HAL
open science

Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique

Ilyes Boukhari

To cite this version:

Ilyes Boukhari. Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique. Autre [cs.OH]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique - Poitiers, 2014. Français. NNT : 2014ESMA0001 . tel-00942081

HAL Id: tel-00942081

https://theses.hal.science/tel-00942081

Submitted on 4 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Laboratoire d'Informatique et d'Automatique pour les Systèmes

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AEROTECHNIQUE

(Diplôme National — Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : INFORMATIQUE ET APPLICATIONS

Présentée par :

Ilyès BOUKHARI

****

Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique

****

Directeurs de Thèse : **Ladjel BELLATRECHE** et **Stéphane JEAN**

Soutenue le 14 Janvier 2014
devant la Commission d'Examen

JURY

Rapporteurs :	**Bernard ESPINASSE**	**Professeur, Université d'Aix-Marseille, Marseille**
	Jérôme GENSEL	**Professeur, LIG, Grenoble**
Examinateurs :	**Yamine AIT AMEUR**	**Professeur, ENSEEIHT, Toulouse**
	Omar BOUSSAID	**Professeur, ERIC, Université Lyon 2**
	Hendrik DECKER	**Professeur, Instituto Tecnológico de Informática, Valencia**
	Dominique MERY	**Professeur, LORIA, Nancy**
	Ladjel BELLATRECHE	**Professeur, ISAE-ENSMA, Poitiers**
	Stéphane JEAN	**Maître de conférences, Université de Poitiers**

Remerciements

Je remercie avant tout, **ALLAH** tout puissant qui m'a donné la volonté et la patience pour accomplir ce travail.

Mes remerciements les plus sincères s'adressent à :

Ladjel BELLATRECHE, mon directeur de thèse, pour m'avoir guidée pendant les trois années de thèse. Je le remercie pour la confiance qu'il m'a témoignée, pour ca disponibilité, pour ses précieuses orientations, pour sa passion pour la recherche et pour son soutien aussi bien sur le plan humain que scientifique;

Stéphane JEAN, mon co-directeur de thèse, pour m'avoir conseiller tout au long de ma thèse;

Yamine AIT AMEUR et **Emmanuel GROLLEAU**, directeurs du LISI et du LIAS, pour m'avoir accueillie au sein du laboratoire;

Jérôme GENSEL et **Bernard ESPINASSE** de m'avoir fait l'honneur d'être rapporteurs de cette thèse et à qui je souhaite exprimer ma profonde reconnaissance;

Dominique MERY, **Omar BOUSSAID**, **Hendrik DECKER** et **Yamine AIT AMEUR** pour avoir accepté d'être membres du jury en tant qu'examinateurs. Je suis très honorée de l'intérêt qu'ils ont porté à mes travaux;

Tout le personnel du LIAS pour leur présence et leur soutien cordial. Je pense en particulier à **Laurent GUITTET**, **Zoe FAGET**, **Michaël BARON**, **Allel HADJALI**, **Frédéric CARREAU** et **Claudine RAULT**.

Tous les membres du LIAS et particulièrement **Ahcene BOUKORCA, Selma BOUARAR, Selma KHOURI, Yassine OUHAMMOU, Amira KERKAD, Chedlia CHAKROUN, Soraya CHACHOUA, Youness BAZHAR, Samia BOULKRINAT,Zahira, Soumia BENKRID, Thomas LACHAUME, etc**;

Mes amis et collègues **Salim, Walid, Tarek, Fateh Ahcene, Messaoud, Lotfi, Mohamed, Yassine, Nourddine, Abdelghani, Zakaria, Zouhir, Nadir, Okba, Moncef, Issam, Lassad, Idir, etc.**;

Enfin et surtout à ma famille et particulièrement à **mes parents, ma femme, mes enfants**, mes deux frères **Billel** et **Zakaria**, mes sœurs **Ahlem, Liela, Hadjer** et **Maroua** pour leur soutien sans faille.Je leur dédie cette thèse.

A mon Algérie bien-aimée. Mes parents,
Ma grand-mère
Ma femme Amina,
Mes enfants (Youcef et Ayoub),
Mes sœurs et mes frères,

Table des matières

Chapitre 1 Introduction générale **1**

1 Contexte et problématique 3

1.1 Nombre croissant de partenaires 4

1.2 Hétérogénéité des vocabulaires 4

1.3 Hétérogénéité des langages de modélisation des besoins 7

1.4 Exigence des relations complexes entre les besoins 7

1.5 Évolution des besoins 7

2 Notre Démarche 8

3 Contributions 9

4 Organisation du mémoire 11

Partie I État de l'art

Chapitre 2 L'expression des besoins : un état de l'art **17**

1 Introduction 20

2 Définitions 21

3 Processus d'ingénierie des Besoins 24
4 Langages de modélisation des besoins 26
4.1 Langages informels (langage naturel) 26
4.2 Langages semi-formels 27
4.3 Langages formels 27
5 Approches et méthodes pour l'expression des besoins 29
5.1 Approches dirigées par les buts 29
5.1.1 GQM : Goal Question Metric 30
5.1.2 KAOS 31
5.1.3 iStar (i*) 33
5.2 Approches à base de scénarii 33
5.3 Approches basées sur les automates et réseaux de Petri 35
6 Problématiques liées à l'expression des besoins 37
7 État des lieux sur l'intégration et l'analyse des besoins 37
7.1 L'intégration des besoins 37
7.1.1 Exemples d'approches dirigées par des méthodes issues de la logique mathématique 37
7.1.2 Approches dirigées par IDM 38
7.2 L'analyse des besoins 40
7.3 Synthèse 41
8 Conclusion 42

Chapitre 3 Les ontologies au service de l'Ingénierie des Besoins **43**

1 Introduction 45
2 Ontologies de domaine 45
2.1 Définitions et caractéristiques 45
2.2 Une taxonomies des ontologies de domaine 46
2.2.1 Les Ontologies Linguistiques (OL) 47
2.2.2 Les Ontologies Conceptuelles (OC) 47

2.3 Les langages de définitions des ontologies 48
2.3.1 Langages orientés gestion et échange de données 48
2.3.2 Langages orientés inférence 50
2.4 Les éditeurs d'ontologies 52
2.5 Représentation formelle d'une ontologie 53
3 Ontologies et l'intégration : application aux sources de données hétérogènes 54
3.1 Intégration sémantique *a posteriori* 54
3.2 Intégration sémantique *a priori* 56
4 Les ontologies dans l'IB : état de l'art 57
4.1 L'analyse des besoins 57
4.2 L'intégration des besoins 60
4.3 Ontologies, besoins et le cycle de vie de conception 60
4.4 Synthèse 61
5 Conclusion 61

Partie II Contributions

Chapitre 4 Unification des vocabulaires et des langages de modélisation des besoins 65
1 Introduction 67
2 Nos hypothèses 68
3 Unification des vocabulaires 69
3.1 Étude de cas 70
3.1.1 Le modèle Orienté buts 70
3.1.2 Le modèle des cas d'utilisation 72
3.1.3 Le Modèle Conceptuel de Traitements (MCT) 73

3.2 Couplage des modèles utilisés avec l'ontologie 76
3.2.1 Méta-modèle de l'ontologie partagée 76
3.2.2 Couplage des trois langages proposés avec les ontologies locales . 77
4 Unification des langages de modélisation des besoins 81
4.1 Proposition d'un modèle pivot des besoins 82
4.2 Couplage de modèle pivot à l'ontologie 83
5 Un exemple d'utilisation de notre méthode d'intégration 85
6 Conclusion 87

Chapitre 5 Vers une Fusion de méta-modèles des langages de modélisation des besoins 89

1 Introduction 91
2 Fusion des méta modèles 91
3 Connexion du méta-modèle générique avec l'ontologie 93
4 Raisonnement sur les besoins 94
4.1 Scénarii de raisonnement 96
4.1.1 Scénario 1 : *ship whole* 96
4.1.2 Scénario 2 : *reason as needed* 97
4.2 Étude de cas 98
4.3 Relations sémantiques entre les besoins : Définition & Formalisation 99
4.4 Évaluation du processus de raisonnement 102
5 Conclusion 103

Chapitre 6 Exploitation des besoins pour la conception physique 105

1 Introduction 107
2 La conception physique 108
3 Conception physique dirigée par les besoins 109
3.1 La sélection des index 110
3.2 La fragmentation horizontale 111
4 Persistance des besoins 111
4.1 Le langage d'exploitation OntoQL 112

4.2 Persistance des besoins 112
4.3 Génération des requêtes SQL 114
5 Expérimentation et évaluation de notre proposition 116
5.1 Structures d'optimisation comme des services 116
5.2 La sélection des index 116
5.3 La fragmentation horizontale 117
6 Conclusion 119

Chapitre 7 Prototype de validation **121**
1 Introduction 123
1.1 Langages et outils utilisés 123
2 Architecture fonctionnelle de OntoReqTool 123
2.1 Les modules de la partie intégration 123
2.2 Les modules de la phase d'exploitation 127
3 Conclusion 132

Chapitre 8 Conclusion générale **133**
1 Conclusion 135
1.1 Bilan des contributions 135
2 Perspectives 136
2.1 Autres langages de modélisation des besoins 136
2.2 Etude du passage à l'échelle de nos propositions 137
2.3 Définition des architectures d'intégration 137
2.4 Prise en compte de l'évolution 137
2.5 Elaboration d'un cycle de vie de conception d'applications avancées 137

Bibliographie **139**

Annexes **149**

Table des figures **155**

Liste des tableaux **159**

Glossaire **161**

Chapitre 1

Introduction générale

Sommaire

1 Contexte et problématique 3
2 Notre Démarche 8
3 Contributions 9
4 Organisation du mémoire 11

1 Contexte et problématique

L'une des conséquences de la mondialisation est la multiplication des entreprises étendues. Une entreprise étendue est caractérisée par un nombre important de sous-entreprises et/ou des sous-traitants. Nous pouvons citer l'exemple de l'entreprise *Airbus* qui implique quatre sites dans différents pays : Hambourg (Allemagne), Filton (Angleterre), Toulouse (France) et Madrid (Espagne). Chaque site possède un nombre important de départements et de services. Pour développer un système ou une application complexe dans une telle entreprise, plusieurs acteurs qui n'utilisent pas le même vocabulaire, voir le même langage, sont impliqués. Pour faciliter leur travail, des solutions d'intégration pourraient être utilisées tout en omettant les barrières fonctionnelles, organisationnelles, technologiques et culturelles. Rappelons qu'un système d'intégration consiste à fournir une interface *unique, uniforme* et *transparente* aux objets concernés par le processus d'intégration. Ces derniers peuvent être des *données*, des *applications*, des *plateformes*, etc. via un schéma global qui peut être localisé dans un médiateur [125] ou un entrepôt de données [82]. Les solutions d'intégration permettent aux entreprises qui les utilisent : **(i)** d'augmenter le pouvoir décisionnel de l'entreprise étendue. Par exemple, plusieurs entreprises ayant développé des entrepôts de données (un cas particulier des systèmes d'intégration de données) ont qualifié ce projet de *succès story*. On peut citer l'exemple de l'entreprise *Coca Cola*, *Wal-mart* [1] et le groupe français Casino [2], **(ii)** de réduire les coût de traitement en partageant les ressources (par exemple, la puissance de calcul), **(iii)** d'identifier les *activités redondantes* qui pourraient être traitées dans certaines entreprises partenaires. Cette identification permet de réduire les coûts de développement ce qui est particulièrement important en période de crise économique. Si nous reprenons notre exemple d'Airbus, pour faire face aux problèmes de retard dans le programme de l'avion A380, cette entreprise a lancé le projet *Power 8* en 2007 qui a permis entre autre d'identifier les *activités redondantes* dans certains services délocalisés, et **(iv)** d'augmenter de la compétitivité de l'entreprise.

Depuis deux décennies, de nombreux travaux académiques liés à l'intégration ont été développés. Ils concernent plusieurs volets : **(i)** l'intégration des données (*Entreprise Information Integration (EII)*), **(ii)** les applications (*Entreprise Application Integration (EAI)*) et **(iii)** les plateformes. Une véritable industrie autour des systèmes d'intégration a été créée [46, 63] et plusieurs entreprises proposent des solutions d'intégration. Parmi les trois volets citées et illustrées sur la figure 1.1, l'intégration des données occupe une place prépondérante. Elle est en effet un enjeu important, comme l'indiquait *Alon Halevy* et al. dans son article intitulé : *Data Integration : The Teenage Years*, qui a eu le prix *10 Year Best Paper Award* à la conférence Very Large Databases (VLDB'2006).

En analysant les dimensions concernées par l'intégration, nous avons identifié l'absence des besoins des utilisateurs. Cependant, un nombre important d'études a montré que les échecs dans la mise en œuvre et l'exploitation des projets informatiques sont souvent dus à une mauvaise compréhension des besoins ainsi qu'à des incohérences, et des ambiguïtés de leurs définitions [1, 4, 57, 87]. Les besoins peuvent également contribuer à l'identification des activités redondantes. Rappelons que plusieurs approches de conception des applications avancées sont censées être *centrées utilisateur*. Cette approche consiste à considérer les utilisateurs et leurs besoins tout au long du processus de développement d'applications informatiques. Parmi les systèmes concernés par cette approche, nous pouvons citer les *base de données*

1. Computer Business Review, October 1996
2. http://www.lsis.org/espinasseb/Supports/DWDM-2013/2-Intro-Entrepots-2013.pdf

FIGURE 1.1 – Différents niveaux d'intégration [50].

[115], *les entrepôts de données* connu sous le nom d'approche orientée besoins [111], *les systèmes d'intégration des données* [54], et les interfaces homme machines [10]. Dans ce travail, nous nous sommes concentrés sur les entrepôts de données.

La plupart des études de conception des systèmes de gestion de données, élaborées selon une approche centrée utilisateur, supposent l'intervention des concepteurs *homogènes*. Par homogène, nous entendons que les concepteurs impliqués dans le processus de construction de la base/entrepôt de données ou le système d'intégration utilisent le même langage de modélisation des besoins et le même vocabulaire pour modéliser leurs besoins. Nous appelons ce scénario *1-vocabulaire-1-langage de modélisation*. Dans le contexte d'une entreprise étendue, les besoins peuvent parvenir d'acteurs hétérogènes, ce qui rend leur définition et exploitation difficiles. Cette difficulté est due à plusieurs facteurs : **(1)** le nombre croissant de partenaires, **(2)** la diversité (hétérogénéité) des vocabulaires d'expression de besoins, **(3)** l'hétérogénéité des langages de modélisation des besoins, **(4)** la présence de relations complexes entre les besoins et (5) l'évolution des besoins. Ces facteurs peuvent être détaillés comme suit.

1.1 Nombre croissant de partenaires

La fabrication d'une application complexe consommatrice de données nécessite souvent sa décomposition en un nombre important de modules (sous-systèmes), chacun géré par une sous-entreprise ou un sous-traitant particulier. Cela fait naître un besoin *d'automatisation* du processus d'intégration des besoins.

1.2 Hétérogénéité des vocabulaires

Souvent chaque collecteur de besoins utilise son propre vocabulaire pour exprimer ses besoins, ce qui peut générer des conflits syntaxiques et sémantiques.

L'hétérogénéité syntaxique provient du fait que les collecteurs de besoins utilisent des terminologies qui leur sont propres. En effet, les propriétaires de sous-systèmes sont souvent autonomes et chacun a

ses habitudes de travail et ses méthodes de recueil des besoins. Cette autonomie fait augmenter cette hétérogénéité d'une manière significative. Elle est similaire à celle identifiée dans les contextes des bases de données fédérées et des multi-bases de données.

L'hétérogénéité sémantique par contre, présente un défi majeur dans le processus d'élaboration d'un système d'intégration. Elle est due aux différentes interprétations des objets du monde réel. En effet, chaque collecteur de besoins peut avoir un point de vue différent sur le même concept. Dans [54], quatre types de conflits ont été identifiés: (i) conflits de représentation, (ii) conflits de noms, (iii) conflits de contexte et (iv) conflits de mesure de valeur.

- *Conflits de représentation* : ce type de conflit survient lorsqu'un même type de données est modélisé différemment, c'est-à-dire par des propriétés différentes ou des schémas différents (le nombre de classes et de propriétés représentant un concept n'est pas le même entre deux sources). A noter que la richesse d'un modèle de données augmente la probabilité d'occurrence de ces conflits.
- *Conflits de noms* : nous distinguons les deux sortes de conflits de noms suivants.
  - Même nom d'entité avec des significations différentes (homonymie) : ce conflit apparaît lorsque différents schémas utilisent des noms identiques pour des concepts différents.
  - Noms différents avec des significations identiques (synonymie) : ce conflit apparaît lorsque les différents schémas utilisent des noms différents pour représenter le même concept ou propriété.
- *Conflits de contextes* : le contexte est une notion très importante dans les systèmes d'information répartis. En effet, un même objet du monde réel peut être représenté dans les sources de données par plusieurs représentations selon un contexte local correspondant à chaque source. Ces conflits sont identifiés dans le cas où les concepts semblent avoir la même signification, mais qu'ils sont évalués dans différents contextes.
- *Conflits de mesure de valeur* : ces conflits sont liés à la manière de coder la valeur d'une propriété du monde réel dans différents systèmes. Ce conflit apparaît dans le cas où des unités différentes sont utilisées pour mesurer la valeur d'une propriété.

Pour réduire l'hétérogénéité sémantique, trois types d'approches ont été proposées : (i) approches manuelles, approches semi automatiques et approches automatiques.

- **Les approches manuelles** : l'intégration manuelle de données est la technique qui a été utilisée dans les premières générations de systèmes d'intégration. Ces approches permettent d'automatiser l'intégration des données au niveau syntaxique. Cependant, les conflits sémantiques sont résolus manuellement et nécessitent donc la présence d'un expert humain pour interpréter la sémantique des données (Figure 1.2).
- **Les approches semi-automatiques** : du fait que le nombre de sources de données devient important et/ou lorsque les sources évoluent fréquemment, les approches d'intégration manuelles deviennent très coûteuses et même impraticables. Des traitements plus automatisés deviennent nécessaires pour faciliter la résolution des conflits sémantiques. Dans cette partie, il s'agit d'une deuxième génération des systèmes d'intégration qui utilise les ontologies linguistiques *(thésaurus)* (Figure 1.3). Ces dernières permettent d'automatiser partiellement la gestion des conflits sémantiques. Les ontologies linguistiques sont ainsi utilisées pour comparer automatiquement les noms des relations ou des attributs. Cependant, cette comparaison est orientée "terme" et non "concept". Cela peut créer notamment des conflits de noms. De plus, les relations entre termes sont très contextuelles. Ainsi le traitement par ontologie linguistique est nécessairement supervisé par un expert et ne peut donc être que partiellement automatique. Deux ontologies linguistiques assez

FIGURE 1.2 – Résolution manuelle de l'hétérogénéité

FIGURE 1.3 – Résolution semi-automatique de l'hétérogénéité

connues ont utilisé ce genre d'approche : WORDNET[3] et MeSH (Medical Subject Heading)[4]. MOMIS [21] est un exemple de projet utilisant des ontologies linguistiques *Wordnet* pour intégrer semi-automatiquement des données de sources structurées et semi-structurées.

– **Les approches automatiques** : dans les deux générations précédentes la signification des données n'est pas représentée explicitement. Avec l'avènement des ontologies conceptuelles, un progrès important a pu être réalisé dans l'automatisation du processus d'intégration de sources hétérogènes grâce à la représentation explicite de la signification des données. Dans une ontologie conceptuelle, la sémantique du domaine est spécifiée formellement à travers des concepts, leurs propriétés ainsi que les relations entre les concepts. Une ontologie conceptuelle regroupe ainsi les définitions d'un ensemble structuré des concepts qu'elle veut représenter. Ces définitions sont

3. Wordnet est une base de données lexicale de l'anglais-américain avec plus de 100 000 synsets (http://wordnet.princeton.edu)

4. Un thésaurus médical(http://www.chu-rouen.fr/cismef/)

traitables par machine et partageables par les utilisateurs du système. La référence à une telle ontologie est alors utilisée pour éliminer automatiquement les conflits sémantiques entre les sources dans le processus d'intégration de données. Le processus d'intégration de données peut alors être entièrement automatisé. PICSEL [53], OBSERVER [91] et OntoDaWa [128], développé dans le cadre de la thèse de Dung NGUYEN XUAN (pour l'intégration des catalogues de composants de l'entreprise Renault), sont des exemples de projets utilisant des ontologies conceptuelles qui visent à intégrer automatiquement des données.

1.3 Hétérogénéité des langages de modélisation des besoins

Dans le contexte de l'intégration des sources de données, l'hétérogénéité de représentation de ces dernières n'est pas souvent prise en compte, car les modèles conceptuels des sources sont souvent traduits vers des modèles physiques qui sont les seuls à être pris en considération par le processus d'intégration. Dans le contexte d'expression des besoins, plusieurs langages peuvent être utilisés. Récemment, certains efforts de recherche on été entrepris pour remonter l'intégration des sources de données au niveau conceptuel, en établissant des procédures de rétro-conception des modèles physiques des sources pour remonter aux modèles conceptuels [76, 77]. Dans le contexte des besoins, leurs langages de modélisation doivent être unifiés car l'expression d'un seul besoin peut se faire avec une multitude de langages. Ceux-ci sont classés en trois principales catégories: *langages linguistiques* (informels), *langages semi-formels* (diagramme des cas d'utilisation d'UML, le langage de modélisation conceptuelle de Merise [101], Orienté but, etc.) et *langages formels* (ex. la méthode B). Nous classifions cette hétérogénéité en deux catégories principales :

- L'hétérogénéité intra-classe qui existe entre les langages de modélisation qui appartiennent à la même catégorie. Par exemple, le diagramme des cas d'utilisation et le langage orienté but appartiennent à la catégorie des langages semi-formels;
- L'hétérogénéité inter-classe qui existe entre des langages de modélisation qui n'appartiennent pas à la même catégorie. Par exemple, la méthode B et le langage orienté but appartiennent à deux catégories différentes : langages formels et semi-formels.

1.4 Exigence des relations complexes entre les besoins

Lors de la définition des besoins, l'identification des conflits et des contradictions entre eux est nécessaire. Différents types de relations ont été proposés dans la littérature. Nous pouvons citer, par exemple, les relations de conflits, les relations d'inclusion et d'équivalence.

1.5 Évolution des besoins

Au cours du cycle de vie d'un système complexe, les besoins des concepteurs, dans chaque sous-système, sont amenés à évoluer. Cette évolution a un effet négatif sur le succès du projet étant donné qu'elle entraîne de nombreuses modifications. Une modification de besoin peut être : l'ajout, la suppression et la modification. Plusieurs facteurs affectent l'évolution des besoins [109] : changement d'un besoin, sa priorité, la technologie, conflit entre les besoins, complexité du système, etc. En conséquence, cette évolution peut impacter le développement du système.

2 Notre Démarche

Pour répondre au problème d'intégration des besoins, nous sommes parti de travaux au laboratoire LIAS effectués sur l'intégration des sources de données hétérogènes [19, 20], où des solutions sémantiques en utilisant des ontologies de domaine ont été proposées. Ces dernières ont contribué largement à l'automatisation du processus d'intégration malgré la présence d'un nombre important de sources de données. Rappelons qu'actuellement, nous vivons dans une ère où plusieurs ontologies de domaine existent. C'est le cas du domaine de l'ingénierie où les catalogues de composants électroniques sont souvent décrits par des ontologies normalisées par l'ISO (PLIB : ISO 132584 "parts Library"), de la médecine où nous trouvons l'ontologie UMLS (Unified Medical Language System) ou dans le domaine de l'environnement où un nombre important d'ontologies a été développé (par exemple, Influenza Infectious Disease Ontology [5]). De ce fait, nous supposons dans nos travaux qu'il existe une ontologie de domaine et que chaque collecteur de besoins reprend a priori des concepts de cette ontologie pour construire son ontologie locale couvrant son problème. Cette dernière peut être soit un sous-ensemble de l'ontologie de domaine soit une spécialisation, c'est-à-dire un sous-ensemble éventuellement étendu par des classes plus spécialisées et/ou des propriétés additionnelles. De part leur aspect formel, les ontologies offrent également des mécanismes de raisonnement permettant de vérifier la consistance des besoins et d'inférer de nouveaux relations entre les besoins.

La présence de l'ontologie, certes résout les problèmes d'hétérogénéité syntaxique et sémantique. Par contre l'hétérogénéité des langages de modélisation des besoins reste présente. Pour éviter un nombre important de correspondances entre les différents langages ($\frac{n\times(n-1)}{2}$), où n représente le nombre de langages, nous présentons une méthode pivot. En d'autres termes, nous supposons que chaque collecteur de besoins a son propre langage de modélisation couplé à son ontologie locale, et au final l'ensemble des besoins sont transformés sur le langage de modélisation pivot que nous appelons "modèle pivot". Ce modèle pivot jouant le rôle de médiateur est également couplé à l'ontologie partagée pour faciliter son exploitation.

Le scénario précédent exige une correspondance de tout langage de modélisation des besoins vers le modèle pivot, ce qui peut être coûteux dans le cas des gros projets. Dans le deuxième scénario d'intégration, nous proposons de fusionner l'ensemble des langages en un seul langage de modélisation que nous appelons "modèle générique" et au lieu de faire des correspondance, nous utilisons l'instanciation. Cette fusion est réalisée à partir des méta-modèles des langages utilisés. En d'autres mots chaque langage devient une instance de ce modèle générique.

Étant donné que nos solutions d'intégration des besoins sont sémantiques, nous exploitons les mécanismes de raisonnement offerts par les ontologies pour identifier les besoins conflictuels et contradictoires.

Contrairement aux travaux existants qui traitent le problème d'intégration des besoins d'une manière isolée sans prendre en compte le système cible (une base/entrepôt de données, une interface homme machine, etc.), nos propositions apportent des solutions pour les phases de cycle de vie de conception du système cible. Pour illustrer cela, nous considérons la phase de conception physique d'un entrepôt de données. Durant cette phase, un ensemble de structures d'optimisation (les index, la fragmentation de données, les vues matérialisées, etc.) est sélectionné. Souvent cette sélection est effectuée à partir d'un

5. http://bioportal.bioontology.org/ontologies/ENVO

ensemble de requêtes. Dans ce travail, nous proposons une approche de sélection dirigée par les besoins.

3 Contributions

Nos contributions se résument en trois points illustrés sur la figure 1.5 : **(i)** le *placement des besoins dans l'architecture d'intégration des objets*, **(ii)** *l'intégration des besoins en prenant en compte deux types d'hétérogénéité* : une liée aux vocabulaires utilisés [25, 26] et l'autre liée aux langages de modélisation des besoins [26] et **(iii)** *le raisonnement et l'exploitation des besoins sur la phase physique* [79, 78, 18]. En ce qui concerne le premier point, nous valorisons les besoins des utilisateurs dans le

FIGURE 1.4 – Les quatre niveaux d'intégration.

contexte de conception des bases/entrepôts de données par l'ajout d'une nouvelle dimension d'intégration représentant les besoins (figure 1.4).
Pour offrir une intégration automatique dans le cadre des entreprises étendues, où chaque partenaire est autonome, notre approche doit prendre en compte le fait que chacun peut utiliser son propre langage de modélisation des besoins et sa propre terminologie. Pour remédier au problème de vocabulaire, nous proposons l'utilisation d'une ontologie couvrant le domaine de l'étude, dont nous faisons l'hypothèse qu'elle existe (figure 1.5). Dans ce scénario, nous supposons que chaque collecteur de besoins pioche ses concepts et propriétés à partir de cette ontologie qui joue le rôle d'un dictionnaire. Pour unifier les différents langages de modélisation des besoins, nous proposons un modèle pivot. Vu la diversité des langages de modélisation et dans le but de définir les composantes du modèle pivot, nous avons considéré une étude de cas basée sur l'ontologie du domaine universitaire (Lehigh University Benchmark (LUBM)) et trois langages de modélisation des besoins : *Use case d'UML*, le langage *orienté buts* et le *modèle de traitement de la méthode Merise*. Pour établir le couplage entre les modèles utilisés et l'ontologie, un travail de méta-modélisation de chaque modèle ainsi que de l'ontologie est effectué. Les concepts et les propriétés utilisés dans chaque modèle sont référencés par l'ontologie. Ceci permet de faciliter l'intégration des besoins. Deux scénarii d'intégration sont alors présentés, offrant plus d'autonomie aux collecteurs des besoins et le deuxième impose un modèle unifié (générique) intégrant l'ontologie de domaine.

FIGURE 1.5 – Aperçu des contributions de la thèse

Pour identifier les relations complexes entre les besoins, nous proposons un mécanisme de raisonnement à base ontologique. Rappelons que les ontologies fournissent une représentation formelle des connaissances et des relations entre concepts. Elles peuvent donc être exploitées pour raisonner sur les objets du domaine concerné. Dans notre cas, elles permettent des raisonnements automatiques ayant pour objet soit d'effectuer des vérifications de consistance sur les besoins (des conflits, des contradictions, etc.), soit d'inférer de nouvelles relations. Une étude de cas est proposée afin d'évaluer notre mécanisme de raisonnement où deux scénarii de raisonnement sont étudiées : *ship whole* et *fetch as needed*. Dans le premier scénario, aucun traitement n'est fait localement, le modèle générique a la charge d'établir l'ensemble des raisonnements. Dans le deuxième scénario, chaque partenaire effectue localement le raisonnement et ne renvoie que des besoins valides au modèle générique pour déclencher d'autres

raisonnements.
Comme nous l'avons déjà évoqué, nos travaux visent à montrer l'intérêt de la prise en compte des besoins sur le cycle de vie de conception de bases de données traditionnelles et avancées. Dans cette thèse, nous étudions en particulier la contribution de la prise en compte des besoins dans la conception physique. Durant cette phase, l'administrateur doit sélectionner des structures d'optimisation comme les index, les vues matérialisées, le partitionnement, etc. Cette sélection est souvent guidée par un ensemble de requêtes extraites à partir de fichiers de journalisation (logs). Nous montrons dans le cadre de cette thèse que cette tâche est primordiale dans la conception physique et peut se faire dès la collection des besoins finaux. Cela a un intérêt considérable sur la réduction des tâches de l'administrateur de bases de données vu quelles sont très complexes et surtout coûteuses. Une autre motivation de nos propositions est que les bancs d'essai utilisés par la communauté de bases de données, pour évaluer la qualité des algorithmes de sélection de structures d'optimisation, proposent des requêtes sous formes SQL et leur description sous forme de besoins [6]. Ils peuvent donc être directement pris en compte par notre approche.
Finalement toutes les propositions de cette thèse ont été prototypées et testées. Un outil d'aide à l'intégration et l'exploitation des besoins hétérogènes est développé. La partie liée à la conception physique est prototypée sur OntoDB, une plateforme de bases de données sémantique développée au laboratoire LIAS. Elle permet de stocker à la fois les données et l'ontologie qui décrit leurs sens.

4 Organisation du mémoire

Cette thèse est structuré en deux parties. La première partie présente les concepts permettant d'élaborer nos propositions. Vu la pluridisciplinarité de notre domaine d'étude, deux états de l'art sont présentés : le premier abord les concepts fondamentaux de l'ingénierie des besoins ($\mathcal{IB}$) quant au deuxième, il décrit les ontologies. Le chapitre 2 vise à faire une présentation succincte permettant d'introduire le domaine de l'ingénierie des besoins, les problèmes rencontrés lors de leurs expression et de leurs analyse dans le cas d'une entreprise étendue, notamment les problèmes liés à l'hétérogénéité des vocabulaires et des langages de modélisation des besoins. Ensuite, nous débattrons un ensemble de travaux que nous considérons majeurs liés à l'intégration et à l'analyse des besoins. Une synthèse qui nous a permis de positionner nos travaux par rapport à l'existant que nous présentons par la suite.

Le chapitre 3 décrit un état de l'art portant sur l'utilisation des ontologies dans le domaine de l'$\mathcal{IB}$. Nous décrivant, dans un premier temps, la notion d'ontologie de domaine, une classification des ontologies (ontologies linguistiques et ontologies conceptuelles), les langages de définitions des ontologies langages de modélisation, une formalisation des ontologies conceptuelles. Puis, dans un deuxième temps, nous présentons les principaux travaux relatifs au couplage entre les ontologies et l'expression et l'analyse des besoins. Une synthèse sur ces derniers est également présentée.

La deuxième partie de ce mémoire est dédiée aux contributions dans le cadre de l'intégration sémantique des besoins. Dans le chapitre 4, nous décrivons notre première proposition d'un modèle unifiant à la fois les vocabulaires et les langages de modélisation des besoins. Pour réaliser l'unification des vocabulaires, nous proposons une solution dirigée par une ontologie conceptuelle supposée existante. La présence de cette ontologie permet à l'ensemble des concepteurs d'exprimer leurs besoins en utilisant ses

6. http://www.tpc.org

concepts et ses propriétés afin d'éviter les conflits syntaxiques et sémantiques. La deuxième unification est plus difficile que la première, vu la diversité des langages de modélisation des besoins. Pour ce faire, nous nous sommes appuyés sur une étude de cas comprenant trois langages semi-formels à savoir : *Use case d'UML*, un langage orienté but et le modèle de traitements de la Merise. Le choix de ces langages est justifié par leur usage dans le domaine industriel et académique. Après une analyse des composantes de chaque langage utilisé, nous proposons leurs modèles génériques. Cette généricité nous a permis de définir un modèle pivot pour l'ensemble de langages étudiés. Finalement, pour assurer l'interopérabilité entre l'ensemble des langages, des correspondances entre le méta modèle de chaque langage et le modèle pivot sont définies. Un couplage entre le modèle pivot et le modèle d'ontologie est alors proposé, ce qui explicite la sémantique des langages utilisés.

Afin de réduire la complexité des correspondances faites entre le modèle pivot et l'ensemble des langages de modélisation des besoins, notre deuxième contribution qui fait l'objet du Chapitre 5 consiste d'abord à fusionner les méta-modèles des langages de modélisation des besoins ensuite d'instancier chaque langage de modélisation des besoins. De la même façon que dans le chapitre précédent, un couplage entre le modèle générique et le modèle d'ontologie est établi.

Le chapitre 6 étudie l'impact des besoins sur la conception physique d'un entrepôt de données sémantique. D'abord nous motivons le besoin de réduire le coûts d'administration d'un entrepôt de données pour une entreprise étendue. Ensuite, nous formalisons le problème de la conception physique d'une manière générale, ensuite une instanciation de cette formalisation est donnée pour deux probèmes classiques, qui sont la sélection des index de jointure binaire et la sélection de schéma de fragmentation horizontale. Finalement, une approche dirigée par les besoins pour la sélection des deux structures d'optimisation est proposée et validée.

Le chapitre 7, intitulé *"Prototype de validation"*, propose une implémentation de nos propositions à travers un outil prototype, nommé *OntoReqTool*. Celui-ci fournit une aide au concepteur pour intégrer les besoins hétérogènes et pour les exploiter dans un environnement de stockage de données sémantiques (ED). Pour intégrer les besoins, nous avons choisi l'éditeur d'ontologie *Protégé* car il facilite la manipulation des ontologies et offre la possibilité de faire du raisonnement sur les besoins à l'aide de ses moteurs d'inférences. Pour exploiter les besoins, nous les avons persistés dans un ED afin de les exploiter dans la phase physique. L'implémentation a été faite sur la base de données sémantique *OntoDB* qui a été développée au sein du laboratoire et dont on avait un accès total au son code et à sa documentation.

Pour conclure ce mémoire de thèse, nous établissons un bilan des contributions apportées et nous traçons différentes perspectives de recherche qui pourraient être menées ultérieurement.

Ce manuscrit comporte deux annexes. L'annexe A décrit les règles de transformation utilisées dans la validation des deux premières contributions. L'annexe B fournit les requêtes du banc d'essai Star Schema Benchmark (SSB) utilisées pour valider la troisième contribution.

Publications

La liste suivante représente les articles publiés dans le cadre de cette thèse.

1. **Ilyès BOUKHARI**, Ladjel BELLATRECHE, Stéphane JEAN, An Ontological Pivot Model to Interoperate Heterogeneous User Requirements, Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2012), pp. 344-358, LNCS, Springer, Heraklion, Crete, Greece, October 15-18, 2012.
2. **Ilyès BOUKHARI**, Ladjel BELLATRECHE, Efficient, Unified, and Intelligent User Requirement Collection and Analysis in Global Enterprises, Proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2013), ACM, December 2013, Vienna, Austria.
3. Selma KHOURI, **Ilyès BOUKHARI**, Ladjel BELLATRECHE, Stéphane JEAN, Eric SARDET, Mickael BARON, Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool, Computers in Industry Journal, Elsevier (Factor Impact: 1.529), 63(8): 799-812, 2012.
4. Ladjel BELLATRECHE, Selma KHOURI, **Ilyès BOUKHARI**, Rima BOUCHAKRI, Using Ontologies and Requirements for Constructing and Optimizing Data Warehouses, Proceedings of the 30th IEEE International Convention MIPRO, pp. 1568 - 1573, Opatija, Croatia, May, 2012.
5. Selma KHOURI, Ladjel BELLATRECHE, **Ilyès BOUKHARI**, Selma BOUARAR, More Investment in Conceptual Designers: Think about it!, 15th IEEE International Conference on Computational Science and Engineering, pp. 88-93, Paphos, Cyprus, December 5-7, 2012.
6. Selma KHOURI, Ladjel BELLATRECHE, **Ilyès BOUKHARI**, Stéphane JEAN, Amira KERKAD, Peut-on prévoir le comportement d'un ED dès sa conception ?, submitted to Ingénierie des Systèmes d'Information Journal.

Première partie

État de l’art

Chapitre 2

L'expression des besoins : un état de l'art

Sommaire

1 Introduction 20

2 Définitions 21

3 Processus d'ingénierie des Besoins 24

4 Langages de modélisation des besoins 26

4.1 Langages informels (langage naturel) 26

4.2 Langages semi-formels 27

4.3 Langages formels 27

5 Approches et méthodes pour l'expression des besoins 29

5.1 Approches dirigées par les buts 29

5.1.1 GQM : Goal Question Metric 30

5.1.2 KAOS 31

5.1.3 iStar (i*) 33

5.2 Approches à base de scénarii 33

5.3 Approches basées sur les automates et réseaux de Petri 35

6 Problématiques liées à l'expression des besoins 37

7 État des lieux sur l'intégration et l'analyse des besoins 37

7.1 L'intégration des besoins 37

7.1.1 Exemples d'approches dirigées par des méthodes issues de la logique mathématique 37

7.1.2 Approches dirigées par IDM 38

7.2 L'analyse des besoins 40

7.3 Synthèse 41

8 Conclusion 42

Résumé. Ce chapitre fait une présentation succincte introduisant le domaine de l'ingénierie des besoins. Nous nous intéressons aux problèmes d'expression et d'intégration de besoins hétérogènes et à leur exploitation dans un entrepôt de données dans le cadre d'une entreprise étendue, notamment aux problèmes liés à l'hétérogénéité des vocabulaires et des langages de modélisation de besoins. Ensuite, nous présentons un état de l'art des travaux d'analyse et d'intégration des besoins. Nous présentons par la suite une synthèse qui nous à permis de positionner nos travaux par rapport à l'existant.

1 Introduction

Dans une entreprise étendue, l'utilisation des bases de données avancées comme les entrepôts de données est importante. Ces entrepôts permettent de stocker d'une manière efficace des données pour des besoins opérationnels et/ou décisionnels. Le cycle de développement des bases/entrepôts de données comporte un nombre important de phases impliquant plusieurs acteurs. Huit phases principales composent ce cycle (figure 2.1) : **(a)** la collecte et l'analyse des besoins [90, 74, 102], **(b)** la phase conceptuelle [35, 93, 118], **(c)** la phase logique [117], **(d)** le processus d'extraction, de transformation et de chargement [106], **(e)** la phase de déploiement [8], **(f)** la phase physique [15], **(g)** la phase de personnalisation et de recommandation [17, 59] et **(h)** le *tuning* [34], chacune impliquant un ensemble de sous phases. En ce qui concerne les acteurs (parties prenantes), nous avons les collecteurs et les analystes de besoins, les concepteurs, les administrateurs, les data architects [7], les data analysts [8], etc. Notons que l'ensemble des travaux couvrant ce cycle de vie utilise les besoins des utilisateurs (fonctionnels ou non fonctionnels). Prenons l'exemple de la méthode de conception francophone Merise, la première tâche à faire est d'extraire l'ensemble des propriétés/attributs à partir d'un cahier de charge, caractériseront la couche conceptuelle. Par la suite, chaque entité sera décrite par un sous ensemble de propriétés. Lorsque nous abordons la conception physique, toutes les structures d'optimisation comme les index, la fragmentation, etc. sont dirigées par les requêtes qui peuvent être extraites des besoins. Pour personnaliser les accès à une base de données, nous nous intéressons aux besoins des utilisateurs. Mais la phase de collection et l'analyse des besoins est souvent négligée dans le processus de conception de bases de données avancées. Prenons l'exemple de conception des entrepôts de données : dans ses premiers travaux, Inmon [66] a proposé une méthode de conception dirigée par les sources, mais cette dernière a été critiquée par Ralph Kimball [80], qui a fait ressortir la nécessité de définir une conception à partir de besoins des utilisateurs. Sa proposition a été suivie par plusieurs autres études proposant des méthodes de conception orientées besoins. Récemment, Golfarelli et Rizzi dans leur livre intitulé *Data Warehouse Design: Modern Principles and Methodologies* [58] ont insisté sur le rôle des besoins dans la conception et la réussite des projets d'entreposage de données.

Nous sommes conscients de l'étendue de l'Ingénierie des Besoins (IB) du fait qu'elle couvre un ensemble de phases (cf. Section 2.3) : élicitation, modélisation, spécification et validation. Nous n'allons pas proposer de contribution dans chacune de ces phases, mais nous allons considérer l'expression, l'intégration et l'exploitation des besoins hétérogènes dans le cadre d'une entreprise étendue.

Ce chapitre est organisé comme suit. Dans la section 2, nous présentons quelques définitions de l'IB nécessaires à la compréhension de nos contributions. Ensuite, dans la section 3, nous situons le processus d'IB dans les approches traditionnelles de développement des systèmes d'information. Les langages de modélisation des besoins sont ensuite présentés dans la section 4. La section 5 détaille les principales approches d'expression et d'analyse des besoins. Dans la section 6, nous présentons les différents problèmes rencontrés lors de l'expression et de l'analyse des besoins dans le cas d'une entreprise étendue. Dans la section 7, nous présentons un état de l'art des travaux d'intégration et d'analyse des besoins. Avant de conclure le chapitre, nous présentons une synthèse qui nous a permis de positionner nos travaux par rapport à l'existant.

7. http://en.wikipedia.org/wiki/Data_architect
8. http://www.journaldunet.com/solutions/analytics/metier-big-data-data-scientist.shtml

FIGURE 2.1 – Les phases du cycle de vie d'un ED.

2 Définitions

L'Ingénierie des Besoins (IB) est un vaste domaine de recherche s'intéressant principalement aux approches et techniques visant à améliorer la phase de spécification d'un système informatique. Nous présentons tout d'abord quelques définitions des concepts importants pour la compréhension de celui-ci.

Parties prenantes : connues en anglais sous le nom *stakeholders*. Selon l'AFIS (Association Française de l'ingénierie Système) [9], une partie prenante constitue une partie intéressée par l'utilisation et l'exploitation du système (voir par ses impacts sur son environnement), mais aussi un agent participant à sa conception, sa production, son déploiement, sa commercialisation, son maintien en condition opérationnelle et son retrait de service. Les parties prenantes les plus couramment rencontrées sont les : *utilisateurs, concepteurs, développeurs, administrateurs, etc.*

Besoin : selon Abran et al. [6], un besoin est une description d'une propriété ou des propriétés du système qui doivent être remplies. Les besoins sont exprimés en termes de phénomènes du monde réel ou objets partagés par le système et son environnement, avec un vocabulaire accessible aux utilisateurs [67]. Souvent, les besoins peuvent être classés dans les deux catégories suivantes.

– *Besoins fonctionnels* : *les besoins fonctionnels spécifient une fonction que le système ou un com-*

9. http://www.afis.fr/upload/SDD

posant du système doit être capable d'exécuter. Les auteurs de cette définition soulignent que le système peut être vu comme un ensemble de composants.

Exemple 1

> *Dans un établissement, un système doit permettre à la scolarité de vérifier les moyennes des étudiants qui ont obtenu le master informatique.*

– *Besoins non fonctionnels* : un besoin non fonctionnel est défini par *un attribut ou une contrainte du système comme la flexibilité, la performance, la sécurité, etc.* qui ne porte pas sur la fonction de ce système.

Exemple 2

> *Le système doit permettre aux étudiants de ne voir uniquement que leurs propres moyennes.*

L'Ingénierie des Besoins (IB) : dans le cadre des systèmes d'information, elle est définie par Rolland comme *l'activité qui transforme une idée floue en une spécification précise des besoins, souhaits et exigences exprimés par une communauté d'usagers et donc qui définit la relation existante entre un système et son environnement* [102]. Van Lamsweerde [120] définit trois objectifs de l'ingénierie des besoins: (1) l'identification des buts que le système envisagé doit accomplir, (2) l'opérationnalisation de ces buts sous forme de fonctions et de contraintes et (3) l'assignation des fonctionnalités aux agents.

Modèle : une abstraction d'un système conçu sous la forme d'un ensemble de faits construits selon une intention particulière. Il doit pouvoir être utilisé pour répondre à des questions sur le système étudié.

Dans le domaine de l'IB, les rôles des modèles sont multiples car ils sont utilisés durant tout le processus de l'ingénierie des besoins. Les modèles peuvent être utilisés pour capturer et formaliser les besoins des utilisateurs. Ils peuvent ainsi intégrer et vérifier les besoins.

Méta-modèle : selon la définition du MOF *(Meta Object Facility)* [10], un méta-modèle définit la structure que doit avoir tout modèle conforme à ce méta-modèle. On dit qu'un modèle est conforme à un méta-modèle si l'ensemble des éléments qui constituent le modèle est défini par le méta-modèle. Par exemple, le méta-modèle UML définit que les modèles UML contiennent des packages, leurs packages des classes, leurs classes des attributs et des opérations, etc.

Méta-méta-modèle : un méta-méta-modèle est un modèle qui décrit des langages de méta-modélisation. Autrement dit, le méta-méta-modèle contient les éléments de modélisation nécessaires à la définition des langages de modélisation et il a également la capacité de se décrire lui-même.

Langage de modélisation : il sert à décrire un système général ou spécifique à un domaine et/ou un contexte par ses composants et leurs relations. Il est défini par une syntaxe abstraite, une sémantique et une syntaxe concrète. La syntaxe abstraite définit les concepts de base du langage. La sémantique définit comment les concepts du langage doivent être interprétés par l'homme ou par les machines [64]. Chaque modèle est construit en utilisant un formalisme de modélisation, c'est à dire un langage.

Dans le domaine de l'IB, chaque modèle est construit en utilisant un langage de modélisation des besoins. La modélisation des besoins peut suivre un principe de méta-modélisation à plusieurs niveaux d'abstraction. La figure 2.2 illustre ce principe en se basant sur l'architecure à 4 niveaux définie par le MOF. Le nivaux *(M0)* décrit les objets du monde réel (instances). Dans notre exemple, portant sur le

10. http://www.omg.org/mof/

FIGURE 2.2 – Exemple de métamodèlisation.

domaine universitaire, un objet pourrait être un étudiant particulier (dans notre exemple, Anne, 22 ans, Poitiers). Le niveau *(M1)* décrit le modèle utilisé pour définir les objets. Dans notre exemple, une classe *Étudiant* permet d'indiquer que les étudiants sont caractérisés par un *âge* et une *adresse*. Le niveau *(M2)* contient les primitives permettant de définir un modèle. Dans notre exemple, ce niveau contient les primitives *Classe*, *Méthode* et *Attribut* pour pouvoir définir la classe *Étudiant*. Enfin, le niveau *(M3)* contient le méta-méta-modèle. Celui-ci est réflexif (il se décrit lui-même) et permet de définir différents langages de modélisation au niveau *(M3)*.

Transformation de modèles : une transformation de modèle est une fonction, $\Phi : S \rightarrow T$, telle que: Φ prend en entrée un ensemble de modèles source S, et produit en sortie un ensemble de modèles cible T. S et T sont des modèles conformes aux deux ensembles de méta-modèles. Si les deux ensembles de méta-modèles sont identiques, la transformation de modèle Φ est appelé *(transformation endogène)*, autrement on l'appelle *(transformation exogène)*.

Il existe de nombreux critères de classification liés à la transformation de modèles [41, 108]. Dans notre travail, nous nous intéressons à deux types d'approches de transformation de modèles :

- **modèle à modèle *(model-to-model, M2M)*** : ce type de transformation permet de produire des modèles conformes aux méta-modèles cibles. ATL [11] *(Atlas Transformation Language)* et QVT [12] *(Query-View-Transformation)* sont parmi les langages de transformation assurant la transformation *model-to-model*.
- **modèle à texte *(model-to-text, M2T)*** : ce type de transformation permet d'obtenir des modèles cibles qui consistent essentiellement en chaînes de caractères. Généralement, la transformation *model-to-text* est utilisé pour effectuer la génération de code, comme la transformation de modèles

11. http://www.eclipse.org/atl/
12. http://www.omg.org/spec/QVT/

UML à un programme Java. Acceleo [13] et Xpand [14] sont des exemples de langages assurant la transformation model-to-text.

3 Processus d'ingénierie des Besoins

Généralement, un processus d'IB comporte plusieurs étapes consistant à découvrir, analyser, valider et faire évoluer l'ensemble des besoins relatifs aux fonctionnalités du système. Selon Nuseibeh et Easterbrook [95], un processus d'IB inclut les étapes suivantes : élicitation, modélisation, analyse, validation et gestion des besoins (figure 2.3). Ces étapes sont détaillées dans ce qui suit.

FIGURE 2.3 – Processus d'ingénierie des besoins.

1. **Phase d'élicitation des besoins** : dans un processus d'IB, la première étape est l'élicitation des besoins. Elle consiste à collecter, capturer, découvrir et développer les besoins à partir des sources variées (utilisateurs, concepteurs, etc.). L'élicitation des besoins repose sur deux tâches principales : (i) l'identification des parties prenantes et (ii) l'identification des besoins. La première tâche consiste à identifier l'ensemble des personnes ou des groupes qui sont mis en relation avec le système au cours de son cycle de développement. Parmi les parties prenantes nous pouvons citer : le décideur, le concepteur, le développeur, l'administrateur, l'utilisateur final, etc. La deuxième tâche est l'identification des besoins dans laquelle de nombreuses techniques, méthodes et approches peuvent être utilisées pour étudier en détail les besoins des différentes parties prenantes. *Dans cette thèse, nous avons supposé que la phase d'élicitation est déjà réalisée.*

13. http://www.acceleo.org/pages/accueil/fr
14. http://www.eclipse.org/modeling/m2t/?project=xpand

2. **Phase de modélisation des besoins** : la modélisation est une abstraction du problème par différentes représentations. Elle consiste en la construction d'une description et d'une représentation abstraite d'un système pour les besoins de l'interprétation [95]. La modélisation des besoins consiste à détailler et structurer les besoins en utilisant différents types de langages de modélisation (informels, semi-formels et formels) que nous détaillons dans la section 4 de ce chapitre.

3. **Phase d'analyse des besoins** : cette phase consiste à analyser la compréhension des besoins élicités et vérifier leur qualité en termes d'exactitude, de complétude, de clarté et de consistance. Le but de cette phase est de clarifier les besoins, de supprimer les incohérences et d'assurer la complétude et la non redondance.

   La norme IEEE 830-1998 [15] relative à la spécification des besoins liste huit principales caractéristiques que toute spécification des besoins doit assurer. Celle-ci doit être correcte, sans ambiguïté, complète, consistante, stable, vérifiable, modifiable et traçable. Nous détaillons ces caractéristiques ci-après :
   - *Spécification correcte* : la spécification doit être comparée à toute spécification de niveau supérieur (comme la spécification des besoins de l'ensemble du système), ainsi qu'avec d'autres normes applicables, afin de s'assurer que la spécification est correcte. Alternativement, le client ou l'utilisateur peut déterminer si la spécification reflète correctement leurs besoins réels. La traçabilité des besoins rend cette procédure plus facile et moins sujette à l'erreur.
   - *Spécification sans ambiguïté* : une spécification est sans ambiguïté si, et seulement si, toutes les exigences qui y sont énoncées ne possèdent qu'une seule interprétation.
   - *Spécification complète* : une spécification est complète si, et seulement si, elle vérifie les critères suivants.
     - Toutes les exigences importantes sont exprimées, qu'il s'agisse de fonctionnalités, de performances, de contraintes de conception, d'attributs, ou d'interfaces externes.
     - La spécification permet la définition des réponses du logiciel pour toutes les données d'entrée.
     - La spécification comprend une définition de tous les labels, références, figures, tableaux et illustrations qui y figurent, ainsi que la définition de tous les termes et unités de mesure utilisées.
   - *Spécification consistante/ cohérence* : une spécification est cohérente si, et seulement si, aucun sous-ensemble des exigences individuelles décrites ne présente un conflit.
   - *Spécification stable* : une spécification est dite stable et triée (classée) par importance si chaque besoin a un identifiant qui indique son rang d'importance ou sa stabilité.
   - *Spécification vérifiable* : une spécification est vérifiable si, et seulement si, toutes les exigences énoncées sont vérifiables. Une exigence est vérifiable si, et seulement s'il existe un processus avec lequel une personne ou une machine peut vérifier que le produit logiciel répond à cette exigence. En général, toute exigence ambiguë et non quantifiable n'est pas vérifiable.
   - *Spécification modifiable* : une spécification est modifiable si, et seulement si, sa structure et son style sont construits de telle sorte que, toute modification des exigences peut être facilement et complètement mise en oeuvre.
   - *Spécification traçable* : une spécification est traçable si chacune de ces exigences est claire et si la spécification facilite le référencement de chaque exigence dans le développement futur du système ou dans sa documentation.

15. http://www.math.uaa.alaska.edu/~afkjm/cs401/IEEE830.pdf

La spécification consiste à établir la liste finale des besoins en les organisant suivant des catégories fonctionnels et non fonctionnels. La sortie de la phase de spécification est la première version du cahier des charges.

4. **Phase de validation des besoins** : l'objectif de cette phase est la confirmation de la qualité des besoins et de leur conformité aux attentes des parties prenantes. Selon la norme EIA-632 [16], la validation est focalisée sur la vérification de la version finale du document des besoins pour détecter les conflits, les omissions et les déviations par rapport aux normes. La validation cherche à certifier que les besoins satisfont les attentes des parties prenantes et à assurer qu'elles définissent les fonctionnalités attendues du système. En cas de conflits, une négociation est envisagée pour amener les parties prenantes à un accord.
5. **Phase de gestion des besoins** : cette phase consiste à suivre l'évolution ou le changement des besoins, ainsi qu'à faire la traçabilité et le contrôle des différentes versions de ces besoins [126]. La traçabilité permet d'identifier l'origine des besoins ainsi que tous leurs liens internes ou les liens avec le reste du projet ou le contexte (réalisation, tests, etc.).

Dans le processus, décrit précédemment, nous remarquons que la modélisation et l'expression des besoins joue un rôle crucial dans le développement de projets. Aussi, nous détaillons, dans la section suivante, les différents langages de modélisation des besoins utilisés.

4 Langages de modélisation des besoins

Nous présentons dans ce qui suit une classification en trois catégories des langages de modélisation des besoins suivie par une analyse qui nous a permis de les comparer.

4.1 Langages informels (langage naturel)

Il s'agit de langages construits en langue naturelle avec ou sans règles de structuration. Parmi les méthodes reposant sur ce type de langages, nous pouvons citer le questionnaire, l'interview et le cahier des charges. L'avantage principal de l'utilisation des langages informels est sa facilité apparente de compréhension qui offre une manière familière de communiquer entre les parties prenantes du système. Le deuxième avantage est le faible coût de formation nécessaire pour savoir définir un besoin. L'utilisation d'un langage informel ne demande aucune formation particulière et les restrictions imposées par son utilisation nécessitent simplement la compréhension des règles de rédaction à suivre. Généralement, l'utilisation des langages informels dans l'expression des besoins induit souvent des besoins incohérents et ambigüs car ni leurs syntaxes, ni leur sémantique ne sont parfaitement définies. Ces besoins peuvent mener à une mauvaise compréhension du système. Ainsi il peut exister un décalage entre les besoins de l'utilisateur et ceux compris par les concepteurs ou un malentendu entre concepteurs et développeurs. Dans ce cas, le système obtenu ne correspond pas aux attentes du l'utilisateur. La figure (2.4) présente un besoin modélisé par un langage informel (langage naturel).

16. Electronic Industries Alliance. EIA-632: Processes for systems engineering. ANSI/EIA-632-1998 Approved: January 7, 1999.

Besoin fonctionnel

Le système doit permettre au service de scolarité de vérifier les moyennes des étudiants qui ont obtenu le master de recherche. Le système doit d'abord calculer la moyenne de l'étudiant. Si la moyenne est supérieure ou égal 10 sur 20, le système imprime l'attestation de réussite au master de recherche.
…

FIGURE 2.4 – Un besoin modélisé par un langage naturel.

4.2 Langages semi-formels

Les langages semi-formels sont les plus couramment utilisés dans la phase de modélisation des besoins. Ils sont généralement basés sur des notations graphiques avec une syntaxe précise. Le caractère graphique de ces langages est utile pour les échanges entre les différents acteurs du système. Parmi ces langages nous citons la modélisation UML *(Unified Modeling Language)* [23]. Les langages semi-formels aident à élaborer et à structurer les besoins des utilisateurs sous forme de diagrammes. Les notations graphiques permettent d'avoir une vision claire et abstraite du système. Les concepteurs peuvent les utiliser pour mieux comprendre les besoins et soumettre leurs propositions aux utilisateurs. En bref, l'utilisation des langages semi-formels offre une vue synthétique, structurante et intuitive du système. Un des inconvénients des langages semi-formels est le manque de sémantique précise des besoins. Ce dernier peut conduire à des besoins ambigüs et incohérents. La figure (2.5) présente un besoin modélisé par un langage semi-formel (modèle des cas d'utilisation d'UML).

FIGURE 2.5 – Un besoin modélisé par un langage semi-formel (Cas d'utilisation).

4.3 Langages formels

Parmi les recommandations de *Edmund Melson Clarke, Jr.* dans son article intitulé *Formal Methods: State of the Art and Future Directions* est l'utilisation des méthodes formelles dans l'analyse des besoins

utilisateurs [88]. Les langages formels sont basés sur des notations mathématiques avec une syntaxe et une sémantique précise. Nous pouvons citer le langage B [7] et le langage Z [113]. Les langages formels ont donc été proposés afin d'exprimer les besoins par un formalisme rigoureux permettant de définir correctement les termes employés pour limiter les incompréhensions entre les parties prenantes du système. Ainsi, ils permettent de faciliter la vérification et le raisonnement sur les besoins. Malgré ces quelques avantages certains, il existe des inconvénients avérés tels que le coût et l'effort demandés pour former le plus de personnes possible aux langages formels. Un besoin modélisé par un langage formel Event-B (Méthode B) est présenté comme suit :

MACHINE CALCUL-MOYEN
SEES ETUDIANT
VARIABLES
    etudiants
    moyenne
    etudiants_admis
INVARIANTS
    inv1 : $etudiants \subseteq ETUDIANTS$
    inv2 : $etudiants_admis \subseteq etudiants$
    inv3 : $moyenne \in etudiants \longrightarrow N$
    inv4 : $\forall x(x \in etudiants_admis \Longrightarrow moyenne(x) \geq 10)$
EVENTS
Initialisation
    begin
        act1 : etudiants : $= \phi$
        act2 : etudiants_admis : $= \phi$
        act3 : moyenne : $= \phi$
    end
Event calcul_moyenne $\simeq$
    any
        x
        m
    where
        grd1 : $x \in etudiants \setminus etudiants_admis$
        grd2 : $m \in N$
    then
        act1 : $moyenne(x) := m$
    end
Event admis $\simeq$
    any
        x
    where
        grd1 : $x \in etudiants \setminus etudiants_admis$
        grd2 : $moyenne(x) \geq 10$

```
        then
                act1 : etudiants_admis := etudiants_admis ∪ {x}
        end
END

CONTEXT ETUDIANT
SETS ETUDIANTS
END
```

Cette présentation de différents langages de modélisation des besoins nous montre la complémentarité des trois classes de langages. Notons que certains langages de modélisation des besoins offrent des outils graphiques qui facilitent la visualisation de l'ensemble de besoins, comme les cas d'UML, chose nécessaire dans le contexte d'une entreprise étendue.

Pour que les concepteurs mènent bien leurs travaux, des méthodes et des outils d'expression de besoins ont été proposés. Dans la section suivante, nous présentons quelques méthodes accompagnées par des outils graphiques.

5 Approches et méthodes pour l'expression des besoins

La littérature propose plusieurs approches et méthodes pour l'expression des besoins des utilisateurs. Nous pouvons citer : *les approches dirigées par les buts, les approches à base de scénarios, les approches (Problem frames) schéma de problème, les approches basées sur des automates et réseaux de Petri [52] (le cas du modèle de traitement de la méthodologie MERISE), l'approche L'Ecritoire, etc.*. Dans ce qui suit, nous présentons brièvement les approches les plus pertinentes par rapport à notre travail, et pour chacune d'entre elles nous explicitons l'angle sous lequel elle étudie le système.

5.1 Approches dirigées par les buts

L'approche orientée but doit sa naissance à Yue [131]. Elle s'intéresse à l'utilisation des buts pour le recueil, l'analyse, la validation et la gestion des besoins. Comme le définit Lamsweerde [119], *"un but est un objectif que le système considéré doit atteindre"*. Il peut donc être fonctionnel (décrit un service fourni), ou non fonctionnel (détermine la qualité de service). Dans la plupart des cas, les buts sont généralement fixés implicitement par les participants dans l'analyse préliminaire des besoins. D'autres buts peuvent être obtenus par raffinement, abstraction, etc [36]. Le but est constitué de composants actifs comme des humains, outils ou du logiciel, qu'on nomme " agents " ou " acteurs ". Un but peut faire intervenir plusieurs acteurs, et il est souvent stable à long terme. D'une manière générale, les approches dirigées par les buts offrent des mécanismes pour l'identification, l'organisation et la gestion des besoins. Ainsi elles proposent des mécanismes pour le raisonnement sur les besoins des utilisateurs.

Les approches orientées buts ont montré leur utilité et leur importance dans nombre d'activités de l'IB [103], [131], [107], [129], [104], [120], [100], entre autres :

- Vérifier la complétude de la spécification des besoins : les buts fournis un critère de mesure de la complétude d'une spécification de besoins; celle-ci est complète si l'on montre que l'ensemble

des buts peut être satisfait par l'ensemble des besoins de la spécification.
- *Expliquer les besoins aux parties prenantes du processus d'IB* : les buts notamment de haut niveau fournissent les raisons du développement du système. Un besoin n'existe que s'il y a un but qui justifie sa présence dans la spécification.
- *Explorer des alternatives de conception* : le système à développer peut fonctionner et interagir avec son environnement de multiples façons ; le mécanisme de réduction des buts et le graphe ET/OU qui en résulte aide les ingénieurs d'IB à expliciter de nombreuses alternatives et à raisonner pour déterminer celle qui semble la mieux appropriée à la situation du projet.
- *Structurer la collection des besoins* : le mécanisme de réduction des buts y contribue.
- *Etablir les liens de traçabilité* : le graphe de réduction des buts est un moyen d'établir les liens qualifiés de pré-traçabilité. On sait combien ces liens qui assurent la relation entre les objectifs organisationnels et les besoins à l'égard du système sont utiles pour propager un changement des premiers vers les seconds.
- *Identifier et gérer les conflits* : les différents acteurs du processus d'IB apportent des points de vue intéressants sur le système à développer; on sait que ces points de vue peuvent être divergents et générer des conflits. Les buts aident l'explicitation des conflits et à leurs résolution.

Différentes modèles suivent l'approches orientées buts : GQM [123], i* framework [129, 130], KAOS [43], TROPOS [28]. Nous présentons dans ce qui suit les modèles que nous considérons comme étant les plus représentatifs de cette approche. Ces modèles seront utilisés par la suite pour construire le modèle orienté but d'expression des besoins que nous proposons.

5.1.1 GQM : Goal Question Metric

Selon Basili [123]. *GQM représente une approche systématique pour établir des Buts en accord avec les besoins spécifiques d'une organisation, pour les définir d'une manière opérationnelle et traitable en les raffinant en un ensemble de questions qui à leur tour impliquent un ensemble de métriques spécifiques et de données à collecter*. Cette approche propose trois niveaux d'abstraction.

1. **Niveau conceptuel (Goal)** : Selon GQM, un but est défini de la manière suivante : *un but est défini pour un objet, pour plusieurs raisons, par rapport à plusieurs modèles de qualité, depuis plusieurs points de vue et est relatif à un environnement particulier*. Ex. Améliorer la qualité de l'enseignement des cours de base de données selon le directeur de l'université.
   - *Objet d'étude* : *(Qualité d'enseignement)*: représente la partie de la réalité qui est observée et étudiée.
   - *Raisons* : *(Améliorer)*: représente les motivations (les raisons) de l'étude de l'objet.
   - *Modèles de qualité* : *(Base de données)*: représente les caractéristiques de l'objet qui sont considérées dans l'étude.
   - *Points de vue* : *(Directeur de l'université)*: représente la personne ou le groupe de personnes intéressé par l'étude de l'objet.
   - *Environnement* : *(Université)*: représente le contexte d'application où l'étude est réalisée.
2. **Niveau opérationnel (Question)** : un ensemble de questions est utilisé pour caractériser la façon d'évaluer un but spécifique. Le rôle d'une question est d'essayer de caractériser un objet à mesurer et essayer de déterminer sa qualité par rapport à un point de vue précis. *Ex. Est ce que le créneau*

FIGURE 2.6 – Approche GQM *(Goal Question Metric)*.

horaire de cours convient aux étudiants ?

3. **Niveau quantitatif (Metric)** : Un ensemble de données est associé à chaque question pour lui apporter une réponse dans un but quantitatif. La métrique est constituée d'une manière quantitative pour répondre à une question spécifique. Ex. Le nombre d'étudiants inscrits au cours.

La figure2.6 présente la structure hiérarchique du l'approche GQM. Cette structure commence par déterminer un but spécifique, ensuite raffine le but en un ensemble de questions et chaque question est raffinée à son tour en plusieurs métriques. Une même métrique peut être utilisée pour répondre à différentes questions pour un même but.

5.1.2 KAOS

KAOS est une méthodologie d'ingénierie des besoins dirigée par les buts, et signifie *Keep All Objects Satisfied* [43]. Le concept principal dans KAOS est le concept de But, dont la satisfaction nécessite la coopération d'agents pour configurer le système. Les buts sont liés entre eux par des relations de décomposition AND/OR. KAOS fournit quatre modèles.

– Le modèle central est le modèle de buts qui décrit les buts du système et de son environnement. Ce modèle est organisé dans une hiérarchie obtenue grâce au raffinement de buts de plus haut niveau (les buts stratégiques) vers des buts de bas niveau (les exigences).

- Le modèle objet : il permet de décrire le vocabulaire du domaine. Il est représenté par un diagramme de classes UML.
- Le modèle des responsabilités : il permet d'assigner les exigences (les buts feuilles) aux différents agents. Ces agents appartiennent au système à construire (agents internes) ou à son environnement (agents externes).
- Le modèle des opérations : il représente les opérations du système en termes de leurs caractéristiques individuelles et de leurs liaisons avec les modèles de buts (liens d'opérationnalisation des exigences), objets (liens d'entrée-sortie) et responsabilités (liens d'exécution).
- Le modèle des comportements : il résume tous les comportements que les agents doivent accomplir pour satisfaire les besoins. Ces comportements sont exprimés sous la forme d'opérations exécutées par les agents responsables.

La figure 2.7 illustre un exemple de modèle de but selon KAOS. Le principal but *qualité d'un article de publication* est décomposé en plusieurs sous buts qui doivent tous être satisfaits pour satisfaire le but principal comme : *la définition du comité de programme*, *la réception des articles*, *la lecture des articles*, *la sélection des articles acceptés* et *l'édition finale*. Les buts qui sont des feuilles dans le graphe de but sont appelés des pré-requis. Ces derniers sont rendus opérationnels par des actions.

FIGURE 2.7 – Exemple d'un modèle orienté buts KAOS [13].

5.1.3 iStar (i*)

iStar (i*) acronyme de *'Intentional STratégic Actor Relationships'* est un framework de modélisation standardisé proposé par Eric Yu [129],[130] traitant la phase initiale d'analyse des besoins. Le framework i* distingue les deux phases suivantes dans l'IB.

- Phase initiale : elle vise à analyser et à modéliser les intérêts des utilisateurs et la manière de les intégrer dans différents systèmes et environnements.
- Phase finale: elle se concentre principalement sur la complétude, la consistance et la vérification automatique des besoins.

i* est fondée sur les sept concepts suivants.

- *Acteur* : un acteur représente un agent physique (par exemple une personne, un animal, une voiture) ou un agent logiciel.
- *But* : un but représente un intérêt stratégique d'un acteur. i* distingue les buts fonctionnels (hard-goals ou goals) des buts non-fonctionnels (soft-goals). Les soft-goals sont utilisés pour modéliser des qualités du logiciel, tels que la sécurité, la performance et la maintenabilité.
- *Croyance* : les croyances sont utilisées pour représenter les connaissances de chaque acteur sur le monde (l'environnement).
- *Plan (Task)* : un plan représente une manière ou un ensemble d'actions à réaliser pour satisfaire un but.
- *Ressource* : une ressource représente une entité physique ou informationnelle recherchée par un acteur et fournie par un autre acteur.
- *Capacité* : la capacité représente l'aptitude d'un acteur à définir, choisir et exécuter un plan pour réaliser un but, dans un environnement donné.
- *Dépendance*: une dépendance entre deux acteurs indique qu'un acteur dépend d'un autre acteur pour atteindre un but, exécuter un plan ou délivrer une ressource.

Un exemple (une instance du méta-modèle) d'un modèle iStar est présenté dans la figure 2.8. Le but principal de l'acteur *Auteur* est la publication de son article *(Article be Published)*. Ce but est accompli par la tâche soumettre l'article *(Submit article in conference)*. Cette tâche est accomplie par deux sous-buts article à soumettre *(Article be Submitted)* et version finale à transmettre*(Camera Ready Be Transmitted)*. Les autres nœuds représentent les buts et les tâches nécessaires pour accomplir ces deux sous buts.

5.2 Approches à base de scénarii

Dans les approches à base de scénarii, les besoins sont décrits à l'aide de scénarii visant à la compréhension du système par les participants. Les événements de scénarii sont des interactions entre l'utilisateur et le système. Un scénario peut être défini comme *l'ordre des actions ou des événements pour un cas spécifique d'une certaine tâche générique qu'un système doit accomplir* [105]. Il permet de capturer des scènes, des descriptions narratives du contexte et du fonctionnement du système, des cas d'utilisation et des exemples de comportement d'utilisateur. Les scénarios sont exprimés dans différents langages : informels (langage naturel), semi-formels (diagramme des cas d'utilisation (UML), tableaux, etc.) ou formels (langages basés sur des grammaires régulières, diagrammes d'états, etc.). Nous nous sommes intéressés plus particulièrement aux diagrammes des cas d'utilisation. Rappelons qu'il existe des outils

FIGURE 2.8 – Exemple d'un modèle orienté buts iStar (i*) [13].

comme Rational Rose [17] qui offrent des visualisations graphiques.

Un cas d'utilisation est un moyen pour décrire les exigences fonctionnelles d'un système. D'après Jacobson et al, *un cas d'utilisation est une description d'un ensemble de séquences d'actions, incluant des variantes, qu'un système effectue pour fournir un résultat observable et ayant une valeur pour un acteur*[68]. Chaque cas d'utilisation contient un ou plusieurs scénarii qui définissent comment le système devrait interagir avec les utilisateurs (appelés acteurs) pour atteindre un but ou une fonction spécifique d'un travail. Un acteur d'un cas d'utilisation peut être un humain ou un autre système externe à celui que l'on tente de définir. Pour clarifier le diagramme, UML permet d'établir des relations entre les cas d'utilisation. Il existe principalement deux types de relations : les relations de dépendances (Inclusion, Extension) et la généralisation/spécialisation.

- *La relation d'inclusion* : un cas A est inclus dans un cas B si le comportement décrit par le cas A est inclus dans le comportement du cas B : on dit alors que le cas B dépend de A. Par exemple, l'accès aux informations des étudiants inclut nécessairement une phase d'authentification avec un mot de passe (figure 2.9). Les inclusions permettent aussi de décomposer un cas complexe en sous-cas plus simples.
- *La relation d'extension* : si le comportement de B peut être étendu par le comportement de A, on dit alors que A étend B. Une extension est souvent soumise à condition. Graphiquement, la

17. http://www-03.ibm.com/software/products/fr/enterprise/

FIGURE 2.9 – Diagramme des cas d'utilisation *(Use case)*.

condition est exprimée sous la forme d'une note. La figure 2.9 présente l'exemple d'une scolarité où l'admission d'un étudiant n'intervient que si sa moyenne obtenue est supérieure ou égale 10 sur 20.

– *La relation de généralisation* : un cas A est une généralisation d'un cas B si B est un cas particulier de A. Pour la figure2.9, *Ajouter un étudiant* est un cas particulier *Gérer les étudiants*. La relation de généralisation/spécialisation existe dans la plupart des diagrammes UML et elle est interprétée par le concept d'héritage dans les langages orientés objet.

5.3 Approches basées sur les automates et réseaux de Petri

Dans cette section, nous nous concentrons que sur le modèle de traitements de méthodologie Merise [101] qui représente un bon exemple de méthode d'expression de besoins avec un réseau de Petri. Rappelons que Merise est un produit français, développé à l'initiative du Ministère de l'industrie en 1977 pour offrir aux entreprises publiques une méthodologie rigoureuse tout en intégrant les aspects nouveaux pour l'époque : les bases de données (modélisation entité association, le modèle relationnel, le déploiement sur des SGBD, etc.) et l'informatique répartie (le réseau, la fragmentation du schéma de la base de données, etc.). La principale caractéristique de cette méthode, est la séparation entre les données et les traitements. Un formalisme inspiré des réseaux de Pétri est proposé pour exprimer les traitements identifié lors de la phase des besoins (Figure 2.10).

Le Modèle Conceptuel de Traitement (MCT) utilisé par la méthodologie de Merise *est une succession de traitements déclenchés par des événements et qui donnent naissance à de nouveaux événements. Un traitement (opération) est une suite d'actions réalisées en vue d'obtenir un résultat*[99]. Dans notre

FIGURE 2.10 – Exemple de modèle conceptuel de traitement.

travail, nous nous intéressons à l'utilisation du MCT comme modèle des besoins des utilisateurs. (Voir figure 2.10.(a)).

- *Acteur :* est une personne morale ou physique capable d'émettre ou de recevoir des informations. Par exemple *la scolarité*.
- *Évènement :* C'est un fait réel qui, seul ou synchronisé avec d'autres évènements, a pour effet de déclencher une ou plusieurs actions. Un événement peut:
  - déclencher un traitement (ex : *Le système permet à la scolarité de vérifier l'admission des étudiants* déclenche le traitement *Vérifier l'admission des étudiants* (voir l'exemple de la figure 2.10.(b)).
  - être le résultat d'un traitement (ex : *(Étudiant admis)*, et à ce titre être, éventuellement, un événement déclencheur d'un autre traitement.
- *Traitement (opération) :* est un ensemble d'actions exécutées par le système suite à un événement, ou à une conjonction d'événements.
- *Règles d'émissions :* représentent des conditions permettant l'expression des solutions alternatives et l'exécution conditionnelle du traitement. Par exemple *(si la moyenne = > 10 alors (Étudiant admis). Sinon (Étudiant ajourné)).*
- *Synchronisation :* est une relation entre les événements déclencheurs qui est représentée par une condition booléenne grâce aux opérateurs logiques *" Et ", " Ou " et " Non "*. Le traitement n'est pas déclenché si la condition n'est pas réalisée.

Notons que le MCT est intégré dans l'outil Power AMC [18] de Sybase.

18. `http://www.sybase.fr/products/modelingdevelopment/poweramc`

Nous venons de présenter différentes approches pour exprimer les besoins des utilisateurs. Ces méthodes peuvent amener différents problèmes qui ont amené le travail réalisé dans cette thèse. Ces problèmes sont décrits dans la section suivante.

6 Problématiques liées à l'expression des besoins

L'importance des besoins augmente dans le contexte des entreprises étendues. Ces dernières nécessitent la collaboration d'un grand nombre de concepteurs qui peuvent provenir de différents domaines, départements, laboratoires de recherche, etc. Dans cette situation, produire un document des besoins de qualité devient difficile dû à plusieurs problèmes. Nous pouvons citer : (1) l'hétérogénéité des référentiels (vocabulaires), (2) l'hétérogénéité au niveau des langages de modélisation des besoins, (3) les relations complexes entre les besoins des utilisateurs et (4) l'évolution des besoins. Ces problèmes sont déjà détaillés dans le chapitre 1.

Les autres problèmes suivants peuvent conduire à des besoins ambigus, incomplets, incohérents, et incorrects [38].

- Problèmes de périmètre : si les frontières et les objectifs du système cible sont mal définis après l'analyse du contexte de l'organisation. Des besoins incomplets, non vérifiables, et inutilisables peuvent également être fournis. Cela conduit à l'inachèvement du projet, au dépassement de ses budgets, ou à la défaillance du système développé.
- Une compréhension incomplète des besoins.
- Une faible collaboration entre les différents utilisateurs ou concepteurs du système.
- Une faible connaissance du domaine par les concepteurs.

Dans le cadre d'une entreprise étendue impliquant un nombre important de concepteurs hétérogènes, une bonne étude des besoins nécessite trois phases principales : (i) l'expression des besoins, (ii) l'intégration des besoins et (iii) leur analyse. Nous avons déjà évoqué la première phase dans les sections précédentes, où nous avons montré la diversité des langages de modélisation des besoins. Dans la section suivante, nous allons détailler les travaux concernant deux dernières phases.

7 État des lieux sur l'intégration et l'analyse des besoins

7.1 L'intégration des besoins

Un nombre important de travaux sur l'intégration de besoins ont été proposés que nous pouvons classer en deux catégories : les approches dirigées par des méthodes issues de la logique mathématique [83, 127] et les approches d'ingénierie dirigée par les modèles (IDM) [86, 94, 5, 121, 30].

7.1.1 Exemples d'approches dirigées par des méthodes issues de la logique mathématique

Nous prenons l'exemple du travail de Wieringa et al. [127]. La particularité de ce dernier est qu'il s'intéresse à l'intégration des besoins exprimés à l'aide de trois langages appartenant aux trois catégories,

à savoir formelles, semi-formelles et informelles. Pour ce faire, les auteurs proposent un Framework appelé *TRADE (Toolkit for Requirements And Design Engineering)*, au sein duquel plusieurs techniques de spécification semi-formelles connues sont placées (par exemple, *Class-Relationship Diagrame, CRD*, *Communication diagram* et *Function refinement tree*). TRADE est basé sur une méthode de spécification structurée et orientée objet pour analyser les besoins. Dans ce travail, les auteurs combinent TRADE avec un langage formel de spécification (Albert II) [47]. Ce dernier a été développé et expérimenté depuis plusieurs années, il est destiné à la modélisation et la spécification des besoins exprimés par les utilisateurs. Les auteurs montrent que cette intégration conduit à une spécification cohérente des besoins formels et semi-formels. Cette approche a été illustrée par quelques exemples d'applications dans le domaine de la télécommunication distribuée.

Un autre exemple de travaux traitant deux langages appartenant à deux catégories de langages de modélisation des besoins différentes, à savoir semi-formelle et formelle est celui de Laleau et al. [83]. Les auteurs de ce travail proposent une méthode intégrant le langage SysML considéré comme langage semi-formel et la méthode B appartenant à la catégorie formelle. L'idée principale de ce travail est d'étendre le langage SysML avec les concepts appartiennent à des méthodes d'ingénierie des besoins existantes. L'approche s'effectue selon les étapes suivantes : (1) étendre le langage SysML par des concepts représentant les concepts de base du modèle de but de la méthode KAOS. Cette extension est concrètement établie au niveau du méta-modèle à l'aide des techniques de méta-modèlisation. (2) définition des règles pour dériver une spécification formelle de B à partir de ce modèle étendu. Cette étape permet de fournir une sémantique précise aux éléments du modèle de besoins.

7.1.2 Approches dirigées par IDM

L'IDM offre un cadre technique qui peut concerner les activités de l'IB autour de méta-modèles et les transformations de modèles, afin de fournir des spécifications des besoins qui soient correctes, complètes, cohérentes, sans ambiguïté, modifiables et faciles à lire et à sauvegarder. L'IDM peut s'appliquer dans différentes phases du processus d'IB, depuis l'élicitation jusqu'à la gestion des besoins. Elle est utilisée dans la modélisation, la persistance, l'intégration, la vérification, la traçabilité et la collaboration des besoins. Plusieurs travaux d'intégration des besoins, proposant l'utilisation des approches dirigées par les modèles, ont été proposés dans la littérature [86],[94],[5], [121], etc. Parmi ces travaux, voici une description de ceux étant les plus proches de nos préoccupations.

- **Lopez et al. [86]** : proposent un méta-modèle comme un schéma conceptuel (défini à priori) pour intégrer certains types de diagrammes semi-formels des besoins dans une approche de réutilisation des besoins (Figure:2.11). Le métamodèle généré supporte six techniques de modélisation des besoins connues, qui se concentrent principalement sur les besoins fonctionnels : scénarii, cas d'utilisation, diagrammes d'activité, flux de données, document tâches et workflows.
- **Navarro et al. [94]** : proposent une approche de modélisation des besoins qui permet l'intégration de l'expressivité de certaines techniques les plus pertinentes dans le domaine de l'IB, dans un Framework commun. Ce travail est basé sur les techniques de méta-modélisation de l'IDM comme un moyen d'intégration et de personnalisation. Pour établir cette intégration, les auteurs ont étudié cinq approches connues dans le domaine de l'IB : traditionnel (basées sur la norme IEEE 830-1998), Use Cases, Goal Oriented, Aspect Oriented et variability management. Les auteurs ont commencé par un ensemble limité de concepts, de manière à ce qu'il soit plus simple et

FIGURE 2.11 – Méta-modèle des besoins modélisés en UML. [86].

facile de parvenir à un méta-modèle consensuel. En outre, ils fournissent quelques conseils pour étendre cet ensemble de concepts, de telle sorte qu'une cohérence sémantique appropriée puisse être maintenue.

- **ITU-T12. [5]** : selon [Recommendation ITU-T Z.151], URN *(User Requirements Notation)* est un langage de modélisation qui permet l'élicitation, l'analyse, la spécification et la validation des besoins. URN combine deux langages de modélisation : le langage GRL *(Goal-oriented Requirement Language)* pour modéliser les acteurs et leurs besoins non-fonctionnels et la notation UCM *(Use Case Maps)* pour décrire les scénarios des besoins fonctionnels et les architectures. URN permet aux analystes des besoins de découvrir et de spécifier les besoins pour un système proposé ou un système évolutif, et d'analyser ces besoins pour détecter les incohérences et les ambigüités.
- **Vicente et al. [121]** : proposent un méta-modèle des besoins appelé REMM *(Requirements Engineering MetaModel)*. Ce travail est une première tentative pour apporter les avantages de l'approche *IDM* à la pratique de l'IB. Les auteurs notent que, se référant à [11], les éléments du méta-modèle des besoins dépendent fortement du contexte. Ainsi, le méta-modèle REMM est conçu dans un contexte de réutilisation des besoins, même s'ils estiment que les concepts et les relations du méta-modèle sont généralement applicables dans les approches de l'IB. REMM supporte trois types de relations entre les besoins : *DependenceTrace*, *InfluenceTrace*, et *ParentChildTrace*.
- **Brottier et al. [30]** : proposent un mécanisme dirigé par les modèles pour intégrer un ensemble de spécifications textuelles des besoins, écrites dans différentes syntaxes, dans un modèle global.

Ce mécanisme est intégré dans une plateforme appelée R2A qui signifie *(Requirements to Analysis)*. L'élément central de R2A est son méta-modèle des besoins qui a été défini pour capturer les besoins du modèle global. Le processus est composé de deux étapes basées sur les techniques de l'IDM. La première analyse chaque spécification des besoins textuels pour produire un modèle de syntaxe abstraite. La deuxième étape, interprète la sémantique abstraite de chaque modèle dans un modèle de besoins intermédiaire, puis fusionne ces modèles dans un modèle de besoins global des besoins (Figure:2.12). Ces deux étapes emploient les techniques de l'IDM. Ce travail fournit un modèle global des besoins.

FIGURE 2.12 – Etape de production d'un modèle global des besoins [30].

7.2 L'analyse des besoins

Nous rappelons que la phase d'analyse des besoins consiste à analyser la compréhension des besoins, de les exprimés et de les clarifier ainsi que de supprimer leurs incohérences, d'assurer la complétude de leurs description et déceler les redondances. Plusieurs travaux qui peuvent surgir d'analyse des besoins ont été proposés dans la littérature que nous détallions ci-après.

– **Mirbel et al. [92]** : proposent une approche permettant de vérifier la cohérence dans le modèle orienté buts (Goal-oriented). L'approche vise à détecter des relations implicites entre les buts et à vérifier les incohérences possibles entre eux. Dans ce travail, les auteurs ont utilisé la technique de la théorie de l'argumentation pour formaliser les besoins (buts) et leurs relations, et pour détecter les incohérences. Cette approche peut être considérée comme une aide apportée aux acteurs afin

de les aider à atteindre une compréhension commune d'un ensemble des besoins basés sur les buts.

- **Heitmeyer et al. [65]** : proposent une technique d'analyse formelle, appelée vérification de la cohérence pour la détection automatique des erreurs dans les spécifications des besoins. La technique est conçue afin d'analyser les spécifications des besoins exprimés par des notations tabulaires SCR *(Software Cost Reduction)*. Pour fournir une sémantique formelle pour la notation SCR, un modèle formel de spécification est introduit. Ce dernier représente le système comme un automate à états finis. Il définit un état du système en termes d'entités, une condition en tant que prédicat sur l'état du système, et un événement d'entrée comme un changement qui déclenche un nouvel état du système.
- **Giorgini et al. [51]** : proposent un framework de raisonnement sur les modèles de buts. Ces derniers utilisent des relations ET/OU, ainsi que des relations plus qualitatifs entre les buts comme par exemple les relations de contradiction. Le framework fournit une sémantique précise pour toutes les relations de but dans une forme qualitative et numérique en utilisant des algorithmes de propagation des étiquettes (*Label propagation algorithms*). L'analyste attribue des valeurs de contribution (+) et (-) pour les arêtes d'un graphe de buts. La valeur de contribution d'une arête représente le degré de la contribution positive ou négative entre deux buts. Ces valeurs quantitatives peuvent aider un analyste à reconnaître les conflits entre les buts. La principale limitation de ce framework est la définition des liaisons de contribution et l'affectation des étiquettes. C'est un problème que l'analyste peut rencontrer dans la modélisation de situations complexes où de nombreux buts sont liés entre eux, et lorsque différents points de vue doivent être pris en compte dans l'attribution des valeurs initiales aux buts. De plus, le framework de raisonnement présenté est très spécifique aux modèles de but.
- **Lamsweerde [84]** : propose une approche pour raisonner sur les besoins alternatifs. Ils ont introduit des relations d'influence entre les besoins, en indiquant si un besoin a une influence positive ou négative sur un autre besoin. Les auteurs appliquent des techniques de raisonnement formel pour identifier les besoins qui ont un impact négatif.

7.3 Synthèse

Dans cette section, nous avons fait un tour d'horizon sur les nombreuses phases de l'ingénierie des besoins. Cette étude nous a permis d'identifier certaines lacunes concernant l'intégration et l'analyse des besoins : (1) l'absence d'autonomie des concepteurs à cause de l'unique vocabulaire imposé pour l'expression des besoins, (2) l'indépendance des phases, en d'autres termes, l'expression (modélisation) et l'analyse des besoins sont faits *d'une manière indépendante*, d'où la présence de deux communautés de recherche distinctes, une qui traite la partie intégration et l'autre la partie analyse, malgré l'interdépendance des deux phases, (3) les relations entre les besoins (par exemple, les conflits) sont détectées trop tard ou pas du tout et (4) les besoins ne sont pas intégrés dans le cycle de vie de conception de bases de données avancées.

Pour résumer, nous proposons une comparaison des travaux étudiés selon 7 critères qui sont : (1) l'intégration langages de modélisation des besoins (IL), (2) les classes de langages utilisés (informels, semi formels, formels) (CLU), (3) l'intégration vocabulaires (IV), (4) l'analyse (A), (5) la couverture de cycle de vie (CCV), (6) la présence des outils (PO), (7) l'évolution (E). (Voir Tableau 2.1)

Travaux / Critères	IL	CLU	IV	A	CCV	PO	E
Wieringa et al.[127]	Oui	informels, semi-formels, formels	Non	Oui	Non	Oui	Non
Laleau et al. [83]	Oui	semi-formels, formels	Non	Oui	Non	Non	Non
Lopez et al.[86]	Oui	semi-formels	Non	Non	Non	Non	Non
Navaro et al.[94]	Oui	semi-formels	Non	Oui	Non	Non	Non
ITU-T12.[5]	Oui	semi-formels	Non	Oui	Non	Oui	Non
Vicente et al.[121]	Oui	semi-formels	Non	Oui	Non	Oui	Non
Brottier et al.[30]	Oui	informels, semi-formels	Non	Oui	Non	Oui	Non
Mirbel et al.[92]	Non	formels	Non	Oui	Non	Non	Non
Heitmeyer et al.[65]	Non	informels	Non	Oui	Non	Oui	Non
Giorgini et al.[51]	Non	semi-formels	Non	Oui	Non	Oui	Non
Lamsweerde et al.[84]	Non	semi-formels	Non	Oui	Non	Non	Non

TABLE 2.1 – Comparaison des travaux étudiés.

Cette comparaison nous montre l'intérêt de développer des approches unifiant les langages de modélisation des besoins et les vocabulaires et surtout "consolider" les besoins avec les phases de conception des bases de données avancées.

8 Conclusion

Dans ce chapitre, nous avons d'abord montré le rôle crucial des besoins dans le cycle de vie de conception des bases de données avancées. Les traités indépendamment dans l'ensemble des phases de ce cycle pénalise l'application finale. Pour mieux comprendre la communauté de l'ingénierie des besoins, nous avons revu certaines définitions et concepts liés à leurs phases : élicitation, modélisation, analyse et validation. Nous avons longuement insisté sur les langages de modélisation des besoins et surtout leur diversité. Trois classes de langages sont présentées : les langages informels considérés comme les pionniers de langages, les langages semi-formels, et les langages formels. Dans chaque classe, une large panoplie de langages existe. Cette diversité rend difficile leur intégration.

Dans le but d'intégrer les langages de modélisation des besoins hétérogènes, nous nous sommes focalisés sur la classe semi-formelle, où nous avons étudié trois familles de langages qui seront considérées dans notre étude de cas : approches dirigées par les buts, approches à base de scénarii et les approches basées sur des automates et réseaux de Petri (le cas du modèle de traitement de la méthode MERISE).

L'analyse des travaux d'intégration et d'analyse des besoins présentés dans la section 7 nous a montré l'ignorance de l'hétérogénéité des vocabulaires. Pour réduire cette hétérogénéité, nous proposons l'utilisation des ontologies que nous allons détailler dans le chapitre suivant. Notre vision est d'étendre les besoins à tout le cycle de vie de conception de bases de données avancées plutôt que de les limiter à la phase initiale.

Chapitre 3

Les ontologies au service de l'Ingénierie des Besoins

Sommaire

1 Introduction 45

2 Ontologies de domaine 45

2.1 Définitions et caractéristiques 45

2.2 Une taxonomies des ontologies de domaine 46

2.2.1 Les Ontologies Linguistiques (OL) 47

2.2.2 Les Ontologies Conceptuelles (OC) 47

2.3 Les langages de définitions des ontologies 48

2.3.1 Langages orientés gestion et échange de données 48

2.3.2 Langages orientés inférence 50

2.4 Les éditeurs d'ontologies 52

2.5 Représentation formelle d'une ontologie 53

3 Ontologies et l'intégration : application aux sources de données hétérogènes . 54

3.1 Intégration sémantique *a posteriori* 54

3.2 Intégration sémantique *a priori* 56

4 Les ontologies dans l'IB : état de l'art 57

4.1 L'analyse des besoins 57

4.2 L'intégration des besoins 60

4.3 Ontologies, besoins et le cycle de vie de conception 60

4.4 Synthèse 61

5 Conclusion 61

Résumé. Nous présentons dans ce chapitre un état de l'art portant sur l'utilisation des ontologies dans le domaine de l'IB. Nous abordons tout d'abord, la notion d'ontologie de domaine, ensuite la contribution des ontologies dans l'intégration des sources de données hétérogènes. Nous présentons également l'utilisation des ontologies dans le domaine de l'IB. Finalement, nous positionnons notre travail par rapport à l'état de l'art sur les travaux d'intégration des donnée à base ontologique.

1 Introduction

Au cours de ces dernières années, la notion d'ontologie s'est rapidement diffusée dans un grand nombre de domaines de recherche tels que les bases de données et l'intelligence artificielle (IA). Dans le domaine de base de données, elles ont contribué principalement à la conception de bases de données vu leur similarité avec les modèles conceptuels [49] et l'intégration de sources de données [20], tandis que dans le domaine de l'IA, les ontologies ont notamment offert des mécanismes de raisonnement et de déduction. Une ontologie est définie comme la représentation formelle et consensuelle au sein d'une communauté d'utilisateurs des concepts propres à un domaine et des relations qui les relient. Le but d'une ontologie est aussi de permettre de partager et de représenter formellement la sémantique des objets d'un domaine, ainsi que de supporter certains mécanismes de traitement automatique tel que le raisonnement. Compte tenu du caractère prometteur de cette notion, de nombreux travaux portent sur l'utilisation des ontologies dans des domaines aussi divers que le traitement automatiques de la langue naturelle, la recherche d'information, le commerce électronique, le web sémantique, la spécification des composants logiciels, l'ingénierie et l'intégration d'informations, etc. Récemment, les ontologies ont été utilisées dans le domaine d'IB afin de faciliter l'expression des besoins ainsi que pour détecter l'incohérence et l'ambiguïté sémantique entre les besoins utilisateurs. La fusion entre les deux domaines (ingénierie des besoins et ontologies) a suscité l'intérêt de la communauté de recherche depuis les années 80 et depuis, les ontologies ont démontré leur efficacité à résoudre d'autres problèmes comme l'indexation des données, le traitement de langage naturelle, l'échange et le partage des données, etc. En effet, l'ontologie peut servir de vocabulaire précis et partagé pour définir la sémantique des besoins à l'aide de concepts communs, formalisés et référençables.

Nous présentons dans ce chapitre un état de l'art portant sur l'utilisation des ontologies dans le domaine de l'IB. Il est organisé en cinq sections. Dans la section 2, nous introduisons les ontologies, leurs principales définitions, caractéristiques, langages de définition, éditeurs, et formalisation. Nous donnons ensuite leur classification en se basant sur la nature de ce qu'elles décrivent. La section 3 est consacrée à l'utilisation des ontologies pour l'intégration des sources de données hétérogènes. La section 4 présente un état de l'art sur les travaux qui utilisent des ontologies dans le domaine de l'IB. La section 5 conclut ce chapitre.

2 Ontologies de domaine

Dans cette section, nous définissons la notion d'ontologie tout en dégageant ses caractéristiques. Nous présentons ensuite ses principaux langages de définition, ses éditeurs et nous donnons sa représentation formelle. Enfin, nous présentons une taxonomie des ontologies de domaine, afin de bien préciser celles concernés par le travail réalisé dans le cadre de cette thèse.

2.1 Définitions et caractéristiques

Plusieurs définitions d'ontologies ont été proposées dans la littérature [61],[71],[98], etc. Pierra définit une ontologie de domaine comme *"a domain conceptualization in terms of classes and properties that is formal, multilingual, consensual and referencable"*[98]. Cette définition met en avant quatre ca-

ractéristiques qui distinguent une ontologie de domaine des autres modèles informatiques (tels que les modèles conceptuels et les modèles de connaissances).

1. *Formelle* : signifie que l'ontologie est basée sur des théories formelles qui permettent à une machine de vérifier automatiquement certaines propriétés de consistance et/ou de faire certains raisonnements automatiques sur l'ontologie et ses instances.
2. *Multilingue* : les ontologies dans certains domaines comme l'ingénierie sont multilingues [73]. Cette caractéristique favorise leur utilisation dans les grandes entreprises.
3. *Consensuelle* : signifie que l'ontologie est une conceptualisation validée par une communauté. C'est aussi un accord sur une conceptualisation partagée.
4. *Capacité à être référencée* : signifie que tout concept de l'ontologie est associé à un identifiant unique permettant de le référencer à partir de n'importe quel environnement, afin d'expliciter la sémantique de l'élément référençant.

Parmi les autres caractéristiques des ontologies qui distinguent une ontologie de domaine des autres modèles informatiques, nous pouvons citer les suivantes.

- *Capacité à raisonner* : le caractère formel des ontologies permet d'appliquer des opérations de raisonnement sur les ontologies. En l'occurrence, dériver des faits qui ne sont pas exprimés dans l'ontologie en utilisant des langages tels que la logique de description (LD). Dans notre contexte, les ontologies offrent des capacités de raisonnement qui peuvent être utilisées pour détecter les incohérences et les ambiguïtés sémantiques des besoins des utilisateurs. En effet, dans la plupart des modèles d'ontologies (OWL, PLIB), pour une ontologie et une classe données, on peut calculer (1) toutes ses super-classes (directes ou non), (2) ses sous-classes (directes ou non), (3) ses propriétés caractéristiques (héritées ou locales), (4) toutes ses instances (polymorphes ou locales), etc.
- *Objectif de modélisation* : une ontologie décrit les concepts d'un domaine d'un point de vue assez général, indépendamment d'une application particulière et de tout système dans lequel elle est susceptible d'être utilisée. A l'opposé, un modèle conceptuel de données prescrit l'information qui doit être représentée dans un système informatique particulier pour faire face à un besoin applicatif donné.
- *Atomicité des concepts* : à la différence d'un modèle conceptuel de données, où chaque concept ne prend son sens que dans le contexte du modèle dans lequel il est défini, dans une ontologie, chaque concept est identifié et défini explicitement de façon individuelle et constitue une unité élémentaire de connaissance. Un modèle conceptuel de données peut donc extraire d'une ontologie seulement les concepts (classes et propriétés) pertinents pour son objectif applicatif. Il peut également, sous réserve de respecter leur sémantique (la subsomption par exemple), les organiser de façon assez différente de leur organisation dans l'ontologie, la référence au concept ontologique permettant de définir avec précision la signification de l'entité référençante.

2.2 Une taxonomies des ontologies de domaine

Deux principales catégories d'ontologies de domaine ont émergé dans la littérature [98] : les *ontologies linguistiques* et les *ontologies conceptuelles*. Les ontologies linguistiques (OL) représentent les termes d'un domaine éventuellement selon différents langages naturels. Les ontologies conceptuelles

(OC) représentent les catégories d'objet et leurs propriétés d'un domaine donné. Les OCs diffèrent cependant selon le modèle d'ontologie utilisé pour les définir. On distingue ainsi deux catégories d'OC basées sur les notions de concepts *primitifs* et de concepts *définis* énoncées par Gruber [61]. Les concepts primitifs définissent le premier noyau du domaine conceptualisé par l'ontologie. Les concepts définis sont des concepts dont la définition utilise d'autres concepts (primitifs ou définis). Par exemple, le *prix total* d'une commande est défini en fonction du *prix unitaire* et de la *quantité commandée*.

2.2.1 Les Ontologies Linguistiques (OL)

Les ontologies linguistiques telle que WordNet [19] visent à définir le sens des mots et les relations linguistiques entre ces mots (synonyme, antonyme, hyperonyme, etc.). Ces relations servent à capturer de façon approximative et semi-formelle les relations entre les mots en utilisant des mesures de similarité. La construction de telles ontologies est souvent faite de façon semi-formelle par un processus d'extraction de termes dans un ensemble de documents du domaine qui est ensuite validé et structuré par un expert du domaine [116]. Plusieurs outils existent permettant de construire des ontologies à partir du texte. Nous pouvons citer Text2Onto [39] et Terminae [29], développé par le laboratoire d'Informatique de Paris-Nord.

2.2.2 Les Ontologies Conceptuelles (OC)

Les ontologies Conceptuelles (formelles) constituent des spécifications explicites de conceptualisations de domaines indépendamment de leurs représentations dans un langage particulier. Elles permettent de définir formellement les concepts de ce domaine et les relations entre ces concepts. Les ontologies conceptuelles adoptent une approche de structuration de l'information en termes de classes et de propriétés et leur associent des identifiants réutilisable dans différents langages.

Nous distinguons deux catégories d'ontologies conceptuelles : (1) les ontologies canoniques et (2) les ontologies non canoniques.

1. **Les Ontologies Conceptuelles Canoniques (OCC)** : sont des ontologies ne contenant que des concepts primitifs. Par exemple, on peut considérer une OCC contient une classe *Personne* caractérisée par deux propriétés *Nom* et *Sexe* (masculin ou féminin). Les OCC fournissent une base formelle pour modéliser et échanger les connaissances d'un domaine. Le modèle PLIB permet de définir des OCC.

2. **les Ontologies Conceptuelles Non Canoniques (OCNC)** : sont des ontologies contenant à la fois des concepts primitifs et des concepts définis. Par exemple, le concept primitif *Personne* qui se spécialise dans les deux concepts définis *Homme* et *Femme*, qui représentent les concepts d'une OCNC. Les OCNC fournissent des mécanismes permettant de représenter les connaissances d'un domaine par différentes conceptualisations. Les langages ontologiques issus du Web Sémantique (par exemple, OWL) permettent de définir des OCNC.

19. `http://wordnet.princeton.edu/wordnet/`

2.3 Les langages de définitions des ontologies

Plusieurs langages de définition des ontologies ont été développés dans la littérature. Ces langages offrent des constructeurs permettant l'expression des axiomes, des instances, des entités et des relations que l'on retrouve dans une ontologie. Ces langages peuvent être classés, selon les objectifs spécifiques visés par les différentes spécialités, en deux catégories: les langages orientés gestion et échange de données et langages orientés inférence.

2.3.1 Langages orientés gestion et échange de données

Dans cette section, nous présentons les langages d'ontologie RDF/RDF-schéma et PLIB. Ces langages, offrent respectivement dans le domaine du Web et de l'ingénierie, des constructeurs permettant de représenter les données du domaine de manière à faciliter le partage et l'échange des ontologies et des instances associées.

2.3.1.1 RDF (Resource Description Framework) [20] : est un modèle standard pour l'échange de données sur le Web. Associé à une syntaxe, il permet (i) la représentation et le partage des données structurées et semi-structurées et leur partage à travers différentes applications et (ii) l'annotation des éléments du Web. RDF étend la structure de liaison du Web pour utiliser les URI (Uniform Resource Identifier) pour le nommage des ressources. Une ressource est une entité d'information pouvant être exprimée avec RDF et référencée dans un bloc, par un nom symbolique (littéral) ou un identificateur (URI). Notons que le triplet est la construction de base en RDF qui est spécifié à travers le langage de balisage XML. Un modèle RDF est défini à partir de quatre ensembles suivants.

1. *Les ressources* : tout élément référencé par un URI est appelé ressource.
2. *Les littéraux* : un littéral est une valeur typée par un des types de données primitifs (chaîne de caractères, entier, etc.) définis par XML schéma.
3. *Les prédicats* : un prédicat est tout ce qui peut être utilisé pour décrire une ressource (propriété, attribut, relation, etc.).
4. *Les déclarations* : une déclaration est un triplet de la forme (sujet, prédicat, objet) permettant la description d'un élément. Le sujet est une ressource. Le prédicat est une propriété et l'objet est soit une ressource, soit un littéral.

Notons qu'un modèle RDF peut être représenté sous forme d'un graphe orienté où les nœuds sont des ressources ou des littéraux et les arcs représentent des prédicats.

2.3.1.2 RDF Schéma : Nous remarquons que RDF définit lui-même très peu de prédicats. De plus, il ne permet pas de définir les primitives pour modéliser un domaine en termes de classes et de propriétés. Par conséquent, il a été très rapidement complété par RDFS. En effet, RDFS est un langage de description de vocabulaire RDF permettant la description formelle de hiérarchies et de relations entre les ressources [114]. RDFS fournit les prédicats essentiels pour la représentation d'une ontologie. Ces constructeurs spécifiques sont les suivants :

20. http://www.w3.org/RDF/

– rdfs:class permettant la création d'une classe;
– rdfs:subClassOf spécifiant la relation de subsomption entre classes;
– rdfs:property permettant la spécification des propriétés caractéristiques d'une classe;
– rdfs:subProperty spécifiant une organisation hiérarchique des propriétés;
– rdfs:domain spécifiant le domaine d'une propriété;
– rdfs:range qui permet de spécifier le co-domaine de valeur d'une propriété.

La figure 3.1 montre un exemple d'ontologie présentée sous la forme d'un graphe. Cette ontologie comporte les classes *Person, University et Student (sous-classe de Person)*. Notons que le dernier concept décrit l'ensemble des étudiants. Les ovales représentent les URI des instances tandis que les valeurs littérales sont représentées dans des rectangles. Un extrait de l'ontologie et des données suivant la syntaxe rdf/xml est ensuite présenté. Après avoir étudié les deux langages, nous concluons que :

FIGURE 3.1 – Exemple d'ontologie exprimée en RDF Schéma.

– RDF n'est pas considéré totalement comme un langage d'ontologies étant donné qu'il ne permet pas de définir des classes et des propriétés;
– RDFS a enrichi RDF par l'ajout d'un ensemble de constructeurs permettant la définition des ontologies sur le Web. Malheureusement, RDFS n'offre aucune primitive pour la définition des équiva-

lences conceptuelles, du fait que tous les concepts décrits avec RDFS sont des concepts primitifs.

2.3.1.3 PLIB : (Parts LIBrary) [2], [3] est un modèle d'ontologies qui définit les représentations canoniques (uniques), supposées consensuelles, pour l'ensemble des concepts existant dans un domaine donné. Chaque concept peut alors être représenté par l'appartenance à une classe de l'ontologie et par un ensemble de valeurs de propriétés. Chaque concept est identifié de manière unique par un GUI (Globally Unique Identifier) [98]. A l'origine, l'objectif de PLIB était de permettre une modélisation informatique des catalogues de composants industriels afin de les rendre échangeables entre fournisseurs et utilisateurs. Il repose sur un langage formel EXPRESS pour la description des catalogues sous forme d'instances de modèle. Un tel catalogue contient une ontologie décrivant les catégories et les propriétés qui caractérisent un domaine donné et les instances de celle-ci. Le modèle d'ontologie PLIB offre une relation d'extension particulière, appelée case-of (est-un-cas-de). Cette relation fournit un mécanisme de modularité entre ontologies PLIB en permettant à un concepteur de base de données de définir sa propre ontologie (ontologie "locale") à partir d'une ontologie partagée.

2.3.2 Langages orientés inférence

Le besoin de réaliser des inférences sur les ontologies a conduit à la définition d'autres types de langages de modélisation d'ontologies. En particulier dans le domaine du Web, le pouvoir d'expression du langage d'ontologie RDFS a été étendu par un ensemble d'autres langages d'ontologies parmi lesquels OIL, DAMLONT, DAML-OIL, OWL et SWRL. Ceci afin de concevoir des ontologies permettant de réaliser davantage de raisonnement sur les ontologies, les données et les besoins des utilisateurs. Dans cette partie, nous proposons d'étudier deux langages (OWL et SWRL) manipulées dans ce manuscrit.

2.3.2.1 OWL : [21] un des langages les plus connus pour la définition d'ontologies est OWL *(Ontology Web Language)*. OWL repose sur RDF Schema est inspiré de DAML+OIL. OWL offre un vocabulaire riche avec une sémantique formelle pour la description des propriétés et des classes, ainsi que les relations entre classes. Il intègre divers constructeurs sur les propriétés et les classes comme l'identité, l'équivalence *(owl:equivalentClass)*, l'union *(unionOf)*, le complément d'une classe *(complementOf)*, l'intersection *(intersectionOf)*, les cardinalités, la symétrie, la transitivité, la disjonction *(owl:disjointWith)*, etc. L a figure 3.2 illustre un exemple d'ontologie de domaine universitaire exprimée en OWL.

OWL définit les trois sous-langages OWL Lite, OWL-DL et OWL Full, allant du moins au plus expressif.

- *OWL Lite* : est le langage le plus simple syntaxiquement. Il permet de définir des hiérarchies de classes et d'appliquer des contraintes simples. Par exemple une contrainte de cardinalité *(minCardinality, maxCardinality)* ne peut prendre que les valeurs 0 et 1.
- *OWL DL* : possède un pouvoir expressif plus important qu'OWL Lite. OWL DL garantit la complétude des raisonnements (toutes les inférences sont calculables) et leur décidabilité (leur calcul se fait en une durée finie) [40].
- *OWL Full* : est le langage le plus expressif, mais ne garantit pas toujours la complétude des raisonnements ni leur décidabilité (OWL Full n'est pas un langage décidable).

21. http://www.w3.org/TR/owl-ref/

```
<owl:Class rdf:ID="GraduateStudent">
   <rdfs:subClassOf>
      <owl:Restriction>
       <owl:onProperty>
     <owl:ObjectProperty rdf:ID="takesCourse"/>
       </owl:onProperty>
       <owl:someValuesFrom>
       <owl:Class rdf:ID="GraduateCourse"/>
       </owl:someValuesFrom>
     </owl:Restriction>
     </rdfs:subClassOf>
     <rdfs:subClassOf>
  <owl:Class rdf:ID="Person"/>
      </rdfs:subClassOf>
</owl:Class>
     <owl:Class rdf:ID="Program">
       <rdfs:subClassOf>
       <owl:Class rdf:ID="Organization"/>
      </rdfs:subClassOf>
</owl:Class>
.....
```

FIGURE 3.2 – Exemple d'ontologie exprimée en OWL.

OWL est un langage basé sur la logique de description qui est un formalisme logique permettant la représentation de la connaissance [12]. Une base de connaissances en LD est composée d'une TBox *(Terminological Box)* et d'une ABox *(Assertionnal Box)*. La TBox définit les connaissances *intensionnelles* du domaine sous forme d'axiomes. Dans la TBox, un nouveau concept peut être défini sur la base d'autres concepts (concepts primitifs). Il existe aussi des concepts atomiques (ou définis), ce qui signifie qu'ils ne sont pas définis sur la base d'autres concepts. Par exemple, on peut définir le concept *Student-Employee* comme étant l'intersection des descriptions des concepts *Student* et *Employee*: *Student-Employee* $\equiv$ *Student* $\cap$ *Employee*. La ABox définit les connaissances *extensionnelles* du domaine (les instances). Par exemple, le fait *Student (Anne)* fait partie de la ABox et permet d'indiquer que Anne est une étudiante *(Student)*.

2.3.2.2 SWRL[22] : (Semantic Web Rule Language) est un langage de règles qui enrichit la sémantique d'une ontologie définie en OWL, combinant le langage OWL-DL et le langage RuleML (Rule Markup Language). SWRL permet, contrairement à OWL, de manipuler des instances par des variables ($?x$, $?y$, $?z$). SWRL ne crée pas des concepts ni des relations, il permet simplement d'ajouter des relations suivant les valeurs des variables et la satisfaction de la règle. Les règles SWRL sont construites suivant ce schéma : *antécédent -> conséquent*. L'antécédent et le conséquent sont des conjonctions d'atomes. Un atome est une instance de concept, une relation OWL ou une des deux relations SWRL same-as ($?x$, $?y$) ou different-from ($?x$, $?y$). Le fonctionnement d'une règle est basé sur le principe de satisfiabilité de l'antécédent ou du conséquent. Pour une règle, il existe trois cas de figure :

- l'antécédent et le conséquent sont définis. Si l'antécédent est satisfait alors le conséquent doit l'être;
- l'antécédent est vide. Cela équivaut à un antécédent satisfait ce qui permet de définir des faits;

22. http://flashinformatique.epfl.ch/spip.php?article1212

– le conséquent est vide. Cela équivaut à un conséquent insatisfait, l'antécédent ne doit pas être satisfiable.

SWRL permet de raisonner sur les instances OWL, principalement en termes de classes et de propriétés. Par exemple, une règle SWRL définir le fait qu'une personne est considérée comme adulte. La règle en SWRL serait alors: $Person(?p), hasAge(?p, ?age), swrlb : greaterThan(?age, 18) -> Adult(?p)$

Après avoir introduit les principaux langages d'ontologies, nous présentons dans la section suivante les outils permettant leur manipulation.

2.4 Les éditeurs d'ontologies

Une autre particularité des ontologies est la présence d'outils facilitant leurs manipulation. Plusieurs éditeurs ont été proposés dans la littérature, citons à titre d'exemple OilEd [23], PLIBEditor [24], WebODE [25] et Protégé [26]. Dans notre travail, nous avons choisi l'utilisation de l'éditeur des ontologies Protégé. Ce choix est justifié par le grand succès qu'a connu cet éditeur, du fait qu'il est fortement soutenu par les communautés de développeurs et d'universitaires dans différents domaines.

2.4.0.3 Protégé : est un éditeur d'ontologies distribué en open source développé par l'université en informatique médicale de Stanford. Il possède une interface utilisateur graphique (GUI) lui permettant de manipuler aisément tous les éléments d'une ontologie: classe, méta-classe, propriété, instance,etc. (figure 3.3). Protégé permet aussi de créer ou d'importer des ontologies écrites dans les différents langages d'ontologies tel que : RDF-Schéma, OWL, DAML, OIL, etc. Cette fonctionnalité est disponible grâce à l'utilisation de plugins qui sont disponibles en téléchargement pour la plupart de ces langages. Aujourd'hui, il regroupe une large communauté d'utilisateurs et bénéficie des toutes dernières avancées en matière de recherche ontologique. Des applications développées avec Protégé sont employées dans la résolution des problèmes et la prise de décision dans un domaine particulier. Protégé est aussi une plate-forme extensible, grâce au système de plug-ins, qui permet de gérer des contenus multimédias, interroger, évaluer et fusionner des ontologies, etc. Il offre ainsi la possibilité de raisonner sur les ontologies en utilisant un moteur d'inférence général tel que Fact++ [27], JESS [28] ou des outils d'inférence propres au web sémantique basés sur des logiques de description tels que RACER [29].

Comme nous venons de le voir, différents modèles d'ontologies existent permettant de définir des ontologies grâce à un éditeur. Pour bien préciser les ontologies utilisées dans cette thèse, nous proposons dans la section suivante une représentation formelle que nous utilisons par la suite.

23. http://www.xml.com/pub/r/861
24. www.plib.ensma.fr
25. http://mayor2.dia.fi.upm.es/oeg-upm/index.php/en/technologies/60-webode
26. http://protege.stanford.edu/
27. http://owl.man.ac.uk/factplusplus/
28. http://www.jessrules.com/
29. http://www.sts.tu-harburg.de/r.f.moeller/racer/

FIGURE 3.3 – L'éditeur d'ontologies Protégé.

2.5 Représentation formelle d'une ontologie

Formellement, une ontologie O peut être définie comme un quintuplet [22]:
$O :< C, R, Ref(C), Ref(R), F >$, où :

- C représente les classes du modèle ontologique.
- R représente les rôles (relationships) du modèle ontologique. Les rôles peuvent être des relations liant des classes à d'autres classes ou des relations liant des classes aux valeurs de données (comme Integer, Float, String, etc.)
- $Ref(C) : C \rightarrow (operateur, Exp(C, R))$ est une fonction définissant les classes de la TBOX. *Ref(C)* associe à chaque classe un *opérateur* (d'inclusion ou d'équivalence) et une *expression, Exp* sur d'autres classes et propriétés. Les expressions définies pour les ontologies OWL basées sur les logiques de description présentent à notre point de vue un ensemble d'opérateurs complet couvrant plusieurs formalismes ontologiques. Ces expressions utilisent les opérateurs suivants : opérateurs ensemblistes (intersectionOf $\cap$, unionOf $\cup$, complementOf $\neg$), restrictions de propriétés (AllValuesFrom $\forall p.C$, SomeValuesFrom $\exists p.C$, HasValue $\ni p.C$) et les opérateurs de cardinalités ($\geq nR.C$, $\leq nR.C$). Exp peut être la fonction identité qui associe à une classe la même classe (la classe se définit par elle-même comme la plus grande classe de la hiérarchie "Thing"). Par exemple, *Ref(Student)*$\rightarrow$ *(*$\subseteq$*, Person* $\cap$ $\forall$ *takesCourse(Person, Course)*.

– $Ref(R) : R \rightarrow (operateur, Exp(C, R))$ est une fonction définissant les propriétés de la TBOX. $Ref(R)$ associe à chaque propriété un opérateur (d'inclusion ou d'équivalence) et une expression sur d'autres classes et propriétés. Par exemple :
Ref($GeneralHeadOf$) → (≡, (headOf(Person, Person) ∘ headOf$_{Departement}$(Person, Department)))
– F représente le formalisme du modèle ontologique adopté comme RDF, OWL, PLIB, etc.

3 Ontologies et l'intégration : application aux sources de données hétérogènes

Comme nous l'avons évoqué dans le chapitre Introduction, les ontologies linguistiques et conceptuelles ont largement contribué à l'intégration des sources de données hétérogènes. Dans cette section, nous résumons les travaux d'intégration de sources de données utilisant les ontologies conceptuelles qui sont conformes aux solutions d'intégration que le laboratoire LIAS a proposées. Les ontologies ont été utilisées selon trois architectures principales [124] (Figure 3.4) : une architecture avec une seule ontologie, une avec des ontologies multiples et une hybride. Dans la première architecture, chaque source référence la même ontologie globale de domaine (le cas du projet OntoDaWA [128] développé à LIAS). Cette architecture a un sens dans les entreprises donneuses d'ordre (exerçant une autorité). Dans une architecture à multiples ontologies, chaque source a sa propre ontologie développée indépendamment des autres (cas du projet OBSERVER [91]). Dans ce cas, les correspondances inter-ontologies sont parfois difficiles à mettre en oeuvre. Le processus d'intégration des ontologies est fait soit manuellement soit automatiquement [14]. Pour surmonter les inconvénients des approches simples ou multiples d'ontologies, une architecture hybride a été proposée dans laquelle chaque source a sa propre ontologie, mais toutes les ontologies utilisent un vocabulaire partagé commun. Le projet KRAFT [122] est un exemple de cette architecture.

Dans cette classification, [124] n'a pas considéré l'aspect temporel entre les ontologies locales et partagée. La prise en compte de ce facteur a fait naitre deux architectures ontologiques d'intégration de sources de données hétérogènes : (1) intégration sémantique a posteriori, et (2) intégration sémantique a priori.

3.1 Intégration sémantique *a posteriori*

Dans l'intégration sémantique *a posteriori*, chaque source est conforme à une ontologie locale indépendante. Dans ce cas, les ontologies locales aux sources sont développées soit d'une manière totalement indépendante les unes des autres, soit indirectement à travers une ontologie qui sera partagée par toutes les ontologies locales. L'étape d'intégration sémantique est donc effectuée de façon manuelle ou semi-automatique. Dans ce contexte, deux architectures d'intégration d'ontologies sont possibles.

1. **L'architecture en réseau** : dans cette architecture, la correspondance entre deux ontologies locales est établie directement de l'une à l'autre (figure 3.5-(a)). Les correspondances entre les différentes ontologies sont difficiles à mettre en œuvre dans une telle architecture. Pour un système d'intégration contenant n ontologies locales, il faut donc créer $[\frac{n\times(n-1)}{2}]$ correspondances. Ce nombre devient vite important et rend cette architecture impraticable si le nombre de sources à intégrer est

(a) Ontologie unique.

(b) Ontologies multiple.

(c) Ontologie hybride.

FIGURE 3.4 – Différentes architectures d'intégration à base ontologique.

(a) Architecture en réseau.

(b) Architecture en étoile.

FIGURE 3.5 – Architectures d'intégration sémantique *a posteriori*.

important. Parmi les systèmes d'intégration qui reposent sur cette architecture, nous pouvons citer OBSERVER [91].

2. **L'architecture en étoile** : dans cette architecture, la correspondance entre deux ontologies locales est établie indirectement via une ontologie référencée (figure 3.5-(b)). Cette ontologie est appelée *l'ontologie partagée*. Une ontologie locale n'est mise en correspondance qu'avec l'ontologie partagée, et seuls les concepts locaux intéressés par le système sont mis en correspondance avec les concepts de l'ontologie partagée. Pour un système d'intégration contenant n ontologies locales,

il faut donc créer n correspondances. Ce nombre est nettement plus réduit que de l'architecture précédente. Parmi les systèmes d'intégration qui reposent sur cette approche, nous pouvons citer KRAFT[122].

3.2 Intégration sémantique *a priori*

Dans l'intégration sémantique a priori, il existe une ontologie globale qui peut être utilisée comme dénominateur commun à tout ce qui a été défini par les sources de données. Le processus d'intégration consiste soit à créer les ontologies locales qui ne sont pas conçues d'une manière indépendante référençant l'ontologie partagée tout en préservant une certaine autonomie, soit à mettre en correspondance directe les sources avec l'ontologie globale. Deux structures d'intégration d'ontologies sont possibles: (1) l'architecture avec une ontologie globale commune et (2) l'architecture en étoile a priori.

1. **L'architecture avec une ontologie globale commune :** dans l'architecture avec une ontologie globale commune (figure 3.6-(a)), il n'y a qu'une seule et unique ontologie appelée ontologie globale et chaque source doit référence celle-ci. L'inconvénient majeur de cette approche réside dans le fait qu'aucun nouveau concept ne peut être ajouté localement à une source sans exiger le changement de l'ontologie globale. Cette architecture peut être vue comme le cas particulier de l'architecture précédente où les ontologies locales et l'ontologie globale sont confondues. Elle peut être assimilée également au cas des bases de données réparties où toutes les sources seraient plus ou moins développées d'une manière similaire. Parmi les systèmes d'intégration qui reposent sur l'architecture avec une ontologie globale commune, nous pouvons citer le projet ONTOBROKER [44].
2. **L'architecture en étoile a priori :** dans l'architecture en étoile a priori (figure 3.6-(b)), les ontologies locales sont construites à partir d'une ontologie globale de domaine en référençant a priori et localement ses concepts. Toutes les ontologies locales utilisent donc un vocabulaire partagé commun de l'ontologie partagée. L'inconvénient de cette approche est qu'elle n'intègre que les données dont la sémantique est représentée par l'ontologie globale. L'indépendance des sources de données ainsi est limitée, par contre, elle intégrer plus facilement une nouvelle source dans le système si la sémantique de cette source est couverte par l'ontologie globale. Le projet PICSEL (Production d'Interfaces à base de Connaissances pour des Services En Ligne)[53] est un exemple de système utilisant l'architecture en étoile a priori.

Une intégration sémantique *a priori* peut assurer l'automatisation complète du processus d'intégration si les deux conditions suivantes sont satisfaites [20] :

1. Chaque source doit représenter explicitement la signification de ses propres données ; c'est la notion d'ontologie locale qui doit exister dans chaque source;
2. Il doit exister une ontologie partagée de domaine, et chaque ontologie locale doit référencer explicitement l'ontologie partagée pour tous les concepts qu'elle définit et qui sont déjà définis dans l'ontologie partagée. Ceci définit la totalité des relations sémantiques existant entre les concepts des ontologies locales et ceux de l'ontologie partagée (articulations).

Sur la base de cette hypothèse, nous pouvons réaliser une intégration automatique des besoins hétérogènes des utilisateurs. Il s'agit d'utiliser des ontologies conceptuelles dans une perspective d'intégration sémantique a *priori*.

FIGURE 3.6 – Architectures d'intégration sémantique *a priori*.

4 Les ontologies dans l'IB : état de l'art

Aujourd'hui, plusieurs approches utilisent des ontologies de domaine pour faire face aux problèmes de l'IB. Breitman et Leite [27] soutiennent que les ontologies devraient être un sous-produit de la phase de l'IB (figure 3.7). Durant la phase d'élicitation, l'ontologie permet d'assister le concepteur pour identifier les besoins des différentes parties prenantes. Dans la phase de modélisation, l'ontologie fournit une représentation formelle offrant des mécanismes suffisamment expressifs pour représenter le modèle des besoins. L'aspect formel des ontologies permet de modéliser explicitement les besoins des utilisateurs d'une manière automatique et interprétable par la machine. L'utilisation d'une ontologie de domaine dans la phase d'analyse des besoins contribue à assurer une spécification correcte, complète et vérifiable des besoins. Selon [32], les utilisations potentielles des ontologies dans le processus d'IB comprennent : (i) la représentation du modèle des besoins, en imposant et en permettant notamment leurs structuration, (ii) les structures d'acquisition des connaissances du domaine et (iii) la connaissance du domaine d'application.

Les ontologies quelques soient leurs natures (linguistiques ou conceptuelles) ont contribué sur les deux phases de l'IB à savoir : l'analyse des besoins et l'intégration des besoins. Dans la section suivante, nous détaillons les travaux relatifs à ces deux aspects.

4.1 L'analyse des besoins

Parmi les problèmes lies à l'IB cités précédemment se trouve le risque de présence des incohérences et des ambiguïtés sémantiques. Ces dernières apparaissent lors de l'identification, l'analyse et l'évolution des besoins provenant de multiples utilisateurs et de sources. Les travaux existants couvrant la phase d'analyse utilisent soit une ontologie linguistique soit une ontologie conceptuelle (cf. section 2.2).

1. **Travaux dirigés par des ontologies linguistiques** : dans ces travaux, nous pouvons citer les travaux suivants :

FIGURE 3.7 – Contributions des ontologies dans l'IB.

- **Kaiya et al. [75] :** proposent une méthode d'analyse sémantique des besoins basée sur une ontologie de domaine. Des correspondances sont établies entre les spécifications des besoins et l'ontologie représentant les composants sémantiques. Dans ce travail, le système d'ontologie se compose d'un thésaurus et règles d'inférence. Le thésaurus comprend des concepts et des relations convenables pour le traitement sémantique du domaine. Cette méthode permet : (1) la détection d'incomplétude et d'incohérence inclues dans un cahier des charges, (2) la mesure de la qualité d'un cahier des charges par rapport à sa signification et (3) la prévision des changements futures des besoins en se basant sur une analyse sémantique de l'historique des changements.
- **Sven et al. [81] :** proposent une approche ontologique visant à améliorer le langage naturel pour spécifier les besoins. Les auteurs proposent un outil, appelé *RESI (Requirements Engineering Specification Improver)* pour aider les analystes des besoins à définir des besoins textuels. RESI identifie les lacunes linguistiques et propose un système de dialogue qui fait des suggestions afin d'améliorer la qualité des besoins en marquant les besoins ambigüs et erronés. Pour cette tâche, RESI utilise l'information sémantique supplémentaire donnée par l'ontologie.

2. **Travaux dirigés par des ontologies conceptuelles** :
   - **Greenspan [60] :** a proposé, dans le début des années 80, un langage de modélisation des besoins appelé RML *(Requirements Modelling Language)*. Ce langage a donné lieu à de nombreux travaux d'utilisation des ontologies dans le domaine de l'IB. RML a été développé selon les principes suivants : (1) L'écriture des besoins et leurs spécifications fonctionnelles doivent

être fournies. (2) Les besoins doivent être élaborés et présentés sous forme de modèles. (3) En se basant sur les deux points précédents, des modèles conceptuels devraient être développés. (4) L'abstraction et le raffinement sont importants dans l'IB. (5) Les langages de modélisation formelle des besoins sont nécessaires. RML défini son propre langage ontologique pour la modélisation des besoins. En RML, les modèles sont construits et regroupés en classes, qui représentent des instances de méta-classes.

– **Motoshi et al :** présentent un outil *AGORA (Attributed Goal-Oriented Requirements Analysis)* pour l'analyse des besoins orientés buts [110]. Une ontologie de domaine a été utilisée dans cet outil pour assister l'utilisateur afin de décomposer, raffiner, ajouter et supprimer des buts (Goals). L'utilisation d'une ontologie dans cet outil permet aussi de détecter les relations de conflits entre les buts.

Nous savons que les ontologies conceptuelles fournissent une représentation formelle des connaissances et des relations entre concepts. Par conséquent, elles peuvent également être exploitées pour raisonner sur les objets du domaine concerné. Elles permettent ainsi des raisonnements automatiques ayant pour objet soit d'effectuer des vérifications de consistance, soit d'inférer de nouveaux faits. Afin de faciliter la phase de raisonnement, des éditeurs d'ontologies munis de moteurs d'inférence ont été proposés (Pellet, Racer, etc.). Ceci montre l'intérêt majeur porté au processus de raisonnement sur les ontologies conceptuelle et les dispositifs offerts pour la réalisation de ce dernier. Plusieurs travaux ont utilisé les ontologies conceptuelles pour raisonner sur les besoins des utilisateurs. Nous décrivons ci-dessous les principaux travaux sur ce thème.

– **Siegemund et al. [112] :** proposent un méta-modèle ontologique orienté buts. L'idée présentée dans ce travail est d'utiliser une ontologie pour structurer les concepts et les relations sémantiques entre les buts lors de la phase d'élicitation. Cette phase permet de raisonner sur les buts en se basant sur la sémantique formelle. L'approche combine les capacités de raisonnement de l'ontologie OWL pour permettre la vérification de la cohérence et de la complétude des besoins aux cours du processus de l'IB. L'ontologie des besoins présentés dans ce travail est composée de deux concepts principaux: *But* et *Besoin*. Le principal inconvénient de cette approche est l'aspect consensuel qui caractérise les ontologies. L'ontologie des besoins présentée dans ce travail n'est pas générique par rapport au lanagage de modélisation utilisé et elle n'est pas consensuelle.
– **Dzung et al. [48] :** proposent une méthode de raisonnement à base ontologique dans la phase d'élicitation des besoins. L'idée présentée dans ce travail est d'utiliser l'ontologie de domaine pour structurer les concepts et les relations sémantiques entre les besoins lors de la phase d'élicitation. Les auteurs commencent par la définition d'une structure ontologique qui contient les connaissances des besoins fonctionnels et les relations entre eux. Ensuite ils proposent un framework pour éliciter les besoins en utilisant l'ontologie. En premier lieu des correspondances (mapping) sont établies entre les fonctions des besoins et l'ontologie de domaine. Après cette étape, des règles d'inférence sont utilisées pour raisonner sur les besoins afin de détecter les incohérences entre eux. Dans ce travail, les auteurs proposent également un outil de vérification des besoins reposant sur cette méthode.
– **Goknil et al. [56, 55] :** proposent un méta-modèle générique de besoins appelé *core metamodel* et une approche pour adapter ce méta-modèle afin de soutenir les différentes techniques de

spécification des besoins. Le méta-modèle représente les concepts communs extraits de certaines approches courantes. Les auteurs présentent des définitions et une formalisation des relations sémantiques *(requires, refines, conflicts et contains)* dans le méta-modèle. Sur la base de cette formalisation, les analystes peuvent effectuer des raisonnements sur les besoins pour détecter les relations implicites et les incohérences. Un outil appelé (TRIC) [30] est développé basé sur une ontologie OWL. Les différentes règles de formalisations des relations sont implémentées avec Jena (un raisonneur OWL) afin de détecter les relations implicites et les incohérences entre les besoins.

Nous notons que, dans ces études, les ontologies sont utilisées comme des référentiels de stockage des besoins et non comme des ontologies de domaine, où les concepteurs peuvent choisir leurs concepts et leurs propriétés afin de définir leurs besoins parmi l'ensemble des concepts définis dans l'ontologie (qui joue ainsi le rôle de dictionnaire ou catalogue) Une autre dimension ignorée par ces travaux est le passage à l'échelle des besoins dans leurs approches de raisonnement, ce qui peut être considérée comme un problème important dans les grandes entreprises.

4.2 L'intégration des besoins

Le problème d'intégration des besoins est un nouveau domaine, de ce fait il n'existe pas beaucoup de travaux impliquant les ontologies. En explorant la literature, nous avons identifie que les ontologies conceptuelles qui ont été utilisées pour répondre à ce problème [85]. Les auteurs de ce travail proposent un Framework (Onto-ActRE) *Ontology-based Active Requirements Engineering* . Le framework adopte une approche intensive pour éliciter, représenter et analyser la diversité des facteurs associés aux systèmes étendus. Onto-ActRE soutient la collaboration entre plusieurs techniques de modélisation de l'IB (orientée but, scénarios, etc) en exploitant l'expressivité des ontologies de manière structurée afin de faciliter l'élicitation, la spécification et la modélisation des besoins. Ce framework fournit un moyen pour comprendre et évaluer les effets des fonctions et des contraintes du système en fonction des concepts, des propriétés et des relations entre les concepts. Une sémantique complémentaire est fournie par une ontologie dans un processus *(Problem Domain Ontology, PDO)* qui DONNE la définition d'un langage commun pour faciliter la collaboration entre les parties prenantes du système et pour permettre également l'interaction entre les différentes techniques de modélisation des besoins.

Les inconvénients principaux de ce travail sont les suivants : (1) Ce framework n'offre pas une autonomie aux parties prenantes du système pour modéliser leurs besoins avec une technique souhaitée. (2) L'ontologie des besoins présentée dans ce travail est utilisée comme référentiel de stockage des besoins et non comme une ontologie de domaine et ne couvre donc pas toute la sémantique du domaine.

4.3 Ontologies, besoins et le cycle de vie de conception

Dans cette section, nous mettons l'accent sur un travail récent [115] publié dans *ACM Transactions on Database Systems* concernant le rôle des ontologies dans la conception de bases de données. Ce travail est considéré comme le seul à avoir contribué sur les trois aspects que nous avons évoqués dans cette thèse : les ontologies, les besoins et le cycle de vie de conception de bases de données.

30. Tool for Requirements Inferencing and Consistency Checking (TRIC) from http://trese.cs.utwente.nl/tric/

Les auteurs de cette contribution ont proposé un véritable outil d'aide à la modélisation conceptuelle (une phase du cycle de vie de conception de bases de données avancées) à partir d'ontologies de domaine. L'approche proposée permet aussi la validation de modèles conceptuels existants par rapport à une ontologie du même domaine. L'approche suivie est une approche *linguistique* dans laquelle, à partir d'une expression en langage naturel du cahier des charges de l'application et du choix d'ontologie(s) de domaine, le système analyse les termes du cahier des charges (l'ensemble des besoins) et, par référence à l'ontologie, suggère différents éléments à introduire dans le modèle conceptuel. Afin de rendre la méthode plus efficace, les auteurs proposent d'ajouter à l'ontologie des contraintes d'intégrité existentielles permettant de raisonner sur le modèle conceptuel.

4.4 Synthèse

L'IB a un grand impact sur le succès ou l'échec d'un projet informatique. Cependant, l'acquisition, la spécification et l'évolution des besoins de différents acteurs et d'une variété de sources conduit souvent à des besoins incomplets et ambigus. Par conséquent, la capacité à détecter et à réparer les besoins incohérents et incomplets est cruciale pour le succès de la modélisation et la spécification des besoins. Les ontologies sont bien adaptées pour résoudre les problèmes et les lacunes qui peuvent survenir lors de la mise en œuvre de l'IB. En effet, elles ont démontré leur efficacité pour la spécification, l'unification, la formalisation et le raisonnement sur les besoins. L'utilisation d'une ontologie de domaine est utile pour faciliter la communication et la collaboration entre les parties prenantes du système. En effet, l'ontologie peut servir de vocabulaire précis et partagé, pour définir la sémantique des besoins à l'aide de concepts communs, formalisés et référençables.

Généralement, la plupart des études citées précédemment traitent un seul niveau d'hétérogénéité (vocabulaire ou langages de modélisation). Dans cette thèse, nous proposons de résoudre cette problématique en mettant en place une approche ontologique d'intégration des besoins hétérogènes. Cette hétérogénéité concerne à la fois les vocabulaires et les langages de modélisation des besoins identifiés lors de la phase d'expression des besoins. La littérature nous a montré l'intérêt des ontologies et des besoins sur la modélisation conceptuelle de données [115], dans le cadre de cette thèse nous proposons d'étendre ces deux concepts à une autre phase, à savoir la modélisation physique.

5 Conclusion

Dans ce chapitre, nous avons réalisé une étude approfondie sur les ontologies de domaine afin de définir et de déterminer précisément leurs apports dans le domaine de l'IB. Les ontologies étant des modèles permettant d'expliciter la sémantique d'un domaine d'étude, elles sont donc, une solution pertinente pour résoudre l'hétérogénéité sémantique entre les besoins des utilisateurs. En effet, l'ontologie peut fournir un vocabulaire précis et partagé pour définir la sémantique des besoins à l'aide des concepts communs, formalisés et référençables. D'une manière générale, les utilisations potentielles des ontologies dans le domaine de l'IB comprennent la représentation, la spécification et la collaboration. Une autre caractéristique des ontologies est leur capacité de raisonnement qui peut être utilisée pour identifier les besoins conflictuels et contradictoires.

Dans notre travail, nous traitons les différents problèmes liés à l'hétérogénéité identifiée lors de la

phase d'élicitation et de modélisation des besoins. Notre approche se basant sur l'intégration à l'aide d'ontologie, dont l'intérêt et la pertinence ont été bien illustrés à travers ce chapitre. Nous présentons un état de l'art sur cette technique dans le chapitre suivant.

Deuxième partie

Contributions

Chapitre 4

Unification des vocabulaires et des langages de modélisation des besoins

Sommaire

1 Introduction 67

2 Nos hypothèses 68

3 Unification des vocabulaires 69

3.1 Étude de cas 70

3.1.1 Le modèle Orienté buts 70

3.1.2 Le modèle des cas d'utilisation 72

3.1.3 Le Modèle Conceptuel de Traitements (MCT) 73

3.2 Couplage des modèles utilisés avec l'ontologie 76

3.2.1 Méta-modèle de l'ontologie partagée 76

3.2.2 Couplage des trois langages proposés avec les ontologies locales 77

4 Unification des langages de modélisation des besoins 81

4.1 Proposition d'un modèle pivot des besoins 82

4.2 Couplage de modèle pivot à l'ontologie 83

5 Un exemple d'utilisation de notre méthode d'intégration 85

6 Conclusion 87

Résumé. Dans ce chapitre, nous détaillons notre première contribution consistant à unifier les vocabulaires et les langages hétérogènes [25]. Pour la partie vocabulaire, nous proposons aux concepteurs d'utiliser une ontologie de domaine qui joue le rôle d'un dictionnaire. Pour la partie langages, nous suivons une démarche par l'exemple. Cela est justifié par la diversité des langages de modélisation des besoins existants. Comme nous l'avons indiqué dans l'état de l'art, il existe trois catégories de langages : informels, semi formels, et formels. Dans chaque catégorie, une large panoplie de langages existe. Due à cette diversité, nous avons opté pour l'utilisation de la catégorie semi-formelle dans laquelle nous avons considéré trois langages : le modèle use case d'UML, le modèle orienté buts (goal-oriented) et le modèle de traitement (MCT) de la méthode Merise. Cela du fait qu'ils sont très utilisés dans le domaine industriel et académique. Après une étude approfondie de chaque langage, leurs méta modèles sont définis et le couplage avec un méta modèle de l'ontologie est établi. Finalement, un modèle pivot compilant les trois langages est dégagé avec le couplage de l'ontologie de domaine.

1 Introduction

Depuis la naissance des besoins d'intégration, l'hétérogénéité des modèles est devenue une des problématiques ayant suscité beaucoup de travaux de recherche. En effet, nous constatons que les concepteurs dans différents domaines (bases de données, services, etc.) n'ont pas pensé que leurs modèles seront un jour intégré (voir la figure 3.1). Le jour où les entreprises commencent à s'intéresser à l'intégration, les concepteurs se trouvaient face à des hétérogénéités de deux types : syntaxique/sémantique et langagière. Concernant les hétérogénéités de type syntaxique et sémantique, les ontologies de domaine ont largement contribué à les résoudre, notamment dans le cadre d'intégration des besoins. D'un autre coté, un ensemble de modèles pivots ont été proposés pour faire face à l'hétérogénéité langagière. Le rôle d'un modèle pivot est d'assurer une équivalence sémantique entre les différents modèles.

FIGURE 4.1 – Hétérogénéité des vocabulaires et des langages de modélisation des besoins.

Pour présenter nos contributions, nous commençons par définir quelques hypothèses suivies par l'analyse de trois modèles d'expression des besoins afin de comprendre leurs principes et fonctionnement. Cette analyse nous permet de méta-modéliser ces langages de besoins. Cette méta-modélisation permet d'instancier n'importe quel modèle de besoin exprimer à l'aide d'un langage étudié. Un modèle pivot est alors proposé à partir des trois modèles étudiés. Pour établir des ponts entre le modèle pivot et les trois modèles utilisés, des transformations de type modèle-vers-modèle sont définies.

Ce chapitre est organisé comme suit. Dans la section 2, nous présentons nos hypothèse afin d'élaborer notre approche d'intégration. Dans la section 3, nous présentons notre méthode unifiant les vocabulaires. Par la suite, dans la section 4, nous présentons notre méthodes afin d'unifier les langages de d'expression des besoins. Avant de conclure le chapitre, nous présentons un exemple d'utilisation de notre approche d'intégration.

2 Nos hypothèses

Deux hypothèses majeures sont considérées pour l'élaboration de notre approche.

1. L'existence d'une ontologie partagée (OP) qui précède la conception de l'application cible.
2. Chaque concepteur extrait un fragment de l'ontologie partagée (appelée Ontologie Locale (OL)). Afin de garder son autonomie, le concepteur peut étendre son ontologie locale en ajoutant des concepts et des propriétés qui n'existent pas dans l'ontologie partagée [20].

L'extraction de l'ontologie locale se fait selon une méthode de modularité, qui permet d'extraire une ontologie à partir d'une autre, tout en assurant la *complétude* et la *cohérence* de l'ontologie extraite. Trois scénarii d'extraction sont possibles :

1. $OL \equiv OP$: l'ontologie partagée correspond exactement aux besoins des concepteurs. L'expression des besoins en termes de classes, de propriétés ainsi que des fonctions *(Ref)* coïncide exactement avec la description de l'ontologie globale. Autrement dit, la description de l'ontologie locale correspond parfaitement à la description de l'ontologie partagée.

Formalisation 1

$\mathcal{OL}_i :< C_{OLi}, R_{OLi}, \mathcal{R}ef(C)_{OLi}, \mathcal{R}ef(R)_{OLi}, \mathcal{F}_{OLi} >$, *avec:*
- $C_{OLi} \equiv C_{OP}$.
- $R_{OLi} \equiv R_{OP}$.
- $Ref(C)_{OLi} \equiv Ref(C)_{OP}$.
- $Ref(R)_{OLi} \equiv Ref(R)_{OP}$.

2. $OL \subseteq OP$: l'ontologie locale est extraite à partir de l'ontologie partagée et couvre tous les besoins des utilisateurs. En effet, la description des besoins en termes de concepts ontologiques décrit un fragment de l'ontologie globale. Plusieurs méthodes et mécanismes de modularité ont été proposés et définis selon le formalisme d'ontologie utilisé, comme l'opérateur *Case-of* pour le langage PLIB. Ainsi, $OL \subseteq OP$ signifie que:

Formalisation 2

$\mathcal{OL}_i :< C_{OLi}, R_{OLi}, \mathcal{R}ef(C)_{OLi}, \mathcal{R}ef(R)_{OLi}, \mathcal{F}_{OLi} >$, *avec:*
- $C_{OLi} \subseteq C_{OP}$.
- $R_{OLi} \subseteq R_{OP}$.
- $Ref(C)_{OLi} \subseteq Ref(C)_{OP}$.
- $Ref(R)_{OLi} \subseteq Ref(R)_{OP}$.

3. $OL \supseteq OP$: le sous-ensemble de concepts (classes et propriétés) importé de l'ontologie partagée ne couvre pas tous les besoins des utilisateurs. Ce sous-ensemble nécessite en général d'être enrichi et spécialisé pour répondre aux objectifs applicatifs spécifiques. Ainsi, le concepteur peut, si l'ontologie partagée ne couvre pas de façon suffisamment détaillée le domaine de son application, étendre l'ontologie locale par ajout de nouvelles classes et/ou de nouvelles propriétés. Les concepts et propriétés pourront aussi être renommés pour mieux refléter le contexte applicatif. Notons que ce renommage est sans conséquence car chaque concept de chaque ontologie est associé à un unique identifiant et que les relations formelles sont des relations entre identifiants. Ainsi, $OL \supseteq OP$ signifie que :

Formalisation 3

$\mathcal{OL}_i :< C_{OLi}, R_{OLi}, \mathcal{R}ef(C)_{OLi}, \mathcal{R}ef(R)_{OLi}, \mathcal{F}_{OLi} >$, avec:

- $C_{OLi} \supseteq C_{OP}$.
- $R_{OLi} \supseteq R_{OP}$.
- $Ref(C)_{OLi} \supseteq Ref(C)_{OP}$.
- $Ref(R)_{OLi} \supseteq Ref(R)_{OP}$.

Exemple 3

La figure 4.2 présente un exemple de définition d'une ontologie locale à partir de l'ontologie globale LUBM. Dans un premier temps, nous définissons OL par l'importation d'un ensemble de classes couvrant une partie des besoins du concepteur. Dans un deuxième temps, les besoins non couverts sont traités par l'ajout de nouveaux concepts. Par exemple, les concepts décrivant les étudiants diplômés (DegreeStudent), étudiants sans diplôme (Non-degreeStudent) et la propriété (Degree) ne sont pas définis à travers l'ontologie globale (DegreeStudent, Non-degreeStudent, Degree) $\notin C_{LUBM} \cup R_{LUBM}$. *Par conséquent, ces deux classes et leurs propriétés, sont rajoutées au niveau de l'ontologie locale. De plus, la description de la classe étudiant (Student) est modifiée de manière à ce qu'elle soit définie comme l'union des étudiants diplômés (DegreeStudent) et des étudiants sans diplôme (Non_-degreeStudent). L'ontologie locale sera complétée comme suit.*

- C_{OL} = *{Person, Student, DegreeStudent, Non-degreeStudent}*
- R_{OL} = *{PersonId, Name, Age, Marks, Degree }*
- $Ref(C)_{OL}$ = *{Ref(Student)*$\rightarrow$ *(*$\subseteq$*, Person* $\cap$ $\forall$ *takesCourse(Person, Course))}.*
- *la définition de la classe étudiant Student est modifiée de la manière suivante :* $Student \equiv DegreeStudent \cup Non-degreeStudent$.

3 Unification des vocabulaires

Maintenant nous avons tous les ingrédients pour proposer notre méthode unifiant les vocabulaires. Pour cela, nous considérons une étude de cas de trois langages qui sera détaillée dans la section suivante. Pour chaque cas, nous donnerons une formalisation détaillée et un exemple.

FIGURE 4.2 – Exemple d'extraction de l'ontologie locale.

3.1 Étude de cas

Comme nous l'avons déjà évoqué, nous considérons une étude de cas comprenant trois langages de modélisation des besoins appartenant à la famille semi formelles, à savoir *orienté buts* (Goal oriented), *Cas d'utilisation UML*, et *MERISE (Modèle conceptuel de traitements, MCT)*. Pour rester dans le contexte d'une entreprise étendue, nous supposons que les trois langages de besoins sont exprimés respectivement par trois départements différents dont l'origine est l'Espagne, l'Angleterre et la France. L'ontologie partagée dans cette étude de cas représente l'ontologie du banc d'essai LUBM [62] (voir la figure 4.3). Elle comporte différentes classes relatives au domaine universitaire *(Person, Student, Course, etc.)*.

Pour faciliter la compréhension de notre proposition, nous détaillons dans les sections suivantes l'ensemble de trois modèles utilisés.

3.1.1 Le modèle Orienté buts

Comme nous l'avons vu dans le chapitre 2, différents modèles orientés buts existent. Aussi, après une analyse des approches orientées buts, nous avons proposé notre modèle de besoin dirigé par les buts, qui se veut un modèle pivot des approches étudiées (KAOS, IStar, et Tropos) et qui repose sur le paradigme GQM (Goal/Question/Metric).

Le modèle orienté but (Goal-oriented) pivot est formalisé comme suit :

FIGURE 4.3 – Une partie de l'ontologie *University* proposée par le benchmark LUBM.

Formalisation 4

$Goal_{model}$: $< \mathcal{A}ctor, \mathcal{G}oal, \mathcal{G}oal_{relationships} >$, où :

- *$\mathcal{A}ctor$: $\{actor_1, actor_2, ..., actor_n\}$ est l'ensemble des acteurs interagissant avec le système pour répondre au but.*
- *$\mathcal{G}oal$: $\{goal_1, goal_2, ..., goal_n\}$ est l'ensemble des buts exprimés par un acteur (ex: le concepteur). Un but représenté par le triplet : $\mathcal{G}oal$: $< \mathcal{T}, \mathcal{R}, \mathcal{M} >$, où :*
  - *$\mathcal{T} = \{t_1, t_2, ..., t_n\}$ est l'ensemble des tâches qu'un système effectue pour atteindre un but. $\forall t_i \in \mathcal{T}, \exists p_j \subseteq \mathcal{P}$, tel que $f(t_i) = p_j$, où $p_j \in \{p_1, p_2, ..., p_n\}$ est l'ensemble des propriétés satisfaites par un système. Ici, $f(t_i) = p_j$ est une fonction qui représente l'ensemble des propriétés p_j définissant t_i.*
  - *$\mathcal{R} = \{r_1, r_2, ..., r_n\}$ est l'ensemble des résultats réalisés par le système.*
  - *$\mathcal{M} = \{m_1, m_2, ..., m_n\}$ est l'ensemble des métriques, formelles ou semi-formelles, selon lesquelles un résultat est quantifié.*
- *$\mathcal{G}oal_{relationships} = \{relation_1, relation_2, ..., relation_n\}$ est l'ensemble des relations entre les buts. $\mathcal{G}oal_{relationships}$ = (AND/OR decomposition, Contains, Requires, Refines, Conflicts and Equivalence).*

Exemple 4

La figure 4.4 présente un modèle de buts utilisé par le concepteur Espagnol pour exprimer ses be-

soins. Considérons le besoin : "El sistema permitirá a la administración para eliminar toda la información de los estudiantes mayores de 40 años", en Français "Le système doit permettre à l'administration de supprimer les informations des étudiants ayant plus de 40 ans".

Ce but est décrit par les attributs suivants : un identifiant id (001), un nom (But G1.1), une description, un contexte (Universitaire) et une priorité. On distingue deux types de priorité : obligatoire et optionnelle.

FIGURE 4.4 – Exemple de modèle orienté buts modélisant les besoins du concepteur *Espagnol*.

Ce besoin est formalisé de la manière suivante.

- Tâches ($\mathcal{T}$) = {*Eliminar notas, Eliminar curso, Eliminar el nombre, Eliminar la fecha de nacimiento, Eliminar la edad, Calcular la edad*}, tel que les Propriétés (P) = {*notas, curso, nombre, fecha de nacimiento, edad*} sont satisfaites.
- Resultats ($\mathcal{R}$) = {*Información estudiantes eliminados*}.
- Métriques ($\mathcal{M}$) = {*Edad > 40 años*}.

3.1.2 Le modèle des cas d'utilisation

Selon la définition donnée par Jacobson, *Un cas d'utilisation est une description d'un ensemble de séquences d'actions, incluant des variantes, qu'un système effectue pour fournir un résultat observable et ayant une valeur pour un acteur*[68].

Le modèle des cas d'utilisation peut être formalisé comme suit :

Formalisation 5

UC_{model}: $< Actor, UseCase, Relationship >$, où :

- *Actor = {$actor_1$,$actor_2$,...,$actor_n$} est l'ensemble des acteurs qui interagissent avec le système pour répondre au cas d'utilisation.*
- *UseCase = {$case_1$,$case_2$,...,$case_n$} est l'ensemble des cas d'utilisation (Use cases) exprimés par un acteur (ex : le concepteur). Un cas d'utilisation est représenté comme suit : $UseCase$:$< \mathcal{A}, \mathcal{R}, \mathcal{E}p, Cdt >$, où :*
  - *$\mathcal{A} = \{a_1, a_2, ..., a_n\}$ est l'ensemble d'actions qu'un système effectue pour fournir un résultat. $\forall a_i \in \mathcal{A}, \exists p_j \subseteq \mathcal{P}$, telle que $f(a_i) = p_j$, où $p_j \in \{p_1, p_2, ..., p_n\}$ est l'ensemble des propriétés satisfaites par un système. Ici, $f(a_i) = p_j$ est une fonction qui représente l'ensemble des propriétés p_j définissant a_i.*
  - *$\mathcal{R} = \{r_1, r_2, ..., r_n\}$ est l'ensemble des résultats réalisés par le système.*
  - *$\mathcal{E}p = \{ep_1, ep_2, ..., ep_n\}$ est l'ensemble des points d'extension.*
  - *$Cdt = \{cdt_1, cdt_2, ..., cdt_n\}$ est l'ensemble des conditions.*
- *Relationships = {$relation_1$,$relation_2$,...,$relation_n$} est l'ensemble des relations entre les cas d'utilisation. Ces relations sont de trois types: (i) les relations d'inclusion, (ii) les relations d'extension et (iii) les relations de généralisation.*

Exemple 5

La figure 4.5 présente un modèle des cas d'utilisation utilisé par le concepteur Anglais pour exprimer ses besoins. Considérons le besoin: "The system shall allow the administration to delete information of all students older than 40 years", en Français "Le système doit permettre à l'administration de supprimer les informations des étudiants ayant plus de 40 ans". Ce besoin est un cas d'utilisation décrit par les attributs suivants : un identifiant id (001), un nom (Ucas1.1), une description, un contexte (Universitaire) et une priorité. On distingue deux types de priorité: obligatoire et optionnelle.

Suivant la formalisation précédente, ce besoin est décrit par :

- Action ($\mathcal{A}$) = {*Delete marks, Delete courses, Delete name, Delete birth day, Delete age, Delete weight*}, tel que les Propriétés (P) = {*marks, courses, name, birth day, age, weight*} soient satisfaites.
- Résultat ($\mathcal{R}$) = {*Students information deleted*}.
- Point d'extension $\mathcal{E}p$ = {*Calculate age*}.
- Condition Cdt = {*age > 40*}.

3.1.3 Le Modèle Conceptuel de Traitements (MCT)

Selon la définition donnée par Pierre [99], *un Modèle Conceptuel de Traitements (MCT) est une succession de traitements déclenchés par des événements et qui donnent naissance à de nouveaux événements*. Un traitement (opération) est une suite d'actions réalisées en vue d'obtenir un résultat. Dans notre travail, nous nous intéressons à l'utilisation du MCT pour l'expression des besoins utilisateurs.

Le modèle de traitements peut être formalisé comme suit :

FIGURE 4.5 – Exemple de modèle de cas d'utilisation modélisant les besoins du concepteur *Anglais*.

Formalisation 6

MCT_{model}: $< Actor, Event, Traitement, Synchronization >$, où :

- $Actor = \{actor_1, actor_2, ..., actor_n\}$ est l'ensemble des acteurs qui interagissent avec le système.
- $Event = \{event_1, event_2, ..., event_n\}$ est l'ensemble des événements.
- $Traitement = \{traitement_1, traitement_2, ..., traitement_n\}$ est l'ensemble des traitements accomplies par le système. Un traitement est représenté par le triplet : $Traitement{:}< \mathcal{A}, \mathcal{R}, \mathcal{E}mr >$, où :
  - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ est l'ensemble d'actions qu'un système exécute en réponse à un événement. $\forall a_i \in \mathcal{A}, \exists p_j \subseteq \mathcal{P}$, telle que $f(a_i) = p_j$, où $p_j \in \{p_1, p_2, ..., p_n\}$ est l'ensemble des propriétés satisfaites par un système. Ici, $f(a_i) = p_j$ est une fonction qui représente l'ensemble des propriétés p_j définissant a_i.
  - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ est l'ensemble des résultats créés par un traitement qui peut lui-même jouer le rôle d'événement.
  - $\mathcal{E}mr = \{emr_1, emr_2, ..., emr_n\}$ est l'ensemble des règles d'émission *emission rules* qui permettent d'exprimer des conditions de sortie des résultats.
- $Synchronization = \{synch_1, synch_2, ..., synch_n\}$ est l'ensemble des synchronisation (ET/OU) qui déclenchent un ou plusieurs traitements.

Exemple 6

La figure 4.6 présente un modèle conceptuel de traitements utilisé par le concepteur Français pour exprimer ses besoins. Considérons le besoin : "Le système doit permettre à l'administration de supprimer toutes les informations des étudiants ayant plus de 40 ans". Ce besoin est un événement décrit par les attributs suivants : un identifiant id (001), un nom (Event 1.1), une description, un contexte (Universitaire) et une priorité. On distingue deux types de priorité : obligatoire et optionnelle.

FIGURE 4.6 – Exemple de modèle de traitements modélisant les besoins du concepteur *Français*.

Suivant la formalisation précédente, ce besoin est décrit par :

- Action ($\mathcal{A}$) = {*Supprimer le nom, Supprimer la date de naissance, Calculer l'âge, Supprimer les notes, Supprimer les cours*}, tel que les Propriétés (P) = {*nom, date de naissance, âge, notes, cours*} soient satisfaites.
- Résultat ($\mathcal{R}$) = {*Informations des étudiants supprimées*}.
- Règle d'émission ($\mathcal{E}mr$) = {$\hat{A}ge > 40$}.

3.2 Couplage des modèles utilisés avec l'ontologie

Pour réaliser le couplage entre les trois modèles étudiés et l'ontologie, nous utilisons les techniques de la méta-modélisation. Cette dernière nous permis de définir les concepts à utiliser pour modéliser les besoins des utilisateurs. Nous proposons d'étendre le méta-modèle d'ontologie par des méta-classes constituant le méta-modèle des besoins (voir la figure 4.7). Ensuite en liant les coordonnées de cette classe avec les méta-classes ontologiques (Concepts et propriétés). Nous utilisons le terme *(Domaine)* pour désigner les ressources de l'ontologie (concepts et propriétés) qui peuvent être utilisées pour définir le besoin.

FIGURE 4.7 – Couplage des modèles des besoins avec le modèle d'ontologie.

3.2.1 Méta-modèle de l'ontologie partagée

Le standard Meta-Object Facility (MOF) de l'Object Management Group (OMG) fournit un méta-modèle OWL [31] décrivant les principaux constructeurs d'une ontologie OWL. La figure 4.8 représente un fragment de ce méta-modèle qui représente le méta-modèle de l'ontologie partagée. Dans ce méta-modèle, une ontologie est représentée principalement par les classes suivantes : *OWLClass, DataTypeProperty* et *ObjectProperty*, représentant respectivement les classes, les propriétés de type attribut et les propriétés de type relations. La subsomption des classes est décrite par la relation réflexive *subclassOf*. Différentes classes existent dans le méta-modèle OWL pour définir les opérateurs entre les classes *(IntersectionClass, UnionClass, etc.)*.

31. https://www.w3.org/2007/OWL/wiki/MOF-Based_Metamodel

FIGURE 4.8 – Méta-modèle d'ontologie OWL.

3.2.2 Couplage des trois langages proposés avec les ontologies locales

Nous avons montré comment nous avons représenté les trois méta-modèles proposés, en utilisant une approche de méta-modélisation. Nous présentons maintenant le couplage entre ces méta-modèles et les ontologies locales.

En étendant la première formalisation d'une ontologie (section 2.5 du chapitre 3), l'ontologie locale des besoins sera définie formellement de la manière suivante :
$\mathcal{OLB} :< C, R, \mathcal{R}ef(C), \mathcal{R}ef(R), \mathcal{MB}, \mathcal{F}ormalisme >$ où $\mathcal{MB}$ est le modèle des besoins définis en utilisant les concepts et propriétés ontologiques $\mathcal{MB} \in 2^C U 2^R$. Dans notre cas d'étude : $\mathcal{MB} = Goal_{model} \oplus UC_{model} \oplus MCT_{model}$ où chacun des modèles est défini comme suit. ($\oplus$ signifie OU exclusif.)

Ontologie des buts (OntoGoal). Cette ontologie consiste essentiellement à connecter les buts sur l'ontologie locale. Concrètement, cette connexion se fait par la création de nouvelles méta-classes *(Goal, Result, Metric, Task, Relationships, etc.)*, et en liant les coordonnées des buts *(Action, Result et Metric)* avec la méta-classe *rdfs:Class* de l'ontologie OWL. La figure 4.9 (a) représente un fragment du méta-modèle d'ontologie auquel est connecté le méta-modèle de buts 4.9 (b). Les buts de concepteurs sont spécifiés au niveau ontologique où nous avons défini une connexion entre les coordonnées de chaque but *(Task, Result and Metric)* les concepts et les propriétés de l'ontologie locale.

Une ontologie des buts (*OntoGoal*) représente la connexion entre le modèle de buts et le modèle ontologique. Formellement, *OntoGoal* sera définie de la manière suivante :

FIGURE 4.9 – Connexion entre le modèle ontologique et le modèle orienté buts.

Formalisation 7

OntoGoal $:< \mathcal{OL}_i, \mathcal{G}oal_{model} >$ *est l'ontologie locale des buts, où :*

- $\mathcal{OL}_i : < C_i, R_i, Ref(C_i), Ref(R_i), F_i >$ *est l'ontologie locale.*
- $\mathcal{G}oal_{model}$: $< \mathcal{A}ctor, \mathcal{G}oal, \mathcal{R}elationship >$, *où :*
  - $\mathcal{G}oal$:$< \mathcal{T}, \mathcal{R}, \mathcal{M} >$, *avec :*
    - $\mathcal{T} = \{t_1, t_2, ..., t_n\}$ *est l'ensemble de tâches .* $\forall t_i \in \mathcal{T}$, $\exists d_j \subseteq \mathcal{D}$, *telle que* $f(t_i) = d_j$, *où* $d_j \in (Domaine(t))$ *est l'ensemble des ressources de l'ontologie de domaine.*
    - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ *est l'ensemble de résultats.*
    - $\mathcal{M} = \{m_1, m_2, ..., m_n\}$ *est l'ensemble de métriques.*
  - $\mathcal{R}elationships = \{relation_1, relation_2, ..., relation_n\}$ *est l'ensemble des relations entre les buts.* $relation_i \in \mathcal{R}elationships \subseteq R_i$, *l'ensemble de relations (rôles) de l'ontologie locale.*

La figure 4.10 illustre un exemple d'instanciation de modèle d'ontologie (partie (a)) et le modèle orienté but (partie (b)). Le concepteur peut exprimer le but : *"Le système doit permettre à l'administration de supprimer les informations des étudiants ayant plus de 40 ans"* au niveau ontologique en utilisant les propriétés et les concepts de l'ontologie. Par exemple, les propriétés *age, name, marks* sont des propriétés ontologiques. **Ontologie des cas d'utilisation (OntoUcase)**. Les besoins de concepteurs modélisés par les cas d'utilisation sont spécifiés au niveau ontologique où nous avons défini une connexion entre les coordonnées de chaque cas d'utilisation *(Action, Résultat, Condition et Point d'extension)* et les concepts et propriétés de l'ontologie locale (voir la figure 4.11). Une ontologie des cas d'utilisation (*OntoUcase*)

FIGURE 4.10 – Instanciation des modèles de l'ontologie et du modèle orientés buts.

représente la connexion entre le modèle des cas d'utilisation et le modèle ontologique.

Formellement, $OntoUcase$ est définie de la manière suivante.

Formalisation 8

$OntoUcase :< \mathcal{OL}_i, \mathcal{UC}_{model} >$ *est l'ontologie locale des cas d'utilisation, où :*

- $\mathcal{OL}_i : < C_i, R_i, Ref(C_i), Ref(R_i), F_i >$ *est l'ontologie locale.*
- $\mathcal{UC}_{model}$: $< \mathcal{A}ctor, \mathcal{U}seCase, \mathcal{R}elationship >$, *avec :*
  - $\mathcal{U}seCase$:$< \mathcal{A}, \mathcal{R}, \mathcal{E}p, \mathcal{C}dt >$, *où :*
    - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ *est l'ensemble des actions.* $\forall a_i \in \mathcal{A}$, $\exists d_j \subseteq \mathcal{D}$, *telle que* $f(a_i) = d_j$, *où* $d_j \in (Domaine(a))$ *est l'ensemble des ressources de l'ontologie de domaine.*
    - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ *est l'ensemble des résultats.*
    - $\mathcal{E}p = \{ep_1, ep_2, ..., ep_n\}$ *est l'ensemble des points d'extension.*
    - $\mathcal{C}dt = \{cdt_1, cdt_2, ..., cdt_n\}$, *est l'ensemble des métriques.*
- $\mathcal{R}elationships = \{relation_1, relation_2, ..., relation_n\}$ *est l'ensemble des relations entre les cas d'utilisation.* $relation_i \in \mathcal{R}elationships \subseteq R_i$, *l'ensemble de relations (rôles) de l'ontologie locale.*

FIGURE 4.11 – Connexion entre le modèle ontologique et le modèle use case.

Ontologie des traitements (OntoMCT). Les besoins de concepteurs modélisés par les traitements sont spécifiés au niveau ontologique où nous avons défini une connexion entre les coordonnées de chaque traitement *(Action, Résultat et Règles d'émission)*, les concepts et les propriétés de l'ontologie locale (voir la figure 4.12). Une ontologie de traitements (*OntoMct*) représente la connexion entre le modèle conceptuel de traitements et le modèle ontologique.

Formellement, *OntoMCT* est définie de la manière suivante :

Formalisation 9

OntoMCT $:< \mathcal{OL}_i, \mathcal{MCT}_{model} >$ *est l'ontologie locale des traitements, avec :*

- $\mathcal{OL}_i : < C_i, R_i, Ref(C)_i, Ref(R)_i, F_i >$ *est l'ontologie locale.*
- $\mathcal{MCT}_{model}: < \mathcal{A}ctor, \mathcal{E}vent, \mathcal{T}raitement, \mathcal{S}ynchronization >$, *où :*
  - $\mathcal{E}vent = \{event_1, event_2, ..., event_n\}$ *est l'ensemble des événements. Pour chaque* $event \in \mathcal{E}vent$, $event \in 2^C U 2^R$
  - $\mathcal{T}raitement: < \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{E}mr >$, *où :*
    - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$, *l'ensemble des actions.* $\forall a_i \in \mathcal{A}, \exists d_j \subseteq \mathcal{D}$, *telle que* $f(a_i) = d_j$, *où* $d_j \in (Domaine(a))$ *est l'ensemble des ressources de l'ontologie de domaine.*
    - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ *l'ensemble des résultats.*
    - $\mathcal{E}mr = \{emr_1, nemr_2, ..., emr_n\}$ *est l'ensemble des règles d'émission.*

FIGURE 4.12 – Connexion entre le modèle ontologique et le modèle MCT.

– *Synchronization* = $\{synch_1, synch_2, ..., synch_n\}$, *est l'ensemble des relations de sychronisation entre les événements.* $synch_i \in \mathcal{R}elationships \subseteq R_i$, *l'ensemble de relations (rôles) de l'ontologie locale.*

Pour résumer, nous avons établi une connexion entre l'ontologie de domaine et les trois langages de modélisation des besoins. Cela réduit considérablement l'hétérogénieté syntaxique et sémantique.

4 Unification des langages de modélisation des besoins

Comme nous l'avons indiqué, la solution pivot obtenue par des exemples est proposée pour unifier l'ensemble des langages de modélisation des besoins. Le modèle pivot est obtenu après l'analyse des concepts fondamentaux des trois langages étudiés. Soit $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_n\}$, l'ensemble des concepteurs *(Designers)* impliqués dans le développement d'une application donnée. Chaque concepteur $\mathcal{D}_i$ doit utiliser un langage de modélisation des besoins $\mathcal{LB}_i$ et une terminologie (vocabulaire) $\mathcal{V}_i$ pour exprimer ses besoins $\mathcal{B}^{D_i}$. Pour intégrer les différents langages des besoins, une solution naïve consiste à établir $(\frac{n\times(n-1)}{2})$ correspondances entre les n langages. Cette solution est coûteuse, car elle nécessite des efforts considérables pour établir les correspondances entre les différents langages des besoins, notamment si les langages utilisés sont nombreux. Pour réduire cette complexité, le langage pivot est approprié. Du

fait qu'il offre une représentation générique de différents langages. Plus précisément, lorsque un langage pivot est utilisé, le nombre de correspondance est réduit à n si n langages sont pris en considération. Les différentes correspondances établies entre les langages sont illustrées par la figure 4.13.

FIGURE 4.13 – Intégration centralisée avec langage Pivot.

Formellement, selon une approche pivot, un système d'intégration des besoins hétérogènes *RIS (Requirements integration system)* est représenté par un triplet :

Formalisation 10

RIS $:< \mathcal{P}; \mathcal{LB}; \mathcal{M} >$ *avec :*

- $\mathcal{P}$ *: un schéma pivot (générique) exprimé par un langage (modèle) de modélisation donné* $\mathcal{LB}^P$ *sur une terminologie (vocabulaire)* $\mathcal{V}^{\mathcal{P}}$*;*
- $\mathcal{LB} = \{\mathcal{LB}_1, \mathcal{LB}_2, ..., \mathcal{LB}_n\}$*, l'ensemble de langages de modélisation des besoins utilisés par des concepteurs hétérogènes impliqués dans le développement du système complexe. Chaque* $\mathcal{LB}_i$ $(1 \leq i \leq n)$ *doit utiliser une teminologie* $\mathcal{V}^{\mathcal{LB}}$*;*
- $\mathcal{M} = \{\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_n\}$*, l'ensemble des correspondances (mapping) entre les langages de modélisation* $\mathcal{LB}_i$ *et le langage pivot* $\mathcal{P}$*. Cela conduit à* $[n]$ *correspondances si* n *langages sont pris en considération.*

4.1 Proposition d'un modèle pivot des besoins

Le Tableau 4.1 présente une comparaison entre les trois modèles des besoins (Goal-Oriented, Use case (UML) et MCT de MERISE) pour extraire les unités de modélisation (les concepts) communs. Cette comparaison nous a permis d'élaborer le modèle pivot qui est défini comme suit et illustré par la figure 4.14.

Requirements models	Concepts		Pivot model				
			Actor	Requirement			Relationships
				Action	*Criterion*	*Result*	
Goal-Oriented	Actor		✓				
	Goal	*Task*		✓			
		Metric			✓		
		Result				✓	
	AND/OR						✓
	Contribution Relationships						✓
Use Case (UML)	Actor		✓				
	Use case	*Action*		✓			
		Result				✓	
		Extension point					
		Condition			✓		
	Generalization						✓
	Include						✓
	Extend						✓
MCT (MERISE)	Actor		✓				
	Event						
	Treatment	Action		✓			
		Result				✓	
		Emission rules			✓		
	Synchronization						✓

TABLE 4.1 – Les concepts communs entre les modèles de besoins.

Formalisation 11

$\mathcal{P}ivot_{model}$: $< \mathcal{A}ctor, \mathcal{R}equirement, \mathcal{R}elationships >$, où :

- $\mathcal{A}ctor = \{actor_1, actor_2, ..., actor_n\}$ un ensemble d'acteurs (e.g. concepteur)
- $\mathcal{R}equirement = \{req_1, req_2, ..., req_n\}$ un ensemble de besoins exprimés par un acteur. Nous définissons un besoin (requirement) comme suit : $\mathcal{R}equirement_i : < \mathcal{A}, \mathcal{R}, \mathcal{C} >$ un ensemble de besoins où :
  - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ est l'ensemble d'actions qu'un système effectue pour fournir un résultat. $\forall a_i \in \mathcal{A}, \exists p_j \subseteq \mathcal{P}$, telle que $f(a_i) = p_j$, où $p_j = \{p_1, p_2, ..., p_n\}$ est l'ensemble des propriétés satisfaites par un système.
  - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ est l'ensemble des résultats réalisés par le système.
  - $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ est l'ensemble des critères selon lesquels un résultat est quantifié.
- $\mathcal{R}elationships = \{relation_1, relation_2, ..., relation_n\}$ est l'ensemble de relations entre les besoins.

4.2 Couplage de modèle pivot à l'ontologie

Après la construction du méta-modèle pivot, on relie les caractéristiques de chaque concept *(Action, Résultat, Critère, etc.)* avec des classes et des propriétés ontologiques. Un modèle ontologique pivot (*OntoPivot*) représente la connexion entre le modèle pivot et le modèle de l'ontologie globale (figure

FIGURE 4.14 – Le méta-modèle pivot.

4.15).

Formellement, *OntoPivot* est définie de la manière suivante :

Formalisation 12

OntoPivot $:< \mathcal{OP}, \mathcal{P}ivot_{model} >$ *est ontologie des besoins, où :*

- $\mathcal{OP} : < C, Ro, Ref(C), Ref(R), F >$ *est l'ontologie partagée.*
- $\mathcal{P}ivot_{model}$: $< \mathcal{A}ctor, \mathcal{R}equirement, \mathcal{R}elationships >$, *avec :*
  - $\mathcal{R}equirement$:$< \mathcal{A}, \mathcal{R}, C >$, *où:*
    - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ *est l'ensemble des actions.* $\forall a_i \in \mathcal{A}, \exists d_j \subseteq \mathcal{D}$, *telle que* $f(a_i) = d_j$, *où* $d_j \in (Domaine(a))$ *est l'ensemble des ressources de l'ontologie de domaine.*
    - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ *est l'ensemble des résultats.*
    - $C = \{c_1, c_2, ..., c_n\}$ *est l'ensemble des critères.*
  - $\mathcal{R}elationships = \{relation_1, relation_2, ..., relation_n\}$ *est l'ensemble des relations entre les besoins.* $relation_i \in \mathcal{R}elationships \subseteq R_o$ *est l'ensemble de relations de l'ontologie de domaine.*

FIGURE 4.15 – Connexion entre le modèle ontologique et le modèle pivot.

5 Un exemple d'utilisation de notre méthode d'intégration

Pour faciliter la compréhension de notre approche, nous considérons le scénario d'utilisation indiqué par la figure 4.16, où les trois langages de modélisation des besoins sont considérés.

Chaque concepteur extrait d'abord une ontologie locale référençant l'ontologie partagée qui satisfait ses exigences. Une fois l'ontologie locale extraite, le concepteur décrit ses besoins en fonction de celle-ci. Ces besoins sont ensuite transformés à l'aide des règles de transformation au modèle pivot.

Nous rappelons que la transformation de modèles fait partie de l'approche IDM. Elle permet d'obtenir différents modèles cibles suite au traitement de différents modèles sources (figure 4.17). La transformation de modèle est également utilisée pour établir les correspondances *(mapping)* et les traductions entre différents langages de modélisation.

Dans ce scénario, nous transformons les langages des besoins locaux (Ontologie Orienté-buts (OntoGoal), Ontologie des Cas d'utilisation (OntoUcase) et Ontologie des traitements (OntoMCT)) en un langage commun (Ontologie Pivot, OntoPivot). Cette transformation se fait par l'intermédiaire des fonctions qui décrivent les règles de transformation suivantes.

FIGURE 4.16 – Scénario d'intégration des besoins selon un langage pivot.

```
Transf 1 : Goal -----> Requirement.

Transf 2 : UseCase -----> Requirement.

Transf 3 : Treatment -----> Requirement.

Transf 4 : Metric -----> Criterion.

Transf 5 : Condition -----> Criterion.

Transf 6 : Emission-rule -----> Criterion.
```

Exemple 7

Goal001 (Méta-modèle OntoGoal) —> Requirement001 (Méta-modèle Ontologique Pivot).
UseCase001 (Méta-modèle OntoUcase) —> Requirement002 (Méta-modèle Ontologique Pivot).
Treatement001 (Méta-modèle OntoMCT) —> Requirement003 (Méta-modèle Ontologique Pivot).
Task01 (Méta-modèle OntoGoal) —> Action (Méta-modèle Ontologique Pivot).
Condition001 (Méta-modèle OntoUcase) —> Criteria001 (Méta-modèle Ontologique Pivot).
Metric002 (Méta-modèle OntoGoal) —> Criteria0012 (Méta-modèle Ontologique Pivot).
Emission-rule003 (Méta-modèle OntoMCT) —> Criteria003 (Méta-modèle Ontologique Pivot).

FIGURE 4.17 – Principe de transformation.

6 Conclusion

Dans ce chapitre, nous avons proposé une solution d'intégration de besoins hétérogènes, unifiant les vocabulaires et les langages de modélisation des besoins utilisés. Pour unifier les vocabulaires, nous avons proposé l'utilisation d'une ontologie de domaine supposée existante. Dans ce cas, chaque concepteur référence cette ontologie pour exprimer ses besoins. Pour unifier les langages de modélisation des besoins, nous avons étudié trois langages à savoir, Use case d'UML, le langage orienté buts, et le modèle de traitement de la méthode Merise. Cette étude nous a permis de définir un langage pivot factorisant l'ensemble de langages étudiés. Un couplage entre l'ensemble des méta-modèles utilisés et l'ontologie est établi ainsi qu'un scénario d'intégration des besoins est alors défini selon les spécifications de ce langage pivot. Chaque concepteur extrait d'abord une ontologie locale référençant l'ontologie partagée qui satisfait ses exigences. Une fois extraite, il décrit ses besoins en fonction de celle-ci. Les besoins sont ensuite transformés à l'aide des règles sur le modèle pivot. Dans ce scénario, nous avons utilisé des transformations de type *Model-to-Model*.

Dans ce chapitre, nous avons constaté que l'intégration des différents langages de modélisation des besoins selon la solution du langage pivot devient complexe et coûteuse si le nombre de langages de modélisation des besoins augmente. Donc, elle nécessite des efforts considérables pour établir les correspondances entre les différents langages des besoins et le langage pivot via des transformation *Model-to-Model*. Pour réduire cette complexité, une deuxième solution d'intégration qui repose sur la définition d'un méta-modèle générique (unifié) représentant les besoins est proposée dans le chapitre suivant.

Chapitre 5

Vers une Fusion de méta-modèles des langages de modélisation des besoins

Sommaire

1 Introduction 91
2 Fusion des méta modèles 91
3 Connexion du méta-modèle générique avec l'ontologie 93
4 Raisonnement sur les besoins 94
4.1 Scénarii de raisonnement 96
4.1.1 Scénario 1 : *ship whole* 96
4.1.2 Scénario 2 : *reason as needed* 97
4.2 Étude de cas 98
4.3 Relations sémantiques entre les besoins : Définition & Formalisation . . . 99
4.4 Évaluation du processus de raisonnement 102
5 Conclusion 103

Résumé. Dans le chapitre précédent, nous avons proposé une approche d'intégration des langages de besoins basée sur un modèle pivot défini par un méta-modèle. L'inconvénient majeur de cette approche est le nombre de transformations faites du nombre de n, si n langages sont considérés. Pour réduire cette complexité, nous proposons une autre approche sémantique consistant à fusionner l'ensemble des langages de besoins (trois dans notre cas). Cette fusion donnera lieu à un modèle appelé *générique*, où chaque langage devient une instance de ce dernier. Une mécanisme de raisonnement est fourni afin d'identifier les besoins conflictuels et contradictoires d'une manière efficace [26].

1 Introduction

Afin de réduire la complexité de la solution d'intégration proposée dans le chapitre précédent, nous proposons une nouvelle approche dirigée par la fusion des méta-modèles de langages de modélisation utilisés (figure 5.1). Le méta-modèle générique est également couplé à l'ontologie partagée. Au lieu de faire des transformations de type *modèle-à-modèle*, des instanciations sont faites. Dans cette approche, un concepteur donne seulement le nom de son langage d'expression de besoins et son ontologie locale et une instanciation est générée. Pour identifier les besoins contradictoires, un processus de raisonnement

FIGURE 5.1 – Méta-modèle générique fusionnant les trois modèles étudiés.

est effectué en exploitant la présence des ontologies (locales et partagée).

Ce chapitre est organisé comme suit. Nous présentons d'abord une notre approche dirigée par la fusion des méta-modèles est proposée. Ensuite un mécanisme de raisonnement sur les besoins est proposé, incluant deux stratégies de raisonnement. Ces dernières sont implémentées et comparées en termes de temps de réponse.

2 Fusion des méta modèles

Dans le chapitre précédent, nous avons proposé un méta-modèle pivot qui contient un ensemble de concepts communs entre les méta-modèles étudiés (orientés buts, cas d'utilisation et modèle de traitements). Le passage d'un de ces derniers vers notre méta-modèle pivot requiert des transformations, ainsi qu'une connaissance profonde du notre méta-modèle pivot. La solution que nous proposons dans ce chapitre est basée sur la fusion de l'ensemble des méta-modèles étudiés. Cette fusion est représentée par un méta-modèle générique qui va permettre aux concepteurs d'éviter le processus de transformation. En outre, les concepts présentés précédemment dans le méta-modèle pivot feront aussi partie de ce méta-modèle car chaque concept pivot va représenter un concept générique au sein du nouveau méta-modèle. Chaque concept générique (super-classe) supervise les concepts correspondants des différents méta-modèles. En conséquence, l'instanciation de notre méta-modèle générique donne lieu à un modèle regroupant les besoins exprimés avec les différents langages. En effet, chaque instance est un ensemble de besoins, où chaque besoin peut être exprimé par un des langages étudiés.

La figure 5.2 présente notre méta-modèle générique fusionnant les trois méta-modèles de besoins développés dans le chapitre précédent. Chaque besoin *(Requirement)* est caractérisé par un identifiant *(IdReq)*, un nom *(NameReq)*, une description textuelle *(DescriptionReq)*, un objet *(PurposeReq)* , un contexte *ContextReq*, une priorité *PriorityReq* (en prenant les valeurs : Mandatory, High, Medium)).

FIGURE 5.2 – Métamodèle générique des besoins.

Un besoin est décrit par les caractéristiques suivantes.

- *Résultat (Result)* : il représente la finalité du besoin d'une manière générale. Par exemple le besoin suivant *"Une entreprise a besoin d'un nombre prédéfini d'employés"*, le nombre prédéfini d'employés constitue le résultat du besoin. *Result* possède une propriété *measure* qui sert à quantifier (mesurer) le résultat à l'aide des fonctions statistiques dédiées (Sum, Max, Min, AVG, etc).
- *Action* : elle reflète un ensemble de séquences d'actions qui permettent de réaliser un besoin.
- *Critère (Criterion)* : il permet de filtrer éventuellement un besoin en lui associant des conditions à satisfaire, Ainsi, pour l'exemple précédent : *Une entreprise a besoin d'un nombre prédéfini d'employés ne dépassant pas un certain âge"*, le critère précisé ici, est que l'âge des employés doit être inférieur à un seuil donné.
- *Type* : il permet de spécialiser un besoin. Dans notre étude de cas un besoin peut être un but *(Goal)*, un cas d'utilisation *(Use case)* ou un traitement *(Treatement)*.
- *Acteur* : chaque besoin est proposé par un *Acteur* (personne, entreprise, ou unité de système autonome).

– *Relations (Relationships)* : les besoins peuvent être liés les uns aux autres à travers des relations *(Requires, Conflicts, Contains, Equivalence, AND/OR, etc.)*.

Deux catégories de besoins sont identifiées : les besoins fonctionnels et non-fonctionnels.

Notons que les coordonnées d'un besoin à savoir : *Result*, *Action*, *Criterion* portent sur des objets cibles qui dépendent du contexte d'usage. Dans le contexte des bases de données sémantiques, ces objets peuvent être représentés par des *Concepts* dans PLIB, *rdfs:Resources* dans RDF, *OWLClasses* dans OWL, etc. Nous allons détailler ce point dans la section suivante expliquant la connexion entre le modèle des besoins et le modèle de l'ontologie.

Formellement, le méta modèle générique est défini de la manière suivante.

Formalisation 13

$\mathcal{G}eneric_{model}$: $< \mathcal{A}ctor, \mathcal{R}equirement, \mathcal{R}elationships >$, où :

- $\mathcal{A}ctor = \{actor_1, actor_2, ..., actor_n\}$ est l'ensemble d'acteurs (e.g. concepteur)
- $\mathcal{R}equirement = \{req_1, req_2, ..., req_n\}$ est l'ensemble de besoins exprimés par un acteur. Nous définissons un besoin (requirement) comme suit: $\mathcal{R}equirement_i$: $< \mathcal{A}, \mathcal{R}, \mathcal{C}, \mathcal{T} >$, $(1 \leq i \leq n)$ est un ensemble de besoins où :
  - $\mathcal{A} = \{a_1, a_2, ..., a_n\}$ est l'ensemble de séquences d'actions qu'un système effectue pour fournir un résultat. $\forall a_i \in \mathcal{A}, \exists p_j \subseteq \mathcal{P}$, telle que $f(a_i) = p_j$, où $p_j \in \{p_1, p_2, ..., p_n\}$ est l'ensemble des propriétés satisfaites par un système.
  - $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ est l'ensemble des résultats réalisés par le système.
  - $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ est l'ensemble des critères selon lesquels un résultat est quantifié.
  - $\mathcal{T} = \{t_1, t_2, ..., t_n\}$, est l'ensemble des types de besoins. Dans notre cas d'étude, $t_i \in \{Goal, Usecase, Treatment\}$.
- $\mathcal{R}elationships = \{relation_1, relation_2, ..., relation_n\}$, est l'ensemble de relations entre les besoins. Dans notre cas, $\mathcal{R}elationship = \{\mathcal{G}oal_{relationships} \cup \mathcal{U}seCase_{relationships} \cup \mathcal{M}CT_{relationships}\}$ = (Contains, Requires, Refines, Conflicts, Equivalence, AND/OR decomposition, Generalization, Synchronization, etc.).

3 Connexion du méta-modèle générique avec l'ontologie

Cette étape consiste essentiellement à étendre l'ontologie globale par des méta-classes constituant le méta-modèle générique de besoins. Ensuite en liant les coordonnées de chaque besoin (Requirement) *Action*, *Result* et *Criterion* avec des classes et propriétés ontologiques. (voir la figure 5.3)

Dans la pratique, ces coordonnées peuvent porter non seulement sur un seul objet, mais également sur des expressions (entre autres relations arithmétiques) entre plusieurs objets cibles. Dans la figure 4.10 (section 3.2.2 du chapitre 4), le résultat du besoin porte sur un seul objet qui est la classe ontologique *Student*, comme il peut bel et bien porter sur une expression entre propriétés ontologiques, comme par exemple le besoin du calcul de la moyenne des étudiants où on aura affaire à un produit entre la propriété *note* et *coefficient* de la classe *Course*. D'autres exemples sont étudiés plus loin dans le chapitre 6.

Au final , les besoins sont spécifiés au niveau ontologique où nous avons substitué les classes du modèle des besoins par son équivalent dans le modèle d'ontologie. Un modèle ontologique (OntoReq)

FIGURE 5.3 – Connexion entre le méta-modèle ontologique et le méta-modèle générique des besoins.

représente cette connexion.

$$OntoReq = \prod_{Requirements} (OG)$$

Maintenant nous avons tous les ingrédients pour intégrer les besoins. En suivant les mêmes étapes du scénario d'intégration présenté dans le chapitre 4. Ici, chaque concepteur extrait d'abord une ontologie locale référençant l'ontologie partagée qui satisfait ses exigences. L'ontologie locale extraite représente le vocabulaire partagé et le langage d'expression demandé par le concepteur. Une fois l'ontologie locale est extraite, le concepteur décrit ses besoins en fonction de son ontologie locale. Ensuite, ses besoins sont transformés à l'aide des règles de transformation au modèle générique afin d'établir le mécanisme de raisonnement.

4 Raisonnement sur les besoins

Dernièrement, l'inférence à base ontologique apparaît comme un moyen pertinent pour améliorer la gestion de l'information au cours de cycle de vie des applications informatiques. Les ontologies permettent des raisonnements automatiques ayant pour objet soit d'effectuer des vérifications de consistance, soit d'inférer de nouveaux faits. Fankam.[37] décrit, qu'il existe deux types d'ontologies : les ontologies

de stockage *(Storage ontologies)* et les ontologies d'inférence *(Inference ontologies)*. Les ontologies de stockage sont utilisées pour la capture de l'information, le stockage, la classification et la réutilisation des sources hétérogènes. Contrairement au premier type, les ontologies d'inférence peuvent raisonner et inférer sur des informations. L'inférence est la capacité de faire des déductions sur les instances, les classes et les propriétés ontologiques. Ce mode de raisonnement est effectué par un moteur d'inférence. De nombreux langages et outils informatiques sont apparus pour un raisonnement à base ontologique. Nous pouvons citer *OWL* et *SWRL*.

Le raisonnement, en générale, est un processus cognitif permettant soit d'obtenir de nouveaux résultats soit de faire la vérification d'un fait. Le raisonnement à base de règles *(rules based reasonning)* s'inscrit dans une logique mathématique, en utilisant la déduction, l'abduction ou l'induction. Il présente certains avantages : tout d'abord, il est facile à comprendre et à interpréter. De plus, il est assez naturel car l'être humaine raisonne souvent sous forme de règles : *(s'il pleut dehors, alors je vais prendre mon parapluie)*. Un autre atout de ce type de raisonnement est sa modularité, en effet, il est possible d'ajouter des règles ou d'en enlever simplement selon les besoins de l'utilisateur [72]. Pour raisonner à base de règles, il faut :

- Une base de connaissance : les règles. Une règle est une connaissance qui sert à faire le lien entre des connaissances connues et d'autres connaissances que l'on peut déduire ou inférer. Une règle est une expression de la forme : "Si X est Y, alors V est Z ". Elle exprime une relation entre les variables X et Y. On appelle *"X est Y"* la prémisse de la règle, et *"V est Z"* la conclusion de la règle. En général, on estime que la connaissance exprimée par cette règle est valide. Le type de raisonnement utilisé est donc déductif car on essaye de déduire des connaissances sur les valeurs de sortie à partir des valeurs des entrées.
  Exemple. Cette règle modélise la connaissance suivante : *(si un étudiant X est inscrit à un cours Y alors il existe un enseignant Z qui enseigne Y et qui enseigne à X)*. La traduction logique de cette règle (R) est: $\phi(R) = \forall x \forall y((Etudiant(x) \wedge Cours(y) \wedge inscrit(x,y)) \rightarrow \exists z(Enseigant(z) \wedge enseigne(z,y) \wedge enseigneA(z,x)))$
- Un ensemble de faits : les données.
- Un moteur d'inférence : un processus de raisonnement qui s'appuie sur des connaissances acquises, et qui s'articule autour de règles fondamentales pour permettre d'obtenir de nouvelles connaissances. Pour simplifier, on peut considérer que l'inférence est un mode de raisonnement.

Le couplage entre le modèle des besoins et l'ontologie permet de détecter les besoins incohérents et contradictoires. Le concepteur peut spécifier les besoins comme des individus (instances) dans l'ontologie. Les règles de vérification de la cohérence *(consistency checking rules)* sont exécutées sur cette ontologie (Fig 5.4).

Plus formellement notre mécanisme de raisonnement *OntoRR, (Ontological Reasoning about Requirements)* est définit comme suit :

Formalisation 14

OntoRR: $< O, \mathcal{R}equirement_{model}, \mathcal{R}ules >$, *où :*

- $O : < C, R, Ref(C), Ref(R), F >$, *an ontology.*
- $\mathcal{R}equirement_{model} = \{model_1, model_2, ..., model_n\}$, *est l'ensemble de modèles. Dans notre cas,* $\mathcal{R}equirement_{model} = Pivot_{model} \oplus Goal_{model} \oplus Ucase_{model} \oplus MCT_{model}$.
- $\mathcal{R}ules = \{rule_1, rule_2, ..., rule_n\}$, *l'ensemble des relations entre les besoins. Où* $rule_i \subseteq \mathcal{R}ules$,

FIGURE 5.4 – Processus de raisonnement.

$\mathcal{R}ules = \{Contains_{rule}, Requires_{rule}, Refines_{rule}, Conflicts_{rule}$
$andEquals_{rule}\}$.

4.1 Scénarii de raisonnement

La présence de la connexion entre le modèle de l'ontologie et le modèle des besoins nous permet de proposer deux scénarii de raisonnement.

4.1.1 Scénario 1 : *ship whole*

Dans ce scénario appelé *ship whole*, chaque concepteur envoie ses besoins sans aucun traitement local au modèle générique qui effectue le raisonnement pour identifier les besoins conflictuels. Cette phase est réalisée après la transformation des besoins locaux en modèle générique. Cette solution peut être coûteuse si le nombre de besoins est important.(figure 5.5).

Ce scénario est défini formellement par :

Formalisation 15

$OntoRU : < OntoReq, \mathcal{R}ules >$, où :

- $OntoReq :< \mathcal{OP}, \mathcal{G}eneric_{model} >$, ontologie des besoins.
- $\mathcal{R}ules = \{rule_1, rule_2, ..., rule_n\}$, l'ensemble des relations entre les besoins. Où $rule_i \subseteq \mathcal{R}ules$, $\mathcal{R}ules = \{Contains_{rule}, Requires_{rule}, Refines_{rule}, Conflicts_{rule} andEquals_{rule}\}$.

FIGURE 5.5 – Scénario (1) : ship whole.

4.1.2 Scénario 2 : *reason as needed*

Pour pallier le problème de coût, nous proposons l'architecture *reason as needed* qui permet à chaque concepteur de raisonner localement autant qu'il le peut, puis d'envoyer ensuite des besoins valides au modèle générique, qui à son tour effectue d'autres raisonnements.(figure 5.6).

Dans notre étude de cas, le raisonnement est effectué localement sur les ontologies locales des besoins *(OntoGoal,OntoUcase* et *OntoMCT)* et d'une manière globale après la phase de transformation sur l'ontologie de besoins globale *(OntoReq)*.

Ce scénario est défini formellement par :

1. Sur l'ontologie de buts, *OntoGoal*

Formalisation 16

OntoRG: $< OntoGoal, \mathcal{R}ules >$, *où :*

- $OntoGoal :< O\mathcal{L}_i, \mathcal{G}oal_{model} >$, *l'ontologie locale des buts.*
- $\mathcal{R}ules = \{rule_1, rule_2, ..., rule_n\}$, *l'ensemble des relations entre les buts. Où* $rule_i \subseteq \mathcal{R}ules$, $\mathcal{R}ules = \{Contains_{rule}, Requires_{rule}, Refines_{rule}, Conflicts_{rule} and Equals_{rule}\}$.

FIGURE 5.6 – Scénario (2) : reason as needed.

2. Sur l'ontologie des cas d'utilisation, *OntoUcase*

Formalisation 17

$OntoRU: < OntoUcase, \mathcal{R}ules >$, où :

- $OntoUcase :< \mathcal{OL}_i, \mathcal{UC}_{model} >$, l'ontologie locale des cas d'utilisation.
- $\mathcal{R}ules = \{rule_1, rule_2, ..., rule_n\}$, l'ensemble des relations entre les besoins. Où $rule_i \subseteq \mathcal{R}ules$, $\mathcal{R}ules = \{Contains_{rule}, Requires_{rule}, Refines_{rule}, Conflicts_{rule} and Equals_{rule}\}$.

3. Sur l'ontologie des traitement, *OntoMCT*

Formalisation 18

$OntoRU: < OntoMCT, \mathcal{R}ules >$, où :

- $OntoMCT :< \mathcal{OL}_i, \mathcal{MCT}_{model} >$, l'ontologie locale des traitements
- $\mathcal{R}ules = \{rule_1, rule_2, ..., rule_n\}$, l'ensemble des relations entre les besoins. Où $rule_i \subseteq \mathcal{R}ules$, $\mathcal{R}ules = \{Contains_{rule}, Requires_{rule}, Refines_{rule}, Conflicts_{rule} and Equals_{rule}\}$.

Les deux scénarii s'appuient sur les relations sémantiques définis dans la section 4.3.

4.2 Étude de cas

Pour évaluer notre processus de raisonnement, nous proposons l'utilisation d'un ensemble de besoins sur le domaine universitaire, avec des besoins concernant *les étudiants, les départements, les cours, etc.*

Cet ensemble est sélectionné du document de besoins concernant le système gestion des cours (CMS, Course Management System) [32]. L'ensemble sélectionné contient 60 besoins. Nous supposons qu'ils sont collectés auprès de trois concepteurs différents qui utilisent nos modèles : 25 par le modèle *(Goal)*, 15 par *(Use case)* et 20 par *MCT*. Nous avons utilisé l'ontologie LUBM contenant 45 classes et 38 propriétés (dont 31 *Object properties*, et 7 *Data type properties*). Nous avons sélectionné 10 besoins $(R_1, R_2, ..., R_{10})$ de ce document, pour bien comprendre la définition et la formalisation des relations entre les besoins. Les besoins qui suit est une partie de ce document.

- R_1: The system shall allow the department to manage courses.
- R_2: The system shall allow the department to create courses.
- R_3: The system shall provide database course information.
- R_4: The system shall be able to store database course information.
- R_5: The system shall allow department to delete all studies and personal information of students more than 40 years old.
- R_6: The system shall allow department to delete all studies and personal information of students less than 40 years old.
- R_7: The system shall allow department to create entirely new courses.
- R_8: The system shall allow department to create courses.
- R_9: The system shall allow lecturers to create courses.
- R_{10}: The system shall allow lecturers to create courses.

4.3 Relations sémantiques entre les besoins : Définition & Formalisation

Nous avons identifié cinq types de relations : *Contains, Requires, Conflicts, Refines et Equals*. Nous utilisons les définitions informelles définies par [55]. Dans le reste de cette section, nous donnons une définition formelle de ces relations. Nous utilisons le terme "domaine" pour désigner les ressources de l'ontologie (concepts et propriétés) qui peuvent être utilisés pour définir le besoin.

1. **Relation "Contains"** :

Définition 1

Un besoin B_1 contient des besoins $\{B_2 ... B_n\}$ si $\{B_2 ... B_n\}$ font parties de l'ensemble B_1 (part-whole hierarchy). Cette relation permet de décomposer un besoin complexe en plusieurs parties. La relation contains est non réflexive, non symétrique et transitive.

Soit: B_1:$< \mathcal{A}_1, \mathcal{R}_1, \mathcal{M}_1 >$, B_2:$< \mathcal{A}_2, \mathcal{R}_2, \mathcal{M}_2 >$,...,$B_n$:$< \mathcal{A}_n, \mathcal{R}_n, \mathcal{M}_n >$ des besoins où n $\geq$ 2.

Domain(B_1)=domain ($\mathcal{A}_1$)$\cup$ domain ($\mathcal{R}_1$) $\cup$ domain ($\mathcal{M}_1$)

Domain(B_2)= domain ($\mathcal{A}_2$)$\cup$ domain ($\mathcal{R}_2$)$\cup$ domain ($\mathcal{M}_2$)

Domain(B_n)= domain ($\mathcal{A}_n$)$\cup$ domain ($\mathcal{R}_n$)$\cup$ domain ($\mathcal{M}_n$)

Formalisation 19

*$Contains_{rule}$:B_1 **Contains** B_2,B_3,...,B_n: domain(B_2) $\subset$ domain(B_1), domain(B_3) $\subset$ domain(B_1),..., domain(B_n) $\subset$ domain(B_1)*

32. http://wwwhome.cs.utwente.nl/ goknila/sosym/Requirements_Document_for_CMS.pdf

Exemple 1:

B_1 : The system shall allow the department to manage courses.

B_2 : The system shall allow the department to create courses.

– **Domain**(B_1)= *(department, Course, Add-course, Delete-course, Update-course, Select-course).*
– **Domain**(B_2)= *(department, Course, Add-course).*

Nous observons que (**domain**(B_2) $\subset$ **domain**(B_1)). Nous concluons que B_2 est l'une des besoins décomposés de B_1. Donc, (B_1 contains B_2).

2. **Relation "Requires"** :

Définition 2

> *Un besoin B_1 requiert un besoin B_2 si B_1 n'est satisfait que lorsque B_2 est satisfait. La relation requires peut être considérée comme pré-condition pour le besoin exigeant. La relation requires est non-réflexive, non symétrique et transitive.*

Soit: $B_1 : < \mathcal{A}_1, \mathcal{R}_1, \mathcal{M}_1 >$, $B_2 :< \mathcal{A}_2, \mathcal{R}_2, \mathcal{M}_2 >$ deux besoins des utilisateurs où :

Domain(B_1)= domain ($\mathcal{A}_1$)$\cup$ domain ($\mathcal{R}_1$) $\cup$ domain ($\mathcal{M}_1$)

Domain(B_2)= domain ($\mathcal{A}_2$)$\cup$ domain ($\mathcal{R}_2$)$\cup$ domain ($\mathcal{M}_2$)

Formalisation 20

> $Requires_{rule}$: B_1 **requires** B_2: $\forall x \in domain(B_1)$: $x \in domain(B_2) \wedge \exists x \in domain(B_2) : x \notin domain(B_1)$

Nous expliquons la relation requires avec l'exemple suivant.

Exemple 2 :

B_3 : The system shall provide database course information.

B_4 : The system shall be able to store database course information.

– **Domain**(B_3)= *(Course, Provide-course, Stor-course).*
– **Domain**(B_4)= *(Course, Stor-course).*

Afin de fournir des informations sur les cours de base de données, le système a besoin de stocker des informations sur ces cours. Par conséquent, nous concluons que B_3 requires B_4 pour être satisfait.

3. **relation "Conflicts with"** :

Définition 3

> *Un besoin B_1 est en conflit avec un besoin B_2 si la réalisation de B_1 exclut la réalisation de B_2 et vice versa. La relation de conflit adresse une contradiction entre les besoins. Notre approche de raisonnement peut exécuter des relations de conflits n-aires entre des besoins multiples. La relation de conflit binaire est symétrique, non-réflexive et elle n'est pas transitive.*

Formalisation 21

> $Conflicts_{rule}$: B_1 **Conflicts with** B_2: $(\neg\exists x : (x \in (domain(B_1)) \wedge x \in (domain(B_2)) \Leftrightarrow domain(B_1) \cap domain(B_2) = \emptyset) \Leftrightarrow \forall x \in domain(B_1) \Rightarrow \exists y \in domain(B_2) \wedge Contradict(x, y)$.

Nous expliquons la relation "conflicts with" avec l'exemple suivant.

Exemple 4 :

B_5 : The system shall allow department to delete all studies and personal information of students with more than 40 years.

B_6 : The system shall allow department to delete all studies and personal information of students with less than 40 years.

- **Domain**(B_5)= *(department, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course), Name, Age, Marks, Birth date, More, Delet-name, Delet-Birthdate, Delet-course, Delet-marks, Age>=40, DisjointWith (More, Less),...).*
- **Domain**(B_6)= *(department, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course), Name, Age, Marks, Birth date, Less, Delet-name, Delet-Birthdate, Delet-course, Delet-marks, Age<40, DisjointWith (More, Less), DisjointWith (Age<40, Age>=40)...).*

Nous observons que *DisjointWith (More, Less)* et *DisjointWith (Age<40, Age>=40)* sont des concepts conflictuels (une relation de contradiction entre les concepts). Donc, nous concluons que: B_5 *conflicts with* B_6.

4. **Relation "Refines"** :

Définition 4

> *Un besoin B_1 affine un besoin B_2 si B_1 est dérivé de B_2 en ajoutant plus de détails à ses propriétés. Pareillement à la relation requires, le relation refines est non-réflexive, non symétrique et transitive.*

Formalisation 22

> $Refines_{rule}$: B_1 **Refines** B_2: domain(B_2) $\subset$ domain(B_1)

Nous expliquons la relation refines avec l'exemple suivant.

Example 3 :

B_7 : The system shall allow department to create courses.

B_8 : The system shall allow department to create entirely new courses.

- **Domain**(B_7)= *(department, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course), Add-course,etc.).*
- **Domain**(B_8)= *(department, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course), Add-New-course, etc.).*

Nous observons que (**domain**(B_7) $\subset$ **domain**(B_8)). Le besoin B_7 nécessité seulement d'une propriété pour la description du cours *Course*. Cependant, B_8 explique en détails les cours par l'ajout d'une propriété *New-course*. Nous concluons que B_8 affine B_7. Par conséquent, on peut noter également que B_8 requires B_7.

5. **Relation "Equals"** :

Définition 5

> *Un besoin B_1 est égale à un besoin B_2, si les propriétés de B_1 sont exactement les mêmes que celles de B_2 et vice versa. La relation égale est symétrique, reflexive et transitive.*

Formalisation 23

> $Equals_{rule}$: B_1 **Equals** B_2: domain(B_1) $\equiv$ domain(B_2)

Nous expliquons la relation "Equals" avec l'exemple suivant.

Example 5 :

B_9: The system shall allow lecturers to create courses.

R_{10}: The system shall allow lecturers to create courses.

- **Domain**(B_9)= *(Lecturers, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course),Add-course,etc.).*
- **Domain**(R_{10})= *(Lecturers, Student, Course, Person* $\cap \forall$ *takesCourse(Person, Course), Add-course, etc.).*

Nous observons que (**domain**(B_9) $\equiv$ **domain**(R_{10})). Donc, B_9 est égale R_{10}.

4.4 Évaluation du processus de raisonnement

Nous effectuons des expériences pour déterminer le nombre de relations inférées dans les deux scénarii de raisonnement. Les résultats obtenus sont illustrés dans le tableau 5.1. Toutes les expérimentations ont été effectuées sur une machine Intel (R) Core (TM) i5 ayant un processeur d'une fréquence de 2.67 GHz, équipée d'une mémoire centrale de 4 Go, tournant sous le système d'exploitation Windows 7 professionnel.

Les résultats montrent que le nombre de relations vérifiées dans le modèle générique *(OntoReq)* est plus que le nombre total de relations vérifiées dans les modèles de besoins locaux *(OntoGoal, OntoUcase, OntoMCT)*. Enfin, pour mesurer l'efficacité des scénarii de raisonnement, nous avons calculé le temps

Scénarii	**Modèles**	**Nombre de relations**						**Domain (C&R)**
		Requires	*Conflicts*	*Contains*	*Refines*	*Equals*	*Totale*	
Scénario (1)	*OntoGoal*	2	1	2	2	0	7	38
	OntoUcase	2	0	2	1	0	6	25
	OntoMCT	2	4	2	0	0	9	34
	OntoReq	4	2	2	2	2	12	40
Scénario (2)	*OntoReq*	8	8	8	4	2	30	66

TABLE 5.1 – Nombre de relations vérifiées pendant le raisonnement.

FIGURE 5.7 – Temps d'exécution de raisonnement.

d'exécution (en secondes) nécessaire pour exécuter le raisonnement dans chaque scénario. (Figure 5.7).

Les résultats obtenus montrent que le scénario 1 est plus performant que scénario 2. Cela montre l'intérêt de répartir les efforts de raisonnement au niveau local, puis de les raffiner au niveau global. Le

raisonneurs ne sont efficaces que sur les ontologies de petites tailles (les ontologies locales des besoins). Plus les ontologies sont grandes plus les performances diminuent.

5 Conclusion

Dans ce chapitre, nous avons présenté une deuxième contribution. Elle consiste à exploiter la puissance de l'ingénierie dirigée par les modèles pour offrir un deuxième scénario d'intégration des besoins. Nous avons défini un méta-modèle générique représentant les besoins. Les trois modèles utilisés deviennent alors des instances de ce méta modèle. Nous nous somme basés sur les mêmes hypothèses que la première approche d'intégration concernant l'existence d'une ontologie partagée qui couvre la sémantique du domaine et la définition des ontologies locales qui référencent l'ontologie partagée pour fournir une autonomie aux concepteurs. Dans ce cas, chaque concepteur donne seulement son langage préféré pour concevoir une application, il extrait par la suite son langage de besoins couplé à son ontologie. Nous avons présenté également un mécanisme de raisonnement sur l'ensemble des besoins émis par les concepteurs. Deux scénarios d'implémentation de raisonnement sont proposées (i) *ship whole* et (ii) *reason as needed*. L'évaluation des deux solutions a montré l'intérêt du raisonnement, ainsi que la supériorité de la solution *reason as needed* sur la solution *ship whole* en termes de temps de traitement.

Dans le chapitre suivant, nous allons étudier la contribution de la prise en compte des besoins dans la conception physique des entrepôts de données.

Chapitre 6

Exploitation des besoins pour la conception physique

Sommaire

1 Introduction . . . 107
2 La conception physique . . . 108
3 Conception physique dirigée par les besoins . . . 109
3.1 La sélection des index . . . 110
3.2 La fragmentation horizontale . . . 111
4 Persistance des besoins . . . 111
4.1 Le langage d'exploitation OntoQL . . . 112
4.2 Persistance des besoins . . . 112
4.3 Génération des requêtes SQL . . . 114
5 Expérimentation et évaluation de notre proposition . . . 116
5.1 Structures d'optimisation comme des services . . . 116
5.2 La sélection des index . . . 116
5.3 La fragmentation horizontale . . . 117
6 Conclusion . . . 119

Résumé. Dans les deux chapitres précédents, nous avons proposé deux approches d'intégration sémantique des besoins hétérogènes dans le contexte des entreprises étendues. Dans ce chapitre, nous allons étudier comment ces approches peuvent contribuer à la conception physique des entrepôts de données. Cette conception consiste à sélectionner des structures d'optimisation comme les index et la fragmentation [18, 79, 78]. Actuellement, la conception physique est réalisée à l'aide des requêtes extraites des fichiers de journalisation (logs) et supposent que la base/entrepôt de données est en cours d'utilisation. Ce processus augmente la charge de travail des administrateurs de ce type de bases de données. Comme processus alternatif, nous proposons une approche de conception physique qui se déroule pendant la modélisation conceptuelle, une fois les besoins identifiés.

1 Introduction

La collecte des besoins est une pré-condition pour réussir un projet de conception de bases de données, du fait qu'elle contribue à chaque phase du cycle de vie de conception des bases de données avancées. Ce cycle est composé de six étapes principales : (a) la définition des besoins des utilisateurs, (b) la modélisation conceptuelle, (c) la modélisation logique, (d) la modélisation physique, (e) l'implémentation de la base de données et (f) son tuning. Dans ce chapitre, nous allons étudier la contribution de notre approche de gestion des besoins sur la conception physique. Nous avons choisi cette phase car elle est l'objet de nombreux travaux menés au sein de l'équipe Ingénierie des Données et des Modèles du laboratoire LIAS de l'ISAE-ENSMA. Ces travaux concernent la sélection des structures d'optimisation redondantes (du fait qu'elles dupliquent les données) comme les vues matérialisées, les index, la réplication, etc. et des structures d'optimisation non redondantes comme le partitionnement horizontale, l'ordonnancement des requêtes, etc. La sélection de ces structures se fait selon trois modes : *statique*, *incrémental* et *dynamique*.

Dans la sélection statique, les entrées du problème sont un schéma d'un entrepôt de données ou d'une base de données, un ensemble de k requêtes les plus fréquemment utilisées (les requêtes sont donc connues a priori) et une contrainte particulière liée au problème traité. Le problème de sélection de structure d'optimisation consiste alors à trouver un ensemble d'instances d'optimisation réduisant le coût de la charge des requêtes et satisfaisant la contrainte. Dans la sélection incrémentale, on suppose à nouveau que certaines requêtes sont connues mais cette fois-ci, on traite le fait que d'autres requêtes puissent être posées sur le système. Dans la sélection incrémentale, on lève l'hypothèse sur la connaissance a priori des requêtes à optimiser. Les travaux existants ont montré la complexité de ces trois types de sélection.

D'après l'analyse des travaux existants, nous avons identifié que les requêtes sont au coeur de la phase de la conception physique. Ces dernières peuvent être facilement extraites à partir des besoins identifiés dans la phase conceptuelle, que nous allons détailler dans ce chapitre. Cette démarche nous a amené à une nouvelle vision de la conception de bases de données, dans laquelle deux types d'acteurs sont présents tout au long du cycle de vie : le concepteur et l'administrateur. Jusqu'à présent, le concepteur a pour tâches de (i) proposer une bonne méthodologie de conception tout en identifiant les entités et les propriétés de l'univers de discours et (ii) réaliser un cahier des charges informatique pour évaluer la faisabilité et/ou le coût de passage de cette modélisation conceptuelle à une base de données. Tandis que l'administrateur est généralement responsable de plusieurs tâches qu'il doit assurer simultanément. Parmi ces tâches, nous pouvons citer la gestion du recouvrement des données, le tuning, le suivi des structures de données, la gestion de l'intégrité et des droits d'accès, etc. Fabio et al. [96] ont fait une étude sur le temps passé par l'administrateur pour assurer les tâches d'administration. La figure 6.1 présente le taux d'effort (en pourcentage de temps) consacré par l'administrateur pour les différentes tâches d'administration. Cette figure montre que la tâche de conception physique et de tuning consomme 17% du temps d'administration. Cet effort est dû au nombre important de choix qu'il doit effectuer. Vu que la conception physique se fait à l'aide des requêtes qui peuvent être identifiées à l'aide de l'expression des besoins, dans notre vision, cette tâche pourrait être déléguée aux concepteurs.

Ce chapitre est organisé comme suit. Nous présentons d'abord la conception physique d'une manière générale dans la section 2. Puis, nous montrons comment les besoins peuvent être utilisés pour répondre à cette conception dans la section 3. Afin de pouvoir utiliser les besoins dans tout le cycle de vie d'un

FIGURE 6.1 – Répartition de l'effort de l'administrateur.

système de gestion de données, nous montrons dans la section 4 comment ils peuvent être persistés. Enfin, une étude de cas est présentée dans la section 5 montre que la conséquence de l'utilisation des besoins dans la conception physique est la réduction des tâches de l'administrateur.

2 La conception physique

Dans les applications décisionnelles, la conception physique est devenue un enjeu important vu les exigences des décideurs en termes de temps de réponse de requêtes complexes, impliquant un nombre important de jointures entre des tables de faits extrêmement larges et des tables de dimension et des opérations d'agrégation [34]. Durant cette phase, l'administrateur doit sélectionner un ensemble de structures d'optimisation pour satisfaire ses requêtes (figure 6.2). Une large panoplie de structures d'optimisation a été proposée dans le contexte des entrepôts de données dont une majorité supportée par les systèmes de gestion de bases de données commerciaux et non commerciaux. Nous pouvons ainsi citer les vues matérialisées, les index avancés, la réplication, la fragmentation, le traitement parallèle, la compression, etc. Certaines structures sont issues des bases de données traditionnelles comme la fragmentation, la réplication, certains types d'index (par exemple, le B-tree), le regroupement, etc. Ces structures sont divisées en deux catégories que nous appelons les *structures redondantes* et les *structures non redondantes* [14]. Les structures redondantes optimisent les requêtes mais entraînent des coûts de stockage et de maintenance. Les vues matérialisées, les index, la fragmentation verticale sont trois principaux exemples de cette catégorie. Les structures non redondantes ne nécessitent ni coût de stockage, ni coût de maintenance. Deux exemples de cette catégorie sont la fragmentation horizontale et les traitements parallèles.

FIGURE 6.2 – La conception physique.

Avant de définir le problème de sélection de structure d'optimisation, nous donnons quelques notations. Une famille de structures d'optimisation, comme les index, les vues matérialisées, le partitionnement, est dénotée par $\bar{SO} = \{\mathcal{VM}, \mathcal{HF}, \mathcal{I}\}$. Une structure d'optimisation est une instance d'une famille donnée. Si par exemple, une classe $SO \in \bar{SO}$ est $\mathcal{VM}$, alors la structure d'optimisation représentera un ensemble de vues à matérialiser. A chaque classe de $\bar{SO}$ est associé un ensemble de contraintes dénoté par $contr(SO)$. Les contraintes liées aux index et aux vues matérialisées représentent le coût de stockage.

Le problème de sélection de ces structures est défini comme suit.

Entrées :

- Charge de requêtes $Q = \{Q_1, Q_2, \cdot, Q_k\}$;
- Ensemble des structures d'optimisation $\bar{SO}$;
- Ensemble des contraintes $contr(SO)$.

Sorties : un ensemble de structures optimisant le temps d'exécution de Q et satisfaisant les contraintes $contr(SO)$.

Ce problème est connu comme étant NP-difficile pour chaque instance. Une large panoplie d'algorithmes a été proposée pour résoudre cette complexité, en prenant en compte un ensemble de requêtes. Ces dernières sont souvent issues de l'ensemble des besoins identifiés dans la phase de recueil de besoins. Donc, on peut se demander pourquoi attendre la conception physique pour sélectionner des structures d'optimisation, si nous pouvons la faire lors de la phase de conception.

3 Conception physique dirigée par les besoins

Les différentes techniques d'optimisation s'effectuent habituellement au niveau physique comme l'indexation ou la fragmentation horizontale qui reposent sur une charge de requêtes fréquentes et un ensemble de contraintes données (espace de stockage, coût de mise à jour, etc). Cette charge de requêtes n'est obtenue qu'après une période d'exploitation de la base de données. Or, il est probable que cette charge de requêtes soit incluse dans l'ensemble des besoins des utilisateurs qui eux sont disponibles dès le début de la conception puisqu'il sont fournis par la phase de collection et d'analyse des besoins. Par exemple, l'ensemble des contraintes (l'espace de stockage, le coût de mise à jour,etc.) est collecté auprès

des parties prenantes du système sous forme de besoins non fonctionnels.

Dans ce travail, nous proposons d'exploiter les besoins des utilisateurs pour effectuer une optimisation dès le niveau logique. Ainsi, nous envisageons de faire remonter ces tâches d'optimisation du niveau physique au niveau logique, et donc d'alléger un peu la lourde responsabilité de l'administrateur en délégant cette tâche au concepteur. Nous tentons ici de fournir un lien "humain" entre la conception logique et la conception physique et de réduire ainsi l'écart entre ces deux niveaux de conception.

Le problème de sélection de ces structures est défini comme suit.
Entrées :
- Ensemble des besoins $B = \{B_1, B_2, \cdot, B_k\}$;
- Ensemble des structures d'optimisation $\bar{SO}$;
- Ensemble des contraintes $contr(SO)$.

Sorties : un ensemble de structures d'optimisation optimisant le temps d'exécution de Q et satisfaisant les contraintes $contr(SO)$.

Nous considérons dans ce chapitre deux cas d'étude : le problème de sélection des index $\mathcal{I}$ et celui de la fragmentation horizontale $\mathcal{HF}$. Nous reformulons ces problèmes en prenant comme entrée les besoins des utilisateurs.

3.1 La sélection des index

Afin d'offrir des solutions d'indexation adaptées au contexte des $\mathcal{ED}$, de nouveaux index ont été proposés. On peut ainsi citer (a) les *index binaires* [33] qui optimisent les opérations de sélection définies sur des attributs appartenant à des tables de dimension. Ces index sont largement utilisés dans les bases de données XML [14] et dans la recherche d'information [14]. (b) Les *index de jointures en étoile* permettant de stocker le résultat d'exécution d'une jointure en étoile entre plusieurs tables. (c) Les *index de jointure binaire* ($\mathcal{IJB}$) [97] quant à eux permettent d'optimiser à la fois les jointures en étoile et les opérations de sélections définies sur les tables de dimensions. Les index de jointure binaire sont supportés par la plupart des SGBD commerciaux. Nous distinguons deux types d'$\mathcal{IJB}$: *les $\mathcal{IJB}$ simples* (mono-attribut) définis sur un seul attribut d'une table de dimension et *les $\mathcal{IJB}$ multiples* (multi-attributs) définis sur plusieurs attributs issus d'une ou plusieurs tables de dimensions. Pour faciliter l'exploitation et la gestion des $\mathcal{IJB}$ multiples volumineux, certains travaux ont proposé l'utilisation de la fragmentation verticale et horizontale [31].

La sélection d'index de jointure en étoile est un problème difficile [24]. Il est défini comme suit.
- $I = \{I_1, ..., I_n\}$ un ensemble d'index candidats (obtenus à partir des prédicats de sélection des **requêtes**).
- $Q = \{Q_1, ..., Q_k\}$ un ensemble de **requêtes**.
- S la taille de l'espace de stockage allouée pour les index.

L'objectif du problème de sélection des index de jointure binaire est de trouver une configuration d'index réduisant le coût d'exécution des requêtes et satisfaisant la contrainte de stockage [33].

D'après cette formalisation, nous constatons la présence des requêtes et avec la présence des besoins, nous proposons une autre formalisation dirigée par les besoins :

33. Le stockage de l'ensemble des index ne doit pas dépasser la contrainte S

– $I = \{I_1, ..., I_n\}$ un ensemble d'index candidats (obtenus à partir des **besoins**).
– $B = \{B_1, ..., B_m\}$ un ensemble de **besoins**.
– S la taille de l'espace de stockage allouée pour les index.

Cette formalisation a le même objectif que la précédente.

3.2 La fragmentation horizontale

La fragmentation horizontale ($\mathcal{FH}$) est une technique d'optimisation considérant comme l'une des structures d'optimisation dans le cadre des entrepôts de données relationnels. Elle permet de décomposer une table en plusieurs sous ensembles disjoints appelés fragments horizontaux, chacun contient un sous ensemble de tuples. On distingue deux types de $\mathcal{FH}$: (1) $\mathcal{FH}$ primaire définie sur une table de dimension en fonction de ses propres attributs, et (2) $\mathcal{FH}$ dérivée définie sur la table des faits en fonction des dimensions fragmentées. La fragmentation dérivée est adaptée au contexte des entrepôts de données relationnels. Le problème de sélection d'un schéma de $\mathcal{FH}$ est un problème difficile [16]. Traditionnellement, il est formalisé de la manière suivante.

Étant donnés :
– un schéma d'un entrepôt de données composé d'un ensemble de tables de dimensions $D = \{D_1, D_2, ..D_d\}$ et une table de faits F ;
– un ensemble de requêtes $Q = \{Q_1, ..., Q_k\}$;
– un seuil maximal de fragments finaux W de la table des faits. Ce seuil est fixé par l'administrateur.

Le problème de la $\mathcal{FH}$ consiste alors à sélectionner un schéma de fragmentation optimisant le coût d'exécution des requêtes et satisfaisant le seuil W. D'une manière similaire à ce que nous avons proposé pour les index, une autre formalisation dirigée par les besoins peut être élaborée.

Le passage d'une formalisation dirigée par des requêtes à une formalisation dirigée par des besoins nécessite une analyse approfondie des besoins pour extraire les requêtes. Pour ce faire, nous proposons d'abord de persister les besoins dans une base de données pour pouvoir ensuite les exploiter afin d'identifier les requêtes.

4 Persistance des besoins

Rappelons que les besoins représente le coeur de toute conception. Pour cela, il est important de les stocker au sein de la base de données de manière à ce que les concepteurs puissent les trouver facilement via des procédures d'intégration. Il est à noter aussi que dans n'importe quel projet lié aux bases/entrepôts de données, aucune trace du modèle des besoins n'est sauvegardée. Etant donné que nos besoins sont sémantiques (liés à une ontologie), nous proposons une solution de stockage au sein d'une base de données à base ontologique conçue au laboratoire LIAS, appelée OntoDB [45] qui est implantée sur le SGBD (Système de Gestion de Base de Données) PostgreSQL [34]. Comme illustré dans la Figure 6.3, l'architecture d'OntoDB se compose des parties suivantes :
– *Partie méta-base* : souvent appelée *system catalog*, elle représente la partie traditionnelle des bases de données classiques qui contient les tables décrivant les tables (méta-base). Dans OntoDB, elle

34. http://www.postgresqlfr.org/

contient la description des structures définies dans les trois autres parties ;
- *Partie données* : contient les instances des classes de l'ontologie, et les propriétés associées à ces dernières. Elle est stockée sous forme horizontale (Une table pour chaque classe);
- *Partie Ontologie* : elle contient les concepts (classes, propriétés et relations) de l'ontologie, formalisés conformément au modèle d'ontologie PLIB;
- *Partie Méta-schéma* : elle décrit le modèle d'ontologie utilisé, et donc le méta-schéma des instances ontologiques.

FIGURE 6.3 – Architecture d'OntoDB.

L'architecture d'OntoDB permet le support des évolutions du modèle d'ontologies PLIB et l'extension de ce modèle lorsque le besoin se présente. L'interrogation des données au niveau ontologique est assurée par le langage de requête OntoQL [70].

4.1 Le langage d'exploitation OntoQL

OntoDB est dotée du langage de requêtes OntoQL [70] qui permet d'interroger les données ainsi que l'ontologie et son modèle persisté sur OntoDB. Il étend le langage SQL pour pouvoir exploiter la partie ontologique. Il est basé sur un noyau commun aux différents modèles d'ontologie et du fait qu'il n'est pas statique, il peut être donc facilement étendu par des instructions dédiées. Ainsi, OntoQL est indépendant de la représentation de l'ontologie (formalisme) et de la représentation des données (logique).

4.2 Persistance des besoins

Nous avons implémenté notre mécanisme de persistance des besoins dans la base de données à base ontologique OntoDB possédant une architecture de type III (figure 6.4). La structure d'OntoDB offre une

FIGURE 6.4 – BDBO de type III étendu par les besoins des utilisateurs.

grande flexibilité et permet son extension par le méta-modèle des besoins. Cette extension se fait selon un schéma horizontal où chaque classe du méta-modèle des besoins est représentée par une table Entity dans le métaschéma d'*OntoDB*. Cette extension se fait en utilisant le langage ontologique OntoQL, plus précisément via le Langage de Définition des Ontologies (LDO) [69] à travers la syntaxe suivante :

```
CREATE Entity #[nom de l'entité]([nom d'attribut, Type d'attribut (Property)])
```

Le script suivant contient les requêtes OntoQL permettant la création du méta-modèle des besoins dans le méta-schéma :

```
CREATE Entity #GenericMetamodel(#Domain String,
#Name String, #Date Date, #collects REF(#Requirements))

CREATE Entity #Result(#its_properties REF (#Property) Array)

CREATE Entity #Action(#its_properties REF (#Property) Array)

CREATE Entity #Criterion(#its_properties REF (#Property) Array)

CREATE Entity #Requirement (#IdReq Int, #NameReq String,
#DescriptionReq String, #ContextReq String, #PriorityReq Int,
#defined_for REF(#Result), #ComposedOf REF(#Action),
#MesuredBy REF(#Criterion), #IsResponsible REF(#Actor))

CREATE Entity #Operation UNDER #Criterion
CREATE Entity #Function UNDER #Criterion

CREATE Entity #Goal UNDER #Requirement
CREATE Entity #UseCase UNDER #Requirement
CREATE Entity #Treatment UNDER #Requirement

CREATE Entity #Functional_Requirement UNDER #Requirement
CREATE Entity #NonFunctional_Requirement UNDER #Requirement
```

4.3 Génération des requêtes SQL

Notre proposition consiste à identifier une charge de requêtes en s'appuyant sur les besoins exprimés avec notre modèle générique (voir section 2 du chapitre 5). Pour ce faire, nous devons parcourir les besoins et les transformer en requêtes SQL. Après analyse, il s'avère qu'il existe deux types de besoins :

- Besoin « mesurable » (type requête) : C'est un besoin qui peut être mesurable, car il possède un résultat (souvent quantifiable), et éventuellement un critère, par exemple, le besoin qui « mesure le nombre des étudiants de sexe féminin », représente un besoin dont le résultat est le nombre des étudiants et le critère est le sexe féminin. Il est souvent lié aux besoins fonctionnels.
- Besoin « non mesurable » (type programme) : C'est un besoin qui ne possède pas de résultat (pouvant être exprimé en fonction des concepts ontologiques) tel que nous l'avons défini dans notre modèle, Par exemple, le besoin qui s'inscrit dans la maintenance web du domaine universitaire « l'enseignant a besoin d'une interface contenant tel ou tel champs pour évaluer l'étudiant », ne possède pas de résultat, il décrit plutôt un programme informatique.

Au niveau d'instances physiques, les besoins mesurables sont ceux qui possèdent une liaison avec la classe « Result » et éventuellement avec la classe « Criterion », contrairement aux besoins non mesurables qui se contentent d'avoir une description.

En effet, seuls les besoins type requêtes peuvent être transformés en requêtes (d'où leurs noms d'ailleurs), en opérant une transformation de type *Model To Text* définie dans l'ingénierie dirigée par les modèles (IDM). Cette étape est réalisée à l'aide du langage de transformation *Acceleo* [35] disponible sous l'environnement Eclipse, pour la définition des règles de transformation appliquées sur chaque besoin ontologique comme décrit ci-après :

```
SELECT [Result.Measure] (Property)[Result.DefinedBy]
FROM (Class) [Result.DefinedBy]
WHERE Criterion.DefinedBy Criterion.Opr Criterion.Literal
```

Prenons l'exemple du besoin suivant extrait des besoins décisionnels de la spécification de SSB : *Ce besoin mesure l'augmentation des revenus des commandes engendrée par l'éliminiation de certaines réductions de commandes selon des quantités données dans une année donnée. Sachant que l'augmentation des revenus est égale au produit [Prix-étendu*remise* [36]*].*

Tout d'abord, ce besoin est modélisé suivant notre modèle générique des besoins comme suit : (figure 6.5)

Notons qu'ici le *Result* est défini non pas par un seul concept comme c'est le cas pour les quatre critères, mais plutôt par une expression de concepts. Cette expression est composée de l'opérande gauche qui est la propriété *lo_extendedPrice*, l'opérande droit *lo_discount* et l'opérateur arithmétique *.

Une fois ce besoin est persisté, nous pourrions le transformer en requêtes (si besoin est), en appliquant la règle de transformation d'*Acceleo* comme suit :

- *Action* définit le type de la requête, si sa valeur est égale à *Display*, il s'agit d'une requête SELECT, sinon (Delete, Modify, Create) la clause SELECT est remplacée par la clause appropriée (DELETE, UPDATE, CREATE) avec les modifications nécessaires.

35. http://www.acceleo.org/pages/accueil/fr
36. Le prix étendu est représenté ontologiquement par la propriété *lo_extendedPrice* de la commande, représentée à son tour par la classe *Lineorder*. Quant à la remise, elle est représentée par la propriété *lo_discount* de *Lineorder*

FIGURE 6.5 – Un besoin SSB modélisé suivant notre modèle générique des besoins.

- La clause SELECT de la requête porte sur la valeur de l'attribut « measure » (SUM, AVG, etc. s'il existe) de la classe « Result », et les **propriétés** constituant l'expression liée à cette dernière.
- La clause FROM, quant à elle, porte sur les **classes** constituant l'expression liée à la classe « Result ».
- Enfin, pour la clause WHERE, elle est définie par la classe « Criterion », qui se compose dans l'ordre, d'une opérande gauche (Concept), d'un opérateur de comparaison, et d'une opérande droite (Litérale).

Ainsi, notre besoin est transformé en la requête suivante :

```
SELECT {somme (lo_extendedprice*lo_discount)}
FROM {lineorder, date}
WHERE {d_year = 1993, lo_discount <3, lo_discount >1, lo_quantity <25}
```

5 Expérimentation et évaluation de notre proposition

Afin d'illustrer notre démarche, nous présentons une étude de cas basée sur le domaine des transactions commerciales.

5.1 Structures d'optimisation comme des services

Pour évaluer nos propositions concernant la conception physique, nous avons développé une approche orientée service pour lier nos travaux à ceux des doctorants travaillant sur la conception physique au laboratoire. Plus précisément nous avons demandé à Amira Kerkad et Rima Bouchakri qui s'intéressent respectivement à la fragmentation horizontale et les index de jointure binaire de nous fournir un service qui reçoit en paramètre les entrées de chaque problème : le schéma de l'entrepôt de données, l'ensemble des besoins et les contraintes et qui retourne les résultats. Nous avons ensuite alimenté ce service avec les requêtes identifiées à partir des besoins. Par cette démarche, notre but est d'évaluer si l'approche de sélection des structures d'optimisation à partir des besoins permet d'obtenir des résultats proches de ceux obtenus par l'approche basée sur les requêtes.

Pour le schéma nous avons utilisé les données du banc d'essai *Star Schema Benchmark* [37] (SSB), qui est basé sur le schéma de données du benchmark TPC-H. La figure 6.6 présente une partie de l'ontologie SSB sous la forme d'un diagramme de classes. Il comporte différentes classes relatives au domaine des transactions commerciales. La classe centrale est : *Order*, liée aux classes *Customer*, *Supplier*, *Part-product* et *Time*. D'autres classes décrivent la localisation du client comme *City*, *Nation* et *Region*. Les classes *Category* et *Brand* décrivent le type et la marque du produit. Dans ce chapitre, nous considérons le schéma de données SSB comme l'ontologie globale couvrant la sémantique du domaine. Nous considérons également les besoins décisionnels (*business questions*) définis au niveau de la spécification du benchmark SSB comme les besoins exprimés par les concepteurs du système. Dans les sections suivantes, nous présentons les résultats des deux sélections dirigées par les besoins concernant les index de jointure binaire et la fragmentation horizontale.

5.2 La sélection des index

Les expérimentations sont effectuées sur le schéma logique du benchmark SSB. L'ensemble des besoins décisionnels (13 besoins) spécifiés dans la spécification du banc d'essai SSB sont exploités (voir annexe 2.5). Les expérimentations ont donné les résultats résumés dans le tableau 6.1. Ce tableau présente les index générés et le taux d'optimisation du coût d'exécution de chaque besoin. Ces résultats sont confrontés aux index et aux coût d'optimisation obtenus à partir de la charge des requêtes finales (fournies au niveau de la spécification du banc d'essai SSB). Ces résultats confirment que les index proposés par les besoins couvrent les index proposés par les requêtes du banc d'essai. Les coûts d'optimisation sont également similaires. Ceci s'explique par le fait que les requêtes issues des besoins sont similaires aux requêtes du banc d'essai (sans les clauses *group by* et *order by* que nous n'avons pas pu générer).

37. http://www.cs.umb.edu/ poneil/StarSchemaB.PDF

FIGURE 6.6 – Ontologie du benschmark SSB.

5.3 La fragmentation horizontale

Le service de la fragmentation horizontale utilise l'algorithme basé sur un recuit simulé [15]. Nous avons comparé les résultats de la fragmentation effectuée en utilisant 12 requêtes de la vraie charge de requêtes du banc d'essai SSB et les requêtes que nous avons obtenues à partir des besoins. La figure 6.7 illustre les résultats des expérimentations effectuées. Nous remarquons que les besoins apportent un gain en performance supérieur de 15% comparé aux requêtes. Cette amélioration s'explique par l'absence des *Group By*. Ces résultats sont encourageants dans le sens où nous montrons que les besoins permettent de fournir une optimisation par la $\mathcal{FH}$ pratiquement équivalente à une fragmentation effectuée lors de la phase physique.

La tableau 6.2 illustre les gain en performance par requête.

Ces deux études ont montré la faisabilité de la sélection des structures d'optimisation en se basant sur les besoins. Ce qui nous incite à penser un autre modèle économique de conception de bases de données (figure 6.8). Certaines tâches que les concepteurs peuvent faire leur seront alors attribués ce qui soulage le travail des administrateurs. Rappelons que plusieurs outils d'administration ont été proposés par les éditeurs commerciaux comme : Oracle SQL Tuning Advisor [42], DB2 Design Advisor [132] et Microsoft Database Tuning Advisor [9]. Par exemple Oracle SQL Tuning Advisor est un outil

Stockage (GO)	Besoins		Requêtes	
	Index	Coût (%)	Index	Coût (%)
1	d_year	46.7	d_year	46.7
1.5	d_year, s_region	72.7	d_year, s_region	72.7
2	d_year, s_region, p_category	75.1	d_year, s_region, p_category	75.1
2.5	d_year, d_yearmonth, s_region	87.9	d_year, d_yearmonth, s_region	87.9
3	d_year, d_yearmonth, s_région	87.9	d_year, d_yearmonth, s_région	87.9
3.5	d_year, d_yearmonth, s_region, p_category	90.2	d_year, d_yearmonth, s_region, p_category	90.2
4	d_year, d_yearmonth, c_region, s_region, p_category	91.7	d_year, d_yearmonth, c_region, s_region, p_category	91.7
4.5	d_year, d_yearmonth, c_region, s_region, p_mfgr, p_category	92	d_year, d_yearmonth, c_region, s_region, p_mfgr, p_category	91.9
5	d_year, d_yearmonth, c_region, s_region, s_nation, p_category	92.7	d_year, d_yearmonth, c_region, s_region, s_nation, p_category	92.6

TABLE 6.1 – Index générés et taux d'optimisation du coût d'exécution des besoins par rapport aux requêtes

Besoins		Requêtes	
N°	Gain (%)	N°	Gain (%)
Q1	75	Q1	75
Q2	50	Q2	87,5
Q3	87,5	Q3	75
Q4	88	Q4	50
Q5	66	Q5	50
Q6	66	Q6	50
Q7	66	Q7	75
Q8	0	Q8	50
Q9	50	Q9	50
Q10	50	Q10	50
Q11	68	Q11	75
Q12	50	Q12	50

TABLE 6.2 – Index générés et taux d'optimisation du coût d'exécution des besoins par rapport aux requêtes

FIGURE 6.7 – Fragmentation horizontale à partir des besoins.

permettant de générer des conseils pour optimiser une charge de requêtes afin d'améliorer leurs performances. Les conseils se présentent sous forme de recommandations, chacune avec le bénéfice qu'elle apporte. L'administrateur a le choix soit d'accepter les conseils soit de les compléter. Les recommandations concernent trois structures d'optimisation : les vues matérialisées, les index définis sur une seule table et la fragmentation horizontale primaire. Ces recommandations sont établies en utilisant les modèles de coût basés sur des statistiques sur la base de données. Cet outil utilise souvent une sélection isolée. DB2 Design Advisor est une amélioration de l'outil DB2 Index Advisor tool défini initialement pour automatiser la sélection des index. DB2 Design advisor permet de générer des recommandations sur quatre structures d'optimisation : les vues matérialisées, les index, la fragmentation horizontale primaire et le groupement. La fragmentation horizontale est limitée au mode de Hachage dans une architecture parallèle où plusieurs processeurs sont interconnectés à travers un réseau. L'outil Microsoft Database Tuning Advisor, par exemple fait partie intégrante de SQL Server 2005. Les recommandations générées par cet outil concernent quatre techniques d'optimisation : la fragmentation horizontale primaire, la fragmentation verticale, les index et les vues matérialisées. Contrairement à l'outil d'Oracle, Database Tuning Advisor utilise l'optimiseur de requêtes pour évaluer la qualité des techniques sélectionnées et les différentes alternatives. Pour réduire le coût des appels à l'optimiseur, l'outil utilise des serveurs de tests pour estimer la qualité des différentes structures. D'autres outils académiques ont été également proposés, on peut citer l'exemple de l'outil PARINDA [89], développé à l'Ecole Polytechnique Fédérale de Lausanne.

Pour conclure, notre travail soulève la question suivante : déléguer les tâches d'administration aux concepteurs remplacera t-il les advisors?

6 Conclusion

Dans ce chapitre, nous avons montré la contribution de notre approche de gestion des besoins sur la phase physique des bases de données volumineuses comme les entrepôts de données. Au lieu de sélectionner les structures d'optimisation lors de la phase physique, nous avons proposé une méthodologie permettant de les sélectionner lors de la conception de la base de données. Cette démarche pourrait ré-

FIGURE 6.8 – Nouvelle vision de conception des bases de données.

duire considérablement les tâches de l'administrateur. Nous avons montré l'intérêt de notre approche sur deux structures d'optimisation, les index de jointure binaire et la fragmentation horizontale. Une validation sur le système de gestion de bases de données ontologique OntoQL est proposée en utilisant les données du banc d'essai SSB.

Dans le chapitre suivant, nous proposons un prototype d'outil implémentant nos contributions.

Chapitre 7

Prototype de validation

Sommaire

1 Introduction 123

1.1 Langages et outils utilisés 123

2 Architecture fonctionnelle de OntoReqTool 123

2.1 Les modules de la partie intégration 123

2.2 Les modules de la phase d'exploitation 127

3 Conclusion 132

Résumé. Dans ce chapitre, nous présentons un outil, appelé *OntoReqTool* assistant les concepteurs dans leur tâches d'intégration et d'exploitation des besoins. Il offre aux concepteurs la possibilité de visualiser l'ontologie de domaine partagée, l'extraction de leurs ontologies locales, l'expression des besoins selon leurs ontologies locales, la transformation de chaque langage de besoins vers le modèle pivot, et le raisonnement. Pour la partie exploitation, l'outil offre aux concepteurs la possibilité de persister les besoins et de les utiliser pour la conception physique.

1 Introduction

Pour que les concepteurs d'une entreprise étendue s'approprient les concepts et les méthodes proposées, le développement d'un outil est une tâche primordial. L'une des caractéristiques de cet outil est sa simplicité. L'outil développé couvre les deux principales phases de nos contributions : l'intégration des besoins hétérogènes et leur exploitation. Pour la première contribution, l'outil doit d'abord offrir aux concepteurs la possibilité de charger une ontologie dans l'espace des ontologies et de la visualiser. L'affichage doit être ergonomique offrant diverses fonctionnalités de navigation comme l'accès à n'importe quelle hiérarchie de l'ontologie. Une fois chargée, un concepteur peut facilement identifier une partie de l'ontologie qui correspond à sa tâche et qui représentera son ontologie locale. Ensuite, il exprime ses besoins selon son langage favori. Une fois les besoins exprimés, l'ensemble des langages sont transformés sur le modèle pivot. Toutes les transformations sont visibles aux concepteurs. Le processus de raisonnement quant à lui, permet de montrer aux concepteurs l'ensemble des besoins conflictuels.

Pour la partie exploitation, l'outil offre aux concepteurs la possibilité de visualiser l'ensemble des besoins valides, les persister, les transformer en requêtes et lancer le type d'optimisation (les index de jointure binaire ou la fragmentation horizontale) qu'il souhaite considérer durant sa conception physique.

Dans les sections suivantes, nous commençons par détailler les langages et outils utilisés pour réaliser notre outil, baptisé *OntoReqTool*. Nous décrivons ensuite son architecture fonctionnelle et l'ensemble de ses modules et ses composants.

1.1 Langages et outils utilisés

Nous avons travaillé avec l'éditeur d'ontologie *Protégé 3.4.4* [38] pour gérer des ontologies, quant à la partie raisonnement, nous avons utilisé le moteur JESS disponible sous l'éditeur d'ontologié Protégé. Pour la partie transformation Model-to-Model, nous avons utilisé l'outil *Ecore* et *ATL* [39]*(ATLAS Transformation Language)* de l'environnement Eclipse. Eclipse est un environnement de développement intégré qui permet d'incorporer des plugins pour utiliser les techniques d'IDM.

Pour persister les besoins, nous avons utilisé le système de gestion de bases de données OntoDB comme nous l'avons déjà évoqué dans le chapitre précédent.

2 Architecture fonctionnelle de OntoReqTool

L'architecture générale de notre outil est composée de huit modules dont cinq concernant la partie intégration et trois la partie exploitation (Figure 7.1).

2.1 Les modules de la partie intégration

les modules concernant la partie d'intégration des besoins se déclinent en cinq parties : (a) le chargement et la visualisation de l'ontologie, (b) le couplage de l'ontologie globale aux besoins, (c) l'extraction

38. http://protege.stanford.edu/download/registered.html
39. http://www.eclipse.org/atl/

FIGURE 7.1 – Architecture fonctionnelle de l'outil (OntoReqTool).

de l'ontologie locale, (d) la transformation des modèles de besoins vers le langage pivot, et (e) la partie raisonnement.

1. Chargement et visualisation de l'OG : Ce module permet de charger et de visualiser l'ontologie globale sous forme d'une arborescence. Grâce à l'utilisation de l'éditeur d'ontologie *Protégé 3.4.4* qui possède une interface utilisateur graphique (GUI). Cette dernière permet de manipuler aisément tous les éléments de l'ontologie : classe, méta-classe, propriété, instance, etc. La figure 7.3 illustre un exemple d'affichage de l'ontologie globale *LUBM*[62] décrivant le domaine universitaire. L'affichage de visualise les classes "Person", "Student", etc.), les propriétés ("name", "age", etc.) et les instances ontologiques.

2. Projection des besoins sur l'ontologie globale : ce module consiste essentiellement à étendre l'ontologie globale par des méta-classes constituant le méta-modèle générique des besoins. Le concepteur peut alors définir ses besoins à partir de l'ensemble des concepts et propriétés de l'ontologie. L'éditeur d'ontologies *Protégé 3.4.4* permet de manipuler les méta-classes de l'ontologie locale afin d'ajouter d'autres méta-classes. La figure 7.3 présente l'ontologie des besoins (OntoReq) qui est définie par l'ajout des méta-classes constituant le méta-modèle générique. Pour chaque besoin, le concepteur sélectionne les cordonnées d'un besoin (action, résultat et critère) en choisissant les concepts et propriétés pertinents de l'ontologie globale.

FIGURE 7.2 – Diagramme de cas d'utilisation de notre outil.

3. Extraction des ontologies locales (OL) : ce module consiste à extraire les ontologies locales à partir de l'ontologie globale. Chaque ontologie locale peut être vu comme un fragment (partiel ou total) de l'ontologie globale. Elle représente les concepts nécessaires utilisés pour spécifier les besoins au niveau ontologique. Son extraction s'effectue en utilisant le plugin de modularité *ProSé* disponible sous l'éditeur d'ontologies *Protégé 3.4.4*. Ce plugin permet aux concepteurs d'ajouter ou de supprimer des classes et des propriétés au cas où les OL définies ne couvrent pas tous leurs besoins, tout en assurant la cohérence et la complétude des ontologies définies. Dans notre étude de cas, nous avons généré trois ontologies locales correspondant aux trois langages de besoins utilisés (langage orienté buts, use case d'UML, et le modèle de traitements de Merise). La figure 7.4 présente un exemple d'une ontologie locale des buts (OntoGoal).

4. Transformation des besoins en modèle générique : l'objectif de ce module est d'assurer une transformation des instances ontologiques qui sont définies localement dans les ontologies locales des besoins (OntoGoal, OntoUcase et OntoMct) en un langage générique (Ontologie Pivot nommé (OntoPivot)). Cette transformation est réalisée par des règles décrivant les correspondances entre les entités des langages sources et celles de chaque langage cible. Plus concrètement, la transformation se situe entre les méta-modèles sources "OntoGoal-Metamodel", "OntoUcase-Metamodel" et "OntoMCT-Metamodel" et le méta-modèle cible "OntoReq-Metamodel" (Figure7.5). L'ensemble de règles de transformation sont décrits dans l'annexe 2.5.

FIGURE 7.3 – Projection des besoins sur l'OG (OntoReq).

5. Le raisonnement sur les besoins : ce module consiste essentiellement à raisonner sur l'ontologie de besoins afin d'identifier ceux qui sont contradictoires et ambigüs. Le mécanisme de raisonnement utilisé par notre méthode est implémenté en exploitant les règles de raisonnement décrivant les relations sémantiques entre les besoins. Ces règles sont définies en utilisant le langage SWRL [40] combinées au moteur d'inférence JESS comme illustré dans la figure 7.6.

40. http://flashinformatique.epfl.ch/spip.php?article1212

FIGURE 7.4 – Ontologie locale des buts (OntoGoal).

2.2 Les modules de la phase d'exploitation

Deux modules caractérisent la phase d'exploitation : la persistance des besoins et la sélection des structures d'optimisation.

1. Persistance des besoins dans un entrepôt de données sémantique : dans ce module l'ensemble des besoins sont stockés dans une base de données sémantique développée au laboratoire. Cette base est dotée d'un langage de requêtes sémantique OntoQl [69].

   Afin de permettre au concepteur de persister ses besoins, nous lui offrons une interface graphique (pour l'instanciation du méta-modèle en fonction des concepts ontologiques) (figure 7.8).

   Prenons un besoin simple afin de montrer la manipulation des besoins via notre prototype qui utilise OntoQL comme langage de requêtes. Cette manipulation doit assurer la création, la modifi-

FIGURE 7.5 – Transformation des besoins en modèle générique.

cation, l'interrogation et la transformation en requête des besoins.

Besoin : *Le système doit permettre à l'administration de supprimer les étudiants du département "médecine" ayant une moyenne inférieure à 10.*

- Modélisation (instanciation) du besoin via l'interface du prototype : suivant la description du modèle des besoins détaillée dans le chapitre 5, notre besoin sera modélisé comme suit.
  - Result : suppression des étudiants.
  - Action : le résultat peut souvent être raffiné en plusieurs actions. Dans notre exemple, Il pourrait s'agir tout d'abord d'une suppression de ces entités *étudiant*, puis d'une suppression des entités ayant une relation avec ces dernières. OntoQL réalise ce processus implicitement via une seule requête (action).
  - Criterion : Le résultat peut également être contrôlé par une ou plusieurs métriques. Ici, deux critères sont mises en évidence :
    - Criterion 1 : les étudiants doivent appartenir au département de *Médecine*.
    - Criterion 2 : les étudiants doivent avoir une moyenne inférieure à 10.

    Autrement dit, on ne peut exécuter les actions (suppression) sur les entités étudiants, que s'ils remplissent ces deux critères.
- Création des besoins : une fois les champs correspondants sont bien remplis, des requêtes OntoQL sont générées automatiquement afin de persister (créer) les besoins introduits par l'utilisateur dans la base de données (figure7.8). Pour cet exemple, les requêtes générées sont :

FIGURE 7.6 – Raisonnement sur les besoins.

```
INSERT into #Result (#name, #On, #Description) VALUES ('Delete', (SELECT p.#oid)
FROM #Class p WHERE p.#name = 'Student'), DELETE the students of medical departement,
owning an average lesser than 10')

INSERT into #Criterion (#name, #relation, #source, #target ) VALUES ('Criterion1', 'equal to',
(SELECT p.#oid FROM #Property p WHERE p.#name = 'name'
AND p.#scope.#name = 'Departement'), 'Medecine')

INSERT into #Criterion (#name, #relation, #source, #target )
VALUES ('Criterion2', 'lesser than',
(SELECT p.#oid FROM #Property p WHERE p.#name = 'average'
AND p.#scope.#name = 'Student'), '10')

UPDATE #Criterion SET #andC = (SELECT max (c.#oid))
FROM #Criterion c) WHERE #oid = (SELECT max (cr.#oid)
FROM #Criterion cr WHERE cr.#name = 'Criterion1')

INSERT into #Goal (#name, #description, #hasResult, #hasCriterion )
VALUES ('Goal_0001',etc)

INSERT into #Requirement (#IdReq, #NameReq, #DescriptionReq, #ContextReq, #PriorityReq,
#DefinedBy, #Achieves, #ComposeOf, #MesuredBy, #IsResponsible)
VALUES ('001', 'Req_00000001', 'The system shall allow the administration
to delete the students belongin to the medical departement and having an
average lesser than 10', 'Scolarity', 1, Student, (SELECT max (r.#oid)
FROM #Resilt r WHERE r.#name = 'Result1'), null, (SELECT max (c.#oid FROM #Criterion c
WHERE c.#name = 'Criterio1'), Dean of Studies)
```

Une fois les besoins persistés, nous avons la possibilité de les modifier, de les interroger ou de

FIGURE 7.7 – Interface graphique de la phase d'exploitation de l'ED.

```
persistanceQueries.txt
1 INSERT INTO #Result (#name, #definedBy, #description) VALUES ('Delete', (SELECT p.#oid FROM #Class p WHERE p.#name = 'Student'),
2       'Delete the students of medical department, owning an average lesser than 10')
3 INSERT INTO #criterion (#name, #opr, #definedBy, #literal) VALUES ('Critere2', 'equal to',
4       (SELECT p.#oid FROM #Property p WHERE p.#name = 'name' AND p.#scope.#name = 'Department'),'Medecine')
5 INSERT INTO #criterion (#name, #opr, #definedBy, #literal) VALUES ('Critere1', 'lesser than',
6       (SELECT p.#oid FROM #Property p WHERE p.#name = 'average' AND p.#scope.#name = 'Student'),'10')
7 UPDATE #criterion SET #andC = (SELECT max(c.#oid) FROM #criterion c) WHERE #oid = (SELECT max(cr.#oid) From #Criterion cr WHERE cr.#name = 'Critere1')
8 INSERT INTO #Requirement (#idReq, #nameReq, # descriptionReq, #contextReq, #priorityReq, #defined_for, #achieves, #composedOf, #mesuredBy, #isResponsible)
9       values ('1', 'Req_1384339723956', 'The system shall allow the administration to delete the students belonging to the medical department and having an average lesser than 10',
10      'Scolarity' , 1, Student, (select max(r.#oid) from #Result r WHERE r.#name = 'Result1'), null, (select max(c.#oid) from #Criterion c WHERE c.#name = 'Critere1'), Dean of Studies)
```

FIGURE 7.8 – Persistance des besoins dans l'ED.

les transformer principalement en requêtes SQL.

– Modification des besoins : notre interface ne permet pas de modifier le besoin en tant que tel vu les différents paramètres pouvant être modifiés. Cependant le besoin peut être supprimé et réintroduit, ou modifié par la console via OntoQL. A ce propos, les requêtes suivantes montrent une modification du besoin initial, qui consiste à lui ajouter un critère (L'âge des étudiants concernés doit être supérieur à 40 ans) :

```
INSERT into #Criterion (#name, #relation, #source, #target )
VALUES ('Criterion3', 'greater than',
(SELECT p.#oid FROM #Property p WHERE p.#name = 'age' AND p.#scope.#name = 'Student'),
 '40')

UPDATE #Criterion SET #andC = (SELECT max (c.#oid)) FROM #Criterion c)
WHERE #oid = (SELECT max (cr.#oid) FROM #Criterion cr WHERE cr.#name = 'Criterion2')
```

– Interrogation des besoins : si l'utilisateur de notre prototype désire afficher les besoins persistés jusque là, soit par curiosité ou dans le but d'en modifier/supprimer certains. Les requêtes générées suivent le processus suivant :

(a) Nous parcourons d'abord les besoins, via la requête OntoQL

```
SELECT #oid, #name FROM #Requirement
```

(b) Pour chaque besoin (résultat retourné par la requête précédente), nous extrayons ses résultats comme suit :

```
SELECT r.#Achieves FROM #Requirement WHERE r.#oid = + rsultSet.getInt (1));
```

(c) Pour chaque *Result* (résultat retourné par la requête précédente), nous extrayons les propriétés/concepts ontologiques sur lesquels il porte :

```
SELECT r.#On FROM #Result r WHERE r.#oid = "+ rsultSet2.getInt (1));
```

De la même manière, nous traitons les critères (respectivement les actions) s'il en existe, sauf que nous prenons le chemin de *Criteria* (respectivement de *Action* et cela autant de fois qu'il y a de critères (respectivement d'actions composant le résultat).

2. Génération des requêtes SQL : les besoins des utilisateurs, seront traduits en requêtes ontologiques, i.e. manipulant des classes d'ontologie. Cette transformation est déjà détaillée dans la section du chapitre 6. Pour cet exemple, la transformation aboutit à la requête suivante : (figure7.9).

```
DELETE{} FROM {STUDENT}
WHERE {average < 10 AND nameDep = 'Medecine'}
```

3. La sélection des structures d'optimisation : dans ce module, l'ensemble des requêtes générées sont prises en compte par les algorithmes de sélection des structures d'optimisation. Dans cette étude, nous avons considéré le problème de la sélection des index de jointure binaire et la fragmentation horizontale (figure 7.8). Ce module peut exécuter tout type de services permettant la sélection des structures d'optimisation.

extractedQueries.txt

```
1 SELECT stu.name
2 FROM STUDENT stu, DEPARTMENT dep
3 WHERE stu.average> 10
4 AND dep.name = 'Medecine'
```

FIGURE 7.9 – Requêtes SQL générées.

3 Conclusion

Dans ce chapitre, nous avons présenté un outil, appelé, *OntoReqTool*, permettant d'assister les concepteurs dans leur tâche d'intégration et d'exploitation des besoins. Il offre des interfaces conviviales et simples pour permettre aux concepteurs de naviguer et de visualiser l'ensemble des tâches effectuées. Une autre caractéristique de l'outil est sa capacité à se connecter aux services de conception physique dans le but de sélectionner des structures d'optimisation à partir des besoins.

Dans le chapitre suivant, nous établissons un bilan des contributions apportées et nous traçons différentes perspectives de recherche qui pourraient être menées ultérieurement, tant du point de vue théorique que pratique.

Chapitre 8

Conclusion générale

Sommaire

1 Conclusion 135
2 Perspectives 136

1 Conclusion

Dans cette thèse, nous avons étudié le problème de l'intégration et de l'exploitation des besoins dans le cadre des entreprises étendues. Dans cette section nous résumons les principaux apports de nos travaux.

Nous avons d'abord positionné le rôle des besoins et leurs intérêts dans la conception d'applications avancées. Les besoins se trouvent au cœur des systèmes d'intégration, qui regroupaient initialement les données, les applications et les plateformes. Nous avons également identifié l'importance de les intégrer dans le cadre d'une entreprise étendue qui peut regrouper un grand nombre de participants pour élaborer un projet donné. Cette intégration pose énormément de problèmes d'hétérogénéité : hétérogénéité sémantique et syntaxique, diversité des langages de modélisation des besoins, identification des besoins conflictuels, etc. Après avoir fait le parallèle et surtout exploité les travaux d'intégration de sources de données, nous avons proposé une démarche d'intégration des besoins dirigée par des ontologies conceptuelles qui offrent un vocabulaire unifié à l'ensemble des concepteurs. Pour répondre au problème d'hétérogénéité des langages de modélisation des besoins, une solution d'intégration basée sur un modèle pivot est proposée. Nous avons également montré l'intérêt des besoins pour le cycle de vie de conception des bases et des entrepôts de données, plus particulièrement lors de la phase physique. Ces contributions ont été motivées et expérimentées par l'utilisation des données issues de bancs d'essai utilisées par la communauté d'entreposage de données.

1.1 Bilan des contributions

Vu la dimension multidisciplinaire de notre problème, nous avons été amené à faire deux états de l'art conséquents, le premier sur l'ingénierie des besoins et l'ingénierie dirigée par des modèles, le second sur les ontologies et leurs rôles dans la résolution des problèmes d'intégration des données et des besoins.

Notre première contribution est la proposition d'une solution d'intégration des besoins hétérogènes unifiant les vocabulaires et les langages de modélisation des besoins utilisés. Pour unifier les vocabulaires, nous avons travaillé sous l'hypothèse de l'existence d'une ontologie de domaine. Cette hypothèse est tout à fait acceptable vu l'émergence des ontologies de domaine dans des univers aussi divers que l'ingénierie, la médecine, l'environnement, etc. Chaque concepteur référence cette ontologie pour exprimer ses besoins. Pour unifier les langages de modélisation des besoins et vu leur diversité, nous avons étudié trois langages, précisément: Use case d'UML, le langage orienté buts, et le modèle de traitement de la méthode Merise. Ce choix est motivé par leur popularité et leur utilisation intensive dans les projets industriels et académiques. Cette étude nous a permis de définir un langage pivot factorisant l'ensemble des langages étudiés. Un couplage entre l'ensemble des méta modèles utilisés et l'ontologie est établi. Un scénario d'intégration des besoins est alors défini selon ce langage pivot. Chaque concepteur extrait d'abord une ontologie locale qui référence l'ontologie globale satisfaisant ses exigences. Une fois extraite, il décrit ses besoins en fonction de son ontologie locale. Ensuite, ses besoins sont transformés à l'aide des règles sur le modèle pivot. Dans ce scénario, nous avons utilisé des transformations de type *Model-to-Model*.

La deuxième contribution consiste à exploiter la puissance de l'ingénierie dirigée par les modèles pour offrir un autre scénario d'intégration. Au lieu d'utiliser des transformations *Model-to-Model*, nous

avons défini un méta-modèle représentant les besoins. Les trois modèles utilisés deviennent alors des instances de ce méta-modèle. Dans ce cas, chaque concepteur donne seulement son langage préféré pour concevoir une application, à la suite de quoi il extrait son langage des besoins couplé à son ontologie. La présence de l'ontologie permet également de raisonner sur l'ensemble des besoins émis par les concepteurs. Deux architectures d'implémentation de raisonnement sont considérées (i) *ship whole* et (ii) *reason as needed*. Dans le premier scénario chaque concepteur envoie ses besoins sans aucun traitement local au modèle générique qui effectue le raisonnement pour identifier les besoins conflictuels. Cette solution peut être coûteuse si le nombre de besoins est important. Pour pallier ce problème, nous avons proposé l'architecture *reason as needed* qui permet à chaque concepteur de raisonner localement autant qu'il le peut, puis d'envoyer ensuite des besoins valides au modèle générique, qui à son tour effectue d'autres raisonnements. L'évaluation des deux solutions a montré l'intérêt du raisonnement, ainsi que la supériorité de la solution *reason as needed* sur la solution *ship whole* en termes de temps de traitement.

Notre troisième contribution a montré l'intérêt de l'utilisation des besoins dans la phase physique. Rappelons que cette dernière représente une tâche coûteuse auprès des administrateurs. Pour alléger cette tâche, ces derniers utilisent des *advisors* commerciaux. Notre vision est complètement différente, et consiste à déléguer certaines tâches de la conception physique aux concepteurs. Nous avons montré que le processus de sélection des structures d'optimisation se fait à l'aide d'un ensemble de requêtes extraites à partir des logs. Si nous analysons les besoins, nous pouvons facilement extraire ces requêtes. En conséquence, le processus de sélection peut se faire à l'aide des besoins. Nous avons montré l'intérêt et la faisabilité de notre approche sur la sélection de deux structures d'optimisation, une redondante qui représente les index de jointure binaire et l'autre non redondante représentant la fragmentation horizontale. Nous avons implémenté notre démarche sur le système de gestion de bases de données OntoDB développé au sein de notre laboratoire. Les besoins sont stockés au sein de cette base de données et des programmes de génération de requêtes à partir de ces besoins sont développés. Nous avons comparé nos algorithmes de sélection dirigés par les besoins sur les données du banc d'essai SSB avec ceux utilisés par les requêtes. Rappelons que le banc d'essai SSB offre des requêtes sous forme de besoins et SQL. Les résultats sont pratiquement similaires, ce qui montre que certaines tâches de conception physique peuvent être déléguées aux concepteurs.

L'ensemble de nos contributions est prototypé afin d'assister les concepteurs dans leurs tâches d'expression, d'intégration et d'exploitation des besoins.

2 Perspectives

De nombreuses perspectives tant à caractère théorique que pratique peuvent être envisagées. Dans cette section nous présentons succinctement celles qui nous paraissent être les plus intéressantes.

2.1 Autres langages de modélisation des besoins

Nos propositions sont dirigées par trois langages de modélisation des besoins appartenant à la catégorie semi-formelle. Il serait intéressant de considérer d'autres types de langages appartenant à la catégorie formelle, telle que la méthode B, et d'évaluer leur impact sur la conception physique.

2.2 Etude du passage à l'échelle de nos propositions

Pour évaluer la robustesse de nos propositions concernant l'intégration de besoins, notamment la partie raisonnement, il est nécessaire de considérer un grand nombre de langage des besoins. Cette évaluation doit se faire en développant des modèles de coût impliquant les aspects économiques, le temps de calcul et le temps de transfert. Ces modèles de coût peuvent être utilisés pour développer d'autres stratégies de raisonnement.

2.3 Définition des architectures d'intégration

Dans cette thèse nous n'avons pas pris en compte l'architecture de l'intégration des besoins. Rappelons qu'il existe deux principales architectures pour supporter un système d'intégration : la solution médiateur et la solution matérialisée. Dans l'architecture de type médiateur, l'intégration s'effectue en deux étapes : (i) le système d'intégration génère, à partir d'une requête utilisateur, autant de sous requêtes qu'il y a de partenaires à interroger, (ii) il construit alors la réponse finale à partir des résultats de sous requêtes et la présente à l'utilisateur. Cette architecture repose sur deux composants essentiels : le médiateur et l'adaptateur. Dans l'architecture matérialisée, l'intégration des données s'effectue également en deux étapes : (i) les besoins issus des différentes sources sont dupliqués dans un entrepôt de données, (ii) les requêtes des utilisateurs sont posées directement sur ce dernier. Dans les deux scénarii d'architecture, il serait intéressant d'étudier à quel niveau effectuer la partie raisonnement parmi les choix suivants: au niveau source, au niveau ETL (Extraction, Transformation, Loading) ou au niveau du schéma global.

2.4 Prise en compte de l'évolution

Dans une entreprise étendue, l'évolution est le maître mot. Elle peut concerner l'ontologie et les besoins. L'étude de l'évolution des besoins des utilisateurs est un aspect important pour assurer la faisabilité de l'approche dans une vraie entreprise étendue. L'évolution des besoins ne peut pas être traitée sans prendre en compte l'architecture supportant le système d'intégration des besoins. Si l'architecture matérialisée est adoptée, l'évolution des besoins peut être traitée au niveau ETL. Cette évolution impactera directement la conception physique, et cet impact se ressentira immédiatement contrairement à l'approche classique, où l'administrateur attend que les fichiers logs de requêtes soient établis. L'étude de l'évolution des ontologies manipulées est un enjeu important qui doit être exploré. Cette évolution peut concerner différents aspects, comme l'évolution du schéma de l'ontologie ou l'évolution des instances ontologiques.

2.5 Elaboration d'un cycle de vie de conception d'applications avancées

L'équipe ingénierie des données et des modèles réalise depuis quelques années des travaux sur le cycle de vie de conception d'application avancées, qui regroupent les bases de données, les entrepôts de données, et les bases de données sémantiques. Ces travaux couvraient la modélisation conceptuelle, la phase ETL, la modélisation logique, la modélisation physique, la personnalisation et la recommandation des requêtes. Le travail de cette thèse est fondamentale pour l'élaboration de ce cycle. La prise

en compte des besoins et des acteurs pourrait permettre de proposer facilement des solutions personnalisées aux utilisateurs. Un autre intérêt est lié au fait que les besoins sont stockés dans la base de données, nous pouvons donc facilement annoter les résultats d'exploitation qui pourraient être fournis par des algorithmes de fouille de données exécutés sur la base de données. En conséquence, nous pourrions associer à chaque connaissance trouvée la ou les personnes potentiellement intéressées. Un autre élément qui pourrait être étudier est le *crowdsourcing*, où l'ensemble des concepteurs peuvent contribuer à l'amélioration du produit final en interagissant sur toute la chaîne.

Bibliographie

[1] The standish group, chaos. `http://www.standishgroup.com/chaos.html`, 1995.

[2] Industrial automation systems and integration - parts library - part 42: Description methodology: Methodology for structuring parts families. Technical report, 1998.

[3] Industrial automation systems and integration - parts library - part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content. Technical report, 2004.

[4] The standish group, chaos summary for 2010. `http://insyght.com.au/special/2010CHAOSSummary.pdf`, 2010.

[5] User requirements notation (urn) - language definition. Technical report, 2012.

[6] A. Abran, J.W. Moore, P. Bourque, and R.E. Dupuis. *Guide to the Software Engineering Body of Knowledge.* http://www.swebok.org/.IEEE Computer Society, 2004.

[7] J.R. Abrial. *The B-book: assigning programs to meanings*. Cambridge University Press, New York, USA, 1996.

[8] D. Agrawal, S. Das, and A. El Abbadi. *Data Management in the Cloud: Challenges and Opportunities*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[9] S. Agrawal, S. Chaudhuri, L. Kollar, A. Marathe, V. Narasayya, and M. Syamala. Database tuning advisor for microsoft sql server 2005. In *In Proceedings of the International Conference on Very Large Databases, (VLDB'04)*, pages 1110–1121, 2004.

[10] Y. Aït Ameur, M. Baron, and P. Girard. Formal validation of hci user tasks. In *the International Conference on Software Engineering Research and Practice (SERP'03)*, pages 732–738. CSREA Press, 2003.

[11] G. Dahlstedt Asa and A. Persson. Requirements interdependencies - moulding the state of research into a research agenda. In *Ninth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2003), held in conjunction with (CAiSE 2003)*, pages 71–80, 2003.

[12] F. Baader, D. Calvanese D. McGuinness, D. Nardi, and P. Patel-Schneider. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, USA, 2003.

[13] V. M. Werneck Bejamim, A. O. de Padua Antonio, and J. Cesar Sampaio do Prado. Comparing gore frameworks: i-star and kaos. In *Ibero-American Workshop of Engineering of Requirements, (WER'09)*, 2009.

[14] L. Bellatreche. *Contributions à la Conception et l'Exploitation des Systèmes d'Intégration de Données*. PhD thesis, HDR, Habilitation à diriger les recherches, ENSMA, 2009.

[15] L. Bellatreche. Optimization and tuning in data warehouses. In *Encyclopedia of Database Systems*, pages 1995–2003. 2009.

[16] L. Bellatreche, K. Boukhalfa, P. Richard, and K.Y. Woameno. Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(4):1–23, 2009.

[17] L. Bellatreche, A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent. A personalization framework for olap queries. In *ACM Eighth International Workshop on Data Warehousing and OLAP (DOLAP'05)*, pages 9–18, 2005.

[18] L. Bellatreche, S. Khouri, I. Boukhari, and R. Bouchakri. Using ontologies and requirements for constructing and optimizing data warehouses. In *Proceedings of the 35th International Convention (MIPRO)*, pages 1568–1573, 2012.

[19] L. Bellatreche, G. Pierra, D. Nguyen Xuan, H. Dehainsala, and Y. Aït Ameur. An a priori approach for automatic integration of heterogeneous and autonomous databases. In *15th International Conference on Database and Expert Systems Applications (DEXA'04)*, pages 475–485, 2004.

[20] L. Bellatreche, D. Nguyen Xuan, H., G. Pierra, and H. Dehainsala. Contribution of ontology-based data modeling to automatic integration of electronic catalogues within engineering databases. *Computers in Industry Journal, Elsevier*, 57(8-9):711–724, 2006.

[21] D. Beneventano, S. Bergamaschi, S. Castano, G.Malvezzi A. Corni, R. Guidetti, M. Melchiori, and M. Vincini. Information integration: The momis project demonstration. *In the VLDB journal*, pages 611–614, 2000.

[22] N. Berkani, S. Khouri, and L. Bellatreche. Generic methodology for semantic data warehouse design: From schema definition to etl. In *Intelligent Networking and Collaborative Systems (INCoS)*, pages 404–411, 2012.

[23] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2005.

[24] R. Bouchakri and L. Bellatreche. On simplifying integrated physical database design. In *Proceedings of the 15th International on Advances in Databases and Information Systems,(ADBIS'11)*, pages 333–346. Springer-Verlag, 2011.

[25] I. Boukhari, L. Bellatreche, and S. Jean. An ontological pivot model to interoperate heterogeneous user requirements. In *Proceedings of the 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISOLA 2012)*, pages 344–358. LNCS, Springer, 2012.

[26] I. Boukhari, L. Bellatreche, and S. Khouri. Efficient, unified, and intelligent user requirement collection and analysis in global enterprises. In *(To appear) in proceedings of the 15th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2013), Vienna, Austria.* LNCS, Springer, 2013.

[27] K. K. Breitman and J.C. Sampaio Leite. Ontology as a requirements engineering product. In *Proceedings of the 11th IEEE International Conference on Requirements Engineering, (RE'03)*, pages 309–319. IEEE Computer Society, 2003.

[28] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development me-

thodology. *Autonomous Agents and Multi-Agent Systems*, 8:203–236, 2004.

[29] B. Brigitte, S. Sylvie, and Av.J.B. Clément. Terminae: A linguistics-based tool for the building of a domain ontology. In *Knowledge Acquisition, Modeling and Management*, volume 1621, pages 49–66. Springer Berlin Heidelberg, 1999.

[30] E. Brottier, B. Baudry, Y.Traon, D. Touzet, and B. Nicolas. Producing a global requirement model from multiple requirement specifications. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference, (EDOC'07)*, page 390. IEEE Computer Society, 2007.

[31] G. Canahuate, T. Apaydin, A. Sacan, and H. Ferhatosmanoglu. Secondary bitmap indexes with vertical and horizontal partitioning. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, (EDBT'09)*, pages 600–611, 2009.

[32] V. Castaneda, L. Ballejos, M.Laura Caliusco, and M. Rosa Galli. The use of ontologies in requirements engineering. *Global Journal of Researches in Engineering (GJRE)*, 10 Issue 6:2–8, 2010.

[33] Chee-Yong Chan. *Indexing Techniques in Decision Support Systems*. PhD thesis, The University of Wisconsin-Madison, 1999.

[34] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, pages 3–14. VLDB Endowment, 2007.

[35] P. Chen. The entity relationship model - towards a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[36] B. Chevallereau, A. Bernard, and P. Mévellec. Améliorer les performances de l'industrie logicielle par une meilleure compréhension des besoins. In *CIGI, Conférence Internationale de Génie Industriel, France*, 2009.

[37] F. Chimene. Ontodb2: support of multiple ontology models within ontology based database. In *Proceedings of the 2008 EDBT Ph.D. workshop, (Ph.D.'08)*, pages 21–27. ACM, 2008.

[38] M. Christel and K. Kang. Issues in requirements elicitation. Technical report, 1992.

[39] P. Cimiano and J. Völker. Text2onto. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems, (NLDB)*, pages 227–238. Springer, 2005.

[40] P. Coret, J. Richard, E. Talavet, and T. Trofimoya. Introduction a owl, langage xml d'ontologies. Technical report, 2006.

[41] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.

[42] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30 (VLDB'04)*, pages 1098–1109. VLDB Endowment, 2004.

[43] A. Dardenne, A.van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science. Computer. Program.*, 20:3–50, 1993.

[44] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *Proceedings of the IFIP TC2/WG2.6 Eighth Working Conference on Database Semantics- Semantic Issues in Multimedia Systems, (DS-8)*, pages 351–369, 1998.

[45] H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: An ontology-based data-

base for data intensive applications. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA'07)*, pages 497–508. Springer-Verlag, 2007.

[46] A. Doan, A. Y. Halevy, and Z. G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012.

[47] P. du Bois. *The ALBERT II Language*. PhD thesis, Facultés Universitaires Notre Dame de la Paix, Namur, 1995.

[48] D. V. Dzung and A. Ohnishi. Ontology-based reasoning in requirements elicitation. In *IEEE International Conference on Software Engineering and Formal Methods(SEFM)*, pages 263–272, 2009.

[49] Frederico T. Fonseca and James E. Martin. Learning the differences between ontologies and conceptual schemas through ontology-driven information systems. *Journal of the Association for Information Systems*, 8(2), 2007.

[50] G. Gardarin. Intégration de données et applications via xml. `http://georges.gardarin.free.fr/Cours_Total/X6-BusinessIntegration.ppt`.

[51] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani. Formal reasoning techniques for goal models. *Journal of Data Semantics.*, 1:1–20, 2004.

[52] Claude Girault and Rudiger Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications*. Springer-Verlag New York, Inc., 2001.

[53] F. Goasdoué, V. Lattés, and M. C. Rousset. The use of carin language and algorithms for information integration: The picsel system. *International Journal of Cooperative Information Systems (IJCIS)*, 9(4):383–401, 2000.

[54] C.H. Goh, S. Bressan, E. Madnick, and M.D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–293, 1999.

[55] A. Goknil, i. Kurtev, k. Berg, and j. Veldhuis. Semantics of trace relations in requirements models for consistency checking and inferencing. *Softwar System Model.*, 10:31–54, 2011.

[56] A. Goknil, I. Kurtev, and K. Berg. A metamodeling approach for reasoning about requirements. In *Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications, (ECMDA-FA'08)*, pages 310–325. Springer-Verlag, 2008.

[57] M. Golfarelli. From user requirements to conceptual design in data warehouse design - a survey. In *Data Warehousing Design and Advanced Engineering Applications : Methods for Complex Construction*, pages 1–16. IGI Global, 2009.

[58] M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*. McGraw Hill, 2009.

[59] M. Golfarelli, S. Rizzi, and P. Biondi. myolap: An approach to express and evaluate olap preferences. *IEEE Trans. Knowl. Data Eng.*, 23(7):1050–1064, 2011.

[60] S.J. Greenspan. *Requirements modeling: a knowledge representation approach to software requirements definition*. PhD thesis, 1984.

[61] T. R. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5, pages 199–220, 1993.

[62] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, pages 158–182, 2005.

[63] A. Halevy, A. Rajaraman, and J. Ordille. Data integration: The teenage years. In *Pro-*

ceedings of the 32Nd International Conference on Very Large Data Bases (VLDB), pages 9–16, 2006.

[64] C. Hardebolle. *Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes*. PhD thesis, Université Paris-Sud XI, 2008.

[65] C. L. Heitmeyer, R. D. Jeffords, and B.G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. Softw. Eng. Methodol.*, 5:231–261, 1996.

[66] W.H. Inmon. *Building the Data Warehouse*. QED Information Sciences, Inc., 1992.

[67] M. Jackson. *Problem frames: analyzing and structuring software development problems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[68] I. Jacobson and l. Spence K. Bittner. *Use Case Modeling*. Addison Wesley Professional, 2002.

[69] S. Jean. *OntoQL, un langage d'exploitation des bases de données à base ontologique*. PhD thesis, Université de Poitiers., 2007.

[70] S. Jean, Y. Aít Ameur, and G. Pierra. Querying ontology based database using ontoql (an ontology query language). In *Proceedings of the 2006 Confederated international conference on On the Move to Meaningful Internet Systems: CoopIS, DOA, GADA, and ODBASE, (ODBASE'06/OTM'06)*, pages 704–721. Springer-Verlag, 2006.

[71] S. Jean, G. Pierra, and Y. Aït Ameur. Domain ontologies: A database-oriented analysis. In *Proceedings of the Second International Conference on Web Information Systems and Technologies: Internet Technology / Web Interface and Applications, WEBIST (1)*, pages 341–351, 2006.

[72] H. Jones. *Raisonnement à base de règles implicatives floues-Inférence et Sémantique*. PhD thesis, Université Toulouse III- Paul Sabataire, 2007.

[73] M. Jun, L. Guofu, L. Hao, and X. Yanqiu. The knowledge sharing based on plib ontology and xml for collaborative product commerce. In *Proceedings of the International Conference on Web Information Systems and Mining, (WISM'09)*, pages 518–526. Springer-Verlag, 2009.

[74] k. Pohl. *Process-Centered Requirements Engineering*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[75] H. Kaiya and M. Saeki. Ontology based requirements analysis: Lightweight semantic processing approach. In *Proceedings of the Fifth International Conference on Quality Software, (QSIC'05)*, pages 223–230. IEEE Computer Society, 2005.

[76] C. M. Keet and P. R. Fillottrani. Toward an ontology-driven unifying metamodel for uml class diagrams, eer, and orm2. In *the 32th International Conference, ER*, pages 313–326, 2013.

[77] C. M. Keet and Pablo R. Fillottrani. Structural entities of an ontology-driven unifying metamodel for uml, eer, and orm2. In *3rd International Conference on Model and Data Engineering (MEDI), LNCS, Springer*, pages 188–199, 2013.

[78] S. Khouri, L. Bellatreche, I. Boukhari, and S. Bouarar. More investment in conceptual designers: Think about it! In *15th International Conference on Computational Science and Engineering (CSE), IEEE*, pages 88–93, 2012.

[79] S. Khouri, I. Boukhari, L. Bellatreche, E. Sardet, S. Jean, and M. Baron. Ontology-based structured web data warehouses for sustainable interoperability: requirement modeling, design methodology and tool. *Comput. Ind.*, 63(8):799–812, 2012.

[80] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, and W. Thornwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses with CD Rom*. John Wiley & Sons, Inc., 1st edition, 1998.

[81] J. Sven Körner and B. Torben. Natural language specification improvement with ontologies. *International Journal of Semantic Computing*, 3:445–470, 2009.

[82] W. Labio, Y. Zhuge, J. L. Wiener, H. Gupta, H. Garcia-Molina, and J. Widom. The whips prototype for data warehouse creation and maintenance. In *(SIGMOD Conference)*, pages 557–559, 1997.

[83] R. Laleau, F. Semmak, A Matoussi, D. Petit, A. Hammad, and B.Tatibouet. A first attempt to combine sysml requirements diagrams and b. *Innovations in Systems and Software Engineering (ISSE)*, 6(1-2):47–54, 2010.

[84] A. Lamsweerde. Conceptual modeling: Foundations and applications. chapter Reasoning About Alternative Requirements Options, pages 380–397. Springer-Verlag, 2009.

[85] S. Won Lee and R.A. Gandhi. Ontology-based active requirements engineering framework. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference, (APSEC'05)*, pages 481–490. IEEE Computer Society, 2005.

[86] O. López, M. A. Laguna, and F. J. G.Peñalvo. A metamodel for requirements reuse. In *VII Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*, pages 427–428, 2002.

[87] M. Lubars, C. Potts, and C. Richter. A review of the state of the practice in requirements modeling. In *Proceedings of IEEE International Symposium on Requirements Engineering (RE'93)*, pages 2–14, 1993.

[88] E. m. Clarke and J. M. Wing. Formal methods: State of the art and future directions. *ACM computer surveys*, 28(4):626–643, 1996.

[89] C. Maier, D. Dash, I. Alagiannis, A. Ailamaki, and T. Heinis. Parinda: an interactive physical designer for postgresql. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT'10)*, pages 701–704, 2010.

[90] K. L. Mcgraw and K. Harbison. *User-centered requirements: the scenario-based engineering process*. 1997.

[91] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. Observer: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *Proceedings of the First IFCIS International Conference Cooperative Information Systems, 1996.*, pages 14–25, 1996.

[92] I. Mirbel and S. Villata. Enhancing goal-based requirements consistency: An argumentation-based approach. In *13th International Workshop in Computational Logic in Multi-Agent Systems (CLIMA)* ., pages 110–127, 2012.

[93] J. Mylopoulos. Conceptual modeling in the time of the revolution: Part ii. In *Proceedings of the 28th International Conference on Conceptual Modeling,(ER'09)*, pages 25–25. Springer-Verlag, 2009.

[94] E. Navarro, J.A Mocholi, P. Letelier, and I. Ramos. A metamodeling approach for requirements specification.pdf. *Journal of Computer Information Systems (JCIS)*, 46:67–77, 2006.

[95] B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering, (ICSE'00)*, pages 35–46. ACM, 2000.

[96] F. Oliveira, K. Nagaraja, R. Bachwani, R. Bianchini, R. Martin, and T.D. Nguyen. Understanding and validating database system administration. In *Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference (ATEC'06)*, pages 19–19. USENIX Association, 2006.

[97] E. J. O'Neil, P. E. O'Neil, and K. Wu. Bitmap index design choices and their performance implications. In *Proceeedings of the 11th Internationa Conference In Database Engineering and Applications Symposium, (IDEAS'07)*, pages 72–84, 2007.

[98] G. Pierra. Context representation in domain ontologies and its use for semantic integration of data. pages 174–211. Springer-Verlag, Berlin, Heidelberg, 2008.

[99] G. Pierre. Merise: Modélisation de systèmes d'information. 2005.

[100] W. N. Robinson. Integrating multiple specifications using domain goals. *Sigsoft Software Engineering Notes*, 14(3):219–226, 1989.

[101] A. Rochfeld and H. Tardieu. Merise: An information system design and development methodology. *Information & Management*, 6:143–159, 1983.

[102] C. Rolland. Requirements engineering for cots based systems. *Information and Software Technology (IST)*, 41(14):985–990, 1999.

[103] C. Rolland. Ingènierie des besoins : L'approche l'ecritoire. *Journal Techniques de l'Ingènieur*, pages 1–45, 2003.

[104] C. Rolland, G. Grosz, and R. Kla. Experience with goal-scenario coupling in requirements engineering. In *Proceedings. IEEE International Symposium on Requirements Engineering, (RE 1999).*, pages 74–81, 1999.

[105] C. Rolland, C. Souveyet, and C.B. Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, 1998.

[106] O. Romero, A. Simitsis, and A. Abelló. Gem: Requirement-driven generation of etl and multidimensional conceptual designs. In *Proceedings of the 13th International Conference on Data Warehousing and Knowledge Discovery, (DaWaK 2011)*, pages 80–95, 2011.

[107] D.T. Ross and K.E. Schoman. Structured analysis for requirements definition. *IEEE Transactions on Software Engineering*, 3(1):6–15, 1977.

[108] D. Di Ruscio, R. Eramo, and A. Pierantonio. Model transformations. In *International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM): Formal Methods for Model-Driven Engineering*, pages 91–136, 2012.

[109] M.S. Russell. Assessing the impact of requirements volatility on the se process using bayesian. In *Proceedings of the IInternational Symposium (INCOSE), Toulouse*, 2004.

[110] M. Saeki, S. Hayashi, and H. Kaiya. A tool for attributed goal-oriented requirements analysis. In *24th IEEE/ACM International Conference on Automated Software Engineering*, pages 674–676, 2009.

[111] K. Selma and L. Bellatreche. Dwobs: Data warehouse design from ontology-based sources. In *16th International Conference on Database Systems for Advanced Applications (DASFAA' 2011)*, pages 438–441, 2011.

[112] K. Siegemund, E. J. Thomas, Z. Yuting, J. Pan, and U. Assmann. Towards ontology-driven requirements engineering. In *In 7th International Workshop on Semantic Web Enabled Software Engineering.*, 2011.

[113] J. M. Spivey. *Understanding Z: a specification language and its formal semantics*. Cambridge University Press, 1988.

[114] S. Staab, M. Erdmann, A. Maedche, and s. Decker. Knowledge media in healthcare. chapter An extensible approach for modeling ontologies in RDF(S), pages 234–253. IGI Publishing, 2002.

[115] V. Sugumaran and V. C. Storey. The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Trans. Database Syst.*, pages 1064–1094, 2006.

[116] H. V. Teguiak. *Construction d'ontologies à partir de textes : une approche basée sur les transformations de modéles*. PhD thesis, 2012.

[117] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18(2):197–222, 1986.

[118] A. Tort and A. Olivé. An approach to testing conceptual schemas. *Data Knowledge Engineering*, 69(6):598–618, 2010.

[119] Axel V. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, (RE'01)*, pages 249–263. IEEE Computer Society, 2001.

[120] Axel van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *Proceedings of the 22nd international conference on Software engineering ,(ICSE'00)*, pages 5–19. ACM, 2000.

[121] C. Vicente-Chicote, B. Moros, and A. Toval. Remm-studio: an integrated modeldriven environment for requirements specification, validation and formatting. *Journal of Object Technology*, 6(9):437–454, 2007.

[122] P.R.S. Visser, D.M. Jones, M. Beer, T.J.Bench-Capon, B. Diaz, and M.J. Shave. Resolving ontological heterogeneity in the kraft project. In *Database and Expert Systems Applications (DEXA)*, 1999.

[123] V.R.Basili, G. Gianluigi, and H.D. Rombach. The goal question metric approach. computer science technical report series cs-tr-2956. Technical report, 1992.

[124] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information - a survey of existing approaches. In *Proceedings of the International Workshop on Ontologies and Information Sharing*, pages 108–117, 2001.

[125] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[126] K.E. Wiegers. *More About Software Requirements.* Microsoft Press, USA, 2006.

[127] R. Wieringa, E. Dubois, and H. S. Huyts. Integrating semi-formal and formal requirements. In *Proceedings of the 9th International ConferenceAdvanced Information Systems Engineering, (CAiSE'97)*, pages 19–32, 1997.

[128] D. N. Xuan, L. Bellatreche, and G. Pierra. Ontodawa, un système d'intégration à base ontologique de sources de données autonomes et évolutives. *Ingénierie des Systèmes d'Information (ISI)*, 13(2):97–125, 2008.

[129] E. Siu-Kwong. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1996.

[130] E. Siu-Kwong. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, (RE'97)*, page 226. IEEE Computer Society, 1997.

[131] K. Yue. What does it mean to say that a specification is complete? In *Fourth Internatio-*

nal Workshop on Software Specification and Design (IWSSD-4)., 1987.

[132] D.C. Zilio, J. Rao, S. Lightstone, G. Lohman, A. Storm, C. Garcia-Arellano, and S. Fadden. Db2 design advisor: Integrated automatic physical database design. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB'04)*, pages 1087–1097. VLDB Endowment, 2004.

Annexe A :

L'annexe suivante présente les règles de transformation des besoins locales au modèle générique des besoins.

1. **Règle 1 : OntoGoal-to-OntoReq**. Cette règle décrit la transformation des besoins (instances ontologiques) exprimés par le modèle de buts en modèle générique de besoins comme le montre la figure suivante.

```
--==================================================================================
--=================== RULE INSTANCE ================================================
--==================================================================================
rule OntoGoal-Instance2OntoReq-Instance {

    from

        s: OntoGoal-Metamodel!Individual

    to
        t : OntoReq-Metamodel!Individual (

            uriRef <-s.uriRef,

            label <- s.label,

            type <- OntoReq-Metamodel!OWLClass.allInstances()->select(c|c.label.name='Goal001')

            ),
    label : OntoReq-Metamodel!PlainLiteral (lexicalForm<-s.name),

    u : OntoReq-Metamodel!URIReference ( fragmentIdentifier <- l, uri <- uri ),

    l : OntoReq-Metamodel!LocalName ( name <- s.name    ),

    uri : OntoReq-Metamodel!UniformResourceIdentifier ( name <- s.name )

}
```

FIGURE 1 – Règle 1 : transformation OntoGoal-to-OntoReq

2. **Règle 2 : OntoUcase-to-OntoReq**. Cette règle décrit la transformation des besoins (instances ontologiques) exprimés par le modèle des cas d'utilisation en modèle générique des besoins comme le montre la figure 2.
3. **Règle 3 : OntoMCT-to-OntoReq**. Cette règle décrit la transformation des besoins (instances ontologiques) exprimés par le modèle des traitements en modèle générique des besoins comme le montre la figure 3.

```
--==============================================OntoUcas2OntoReq==============================
--=============================================================================================
rule OntoUcase-Instance2OntoReq-Instance {

    from

        s: OntoUcase-Metamodel!Individual

    to

        t : OntoReq-Metamodel!Individual (

            uriRef <-s.uriRef,

            label <- s.label,

            type <- OntoReq-Metamodel!OWLClass.allInstances()->select(c|c.label.name='Ucase001')

            ),

    label : OntoReq-Metamodel!PlainLiteral (lexicalForm<-s.name),

    u : OntoReq-Metamodel!URIReference ( fragmentIdentifier <- l, uri <- uri ),

    l : OntoReq-Metamodel!LocalName ( name <- s.name    ),

    uri : OntoReq-Metamodel!UniformResourceIdentifier ( name <- s.name )

}
```

FIGURE 2 – Règle 2 : transformation OntoUcase-to-OntoReq

```
--==============================================OntoMCT2OntoReq===================================
--=============================================================================================
rule OntoMCT-Instance2OntoReq-Instance {

    from

        s: OntoMCT-Metamodel!Individual

    to

        t : OntoReq-Metamodel!Individual (

            uriRef <-s.uriRef,

            label <- s.label,

            type <- OntoReq-Metamodel!OWLClass.allInstances()->select(c|c.label.name='Treatment001')

            ),

    label : OntoReq-Metamodel!PlainLiteral (lexicalForm<-s.name),

    u : OntoReq-Metamodel!URIReference ( fragmentIdentifier <- l, uri <- uri ),

    l : OntoReq-Metamodel!LocalName ( name <- s.name    ),

    uri : OntoReq-Metamodel!UniformResourceIdentifier ( name <- s.name )

}
```

FIGURE 3 – Règle 3 : transformation OntoMCT-to-OntoReq

Annexe B

Les requêtes suivantes sont des requêtes typiques du banc d'essai Star Schema Benchmark [41]. Dans nos études expérimentales, nous avons construit plusieurs ensembles de requêtes à partir de celles-ci, afin d'effectuer les différentes évaluations.

Les trois premières requêtes Q1.1, Q1.2 et Q1.3, ont l'objectif de mesurer l'évolution ou l'augmentation des recettes selon les remises accordées, pour certains produits dans une période donnée.

```
Q1.1
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
        and d_year = 1993
        and lo_discount between1 and 3
        and lo_quantity < 25;

Q1.2
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
        and d_yearmonthnum = 199401
        and lo_discount between4 and 6
        and lo_quantity between 26 and 35;

Q1.3
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
        and d_weeknuminyear = 6
        and d_year = 1994
        and lo_discount between 5 and 7
        and lo_quantity between 26 and 35;
```

Les trois requêtes suivantes Q2.1, Q2.2 et Q2.3, décrivent le deuxième besoin décisionnel. Ce besoin permet d'avoir le revenue classé selon plusieurs critères comme les fournisseurs dans une région donnée, les classes de produits et par année.

```
Q2.1:
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
        and lo_partkey = p_partkey
        and lo_suppkey = s_suppkey
```

41. http://www.cs.umb.edu/~poneil/StarSchemaB.PDF

```
        and p_category = 'MFGR#12'
        and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Q2.2

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
        and lo_partkey = p_partkey
        and lo_suppkey = s_suppkey
        and p_brand1 between 'MFGR#2221' and 'MFGR#2228'
        and s_region = 'ASIA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Q2.3

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
        and lo_partkey = p_partkey
        and lo_suppkey = s_suppkey
        and p_brand1 = 'MFGR#2221'
        and s_region = 'EUROPE'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Les quatre requêtes suivantes Q3.1, Q3.2, Q3.3 et Q3.4, décrivent le troisième besoin décisionnel. Ce besoin mesure les revenues pour une région donnée dans une période de temps, selon la provenance des clients et des fournisseurs.

Q3.1

```
select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from
   customer, lineorder, supplier, date
where lo_custkey = c_custkey
        and lo_suppkey = s_suppkey
        and lo_orderdate = d_datekey
        and c_region = 'ASIA' and s_region = 'ASIA'
        and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc;
```

Q3.2

```
select c_city, s_city, d_year, sum(lo_revenue) as revenue from
   customer, lineorder, supplier, date
where lo_custkey = c_custkey
and lo_suppkey = s_suppkey
       and lo_orderdate = d_datekey
       and c_nation = 'UNITED STATES'
       and s_nation = 'UNITED STATES'
       and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;
```

Q3.3

```
select c_city, s_city, d_year, sum(lo_revenue) as reve-nue from
   customer, lineorder, supplier, date
where lo_custkey = c_custkey
       and lo_suppkey = s_suppkey
       and lo_orderdate = d_datekey
       and (c_city='UNITED KI1' or c_city='UNITED KI5')
       and (s_city='UNITED KI1' or s_city='UNITED KI5')
       and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc;
```

Q 3.4

```
select c_city, s_city, d_year, sum(lo_revenue) as reve-nue from
   customer, lineorder, supplier, date
where lo_custkey = c_custkey
       and lo_suppkey = s_suppkey
       and lo_orderdate = d_datekey
       and (c_city='UNITED KI1' or c_city='UNITED KI5')
       and (s_city='UNITED KI1' or s_city='UNITED KI5')
       and d_yearmonth = 'Dec1997'
group by c_city, s_city, d_year
order by d_year asc, revenue desc;
```

Les trois dernières requêtes Q4.1, Q4.2 et Q4.3, décrivent le quatrième besoin décisionnel. Ce dernier est un besoin prévisionnel qui vise à calculer le profit moyen pour certaines classes de produits dans certaines régions et dans une période donnée.

Q4.1

```
select d_year, c_nation, sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
```

```
        and lo_suppkey = s_suppkey
        and lo_partkey = p_partkey
        and lo_orderdate = d_datekey
        and c_region = 'AMERICA'
        and s_region = 'AMERICA'
        and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation;
```

Q4.2

```
select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost)
    as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
        and lo_suppkey = s_suppkey
        and lo_partkey = p_partkey
        and lo_orderdate = d_datekey
        and c_region = 'AMERICA'
        and s_region = 'AMERICA'
        and (d_year = 1997 or d_year = 1998)
        and (p_mfgr = 'MFGR#1 or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category order by d_year, s_nation,
   p_category;
```

Q4.3

```
select d_year, s_city, p_brand1, sum(lo_revenue - lo_supplycost) as
   profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey
        and lo_suppkey = s_suppkey
        and lo_partkey = p_partkey
        and lo_orderdate = d_datekey
        and c_region = 'AMERICA'
        and s_nation = 'UNITED STATES'
        and (d_year = 1997 or d_year = 1998)
        and p_category = 'MFGR#14'
group by d_year, s_city, p_brand1
order by d_year, s_city, p_brand1;
```

Table des figures

1.1 Différents niveaux d'intégration [50]. 4
1.2 Résolution manuelle de l'hétérogénéité 6
1.3 Résolution semi-automatique de l'hétérogénéité 6
1.4 Les quatre niveaux d'intégration. 9
1.5 Aperçu des contributions de la thèse 10

2.1 Les phases du cycle de vie d'un ED. 21
2.2 Exemple de métamodèlisation. 23
2.3 Processus d'ingénierie des besoins. 24
2.4 Un besoin modélisé par un langage naturel. 27
2.5 Un besoin modélisé par un langage semi-formel (Cas d'utilisation). 27
2.6 Approche GQM *(Goal Question Metric)*. 31
2.7 Exemple d'un modèle orienté buts KAOS [13]. 32
2.8 Exemple d'un modèle orienté buts iStar (i*) [13]. 34
2.9 Diagramme des cas d'utilisation *(Use case)*. 35
2.10 Exemple de modèle conceptuel de traitement. 36
2.11 Méta-modèle des besoins modélisés en UML. [86]. 39
2.12 Etape de production d'un modèle global des besoins [30]. 40

3.1 Exemple d'ontologie exprimée en RDF Schéma. 49
3.2 Exemple d'ontologie exprimée en OWL. 51
3.3 L'éditeur d'ontologies Protégé. 53
3.4 Différentes architectures d'intégration à base ontologique. 55
3.5 Architectures d'intégration sémantique *a posteriori*. 55

3.6 Architectures d'intégration sémantique *a priori*. 57
3.7 Contributions des ontologies dans l'IB. 58

4.1 Hétérogénéité des vocabulaires et des langages de modélisation des besoins. 67
4.2 Exemple d'extraction de l'ontologie locale. 70
4.3 Une partie de l'ontologie *University* proposée par le benchmark LUBM. 71
4.4 Exemple de modèle orienté buts modélisant les besoins du concepteur *Espagnol*. 72
4.5 Exemple de modèle de cas d'utilisation modélisant les besoins du concepteur *Anglais*. . . 74
4.6 Exemple de modèle de traitements modélisant les besoins du concepteur *Français*. . . . 75
4.7 Couplage des modèles des besoins avec le modèle d'ontologie. 76
4.8 Méta-modèle d'ontologie OWL. 77
4.9 Connexion entre le modèle ontologique et le modèle orienté buts. 78
4.10 Instanciation des modèles de l'ontologie et du modèle orientés buts. 79
4.11 Connexion entre le modèle ontologique et le modèle use case. 80
4.12 Connexion entre le modèle ontologique et le modèle MCT. 81
4.13 Intégration centralisée avec langage Pivot. 82
4.14 Le méta-modèle pivot. 84
4.15 Connexion entre le modèle ontologique et le modèle pivot. 85
4.16 Scénario d'intégration des besoins selon un langage pivot. 86
4.17 Principe de transformation. 87

5.1 Méta-modèle générique fusionnant les trois modèles étudiés. 91
5.2 Métamodèle générique des besoins. 92
5.3 Connexion entre le méta-modèle ontologique et le méta-modèle générique des besoins. . 94
5.4 Processus de raisonnement. 96
5.5 Scénario (1) : ship whole. 97
5.6 Scénario (2) : reason as needed. 98
5.7 Temps d'exécution de raisonnement. 102

6.1 Répartition de l'effort de l'administrateur. 108
6.2 La conception physique. 109
6.3 Architecture d'OntoDB. 112
6.4 BDBO de type III étendu par les besoins des utilisateurs. 113
6.5 Un besoin SSB modélisé suivant notre modèle générique des besoins. 115

6.6 Ontologie du benschmark SSB. 117
6.7 Fragmentation horizontale à partir des besoins. 119
6.8 Nouvelle vision de conception des bases de données. 120

7.1 Architecture fonctionnelle de l'outil (OntoReqTool). 124
7.2 Diagramme de cas d'utilisation de notre outil. 125
7.3 Projection des besoins sur l'OG (OntoReq). 126
7.4 Ontologie locale des buts (OntoGoal). 127
7.5 Transformation des besoins en modèle générique. 128
7.6 Raisonnement sur les besoins. 129
7.7 Interface graphique de la phase d'exploitation de l'ED. 130
7.8 Persistance des besoins dans l'ED. 130
7.9 Requêtes SQL générées. 132

1 Règle 1 : transformation OntoGoal-to-OntoReq 149
2 Règle 2 : transformation OntoUcase-to-OntoReq 150
3 Règle 3 : transformation OntoMCT-to-OntoReq 150

Liste des tableaux

2.1 Comparaison des travaux étudiés. 42

4.1 Les concepts communs entre les modèles de besoins. 83

5.1 Nombre de relations vérifiées pendant le raisonnement. 102

6.1 Index générés et taux d'optimisation du coût d'exécution des besoins par rapport aux requêtes 118

6.2 Index générés et taux d'optimisation du coût d'exécution des besoins par rapport aux requêtes 118

Glossaire

ABox : Assertionnal Box
API : Application Programming Interface
ATL : ATLAS Transformation Language
BD : Base de données
BDBO : Base de données à base ontologique
CC : Classe canonique
CNC : Classe non canonique
ED : Entrepôt de données
EMF : EclipseModeling Framework
GQM : Goal Question Metric
IB : Ingénierie des Besoins
IDM : Ingénierie Dirigée par les Modèles
iStar : Intentional STratégic Actor Relationships
KAOS : Ingénierie des Besoins
LISI : Laboratoire d'Informatique Scientifique et Industrielle
LO : Ontologie Locale
MC : Modèle Conceptuel
MDA : Model-Driven Architecture
OC : Ontologie Conceptuelle
OCC : Ontologie Conceptuelle Canonique
OCNC : Ontologie Conceptuelle Non Canonique
OL : Ontologie Linguistique
OMG : Object Management Group
OWL : Web Ontology Language
PLIB : Parts Library - Norme ISO 13584
RDF : Ressource Description Framework
SWRL : Semantic Web Rule Language
TBox : Terminological Box
UML : Unified Modeling Language
URI : Uniform Resource Identifier
URL : Uniform Resource Locator

Intégration et exploitation de besoins en entreprise étendue fondées sur la sémantique.

Présentée par :

Ilyés BOUKHARI

Directeurs de Thèse :

Ladjel BELLATRECHE et **Stéphane JEAN**

Résumé. L'ingénierie des besoins (IB) joue un rôle crucial dans le processus de développement d'un système d'information. Elle a pour but de fournir une spécification des besoins qui soit cohérente, non ambiguë, complète, vérifiable, etc. L'importance de la spécification des besoins augmente dans l'environnement distribué où se situent les systèmes complexes ce qui est en particulier le cas dans le contexte des entreprises étendues. Ces systèmes nécessitent, en effet, la collaboration d'un grand nombre de concepteurs qui peuvent provenir de différents domaines, départements, laboratoires de recherche, etc. En général, ces concepteurs sont libres d'utiliser le vocabulaire et les langages qu'ils préfèrent pour définir les besoins liés aux parties du système qui leur sont affectées. Dans ce contexte, fournir une interface unique, uniforme et transparente à l'ensemble des besoins définis sur un système est devenu nécessaire. Autrement dit, d'un point de vue technique, il devient crucial de proposer une démarche d'intégration des besoins via un schéma global. Dans ce travail, nous proposons une approche ontologique d'intégration de besoins hétérogènes. Cette hétérogénéité concerne à la fois les vocabulaires et les langages de modélisation identifiés lors de l'élicitation des besoins. Contrairement aux travaux existants qui traitent le problème d'intégration des besoins d'une manière isolée sans prendre en compte le système cible (une base/entrepôt de données, une interface homme machine, etc.), nos propositions apportent des solutions pour les phases de cycle de vie de conception du système cible. Pour illustrer cela, nous considérons la phase de conception physique d'un entrepôt de données. Durant cette phase, un ensemble de structures d'optimisation (les index, la fragmentation de données, les vues matérialisées, etc.) est sélectionné. Souvent cette sélection est effectuée à partir d'un ensemble de requêtes. Dans ce travail, nous proposons une approche de sélection dirigée par les besoins. Pour valider notre proposition, une implémentation dans un environnement de Base de Données à Base Ontologique nommé OntoDB est proposée.

Mots-clés : Ingénierie des Besoins, Ontologies, Systèmes d'intégration, Entrepôts de données, Ingénierie Dirigée par les Modèles, BDBO, OntoDB.

Abstract. Requirements engineering (RE) plays a crucial role in the process of developing information systems. It aims at providing a specification of requirements that is consistent, unambiguous, complete, verifiable, etc. With the development of distributed applications (such is the case of global enterprises) the importance of RE increases. Indeed, these applications require the collaboration of a large number of heterogeneous, autonomous and distributed designers. As a consequence, designers are free to use their own vocabularies and languages to express their requirements. In this setting, providing a unique, consistent and transparent interface to all requirements defined upon the system is needful. From a technical point of view, it becomes crucial to propose an approach for requirements integration through a global schema. In this work, we suggest a semantic approach for integrating heterogeneous requirements that takes into account the heterogeneity of both vocabularies and modeling languages identified in the requirements' elicitation step in the context of data warehouse design. Unlike existing works which addressed requirements as an isolated integration issue without taking into account the phases of designing data warehouse, our proposals provide solutions to the main important life cycle phases for the data warehouse. To do so, we have mainly focused on the physical data of designing relational warehouse in which a set of optimized structures (indexes, data fragmentation, materialized views, etc.) are selected based on the integrated requirements. To prove the validity of our proposal, an implementation upon the environment of Ontological Database named OntoDB is proposed.

Keywords: Requirements Engineering, Ontologies, Integration Systems, Data Warehousing, Model Driven Engineering, DBOB, OntoDB.