour accéder aux fonctionnalités sémantiques supplémentaires des propriétés dans OWL et RDFS comme le domaine, l'intervalle, l'inverse, etc.
- **Literal** : les implémentations de cette interface doivent être en mesure de prendre en charge les littéraux simples et typés.
- **Container** : cette interface définit des méthodes pour accéder aux ressources de conteneur RDF. Ces méthodes fonctionnent sur les instructions RDF contenues dans un modèle.

Jena présente d'autres fonctionnalités, y compris des outils de ligne de commande, des index spécialisés pour la recherche textuelle.

6.3.1.2 Architecture de Jena

L'architecture de Jena est centrée sur des APIs qui interagissent avec le code de l'application utilisatrice comme le montre la figure 6.5.

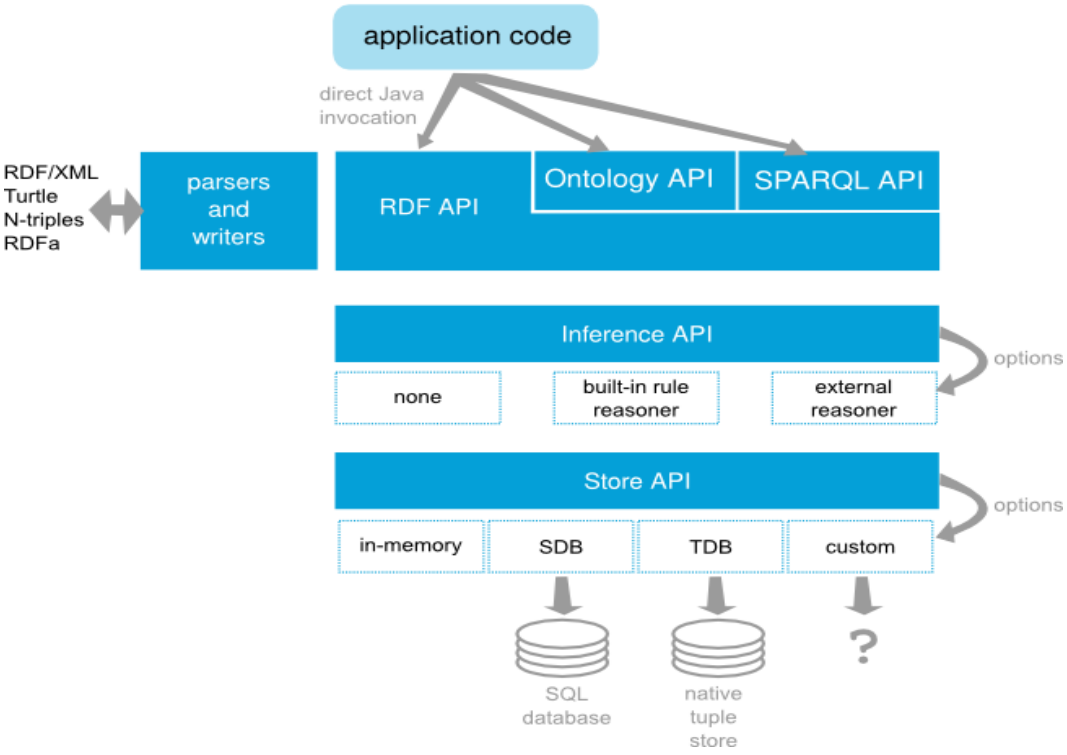

FIGURE 6.5 – Architecture de Jena

6.3.2 mxGraph

mxGraph dont l'architecture est donnée par la figure 6.6 est une librairie qui permet aux applications de visualiser des diagrammes et graphes interactifs. La

technologie mxGraph se déploie dans un environnement composé d'une librairie coté client écrit en Javascript et d'une librairie coté serveur dans un des deux langages .NET ou Java. Le code Javascript utilise un vecteur de langage graphique sur le navigateur pour afficher le diagramme à visualiser. Le client est un composant graphique avec un adaptateur d'applications intégré à une interface web existante. Il peut avoir besoin d'un serveur web pour lui fournir les fichiers requis ou peut être exécuté à partir du système de fichiers local sans serveur web. Les backends peuvent être utilisés tels quels ou intégrés dans un serveur d'applications existant développé dans l'un des langages supportés. Si un backend existe, le client peut être configuré de différentes façons, pour par exemple :

- créer des images
- stocker et charger des diagrammes
- créer une représentation objet d'un graphe.

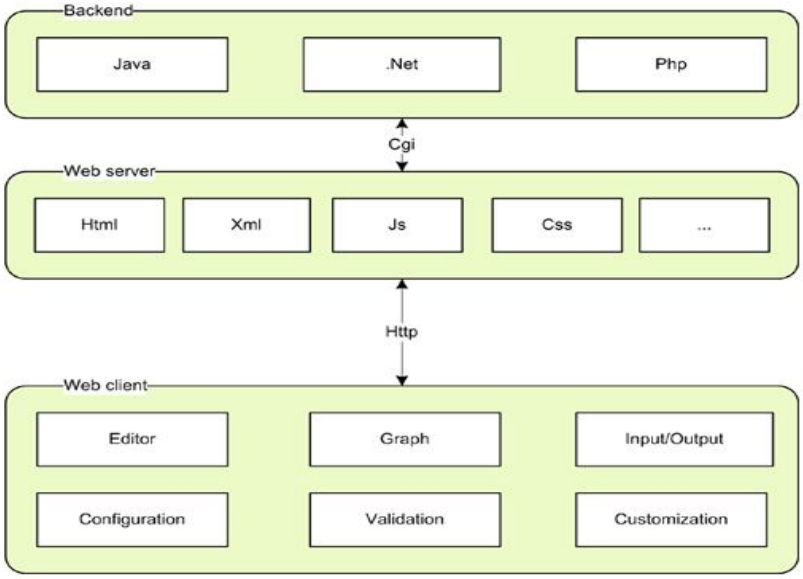

FIGURE 6.6 – Architecture de mxGraph

6.4 Prototype SIC-EvOnto

Dans cette section, nous présentons quelques interfaces de la plateforme SIC-Evonto que nous avons développée pour la gestion de changements sur des ontologies volumineuses. La plateforme comprend plusieurs modules, comme nous l'avons montré avec l'architecture, dont le partitionnement, la mesure d'inconsistances et la propagation d'impacts.

6.4.1 Module de partitionnement d'ontologies

Le module de partitionnement d'ontologies volumineuses prend en entrée l'ontologie à décomposer et le nombre maximum de communautés à obtenir comme le montre la figure 6.7.

FIGURE 6.7 – Interface de partitionnement d'ontologies

La décomposition de l'ontologie fournit le nombre de communautés obtenues et les différentes partitions. Avec les outils graphiques, chaque communauté peut être visualiser sous forme de graphe dynamique. La figure 6.8 montre le partitionnement de l'ontologie donnée en annexe B.1. Le lien **Visualiser** permet de voir le graphe des entités appartenant à une communauté de l'ontologie partitionnée.

6.4.2 Module de mesure d'inconsistances

La mesure d'inconsistances concerne les opérations de changement simples telles que l'ajout et la suppression d'entités. Le module de partitionnement crée pour chaque communauté un fichier en mémoire contenant l'ensemble des entités de cette communauté. Ainsi, le choix d'une opération de modification dans le menu proposé (figure 6.9) charge tous les éléments pouvant faire l'objet de cette modification. Après

FIGURE 6.8 – Interface de visualisation des partitions

la validation de l'opération, le module propose dans la même interface les assertions de l'opération (pré-conditions, invariant et post-conditions). A partir de là, la mesure peut être effectuée avec le bouton **Mesurer**. Elle détermine en plus de la valeur d'inconsistances qu'engendre l'opération, les entités directement et indirectement impactées.

La figure 6.10 montre les résultats de la suppression du concept Destination de l'ontologie en annexe B.1. L'interface propose un bouton pour la visualisation des entités affectées dans le graphe ontologique.

6.4.3 Module de suivi des propagations

Le module de propagation implémente des fonctionnalités de suivi des effets d'onde d'une opération de modification dans tout le graphe ontologique. Ces fonctionnalités incluent des représentations sous forme de graphe renseignant sur une répartition quantifiée des inconsistances. La figure 6.11 montre un exemple de répartition des entités impactées par la suppression du concept Destination de l'ontologie donnée en annexe B.1. Les entités en couleur bleue représentent les entités directement impactées par l'opération et celles en couleur verte constituent les entités indirectement touchées par la suppression du concept.

Ainsi, l'utilisateur dispose d'un outil de suivi des impacts de changements lui permettant d'avoir une vue sur la sévérité de l'opération sur l'ontologie et d'élaborer une stratégie de résolution des impacts.

FIGURE 6.9 – Interface de mesure d'inconsistances

FIGURE 6.10 – Interface des résultats de la mesure d'inconsistances

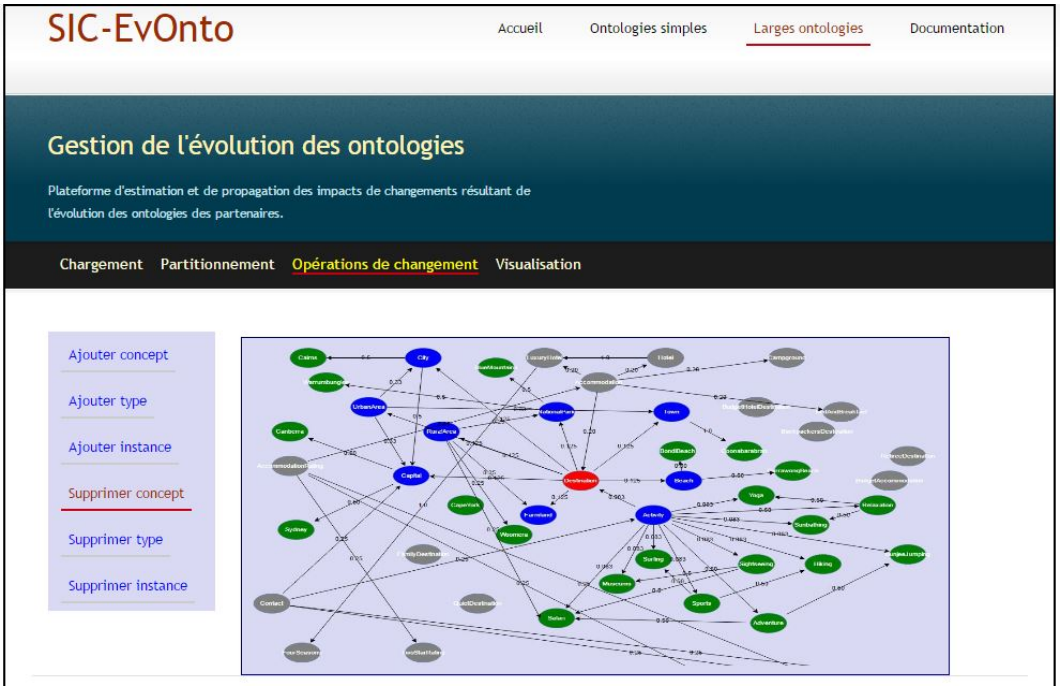

FIGURE 6.11 – Interface de propagation des inconsistances

6.5 Synthèse

L'implémentation et la validation constituent des phases importantes dans le processus de développement d'une plateforme. Elles permettent de mettre en exergue l'architecture du système et les fonctionnalités assurées par celui-ci. Le prototype de gestion des impacts de changements que nous avons proposé dans cette thèse comprend un module de partitionnement d'ontologies de grande taille, un module de mesure d'inconsistances d'une opération de changement, un module de suivi des effets de changements entre autres. L'architecture de cette plateforme repose sur celle du système d'intégration de données hétérogènes mise en place par le projet SIC-Sénégal, architecture à laquelle nous avons ajouté une couche d'applications pour la gestion de l'évolution des ontologies ainsi développées dans le cadre de ce projet.

Les différentes interfaces montrent le fonctionnement du système qui utilise les bibliothèques de Jena et de mxGraph pour la manipulation des ontologies et la visualisation de celles-ci sous forme de graphe dynamique.

6.6 Conclusion

Dans ce chapitre nous avons présenté l'architecture et les fonctionnalités du prototype SIC-Evonto développé dans le cadre de cette thèse. Nous avons commencé

par décrire l'architecture générale du système puis l'architecture fonctionnelle des différents modules composant le système de gestion des inconsistances. Pour chaque module, nous avons montré, à travers des interfaces de la plateforme, le mode de fonctionnement, et ce sur des exemples.

Les travaux effectués dans cette thèse, l'état de l'art, les contributions et leur validation, ont permis de mettre en place un système de gestion de l'évolution des ontologies volumineuses très utilisées dans le domaine de la santé, de la biologie et surtout de l'agriculture. Ces trois secteurs représentent les domaines d'intervention des différents organismes intervenant dans la zone de la vallée du fleuve Sénégal. Cependant, les résultats apportés peuvent toujours être améliorés pour prendre en compte la gestion de l'évolution des ontologies dans le cadre de l'intégration de données par un système médiateur. Cela ouvre plusieurs perspectives de recherche que nous développons dans le prochain chapitre qui conclut cette thèse.

Chapitre 7

Conclusion et Perspectives

Sommaire

7.1 Résumé des travaux

Les travaux effectués dans cette thèse ont consisté à l'estimation et la propagation des effets de changement résultant de l'évolution d'une ontologie de grande taille. Il s'agissait dans un premier temps de mesurer les inconsistances produites par une modification sur une entité ontologique, puis dans un deuxième temps, d'identifier toutes les entités touchées par ce changement et d'évaluer leur degré d'impact. Dans notre démarche, nous avons procédé en trois étapes.

7.1.1 Formalisation des opérations de changement

La première étape de nos travaux a concerné la formalisation des opérations de changement pouvant intervenir dans une ontologie. Il existe deux types de changement ontologique : les changements élémentaires appelés aussi changements simples ou atomiques et les changements complexes. Un changement simple modifie une entité ontologique (concept, type, instance ou relation) alors qu'un changement complexe représente une séquence ordonnée de changements simples [Sto04]. S'appuyant sur des travaux dans ce domaine, nous avons formalisé une opération de modification qui s'exprime en termes d'assertions. Ces assertions représentent des conditions que l'opération doit vérifier avant, pendant et après l'enclenchement de sa phase d'exécution. Il s'agit des pré-conditions, de l'invariant et des post-conditions. Les pré-conditions représentent les assertions que l'opération de modification doit vérifiée avant que l'exécution ne puisse être effective tandis que les post-conditions, si elles sont vérifiées après l'exécution, permettent de valider l'opération. Les assertions à vérifier aussi bien avant et qu'après l'opération de changement constituent les invariants. La non vérification d'un invariant après l'exécution d'un changement

génère des inconsistances et déclenche un processus de propagation d'impacts sur l'ontologie changeante et sur les objets dépendants.

7.1.2 Partitionnement et mesures d'inconsistances

La deuxième étape des travaux de cette thèse est consacrée à l'estimation des degrés d'impacts ou la mesure d'inconsistances induites par une modification ontologique. Une mesure d'inconsistances donne une idée sur la sévérité des impacts de changement dans l'ontologie et permet de prévoir les actions à mener pour rendre celle-ci consistante. En ce sens, deux approches de mesure ont été proposées. La première approche est une mesure d'inconsistances basée sur le degré de connectivité des nœuds du graphe ontologique. Elle consiste à définir un procédé de transformation d'une ontologie représentée dans un langage quelconque en un graphe simple puis à attribuer un poids de dépendance entre les entités du graphe ontologique selon le type de relation les liant. Pour mesurer l'inconsistance causée par une opération de modification, on détermine l'ensemble des entités présentes dans son invariant et utilise le poids de dépendance.

La seconde approche de mesure permet de quantifier les impacts de changement pour une ontologie de grande taille telle que AGROVOC ou Gene Ontology. La mise à jour de ces types d'ontologie est une tâche complexe du fait du nombre important des entités concernées. L'approche proposée procède à une décomposition de l'ontologie en communautés (groupe d'éléments qui partagent plus de propriétés à l'intérieur du groupe qu'en dehors) et établit les mesures dans les communautés puis dans l'ontologie. L'algorithme de partitionnement trouve la ligne séparatrice en fonction des poids des relations entre les nœuds du graphe et du nombre de partitions à obtenir. Il décompose le graphe en îles désignées par leur port(le nœud le plus proche) puis propose comme candidats les îles dont le nombre d'éléments est supérieur au minimum exigé. La mesure d'impacts d'un changement opéré dans une communauté s'exprime alors à l'aide du poids de connexion entre l'entité concernée et les éléments formant l'invariant de l'opération.

7.1.3 Propagation d'impacts de changements

En dernière étape, un algorithme de marquage des entités impactées par un changement et une méthodologie de répartition quantitative des effets de modification ont été proposés. Ainsi, pour toute opération de changement dans une partition de l'ontologie, on peut déterminer le taux d'impact sur chaque entité de la partition de la même manière qu'on peut déterminer le taux d'impact d'une communauté. En définitive, il est possible d'estimer le degré de vulnérabilité de chaque entité ontologique à un changement devant s'opérer. Le degré de vulnérabilité permet à l'ingénieur de maintenance de l'ontologie d'établir un bilan de risque de tous les composants de l'ontologie.

7.2 Contributions scientifiques

Les travaux dans cette thèse ont apporté des contributions scientifiques remarquables sur trois aspects.

- **Méthodologie de décomposition d'ontologies**
  Nous avons souligné dans la revue littéraire l'existence de plusieurs algorithmes de partitionnement d'ontologies volumineuses. Notre approche de décomposition se distingue par la définition des paramètres de partitionnement notamment le poids de connexion entre les entités du graphe ontologique. Ce poids tient compte des différents types de relations existant dans une ontologie et du sens des flux d'impact de ces relations. L'approche diffère aussi des méthodologies de partitionnement prenant en entrée une ontologie cible dont le rôle est de guider la décomposition. C'est le cas de l'approche que nous trouvons dans [HSZR09] orientée surtout pour l'alignement d'ontologies.
- **Estimation des impacts de changement dans une ontologie volumineuse**
  La principale contribution de cette thèse est l'estimation des impacts de changement d'une ontologie de très grand volume. En effet, l'évolution des ontologies volumineuses n'étaient pas prise en charge dans la littérature dans ses aspects mesure et propagation d'impacts de changement. Les approches que nous avons vues se limitaient aux méthodes de partitionnement ou à l'ajout de nouveaux termes dans l'ontologie sans une étude préalable des conséquences induites. Nous avons proposé une approche de mesure capable d'abord de déterminer les entités directement impactées par la modification, puis de quantifier cette inconsistance. La valeur donnée par cette mesure est facilement interprétable par l'ingénieur en charge de la maintenance de l'ontologie. L'approche est aussi implémentée ce qui n'est pas le cas pour la plupart des mesures d'inconsistances de la littérature.
- **Propagation quantitative des impacts de changement dans une ontologie de grande taille**
  Cette contribution permet de donner une idée précise sur la sévérité d'une modification en suivant la répartition à travers le graphe ontologique. Une mesure d'inconsistances quantifie les impacts, l'approche de propagation quantifie la répartition de ces impacts. Nous avons utilisé des outils mathématiques pour définir les différents estimateurs, ce qui leur confère un caractère scientifiquement démontrable.

7.3 Perspectives

Nous avons découvert, dans le cadre des recherches menées dans cette thèse, des problématiques de recherche pertinentes connexes à nos travaux. La mesure et la propagation des inconsistances dans le cadre d'un système d'intégration à base d'ontologies constitue une perspective intéressante à explorer.
Un système d'intégration de données à base d'ontologies peut être formalisé par un

triplet :

$$I = <G, S, M>$$

où :

- G est le schéma global exprimé dans un langage Lg avec un alphabet Ag,
- S, le schéma source exprimé dans un langage Ls avec un alphabet As,
- M, le mapping entre S et G, constitué d'un ensemble d'assertions.

Suivant le mapping entre le schéma global et les schémas sources, deux approches de médiation se distinguent : le Global-As-View (GAV) et le Local-As-View (LAV) et une troisième approche qui combine les deux. L'approche LAV favorise l'extensibilité du système : ajouter une nouvelle source signifie simplement enrichir le mapping avec une nouvelle assertion, sans d'autres changements. L'approche GAV favorise l'exécution des requêtes dans le système, étant donné qu'elle dit au système comment utiliser les sources pour retrouver les données. Étendre le système avec une nouvelle source de données pose problème : la nouvelle source peut avoir un impact sur la définition des différents éléments du schéma global, auxquels les vues associées doivent être redéfinies. L'approche hybride définit uniquement des correspondances entre les schémas des différentes sources et non le schéma médiateur.

Ainsi, il s'avère pertinent d'étudier l'impact de modification d'une ou de plusieurs ontologies du système d'intégration sur les schémas de mapping. La mesure et la propagation de changements résultant de l'évolution des ontologies du système s'intéressera aux questions suivantes, entre autres :

- Suivant l'approche de médiation (LAV, GAV ou GLAV), comment mesurer l'inconsistance induite par un changement dans une ontologie source sur le système d'intégration ?
- Comment se fait la propagation des impacts de changements dans tout le système d'intégration basé sur les ontologies ?
- Suivant les approches de médiation entre les schémas sources et le schéma global (LAV, GAV, GLAV), comment se fait cette propagation des effets de changements dans tout le système d'intégration basé sur les ontologies ? Quelle est l'approche la mieux adaptée aux mises à jour des ontologies sources ?

Une autre piste de recherche à explorer est la résolution des inconsistances résultant d'un changement dans l'ontologie. Les approches mentionnées dans ce manuscrit apportent des débuts de solution mais peinent surtout dans le cas d'une ontologie volumineuse ou d'un système d'intégration basé sur des ontologies. Nous pensons que la piste des processus de décision markovienne pourrait être prometteuse. En effet, un processus de décision markovien est constitué de :

- un ensemble d'états S,
- un ensemble d'actions A,
- une fonction de gain/récompense r,
- une fonction de transition d'état T.

Dans le cadre de la résolution d'inconsistances, nous pouvons reprendre ce modèle en représentant les entités de l'ontologie comme les états, la mesure du taux de

propagation d'impact peut constituer alors la fonction de gain. Les actions à appliquer seront les opérations de changement additionnelles pour rendre le système consistant.

Annexe A

Codes d'implémentation

A.1 Implémentation de l'algorithme de partitionnement

Création du graphe ontologique

```
public Graphe creerGraphe(String cheminFichier){
    Graphe grapheOnto = new Graphe() ;
    ArrayList <OntClass>concepts = new ArrayList<OntClass>() ;
    ArrayList <OntProperty>objectProperties = new ArrayList<OntProperty>() ;
    ArrayList <OntProperty>dataTypeProperties = new ArrayList<OntProperty>() ;
    ArrayList <Individual>individuals = new ArrayList<Individual>() ;
    BasicConfigurator.configure() ;
    OntModel ontologie = ModelFactory.createOntologyModel() ;
    FileInputStream in = null ;
    try{
        in=new FileInputStream(cheminFichier) ;
    }
    catch (FileNotFoundException ex){
        Logger.getLogger (Main.class.getName (), null).log(null, ex) ;
    }
    ontologie.read(in,null,"RDF/XML-ABBREV") ;
    Iterator iter = ontologie.listClasses() ;
    try{
        while(iter.hasNext()){
            OntClass onto = (OntClass) iter.next() ;
            if(onto1.getLocalName() !=null){
                concepts.add(onto) ;
            }
        }
        for(int i = 0 ; i < concepts.size() ; i++){
            s = s.creerSommet(concepts.get(i).getLocalName()) ;
            grapheOnto.ajouterSommet(s) ;
            iter = concepts.get(i).listSubClasses() ;
            while(iter.hasNext()){
                OntClass concept = (OntClass) iter.next() ;
```

```
                        if(concepts.get(i).getLocalName() !=null                    &&
concept.getLocalName() !=null){
                            s = s.creerSommet(concept.getLocalName()) ;
                            d = d.creerSommet(concepts.get(i).getLocalName()) ;
                            grapheOnto.ajouterArc(s, d, 1) ;
                        }
                    }
                }
                iter = ontologie.listObjectProperties() ;
                while(iter.hasNext()){
                    OntProperty onto = (OntProperty) iter.next() ;
                    objectProperties.add(onto) ;
                    if(onto2.getDomain().getLocalName() !=null){
                        Iterator itera = onto.listRange() ;
                        while(itera.hasNext()){
                            OntClass node = (OntClass) itera.next() ;
                            s = s.creerSommet(onto.getDomain().getLocalName()) ;
                            d = d.creerSommet(node.getLocalName()) ;
                            grapheOnto.ajouterArc(s, d, 1) ;
                        }
                    }
                }
                iter = ontologie.listDatatypeProperties() ;
                while(iter.hasNext()){
                    OntProperty onto = (OntProperty) iter.next() ;
                    dataTypeProperties.add(onto) ;
                    grapheOnto.ajouterSommet(onto.getLocalName()) ;
                    if(onto.getDomain().getLocalName() !=null                   &&
onto.getRange().getLocalName() !=null){
                        s = s.creerSommet(onto.getDomain().getLocalName()) ;
                        d = d.creerSommet(onto.getRange().getLocalName()) ;
                        grapheOnto.ajouterArc(s, d, 1) ;
                    }
                }
                iter = ontologie.listIndividuals() ;
                while(iter.hasNext()){
                    Individual onto = (Individual) iter.next() ;
                    individuals.add(onto) ;
                    d = d.creerSommet(onto.getLocalName()) ;
                    grapheOnto.ajouterSommet(d) ;
                    s = s.creerSommet(onto.getOntClass().getLocalName()) ;
                    grapheOnto.ajouterArc(s, d, 1) ;
                }
            }
```

```
        catch (NullPointerException a){
            System.out.println("Erreur : pointeur null") ;
        }
        return grapheOnto ;
    }
```

Détermination des poids de dépendance

```
    public Graphe calculerPoidsDependace(Graphe g){
        for(int i = 0 ; i < g.taille() ; i++){
            for(int j = 0 ; j < g.taille() ; j++){
                double poids = 0 ;
                if(g.existeArc(i, j)){
                    int somme_ik = 0 ;
                    for(int k = 0 ; k < g.taille() ; k++){
                        if(g.existeArc(i, k))
                            somme_ik += 1 ;
                    }
                    poids = (double)1/somme_ik ;
                    g.ajouterArc(g.numeroSommet(i),  g.numeroSommet(j),
poids) ;
                }
            }
        }
        return g ;
    }
```

Division du graphe en îles

```
    public LinkedList<Graphe> creerIles(Graphe g){
        LinkedList<Graphe> sousIles = null ;
        LinkedList<Graphe> iles = g.ordonnerGraphe(g) ;
        Graphe i1 = null, i2 = null, ile = null ;
        for(int i = 0 ; i < g.taille() ; i++){
            for(int j = 0 ; j < g.taille() ; j++){
                for(int k = 0 ; k < iles.size() ; k++){
                    if(iles.get(k).contientSommet(g.numeroSommet(i))){
                        i1 = iles.get(k) ;                              }
                    if(iles.get(k).contientSommet(g.numeroSommet(j))){
                        i2 = iles.get(k) ;
                    }
                }
                if( !i1.equals(i2)){
                    ile.ajouterSommet(g.numeroSommet(i)) ;
                    ile.ajouterSommet(g.numeroSommet(j)) ;
                    ile.setPort(g.numeroSommet(i)) ;
```

```
                        sousIles.add(ile) ;
                    }
                }
            }
            return sousIles ;
    }
```

Détermination des candidats au partitionnement parmi les îles créées

```
    public LinkedList<Graphe> determinerCandidats(LinkedList<Graphe> sousIles, int max){
            LinkedList<Graphe> candidats = null, iles ;
            int min = 1 ;
            // max est égal à la taille du graphe - 1
            int i = 0 ;
            while( !sousIles.isEmpty()){
                Graphe ile = sousIles.get(i) ;
                if(ile.taille()<min){
                    sousIles.remove(i) ;
                }
                else if(ile.taille()>max){
                    iles = this.creerIles(ile) ;
                    Iterator iter = null ;
                    while(iter.hasNext()){
                        sousIles.add((Graphe) iter.next()) ;
                    }
                }
                else{
                    candidats.add(ile) ;
                }
            }
            return candidats ;
    }
```

Extension des candidats aux noeuds isolés

```
    public LinkedList<Graphe> etendreCandidat(LinkedList<Graphe> candidats, Graphe g){
            Graphe communaute = null ;
            boolean trouve = false ;
            double poidsMin = 1 ;
            int indiceCandidat = 0, j = 0 ;
            for(int i = 0 ; i < g.taille() ; i++){
                Iterator iter = null ;
                while(iter.hasNext() && !trouve){
                    Graphe C = (Graphe)iter.next() ;
```

```
                trouve = C.contientSommet(g.Sommet(i)) ;
                if(g.poids(i, C.determinerPort(C)) < poidsMin){
                        poidsMin = g.poids(i, C.determinerPort(C)) ;
                        communaute = C ;
                        indiceCandidat = j ; // on garde l'indice du candidat
vérifiant la condition
                }
                j++ ;
        }
        if( !trouve){
                communaute.ajouterSommet(g.Sommet(i)) ;
                candidats.add(indiceCandidat, communaute) ;
        }
    }
    return candidats ;
}
```

Partitionnement

Pour chaque candidat, on le subdivise en deux sous-ensembles qui seront ajoutés à la liste des communautés et on supprime le candidat, ce jusqu'Ã avoir le nombre de communautés souhaitées.

```
public LinkedList<Graphe> partitionner(LinkedList<Graphe> candidats, int
maxCandidats){
    Graphe C1 = null, C2 = null ;
    int j = 0 ;
    Iterator iter = null ;
    while(iter.hasNext()){
        if(candidats.size() < maxCandidats){
            Graphe C = (Graphe)iter.next() ;
            C1.ajouterSommet(C.Sommet(0)) ;
            C2.ajouterSommet(C.Sommet(C.taille()-1)) ;
            for(int i = 1 ; i < C.taille()-1 ; i++){
                if(C1.poids(C1.determinerPort(C1),          i)          <
C2.poids(C2.determinerPort(C2), i)){
                    C1.ajouterSommet(C.Sommet(i)) ;
                }
                else{
                    C2.ajouterSommet(C.Sommet(i)) ;
                }
            }
        }
        candidats.add(C1) ;
        candidats.add(C2) ;
        candidats.remove(j) ;
```

```
            j++;
        }
        return candidats;
    }
```

Annexe B

Ontologie utilisée dans les exemples

B.1 Représentation de "Travel Ontology" en OWL

```
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns
       \#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema\#"
    xmlns:owl="http://www.w3.org/2002/07/owl\#"
    xmlns:daml="http://www.daml.org/2001/03/daml+oil\#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns="http://www.owl-ontologies.com/travel.owl\#"
  xml:base="http://www.owl-ontologies.com/travel.owl">
  <owl:Ontology rdf:about="">
    <owl:versionInfo rdf:datatype="http://www.w3.org/2001/
       XMLSchema\#string">1.0 by Holger Knublauch (
       holger@smi.stanford.edu)</owl:versionInfo>
  </owl:Ontology>
  <owl:Class rdf:ID="Sunbathing">
   <rdfs:subClassOf>
      <owl:Class rdf:about="\#Relaxation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Accommodation">
  </owl:Class>
  <owl:Class rdf:ID="QuietDestination">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:ID="Destination"/>
          <owl:Class>
            <owl:complementOf>
              <owl:Class rdf:about="\#FamilyDestination"/>
            </owl:complementOf>
          </owl:Class>
```

```
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="BackpackersDestination">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="\#Destination"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="\#
               hasAccommodation"/>
          </owl:onProperty>
          <owl:someValuesFrom>
            <owl:Class rdf:about="\#BudgetAccommodation"
               />
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class>
              <owl:unionOf rdf:parseType="Collection">
                <owl:Class rdf:about="\#Sports"/>
                <owl:Class rdf:about="\#Adventure"/>
              </owl:unionOf>
            </owl:Class>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="\#hasActivity
               "/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Sports">
  <owl:disjointWith>
    <owl:Class rdf:about="\#Adventure"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="\#Relaxation"/>
 </owl:disjointWith>
```

```
    <owl:disjointWith>
      <owl:Class rdf:about="\#Sightseeing"/>
    </owl:disjointWith>
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Activity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Yoga">
    <rdfs:subClassOf>
      <owl:Class rdf:about="\#Relaxation"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BudgetAccommodation">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="\#Accommodation"/>
          <owl:Restriction>
            <owl:someValuesFrom>
              <owl:Class>
                <owl:oneOf rdf:parseType="Collection">
                  <AccommodationRating rdf:ID="
                     OneStarRating">
                    <owl:differentFrom>
                      <AccommodationRating rdf:ID="
                         ThreeStarRating">
                        <owl:differentFrom>
                          <AccommodationRating rdf:ID="
                             TwoStarRating">
                            <owl:differentFrom
                               rdf:resource="\#
                               OneStarRating"/>
                            <owl:differentFrom
                               rdf:resource="\#
                               ThreeStarRating"/>
                          </AccommodationRating>
                        </owl:differentFrom>
                        <owl:differentFrom rdf:resource="
                           \#OneStarRating"/>
                      </AccommodationRating>
                    </owl:differentFrom>
                   <owl:differentFrom rdf:resource="\#
                      TwoStarRating"/>
                  </AccommodationRating>
```

```
                <AccommodationRating rdf:about="\#
                   TwoStarRating"/>
                </owl:oneOf>
              </owl:Class>
            </owl:someValuesFrom>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="\#hasRating"/
                 >
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="LuxuryHotel">
    <rdfs:subClassOf>
      <owl:Class rdf:about="\#Hotel"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:resource="\#ThreeStarRating"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="\#hasRating"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="FamilyDestination">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="\#Destination"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="\#
                 hasAccommodation"/>
            </owl:onProperty>
            <owl:minCardinality rdf:datatype="http://www.
               w3.org/2001/XMLSchema\#int">1</
               owl:minCardinality>
          </owl:Restriction>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="\#hasActivity
```

```
                "/>
            </owl:onProperty>
            <owl:minCardinality rdf:datatype="http://www.
               w3.org/2001/XMLSchema\#int">2</
               owl:minCardinality>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="Beach">
    <rdfs:subClassOf rdf:resource="\#Destination"/>
  </owl:Class>
  <owl:Class rdf:ID="Hotel">
    <owl:disjointWith>
      <owl:Class rdf:about="\#BedAndBreakfast"/>
    </owl:disjointWith>
    <owl:disjointWith>
      <owl:Class rdf:about="\#Campground"/>
    </owl:disjointWith>
    <rdfs:subClassOf rdf:resource="\#Accommodation"/>
  </owl:Class>
  <owl:Class rdf:ID="Museums">
    <rdfs:subClassOf>
      <owl:Class rdf:about="\#Sightseeing"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BudgetHotelDestination">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="\#Destination"/>
          <owl:Restriction>
            <owl:someValuesFrom>
              <owl:Class>
                <owl:intersectionOf rdf:parseType="
                   Collection">
                  <owl:Class rdf:about="\#
                     BudgetAccommodation"/>
                  <owl:Class rdf:about="\#Hotel"/>
                </owl:intersectionOf>
              </owl:Class>
            </owl:someValuesFrom>
            <owl:onProperty>
```

```
            <owl:ObjectProperty rdf:about="\#
               hasAccommodation"/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="City">
  <rdfs:subClassOf>
    <owl:Class rdf:about="\#UrbanArea"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="\#
           hasAccommodation"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="\#LuxuryHotel"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="AccommodationRating">
  <owl:equivalentClass>
    <owl:Class>
      <owl:oneOf rdf:parseType="Collection">
        <AccommodationRating rdf:about="\#OneStarRating"
           />
        <AccommodationRating rdf:about="\#TwoStarRating"
           />
        <AccommodationRating rdf:about="\#
           ThreeStarRating"/>
      </owl:oneOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="BedAndBreakfast">
  <owl:disjointWith rdf:resource="\#Hotel"/>
  <rdfs:subClassOf rdf:resource="\#Accommodation"/>
  <owl:disjointWith>
    <owl:Class rdf:about="\#Campground"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="Campground">
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:hasValue rdf:resource="\#OneStarRating"/>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="\#hasRating"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <owl:disjointWith rdf:resource="\#BedAndBreakfast"/>
    <rdfs:subClassOf rdf:resource="\#Accommodation"/>
    <owl:disjointWith rdf:resource="\#Hotel"/>
  </owl:Class>
  <owl:Class rdf:ID="Safari">
    <rdfs:subClassOf>
      <owl:Class rdf:about="\#Adventure"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class rdf:about="\#Sightseeing"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="RuralArea">
    <rdfs:subClassOf rdf:resource="\#Destination"/>
    <owl:disjointWith>
      <owl:Class rdf:about="\#UrbanArea"/>
    </owl:disjointWith>
  </owl:Class>
  <owl:Class rdf:ID="RetireeDestination">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="\#Destination"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="\#
                 hasAccommodation"/>
            </owl:onProperty>
            <owl:someValuesFrom>
              <owl:Restriction>
                <owl:hasValue rdf:resource="\#
                   ThreeStarRating"/>
                <owl:onProperty>
                  <owl:ObjectProperty rdf:about="\#
                     hasRating"/>
                </owl:onProperty>
```

```
            </owl:Restriction>
          </owl:someValuesFrom>
        </owl:Restriction>
        <owl:Restriction>
          <owl:someValuesFrom>
            <owl:Class rdf:about="\#Sightseeing"/>
          </owl:someValuesFrom>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="\#hasActivity
               "/>
          </owl:onProperty>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Relaxation">
  <owl:disjointWith rdf:resource="\#Sports"/>
  <owl:disjointWith>
    <owl:Class rdf:about="\#Sightseeing"/>
  </owl:disjointWith>
  <owl:disjointWith>
    <owl:Class rdf:about="\#Adventure"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="\#Activity"/>
</owl:Class>
<owl:Class rdf:ID="Capital">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="\#Museums"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="\#hasActivity"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="\#City"/>
</owl:Class>
<owl:Class rdf:ID="Hiking">
  <rdfs:subClassOf rdf:resource="\#Sports"/>
</owl:Class>
<owl:Class rdf:ID="UrbanArea">
  <owl:disjointWith rdf:resource="\#RuralArea"/>
  <rdfs:subClassOf rdf:resource="\#Destination"/>
</owl:Class>
```

```
<owl:Class rdf:ID="BunjeeJumping">
  <rdfs:subClassOf>
    <owl:Class rdf:about="\#Adventure"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Adventure">
  <owl:disjointWith rdf:resource="\#Sports"/>
  <rdfs:subClassOf rdf:resource="\#Activity"/>
  <owl:disjointWith>
    <owl:Class rdf:about="\#Sightseeing"/>
  </owl:disjointWith>
  <owl:disjointWith rdf:resource="\#Relaxation"/>
</owl:Class>
<owl:Class rdf:ID="Contact"/>
<owl:Class rdf:ID="NationalPark">
  <rdfs:subClassOf rdf:resource="\#RuralArea"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="\#Campground"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="\#
           hasAccommodation"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="\#Hiking"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="\#hasActivity"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Town">
  <rdfs:subClassOf rdf:resource="\#UrbanArea"/>
</owl:Class>
<owl:Class rdf:ID="Sightseeing">
  <owl:disjointWith rdf:resource="\#Sports"/>
  <rdfs:subClassOf rdf:resource="\#Activity"/>
  <owl:disjointWith rdf:resource="\#Relaxation"/>
  <owl:disjointWith rdf:resource="\#Adventure"/>
</owl:Class>
<owl:Class rdf:ID="Farmland">
```

```
  <rdfs:subClassOf rdf:resource="\#RuralArea"/>
</owl:Class>
<owl:Class rdf:ID="Surfing">
  <rdfs:subClassOf rdf:resource="\#Sports"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="isOfferedAt">
  <rdfs:range rdf:resource="\#Destination"/>
  <rdfs:domain rdf:resource="\#Activity"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="\#hasActivity"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasRating">
  <rdfs:range rdf:resource="\#AccommodationRating"/>
  <rdfs:domain rdf:resource="\#Accommodation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasActivity">
  <owl:inverseOf rdf:resource="\#isOfferedAt"/>
  <rdfs:range rdf:resource="\#Activity"/>
  <rdfs:domain rdf:resource="\#Destination"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasContact">
  <rdfs:range rdf:resource="\#Contact"/>
  <rdfs:domain rdf:resource="\#Activity"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasAccommodation">
  <rdfs:range rdf:resource="\#Accommodation"/>
  <rdfs:domain rdf:resource="\#Destination"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasPart">
  <rdfs:range rdf:resource="\#Destination"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl
     \#TransitiveProperty"/>
  <rdfs:domain rdf:resource="\#Destination"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="hasCity">
  <rdfs:domain rdf:resource="\#Contact"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl
     \#FunctionalProperty"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/
     XMLSchema\#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="hasEMail">
  <rdfs:domain rdf:resource="\#Contact"/>
```

```
    <rdfs:range rdf:resource="http://www.w3.org/2001/
       XMLSchema\#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl
       \#FunctionalProperty"/>
  </owl:DatatypeProperty>
  <owl:FunctionalProperty rdf:ID="hasZipCode">
    <rdfs:range rdf:resource="http://www.w3.org/2001/
       XMLSchema\#int"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl
       \#DatatypeProperty"/>
    <rdfs:domain rdf:resource="\#Contact"/>
  </owl:FunctionalProperty>
  <owl:FunctionalProperty rdf:ID="hasStreet">
    <rdfs:domain rdf:resource="\#Contact"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/
       XMLSchema\#string"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl
       \#DatatypeProperty"/>
  </owl:FunctionalProperty>
  <RuralArea rdf:ID="Woomera"/>
  <Town rdf:ID="Coonabarabran"/>
  <LuxuryHotel rdf:ID="FourSeasons"/>
  <NationalPark rdf:ID="BlueMountains"/>
  <Capital rdf:ID="Canberra"/>
  <Beach rdf:ID="BondiBeach"/>
  <Beach rdf:ID="CurrawongBeach"/>
  <NationalPark rdf:ID="Warrumbungles"/>
  <RuralArea rdf:ID="CapeYork"/>
  <Capital rdf:ID="Sydney">
    <hasAccommodation rdf:resource="\#FourSeasons"/>
    <hasPart rdf:resource="\#BondiBeach"/>
    <hasPart rdf:resource="\#CurrawongBeach"/>
  </Capital>
  <City rdf:ID="Cairns"/>
</rdf:RDF>
```

< !– Ontologie créée avec Protege http ://protege.stanford.edu –>

Bibliographie

[AA11] HK Asif and Syed Muhammad Ahsan. A critical analysis of the existing ontology evolution approaches. *Journal of American Science*, 7(7) :584–588, 2011. (Cité en page 29.)

[AB93] Robert S Arnold and Shawn A Bohner. Impact analysis-towards a framework for comparison. In *ICSM*, volume 93, pages 292–301, 1993. (Cité en page 41.)

[ABB+00] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology : tool for the unification of biology. *Nature genetics*, 25(1) :25–29, 2000. (Cité en page 71.)

[ADB09] Bali Ahmed, Razika Driouche, and Zizette Boufaida. Détection et représentation des relations entre versions d'ontologie. 2009. (Cité en page 29.)

[Aji95] Samuel Adesoye Ajila. *Maintenance logicielle : Analyse d'impact, problématique et mise en œuvre*. PhD thesis, PhD thesis, CRIN-Universitg Henri Poincarf Nancy 1, 1995. (Cité en page 41.)

[AJP12] Yalemisew M Abgaz, Muhammad Javed, and Claus Pahl. Analyzing impacts of change operations in evolving ontologies. 2012. (Cité en page 42.)

[AMB15] Soraya Setti Ahmed, Mimoun Malki, and Sidi Mohamed Benslimane. Ontology partitioning : Clustering based approach. *International Journal of Information Technology and Computer Science (IJITCS)*, 2015. (Cité en page 5.)

[Bat93] Vladimir Batagelj. Analysis of large networks islands. In *Algorithmic Aspects of Large and Complex Networks*, 1993. (Cité en page 72.)

[BB05] Haider Bilal and Sue Black. Using the ripple effect to measure software quality. In *SOFTWARE QUALITY MANAGEMENT-INTERNATIONAL CONFERENCE-*, volume 13, page 183, 2005. (Cité en page 41.)

[BCB05] A. Buccella, R. Cechich, and N. R. Brisaboa. Ontology-based data integration methods : A framework for comparison. *Revista Colombiana de Computación*, 2005. (Cité en page 3.)

[BDI04] Equipe BDISIC. Projet sic-web sénégal, compte rendu du workshop des 10 et 11 juin. Technical report, Université Gaston Berger de Saint-Louis du Sénégal, 2004. (Cité en page 1.)

[BDSC13] Bruno Berstel-Da Silva and Amina Chniti. Detection of inconsistencies in rules due to changes in ontologies : Let's get formal. In *International Conference on Web Reasoning and Rule Systems*, pages 198–203. Springer, 2013. (Cité en page 58.)

[BG99] Dan Brickley and R Guha. Resource description framework (rdf) model and syntax specification. *W3C Recommendation submitted*, 22, 1999. (Cité en page 18.)

[BG04] Dan Brickley and Ramanathan V Guha. {RDF vocabulary description language 1.0 : RDF schema}. 2004. (Cité en page 18.)

[BHGS01] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. Oiled : a reason-able ontology editor for the semantic web. In *Annual Conference on Artificial Intelligence*, pages 396–408. Springer, 2001. (Cité en page 25.)

[BHS05] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning*, pages 228–248. Springer, 2005. (Cité en page 22.)

[BL98a] Tim Berners-Lee. Cool {URIs} don\'t change. 1998. (Cité en page 17.)

[BL98b] Tim Berners-Lee. Notation 3, 1998. (Cité en page 18.)

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5) :28–37, 2001. (Cité en page 11.)

[BM76] John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory with applications*, volume 290. Citeseer, 1976. (Cité en page 71.)

[BN03] Franz Baader and Werner Nutt. Basic description logics. In *Description logic handbook*, pages 43–95, 2003. (Cité en page 19.)

[Bor97] Willem Nico Borst. *Construction of engineering ontologies for knowledge sharing and reuse*. Universiteit Twente, 1997. (Cité en page 12.)

[BQL07] David Bell, Guilin Qi, and Weiru Liu. Approaches to inconsistency handling in description-logic based ontologies. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 1303–1311. Springer, 2007. (Cité en page 58.)

[Cam13] Gaoussou Camara. *Conception d'un système de veille épidémiologique à base d'ontologies : application à la schistosomiase au Sénégal*. PhD thesis, Paris 6, Université Gaston Berger de Saint-Louis, 2013. (Cité en page 14.)

[CDGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on xml documents : A description logic approach. *Journal of Logic and Computation*, 9(3) :295–318, 1999. (Cité en page 22.)

[CFF+98] Vinay K Chaudhri, Adam Farquhar, Richard Fikes, Peter D Karp, and James P Rice. Okbc : A programmatic foundation for knowledge base interoperability. In *AAAI/IAAI*, pages 600–607, 1998. (Cité en page 25.)

[CFM06] Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching ontologies in open networked systems : Techniques and applications. In *Journal on Data Semantics V*, pages 25–63. Springer, 2006. (Cité en page 27.)

[CM08] Chuming Chen and Manton M Matthews. A new approach to managing the evolution of owl ontologies. In *SWWS*, pages 57–63, 2008. (Cité en page 29.)

[CSG+05] Werner Ceusters, Barry Smith, Louis Goldberg, et al. A terminological and ontological analysis of the nci thesaurus. *Methods of information in medicine*, 44(4) :498, 2005. (Cité en page 5.)

[DA09] Rim Djedidi and Marie-Aude Aufaure. Patrons de gestion de changements owl. In *Ingénierie des Connaissances*, pages 145–156, 2009. (Cité en pages 4 et 29.)

[DA10] Rim Djedidi and Marie-Aude Aufaure. Onto-evoal an ontology evolution approach guided by pattern modeling and quality evaluation. In *International Symposium on Foundations of Information and Knowledge Systems*, pages 286–305. Springer, 2010. (Cité en page 29.)

[DAA07] Rim Djedidi, Marie-Aude Aufaure, and Hassane Abboute. Evolution d'ontologie : Validation des changements basée sur l'évaluation. In *JFO 2007*, pages 55–73, 2007. (Cité en page 4.)

[DAB+93] P.S. Diouf, J.J. Albaret, T. Bousso, A. Diallo, P. Diallo, Deme-Gningue, L. Le Reste, and M. Kebe. Aménagement de la vallée du fleuve sénégal,projet d'étude du "système pêche". Technical report, 1993. (Cité en page 1.)

[Der01] L. Deruelle. *Analyse d'impact de l'évolution des applications distribuées multi-langages et à bases de données hétérogènes*. PhD thesis, Université du Littoral Côte d'Opale, 2001. (Cité en page 91.)

[DHS07] Xi Deng, Volker Haarslev, and Nematollaah Shiri. Measuring inconsistencies in ontologies. In *European Semantic Web Conference*, pages 326–340. Springer, 2007. (Cité en page 56.)

[Din11] Trong Hiêu Dinh. *Grammaires de graphes et langages formels*. PhD thesis, Université Paris-Est, 2011. (Cité en page 30.)

[dOBdAFG11] Ana Christina de Oliveira Bringuente, Ricardo de Almeida Falbo, and Giancarlo Guizzardi. Using a foundational ontology for reengineering a software process ontology. *Journal of Information and Data Management*, 2(3) :511, 2011. (Cité en page 14.)

[dSSS09] Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, and Marta Sabou. Criteria and evaluation for ontology modularization techniques. In *Modular ontologies*, pages 67–89. Springer, 2009. (Cité en page 5.)

[DYH+14] Li Dongmei, Lin Youfang, Huang Houkuan, Hao Shudong, and Wang Jianxin. Dempster-shafer inconsistency values. *Chinese Journal of Electronics*, 23(2), April 2014. (Cité en page 57.)

[ERDN10] Kobra Etminani, Amin Rezaeian Delui, and Mahmoud Naghibzadeh. Overlapped ontology partitioning based on semantic similarity measures. In *5th International Symposium on Telecommunications (IST)*, 2010. (Cité en page 5.)

[Fay07] David Célestin Faye. *Médiation de données sémantique dans SenPeer, un système pair-à-pair de gestion de données*. PhD thesis, Université de Nantes, 2007. (Cité en pages 2 et 6.)

[FF99] Richard Fikes and Adam Farquhar. Distributed repositories of highly expressive reusable ontologies. *IEEE Intelligent systems*, 14(2) :73–79, 1999. (Cité en page 12.)

[FGM00] Enrico Franconi, Fabio Grandi, and Federica Mandreoli. A semantic approach for schema evolution and versioning in object-oriented databases. In *Computational Logic-CL 2000*, pages 1048–1062. Springer, 2000. (Cité en page 29.)

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2) :298–305, 1973. (Cité en page 73.)

[For10] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3) :75–174, 2010. (Cité en page 71.)

[FPA06] Giorgos Flouris, Dimitris Plexousakis, and Grigoris Antoniou. A classification of ontology change. In *SWAP*, 2006. (Cité en page 26.)

[FT06] Frédéric Fürst and Francky Trichet. Raisonner sur des ontologies lourdes à l'aide de graphes conceptuels. In *INFORSID*, pages 879–894, 2006. (Cité en page 30.)

[G+93] Thomas R Gruber et al. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2) :199–220, 1993. (Cité en page 12.)

[Gag07] Michel Gagnon. Logique descriptive et owl. *Cours dispensé à l'école polytechnique de Montréal, Canada*, 2007. (Cité en page 22.)

[GG95] Pierdaniele Giaretta and N Guarino. Ontologies and knowledge bases towards a terminological clarification. *Towards very large knowledge bases : knowledge building & knowledge sharing*, 25 :32, 1995. (Cité en page 12.)

[GH06] John Grant and Anthony Hunter. Measuring inconsistency in knowledgebases. *Journal of Intelligent Information Systems*, 27(2) :159–184, 2006. (Cité en page 56.)

[GH11] John Grant and Anthony Hunter. Measuring consistency gain and information loss in stepwise inconsistency resolution. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 362–373. Springer, 2011. (Cité en pages 45, 52 et 57.)

[GH13] John Grant and Anthony Hunter. Distance-based measures of inconsistency. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 230–241. Springer, 2013. (Cité en pages 46 et 56.)

[GKvdB08] Arda Göknil, Ivan Kurtev, and KG van den Berg. Change impact analysis based on formalization of trace relations for requirements. 2008. (Cité en page 42.)

[GLN08] Fabien Gandon, Moussa Lo, and Cheikh Niang. Un modèle d'index pour la résolution distribuée de requêtes sur un nombre restreint de bases d'annotations rdf. In *19es Journées Francophones d'Ingénierie des Connaissances (IC 2008)*, pages 25–35, 2008. (Cité en page 106.)

[GP99] Asunción Gómez-Pérez. Ontological engineering : A state of the art. *Expert Update : Knowledge Based Systems and Applied Artificial Intelligence*, 2(3) :33–43, 1999. (Cité en page 13.)

[GR93] G. Grahne and K. J. Räihä. Database decomposition into fourth normal form. In *Proceedings of the 9th International Conference on Very Large DataBases*, 1993. (Cité en page 72.)

[GSV04] Thomas Gabel, York Sure, and Johanna Voelker. D3. 1.1. a : Kaon–ontology management infrastructure. *SEKT informal deliverable*, 2004. (Cité en page 30.)

[Gua98] Nicola Guarino. Formal ontology and information systems. In *Proceedings of FOIS*, volume 98, pages 81–97, 1998. (Cité en page 12.)

[Gui06] Giancarlo Guizzardi. The role of foundational ontologies for conceptual modeling and domain ontology representation. In *2006 7th International Baltic Conference on Databases and Information Systems*, pages 17–25. IEEE, 2006. (Cité en page 14.)

[Han72] Frederick M Haney. Module connection analysis : a tool for scheduling software debugging activities. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part I*, pages 173–179. ACM, 1972. (Cité en pages 41 et 42.)

[Has09] Mohamed Oussama Hassan. Contribution à l'analyse d'impact des modifications des architectures logicielles. Université du Littoral Côte d'Opale, 2009. (Cité en page 41.)

[HCD+09] Maja Hadzic, Elizabeth Chang, Tharam Dillon, Janusz Kacprzyk, and Pornpit Wongthongtham. *Ontology-based multi-agent systems*. Springer, 2009. (Cité en page 13.)

[HK05] Anthony Hunter and Sébastien Konieczny. Approaches to measuring inconsistent information. In *Inconsistency tolerance*, pages 191–236. Springer, 2005. (Cité en pages 44 et 56.)

[HK+08] Anthony Hunter, Sébastien Konieczny, et al. Measuring inconsistency through minimal inconsistent sets. *KR*, 8 :358–366, 2008. (Cité en pages 46, 47, 48 et 56.)

[HK10] Anthony Hunter and Sébastien Konieczny. On the measure of conflicts : Shapley inconsistency values. *Artificial Intelligence*, 174(14) :1007–1026, 2010. (Cité en pages 45, 56 et 82.)

[HM01] Volker Haarslev and Ralf Möller. High performance reasoning with very large knowledge bases : A practical case study. In *IJCAI*, volume 1, pages 161–168, 2001. (Cité en page 58.)

[HM04] Patrick Hayes and Brian McBride. Rdf semantics. w3c recommendation, 10 february 2004. *World Wide Web Consortium*, 2004. (Cité en page 17.)

[HN13] Michel Héon and Roger Nkambou. G-owl : Vers un langage de modélisation graphique, polymorphique et typé pour la construction d'une ontologie dans la notation owl. In *IC-24èmes Journées francophones d'Ingénierie des Connaissances*, 2013. (Cité en pages v, 21 et 22.)

[Hor98] Ian Horrocks. The fact system. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 307–312. Springer, 1998. (Cité en page 58.)

[HS05] Peter Haase and Ljiljana Stojanovic. Consistent evolution of owl ontologies. In *European Semantic Web Conference*, pages 182–197. Springer, 2005. (Cité en pages 43, 57 et 58.)

[HSZR09] Fayçal Hamdi, Brigitte Safar, Haïfa Zargayouna, and Chantal Reynaud. Partitionnement d'ontologies pour le passage à l'échelle des techniques d'alignement. In *9eme Journées Francophones" Extraction et Gestion des Connaissances"*. Cepadues, 2009. (Cité en pages 72, 73 et 121.)

[Hun02] Anthony Hunter. Measuring inconsistency in knowledge via quasi-classical models. In *AAAI/IAAI*, pages 68–73, 2002. (Cité en pages 44 et 56.)

[HZQ06] Wei Hu, Yuanyuan Zhao, and Yuzhong Qu. Partition-based block matching of large class hierarchies. In *Asian Semantic Web Conference*, pages 72–83. Springer, 2006. (Cité en page 72.)

[JB08] Christophe Jouis and Julien Bourdaillet. Representation of atypical entities in ontologies. In *LREC*, 2008. (Cité en page 30.)

[JF07] AD Jepson and DJ Fleet. Image segmentation. *University of Toronto [Online], Available : http ://www. cs. toronto. edu/˜ jepson/csc2503/segmentation. pdf*, 2007. (Cité en page 73.)

[JR13] Said Jabbour and Badran Raddaoui. Measuring inconsistency through minimal proofs. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 290–301. Springer, 2013. (Cité en page 46.)

[KBA00] G Kassel, C Barry, and MH ABEL. Programmer au niveau connaissance en def-*. *Ingénierie des Connaissances, évolutions récentes et nouveaux défis, Eyrolle*, pages 145–160, 2000. (Cité en page 15.)

[KC06] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf) : Concepts and abstract syntax. 2006. (Cité en page 17.)

[KEEC05] R Keller, T Eger, CM Eckert, and PJ Clarkson. Visualising change propagation. In *DS 35 : Proceedings ICED 05, the 15th International Conference on Engineering Design, Melbourne, Australia, 15.-18.08. 2005*, 2005. (Cité en page 42.)

[KF01] Michel CA Klein and Dieter Fensel. Ontology versioning on the semantic web. In *SWWS*, pages 75–91, 2001. (Cité en pages 3, 27 et 28.)

[KFKO02a] Michel Klein, Dieter Fensel, Atanas Kiryakov, and D Ognyanov. Ontoview : Comparing and versioning ontologies. *Collected Posters ISWC 2002*, 2002. (Cité en pages 28 et 31.)

[KFKO02b] Michel Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology versioning and change detection on the web. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 197–212. Springer, 2002. (Cité en pages 27 et 36.)

[KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 49(2) :291–307, 1970. (Cité en page 73.)

[Kle04] Michel Christiaan Alexander Klein. *Change management for distributed ontologies*. 2004. (Cité en pages 3, 27, 36, 37 et 39.)

[KLGE07] Natalya Keberle, Yuriy Litvinenko, Yuriy Gordeyev, and Vadim Ermolayev. Ontology evolution analysis with owl-met. In *Proceedings of the International Workshop on Ontology Dynamics (IWOD-07)*, pages 1–12, 2007. (Cité en page 30.)

[KLM03] Sebastien Konieczny, Jerome Lang, and Pierre Marquis. Quantifying information and contradiction in propositional logic through test actions. In *IJCAI*, pages 106–111. Citeseer, 2003. (Cité en page 56.)

[KN03] Michel Klein and Natalya F Noy. A component-based framework for ontology evolution. In *Workshop on Ontologies and Distributed Systems at IJCAI*, volume 3, 2003. (Cité en pages 4, 29 et 39.)

[Kni02] Kevin Knight. Measuring inconsistency. *Journal of Philosophical Logic*, 31(1) :77–98, 2002. (Cité en pages 54, 55 et 56.)

[KPLL10] Asad Masood Khattak, Zeeshan Pervez, Sungyoung Lee, and Young-Koo Lee. After effects of ontology evolution. In *2010 5th International Conference on Future Information Technology*, pages 1–6. IEEE, 2010. (Cité en page 4.)

[Liu06] Weiru Liu. A revision-based approach for handling inconsistency in description logic. *Artificial Intelligence Review*, 26 :115–128, 2006. (Cité en page 58.)

[LJL+13] Lei Liu, Xiang Ji, Libin Li, Shuai Lü, and Rui Zhang. A method for ontology change management based on belief revision. *International Journal of Digital Content Technology and its Applications*, 7(5) :812, 2013. (Cité en pages 30 et 58.)

[LNHE14] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Vowl 2 : user-oriented visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 266–281. Springer, 2014. (Cité en page 21.)

[LO96] Li Li and A Jefferson Offutt. Algorithmic analysis of the impact of changes to object-oriented software. In *Software Maintenance 1996, Proceedings., International Conference on*, pages 171–184. IEEE, 1996. (Cité en page 41.)

[Lo02] Moussa Lo. *DataWeb Bases sur XML. : Modélisation et recherche d'informations pertinentes.* PhD thesis, Université de Pau et Des Pays de l'Adour, 2002. (Cité en pages 2 et 6.)

[LP08] Anh Le Pham. *De l'optimisation à la décomposition de l'ontologique dans la logique de description.* PhD thesis, Université Nice Sophia Antipolis, 2008. (Cité en page 73.)

[LPLTS07] Thi Anh Le Pham, Nhan Le-Thanh, and Peter Sander. Some approaches of ontology decomposition in description logics. In *Complex Systems Concurrent Engineering*, pages 537–546. Springer, 2007. (Cité en pages 72 et 73.)

[LRC02] Philippe Laublet, Chantal Reynaud, and Jean Charlet. Sur quelques aspects du web sémantique. *Assises du GDR I*, 3 :59–78, 2002. (Cité en page 11.)

[LS+98] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. 1998. (Cité en page 17.)

[LSH+02] Suzanna E Lewis, SMJ Searle, N Harris, M Gibson, V Iyer, J Richter, C Wiel, L Bayraktaroglu, E Birney, MA Crosby, et al. Apollo : a sequence annotation editor. *Genome biology*, 3(12) :1, 2002. (Cité en page 25.)

[LTZT12] Kun Liu, Shengqun Tang, Liang Zhang, and Hao Tian. A method based on rdb for detecting changes between multi-version ontologies. *J Comput Inform Syst*, 8 :3293–3300, 2012. (Cité en page 29.)

[MH10] Yue Ma and Pascal Hitzler. Distance-based measures of inconsistency and incoherency for description logics. In *The 23rd International Workshop on Description Logics (DL 2010)*, pages 475–485, 2010. (Cité en pages 44, 45 et 56.)

[MLJ11] Kedian Mu, Weiru Liu, and Zhi Jin. A general framework for measuring inconsistency through minimal inconsistent sets. *Knowledge and Information Systems*, 27(1) :85–114, 2011. (Cité en pages 49, 50 et 56.)

[MLM12] Jianbing Ma, Weiru Liu, and Paul Miller. A characteristic function approach to inconsistency measures for knowledge bases. In *International Conference on Scalable Uncertainty Management*, pages 473–485. Springer, 2012. (Cité en page 41.)

[MMS+02] Alexander Maedche, Boris Motik, Ljiljana Stojanovic, Rudi Studer, and Raphael Volz. Managing multiple ontologies and ontology evolution in ontologging. In *Intelligent Information Processing*, pages 51–63. Springer, 2002. (Cité en page 44.)

[MPSP+09] Boris Motik, Peter F Patel-Schneider, Bijan Parsia, Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, et al. Owl 2 web ontology language : Structural specification and functional-style syntax. *W3C recommendation*, 27(65) :159, 2009. (Cité en page 20.)

[MTFH13] Mariem Mahfoudh, Laurent Thiry, Germain Forestier, and Michel Hassenforder. Adaptation consistante d'ontologie à l'aide des grammaires de graphes. In *IC-24èmes Journées francophones d'Ingénierie des Connaissances*, 2013. (Cité en pages 30 et 58.)

[MVH+04] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10) :2004, 2004. (Cité en page 19.)

[NC04] Jean-Pierre Nakache and Josiane Confais. *Approche pragmatique de la classification : arbres hiérarchiques, partitionnements.* Editions Technip, 2004. (Cité en page 73.)

[NCLM06] Natalya F Noy, Abhita Chugh, William Liu, and Mark A Musen. A framework for ontology evolution in collaborative environments. In *International semantic web conference*, pages 544–558. Springer, 2006. (Cité en page 4.)

[NFF+91] Robert Neches, Richard E Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William R Swartout. Enabling technology for knowledge sharing. *AI magazine*, 12(3) :36, 1991. (Cité en page 12.)

[Nia13] Cheikh Ahmed Tidiane Niang. *Vers plus d'automatisation dans la construction de systèmes mediateurs pour le web semantique : une*

application des logiques de description. PhD thesis, Univetsité de Tours, Université Gsaton Berger de Saint-Louis, 2013. (Cité en pages 2, 3, 6 et 22.)

[NK04] Natalya F Noy and Michel Klein. Ontology evolution : Not the same as schema evolution. *Knowledge and information systems*, 6(4) :428–440, 2004. (Cité en pages 3 et 29.)

[NL13] Stefan Negru and Steffen Lohmann. A visual notation for the integrated representation of owl ontologies. In *WEBIST*, pages 308–315, 2013. (Cité en page 21.)

[NM+02] Natalya Fridman Noy, Mark A Musen, et al. Promptdiff : A fixed-point algorithm for comparing ontology versions. *AAAI/IAAI*, 2002 :744–750, 2002. (Cité en pages 29 et 32.)

[PB90] Shari Lawrence Pfleeger and Shawn A Bohner. A framework for software maintenance metrics. In *Software Maintenance, 1990, Proceedings., Conference on*, pages 320–327. IEEE, 1990. (Cité en page 41.)

[PDBcS97] RSV Pepijn, MJ Dean, TJM Bench-capon, and M Shave. An analysis of ontological mismatches : Heterogeneity versus interoperability. In *Proceedings of AAAI Spring Symposium on Ontological Engineering, Stanford, USA*, 1997. (Cité en page 36.)

[PDT05] Peter Plessers and Olga De Troyer. Ontology change detection using a version log. In *International Semantic Web Conference*, pages 578–592. Springer, 2005. (Cité en pages 3, 4, 29 et 39.)

[PDT06] Peter Plessers and Olga De Troyer. Resolving inconsistencies in evolving ontologies. In *European Semantic Web Conference*, pages 200–214. Springer, 2006. (Cité en page 58.)

[PDTC07] Peter Plessers, Olga De Troyer, and Sven Casteleyn. Understanding ontology evolution : A change detection approach. *Web Semantics : Science, Services and Agents on the World Wide Web*, 5(1) :39–49, 2007. (Cité en page 5.)

[PFF+09] Vicky Papavassiliou, Giorgos Flouris, Irini Fundulaki, Dimitris Kotzinos, and Vassilis Christophides. On detecting high-level changes in rdf/s kbs. In *International Semantic Web Conference*, pages 473–488. Springer, 2009. (Cité en page 27.)

[PJC09] Jyotishman Pathak, Thomas M Johnson, and Christopher G Chute. Survey of modular ontology techniques and their applications in the biomedical domain. *Integrated computer-aided engineering*, 16(3) :225–242, 2009. (Cité en page 5.)

[PTPD09] Ignazio Palmisano, Valentina Tamma, Terry Payne, and Paul Doran. Task oriented evaluation of module extraction techniques. *The Semantic Web-ISWC 2009*, pages 130–145, 2009. (Cité en page 5.)

[QH07] Guilin Qi and Anthony Hunter. Measuring incoherence in description logic-based ontologies. In *Proceedings of the*, 2007. (Cité en pages 45 et 57.)

[Qui68] M. Quillian. Semantic memory. *Semantic Information Processing, MITPress*, 1968. (Cité en page 64.)

[Raj97] Vaclav Rajlich. A model for change propagation based on graph rewriting. In *Software Maintenance, 1997. Proceedings., International Conference on*, pages 84–91. IEEE, 1997. (Cité en pages 43, 89 et 91.)

[Raj00] Václav Rajlich. Modeling software evolution by evolving interoperation graphs. *Annals of Software Engineering*, 9(1-2) :235–248, 2000. (Cité en page 42.)

[Rog08] Codruta Delia Rogozan. Gestion de l'évolution des ontologies : méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions d'ontologie. 2008. (Cité en pages 28, 37 et 39.)

[RPR04] Delia Rogozan, Gilbert Paquette, and Ioan Rosca. Évolution de l'ontologie utilisée comme référentiel sémantique dans un système de téléapprentissage. In *Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et l'Industrie*, pages 243–249. Université de Technologie de Compiègne, 2004. (Cité en page 4.)

[S+76] Glenn Shafer et al. *A mathematical theory of evidence*, volume 1. Princeton university press Princeton, 1976. (Cité en page 30.)

[Sal10] Ousmane Sall. Contribution à la modélisation de données multisources de type dataweb basé sur xml. Université du Littoral, Universié Gaston Berger de Saint-Louis, 2010. (Cité en pages v, 1, 2, 6 et 63.)

[SBF98] Rudi Studer, V Richard Benjamins, and Dieter Fensel. Knowledge engineering : principles and methods. *Data & knowledge engineering*, 25(1) :161–197, 1998. (Cité en pages 13 et 14.)

[Sch02] Guus Schreiber. The web is not well-formed. *IEEE Intelligent Systems*, 17(2) :79–80, 2002. (Cité en page 19.)

[SCM00] Guus Schreiber, Monica Crubézy, and Mark Musen. A case study in using protégé-2000 as a tool for commonkads. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 33–48. Springer, 2000. (Cité en page 25.)

[SEA+02] York Sure, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. Ontoedit : Collaborative ontology development for the semantic web. In *International Semantic Web Conference*, pages 221–235. Springer, 2002. (Cité en page 24.)

[SJ07] N Sassi and W Jaziri. Types de changements et leurs effets sur l'évolution de l'ontologie. *Actes Journées Francophones sur les Ontologies*, 2007. (Cité en page 37.)

[SK04] Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In *International semantic web conference*, pages 289–303. Springer, 2004. (Cité en pages 72 et 73.)

[SK07] Heiner Stuckenschmidt and Michel Klein. Reasoning and change management in modular ontologies. *Data & Knowledge Engineering*, 63(2) :200–223, 2007. (Cité en page 78.)

[SL07] Ousmane Sall and Moussa Lo. Intégration de données environnementales : une approche basée sur les entrepôts de documents xml et les ontologies. In *EDA*, pages 147–160, 2007. (Cité en page 6.)

[SLG+09] Ousmane Sall, Moussa Lo, Fabien Gandon, Cheikh Niang, and Ibrahima Diop. Using xml data integration and ontology reuse to share agricultural data. *International Journal of Metadata, Semantics and Ontologies*, 4(1-2) :93–105, 2009. (Cité en page 3.)

[SMMS02] Ljiljana Stojanovic, Alexander Maedche, Boris Motik, and Nenad Stojanovic. User-driven ontology evolution management. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 285–300. Springer, 2002. (Cité en pages 4, 26, 27, 29, 32, 36, 37 et 39.)

[SMSS03] Ljiljana Stojanovic, Alexander Maedche, Nenad Stojanovic, and Rudi Studer. Ontology evolution as reconfiguration-design problem solving. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 162–171. ACM, 2003. (Cité en page 3.)

[SSFS11] Ansgar Scherp, Carsten Saathoff, Thomas Franz, and Steffen Staab. Designing core ontologies. *Applied Ontology*, 6(3) :177–221, 2011. (Cité en page 14.)

[SSM+04] L. Stojanovic, J. Schneider, A. Maedche, S. Libischer, R. Studer, T. Lumpp, A. Abecker, G. Breiter, and J. Dinger. The role of ontologies in autonomic computing systems. *IBM Systems Journal*, 43(3), 2004. (Cité en page 4.)

[STB+12] Ousmane Sall, Mouhamadou Thiam, Mamadou Bousso, Moussa Lo, and Henri Basson. A model for ripple effects analysis of cascading problems in ontology evolution. *International Journal of Metadata, Semantics and Ontologies 5*, 7(3) :171–184, 2012. (Cité en pages 37, 42 et 91.)

[STBL12] Ousmane Sall, Mouhamadou Thiam, Mamadou Bousso, and Moussa Lo. Using hoare's axiomatic semantics for checking satisfiability of ontology change operations. In *Information Science and Digital Content Technology (ICIDT), 2012 8th International Conference on*, volume 1, pages 61–66. IEEE, 2012. (Cité en pages 4 et 43.)

[STL11] O. Sall, M. Thiam, and M. Lo. Evolution des ontologies : Contribution à la maîtrise des flux de propagation d'impacts. In *CNRIA*, 2011. (Cité en pages 4, 42 et 65.)

[STLB11] O. Sall, M. Thiam, M. Lo, and Henri Basson. A model for ripple effects of ontology evolution based on assertions and ontology reverse engineering. In *Proceedings of MTSR*, pages 155–162. Springer-Verlag Berlin Heidelberg, 2011. (Cité en pages 4, 42 et 65.)

[Sto04] L. Stojanovic. *Methods and Tools for Ontology Evolution*. PhD thesis, Université de Karlsruhe, 2004. (Cité en pages 4, 26, 36, 37, 39, 44, 67 et 119.)

[TB06] Michal Tury and Mária Bieliková. An approach to detection ontology changes. In *Workshop proceedings of the sixth international conference on Web engineering*, page 14. ACM, 2006. (Cité en page 29.)

[Thi12] Matthias Thimm. *Probabilistic reasoning with incomplete and inconsistent beliefs*. AKA, 2012. (Cité en pages 47, 48 et 49.)

[Thi13] Matthias Thimm. Inconsistency measures for probabilistic logics. *Artificial Intelligence*, 197 :1–24, 2013. (Cité en pages 45 et 46.)

[Thi14] Matthias Thimm. Towards large-scale inconsistency measurement. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 195–206. Springer, 2014. (Cité en pages 45, 46 et 57.)

[Thi16a] Matthias Thimm. On the expressivity of inconsistency measures. *Artificial Intelligence*, 234 :120–151, 2016. (Cité en pages 46 et 82.)

[Thi16b] Matthias Thimm. Stream-based inconsistency measurement. *International Journal of Approximate Reasoning*, 68 :68–87, 2016. (Cité en pages 51, 53 et 57.)

[Tic95] Walter F Tichy. *Configuration management*. John Wiley & Sons, Inc., 1995. (Cité en page 42.)

[TM94] Richard J Turver and Malcolm Munro. An early impact analysis technique for software maintenance. *Journal of Software Maintenance : Research and Practice*, 6(1) :35–52, 1994. (Cité en pages 27 et 41.)

[UG96] Mike Uschold and Michael Gruninger. Ontologies : Principles, methods and applications. *The knowledge engineering review*, 11(02) :93–136, 1996. (Cité en page 13.)

[Usc98] Mike Uschold. Knowledge level modelling : concepts and terminology. *The knowledge engineering review*, 13(01) :5–29, 1998. (Cité en page 12.)

[Wei09] Moritz Weiten. Ontostudio® as a ontology engineering environment. In *Semantic knowledge management*, pages 51–60. Springer, 2009. (Cité en page 25.)

[Win02] Eyal Winter. The shapley value. *Handbook of game theory with economic applications*, 3 :2025–2054, 2002. (Cité en page 56.)

[XM12] Guohui Xiao and Yue Ma. Inconsistency measurement based on variables in minimal unsatisfiable subsets. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI'12)*, 2012. (Cité en pages 55 et 57.)

[YFW03] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. *Exploring artificial intelligence in the new millennium*, 8 :236–239, 2003. (Cité en page 96.)

[ZHQ+09] Liping Zhou, Houkuan Huang, Guilin Qi, Yue Ma, Zhisheng Huang, and Youli Qu. Measuring inconsistency in dl-lite ontologies. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 349–356. IEEE Computer Society, 2009. (Cité en page 56.)