République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri de Constantine
Département d'Informatique

N° d'ordre : 164/Mag/2007

Série : 003/Inf/2007

MÉMOIRE

Présenté en vue de l'obtention du diplôme de
Magister en informatique
Option : Génie logiciel & intelligence artificielle
par

HABIB-ELLAH GUERGOUR

Construction d'une ontologie d'application dans le cadre de l'EAI

Soutenu publiquement le 16/06/2007 devant le jury composé de :

Président:	Mr. Mahmoud BOUFAÏDA	Professeur	Univ. Mentouri de Constantine
Rapporteur:	Mme. Zizette BOUFAÏDA	Professeur	Univ. Mentouri de Constantine
Examinateur:	Mr. Nacereddine ZAROUR	Maître de conférences	Univ. Mentouri de Constantine
Examinateur:	Mr. Mohamed-khireddine KHOLLADI	Maître de conférences	Univ. Mentouri de Constantine
Invitée :	Melle. Razika DRIOUCHE	Chargée de cours	Univ. Mentouri de Constantine

Préparé au sein du SIBC, Laboratoire LIRE
Equipe Systèmes d'Information & Bases de Connaissances, Laboratoire d'Informatique REpartie

CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION DANS LE CADRE DE L'EAI

BUILDING APPLICATION ONTOLOGY IN THE EAI CONTEXT

Présenté par : HABIB-ELLAH GUERGOUR
Encadré par : Professeur Mme. ZIZETTE BOUFAÏDA

Equipe SIBC
Laboratoire LIRE

Université Mentouri de Constantine
www.umc-dz.org

REMERCIEMENTS

C'est avec l'aide de DIEU tout puissant que ce modeste mémoire a pu être réalisé, DIEU qui nous a donné fois, raison et lucidité.

Je tiens à remercier mon encadreur, Mme. Zizette BOUFAÏDA, pour m'avoir donné l'opportunité de travailler sur ce projet, pour son grand soutien scientifique et moral, pour les conseils, les suggestions et les encouragements qu'elle m'a apportés pendant l'élaboration de ce mémoire.

Ensuite, j'adresse toute ma profonde gratitude à Melle Razika DRIOUCHE qui m'a beaucoup aidé dans le développement de ce projet ainsi que dans la rédaction des articles. Ses conseils, ses suggestions, ses commentaires et ses critiques ont contribué à l'aboutissement de ce travail.

Plus largement, je souhaite saluer toute l'équipe de SIBC et surtout le professeur M. Mahmoud BOUFAÏDA pour la grande convivialité des réunions et pour m'avoir donné à vivre ces débats passionnés. Je le remercie aussi d'avoir accepté la présidence de ce jury.

Je remercie vivement Monsieur Nacereddine ZAROUR et Monsieur Mohamed-Khireddine KHOLLADI qui m'ont fait, vraiment, l'honneur de prendre part au membre de jury et d'avoir accepté la charge d'examinateur.

Ce mémoire, notamment dans sa phase terminale, doit beaucoup à ma chère mère, pour sa patience et son soutien tant moral que spirituel pendant les moments difficiles. Merci d'avoir crû en moi, d'avoir toujours été là, pendant toutes ces années et parce que, sans lui, sans ses conseils et sans son amour, rien de tout ceci n'aurait pu arriver.

Je ne saurais oublier de remercier mon grand père Ami-El-Hanni, ma sœur Amina, mes frères Walid et Djalil pour leur soutien inconditionnel. Il m'a été simplement très précieux. Une pensée toute particulière à la mémoire de mon père et ma grand-mère qui nous ont malheureusement quittés.

Je tiens également à remercier tous ceux et celles qui de près ou de loin ont contribué de quelque manière que ce soit à l'aboutissement de ce mémoire. Je pense particulièrement aux familles GUERGOUR, DJEGHABA, DJERADI, BATOUCHE, MANCHAR Ma profonde reconnaissance va aussi à l'endroit de tous ceux et celles que j'ai sûrement eu l'indélicatesse d'oublier.

Je tiens énormément à remercier mes amis et mes collègues pour leur soutien pendant toute la période de ce mémoire. Je tiens à partager le plaisir de présenter ce mémoire avec eux.

Habib-ELLAH

A la mémoire de mon père
A ma chère mère
A tous ceux qui me sont chers…

RÉSUMÉ

L'aspect sémantique porte sur l'étude du sens des entités et de leurs relations indépendamment de la façon de les représenter. L'hétérogénéité des modèles de représentations, des langages de programmation, des systèmes d'exploitation et des mécanismes d'échange de données, implique généralement une mauvaise interprétation. De ce fait, le problème d'interopérabilité s'accentue entre les systèmes et rend le processus d'intégration d'applications largement confus.

La notion d'ontologie est une solution candidate. Elle sert à représenter de façon générique et réutilisable la sémantique d'un domaine. A l'origine, elle fournit pour un domaine particulier une compréhension commune pour partager les connaissances et les données. Actuellement, les ontologies sont de plus en plus utilisées pour interopérabilité des applications hétérogènes.

L'objectif de ce mémoire consiste en la construction d'une ontologie d'application afin de pallier au problème de l'hétérogénéité entre les applications pour faciliter ultérieurement leur intégration. La construction de cette ontologie doit se baser sur un modèle cernant toutes les propriétés structurelles et comportementales d'une application, ainsi que des propriétés sur le domaine de son activité. Pour cela nous nous attachons, dans ce mémoire, à proposer un processus de développement d'ontologies d'applications, partant de connaissances brutes et arrivant à une ontologie d'application opérationnelle représentée par le langage OWL. L'ontologie d'application sera ensuite construite en suivant les étapes du processus proposé.

Mots-clés: EAI, Ontologie d'application, hétérogénéité sémantique, Scénarios de communication.

TABLES DES MATIÈRES

RÉSUMÉ 1

INTRODUCTION GÉNÉRALE

1 CONTEXTE GENERAL 2

2 PROBLEMATIQUE ABORDEE 4

3 NOTRE DEMARCHE 5

4 ORGANISATION DU MEMOIRE 6

CHAPITRE I: RELIER LES APPLICATIONS DE L'ENTREPRISE (EAI)

1 INTRODUCTION 9

1.1 ORIGINE DE L'EAI 9
1.2 PRINCIPE DE L'EAI 10

2 APERÇU SUR L'INTEGRATION 11

2.1 INTEGRATION DE DONNEES 11
2.2 INTEGRATION DES PROCESSUS 11
2.3 INTEGRATION DES INTERFACES 12
2.4 INTEGRATION DES FONCTIONS 12

3 INTEGRATION D'APPLICATIONS 12

3.1 DEFINITIONS ET CHALLENGES 13
3.2 APPROCHES D'INTEGRATION D'APPLICATIONS 14
3.2.1 *Intégration d'applications par les données* 14
3.2.2 *Intégration d'applications par les fonctions (traitements)* 14
3.2.3 *Intégration d'applications par les interfaces (présentations)* 15
3.2.4 *Intégration d'applications par les processus* 15

4 TYPOLOGIES DES APPLICATIONS INTEGREES 16

4.1 LES APPLICATIONS BATCH 16
4.2 LES APPLICATIONS TRANSACTIONNELLES 17
4.3 LES APPLICATIONS CLIENT-SERVEUR 17
4.4 LES APPLICATIONS BASEES SUR UN ECHANGE DES MESSAGES 17
4.5 LES PROGICIELS 18

5 TOPOLOGIES DES ARCHITECTURES D'INTEGRATION 18

5.1 TOPOLOGIE POINT A POINT 18
5.2 TOPOLOGIE PAR BUS 19
5.3 TOPOLOGIE PAR HUB-AND-SPOKE 19
5.4 TOPOLOGIE NETWORK CENTRIC 20

6 MODELES D'INTEGRATION D'APPLICATIONS 20

7 TECHNOLOGIES D'INTEGRATION D'APPLICATIONS 21

8 CONCLUSION 23

CHAPITRE II : L'INGENIERIE ONTOLOGIQUE

1 INTRODUCTION 25

2 NOTION D'ONTOLOGIE 25

2.1 DEFINITIONS 25
2.2 QUE REPRESENTE-T-ON DANS UNE ONTOLOGIE ? 27
2.3 DIFFERENTES SORTES D'ONTOLOGIES 29
2.3.1 Objet de conceptualisation 29
2.3.2 Niveau de formalisme de représentation 30
2.4 LES ONTOLOGIES ET LES SYSTEMES A BASES DE CONNAISSANCES 30
2.5 LES ONTOLOGIES ET LE WEB SEMANTIQUE 30

3 LES ONTOLOGIES : DIFFERENTS BESOINS 31

3.1 COMMUNICATION 31
3.2 INTEROPERABILITE 32
3.3 INGENIERIE DES SYSTEMES 32

4 UN SQUELETTE DE METHODOLOGIE POUR CONSTRUIRE DES ONTOLOGIES 32

4.1 EVALUATION DES BESOINS 33
4.2 CONCEPTUALISATION 33
4.3 ONTOLOGISATION 34
4.4 OPERATIONNALISATION 34

5 QUELQUES METHODOLOGIES DE CONSTRUCTION D'ONTOLOGIES 34

5.1 TOVE 34
5.2 ENTERPRISE 35
5.3 METHONTOLOGY 35
5.3.1 Spécification 36
5.3.2 Conceptualisation 36
5.3.3 Implémentation 36
5.3.4 Maintenance 36

6 FORMALISMES DE REPRESENTATION 37

6.1 FRAMES 37
6.2 GRAPHES CONCEPTUELS 37
6.3 LOGIQUES DE DESCRIPTIONS 38
6.3.1 Les constructeurs des LDs 38
6.3.2 Mécanisme de raisonnement 38

7 OUTILS DE DEVELOPPEMENT D'ONTOLOGIES 39

7.1 LANGAGES DE SPECIFICATION D'ONTOLOGIES 39
7.1.1 RDF 40
7.1.2 RDF(S) 40
7.1.3 DAML-OIL 41
7.1.4 OWL 41
7.2 EDITEURS D'ONTOLOGIES 42
7.2.1 Oiled 42
7.2.2 OntoEdit 42
7.2.3 Ontolingua 42
7.2.4 ONTOSAURUS 42
7.2.5 PROTÉGÉ-OWL 43
7.3 ET BIEN D'AUTRES… ! 43
7.3.1 Racer 43
7.3.2 Jena 44

8 CONCLUSION 45

CHAPITRE III: CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION DANS LE CADRE DE L'EAI

1 INTRODUCTION 46

2 ARCHITECTURE BASEE SUR LES ONTOLOGIES POUR L'INTEGRATION D'APPLICATIONS DE L'ENTREPRISE 46

2.1 NIVEAUX D'INTEGRATION 47
2.2 PROCESSUS D'INTEGRATION D'APPLICATIONS DE L'ENTREPRISE 48

3 VERS UN PROCESSUS DE CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION 48

3.1 SPECIFICATION 49
3.2 CONCEPTUALISATION : 49
3.3 FORMALISATION 49
3.3.1 Partie terminologique TBOX 50
3.3.2 Partie assertionnelle ABOX 51
3.4 IMPLEMENTATION 51
3.5 TESTS ET EVOLUTION 51

4 CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION DANS LE CADRE DE L'EAI 52

4.1 SPECIFICATION 52
4.2 CONCEPTUALISATION 54
4.2.1 Construction de glossaire de termes 54
4.2.2 Construction du diagramme de relations binaires et de classification de concepts 55
4.2.3 Dictionnaire de concepts 57
4.2.4 Tableaux de relations binaires 62
4.2.5 Tableaux des attributs 64
4.2.6 Tableau des axiomes logiques 65
4.2.7 Tableau des instances 66
4.3 FORMALISATION 67
4.3.1 Construction de TBOX 67
4.3.2 Construction de ABOX 70
4.4 IMPLEMENTATION 71
4.4.1 Présentation de l'éditeur PROTEGE OWL 71
4.4.2 Création des classes et la hiérarchie des classes 72
4.4.3 Création des propriétés 74
4.4.3.1 Création des attributs (dataTypeProperty) 75
4.4.3.2 Création des relations (ObjectProperty) 76
4.4.3.3 Restrictions sur les propriétés 76
4.4.4 Création des instances 78
4.4.5 Génération du code 79
4.5 TEST & EVOLUTION DE L'ONTOLOGIE 79

5 RETABLIR LA COMMUNICATION ENTRE LES APPLICATIONS DE L'ENTREPRISE. 81

5.1 POURQUOI RETABLIR ? 82
5.2 SCENARIOS DE COMMUNICATION ENTRE LES APPLICATIONS INTERNES DE L'ENTREPRISE 82
5.3 SCENARIOS DE COMMUNICATION ENTRE LE CLIENT ET LES APPLICATIONS DE L'ENTREPRISE 85

6 CONCLUSION 87

CONCLUSION & PERSPECTIVES

1 BILAN 89

1.1 DEVELOPPEMENT DES SCENARIOS DE COMMUNICATION ENTRE LES APPLICATIONS DE L'ENTREPRISE 90
1.2 PROPOSITION D'UN PROCESSUS DE DEVELOPPEMENT DES ONTOLOGIES D'APPLICATIONS DANS LE CADRE DE L'EAI ... 90
1.3 CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION DANS LE CADRE DE L'EAI 90

2 EVOLUTION & PERSPECTIVES 91

2.1 EVOLUTION DU SCENARIO DE COMMUNICATION PROPOSE 91
2.2 EVOLUTION DU PROCESSUS DE DEVELOPPEMENT D'ONTOLOGIES PROPOSE 91

2.3 EVOLUTION DE L'ONTOLOGIE D'APPLICATION 91

REFERENCES 92
ANNEXE 96

INTRODUCTION GÉNÉRALE

1 CONTEXTE GENERAL

Un système d'information d'entreprise est par nature hétérogène : il est constitué de nombreuses applications bâties sur des technologies différentes avec des spécificités propres aux contraintes du moment, et qui, de surcroît, ont évolué dans le temps. Réussir l'intégration de ces applications anciennes et récentes pour former un tout cohérent et opérationnel est un besoin auquel l'EAI (Entreprise Application Integration), en tant que solution logicielle autour de laquelle vont s'articuler les différents échanges, tente de répondre.

L'EAI, traduit en français par "Intégration des Applications d'Entreprise", permet de faire communiquer tout type d'applications, existantes et futures, en s'échangeant des messages entre elles et en partageant des informations de la manière la plus efficace et la plus simple possible. Il ne s'agit plus, dès lors, de développer une nouvelle solution d'interfaçage mais de fournir un cadre d'intégration souple et robuste. Le fait que l'entreprise puisse faire évoluer en douceur son SI, en s'appuyant au maximum sur l'existant, est une qualité intrinsèque de l'EAI. Le système d'information a ainsi la capacité d'absorber les changements technologiques toujours plus fréquents.

Cependant, la mise à disposition d'information et l'échange de données entre les applications n'a aucune sémantique définie vu que ces échanges sont basés au mieux sur des mappings ou bien sur des transformations syntaxiques. De plus, l'hétérogénéité des modèles de représentations, des langages de programmation, des systèmes d'exploitation et des mécanismes

d'échange des données, implique généralement une mauvaise interprétation. De ce fait, l'aspect sémantique va porter sur l'étude du sens des entités et de leur relation indépendamment de la façon de les représenter afin d'aboutir à une interopérabilité sémantique entre les applications de l'entreprise.

Depuis quelques années, l'usage des ontologies est de plus en plus grandissant pour résoudre le problème de l'interopérabilité sémantique. Autre que cet usage, le champ d'application des ontologies ne cesse de s'élargir et couvre aussi les systèmes d'aide à la décision, les systèmes de résolution de problèmes ou les systèmes de gestion de connaissances. Un des plus grands projets basés sur l'utilisation d'ontologies est le Web sémantique qui consiste à ajouter au Web actuel une véritable couche de connaissances permettant, dans un premier temps, des recherches d'information au niveau sémantique et non plus syntaxique. A terme, il est prévu que des applications Internet pourront mener des raisonnements utilisant les connaissances stockées sur le Web [29].

Les ontologies sont à l'heure actuelle le cœur des travaux menés en Ingénierie des Connaissances (IC). Visant à établir des représentations à travers les quelles les machines puissent manipuler la sémantique des informations, la construction des ontologies demande à la fois une étude des connaissances humaines et la définition de langage de représentation, ainsi que la réalisation des systèmes pour les manipuler. Il faut cependant noter que, depuis la naissance de l'intelligence artificielle, plusieurs formalismes de représentation de connaissances ont été développés, permettant de formaliser les connaissances d'un domaine, puis de mettre en œuvre des raisonnements sur ces représentations. Parmi ces formalismes développés au niveau conceptuel pour la modélisation d'ontologie, trois grands modèles sont distingués: les graphes conceptuels, les langages de frames et les logiques de descriptions. Ces formalismes permettent de représenter les concepts sous-jacents à un domaine de connaissance, les relations qui les lient, et la sémantique de ces relations.

Au fur et à mesure des expérimentations, des méthodes de construction et des outils de développement adéquats sont apparues. Emergeant des pratiques artisanales initiales, une véritable ingénierie se constitue autour des ontologies, ayant pour but leur construction mais plus largement leur gestion tout au long d'un cycle de vie. Les ontologies apparaissent ainsi comme des composants logiciels s'insérant dans les systèmes d'information et leur apportant une dimension sémantique qui leur faisait défaut jusqu'ici.

2 PROBLEMATIQUE ABORDEE

L'aspect sémantique porte sur l'étude du sens des entités et de leurs relations indépendamment de la façon de les représenter. L'hétérogénéité des modèles de représentations, des langages de programmation, des systèmes d'exploitation et des mécanismes d'échange des données, implique généralement une mauvaise interprétation. Par conséquent, le problème d'interopérabilité s'accentue entre les systèmes et rend le processus d'intégration d'applications largement confus.

La notion d'ontologie est une solution candidate. Elle sert à représenter de façon générique et réutilisable la sémantique d'un domaine. A l'origine, elle fournit pour un domaine particulier une compréhension commune pour partager les connaissances et les données. Actuellement, les ontologies sont de plus en plus utilisées pour interopérabilité des applications hétérogènes.

Dans ce contexte, une architecture basée sur les ontologies est proposée [60], cette architecture comporte deux niveaux d'intégration : un niveau collaboratif et un niveau applicatif. Le niveau collaboratif, représente une intégration externe B2B, est doté d'une ontologie de processus métiers qui sert à supporter la collaboration entre les partners. Par contre, le niveau applicatif, montre une intégration interne A2A, consiste à masquer l'hétérogénéité entre les applications internes de l'entreprise en développant pour chacune une ontologie d'application, ce niveau est muni aussi d'un composant

mapper, basé sur le processus framework MAFRA (MApping FRAmework) et le concept de semantic bridge [67].

L'objectif de ce mémoire consiste donc en la construction d'une ontologie d'application afin de gommer et celer l'hétérogénéité entre les applications pour faciliter ultérieurement leur intégration. La construction de cette ontologie doit se baser sur un modèle cernant toutes les propriétés structurelles et comportementales d'une application, ainsi que des propriétés sur le domaine de son activité.

3 NOTRE DEMARCHE

Notre contribution dans ce contexte consiste en la construction d'une ontologie d'application dans le cadre de l'EAI. Le développement de cette ontologie doit suivre un processus de construction d'ontologies comptant un ensemble de phases spécifiées de façon très détaillée afin de cerner l'étendue et d'aboutir à une ontologie typique. Pour cela, nous nous attachons dans ce mémoire à:

1. Proposer un processus de développement d'ontologies d'applications partant de connaissances brutes et arrivant à une ontologie d'application opérationnelle représentée par le langage OWL. Ce processus est composé de cinq phases à savoir: spécification des besoins, conceptualisation, formalisation, implémentation, test & évolution de l'ontologie. Ces étapes sont suivies afin d'expliciter et de guider le développement de l'ontologie. La réalisation de ces différentes étapes, s'appuie sur les résultats des travaux réalisés en ingénierie des connaissances et en Intelligence Artificielle (IA). Ces travaux mettent l'accent sur l'aspect de représentation formelle et de raisonnement, et utilisent en particulier le formalisme de la logique de descriptions qui possède une sémantique claire et procure des services d'inférences et des mécanismes de raisonnements puissants.

2. Construire l'ontologie d'application en suivant les étapes du processus proposé. Nous commençons par la phase de spécification qui consiste à établir un document de spécification des besoins servant à décrire l'ontologie à construire à travers les cinq aspects, à savoir, le domaine de connaissance, l'objectif, les utilisateurs, les sources d'information et la portée de l'ontologie. Ensuite, nous organisons et structurons les connaissances acquises en utilisant des représentations intermédiaires semi formelles qui sont faciles à comprendre et indépendantes de tout langage d'implémentation. Après, nous utiliserons le formalisme des logiques de description pour représenter l'ontologie d'application dans un langage formel et finalement nous implémenterons et testerons l'ontologie en utilisant les outils protégé OWL et le raisonner RACER.

Nous suggérerons, en outre, et de manière superficielle, des scénarios de communication entre les applications de l'entreprise afin d'exploiter les ontologies construites et d'avoir un échange plus compréhensible entre les applications. Pour cela, nous leur associons des modules logiciels pour garantir la cohérence et la flexibilité du système. Le développement de ces scénarios va nous permettre de donner un éclaircissement sur le comportement des applications de l'entreprise vis-à-vis les ontologies construites, en d'autres termes, en montrant comment les applications vont communiquer entre elles, à travers les ontologies construites.

4 ORGANISATION DU MEMOIRE

Pour présenter le travail, nous avons élaboré le plan de lecture suivant:

Le premier chapitre est dédié à la présentation du domaine de l'intégration d'applications d'entreprise. Il commence par la définition de la notion d'"intégration". Ensuite, il présente les différentes sortes d'intégrations, les principales approches utilisées pour intégrer les applications, leurs typologies et les

topologies de leurs architectures. Par la suite, il présente en bref les différentes technologies d'intégration à savoir les middlewares orientés données, les middlewares orientés services, etc.

Le deuxième chapitre est consacré à la présentation des ontologies. Nous commençons par la définition de la notion d'ontologie en ingénierie des connaissances. Nous présentons ensuite les principaux formalismes de représentation de connaissances, à savoir, les frames, les graphes conceptuels et les logiques de descriptions. Nous découvrirons après les méthodologies les plus représentatives de leur construction et quelques domaines de leur utilisation. A la fin, nous présenterons les outils nécessaires de leur développement, à savoir, les langages de représentation, les outils d'édition et d'interrogation, etc.

Le troisième chapitre concerne la présentation en détail de notre travail, dans lequel nous avons:

1. proposé un processus de développement d'ontologies afin de faciliter ultérieurement la construction de l'ontologie d'application.
2. utilisé le processus proposé pour arriver à la construction de l'ontologie d'application.
3. proposé un scénario de communication afin de montrer l'acheminement des requêtes entre les applications de l'entreprise.

Le mémoire s'achève par une conclusion générale récapitulant le contexte de recherche de notre étude, la démarche suivie, nos contributions et énonce un ensemble de perspectives.

CHAPITRE I

EAI : RELIER LES APPLICATIONS DE L'ENTREPRISE

1 Introduction

Avec l'évolution d'Internet, les entreprises se regroupent, afin de constituer des holdings sans qu'elles changent leur situation géographique, ni leur infrastructure, ni leur SI (Systèmes d'Information). Elles font collaborer les services concernés et surtout donnent accès à leurs informations. Donc, nous avons un SI multi-sources qui sera forcement hétérogène. Pour que cet accès soit rapide et transparent, on doit recourir à l'intégration, qui sert à rendre l'entreprise plus efficace, plus rentable et plus accessible aux tiers. En d'autres termes, il s'agit d'en faire un partenaire commercial plus efficace [1]. La mondialisation n'a fait qu'accentuer les échanges interentreprises par le biais d'Internet. Depuis quelques années, des méthodes et des outils d'interfaçage ont été créés afin d'échanger, de transformer et de rediriger les données. A présent, l' EAI (Enterprise Application Integration) va plus loin, en proposant une architecture qui constitue un véritable cadre d'intégration des applications existantes et celles à venir. L'intégration pose certains problèmes, tel que, comment traiter l'hétérogénéité des applications et fournir une vue consolidée des données tout en offrant l'accès aux logiciels de l'intérieur ou de l'extérieur de l'entreprise ? De plus, comment peut-on enchaîner automatiquement les tâches liées à une fonction business ?

L'EAI est devenue une nécessité incontournable pour faire face aux exigences sans cesse évolutives du marché. Les fortes contraintes qui pèsent de nos jours sur les entreprises sont souvent directement répercutées sur leurs SI, à qui on demande d'être sans cesse plus flexibles et plus réactifs. Aujourd'hui, plus de 40 % des budgets de développement en informatique sont liés à l'intégration de données dans les systèmes d'information. Il s'agit donc d'une problématique stratégique pour les entreprises L'intégration entre des bases de données et/ou des applications hétérogènes, avec un traitement batch (en différé) ou en temps réel implique, souvent, la mise en oeuvre de projets coûteux et complexes.

Ce rapport vise à apporter un peu de lumière sur la problématique complexe du processus d'intégration. Elle présente l'état de l'art des applications dans les PME et les solutions à envisager pour mieux gérer le processus d'intégration entre les applications internes (A2A ou "application vers application") et mieux les interopérer avec les applications externes (B2B, B2A, etc.).

1.1 Origine de l'EAI

La perpétuelle nécessité de couvrir de nouveaux besoins dans des délais souvent brefs incite à travailler au coup par coup. Les communications entre applications sont donc conçues pour répondre au strict nécessaire dans le contexte d'un projet donné. Au fil du temps, les liens ainsi tissés entre les applications, contribuent à la constitution d'un « plat de spaghettis », comme l'illustre le schéma ci-dessous. (Fig. 1)

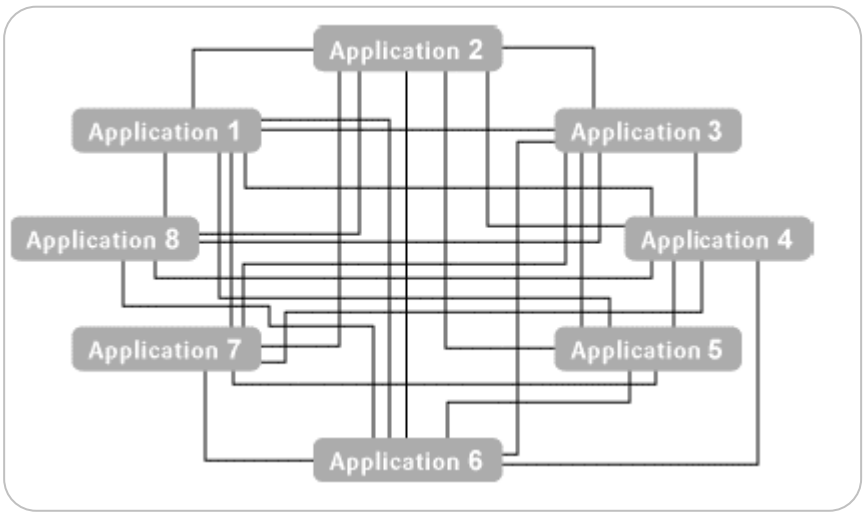


Fig. 1:Intégration en mode point à point.

Ce modèle point à point provoque une explosion combinatoire du nombre d'interfaces à gérer : pour **n** applications, il peut être nécessaire de gérer **n*(n-1)/2** interfaces. Les conséquences de chaque évolution concernent un nombre croissant d'applications, de part les effets de bord qu'une modification peut entraîner sur les liens qui se sont tissés de manière plus ou moins formelle au fil du temps [2].

1.2 Principe de l'EAI

L'objet de l'EAI est de faire en sorte que les applications existantes et futures puissent communiquer entre elles et partager des informations de la manière la plus efficace et la plus simple possible. Pour y parvenir, l'EAI, comme illustré ci-dessous, repose généralement sur l'utilisation d'un bus de communication en lieu et place de communications directes d'application à application. (Voir Fig. 2)

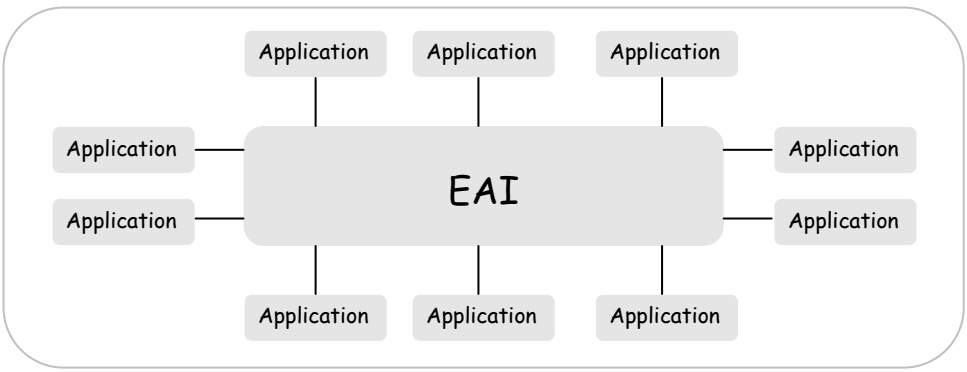


Fig. 2: intégration avec un EAI.

La plupart des applications peuvent assez aisément se brancher sur ce bus de communication purement logiciel par l'intermédiaire d'interfaces génériques. Le modèle en n*(n-1)/2 du point à point est avec un EAI remplacé par un modèle en n. Une fois branchées sur l'EAI, il convient de faire travailler conjointement ces applications. Cela revient à mettre en relation des briques logicielles qui divergent sur différents points : modèle de données, moyens de communication supportés, etc …

De son succès dépend la capacité à partager et manipuler les données de manière consistante, à bâtir des processus automatisés mettant en jeu plusieurs applications (et/ou plusieurs utilisateurs) et à construire de nouvelles applications en agrégeant des services existants. L'intégration d'applications est donc bien plus qu'un simple problème de « tuyauterie inter applicative » comme il est parfois perçu parce qu'abordé d'un point de vue technologique [2].

2 Aperçu sur l'intégration

Que recouvre exactement l'intégration des applications? Quels en sont les bénéfices? Comment les entreprises peuvent-elles réagir aux changements rapides des informations, des technologies et des systèmes?

De nos jours, l'intégration est considérée comme l'un des derniers challenges de l'informatique. Il désigne l'action d'intégrer. L'intégration est l'acte d'incorporation d'éléments constitutifs par lequel on va rendre un ensemble complet, lui conférer les propriétés liées à l'interopérabilité et à la cohérence des SI. Au sein de l'entreprise, il peut exister plusieurs approches permettant d'appréhender le problème d'intégration.

Principalement, on peut distinguer quatre types fondamentaux. Il s'agit respectivement en fonction de leur degré de complexité, de l'intégration de données, de processus, des interfaces et des applications [3].

2.1 Intégration de données

C'est la forme la plus simple de l'intégration. Elle apparaît au niveau des bases de données. D'une part, elle est assurée par duplication des copies d'une partie ou de toute la base de données dans une ou plusieurs applications. D'autre part, l'intégration s'effectue par le transfert des données, en utilisant des outils pour permettre aux données d'émigrer d'une application à une autre. Ce transfert de données est généralement réalisé par ETL (Extract, Transform and Load) [4]. ETL est un moteur qui extrait, transforme, épure puis charge les données à partir de différentes applications vers des entrepôts de données. Il est aujourd'hui la solution la plus préconisée dans l'intégration des données.

2.2 Intégration des processus

C'est la forme la plus complexe de l'intégration. Elle sert à rendre valable une application dans le contexte d'une autre sans la dupliquer. Elle permet aussi de construire de nouveaux processus métier à base des applications et progiciels existants. Ceci crée de nouvelles opportunités pour l'entreprise à moindre coût. Les données circulant dans la nouvelle organisation sont accédées et maintenues selon une logique de métier (business logic) qui a des règles et une sécurité de données. Ces données ne sont plus simples mais des objets métier (BOD : Business Object Document, ex bon de commande) qui portent déjà un sens [5]. Grâce à cette forme d'intégration, les nouveaux processus métier qui les manipulent sont crées.

2.3 Intégration des interfaces

C'est le remplacement des interfaces graphiques des applications par une interface standard. Généralement, les fonctionnalités des applications assurant l'affichage peuvent être projetées une à une sur celles de l'interface utilisateur standard.

La nouvelle couche présentation est alors intégrée avec les applications des systèmes classiques (legacy system) ou les progiciels utilisés. Les portails d'entreprise (Enterprise Business Portals) sont une autre solution sophistiquée pour cette forme d'intégration. Ils consolident la présentation des différentes applications par une présentation graphique. C'est une solution middleware [6].

2.4 Intégration des fonctions

Cette forme d'intégration permet à une application d'invoquer le code d'une fonction ou méthode d'une autre application sur des plateformes différentes à travers son API (Application Programming Interface). Ceci peut être fait de différentes manières et à l'aide de différents outils, tels que : RPC (Remote Procedure Call) CORBA (Commun Object Request Broker Architecture) RMI (Remote Method Invocation) SOAP (Simple Object Access Protocol) [7] …

3 Intégration d'applications

L'EAI est un terme qui regroupe les méthodes et les outils visant à moderniser, consolider et coordonner les différentes solutions logicielles d'une entreprise Typiquement, une entreprise dispose d'applications et de bases de données qu'elle désire continuer à utiliser tout en développant ou en migrant de nouvelles applications dans le but d'exploiter un site web (e-commerce) ou construire un extranet avec ses partenaires [1].

Elle peut également vouloir ajouter de nouvelles fonctionnalités à ses applicatifs, afin de les pérenniser, mais ne pas vouloir/pouvoir modifier ses systèmes d'information. Dans ce cas, elle peut développer une solution en intranet et, grâce à l'EAI, la relier aux autres systèmes existants. De plus, il est ainsi possible d'effectuer un reporting centralisé (consolidé) des données en provenance de sources très diverses. Ainsi, l'EAI peut être vu comme une meta-structure coordonnant des applicatifs intranet, Internet

L'objectif majeur des technologies d'intégration est de rendre l'entreprise plus efficace, plus rentable et plus accessible aux tiers ; en d'autre termes, d'en faire un partenaire commercial plus efficace [8] – Gartner Group-

Avec l'utilisation des techniques d'intégration, les bénéfices pour les systèmes d'informations sont [5] :

- **un gain de flexibilité**: une modification dans une application n'a d'impact que sur le serveur d'applications, et non sur les X destinataires qui l'utilisent,
- **un gain de robustesse**: la centralisation des flux permet un réel suivi, des sauvegardes, des reprises, etc.,

- **un gain en sécurité**: les technologies d'intégration se fondent sur des mécanismes asynchrones et peuvent offrir une gestion poussée de la sécurité.

3.1 Définitions et challenges

L'EAI : c'est un concept qui regroupe un ensemble de méthodes, de technologies et d'outils pour consolider et coordonner différentes applications d'une entreprise afin d'unifier son SI [5].

De point de vue architecture, l'intégration d'applications d'entreprise est une plate-forme reliant les applications hétérogènes du système des systèmes d'information autour d'un bus logiciel commun, chargé du transport des données [9].

L'EAI sert à réaliser les objectifs suivants :

- Créer une infrastructure intégrée pour relier des systèmes (applications et sources de données) dispersés au sein de l'entreprise corporée.
- Fournir une solution full duplex et bidirectionnelle pour partager des informations de manière similaire entre les diverses applications de l'entreprise et les nouveaux progiciels commerciaux.
- Permettre l'échange de données et de processus entre les différentes applications d'une entreprise de manière rapide, efficace et transparente.

L'EAI présente certaines limitations. Il est possible d'en résumer les plus importantes comme suit [5] [7] [10] [11] :

- Les solutions actuelles souffrent du manque de services pour la modélisation et l'intégration des processus métier. Ceci engendre un fossé, en rendant délicat, l'alignement métier-technologie.

- Les solutions actuelles ne résolvent pas le problème lié à la sémantique, dans la mesure où les mécanismes proposés sont basés au mieux sur des mapping ou des transformations syntaxiques.

- Les solutions d'EAI sont généralement propriétaires, ce qui pose le problème de standardisation et d'ouverture et ce malgré la percée des e-services qui n'ont toujours pas atteint le degré de maturité souhaité en demeurant toujours limités, notamment au niveau sécurité, transaction et qualité de services.

- Les solutions actuelles souffrent du manque d'adaptation et d'apprentissage qui peuvent s'avérer nécessaires dans certaines situations, notamment celles liées aux changements métiers, technologiques ou stratégiques.

Ces quelques problèmes, que nous avons cités, et qui s'avèrent de véritables challenges, peuvent constituer, à notre connaissance, des perspectives assez prometteuses.

3.2 Approches d'intégration d'applications

L'analyse de la littérature sur l'EAI permet de mettre en évidence principalement quatre approches qui sont [5][12][13] : intégration des applications par les données, traitements, présentations et processus. Les trois premières approches peuvent être considérées comme le résultat logique de la structuration des applications en couches, tandis que la quatrième approche, qui se décline par la suite en une combinaison des trois approches de bases, résulte de la mise en œuvre d'un médiateur (moteur de workflow), et qui permet ainsi d'interconnecter des applications via l'orchestration des processus.

3.2.1 Intégration d'applications par les données

Cette approche constitue la manière la plus répandue pour intégrer des applications. Elle se réalise en faisant appel à l'intégration de données. Elle consiste également à déplacer les données entre les applications, en utilisant, très souvent, des outils de migration, de réplication ou de fédération de données. (Voir Fig. 3)

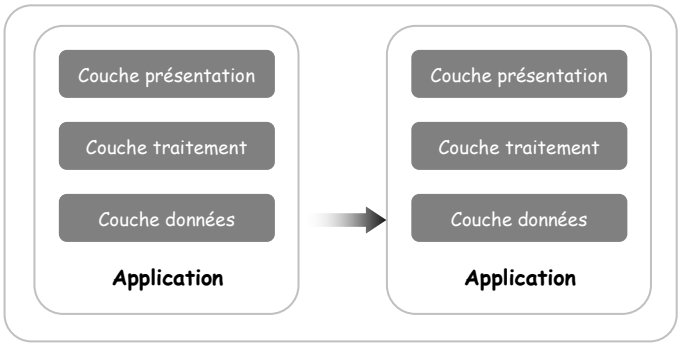


Fig. 3: Intégration d'applications par les données.

Bien que relativement élémentaire, cette approche présente tout de même l'avantage de ne pas induire de changement, ni au niveau de la couche présentation, ni au niveau de la couche traitement. Ce qui permet d'économiser énormément sur le plan financier. Cependant, l'inconvénient majeur réside dans le fait que l'on peut facilement introduire des incohérences au niveau des données en outrepassant les contraintes d'intégrité [14].

3.2.2 Intégration d'applications par les fonctions (traitements)

Cette approche permet d'intégrer des applications en partageant des services offerts par les composants applicatifs (méthodes, objets, etc.…) cette approche repose le plus souvent sur la notion d'événements qui permet d'invoquer ces services. A titre d'exemple, on peut citer des techniques basées sur le mécanisme traditionnel d'API (Application Programming Interface), d'applications composites ou encore les récents e-Services [7]. C'est ce type d'approche qui se trouve généralement à la base de la mise en place d'une application client sur le web et qui peut être destinée à la fois au B2B et au B2C. (Voir Fig. 4)

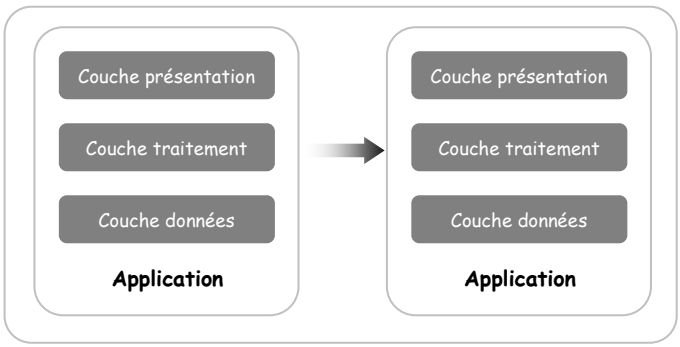


Fig. 4: intégration d'applications par les traitement.

Cette approche offre ainsi l'avantage du partage de la logique applicative entre les applications et/ou les processus, ce qui permet une plus grande réutilisation, distribution ainsi que le support des transactions [5].

3.2.3 Intégration d'applications par les interfaces (présentations)

Cette approche représente sans doute le type d'intégration le plus léger dans la mesure où elle permet d'intégrer les applications au niveau de l'interface utilisateur en fournissant le plus souvent une interface commune. Telle est largement utilisée pour les applications de l'Internet. (Voir Fig. 5)

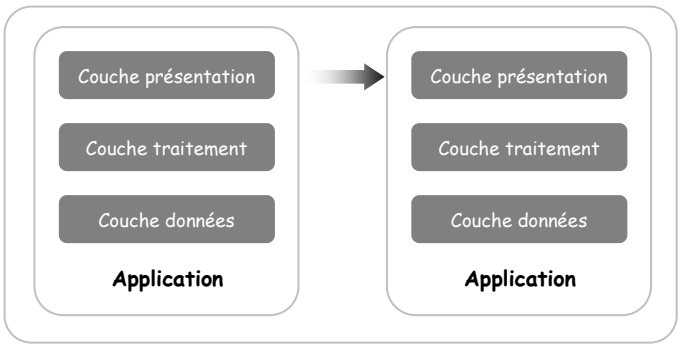


Fig. 5: intégration d'applications par les présentations.

Cette approche ne se préoccupe cependant pas de l'intégration, ni des données, ni des traitements et qui sont laissés, très souvent, ainsi, à l'initiative de l'utilisateur [5][14].

3.2.4 Intégration d'applications par les processus

Cette approche constitue la forme la plus évoluée et la plus flexible afin d'interconnecter des applications. Elle se base souvent sur l'utilisation de l'approche BPI (Business Process Intégration) qui met en œuvre un moteur workflow. Ce moteur est un middleware permettant de relier les applications aux processus en fonction des événements métiers. Ceci permet généralement d'obtenir une plus forte valeur ajoutée [15]. Bien qu'elle soit assez intéressante, cette forme d'intégration manque encore de maturité. (Voir Fig. 6)

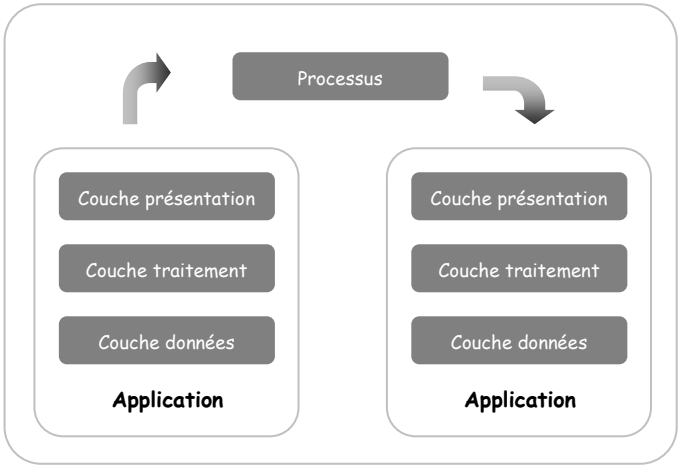


Fig. 6: intégration d'applications par les processus.

Nous signalons que ces différentes approches ne sont pas exclusives car elles peuvent être mises en œuvre simultanément. C'est à dire que l'on peut intégrer certaines applications par les données, d'autres par les traitements, etc. [5].

4 Typologies des applications intégrées

L'intégration des applications ne peut se faire qu'en examinant les différents types d'applications à intégrer (batch, transactionnelles, client-serveur, Web, etc.) et leurs caractéristiques spécifiques.

L'examen de chaque application à intégrer permettra de déterminer les contraintes découlant des caractéristiques propres aux différents types d'applications avec lesquelles on va lui demander de communiquer. Par exemple, une application web mise en relation avec une application "batch", dont le traitement s'effectue de manière différée, ne pourra se faire que sous certaines conditions compte tenu des différences de conception de ces applications spécifiques.

Dans l'analyse des applications à intégrer, il faut tenir compte:

- des formats des données ou des événements.
- du volume de ces données et événements.
- de leur capacité d'échange avec d'autres applications.
- du rythme de fonctionnement de ces applications.

4.1 Les applications batch

Ce sont les plus anciennes applications conçues avec une philosophie d'application monolithe lourde. Elles traitent, en différé, un ensemble d'événements groupés dans des fichiers ou dans des bases de données. La périodicité est particulière à chaque application.

Les volumes supportés par de telles applications peuvent être importants: un lot peut contenir des millions d'événements. Par ailleurs, les formats des données et des événements sont généralement assez simples (par exemple des formats "plats" développés

en langage Cobol). Le calcul des salaires dans une entreprise est un bon exemple d'application batch.

L'intégration de ce type d'applications doit obligatoirement se faire via un fichier (par exemple au format XML) ou via la base des données de cette application.

4.2 Les applications transactionnelles

Ces applications traitent les événements les uns après les autres. La plupart de ces applications assurent des interfaces avec les utilisateurs.

Le format des données et des événements dépend des technologies utilisées. En général, ce type d'applications traite des formats un peu plus complexes que les applications batch. En ce qui concerne les volumes traités, ces applications peuvent absorber un grand nombre d'événements. Cependant, par événement, le volume des données est le plus souvent petit ou moyen. Les seules solutions de communication avec ce type d'applications sont la base de données et l'interface écran.

4.3 Les applications Client-Serveur

Ces applications sont considérées comme une forme plus évoluée des applications transactionnelles. Tout en gardant les caractéristiques des applications transactionnelles, on y trouve en plus:

- la capacité de traiter des structures complexes (par exemple XML),
- les caractéristiques du modèle client-serveur:
    - le client contacte le serveur et lui envoie les demandes.
    - le serveur traite les demandes du client et lui répond.
    - plusieurs clients accèdent en même temps au serveur.
- la communication qui se fait:
    - via des bases de données ou fichiers,
    - via des technologies Internet: http, https, SMTP, SOAP.

Les applications web étant accessibles via Internet, le nombre d'utilisateurs, simultanément connectés, n'est pas maîtrisable et peut être extrêmement important.

4.4 Les applications basées sur un échange des messages

Ces applications sont apparues assez récemment et utilisent les technologies de messages inter applications: MOM (Message Oriented Middleware). Elles peuvent fonctionner d'une manière hybride:

- soit en batch,
- soit en transactionnel en traitant les événements au rythme de leur apparition.

Les structures des données traitées sont complexes et les volumes traités peuvent être très importants. Le dialogue entre ces applications est réalisé via les files d'attente du MOM.

4.5 Les progiciels

Logiciels spécialisés "métier", les progiciels couvrent une large palette de fonctionnalités nécessaires dans un système d'informations d'une entreprise. Parmi les plus répandus, on peut citer les ERP (Entreprise Ressources Planning) offrant des fonctionnalités de plus en plus complexes (gestion des stocks, comptabilité analytique, etc.).

La connectivité a longtemps été pointée du doigt comme leur point faible, donnant parfois à l'entreprise la sensation d'être "prisonnière" du progiciel. Elle est désormais un facteur de différenciation entre les constructeurs. L'échange des informations avec ces progiciels est possible au travers de fichiers, files d'attente, bases de données, API, protocole http, etc. Leur intégration au reste des systèmes de l'entreprise est donc, aujourd'hui, largement favorisée.

5 Topologies des architectures d'intégration

Nous distinguons dans la littérature quatre topologies de base [16] :

5.1 Topologie point à point

C'est la topologie d'intégration d'applications utilisée dans les systèmes actuels de type "spaghetti" avec un couplage fort. Chaque application est spécifiquement connectée avec une autre. Cette approche est peu flexible, un changement dans une application implique un changement dans toutes les applications avec lesquelles elle est connectée. (Voir Fig. 7)

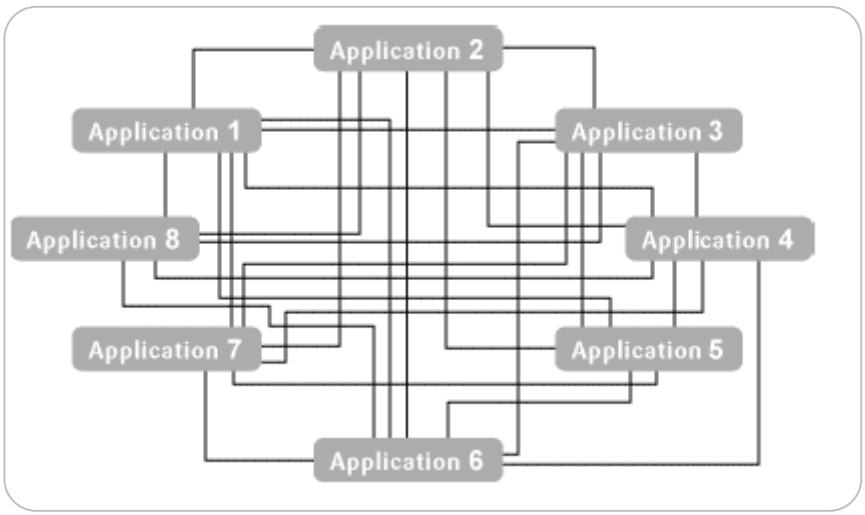


Fig. 7: Topologie point à point.

De très nombreuses entreprises souffrent aujourd'hui du syndrome des ces applications (applications spaghetti), c-à-d à ce point liées voire même emmêlées entre elles, qu'il est devenu très difficile de les gérer et de les faire évoluer sans risque. L'entreprise confrontée à des applications fortement couplées et difficilement gérables doit s'orienter vers un bus ou un hub d'intégration jouera le rôle de chef d'orchestre en assurant la gestion de tous les échanges d'informations.

5.2 Topologie par bus

Cette intégration suit le principe de « «publier et s'inscrire » décentralisé. Chaque application s'abonne (s'inscrire) à une diffusion broadcast ou multicast. Les applications déposent (publier) les messages sur les bus et les applications abonnées les récupèrent. Le bus se trouve sur un serveur dédie, le serveur d'intégration. (Voir Fig. 8)

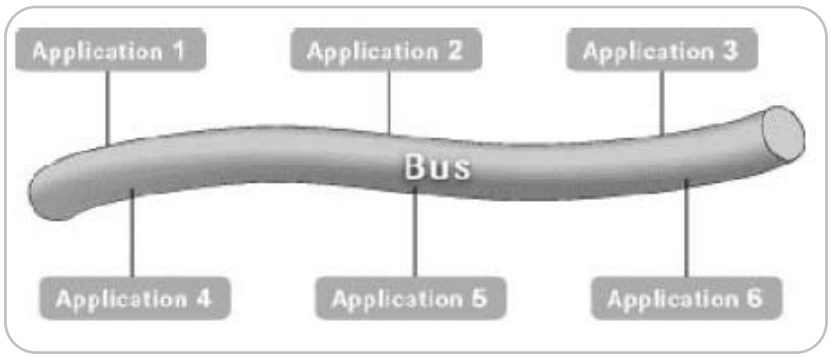


Fig. 8: Topologie par bus.

5.3 Topologie par hub-and-spoke

Dans cette topologie, on connecte les applications à intégrer au hub central en utilisant des adaptateurs appropriés. Le serveur d'intégration prend en charge la transformation des messages d'un format vers un autre ainsi que le problème de routage vers le bon destinataire. (Voir Fig. 9)

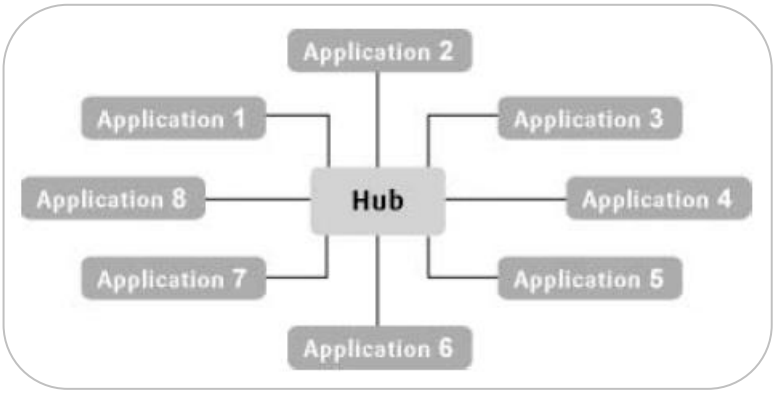


Fig. 9: Topologie Hub and Spoke.

Le modèle d'architecture Hub and Spoke est un modèle centralisé. Ce modèle se base sur un Hub qui a pour rôle de centraliser toutes les informations. Ce modèle utilise aussi le référentiel regroupant toutes les données nécessaires. Cette base de données est également centralisée. Le hub est un point de passage obligé pour tout échange d'information et sans lui aucun flux n'est réalisable.

Son rôle est de recevoir les messages, de les traduire et de les diriger vers les spokes. Un spoke est une branche (connecteur) attachée au hub. Ce modèle est préconisé pour le

mode d'échange "Publish and subscribe centralisé", c'est à dire que l'information passe d'abord par le hub ou bus, puis est transférée à l'utilisateur destinataire.

L'avantage d'un système centralisé se situe au niveau de l'administration du SI, car tout est concentré en un point. Toutefois, la gestion des flux n'est pas aussi simple. En effet, pour relier un réseau à un autre il faut installer un nouveau hub, ce qui alourdit l'administration du SI. Mais l'entreprise bénéficie du fait qu'elle achète un hub en fonction de la montée en charge. Cela signifie que l'entreprise n'a pas un gros investissement à faire et qu'elle peut acheter son matériel au fur et à mesure et répartir ses coûts. On peut dire que c'est un avantage non négligeable pour les entreprises [17].

5.4 Topologie network centric

En opposition au modèle hub and spoke, le modèle Network Centric est décentralisé. Il s'articule autour des référentiels de règles de gestionnaires de messages, relayés vers les nœuds correspondants. Un nœud est un point de connexion à une application. Les nœuds permettent de traiter l'information et de l'orienter vers la bonne application. Cette architecture a pour avantage de répartir l'ensemble des données sur la globalité des nœuds.

Malgré, les différences de coûts, aucun de ces modèles n'est meilleur que l'autre. Chacune des entreprises doit évaluer ses besoins et ses moyens et choisir l'architecture qui lui convient le mieux [17].

6 Modèles d'intégration d'applications

Il existe dans la littérature plusieurs patrons ou modèles d'architecture pour l'EAI. Lutz en recense principalement cinq (architecture pattern) [18]. Ce sont : l'adaptateur, le messager, la façade, le médiateur et l'automate du processus.

- L'adaptateur permet de convertir l'interface d'une application existante en une interface « souhaitée » par d'autres applications ayant besoin d'utiliser cette interface.

- Le modèle de messager permet de minimiser les dépendances de communication entre applications, en jouant le rôle d'intermédiaire pour passer l'information et le contrôle entre des applications.
- Le modèle de la façade permet de fournir une interface simplifiée de l'application, en minimisant les dépendances entre les applications clientes et les applications serveurs.

- Le médiateur permet d'encapsuler la logique d'interaction de l'application, en minimisant ainsi les dépendances entre les applications.

- L'automate du processus permet de décrire une approche architecturale minimisant les dépendances entre la logique d'automatisation du processus et les applications.

Ces cinq modèles constituent la base des architectures pour l'EAI et ils possèdent de nombreuses variations obtenues principalement par hybridation de ces formes de base [16] [18]. Parmi ces cinq patrons, certains sont destinés à la topologie point à point tels que l'adaptateur et la façade, alors que les autres utilisent plutôt la topologie structurelle (hub-and-spoke, bus). De plus, les patrons de médiateur et d'automate du processus tentent de séparer la logique d'interaction entre des applications participantes afin de mieux les gérer, tendance qui se confirme sur le terrain sur de nombreux produits d'EAI tels que NetEAI, Tibco, Weblogic, Websphere, etc.

7 Technologies d'intégration d'applications

Différentes technologies ont tenté d'apporter des réponses aux besoins d'EAI et ce en cherchant à proposer des modèles plus ou moins pertinents. Tout d'abord les progiciels intégrés ERP (Enterprise Resource Planning), CRM (Customer Relationship Management) et SCM (Supply Chain Management) offrent des solutions centralisées, souvent partielles et limitées à leur périmètre de déploiement.

Ainsi, d'autres technologies peuvent être mises en œuvre dans le cadre de l'EAI. Elles reposent toutes sur un concept clé de « middleware ». Un middleware est un ensemble de technologies permettant de faire communiquer des applications en fournissant des services de transport et de routage [5] [7] [19].

Nous pouvons résumer les différentes technologies de middleware dans le tableau suivant:

Type de middleware	Description	Type d'intégration	Exemple
DOM	Intégration en utilisant les bases de données	A2A	ODBC, JDBC, EDI,…
WOM	Intégration en utilisant les entrepôts de données	A2A	ETL, …
ROM	Intégration par appel de procédure à distance	A2A	RPC, …
MOM	Intégration par échanges de messages	A2A	MSMQ, MQSeriies
OOM	Intégration par distribution d'objets	A2A	CORBA, DCOM, OLE,…
COM	Intégration par utilisation de composants	A2A	Java RMI,…
POM	Intégration par utilisation de portails	B2B	Potail B2C, B2B, …
TOM	Intégration via la gestion de transactions par des serveurs d'applications	A2A	Serveurs d'applications
SOM	Intégration en utilisant la notion de service	B2B	ESB, .NET, Websphere
IOM	Intégration en utilisant un serveur d'intégration	A2A	(Hub and Spoke (Vitria BW, …)

Tableau 1: Technolgies d'EAI.

D'après le tableau (Tableau 1), nous pouvons classer les middlewares en cinq catégories :

1. Middlewares orientés portails : ces middlewares permettent une intégration légère d'applications en offrant aux utilisateurs un seul point d'entrée leur permettant

d'accéder à l'aide d'une interface unique. Généralement, on utilise les portails pour l'intégration B2C.

2. Middlewares orientés services : ce sont les middlewares qui se basent sur le concept de e-Services et qui référencent généralement le standard XML. Les systèmes construits autour de ce type de middleware reposent généralement sur un ensemble des technologies permettant de développer WSDL (Web Services Description Language), publier UDDI (Universal Discovery Description and Inventory) et communiquer SOAP (Simple Object Access Protocol) [7].

3. Middlewares orientés données: ces middlewares permettent d'intégrer les applications en déplaçant les données entre les applications en utilisant des outils de migration, de réplication et de fédération de données. Parmi ces middlewares, nous pouvons citer: WOM (Warehouse Oriented Middleware), DOM (Database Oriented Middleware) …etc.

4. Middlewares orientés traitements: cette catégorie permet d'intégrer les applications en partageant les services offerts par les composants applicatifs (objets, méthodes,…). Cette approche repose le plus souvent sur la notion d'évènements qui permet d'invoquer ces services. Parmi ces middlewares: COM (Component Oriented Middleware), ROM (Remote Oriented Middleware)…etc.

5. Middlewares orientés serveurs d'intégration : ils constituent la forme la plus aboutie des technologies d'intégration intra-entreprise. Bien qu'ils reposent très souvent sur des produits propriétaires, ces serveurs offrent des fonctionnalités de base : connectivité, routage, transformation sécurité, management et qualité de service.

8 Conclusion

L'EAI se présente comme une pièce maîtresse du système d'information. Il permet l'informatisation de tous les services de l'entreprise en rendant le système d'information communiquant et réactif. On comprend mieux l'engouement des entreprises pour cette nécessité. Le marché est en plein développement. L'EAI sera un support d'intégration pour les nouvelles technologies à venir. Cependant, la simplicité du concept ne doit pas faire oublier la complexité de la mise en oeuvre. Certes le plat de « spaghetti » n'est plus visible mais il est toujours présent caché à l'intérieur de l'EAI. De plus, il n'existe pas de solution adaptée à toutes les entreprises. Une architecture basée sur un modèle universel devient une nécessité grandissante.

Nous avons présenté, dans ce chapitre, le domaine de l'intégration d'applications d'entreprise. Nous avons découvert ce qu'est la notion d'"intégration", ses différentes formes, ses principales approches, utilisées pour intégrer les applications. Nous avons montré les typologies et topologies de ses architectures. Et finalement, nous avons présenté, brièvement, les différentes technologies d'intégration, à savoir, les middlewares orientés données, les middlewares orientés services,…etc.

L'étude de ce chapitre nous permettra, donc, de recenser le vocabulaire inhérent au domaine de l'intégration d'applications, nous pouvons répertorier ce vocabulaire sur trois niveaux complémentaires: niveau comportement, niveau structure et niveau domaine. Le niveau comportement se base sur les deux approches d'intégration par les traitements et par les processus, le niveau domaine s'appuie sur l'intégration d'applications par les données et enfin le niveau structure qui représente l'ensemble des concepts qui définissent comment une application est reliée à un middleware à savoir CORBA, Java RMI, RPC, etc. Par conséquent, notre ontologie d'application, que nous venons la construire, cernera les trois approches d'intégration (données, traitement, processus) et cela dépendra sur le fonctionnement de l'application vu que certaines applications communiquent, fréquemment, en partageant des bases de données et cela nécessite une intégration d'applications par les données, alors que d'autres des composants et celles-ci exigent une intégration d'applications par les traitements, par contre, d'autres applications interviennent au niveau Web, et ces dernières requièrent une intégration d'applications par les processus.

Le prochain chapitre est consacré à la présentation des ontologies. Nous commençons par la définition de la notion d' "ontologie" en ingénierie des connaissances. Nous présentons ensuite les principaux formalismes de représentation de connaissances à savoir les frames, les graphes conceptuels et les logiques de descriptions. Nous découvrirons, après, les méthodologies les plus représentatives de leur construction et quelques domaines de leur utilisation. A la fin, nous présenterons les outils nécessaires de leur développement, à savoir, les langages de représentation, les outils d'édition et d'interrogation, …etc.

Chapitre II

L'ingenierie ontologique

1 Introduction

L'ontologie est un terme emprunté à la philosophie. Dans cette discipline, l'ontologie est un ensemble d'objets existants. L'ontologie est identiquement une étude de l'être en tant qu'être, c'est-à-dire, une étude des propriétés générales de ce qui existe. Cependant, en informatique et d'après l'encyclopédie Wikipédia [20], une ontologie est un ensemble structuré de concepts. Les concepts sont organisés dans un graphe dont les relations peuvent être des relations sémantiques où des relations de composition et d'héritage (au sens objet). La structuration des concepts, dans une ontologie, permet de définir des termes les uns par rapport aux autres, chaque terme étant la représentation textuelle d'un concept [21].

Le terme d'ontologie est utilisé depuis le début des années 1990 dans les domaines de l'intelligence artificielle, en particulier de l'ingénierie des connaissances et de la représentation des connaissances. Son champ d'application s'élargit considérablement et il fait désormais partie des objets de recherche courants. Une ontologie est un système formel dont l'objectif est de représenter les connaissances d'un domaine spécifique au moyen d'éléments de base, les concepts, définis et organisés les uns par rapport aux autres [22].

Dans ce chapitre, nous relèverons les différentes définitions qui ont été attribuées à la notion d'ontologie, nous verrons aussi les différents éléments dont elle est composée et les besoins auxquels elle répond sont décrits. Nous monterons aussi le but de l'utilisation des ontologies dans le domaine du Web sémantique ainsi que leur place dans les systèmes à bases de connaissances. Ensuite, nous mentionnons le processus et les méthodologies servant leur construction, ainsi que les principaux formalismes utilisés pour servir leur représentation. Finalement, la dernière section va être plus spécifiquement consacrée aux outils de développement d'ontologies y compris les langages de spécification, les éditeurs, les serveurs, les langages et les plateformes d'interrogation d'ontologies.

2 Notion d'ontologie

Afin de faciliter le partage et la réutilisation de connaissances formellement représentées dans des systèmes d'intelligence artificielle, il est très utile de définir un vocabulaire commun dans lequel la connaissance partagée est représentée. La spécification de ce vocabulaire est communément appelée une ontologie. De ce fait, les ontologies définissent actuellement des vocabulaires structurés, regroupant des concepts utiles d'un domaine et de leurs relations et qui servent à organiser et échanger des informations de façon non ambiguë. Leur développement progressif en (IA) Intelligence Artificielle parvient de leur intérêt, pour associer la sémantique à des ressources ou bien à des entités textuelles, pour faciliter la localisation et la gestion des connaissances dans diverses applications [23].

2.1 Définitions

De nos jours, il n'existe aucun consensus sur le sens exact du terme ontologie. Il est difficile de le définir d'une façon définitive. Le mot est en effet employé dans des contextes

très différents touchant la philosophie, la linguistique ou l'intelligence artificielle. De nombreuses définitions ont été offertes pour donner un éclaircissement sur ce terme, mais aucune de ces définitions ne s'est explicitement imposée. Cependant, le terme d'ontologie est employé parfois de façon confuse et abusive [24]. Les définitions de ce terme ne sont pas toujours consistantes et cela dépend des domaines spécifiques [25], [26], [27]. Pour ne pas dévier de notre propos, nous avons recensé les définitions suivantes:

Définition1 : « une ontologie définit les termes et les relations de base du vocabulaire d'un domaine ainsi que les règles qui indiquent comment combiner les termes et les relations de façon à pouvoir étendre le vocabulaire » -Neches, 91-

Dans le cadre de l'intelligence artificielle, Neches proposa cette définition afin de montrer comment l'élaboration des ontologies s'effectuent, et cela en présentant les directives relativement floues : repérer les termes de base et les relations entre les termes, identifier les règles servant à les combiner, fournir des définitions de ces termes et de ces relations. Notons que d'après cette définition, une ontologie inclut non seulement les termes qui y sont explicitement définis, mais aussi les termes qui peuvent être créés par déduction en utilisant les règles.

Définition2: « une ontologie est une spécification explicite et formelle d'une conceptualisation partagée» -Gruber, 93-

En 1993, Gruber propose cette définition afin de montrer qu'une ontologie est un ensemble de définitions, de primitives, de représentations de connaissances spécifiques au contenu : classes, relations, fonctions et constantes d'objet. Cette définition est la plus référencée et la plus synthétique. Nous allons la conserver dans la suite de notre travail.

Le terme conceptualisation réfère dans cette définition à une abstraction d'un phénomène du monde, obtenue en identifiant les concepts appropriés à ce phénomène. « Formelle » indique que les ontologies sont interprétables par la machine. Cependant, «Spécification explicite » signifie que les concepts de l'ontologie et les contraintes liées à leur usage sont définis de façon déclarative. Et enfin le terme «Partagé» signifie que l'ontologie capture la connaissance consensuelle. Mais cette définition laisse, déjà, la porte ouverte à de nombreuses définitions.

Définition3: « une ontologie est une description formelle d'entités et leurs propriétés, relations, contraintes, comportement» -Grüninger, 95-

Cette définition de Grüninger est simplifiée dans [28] où une ontologie est définie comme un ensemble de définitions de concepts et leurs relations. A ne pas confondre avec un modèle qui est un ensemble d'instances de ces concepts.

Définition4: « Les ontologies sont des spécifications partielles et formelles d'une conceptualisation commune» -Guarino, 97-

En 1997, Guarino accentue l'ambigüité du terme conceptualisation qui doit être pris dans son sens intuitif. La spécification des ontologies est partielle, car une conceptualisation ne peut pas toujours être entièrement formalisée dans un cadre logique, du fait d'ambiguïtés ou du fait qu'aucune représentation de leur sémantique n'existe dans le langage de représentation d'ontologies choisi. « Commune » renvoie à l'idée qu'une ontologie rend compte d'un savoir consensuel, c'est-à-dire qu'elle n'est pas l'objet d'un individu, mais qu'elle est reconnue par un groupe [29].

Pour conclure cette section, nous pouvons donc affirmer que les définitions du terme ontologie abondent dans la littérature scientifique. Les définitions, dans leur diversité, offrent des points de vue à la fois différents et complémentaires sur un même concept.

2.2 Que représente-t-on dans une ontologie ?

Comme mentionné plus haut, les ontologies produisent un vocabulaire commun d'un domaine et définissent, de façon plus ou moins formelle, la signification des termes et des relations entre eux. Les connaissances intégrées dans les ontologies sont formalisées en mettant en jeu cinq types de composants : concepts, relations, fonctions, axiomes, instances [30][31].

Concepts : Ils sont appelés aussi termes ou classes de l'ontologie. Un concept est un constituant de la pensée (un principe, une idée, une notion abstraite) sémantiquement évaluable et communicable. L'ensemble des propriétés d'un concept constitue sa compréhension ou son intension et l'ensemble des êtres qu'il englobe, son extension. Selon [32] un concept se définit à trois niveaux Un concept est une signification. Sa place dans un système de significations permet de le comprendre, de le distinguer et de le différencier par rapport à d'autres concepts. Un concept est une construction. Comprendre un concept revient à construire l'objet dont il est le concept. Un concept est une prescription. On le comprend en exécutant l'action qu'il entreprend.

Selon [30], ces concepts peuvent être classifiés selon plusieurs dimensions :

- Niveau d'abstraction (concret ou abstrait) ;
- Atomicité (élémentaire ou composée) ;
- Niveau de réalité (réel ou fictif).

En résumé, un concept peut être tout ce qui peut être évoqué et, partant, peut consister en la description d'une tâche, d'une fonction, d'une action, d'une stratégie ou d'un processus de raisonnement, etc.

Relations : Représentent un type d'interaction, ou bien des associations existant entre les concepts d'un domaine. Elles se définissent formellement à partir d'un produit de n concepts :

$R : C_1 \times C_2 \times \ldots \times C_{n-1} \rightarrow C_n$

sous-classe-de (Spécialisation, généralisation), partie-de (agrégation ou composition), associée-à, instance-de sont des exemples de relations binaires.

Voici quelques relations les plus courantes dans la littérature :

1) L'équivalence : une relation R est une relation d'équivalence si et seulement si : R est symétrique, réflexive et transitive. On écrit :

(R est une relation d'équivalence) ⇔ *((R symétrique)* ^ *(R réflexive)* ^ *(R transitive))*

2) la cardinalité : c'est le nombre possible de relations de ce type entre les mêmes concepts (ou instances de concept). Les relations portant une cardinalité représentent souvent des attributs. Exemple : une pièce a au moins une porte, un humain a entre zéro et deux jambes.

3) L'incompatibilité : Deux relations sont incompatibles si elles ne peuvent lier les mêmes instances de concepts. Exemple : les relations «être rouge » et «être vert » sont incompatibles ;

4) L'inverse : Deux relations binaires sont inverses l'une de l'autre si, quand l'une lie deux instances I1 et I2, l'autre lie I2 et I1. Exemple : les relations « a pour père » et « a pour enfant » sont inverses l'une de l'autre ;

5) L'exclusivité : Deux relations sont exclusives si, quand l'une lie des instances de concepts, l'autre ne lie pas ces instances, et vice-versa. L'exclusivité entraîne l'incompatibilité. Exemple : l'appartenance et la non appartenance sont exclusives.

Et bien d'autres relations…

Fonctions : ce sont des cas particuliers de relations dans lesquelles le N^{éme} élément de la relation est défini de manière unique à partir des n-1 premiers. Formellement, les fonctions sont définies ainsi :

F: $C_1 \times C_2 \times \ldots \times C_{n-1} \rightarrow C_n$.

Comme exemple de fonctions binaires, nous avons la fonction mère de et le carré, et comme exemple de fonction ternaire, le prix d'une voiture usagée sur lequel on peut se baser pour calculer le prix d'une voiture d'occasion en fonction de son modèle, de sa date de construction et de son kilométrage. [33]

Axiomes : constituent des assertions, acceptées comme vraies, à propos des abstractions du domaine, traduites par l'ontologie. Ils ont pour objectif de représenter des concepts et des relations dans un langage logique permettant de représenter leur sémantique. Ils représentent les intentions des concepts et des relations du domaine et, de manière générale, les connaissances n'ayant pas un caractère strictement terminologique [34]. L'utilisation des axiomes sert à définir le sens des entités, mettre des restrictions sur la valeur des attributs, examiner la conformité des informations spécifiées ou en déduire de nouvelles.

Instances: elles constituent la définition extensionnelle de l'ontologie; ces objets véhiculent les connaissances (statiques, factuelles) à propos du domaine du problème.

2.3 Différentes sortes d'ontologies

Cette section présente les types les plus généralement utilisés d'ontologies. On peut avoir une idée de la connaissance qui est incluse dans chaque type d'ontologie. Les ontologies peuvent être classifiées selon plusieurs dimensions. Parmi celles-ci, nous en examinerons deux *1) Objet de conceptualisation ; 2) Niveau de formalisme de représentation*

2.3.1 Objet de conceptualisation

Les ontologies classifiées selon leur objet de conceptualisation (le but de leur utilisation) dans [30], le sont de la façon suivante:*1) Supérieure/ Haut niveau; 2) Domaine ;3) Tâche; 4) Application*.la figure 1.1 montre les différents types d'ontologies selon leur objet de conceptualisation.

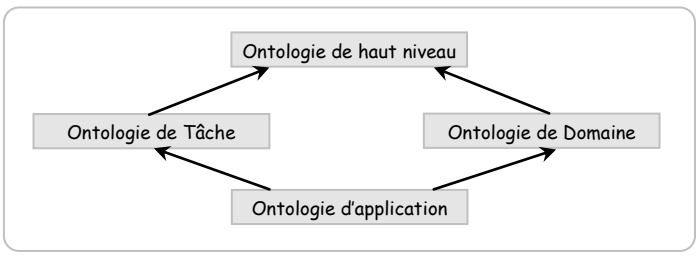


Fig. 1: Type d'ontologies selon leur objet de conceptualisation.

Ontologie de haut niveau : décrit des concepts très généraux comme l'espace, le temps, la matière, les objets, les événements, les actions, etc. Ces concepts ne dépendent pas d'un problème ou d'un domaine particulier, et doivent être, du moins en théorie, consensuels à de grandes communautés d'utilisateurs [35]. Des exemples d'ontologies de haut niveau sont Dolce ou Sumo.

Ontologie de domaine : Contrairement aux ontologies de haut niveau, les ontologies de domaine sont plus spécifiques. Elles synthétisent les connaissances spécifiques à un domaine particulier. Elles décrivent le vocabulaire ayant trait à un domaine générique (ex. : l'enseignement, la médecine…), notamment en spécialisant les concepts d'une ontologie de haut niveau [35].

Ontologie de tâches : Ce type d'ontologies est utilisé pour conceptualiser des tâches spécifiques dans les systèmes, telles que les tâches de diagnostic, de planification, de conception, de configuration, de tutorat. Soit tout ce qui concerne la résolution de problèmes. Ce type d'ontologies décrit le vocabulaire concernant une tâche générique (ex. : enseigner, diagnostiquer…), notamment en spécialisant les concepts d'une ontologie de haut niveau [35]. Certains auteurs emploient le nom « ontologie du domaine de la tâche » pour faire référence à ce type d'ontologie [31].

Ontologie d'application: Cette ontologie est la plus spécifique, elle contient des concepts dépendants d'un domaine et d'une tâche particuliers, qui sont généralement subsumés par des concepts de ces deux ontologies. Ces concepts correspondent souvent aux rôles joués

par les entités du domaine lors de l'exécution d'une certaine activité [35]. Il s'agit donc ici de mettre en relation les concepts d'un domaine et les concepts liés à une tâche particulière, de manière à en décrire l'exécution (ex. : apprendre les statistiques, effectuer des recherches dans le domaine de l'astronomie, etc.).

2.3.2 Niveau de formalisme de représentation

D'autre part, selon le niveau du formalisme de représentation du langage utilisé pour décrire l'ontologie, [36] proposent une classification comprenant quatre catégories:

1. **Informelles** : ontologies opérationnelles dans un langage naturel (sémantique ouverte) ;
2. **Semi-informelles** : utilisation d'un langage naturel structuré et limité ;
3. **Semi-formelles** : langage artificiel défini formellement;
4. **Formelles** : utilisation d'un langage artificiel contenant une sémantique formelle, ainsi que des théorèmes et des preuves de propriétés telles la robustesse et l'exhaustivité [30].

2.4 Les ontologies et les systèmes à bases de connaissances

Les ontologies sont apparues en informatique, plus précisément en Ingénierie des Connaissances, dans le cadre des démarches d'acquisition des connaissances pour les systèmes à base de connaissances (SBC). Ces démarches proposaient de dégager les objets du domaine, leur signification et leur contenu, des objets du raisonnement décrivant les règles heuristiques d'utilisation des objets du domaine, le but étant de faciliter la construction des SBC en permettant la réutilisation des composants génériques [37]. Les SBC proposaient alors de spécifier, d'un coté des connaissances du domaine modélisé, et de l'autre, des connaissances de raisonnement qui manipule et utilise ces connaissances du domaine. L'idée de cette séparation modulaire était de construire mieux et plus rapidement des SBC en réutilisant le plus possible des composants génériques, que ce soit au niveau du raisonnement ou des connaissances du domaine [38].

En conclusion, l'objectif de l'ingénierie ontologique est de diversifier les applications des Systèmes à Base de Connaissances (SBC), et de permettre une représentation des connaissances indépendantes de ces diverses applications, de manière à assurer sa portabilité d'une application à l'autre.

2.5 Les ontologies et le web sémantique

Le Web actuel est essentiellement syntaxique, la structure des ressources étant bien définie, mais leur contenu restant inaccessible aux traitements machines, seuls les humains étant capables de l'interpréter.

Le Web sémantique a alors l'ambition de lever cette difficulté en associant aux ressources du Web des entités ontologiques comme références sémantiques, ce qui permettra aux différents agents logiciels d'accéder et d'exploiter directement le contenu des ressources et de raisonner dessus. Ce référencement sémantique peut aussi résoudre les problèmes

d'interprétation des ressources informationnelles provenant des applications hétérogènes et réparties [42] et de permettre ainsi à ces applications d'être intégrées sémantiquement [43].

L'architecture du Web sémantique repose sur une hiérarchie des langages d'assertion et de description d'ontologies ainsi que sur un ensemble de services pour l'accès aux ressources au moyen de leurs références sémantiques, pour gérer l'évolution et le versionnage des ontologies [44], pour l'utilisation des moteurs d'inférences capables d'effectuer des raisonnements complexes ainsi que des services pour la vérification de la validité sémantique de ces raisonnements [45].

3 Les ontologies : différents besoins

Dans cette section, nous allons voir pourquoi a-t-on besoin des ontologies. Les ontologies sont utilisées dans plusieurs domaines, les plus répandus sont :

- Communication.
- Interopérabilité entre les systèmes.
- Ingénierie des systèmes.

La figure ci-dessous montre les domaines d'utilisations des ontologies

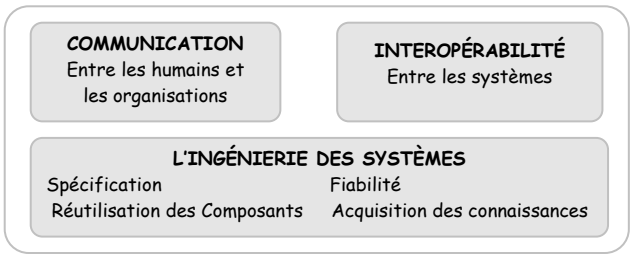


Fig. 2: Domaines d'utilisation des Ontologies.

3.1 Communication

Les humains peuvent communiquer efficacement s'ils ont des connaissances ou des points de vue partagés. Ces connaissances partagées peuvent être obtenues si le domaine est explicitement décrit sans confusion terminologique ou conceptuelle pour être compris de la même façon par tout le monde.

Une ontologie facilite la communication en fournissant une spécification explicite d'un domaine qui représente un modèle normatif. De plus, les ontologies permettent d'assurer la consistante et d'enlever l'ambiguïté dans les descriptions des connaissances concernant un domaine spécifique. Finalement, les ontologies peuvent intégrer différentes perspectives des utilisateurs. Quand les utilisateurs (qui ont différentes perspectives d'un domaine) partagent une ontologie, ils ont une perspective standard.

3.2 Interopérabilité

L'interopérabilité implique la possibilité de pouvoir demander et recevoir des services entre des systèmes interopérables. Deux systèmes sont considérés interopérables s'ils vérifient les deux conditions suivantes :

- Ils opèrent comme une unité afin de réaliser une tâche commune.
- Ils peuvent échanger des messages et des requêtes.

Les ontologies permettent de faciliter l'interopérabilité en intégrant les connaissances concernant différents domaines dont l'objectif est de décrire un domaine unifié ou accomplir une tache commune. Elles permettent aussi d'intégrer les différents vocabulaires concernant certains domaines. Pour ce faire, les ontologies de ces domaines doivent être intégrées par les méthodes d'intégration d'ontologies afin de partager un même vocabulaire.

3.3 Ingénierie des systèmes

Le développement des systèmes basé sur les ontologies a donné un profit à l'ingénierie de systèmes qui peut être résumé comme suit:

- Réutilisabilité : l'ontologie encode les informations relatives à un domaine (y compris les composants logiciels) de sorte que le partage et la réutilisation sont possibles.
- Acquisition des connaissances : l'ontologie guide l'acquisition des connaissances.
- Sûreté : l'ontologie rend possible l'automatisation du processus de vérification de consistance.
- Spécification : l'ontologie aide le processus d'identification des besoins et la définition des spécifications des systèmes [36].

4 Un squelette de méthodologie pour construire des ontologies

Le processus de construction d'une ontologie est une collaboration qui réunit des experts du domaine de connaissance, des ingénieurs de la connaissance, voire les futurs utilisateurs de l'ontologie. Cette collaboration ne peut être fructueuse que si les objectifs du processus ont été clairement définis, ainsi que les besoins qui en découlent. La figure ci-dessous représente le processus de construction d'ontologie.

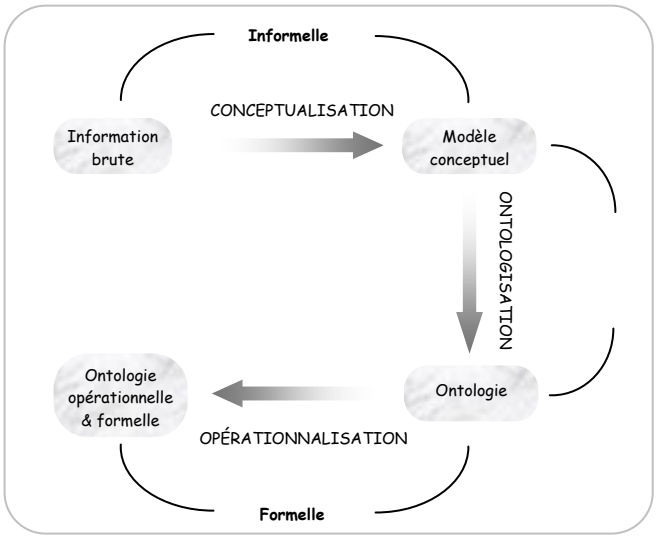


Fig. 3: Processus de construction d'ontologie[29].

4.1 Evaluation des besoins

Le but visé par la construction d'une ontologie se décline en 3 aspects:

L'objectif opérationnel : il est indispensable de bien préciser l'objectif opérationnel de l'ontologie, en particulier à travers des scénarios d'usage.

Le domaine de connaissance : il doit être délimité aussi précisément que possible.

Les utilisateurs : ils doivent être identifiés autant que faire se peut, ce qui permet de choisir, en accord avec l'objectif opérationnel, le degré de formalisme de l'ontologie, et sa granularité.

Une fois le but défini, le processus de construction de l'ontologie peut démarrer, en commençant par la phase de conceptualisation.

4.2 Conceptualisation

Cette étape permet d'aboutir à un modèle informel, donc sémantiquement ambiguë et généralement exprimé en langage naturel. Elle consiste, à partir des données brutes, à dégager les concepts et les relations entre ces concepts permettant de décrire de manière informelle les entités cognitives du domaine.

L'objectif est d'aboutir à un modèle conceptuel : le modèle obtenu consiste en un ensemble de termes désignant les entités du domaine de connaissances (concepts, relations, propriétés des concepts et des relations, …), assortis d'informations exprimant leur sémantique. La découverte des connaissances d'un domaine peut s'appuyer à la fois sur l'analyse de documents et sur l'interview d'experts du domaine. Ces activités doivent être raffinées au fur et à mesure que la conceptualisation émerge.

4.3 Ontologisation

L'ontologisation consiste en une formalisation partielle, sans perte d'information, du modèle conceptuel obtenu dans l'étape précédente. Ce qui permet de faciliter sa représentation ultérieure dans un langage complètement formel et opérationnel.

Elle effectue une transcription des connaissances dans un certain formalisme de connaissances, ce formalisme devant être aussi générique que possible, mais sémantiquement clair. S'imposer de conserver toutes les connaissances conduit à intégrer, à l'ontologie du domaine, des connaissances qui ne peuvent être formalisées, ou dont la sémantique est ambiguë.

Le modèle obtenu est souvent qualifié de semi-formel (car certaines connaissances ne peuvent pas être totalement formalisées). Le caractère semi-formel d'une ontologie lui interdit d'être utilisée telle quelle dans un SBC. En revanche, une ontologie, contenant toutes les connaissances d'un domaine, constitue le support idéal de communication et de partage des connaissances de ce domaine.

4.4 Opérationnalisation

Cette étape consiste à formaliser complètement l'ontologie obtenue dans un langage de représentation de connaissances formel (i.e. possédant une syntaxe et une sémantique) et opérationnel (i.e. doté de services inférentiels permettant de mettre en œuvre des raisonnements), par exemple, le modèle des Graphes Conceptuels ou la Logique de Descriptions.

On obtient alors une représentation formelle des connaissances du domaine. Ainsi, le caractère formel de l'ontologie permet à une machine, via cette ontologie, de manipuler des connaissances du domaine. La machine doit donc pouvoir utiliser des mécanismes opérant sur les représentations de l'ontologie.

5 Quelques méthodologies de construction d'ontologies

Les méthodologies peuvent porter sur l'ensemble du processus et guider l'ontologiste dans toutes les étapes de la construction. Bien qu'aucune méthodologie générale n'ait pour l'instant réussi à s'imposer, de nombreux critères de construction d'ontologies ont été proposés pour des méthodologies. ENTERPRISE, TOVE et METHONTOLOGY sont les méthodologies les plus représentatives pour construire des ontologies.

5.1 Tove

TOVE (Toronto Virtual Enterprise) développé par l'université de Toronto, cette méthodologie repose sur les expériences de développement d'une entreprise [39] [36]. Elle s'appuie également, pour le développement d'une ontologie, sur les principales étapes suivantes :

- Capturer des scénarios de motivations : Cette étape consiste à identifier des scénarios qui clarifient le domaine que l'on investit et les différentes applications dans lesquelles l'ontologie sera employée.

- Formuler des questions de compétences informelles : Cette étape consiste à formuler un ensemble de questions (basées sur les scénarios), exprimées en langage naturel, afin de déterminer la portée de l'ontologie. Ces questions et leurs réponses sont utilisées pour extraire les concepts principaux, leurs propriétés et les relations qui existent entre ces concepts.

- Spécifier la terminologie de l'ontologie : Cette étape consiste à représenter les termes (concepts, propriétés et relations), identifier dans l'étape précédente, en utilisant le formalisme de la logique du premier ordre. Les concepts seront représentés sous forme de constantes ou bien des variables. Par ailleurs, les propriétés et les relations seront représentées par des prédicats.

- Evaluer la complétude de l'ontologie.

5.2 Enterprise

Uschold et King's [36], proposent le squelette d'une méthode basé sur l'expérience de construction d'ontologies dans le domaine de la gestion des entreprises. La méthode ENTERPRISE repose sur les quatre étapes suivantes :

- Identifier le rôle et la portée de l'ontologie ,

- Dans cette étape, l'ontologie est réellement construite. Les activités suivantes sont distinguées : identifier les concepts et relations fondamentaux et des définitions provisoires de ces éléments, coder l'ontologie dans un langage adapté, intégrer des ontologies existantes ,

- Evaluer l'ontologie,

- Rédiger une documentation et une trace des actions réalisées lors des différentes phases .

Les étapes et sous-tâches de la méthode ENTERPRISE, sont décrites de façon abstraite. Les techniques utilisées pour les sous-tâches ne sont pas précisées (par exemple : Comment identifier les concepts fondamentaux ? Quel langage utiliser pour représenter l'ontologie ?

5.3 Methontology

La méthodologie de construction d'ontologies «METHONTOLOGY» se situe entre le GL (Génie Logiciel) et l'IC (Ingénierie des Connaissances). Elle identifie une séquence d'activités techniques à appliquer pour le développement de l'ontologie. Cette méthodologie a été motivée par le constat suivant : l'absence de méthodes ou de guides structurés est un obstacle a la construction d'ontologies partagées et consensuelles. Il est

également un obstacle à l'extension d'une ontologie existante ou à sa réutilisation dans d'autres ontologies. L'approche METHONTOLOGY distingue les étapes suivantes:

5.3.1 Spécification

Le développement d'une ontologie commence par la définition du domaine et de la porté de celle-ci. Cela est basé sur la réponse à certaines questions : Quel est le domaine que l'ontologie va couvrir ? À quoi cette ontologie va servir ? À quels types de questions les informations de l'ontologie doivent fournir des réponses ? Qui va utiliser et maintenir l'ontologie ?, etc. Les réponses à ces questions peuvent changer durant le processus de développement de l'ontologie, mais à chaque étape, elles permettent de limiter la portée du modèle. L'une des solutions qui permet de déterminer la portée d'une ontologie consiste à définir ou planifier une liste de questions auxquelles une base de connaissance, basée sur l'ontologie, doit être capable de répondre (« competency questions ») [39]. Ces questions peuvent servir à un test ultérieur de l'ontologie (Est-ce que l'ontologie contient des informations suffisantes pour répondre à ces questions ? Est-ce que les réponses nécessitent un certain niveau de détail ou la représentation d'un espace particulier ?), mais elles ne doivent pas être exhaustives.

5.3.2 Conceptualisation

Elle consiste à identifier et à structurer les connaissances du domaine, à partir des sources d'informations. L'acquisition de ces connaissances peut s'appuyer à la fois sur l'analyse de documents et sur l'interview des experts du domaine. Une fois que les concepts sont identifiés par leurs termes, leur sémantique est décrite dans un langage semi-formel (tables et graphes) à travers leurs propriétés, leurs instances connues et les relations qui les lient entre eux.

5.3.3 Implémentation

Cette étape consiste à formaliser le modèle conceptuel obtenu dans l'étape précédente par un formalisme de représentation d'ontologie telles que les logiques de description. Puis, à coder l'ontologie dans un langage d'ontologie formel.

5.3.4 Maintenance

Cela peut s'agir d'une maintenance corrective ou évolutive de l'ontologie (nouveaux besoins de l'utilisateur), ce qui permet la validation et l'évolution de celle-ci. Cette activité est généralement faite par le constructeur et des experts du domaine. La validation se base sur l'exploitation des services d'inférences associés aux LDs, et qui sont offerts par des raisonneurs.

Pour conclure, nous avons constaté que la démarche METHONTOLOGY présente un certain nombre de phases spécifiées de manières très détaillées, notamment la phase de conceptualisation. De ce fait, nous allons adopter cette méthodologie et l'adapter pour les besoins de notre travail.

6 Formalismes de représentation

Représenter des connaissances propres à un domaine particulier consiste à décrire et à coder les entités de ce domaine de manière à ce qu'une machine puisse les manipuler afin de raisonner. Comme alternative à la logique classique, l'IA a proposé divers formalismes de représentation : ceux qui ont été le plus utilisés pour représenter les ontologies sont :

- Les frames
- Les graphes conceptuels
- Et les logiques de description

6.1 Frames

Le modèle des Frames est un classique de l'Intelligence Artificielle, et a été initialement proposé comme langage de représentation d'ontologies par T. GRUBER. Le principe de ce modèle est de décomposer les connaissances en classes (ou frames) qui représentent les concepts du domaine. À un frame est rattaché un certain nombre d'attributs (slots), chaque attribut pouvant prendre ses valeurs parmi un ensemble de facettes (facets). Une autre façon de présenter ces attributs est de les considérer comme des relations binaires entre classes dont le premier argument est appelé domaine (domain) et la deuxième portée (range) [39].

Des instances des classes, correspondant à l'extension de chaque concept, peuvent être ajoutées, ainsi que des fonctions qui sont des types particuliers de relations liant un ensemble de classes à une valeur calculée à partir des valeurs des attributs des classes. La spécification de propriétés conceptuelles des attributs (ou relations) recourt à des formules de la logique du premier ordre.

6.2 Graphes conceptuels

Introduit par J. SOWA au début des années 80, le modèle des Graphes Conceptuels (GC) appartient à la famille des réseaux sémantiques. Les réseaux sémantiques modélisent les connaissances sous forme de graphes, les nœuds étant associés à des concepts et les arêtes à des relations. Le modèle des GCs se décompose en deux parties [40]:

- Une partie terminologique dédiée au vocabulaire conceptuel des connaissances à représenter, c'est-à-dire les types de concepts, les types de relations et les instances des types de concepts. Cette partie correspond à la représentation du modèle conceptuel mais intègre également des connaissances sur la hiérarchisation des types de concepts et de relations.
- Une partie assertionnelle dédiée à la représentation des assertions du domaine de connaissance étudié.

6.3 Logiques de descriptions

Les logiques de description (LDs) découlent directement des travaux fondateurs de Bachmann et de son système KL-ONE. Depuis le début des années 90, la recherche en logique de description s'est considérablement développée.

Les logiques de description peuvent être considérées comme un fragment de la logique du premier ordre, dans lequel les formules ont une variable libre pour les descriptions de concept et deux variables libres pour les descriptions de relations [41].

Une LD est composée de deux parties : un langage terminologique TBOX et un langage assertionnel ABOX. Le langage assertionnel est dédié à la description de faits et le langage terminologique à la description de concepts et de rôles. La principale tâche de raisonnement au niveau terminologique est de calculer les relations de subsomption entre concepts [46].

6.3.1 Les constructeurs des LDs

Les entités de base qui sont définies et manipulées dans une logique de descriptions sont les concepts et les rôles. Un concept permet de représenter un ensemble d'individus, et un rôle représente une relation binaire entre concepts.

Un concept et un rôle possèdent une description structurée, élaborée à partir d'un certain nombre de constructeurs, les concepts et les rôles peuvent être primitifs ou définis. Les concepts (éventuellement les rôles) primitifs sont comparables à des atomes et servent de base à la construction des concepts définis (éventuellement les rôles).

Il existe de nombreux constructeurs permettant de former toute une famille de logiques de description. Par ailleurs, la logique de description SHIQ [47] regroupe un ensemble plus riche de constructeurs. Le tableau (Tab. 1) décrit les plus importants.

6.3.2 Mécanisme de raisonnement

Le mécanisme de raisonnement de base pour les LDs est la classification de concepts dans la hiérarchie des concepts, réalisée par un algorithme de classification appelé classifieur [48]. Généralement, les LDs sont hautement expressives, disposent d'une sémantique claire et procurent des mécanismes d'inférences efficaces. Elles constituent, ainsi, le formalisme retenu par le Web sémantique pour la représentation des ontologies [49].

La sémantique d'un concept défini par les constructeurs des logiques de description est donnée par le tableau ci-dessous:

Constructeur	Syntaxe	Sémantique (I : une interprétation)
Universel	$\top$	ΔI (ΔI : ensemble de tous les objets)
Absurde	$\bot$	$\varnothing$
Négation	$\neg C$	$\Delta I \setminus CI$
Conjonction	$C \cap D$	$CI \cap DI$
Disjonction	$C \cup D$	$CI \cup DI$
Restriction universelle	$\forall r.C$	$\{x \in \Delta I \ / \ \forall y, (x,y) \in Ri \Rightarrow y \in CI\}$
Restriction existentielle	$\exists r.C$	$\{x \in \Delta I \ / \exists y, (x,y) \in rI \text{ et } y \in CI\}$
Cardinalité minimum	$(\geq n\ r)$	$\{x \in \Delta I \ / \ \lvert\{y \mid (x,y) \in rI\}\rvert \geq n\}$
Cardinalité maximum	$(\leq n\ r)$	$\{x \in \Delta I / \ \lvert\{y \mid (x,y) \in rI\}\rvert \leq n\}$
Conjonction des rôles	$r \cap s$	$\{(x,y) \in \Delta I x \Delta I / (x,y) \in rI \wedge (x,y) \in sI\}$
Disjonction des rôles	$r \cup s$	$\{(x,y) \in \Delta I x \Delta I / (x,y) \in rI \vee (x,y) \in sI\}$

Tab. 1:Les constructeurs essentiels d'une logique de description.

Pour conclure cette section, nous avons considérablement mis l'accent sur les logiques de descriptions et nous avons insisté sur la famille SHIQ qui propose un ensemble plus riche de constructeurs car notre choix du formalisme de représentation de l'ontologie d'application, va porter sur cette famille SHIQ. Ce choix dépendra, en plus de la richesse du formalisme de représentation, du langage de codification de l'ontologie choisi.

7 Outils de développement d'ontologies

Les outils de développement d'ontologies qui existent sur le marché aujourd'hui sont divers et variés à bien des égards. Cet état de choses suscite beaucoup d'interrogations lorsque vient le moment d'en choisir un pour construire une nouvelle ontologie [50] : L'outil offre-t-il une assistance au développement ? L'outil dispose-t-il d'un moteur d'inférence ? Quels langages d'ontologies l'outil supporte-t-il ? L'outil permet-il d'importer/exporter des ontologies ? L'outil offre t-il un support à la réutilisation d'ontologies existantes ? L'outil permet-il de documenter les ontologies construites ? L'outil offre-t-il un support graphique à la construction des ontologies ? L'outil est-il stable, convivial, « mature » ? Les réponses à toutes ces questions pourraient s'avérer décisives dans le choix de l'un ou l'autre outil. Dans cette section nous passons en revue les principaux outils disponibles.

7.1 Langages de spécification d'ontologies

Plusieurs langages de spécification d'ontologies (ou langages d'ontologies) ont été développés pendant les dernières années, et ils deviendront sûrement des langages d'ontologie dans le contexte du Web sémantique. Certains d'entre eux sont basés sur la syntaxe de XML, tels que XOL (Ontology Exchange Language), SHOE (Simple HTML Ontology Extension - qui a été précédemment basé sur le HTML), OML (Ontology Markup Language), RDF (Resource Description Framework), RDF Schéma. Les 2 derniers sont des langages créés par des groupes de travail du World Wide Web Consortium (W3C).

En conclusion, trois langages additionnels sont établis sur RDF(S) pour améliorer ses caractéristiques: OIL (Ontology Inference Layer), DAML+OIL et OWL (Web Ontology Language). La figure ci-dessous présente des langages de spécification d'ontologie, qui ont été récemment développés. La figure ci-dessous représente les rapports principaux entre tous ces langages sous la forme d'une pyramide des langages du Web sémantique.

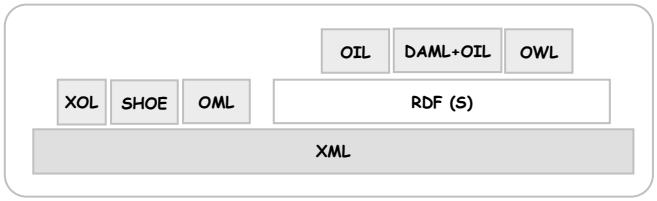


Fig. 4: La pyramide des langages basés Web.

7.1.1 RDF

RDF [51] est un langage d'assertion et d'annotations. Les assertions affirment l'existence de relations entre les objets. Elles sont donc adaptées à l'expression des annotations que l'on veut associer aux ressources du Web. RDF est un langage formel qui permet d'affirmer des relations entre des « ressources ».

Le modèle RDF définit trois types d'objets:

Ressources : les ressources sont tous les objets *décrits* par RDF. Généralement, ces ressources peuvent être aussi bien des pages Web que tout objet ou personne du monde réel. Les ressources sont alors identifiées par leur URI (Uniform Resource Identifier) ;

Propriétés : une propriété est un attribut, un aspect, une caractéristique qui s'applique à une ressource. Il peut également s'agir d'une mise en relation avec une autre ressource ;

Valeurs : les valeurs en question sont les valeurs particulières que prennent les propriétés.

Ces trois types d'objets peuvent être mis en relation par des **assertions**, c'est à dire des triplets (*ressource*, *propriété*, *valeur*), ou encore (*sujet*, *prédicat*, *objet*).

Exemple :

Mohamed est l'auteur de la bibliographie des stars à l'adresse http://www.site.fr/ BibDesStars.xml (ou la bibliographie des stars à l'adresse … a pour auteur Marcel Dugoumier)

Subject (Resource)	http://www.site.fr/ BibDesStars.xml
Predicate (Property)	auteur
Object (literal)	" Mohamed "

7.1.2 RDF(S)

Comme son nom l'indique, RDFS a pour but de définir des schémas de méta-données. Il définit le sens, les caractéristiques et les relations d'un ensemble de propriétés. La définition peut inclure des contraintes pour les valeurs potentielles et l'héritage des

propriétés d'autres schémas. Il est, en effet, une extension sémantique de RDF afin de fournir un mécanisme pour décrire les groupes associés de ressources et les relations entres les ressources [52].

L'intérêt de RDFS est qu'il facilite l'inférence sur les données et renforce la recherche sur ces données.

7.1.3 DAML-OIL

DAML [53] est un langage qui a comme but de fournir les fondations pour la génération suivante du Web sémantique. Le langage a adopté d'abord RDFS comme langage d'ontologie pour l'interopérabilité sémantique entre projets. Comme RDFS n'est pas assez expressif relativement aux exigences du Web sémantique, un nouveau langage nommé DAML-ONT a été développé en tant qu'extension de RDF avec les capacités d'un langage de représentation du savoir : orienté objet et basé sur cadre [54].

En même temps, un groupe des chercheurs (la plupart d'entre eux sont européens) au sein d'IST d'Union européenne, avec le même but, développe un autre langage d'ontologie appelé OIL. Ce langage a une syntaxe basée sur RDF et il est explicitement construit pour que sa sémantique puisse être spécifiée à travers une description logique très expressive, la logique de description de type SHIQ [54].

DAML+OIL est la combinaison de ces deux langages. Il hérite des avantages de ces deux langages. En conséquence, DAML+OIL est un langage très expressif et lisible par la machine ainsi que par un être humain avec une syntaxe basée sur RDF.

7.1.4 OWL

OWL signifie Web Ontology Language. Il est défini par le W3C dans [55], Le langage OWL est basé sur la recherche effectuée dans le domaine de la logique de description. OWL permet de décrire des ontologies, c'est-à-dire qu'il permet de définir des terminologies pour décrire des domaines concrets. Une terminologie se constitue de concepts et de propriétés (aussi appelés rôles en logiques de description). Un domaine se compose d'instance de concepts.

Le langage OWL se compose de trois sous-langages qui proposent une expressivité croissante, chacun conçu pour des communautés de développeurs et des utilisateurs spécifiques: OWL Lite, OWL DL, OWL Full. Chacun est une extension par rapport à son prédécesseur plus simple.

OWL Lite répond à des besoins de hiérarchie de classification et de fonctionnalités de contraintes simples de cardinalité 0 ou 1. Une cardinalité 0 ou 1 correspond à des relations fonctionnelles, par exemple, une personne a une adresse. Toutefois, cette personne peut avoir un ou plusieurs prénoms, OWL Lite ne suffit donc pas pour cette situation.

OWL DL concerne les utilisateurs qui souhaitent une expressivité maximum couplée à la complétude du calcul (cela signifie que toutes les inférences seront assurées d'être prises en compte) et la décidabilité du système de raisonnement (c'est- à -dire que tous les calculs seront terminés dans un intervalle de temps fini). Ce langage inclut toutes les structures

OWL avec certaines restrictions, comme la séparation des types: une classe ne peut pas aussi être un individu ou une propriété. Il est nommé DL car il correspond à la logique descriptive.

OWL Full se destine aux personnes souhaitant une expressivité maximale. Il a l'avantage de la compatibilité complète avec RDF/RDFS, mais l'inconvénient d'avoir un haut niveau de capacité de description, quitte à ne pas pouvoir garantir la complétude et la décidabilité des calculs liés à l'ontologie [55].

7.2 Editeurs d'ontologies

De nombreux éditeurs d'ontologies utilisant des formalismes variés et offrant différentes fonctionnalités ont été développés. Parmi ces outils on trouve : OILed, OntoEdit, , Web ode, DOE, PROTEGE-OWL… voici la description de quelques-uns :

7.2.1 Oiled

Oiled [56] a été conçu pour éditer des ontologies dans le langage de représentation OIL, il est souvent considéré comme une simple interface de la logique de description SHIQ. Cet éditeur offre également les services d'un raisonneur, FaCT qui permet de tester la satisfiabilité des définitions de classes et de découvrir des subsomptions restées implicites dans l'ontologie. L'outil dispose de mécanismes pour la classification et le contrôle de la cohérence des ontologies. La version 3.4 d'OILEd est gratuite et disponible sur le site web d'OILEd.

7.2.2 OntoEdit

OntoEdit [57] est également un environnement de construction d'ontologies indépendant de tout formalisme. Des outils graphiques dédiés à la visualisation d'ontologies sont inclus dans l'environnement. ONTOEDIT intègre un serveur destiné à l'édition d'une ontologie par plusieurs utilisateurs. Un contrôle de la cohérence de l'ontologie est assuré à travers la gestion des ordres d'édition.

7.2.3 Ontolingua

Ontolingua [58] de l'Université Stanford. Le serveur Ontolingua est le plus connu des environnements de construction d'ontologies en langage Ontolingua. Il consiste en un ensemble d'environnements et de services qui supportent la construction en coopération d'ontologies, entre des groupes séparés géographiquement. Il supporte plusieurs langages et dispose de traducteurs pe rmettant de passer de l'un à l'autre. Il est aussi doté d'une bibliothèque d'ontologies, accessible à distance ou localement via des éditeurs d'ontologies ou des applications.

7.2.4 ONTOSAURUS

ONTOSAURUS [59] de l'Information Science Institute de l'Université de Southern California. Ontosaurus consiste en un serveur utilisant LOOM comme langage de représentation des connaissances, et en un serveur de navigation créant dynamiquement des pages HTML qui affichent la hiérarchie de l'ontologie; le serveur utilise des formulaires

HTML pour permettre à l'usager d'éditer l'ontologie. Des traducteurs du LOOM en Ontolingua, KIF, KRSS et C++, ont été développés.

7.2.5 PROTÉGÉ-OWL

PROTEGE-OWL [26] est une interface modulaire, développée au Stanford Medical Informatics de l'Université de Stanford7, permettant l'édition, la visualisation, le contrôle (vérification des contraintes) d'ontologies, l'extraction d'ontologies à partir de sources textuelles, et la fusion semi-automatique d'ontologies. Le modèle de connaissances de PROTEGE-OWL est issu du modèle des frames et contient des classes (concepts), des slots (propriétés) et des facettes (valeurs des propriétés et contraintes), ainsi que des instances des classes et des propriétés. PROTEGE-OWL autorise la définition de méta-classes, dont les instances sont des classes, ce qui permet de créer son propre modèle de connaissances avant de bâtir une ontologie. De nombreux plug-ins sont disponibles ou peuvent être ajoutés par l'utilisateur.

L'interface, très bien conçue, et l'architecture logicielle permettant l'insertion de plug-ins pouvant apporter de nouvelles fonctionnalités (par exemple, la possibilité d'importer et d'exporter les ontologies construites dans divers langages opérationnels de représentation tels que OWL ou encore la spécification d'axiomes) ont participé au succès de PROTEGE-OWL, qui regroupe une communauté d'utilisateurs très importantes et constitue une référence pour beaucoup d'autres outils.

Grâce à toutes ces spécificités que dispose PROTEGE-OWL, notre choix portera sur cet outil pour faciliter l'implémentation de l'ontologie d'application.

7.3 Et bien d'autres…!

Autres que les outils d'implémentation et d'édition d'ontologies, nous pouvons trouvé aussi les outils de raisonnement tels que Racer, Fact,…etc. ainsi que des outils permettant de construire des applications basées sur les ontologies, ils fournissent également un environnement de programmation pour RDF, RDFS et OWL, ainsi qu'un moteur d'inférence basé sur les règles. Parmi ces outils nous pouvons citer Jena, KAON,…etc.

7.3.1 Racer

Le système Racer (Renamed ABox and Concept Expression Reasoner ou Raisonneur d'expression de concepts et de ABox renommées). Il se présente sous la forme d'un serveur qui peut être accédé par le protocole TCP ou HTTP. Considéré aujourd'hui comme le plus intéressant, c'est un système de représentation de connaissances, qui implémente un calcul de tableaux hautement optimisé, pour une logique de description très expressive. Il peut interpréter des documents OWL et offre des services de raisonnement aussi bien pour le niveau terminologique de l'ontologie, que pour le niveau assertionnel. Il permet aussi de vérifier la consistance et la subsomption des concepts et d'autres tests plus élaborés sur les instances, les rôles et les restrictions.

Bien que le système Racer soit plus facilement connecté à l'outil PROTEGE-OWL, donc, nous allons adopter ce système pour tester la consistance de l'ontologie d'application.

7.3.2 Jena

Jena est un cadre de travail Java open source permettant de construire des applications de Web sémantique. Elle fournit un environnement de programmation pour RDF, RDFS et OWL, ainsi qu'un moteur d'inférence basé sur les règles. Le cadre de travail inclut:

- API RDF;
- lecture et écriture RDF en RDF/XML, N3, et N-Triples;
- API OWL;
- stockage en mémoire et persistant;
- RDQL - un langage d'interrogation du RDF.

Le sous-système d'inférence de Jena est conçu pour permettre à certains moteurs d'inférence ou raisonneurs d'être connectés à Jena. Le terme inférence est utilisé pour se référer au processus abstrait de dérivation d'informations supplémentaires, et le terme raisonneur pour faire référence à un code objet spécifique qui effectue cette tâche.

Le développement des applications basées sur les ontologies est une tâche très lourde et qui nécessite, préalablement, des infrastructures ou bien des plateformes déjà construites et qui sont prêtes à l'emploi telle que Jena. De ce fait, nous allons opter la plateforme Jena pour développer les scénarios de communications entre les applications de l'entreprise.

8 Conclusion

Conçues comme réponse aux problèmes posés par l'intégration de connaissances au sein des systèmes informatiques, les ontologies apparaissent désormais comme une clé pour la manipulation automatique de l'information au niveau sémantique. Au fur et à mesure des recherches, des idées se dégagent autour du contenu des ontologies, des méthodes à utiliser pour les construire et des modèles et langages servant à leur représentation.

Au long de ce chapitre, nous avons essayé d'éclaircir la notion d'ontologie en présentant certaines définitions. Nous avons montré aussi les principaux formalismes de représentation de connaissances à savoir les frames, les graphes conceptuels et les logiques de descriptions. Nous avons découvert après les méthodologies les plus représentatives de leur construction et quelques domaines de leur utilisation. Et finalement, nous présenterons les outils nécessaires de leur développement à savoir les langages de représentation, les outils d'édition et d'interrogation, …etc.

Le chapitre suivant présente notre contribution au problème posé par ce mémoire, à savoir la construction d'une ontologie d'application dans le cadre de l'EAI. Pour cela, nous allons présenter une architecture basée sur les ontologies pour l'intégration d'applications d'entreprise proposée par [60]. Nous présenterons, aussi, notre processus de développement d'ontologies qui sera utilisé ultérieurement pour faciliter la construction de l'ontologie d'application qui est l'objectif de ce mémoire. Et finalement, un scénario de communication est proposé afin de montrer l'acheminement des requêtes entre les applications de l'entreprise.

Chapitre III

Construction d'une ontologie d'application dans le cadre de l'EAI

1 Introduction

Ce chapitre présente notre contribution au problème posé par ce mémoire, à savoir le développement d'ontologie d'application dans le cadre de l'EAI. Pour cela, nous présentons d'abord le travail proposé par [60] qui montre une architecture basée sur les ontologies afin d'intégrer les applications de l'entreprise. Cette architecture présente deux niveaux d'intégrations, niveau collaboratif et niveau applicatif, le niveau collaboratif, représente une intégration externe B2B, est doté d'une ontologie de processus métiers qui sert à supporter la collaboration entre les partners, par contre, le niveau applicatif, montre une intégration interne A2A, consiste à masquer l'hétérogénéité entre les applications internes de l'entreprise en développant pour chacune une ontologie baptisée l'«*ontologie d'application* ». Ce chapitre expose aussi le problème de l'ajout des ontologies au sein d'un système d'intégration. De ce fait, un scénario de communication entre les applications de l'entreprise est proposé. Le développement de ce scénario va nous aider, plus ou moins, durant la phase de construction de l'ontologie afin d'extraire, informellement, les concepts de l'ontologie d'application.

La construction de l'ontologie d'application, qui est l'objectif de ce mémoire, doit se baser sur un modèle cernant toutes les propriétés comportementales et structurelles dont doit disposer une application, ainsi que le domaine auquel l'application appartient. Pour y arriver, nous devons d'abord définir un processus de développement d'ontologie d'application définie par le langage OWL. Pour ce faire, nous nous sommes basés sur le travail de [61] qui présente une phase de spécification des besoins très détaillée. Le résultat de cette phase est résumé dans un document formalisé en langage RDF. La majorité des autres phases proposées se sont inspirées de la méthodologie METHONTOLOGY qui est le support de base pour la conceptualisation de l'ontologie à créer, à travers un ensemble de représentations intermédiaires semi-formelles. La logique de descriptions, est le formalisme adopté pour l'expression de l'ontologie semi-formelle, résultat de la phase de conceptualisation. Basé sur cette formalisation, OWL le langage de définition d'ontologies, est choisi afin de codifier l'ontologie en utilisant l'éditeur d'ontologies PROTEGE OWL. Finalement, le système d'inférences RACER est utilisé afin de tester la consistance de l'ontologie tout au long du processus de développement.

2 Architecture basée sur les ontologies pour l'intégration d'applications de l'entreprise

Dans un tel environnement, nous pouvons distinguer plusieurs types d'applications telles que les applications transactionnelles, batch, client/serveur, les applications Web…etc. Ces applications ont été, auparavant, développées par des langages de programmation différents et qui peuvent s'exécuter sur des plateformes de systèmes d'exploitation hétérogènes et utiliser divers formats de données pour s'échanger des messages. Tous ces aspects rendent le processus d'intégration des applications difficile et lourd, malgré l'existence de nombreux travaux [62] [63] [64] qui ont mené à répondre à ce problème.

Cependant, la majorité des solutions qui ont été proposées pour intégrer les applications souffrent du problème de la sémantique, c'est-à-dire que la communication entre les applications de l'entreprise n'est plus qu'un simple échange de messages dénués de toute

sémantique. De ce fait, nous voudrions que les applications de l'entreprise puissent se comprendre efficacement en mieux interprétant les messages envoyés et reçus entre elles. Ceci ne peut se faire hormis de donner une description concrète et formelle sur ce que c'est vraiment une application à travers ses dimensions, nous pouvons désigner ou bien qualifier la description concrète d'une application à une ontologie d'application.

Pour répondre à ce problème, le [60] proposa une architecture basée sur les ontologies afin d'intégrer les applications de l'entreprise, il a développé également pour chaque application faisant partie du système une ontologie baptisée l'ontologie d'application. Cette dernière a pour objectif de masquer l'hétérogénéité des applications et de donner une description matérielle sur ce que faisait réellement une application donnée. (Voir fig. 1)

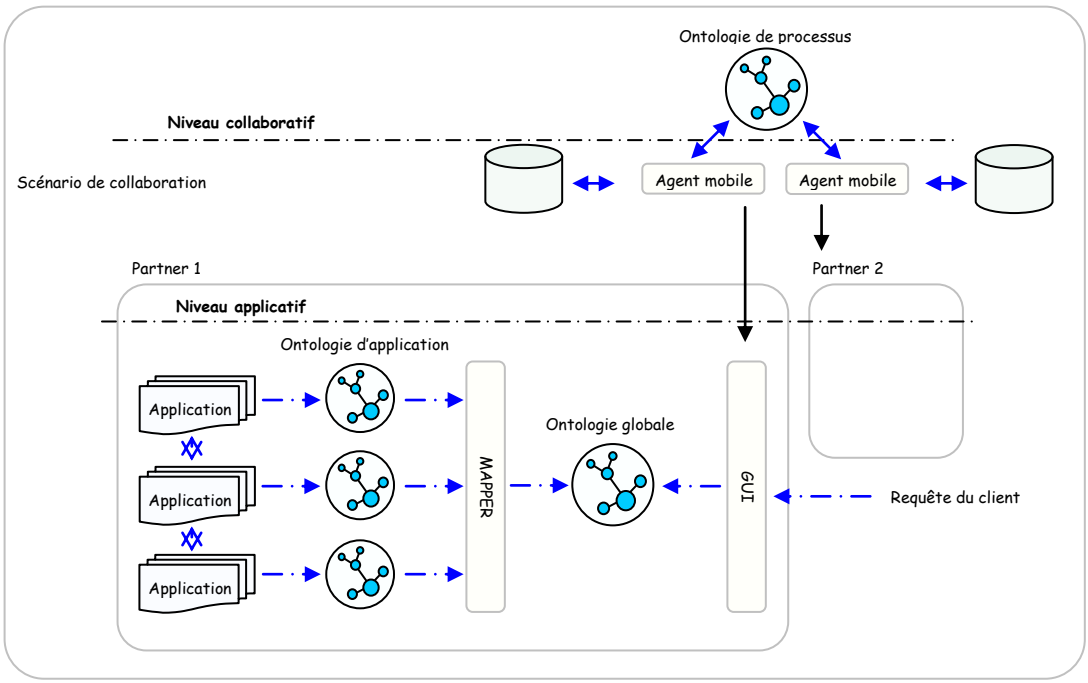


Fig 1: Une architecture basée sur les ontologies pour l'intégration d'applications de l'entreprise.

Nous tenons à décrire dans ce qui suit, les différents composants qui constituent l'architecture d'intégration et plus particulièrement les niveaux d'intégration d'applications ; niveau interne qui désigne le niveau applicatif et le niveau externe qui définit le niveau collaboratif. Nous verrons aussi les différentes sortes d'ontologies engagées dans l'architecture, leurs positions, leurs contenus et leurs intérêts vis-à-vis le système d'intégration. Enfin, nous décrirons brièvement le processus d'intégration à suivre pour parvenir à une intégration d'applications efficace et parfaite.

2.1 Niveaux d'intégration

D'après [60], il existe deux niveaux ou deux approches d'intégration dans l'entreprise, une intégration intra-entreprise et une intégration inter-entreprise. La première est représentée dans l'architecture par le niveau applicatif (A2A). Tandis que la deuxième est montrée par le niveau collaboratif (B2B).

Niveau collaboratif : une ontologie de processus métiers sert à supporter la collaboration entre les partenaires.

- Construction d'ontologie des processus par l'intégration des ontologies de services basés sur un scénario d'intégration.
- Utilisation du langage BPEL4WS pour spécifier l'enchaînement des services modélisant les processus métiers [65].
- Étendre le scénario de collaboration EbXML par des agents de recherches et des agents mobiles de négociation des transactions commerciales [66].

Niveau applicatif : L'hétérogénéité est gérée par le développement des ontologies d'applications.

- Construction d'ontologie d'application
- Proposition d'un processus d'intégration d'ontologies basées sur le framework MAFRA (MApping FRAmework) et le concept de semantic bridge [67].

2.2 Processus d'intégration d'applications de l'entreprise

Le processus d'intégration d'application basé sur des ontologies est une nouvelle approche qui a été rapidement développée avec l'apparition du Web sémantique [68]. C'est principalement dû au fait que l'ontologie est la meilleure manière d'augmenter l'interopérabilité et l'intégration des applications hétérogènes. [60] propose un processus, pour intégrer les applications, caractérisé par l'itération afin de prendre en compte l'ajout de nouvelles applications, l'évolution des exigences de l'entreprise et des critères du marché. Le processus proposé est divisé en quatre étapes de base :

- Analyse des besoins.
- Construction de l'ontologie d'application.
- Intégration des ontologies d'applications.
- Exploitation.

3 Vers un processus de construction d'une ontologie d'application

Nous avons proposé un processus de construction d'une ontologie d'application partant de connaissances brutes et arrivant à une ontologie d'application opérationnelle représentée par le langage OWL. Ce processus est composé de cinq étapes [69] :

- Spécification des besoins.
- Conceptualisation.
- Formalisation.
- Implémentation.
- Test & évolution de l'ontologie.

3.1 Spécification

Cette phase est inspirée du processus proposé de [61] qui consiste à établir un document formel de spécification des besoins représenté dans le langage RDF. Ce dernier permet de décrire l'ontologie à construire à travers les cinq aspects (i.e. propriétés) suivants :

1. **Le domaine de connaissance :** déterminer aussi précisément que possible le domaine que va couvrir l'ontologie.
2. **L'objectif :** le but de l'ontologie à créer pour le domaine considéré.
3. **Les utilisateurs :** identifier au maximum les futurs utilisateurs de l'ontologie à créer.
4. **Les sources d'informations :** déterminer les sources d'informations d'où les connaissances seront obtenues, par exemple, les experts du domaine, les documents techniques, etc., sont des sources de connaissance.
5. **La portée de l'ontologie :** déterminer à priori la liste des termes (les plus importants) pour le domaine à représenter [61].

3.2 Conceptualisation :

C'est l'étape la plus importante dans le processus de construction de l'ontologie. Elle est inspirée de la méthodologie METHONTOLOGY qui consiste à identifier et à structurer, à partir des sources d'informations, les connaissances du domaine. Elles permettent d'aboutir à un ensemble de représentations intermédiaires semi-formelles indépendamment des langages de formalisations à utiliser pour représenter l'ontologie. A la fin de cette phase, nous obtenons une ontologie conceptuelle.

Pour cela on distingue les principales tâches suivantes :

1. Construction du glossaire de termes.
2. Construction du diagramme de relations binaires et de classification des concepts
3. Construction du dictionnaire de concepts (DC).
4. Décrire les relations dans une table de relations binaires.
5. Spécifier des contraintes sur les attributs dans une table d'attributs.
6. Spécifier des axiomes sur les concepts dans une table d'axiomes logiques.
7. Décrire les instances des concepts dans une table d'instances.

La description de chaque étape est définie au fur et à mesure de la construction de l'ontologie d'application.

3.3 Formalisation

Cette phase consiste à formaliser l'ontologie conceptuelle obtenue dans la phase précédente afin de faciliter sa représentation ultérieure dans un langage complètement

formel et opérationnel. Notre choix est porté sur le formalisme de représentation de la logique de descriptions en s'appuyant sur sa syntaxe de type SHIQ qui présente une logique de description très expressive et qui offre un certain nombre de constructeurs pour décrire les concepts.

Syntaxe	Interprétation
T	Le concept le plus général (TOP)
⊥	Le concept le plus spécifique (Buttom)
C	Le nom d'un concept
R	Le nom d'un rôle (une relation binaire ou bien un attribut)
A	Le nom d'un individu
C1∩ *C2*	Conjonction de concepts pour définir de nouveaux concepts
C1∪ *C2*	Disjonction de concepts pour définir de nouveaux concepts
¬*C*	Utilisé pour définir le complément d'un concept
∀R.*C*	Définit le co-domaine du rôle R
∃ R.*C*	Il y a au moins un objet relié par le rôle R au concept *C*
(≥ n R.*C*) (≤ n R.*C*)	Cardinalité Minimum/Maximum (n est un nombre entier non négatif).
R^{-}	Le rôle inverse

Tab 1:Syntaxe du langage SHIQ

La logique de description est constituée de deux parties : une partie terminologique (TBOX) permettant de décrire les concepts et les rôles et d'une partie assertionnelle (ABOX) décrivant les instances.

3.3.1 Partie terminologique TBOX

La description des concepts et les rôles doit respecter la grammaire suivante:

< Concept >→ <concept-primitif> | <T> | <⊥> |
<Concept > ∩ < Concept > | < Concept > ∪ < Concept >|
¬< Concept >| ∀<rôle>.< Concept >| ∃ <rôle>. < Concept > |
≥ n <rôle>. < Concept> | ≤ n <rôle>. <concept> |
<rôle> → < rôle-primitif>
<rôle> → <rôle> ∩ <rôle>| <rôle> ∪ <rôle>

Ce qui montre dans la grammaire précédente que les concepts et les rôles sont exprimés de manière déclarative. Cette dernière comprend des définitions de concepts et de rôles ainsi que des inclusions de concepts.

Une définition de rôle est peut être définie aussi en fonction des concepts qui l'utilisent. Nous pouvons utiliser l'une des formes suivantes :

R :(C1, C2) où R est un nom de rôle, C1 est un concept source et C2 est un concept cible
R^{-} :=R1 où R1 est le rôle inverse pour R

3.3.2 Partie assertionnelle ABOX

Elle définit l'ensemble des axiomes utilisés pour décrire des situations concrètes. Pour chaque individu et pour chaque relation qui relie deux individus, nous décrivons les deux formes suivantes :

a : C où C est un concept défini et a est un individu.
(a1, a2) : R où R est un rôle défini et a1, a2 sont deux individus définis.

3.4 Implémentation

L'ontologie que nous avons obtenue dans la phase précédente est appelée une ontologie formelle. Le but de cette étape sera donc de coder l'ontologie formelle en OWL qui dispose de fonctionnalités sémantiques plus riches que ses prédécesseurs comme RDFS et DAML+OIL. A la fin ce cette phase, nous aurons une ontologie opérationnelle.

Afin de faciliter le processus de codification, nous utilisons PROTEGE OWL version 3.1.1 [26] disposant d'une interface modulaire, développée au Stanford Medical Informatics de l'Université de Stanford7, permettant l'édition, la visualisation, le contrôle (vérification des contraintes) d'ontologies, issu du modèle des frames et contient des classes (concepts), des slots (propriétés) et des facettes (valeurs des propriétés et contraintes), ainsi que des instances des classes et des propriétés.

3.5 Tests et évolution

Cette étape consiste à exploiter les services d'inférence fournis par la logique de description afin de supporter le processus de construction et d'améliorer la qualité de l'ontologie. Pour ce faire, nous proposons l'utilisation de l'outil RACER, un système de la logique de descriptions. Ce dernier, permet de lire un document au format OWL (ontologie OWL) et de le représenter sous forme d'une base de connaissances LD et de fournir des services d'inférence pour les niveaux TBOX et ABOX.

Cette étape sert aussi à suivre l'évolution de l'ontologie, c'est-à-dire les nouveaux concepts à ajouter dans la partie terminologique (TBOX) de l'ontologie. Une classification a lieu chaque fois qu'une définition de concept est nouvellement créée. Le mécanisme de raisonnement de base des logiques de description est la classification de concepts. Elle est réalisée par un algorithme de classification, appelé « le classifieur ». Le classifieur utilise la description d'un nouveau concept pour le placer à l'endroit correspondant dans la hiérarchie. Afin de trouver la place appropriée au nouveau concept, l'algorithme de classification détermine les relations de subsomption entre ce concept et les autres; Ces relations peuvent être spécifiées directement, trouvées par transitivité ou calculées à partir de la sémantique des conditions des rôles. La recherche de la place correcte pour le nouveau concept comporte trois étapes :

1. La recherche des subsumants les plus spécifiques SPS (concepts qui subsument le concept à classer et dont les fils ne le subsument pas)

2. La recherche des subsumés les plus généraux SPG (concepts subsumés par le concept à classer et dont les pères ne sont pas subsumés par lui).
3. Insertion du nouveau concept dans la hiérarchie.

4 Construction d'une ontologie d'application dans le cadre de l'EAI

Dans cette section, nous construisons notre ontologie d'application qui concerne le domaine de l'intégration d'applications de l'entreprise [69]. A cette fin, nous suivrons les étapes du processus de construction d'ontologie développé dans la section 3.

4.1 Spécification

Pour commencer le développement de l'ontologie, nous entamons d'abord la phase de spécification qui consiste à établir un document de spécification des besoins. Au sein de ce document, nous dériverons l'ontologie à construire à travers les cinq aspects suivants:

- **Le domaine de connaissance :** l'ontologie d'application, que nous venons de construire, s'inscrit dans le cadre de l'intégration d'applications de l'entreprise. En effet, elle peut prendre ses concepts depuis différents domaines variés car les applications de l'entreprise n'appartiennent pas forcément à un même domaine. Par exemple, on peut avoir des applications ayant été construites pour la réservation des vols tandis que d'autres sont dédiées à la location de voitures et ainsi de suite. Pour le besoin de notre travail, on se satisfait par le domaine de l'EAI qui résume communément les différents domaines dans lesquels les applications peuvent intervenir.
- **L'objectif :** comme il est montré dans l'architecture de [60], l'objectif majeur de l'incorporation des ontologies au sein d'un système d'intégration est de celer l'hétérogénéité entre les applications de l'entreprise afin de les intégrer ultérieurement.
- **Les utilisateurs :** cet aspect présente l'ensemble des utilisateurs pouvant exploiter l'ontologie. Dans notre cas, les utilisateurs de l'ontologie représentent les applications de l'entreprise qui doivent exploiter les ontologies afin d'atteindre l'objectif visé.
- **Les sources d'informations :** les sources d'informations sur lesquelles nous nous sommes basés pour arriver à la construction de l'ontologie d'applications étaient des documents techniques, la technologie des Web services et le développement des scénarios de communication entre les applications de l'entreprise. Nous venons, dans la suite de cette section, de citer les documents techniques desquels nous avons utilisés pour extraire les connaissances nécessaires à la construction de l'ontologie. La seule chose qui fait le point de l'étonnement est l'absence des experts du domaine qui représente la phase prépondérante pour dégager la majorité des concepts de l'ontologie d'application, pour cela, nous avons recouvert, ou comblé cette absence par le développement des scénarios de

communication entre les applications de l'entreprise afin de retirer les concepts de l'ontologie à créer.

- **La portée de l'ontologie :** Cet aspect consiste à déterminer à priori la liste des termes de l'ontologie d'application (les plus importants), parmi ces termes, nous pouvons citer : Application, Activité, Avoir-domaine, Réalise, Nom, Type, …etc.

Nous résumons cette phase dans un document RDF présenté dans la (Fig. 2). Il peut inclure aussi d'autres aspects tels que : la date de création de l'ontologie, ses créateurs, son niveau de formalité …etc.

```
<rdf:RDF>
…
<rdf: Description about=" URI of ontology" >
<Domaine> intégration d'application de l'entreprise (EAI) </Domaine >
<Date> 16 Octobre 2006 </ Date >
<Développé-par>
<rdf:Sequence>
<rdf:_1 H. Guergour, laboratoire LIRE, Université Mentouri de Constantine >
<rdf:_2 R. Driouche, laboratoire LIRE, Université Mentouri de Constantine >
<rdf:_3 Z. Boufaïda, laboratoire LIRE, Université Mentouri de Constantine >
</rdf:Sequence>
</ Développé-par >
<Objectif> une ontologie modélisant les propriétés comportementales et structurelles d'une application, ainsi des propriétés sur le domaine dans lequel l'application peut intervenir, l'objectif de la construction de cette ontologie est de faciliter l'intégration des diverses applications de l'entreprise.
</ Objectif >
<Niveau de formalité formel />
<Termes>
<rdf:Sequence>
<rdf:_1 Application> <rdf:_2 Application-comportement> <rdf:_3 Application-Structure >
<rdf:_4 Application-Domaine > <rdf:_5 Activité><rdf:_6 Activité-atomique >
<rdf:_7 Activité-complexe ><rdf:_8 Séquence-Activité > <rdf:_9 Activité-compétitive >
<rdf:_10 Choix-Activité > <rdf:_11 Activité-répéter-jusque-à > <rdf:_12 Application-modèle >
<rdf:_13 Input > <rdf:_14 Paramètre ><rdf:_15 Output > <rdf:_16 Précondition><rdf:_17 Effet >
<rdf:_18 Paramètre-calculé> <rdf:_19 Input-calculé > <rdf:_20 Output-calculé >
<rdf:_21 Précondition-calculée > <rdf:_22 Effet-calculé > <rdf:_23 IDL-description >
<rdf:_24 XDR-description > <rdf:_25 WSDL-description > <rdf:_26 Créateur >
<rdf:_27 Date ><rdf:_28 Interface-structure > <rdf:_29 Produit >
<rdf:_30 Méthode-structure > <rdf:_31 Transportation > <rdf:_32 Input-structure >
<rdf:_33 Output-structure > <rdf:_34 Niveau > <rdf:_35 Domaine-description >
<rdf:_36 description-fonctionnelle > <rdf:_37 description-non-fonctionnelle > …
</rdf:Sequence>
</Termes>
<Sources>
<rdf:Sequence>
<rdf:_1 " An ontology for semantic middleware: extending DAML-S beyond Web services">
<rdf:_2 "Enterprise Application Integration". Edition Addison-Wesley, Boston et al. 2003.>
<rdf:_3 Semantic Web Service Tutorial >
<rdf:_4 rdf:resource= " kmi.open.ac.uk/projects/dip/resources/hicss-39/HICSS06-slides.ppt ">
</rdf:Sequence>
</Sources>
</rdf:description>
</rdf:RDF>
```

Fig 2: Un document RDF de spécification de l'ontologie d'application.

4.2 Conceptualisation

Une fois que la majorité des connaissances est acquise, on doit les organiser et les structurer en utilisant des représentations intermédiaires semi-formelles qui sont faciles à comprendre et indépendantes de tout langage d'implémentation. Cette phase contient plusieurs étapes qui sont :

- Construction du glossaire de termes
- Construction du diagramme de relations binaires et de classification de concepts
- Dictionnaire de concepts
- Tableaux des relations binaires
- Tableaux des attributs
- Tableaux des axiomes logiques
- Tableaux des instances

4.2.1 Construction de glossaire de termes

Ce glossaire contient la définition de tous les termes relatifs au domaine (concepts, instances, attributs, relations) qui seront représentés dans l'ontologie finale, par exemple, dans notre cas les termes *Activité* et *E-commerce* sont des concepts, *Ensemble-de* et *Précondition* représentent des relations,…etc.

Le tableau ci-dessous (Tab 2) fournit une liste détaillée des différents termes utilisés dans l'ontologie :

Nom du terme	Description
Application	Une entité, un programme ou bien un ensemble d'activités ayant un certain comportement, une structure et un domaine auquel elle appartient.
Application-comportement	Une sous ontologie modélisant les propriétés comportementales d'une application. caractérisée par un ensemble d'activités et un modèle d'exécution de ces activités.
Application-structure	Une sous ontologie modélisant les propriétés structurelles d'une application. elle permet de spécifier le lien ou bien l'interface entre l'application et le middleware.
Application-domaine	Une sous ontologie modélisant les propriétés sur le domaine auquel l'application appartient.
Activité	Une fonction, une procédure, un service ou bien une entité permettant d'accomplir une tâche particulière.
Activité-atomique	Une activité dont nous ne pouvons pas en extraire d'autres.
Activité-complexe	Une activité dont nous pouvons en extraire d'autres activités.
Input	Un argument ou une donnée qui doit être affectée à une activité.
Output	Représente le résultat de l'exécution d'une activité.
Précondition	Représente l'ensemble des conditions nécessaires pour exécuter une activité.
Effet	Représente les effets produits après l'exécution d'une activité. Un effet peut empêcher l'exécution d'autres activités.
IDL-Structure	Affirme que l'interface d'une application est spécifiée par le langage IDL de CORBA, ce qui fait que l'application est connectée au middleware CORBA.
WSDL-Structure	Affirme que l'interface d'une application est spécifiée par le langage WSDL, ce qui fait que l'application représente un Service Web.
Description-fonctionnelle	Ce terme permet de déterminer le domaine auquel l'activité ou bien l'application appartient.
Description-non-fonctionnelle	Ce terme permet de déterminer le domaine auquel le paramètre de l'activité appartient.

(…)	(…)
produit	Affirme que n'importe quelle application doit avoir un certain comportement.
Avoir-domaine	Affirme que n'importe quelle application doit appartenir à un certain domaine.
Avoir-structure	Affirme que n'importe quelle application doit être connectée à un middleware.
Avoir-input	Permet d'indiquer qu'une activité doit avoir un input en entrée.
Avoir-output	Permet d'indiquer qu'une activité présente un résultat à la fin de son exécution.
Description-type	Permet d'indiquer qu'un domaine peut avoir une description fonctionnelle et une description non fonctionnelle. Exp. Service-voyage est une description fonctionnelle Personne est une description on fonctionnelle.
Mappe-méthode	Affirme que le nom de la méthode dans l'interface du middleware doit correspondre au nom de l'activité.
Se-réfère-à 1	Permet d'indiquer que la description fonctionnelle d'un terme doit se référer à une activité. Exp. Service-vente est une instance de Vente-livres ACTIVITÉ_VENT01 est une instance de Activité Service-vente *se-réfère-à* ACTIVITÉ_VENT01. Donc : ACTIVITÉ_VENT01 est activité de Vente-livres.
Se-réfère-à 2	Permet d'indiquer que la description non fonctionnelle d'un terme doit se référer à un paramètre. Exp. Hôtel-panoramique est une instance de Hôtel. NOM_HOTL est une instance de paramètre. Hôtel-panoramique *se-réfère-à* NOM_HOTL Donc : NOM_HOTL est un paramètre de Hôtel.
(…)	(…)
Nom	Déterminer le nom d'une application, ou d'une activité,…
Type	Déterminer le type d'un paramètre.
Code	Représente le code d'un produit.
(…)	(…)

Tab 2:Glossaire de termes.

4.2.2 Construction du diagramme de relations binaires et de classification de concepts

Dans cette étape, nous construisons le diagramme de relations binaires et de classification de concepts en deux étapes principales. Initialement, nous répertorions les concepts en ensemble d'organisations, ensuite nous relions les concepts entre eux, si nécessaire, par des relations.

La hiérarchie de classification de concepts démontre l'organisation des concepts de l'ontologie en un ordre hiérarchique qui exprime les relations sous classe – super classe.

En utilisant la relation « Sous classe de » entre les classes pour définir leurs classifications, la classe C1 est une sous classe de la classe C2 si et seulement si toute instance de la classe C1 est une instance de la classe C2, par exemple la classe Activité est une sous classe de la classe Application parce que toute activité est une application; nous suivons un procédé de développement de haut en bas en commençant par une définition des concepts les plus généraux du domaine et en poursuivant par la spécialisation des concepts. Par exemple nous pouvons commencer en créant des classes pour les concepts généraux : Application, Application-comportement, Application-structure. Application-domaine, Application-Modèle, Domaine-Description, Description-fonctionnelle, Description-non-fonctionnelle, Paramètre, Paramètre-calculé, Interface-structure, Méthode-Structure, Input-Structure, Output-Structure, Date, Produit, Transportation, Niveau et Créateur.

Nous définissons donc les taxinomies principales du domaine, dont chacune est considérée comme une ontologie séparée, nous avons ainsi, comme il est montré dans la figure ci-dessous (fig. 3), de taxinomies relatives aux concepts généraux mentionnés précédemment.

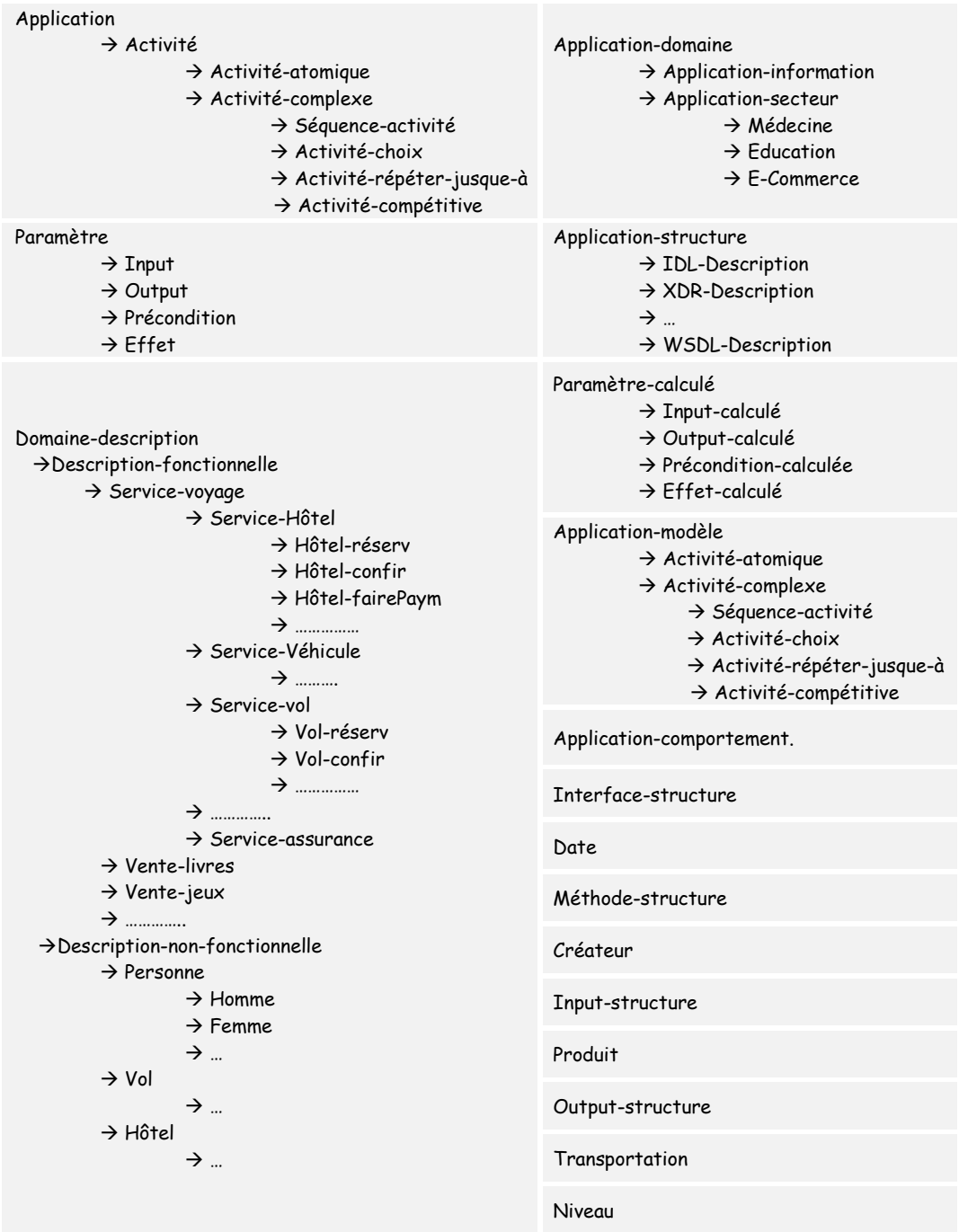


Fig 3:Arbres de classification de concepts.

Dans la deuxième étape, nous représentons les relations binaires et la hiérarchie entre les classes par un diagramme. Dans ce diagramme, les classes sont représentées par des rectangles et les relations par des arcs orientés (du domaine vers le co-domaine) et étiquetés par le nom de la relation, ainsi la relation « classe-de » est représentée par un arc pointillée (du classe-fille vers classe mère). La figure ci-dessous (Fig. 4) représente le diagramme de relations binaires et de classification de concepts de notre ontologie.

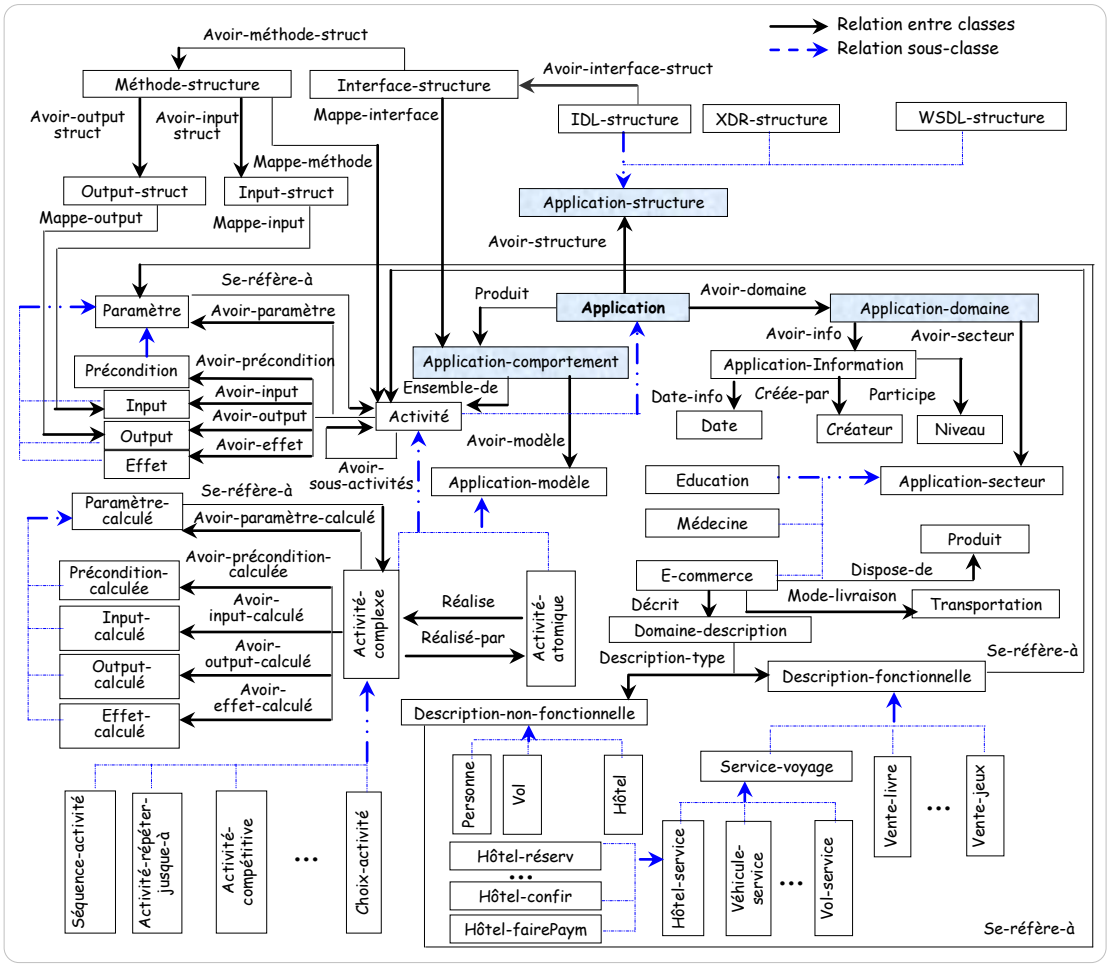


Fig 4:Construction du diagramme de relations binaires et de classification de concepts.

Maintenant, pour chaque arbre de classification de concepts nous construisons les représentations intermédiaires suivantes :

- Dictionnaire de concepts
- Tableaux des relations binaires
- Tableaux des attributs
- Tableaux des axiomes logiques
- Tableaux des instances

4.2.3 Dictionnaire de concepts

Dans cette étape nous allons pourvoir une description formelle des concepts qui ont été présentés dans la hiérarchie des classes, ce processus correspond à la création du dictionnaire de concepts accordé au METHONTOLOGY. Dans ce dictionnaire, nous définissons pour chaque concept : les instances, les attributs, les relations dont la source est ce concept, les synonymes et les acronymes de ce concept; les tableaux Tab 3, Tab 4, Tab 5, Tab 6, Tab 7, Tab 8, Tab 9, Tab 10 et Tab 11 représentent les dictionnaires des concepts pour les ontologies définies par les arbres précédents.

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	relations
Application	Programme ; logiciel ; …	App	-	Nom Texte-description	Produit Avoir-structure Avoir-domaine
Activité	Fonction; Service; Action; …	Act	-	-(attributs hérités)	Avoir-paramètre Avoir-précondition Avoir-input Avoir-output Avoir-effet Avoir-sous-activités
Activité-atomique	Fonction-atomique; Service-atomique; Action-atomique;…..	Act-atom		-	Réalise
Activité-complexe	Fonction-complexe; Service-complexe; Action-complexe; …	Act-cplx	-	-	Avoir-paramètre-calculé Avoir-précondition-calculée Avoir-input-calculé Avoir-output-calculé Avoir-effet-calculé Réalisée-par
Activité-compétitive	Fonction-compétitive; Service-compétitif; Action-compétitive; …	Act-cptv	-	-	Est-un-modèle-de
Séquence-activité	Séquence fonction; Séquence-service; Séquence-action; …	Seq-act	-	-	Est-un-modèle-de
Activité-choix	Fonction-choix-; Service-choix; Action-choix; …	Chx-act	-	-	Est-un-modèle-de
Activité-répéter-jusque-à	Fonction-répéter-jusque-à; Service-répéter-jusque-à; Action-répéter-jusque-à; …	Act-rpt-jusq	-	-	Est-un-modèle-de

Tab 3 : Dictionnaire des concepts dans le domaine "Application".

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	relations
Application-domaine	Programme-domaine; Programme-secteur; …	-	-	Nom Texte-description	Avoir-info Avoir-secteur
Application-information	Programme-information; Programme-information; …	-	-	-	Participe Créée-par Date-info
Application-secteur	Programme-domaine; Programme-area; …	-	-	-	-
Médecine	Thérapie; Traitement-médical;	-	-	(…)	(…)
Education	Enseignement; Formation; …	-	-	(…)	(…)
E-commerce	e-business; …	-	-	-	Décrit Mode-livraison Dispose-de

Tab 4 : Dictionnaire des concepts dans le domaine "Application-domaine"

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	relations
Application-modèle	Application-forme ; Application-disposition; …	-	-	Nom Texte-description	-
Activité-atomique	Fonction-atomique; Service-atomique; Action-atomique; …	Act-atom	-	-	Réalise
Activité-complexe	Fonction-complexe; Service-complexe; Action-complexe; …	Act-cplx	-	-	Avoir-paramètre-calculé Avoir-précondition-calculée Avoir-input-calculé Avoir-output-calculé Avoir-effet-calculé Réalisée-par S'exécute-comme
Activité-compétitive	Fonction-compétitive; Service-compétitif; Action-compétitive; …	Act-cptv	-	-	Est-un-modèle-de
Séquence-activité	Séquence fonction; Séquence-service; Séquence-action; …	Seq-act	-	-	Est-un-modèle-de
Activité-choix	Fonction-choix-; Service-choix; Action-choix; …	Chx-act	-	-	Est-un-modèle-de
Activité-répéter-jusque-à	Fonction-répéter-jusque-à; Service-répéter-jusque-à; Action-répéter-jusque-à; …	Act-rpt-jusq	-	-	Est-un-modèle-de

Tab 5: Dictionnaire des concepts dans le domaine "Application-modèle"

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	Relations
Paramètre (Paramètre-calculé)	Argument; … (Argument-calculé; …)	-	-	Nom Type Texte-description	Se-réfère-à
Input (Input-calculé)	Data-input; … (Data-input-calculé; …)	-	-	-	-
Output (Output-calculé)	Data-output; Résultat; … (Résultat-calculé; …)	-	-	-	-
Précondition (Précondition-calculée)	exigence; condition; … (exigence-calculé; Condition-calculée;…)	-	-	-	-
Effet (Effet-calculé)	Conséquence; … (Consequence-calculée;…)	-	-	-	-

Tab 6 : Dictionnaire des concepts dans le domaine "Application-modèle"

Nom du concept	Synonymes	Acro-nymes	Instances	Attributs	relations
Application-structure	Application-composition ; Application-construction ; Application-constitution ; …	-	-	Nom Texte-description	Avoir-structure
IDL-description	IDL- représentation;	-	-	-	Avoir-interface-struct
XDR-description	XDR-représentation; …	-	-	(…)	(…)
WSDL-description	WSDL-représentation; …	-	-	(…)	(…)

Tab 7: Dictionnaire des concepts dans le domaine "Application-structure"

Nom du concept	Synonymes	Acro-nymes	Instances	Attributs	relations
Domaine-description	Domain-représentation;…..	Dom-des	-	Nom Texte-description	Description-type
Description-fonctionnelle	Représentation-fonctionnelle; …	Des-fct	-	-	Se-réfère-à
Service-voyage	Service-expédition; Service-exploration; …	Svc-voyge	-	(…)	(…)
Hôtel-service	hôtellerie-service; palace-service; …	Svc-htl	-	(…)	(…)
Hôtel-réserv	hôtellerie-réservation; palace-réservation;….	Reser-htl	-	(…)	(…)
Hôtel-confir	hôtellerie-confirmation; palace-confirmation; …	Confir-htl	-	(…)	(…)
Hôtel-fairePaym	hôtellerie-payement; palace-payement; …	Fairepym-htl	-	(…)	(…)
Vehicule-service	Service-voitures; Service-transport; …	Svc-veh	-	(…)	(…)
Vol-service	Air-travel-service; …	Svc-vol	-	(…)	(…)
Vol-réservation	Air-travel – réservation;..	Reser-vol	-	(…)	(…)
Vol-confirmation	Air-travel-confirmation; …	Confir-vol	-	(…)	(…)
Assurance-service	assurance-service; …	Svc-ss	-	(…)	(…)
Vente-livres	Edition-livres; …	Vte-lv	-	(…)	(…)
Vente-jeux	Mise-en-vente-jeux; …	Vte-jx	-	(…)	(…)
Description-non-fonctionnelle	Representation-non-fonctionnelle ; …	Des-n-fct	-	-	Se-réfère-à
Personne	Individu; Humain; …	Pers	-	(…)	(…)
Homme	Male ; …	Hom	-	(…)	(…)
Femme	Female; …	Fem	-	(…)	(…)
Hôtel	hôtellerie; …	Htl	-	(…)	(…)
Vol	Air-travel; …	Vol	-	(…)	(…)

Tab 8 : Dictionnaire des concepts dans le domaine "Domaine-description"

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	relations
Créateur	Concepteur ; …	-	-	Nom Téléphone e-mail Fax Site-web @-physique	-

Tab 9: Dictionnaire des concepts dans le domaine "Créateur"

Nom du concept	Synonymes	Acro-nymes	Instances	Attributs	relations
Application-comportement	Application-agissement	App-cmprt	-	Nom Texte-description	Ensemble-de Avoir-modèle
Interface-structure	Interface-construction; Interface-composition; …	-	-	Nom Texte-description	Avoir-interface-structure Mappe-interface
Méthode-structure	Interface-construction; …	-	-	Nom Retourner-type Texte-description	Avoir-input-structure Avoir-output-structure Mappe-méthode
Input-structure	Input-construction; …	-	-	Nom Type Texte-description	Mappe-input
Output-structure	Output-construction; …	-	-	Nom Type Texte-description	Mappe-output
Transportation	Transport; …	-	-	Nom Texte-description	-
Produit	Marchandise; …	-	-	Nom Code Classification Texte-description	-
Niveau	Degré; point; …	-	-	Liste Texte-description	-

Tab 10: Dictionnaire des concepts dans les domaines : "Application-comportement", "Interface-structure", "Méthode-structure", "Input-structure", "Output-structure", "Transportation", "Produit" & "Niveau".

Nom du concept	Synonymes	Acro-nymes	Ins-tances	Attributs	Relations
Date	Période; Moment; …	-	-	Année Mois Jour	-

Tab 11: Dictionnaire des concepts dans le domaine "Date".

4.2.4 Tableaux de relations binaires

Les relations binaires sont représentées sous forme de propriétés ou attributs qui lient un concept à un autre, ce sont des 'attributs de type instance' : c'est-à-dire les attributs ayant pour type de valeur Instance [71].

Pour chaque relation dont la source est dans l'arbre de classification de concepts, nous définissons : son nom, le nom du concept source, le nom du concept cible, la cardinalité et le nom de la relation inverse; les tableaux Tab 12, Tab 13, Tab 14, Tab 15, Tab 16, Tab 17, Tab 18, Tab 19 et Tab 20 illustrent la spécification des relations binaires dans les différentes taxonomies pour notre ontologie.

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Produit	Application	Application-comportement	(1,1)	Produit-par
Avoir-structure	Application	Application-structure	(1,1)	Stucture-de
Avoir-domaine	Application	Application-domaine	(1,1)	-
Avoir-paramètre	Activité	Paramètre	(0,n)	-
Avoir-précondition	Activité	Précondition	(0,n)	-
Avoir-input	Activité	Input	(0,n)	-
Avoir-output	Activité	Output	(1,n)	-
Avoir-effet	Activité	Effet	(0,n)	
Avoir-sous-activités	Activité	Activité	(0,n)	Avoir-sous-activités
Réalise	Activité-atomique	Activité-complexe	(1,n)	Réalisée-par
Réalisée-par	Activité-complexe	Activité-atomique	(1,n)	Réalise
Avoir-paramètre-calculé	Activité-complexe	Paramètre-calculé	(0,n)	-
Avoir-input-calculé	Activité-complexe	Input-calculé	(0,n)	-
Avoir-output-calculé	Activité-complexe	Output-calculé	(1,n)	-
Avoir-effet-calculé	Activité-complexe	Effet-calculé	(1,n)	-
Est-un-modèle-de	Séquence-activité Activité-compétitive Activité-choix Activité-répéter-jusque-à	Activité-complexe	(1,1)	S'exécute-comme

Tab 12: Relations binaires dans le domaine "Application".

Nom de la relation	Concept source	Concept cible	Cardina-lité	Relation inverse
Avoir-info	Application-domaine	Application-information	(1,1)	-
Avoir-secteur	Application-domaine	Application-secteur	(1,1)	-
Participe	Application-information	Niveau	(1,1)	-
Créée-par	Application-information	Créateur	(1,1)	Crée
Date-info	Application-information	Date	(1,1)	-
Décrit	E-commerce	Domaine-description	(1,1)	-
Dispose-de	E-commerce	Produit	(1,n)	-
Mode-livraison	E-commerce	Transportation	(1,1)	-

Tab 13: Relations binaires dans le domaine "Application-domaine ".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
S'exécute-comme	Application-modèle	Activité-atomique Activité-complexe	(1,1)	-
Réalise	Activité-atomique	Activité-complexe	(1,n)	Réalisée-par
Réalisée-par	Activité-complexe	Activité-atomique	(1,n)	Réalise
Avoir-input-calculé	Activité-complexe	Input-calculé	(0,n)	-
Avoir-output-calculé	Activité-complexe	Output-calculé	(1,n)	-
Avoir-effet-calculé	Activité-complexe	Effet-calculé	(1,n)	-

Tab 14: Relations binaires dans le domaine "Application-modèle".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Se-réfère-à	Paramètre	Activité	(1,n)	Avoir-paramètre
Avoir-paramètre	Activité	Paramètre	(1,n)	Se-réfère-à
Se-réfère-à	Paramètre-calculé	Activité-complexe	(1,1)	Paramètre-calculé
Paramètre-calculé	Activité-complexe	Paramètre-calculé	(1,n)	Se-réfère-à

Tab 15: Relations binaires dans les domaines "Paramètre" & "Paramètre-calculé".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Avoir-structure	Application-structure	IDL-structure XDR-structure WSDL-structure	(1,1)	-
Avoir-interface-struct	IDL-structure	Interface-structure	(1,n)	-
(…)	XDR-structure	(…)	(…)	(…)
(…)	WSDL-structure	(…)	(…)	(…)

Tab 16: Relations binaires dans le domaine "Application-structure".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Mappe-interface	Interface-structure	Application-comportement	(1,1)	-
Avoir-méthode-struct	Interface-structure	Méthode-structure	(1,n)	-
Mappe-méthode	Méthode-structure	Activité	(1,1)	-
Avoir-input-struct	Méthode-structure	Input-structure	(0,n)	-
Avoir-output-struct	Méthode-structure	Output-structure	(1,n)	-
Mappe-input	Input-struct	Input	(1,1)	-
Mappe-output	Output-structure	Output	(1,1)	-

Tab 17: Relations binaires dans les domaines : "Interface-structure", "Méthode-structure", "Input-structure" & "Output-structure".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Crée	Créateur	Application-information	(1,n)	Créée-par
Produit-par	Application-comportement	Application	(1,1)	produit

Tab 18: Relations binaires dans le domaine "Créateur" & "Application-comportement".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
-	-	-	-	-
(…)	(…)	(…)	(…)	(…)

Tab 19: Relations binaires dans les domaines "Date", "Transportation", "Niveau" & "Produit".

Nom de la relation	Concept source	Concept cible	Cardinalité	Relation inverse
Description-type	Domaine-description	Description-fonctionnelle Description-non-fonctionnelle	(1,1)	-
Se-réfère-à	Description-fonctionnelle	Activité	(1,n)	-
Se-réfère-à	Description-non-fonctionnelle	Paramètre	(1,n)	-

Tab 20: Relations binaires dans le domaine "Domaine-description".

4.2.5 Tableaux des attributs

Les attributs sont des propriétés qui prennent ses valeurs dans les types prédéfinis (String, Integer, Boolean, Date…). Par exemple le concept *Paramètre* a comme attributs : Nom, Texte-description, et le type du paramètre.

Pour chaque attribut apparaissant dans le dictionnaire de concepts nous spécifions : son nom, type et intervalle de ses valeurs possibles, et sa cardinalité (pour spécifier qu'une instance possède une ou plusieurs valeurs). Les tableaux Tab 21, Tab 22, Tab 23, Tab 24 et Tab 25 spécifient ces informations pour chaque attribut.

Nom de l'attribut	Type des valeurs	Range des valeurs	Cardinalité
Nom	String	-	(1,1)
Texte-description	String	-	(1,n)
(…)	(…)	(…)	(…)

Tab 21: Les attributs dans les domaines "Application", "Application-modèle", "Application-domaine", "Application-structure", "Application-comportement", "Domaine-description" & "Transportation".

Nom de l'attribut	Type des valeurs	Range des valeurs	Cardinalité
Nom	String	-	(1,1)
Texte-description	String	-	(1,n)
Type	Thing	-	(1,1)

Tab 22: Les attributs dans les domaines "Paramètre" & "Paramètre-calculé".

Nom de l'attribut	Type des valeurs	Range des valeurs	Cardinalité
Retourner-type	Thing	-	(1,1)
Nom	String	-	(1,1)
Texte-description	String	-	(0,n)
Code	String	-	(1,1)
Classification	String	-	(1,1)

Tab 23: Les attributs dans le domaine "Interface-structure", "Méthode-structure", "Input-structure", "Output-structure", "Produit" & "Niveau".

Nom de l'attribut	Type des valeurs	Range des valeurs	Cardinalité
Nom	String	-	(1,1)
Téléphone	String	-	(1,n)
e-mail	String	-	(1,1)
Fax	String	-	(1,n)
Site-web	String	-	(1,1)
@-physique	String	-	(1,1)

Tab 24: Les attributs dans le domaine "Créateur".

Nom de l'attribut	Type des valeurs	Range des valeurs	Cardinalité
Année	Integer	[1970…2020]	(1,1)
Mois	String	{Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre, Octobre, Novembre, Décembre}	(0,1)
Jour	Integer	[01…31]	(0,1)

Tab 25: Les attributs dans le domaine "Date".

4.2.6 Tableau des axiomes logiques

Ces tableaux contiennent des définitions de concepts à l'aide des expressions logiques qui sont toujours vraies. Dans ce tableau nous définissons pour chaque axiome sa description en langage naturel, le nom du concept auquel l'axiome se réfère, les attributs utilisés dans l'axiome et l'expression logique. Pour notre ontologie nous spécifions quelques axiomes comme il est représenté dans le Tab 26.

Nom de concept	Description de l'axiome	Attributs utilisés	Expression logique
Application	Une application doit avoir un domaine	Avoir-domaine	$\forall$ X Application (X) $\exists$ Y Application-domaine(Y) Avoir-domaine(X, Y).
Activité	Une activité doit fournir à la fin de son exécution un résultat.	Avoir-output	$\forall$ X Activité(X) $\exists$ Y Output (Y) Avoir-output(X,Y).
Application-information	Une application peut interagir soit au niveau interne ou externe de l'entreprise.	Participe	$\forall$ X Application-information(X) $\exists$ Y Niveau(Y) participe(X,Y).
Paramètre	Le nom de paramètre d'une activité se réfère à une activité.	Se-réfère-à	$\forall$ X Paramètre(X) $\exists$ Y Activité (Y) Se-réfère-à (X, Y).
Activité-complexe	Une activité complexe doit être une séquence d'activités ou un ensemble d'activités compétitives ou une liste à choix d'activités ou une ensemble d'activités à répéter jusqu'à une condition déterminée.	-	$\forall$ X Activité-complexe (X) $\Rightarrow$ (Séquence-activité (X) $\vee$ Activité-compétitive(X) $\vee$ Choix-activité (X) $\vee$ Activité-répéter-jusque-à(X))
IDL-structure	Une application doit avoir une interface avec un middleware	Avoir-interface-struct	$\forall$ X IDL-structure (X) $\exists$ Y Interface-structure (Y) / Avoir-interface-struct (X, Y).
Créateur	Une application doit être créée par une entreprise.	Crée	$\forall$ X Créateur (X) $\exists$ Y Application-information (Y) / Crée (X, Y).
Domaine-description	Un domaine est spécifié par des propriétés fonctionnelles et non fonctionnelles.		$\forall$ X Domaine-description (X) $\Rightarrow$ (Description-fonctionnelle (X) $\wedge$ Description-non-fonctionnelle (X))

Description-fonctionnelle	Une entité fonctionnelle doit se référer à une activité.	Se-réfère-à	∀ X Description-fonctionnelle (X) ∃ Y Activité (Y) Se-réfère-à (X, Y).
Description-non-fonctionnelle	Une entité non fonctionnelle doit se référer à un paramètre.	Se-réfère-à	∀ X Description-non-fonctionnelle (X) ∃ Y paramètre (Y) Se-réfère-à (X, Y).
(…)	(…)	(…)	(…)

Tab 26: tableau des axiomes logiques.

4.2.7 Tableau des instances

Dans cette section nous allons présenter une description de quelques instances de l'ontologie. Pour cela, nous spécifierons les noms des individus et les valeurs des attributs pour chacun d'eux; le Tab 27 illustre quelques instances pour chaque classe.

Concept	Instance	Propriété	Valeur
Application	**RESERVATION_VOL**	Nom Texte-description	RESERVATION_VOL Réservation des vols
Activité	**GET_DETAILS_VOL**	Nom Texte-description	GET_DETAILS_VOL Prendre tous les détails sur le vol.
Activité-atomique	**CONFIRM_RESERVATION**	Nom Texte-description	CONFIRM_RESERVATION Envoyer la confirmation au client.

Activité-complexe	**OPERATION_RESERVATION_VOL**	Nom
		Texte-description
			OPERATION_RESERVATION_VOL
			=SEQUENCE_ACTIVITE{
			GET_DETAILS_VOL,
			GET_CONTACT_DETAILS,
			RESERVE_VOL,
			CONFIRM_RESERVATION
			}

Input-calculé	**NOM_COMPTE**	Nom Texte-description Type	NOM_COMPTE Le nom du compte ou du domaine String
Input-calculé	**MOTDEPASSE**	Nom	MOTDEPASSE
		Texte-description	Mot de passe du compte
		Type	String
Output-calculé	**ACK**	Nom	ACK
		Texte-description	Retourner un acquittement à l'utilisateur
		Type	Booléen
Précondition-calculée	**EST_MEMBRE(NOM_COMPTE)**	Nom	EST_MEMBRE(NOM_COMPTE)
		Texte-description	Le nom du compte est-il valide ?
		Type	Booléen
Effet-calculé	**LOGGED_IN (NOM_COMPTE, MOTPASSE)**	Nom	LOGGED_IN (NOM_COMPTE, MOTPASSE)
		Texte-description	Ouvrir la session de NOM_COMPTE avec le mot de passe fourni.
		Type	Activité

Créateur	**AIR_ALGERIE**	Nom	AIR_ALGERIE
		Téléphone	00213 31 92 45 74 00213 31 92 46 90
		Fax	00213 31 95 55 84 00213 31 92 48 98
		e-mail	contact@air-algerie.dz infos@air-algerie.dz
		Site-web	www.air-algerie.dz
		@-physique	Aéroport international Mohammed Boudhiaf, Ain-Bey Constantine (Algérie)
(…)	(…)	(…)	(…)

Tab 27: Tableau des instances.

4.3 Formalisation

Dans cette étape, nous utilisons le formalisme des logiques de descriptions pour formaliser le modèle conceptuel que nous avons obtenu dans l'étape de conceptualisation.

4.3.1 Construction de TBOX

Nous définissons ici les concepts et les rôles relatifs à notre domaine, en utilisant les constructeurs fournis par les logiques de descriptions pour donner des descriptions structurées aux concepts et rôles. Par exemple une activité doit avoir un nom et un seul, au moins un paramètre input en entrée, un paramètre output en sortie et produire un moins un effet à la fin de son exécution. On peut décrire cette phrase en logique de description par :

Activité= (∃ Nom.String) ∩ (=1 Nom.String) ∩ (≥1 Avoir-input.Input) ∩ (≥1 Avoir-output.Output) ∩ (≥0 Avoir-précondition.Précondition) ∩ (≥1 Avoir-effet.Effet).

De plus, nous spécifions les relations de subsomption qui existent entre les différents concepts ; par exemple pour spécifier que la classe Activité est subsumée par la classe Application on écrit :

Activité ⊆ Application

Les définitions de différents concepts sont illustrées dans le tableau Tab 28.

Concept	Définition	Relation de subsomption
Application	-(∃ Nom.String) ∩ (=1 Produit.Application-comportement) ∩ (=1 Avoir-structure.Application-structure) ∩ (=1 Avoir-domaine.Application-domaine)	Application ⊆ Thing
Activité	= (∃ Nom.String) ∩ (=1 Nom.String) ∩ (≥1 Avoir-input.Input) ∩ (≥1 Avoir-output.Output) ∩ (≥0 Avoir-précondition.Précondition) ∩ (≥1 Avoir-Effet.Effet).	Activité ⊆ application
Activité-atomique	Activité ∩ Application-modèle ∩ (≥1 Réalise.Activité-simple)	Activité-atomique ⊆ Activité Activité-atomique ⊆ Application-modèle
Activité-complexe	Activité ∩ Application-modèle ∩ (≥1 Paramètre-calculé.Paramètre) ∩ (≥0 Précondition-calculée.Précondition) ∩ (≥1 Input-calculé.Input) ∩ (≥1	Activité-complexe ⊆ Activité Activité-complexe ⊆ Application-modèle

	Output-calculé.Output) ∩ (∀ S'exécute-comme{ Sequence-Activité, Activité-compétitive,Choix-activité, Activité-répéter-jusque-à }) ∩ (≥1 Effet-calculé.Effet)	
Sequence-Activité	Activité-complexe	Sequence-Activité ⊆ Activité-complexe
Activité-compétitive	Activité-complexe	Activité-compétitive ⊆ Activité-complexe
Choix-activité	Activité-complexe	Choix-activité ⊆ Activité-complexe
Activité-répéter-jusque-à	Activité-complexe	Activité-répéter-jusque-à ⊆ Activité-complexe
Application-domaine	(∃ Nom.String) ∩ (=1 Avoir-info.Application-information) ∩ (=1 Avoir-secteur.Application-secteur)	Application-domaine ⊆ Thing
Domaine-description	(∃ Nom.String) ∩ ∀ Description-type.{Description-fonctionnelle, Description-non-fonctionnelle}	Domaine-description ⊆ Thing
Application-information	Application-domaine ∩ (=1 Participe.Niveau) ∩ (=1 Créée-par. Créateur) ∩ (=1 Date-info.Date)	Application-information ⊆ Application-domaine
Application-secteur	Application-domaine ∩ (∀ Avoir-secteur{ Education, Medicine, …, E-commerce })	Application-secteur ⊆ Application-domaine
Medicine	(…)	Medicine ⊆ Application-secteur
Education	(…)	Education ⊆ Application-secteur
E-commerce	Application-secteur ∩ (=1 Décrit.Domaine-description) ∩ (≥1 Dispose-de.Produit) ∩ (=1 Mode-livraison.Transportation)	E-commerce ⊆ Application-secteur
Paramètre	-(∃ Nom.String) ∩ (∃ Type.Thing) ∩ (=1 Nom.String) ∩ (=1 Type.Thing) ∩ (≥1 Se-réfère-à.Activité)	Paramètre ⊆ Thing
Input	Paramètre	Input ⊆ Paramètre
Output	Paramètre	Output ⊆ Paramètre
Précondition	Paramètre	Précondition ⊆ Paramètre
Effet	Paramètre	Effet ⊆ Paramètre
Paramètre-calculé	-(∃ Nom.String) ∩ (∃ Type.Thing) ∩ (=1 Nom.String) ∩ (=1 Type.Thing) ∩ (≥1 Se-réfère-à.Activité-complexe)	Paramètre-calculé ⊆ Thing
Input-calculé	Paramètre-calculé	Input-calculé ⊆ Paramètre-calculé
Output-calculé	Paramètre-calculé	Output-calculé ⊆ Paramètre-calculé
Précondition-calculée	Paramètre-calculé	Précondition-calculée ⊆ Paramètre-calculé
Effet-calculé	Paramètre-calculé	Effet-calculé ⊆ Paramètre-calculé
Application-structure	-(∃ Nom.String) ∩(=1 Nom.String) ∩ (∀ Structure-type{IDL-Structure, XDR-Structure, … WSDL-Structure})	Application-structure ⊆ Thing
IDL-Structure	Application-structure ∩ (≥1 Avoir-interface-struct.Interface-structure)	IDL-Structure ⊆ Application-structure
XDR-Structure	(…)	XDR-Structure ⊆ Application-structure
WSDL-Structure	(…)	WSDL-Structure ⊆ Application-structure
Application-comportement	-(∃ Nom.String) ∩ (=1 Nom.String) ∩ (=1 Avoir-modèle.Application-modèle) ∩ (≥1 Ensemble-de.Activité)	Application-comportement ⊆ Thing
Interface-structure	(=1 Nom.String) ∩ (≥1 Avoir-méthode-structure.Méthode-structure) ∩ (=1Mappe-interface.Application-comportement)	Interface-structure ⊆ Thing
Méthode-structure	(=1 Nom.String) ∩ (≥1 Avoir-input-struct.Input-structure) ∩ (≥1 Avoir-output-struct.Output-structure) ∩ (=1Mappe-méthode.Activité)	Méthode- structure ⊆ Thing
Input- structure	(=1 Nom.String) ∩ (=1Mappe-input.Input)	Input- structure ⊆ Thing

Output-structure	(=1 Nom.String) ∩ (=1Mappe-output.Ouput)	Output- structure ⊆ Thing
Niveau	(∀ Nom.{ Niveau- Collaboratif, Niveau- Applicatif })	Niveau ⊆ Thing
Produit	(=1 Nom.String) ∩ (=1 Code.String) ∩ (=1 Classification.String)	Produit ⊆ Thing
Transportation	(…)	Transportation ⊆ Thing
Créateur	-(=1 Nom.String) ∩ (≥1Téléphone.String) ∩ (≥1 Fax.String) ∩ (=1 e-mail.String) ∩ (=1 Site-web.String) ∩ (=1 @-physique.String).	Créateur ⊆ Thing
Date	(∀ Year.{ 1970, …, 2020 }) ∩ (∀ Month.{Janvier, …Décembre }) ∩ (∀ Day.{01, …31 })	Date ⊆ Thing
(…)	(…)	(…)

Tab 28: Définitions de concepts (dans TBOX).

En ce qui concerne les rôles, nous les définissons en donnant les couples des concepts sources et cibles de chacune, et/ou en spécifiant son rôle inverse. Par exemple la relation *Avoir-paramètre* qui relie une activité avec ses paramètres est spécifiée par :

Avoir-paramètre: (Activité, Paramètre)
Avoir-paramètre: Se-réfère-à ⁻

Le tableau Tab 29 représente les définitions des différents rôles de notre ontologie.

Rôles	Couple (domaine, co-domaine)	Rôle inverse
Produit	(Application, Application-comportement)	Produit-par
Avoir-structure	(Application, Application-structure)	-
Avoir-domaine	(Application, Application-domaine)	-
Avoir-paramètre	(Activité, Paramètre)	Se-réfère-à
Avoir-précondition	(Activité, Précondition)	-
Avoir-input	(Activité, Input)	-
Avoir-output	(Activité, Output)	-
Avoir-effet	(Activité, Effet)	-
Avoir-sous-activités	(Activité, Activité)	-
Réalise	(Activité-atomique, Activité-complexe)	Réalisée-par
Réalisée-par	(Activité-complexe, Activité-atomique)	Réalise
Avoir-paramètre-calculé	(Activité-complexe, Paramètre-calculé)	Se-réfère-à
Avoir-input-calculé	(Activité-complexe, Input-calculé)	-
Avoir-output-calculé	(Activité-complexe, Output-calculé)	-
Avoir-effet-calculé	(Activité-complexe, Effet-calculé)	-
Avoir-précondition-calculé	(Activité-complexe, Précondition-calculée)	-
S'exécute-comme	(Activité-complexe, Sequence-Activité) (Activité-complexe, Activité-compétitive) (Activité-complexe, Choix-activité) (Activité-complexe, Activité-répéter-jusque-à)	-
Avoir-info	(Application-domaine, Application-information)	-
Avoir-secteur	(Application-domaine, Application-secteur)	-
Participe	(Application-information, Niveau)	-
Créée-par	(Application-information, Créateur)	Crée
Date-info	(Application-information, Date)	-

Avoir-secteur	(Application-secteur, Education) (Application-secteur, Medicine) (Application-secteur, E-commerce)	-
Décrit	(E-commerce, Domaine-description)	-
Dispose-de	(E-commerce, Produit)	-
Mode-livraison	(E-commerce, Transportation)	-
S'exécute-comme	(Application-modèle, Activité-complexe) (Application-modèle, Simple-Activité) (Application-modèle, Activité-atomique	-
Se-réfère-à	(Paramètre, Activité)	Avoir-paramètre
Se-réfère-à	(Paramètre-calculé, Activité-complexe)	Paramètre-calculé
Structure-type	(Application-structure, IDL-structure) (Application-structure, XDR-structure) (Application-structure, WSDL-structure)	-
Avoir-interface-struct	(IDL-structure, Interface-structure)	-
Mappe-interface	(Interface-structure, Application-comportement)	-
Avoir-Méthode-struct	(Interface-structure, Méthode-structure)	-
Mappe-Méthode	(Méthode-structure, Activité)	-
Avoir-input-struct	(Méthode-structure, Input-struct)	-
Avoir-output-struct	(Méthode-structure, Output-struct)	-
Mappe-input	(Input-struct, Input)	-
Mappe-output	(Output-struct, Output)	-
Crée	(Créateur, Application-information)	Créée-par
Produit-par	(Application-comportement, Application)	Produit

Tab 29: Définitions des Rôles (dans TBOX).

4.3.2 Construction de ABOX

Le langage assertionnel est dédié à la description des faits, en spécifiant les individus (avec leurs classes) et les relations entre eux de la manière suivante :

$$A : C$$

Pour dire que A est une instance de la classe C ;

Par exemple VOL_RESERVATION : Application.

$$(A1, A2) : R$$

Pour dire que les deux individus A1 et A2 sont reliés par la relation R ;

Par exemple : (AIR_ALGERIE, VOL_RESERVATION_INFOS) : Créateur

Dans les tableaux Tab 30 Tab 31 nous définissons quelques assertions :

Concept	Description
Application	VOL_RESERVATION : Application HOTEL_RESERVATION : Application LOCATION_VOITURE : Application
Activité	GET_VOL _DETAILS: Activité RESERVE_VOL: Activité …
Activité-atomique	CONFIRM_RESERVATION: Activité-atomique …
Activité-complexe	RESERVE_VOL: Activité-complexe
Input-calculé	NOM_COMPTE: Input-calculé
Input-calculé	MOTDEPASSE: Input-calculé
Output-calculé	ACK: Output-calculé
Précondition-calculée	EST_MEMBRE(NOM_COMPTE): Précondition-calculée
Effet-calculé	LOGGED_IN (NOM_COMPTE, MOTDEPASSE): Effet-calculé
Créateur	AIR_ALGERIE : Créateur
(…)	(…)

Tab 30 : Description assertionnelle de concepts.

Relation	Description
Créateur	(AIR_ALGERIE, VOL_RESERVATION_INFOS) : Créateur ; …
Avoir-précondition-calculée	(RESERVE_VOL, EST_MEMBRE(NOM_COMPTE) : Avoir-précondition-calculée ; …
Avoir-input-calculé	(RESERVE_VOL, NOM_COMPTE): Avoir-input-calculé ; …
Avoir-output-calculé	(RESERVE_VOL, ACK): Avoir-output-calculé ; …
Avoir-effet-calculé	(RESERVE_VOL, LOGGED_IN(NOM_COMPTE, MOTDEPASSE): Avoir-effet-calculé; …
(…)	(…)

Tab 31: Description assertionnelle de relations.

Jusqu'à présent, nous avons attribué un modèle conceptuel à l'ontologie d'application, dans la phase suivante, nous monterons son implémentation en utilisant éventuellement les outils qui ont été développés à cet objectif.

4.4 Implémentation

Après avoir conçu l'ontologie, il nous reste maintenant que son implémentation. Pour cela, nous disposons de nombreux langages et d'outils qui permettent d'implémenter l'ontologie proposée.

Notre choix porte sur OWL qui représente un langage de codification utilisé pour implémenter l'ontologie en OWL (un langage de définition d'ontologie pour le Web), et cela pour toutes les fonctionnalités sémantiques que permet OWL et qui sont plus riches que celles des langages RDFS & DAML+OIL. OWL fait partie du paradigme des logiques de descriptions. Sa sémantique peut être définie via une translation (transcription) vers la logique de descriptions SHIQ.

4.4.1 Présentation de l'éditeur PROTEGE OWL

PROTEGE OWL est une interface modulaire, développée au Stanford Medical Informatics de l'Université de Stanford7, permettant l'édition, la visualisation, le contrôle (vérification des contraintes) d'ontologies, l'extraction d'ontologies à partir de sources textuelles, et la fusion semi-automatique d'ontologies [72]. Le modèle de connaissances de PROTEGE OWL

est issu du modèle des frames et contient des classes (concepts), des slots (propriétés) et des facets (valeurs des propriétés et contraintes), ainsi que des instances des classes et des propriétés. PROTEGE OWL autorise la définition de méta-classes, dont les instances sont des classes, ce qui permet de créer son propre modèle de connaissances avant de bâtir une ontologie. De nombreux plug-ins sont disponibles ou peuvent être ajoutés par l'utilisateur.

L'interface, très bien conçue, et l'architecture logicielle permettant l'insertion de plug-ins pouvant apporter de nouvelles fonctionnalités (par exemple, la possibilité d'importer et d'exporter les ontologies construites dans divers langages opérationnels de représentation tels que OWL ou encore la spécification d'axiomes) ont participé au succès de PROTEGE OWL, qui regroupe une communauté d'utilisateurs très importantes et constitue une référence pour beaucoup d'autres outils [73]. La figure (fig. 5) présente l'interface de PROTEGE OWL.

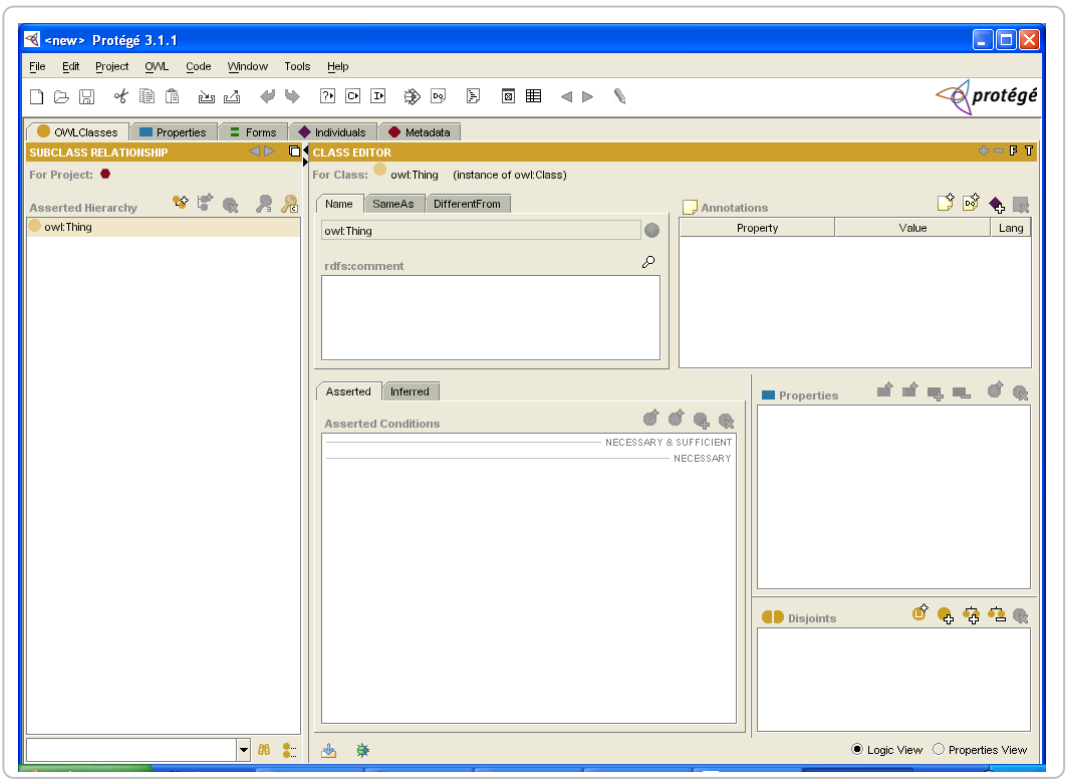


Fig 5: Interface de PROTEGE OWL

4.4.2 Création des classes et la hiérarchie des classes

Nous commencerons tout d'abord par la création des concepts spécifiés dans l'étape de conceptualisation. PROTEGE OWL nous offre également un moyen de construire la hiérarchie de concepts, la figure (fig. 6) présente comment procède-t-on pour créer un concept.

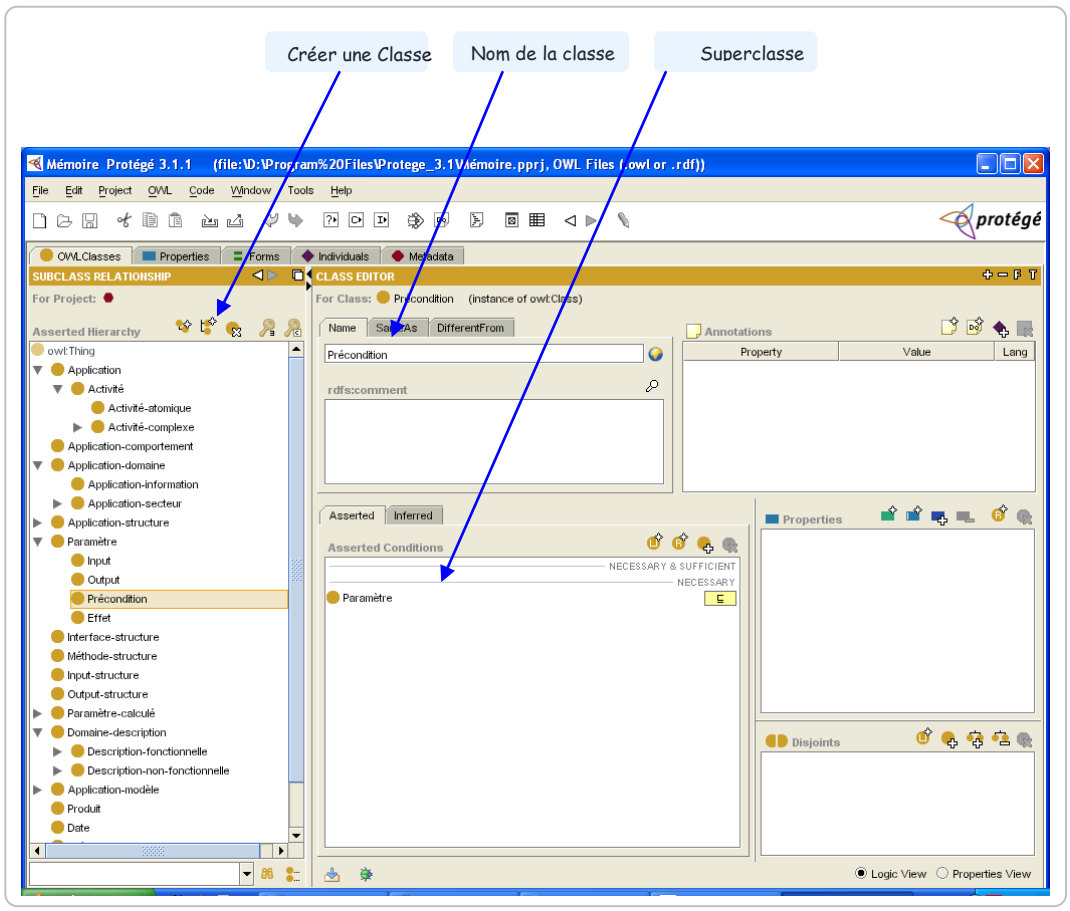


Fig 6: création de Classes.

L'ensemble de concepts créés se synthétise dans une hiérarchie comme le montre la figure (Fig. 7).

Fig 7 : Hiérarchie de Classes.

4.4.3 Création des propriétés

Après avoir construit les concepts, nous allons maintenant créer les propriétés pour chacun d'eux, les attributs vont être créés sous PROTEGE OWL par *'dataTypeProperty'* et les relations par *'objectProperty'*. Les propriétés d'une classe sont les propriétés héritées de sa superclasse, plus ses propres propriétés privées.

La figure (fig. 8) montre les potentialités de PROTEGE OWL pour la création des propriétés.

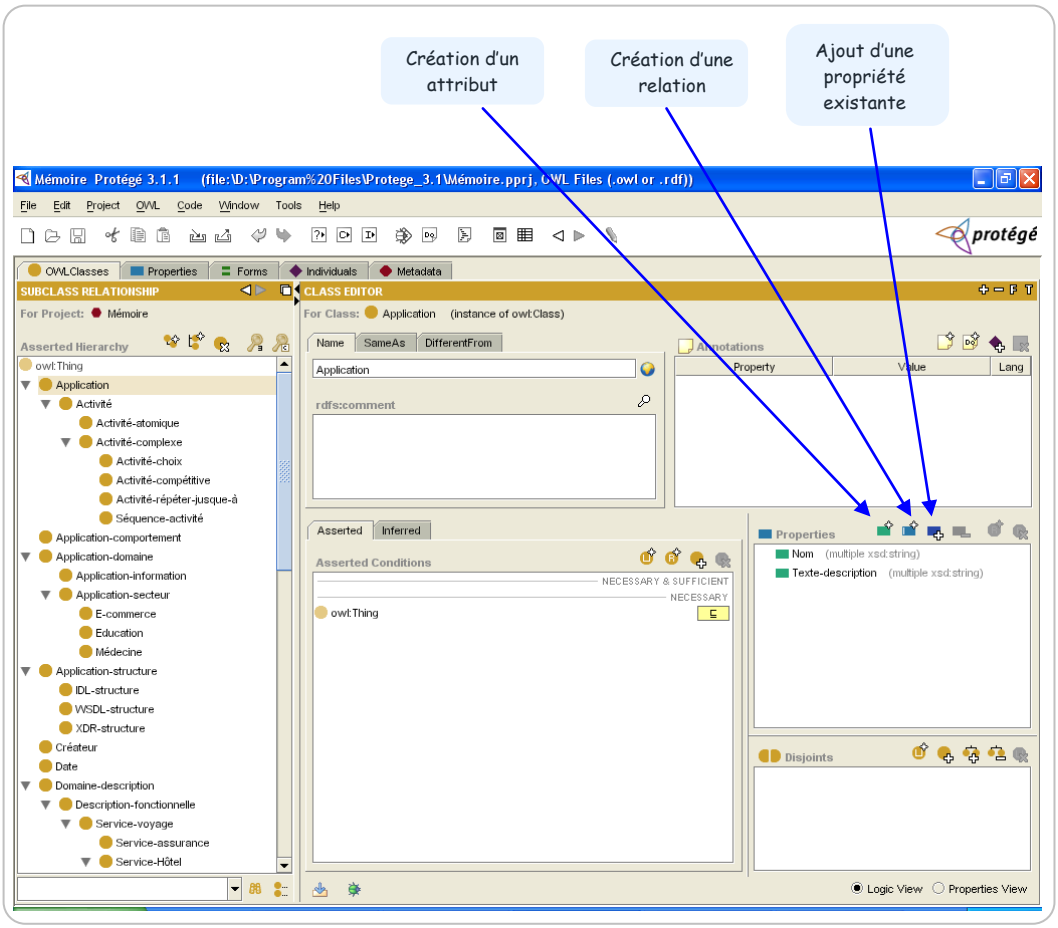


Fig 8: Création de propriétés pour une classe.

4.4.3.1 Création des attributs (dataTypeProperty)

La création des attributs se fait en cliquant au dessus le bouton *'dataTypeProperty'* :  , ce qui génère la fenêtre comme la montre la figure 9 ; qui permet de fournir les champs pour remplir le nom, le domaine, le type de valeurs de l'attribut et le type XML associé.

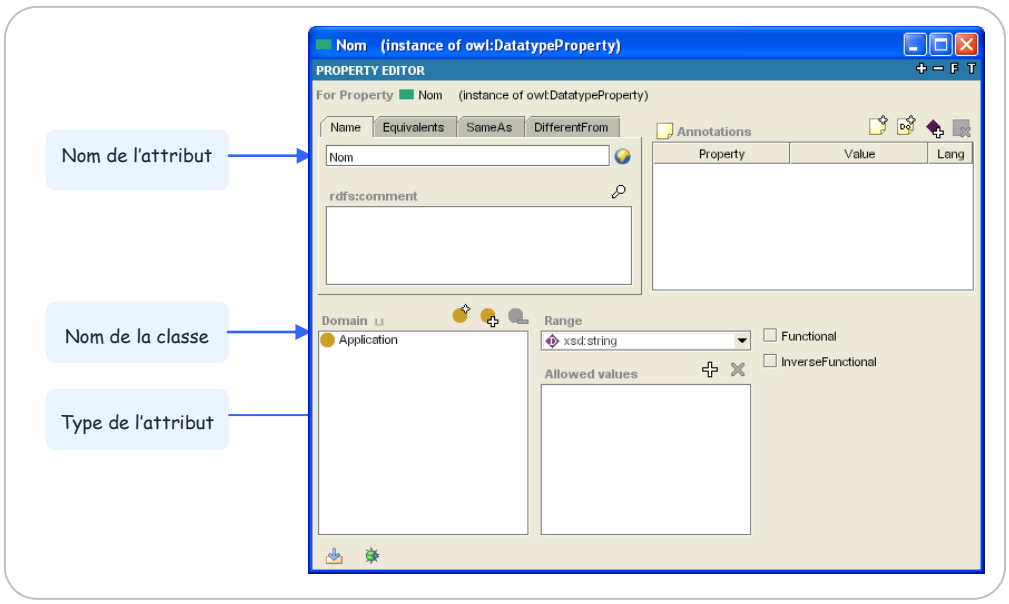


Fig 9: Création d'un attribut.

4.4.3.2 Création des relations (ObjectProperty)

Pour la création de relations, nous devons spécifier le nom de la relation, le domaine, le co-domaine et la relation inverse si elle existe. Nous pouvons aussi associer à cette définition le type de la relation (symétrique, transitive,…)

La figure qui suit montre une définition de la relation "Avoir-structure".

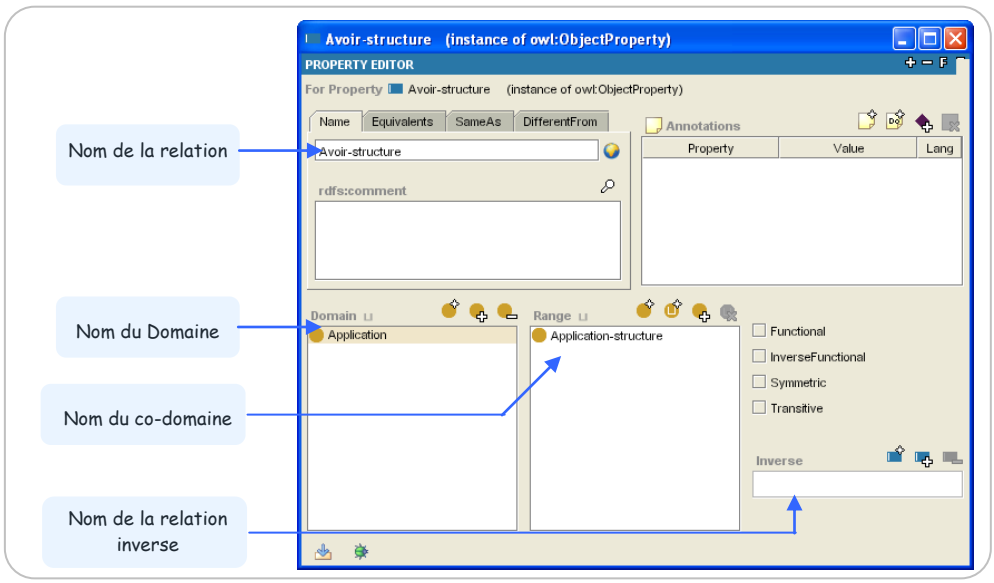


Fig 10: Création d'une relation entre deux classes.

4.4.3.3 Restrictions sur les propriétés

Les restrictions définissent des classes anonymes, les membres de ces classes sont les individus qui vérifient les conditions, les restrictions sont souvent utilisées pour définir les conditions nécessaires pour une classe.

De point de vue Protégé, pour créer une restriction il y a deux manières, pour réaliser ceci voir la figure ci-dessous :

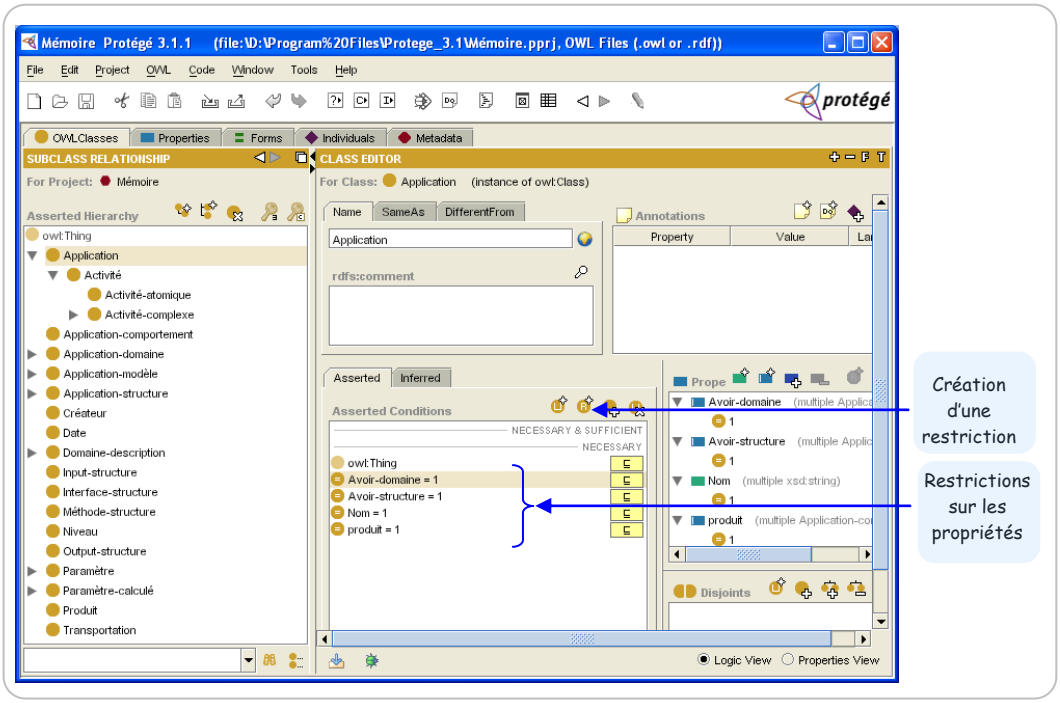


Fig 11: les restrictions créées sur les propriétés d'une classe.

Si vous cliquez au dessus du bouton ( ), PROTEGE OWL vous afficherait la fenêtre suivante :

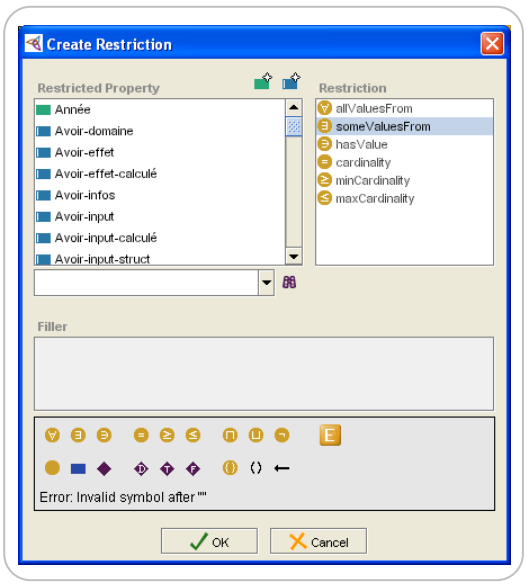
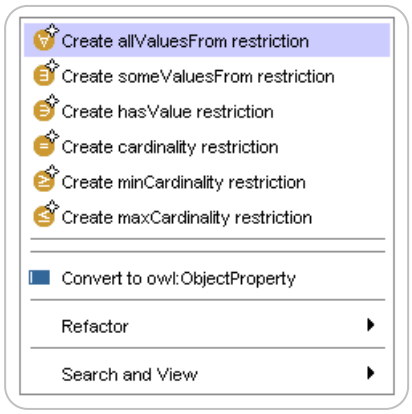


Fig 12: Création d'une restriction.

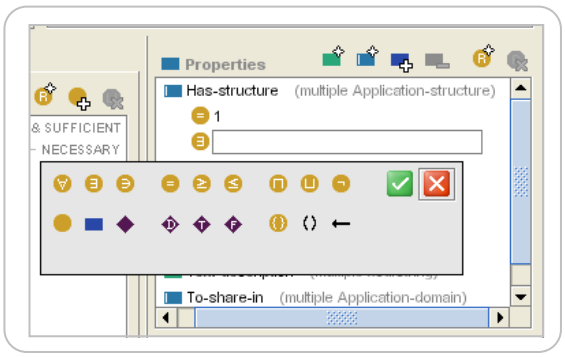
Une fois la fenêtre s'affiche, vous sélectionnez la propriété sur laquelle vous voulez faire la restriction, puis choisissez le type de restriction comme *Cardinality*, *minCardinality*, *maxCardinality*, *hasValue*,…

Vous pouvez aussi faire une restriction par une autre manière en cliquant par le bouton droit de la sourie sur la propriété que vous voulez faire une restriction.

Si vous procédez comme ceci, PROTEGE OWL vous afficherait le menu contextuel suivant :



Vous pouvez en suite introduire l'expression de la restriction à travers les boutons dans le menu qui s'affiche.



4.4.4 Création des instances

La dernière étape consiste à créer les instances des classes dans la hiérarchie. Définir une instance individuelle d'une classe exige

1. choisir une classe
2. créer une instance individuelle de cette classe
3. la renseigner avec les valeurs des attributs.

Par exemple, la figure (fig. 13) montre la création d'un individu de la classe Application.

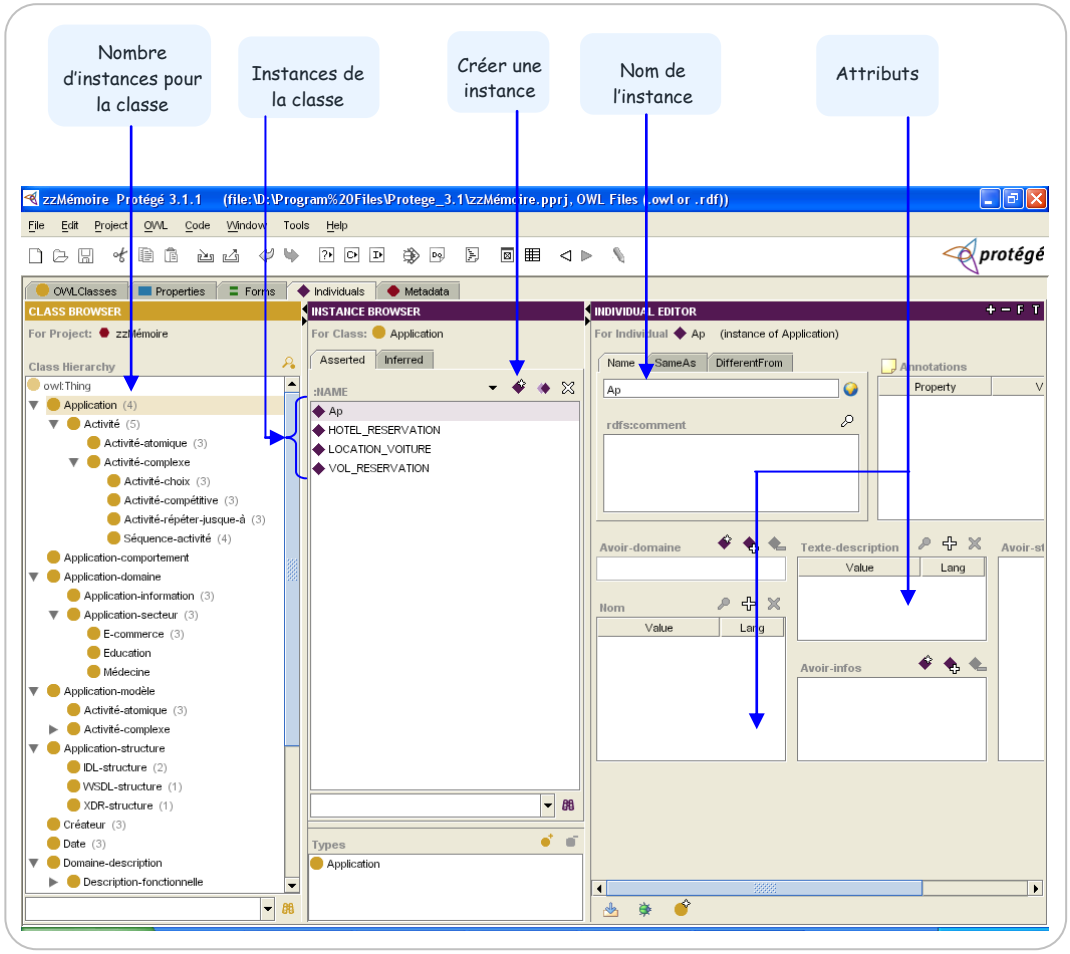


Fig 13: Création des instances.

4.4.5 Génération du code

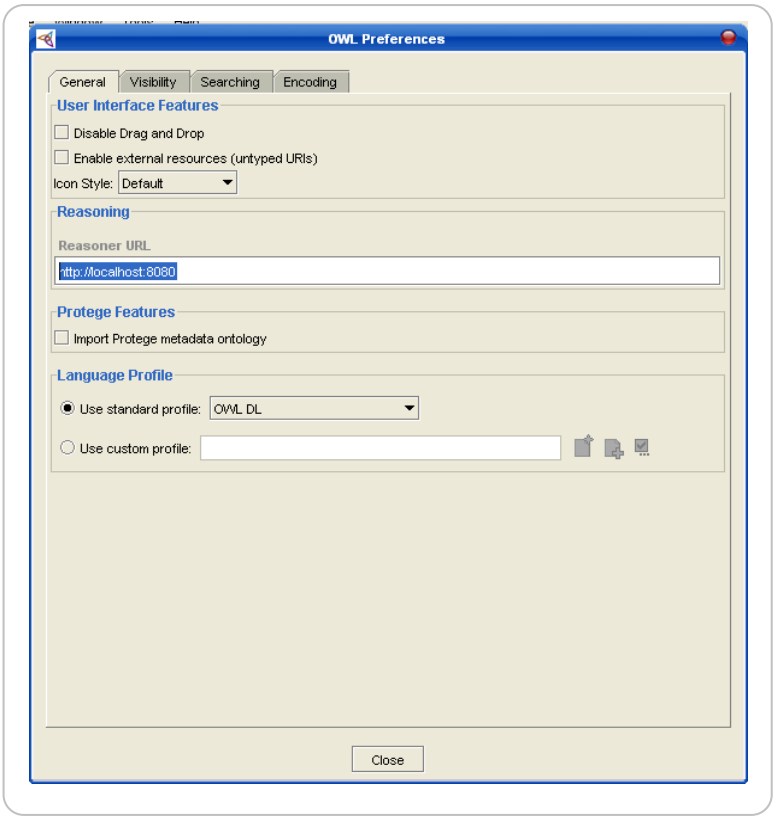
L'outil PROTEGE OWL a été conçu pour dégager et libérer le développeur de la complexité du codage, même pour implémenter une petite ontologie, celle–ci va prendre plusieurs lignes du codes et nécessite un grand effort, ce que nous pouvons le constater après la génération du code pour notre ontologie, une partie de ce code sera donné à l'annexe de ce mémoire (n'inclut pas le code des instances).

4.5 Test & évolution de l'ontologie

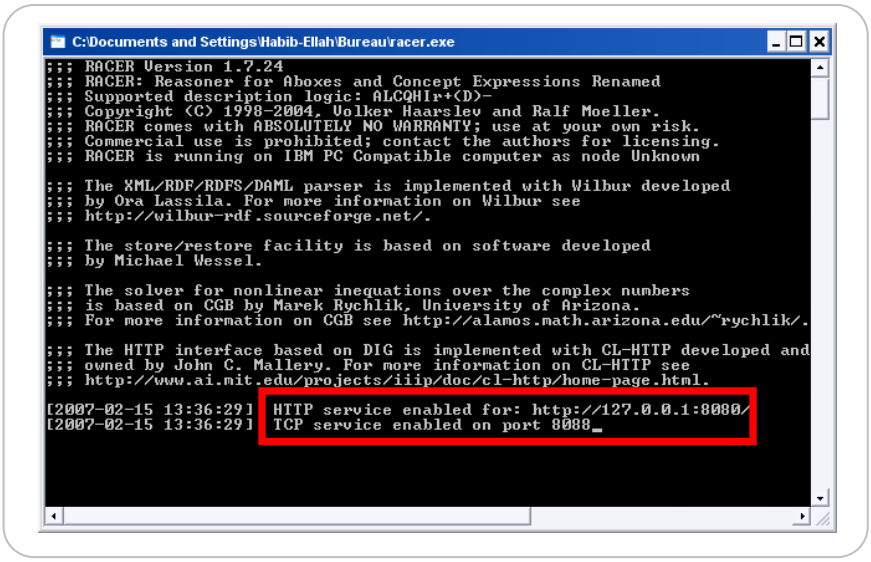
Nous avons utilisé le système Racer pour tester l'ontologie d'application, nous distinguons deux types de test: test de consistance et de satisfiabilité; le premier type de test consiste à enlever l'inconsistance entre les concepts et cela en utilisant la test de subsumption incorporé au système Racer, par contre le test de satisfiabilité permet de vérifier pour chaque concept l'existence des instances; un concept C est satisfiable si et seulement si, il existe au moins une interprétation I (instance) pour le concept C.

Racer se présente sous la forme d'un serveur qui peut être accédé par le protocole TCP ou HTTP. Donc, nous devons d'abord configurer la connexion au serveur hébergeant le système Racer. Mais ce que nous souhaitons, est de tester l'ontologie localement. Pour cela nous allons procéder comme suit:

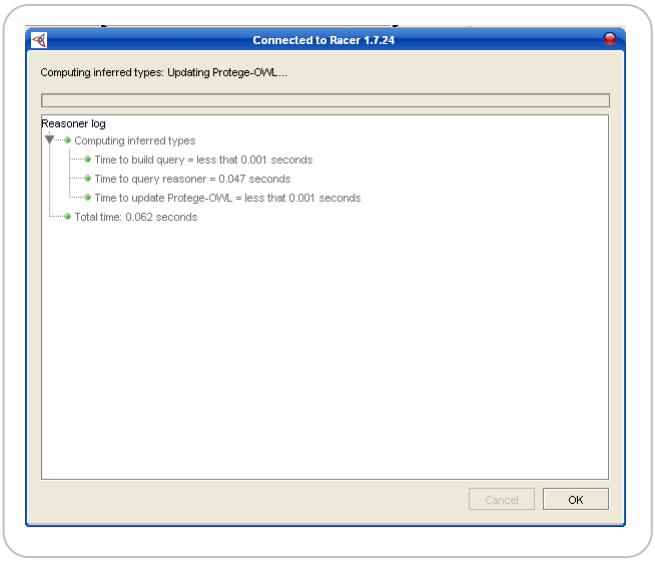
1. Activer le menu *OWL* de Protégé.
2. Choisir l'option *Préférences…*
3. Dans le fenêtre qui s'affiche, saisir *'http://localhost:8080'*. Puis valider en pressant sur le bouton 'Close'.

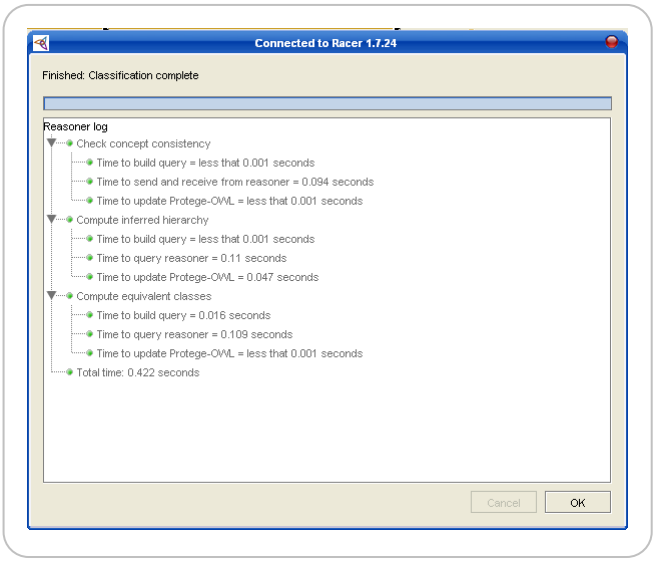


4. Télécharger et exécuter Racer localement, les services http et TCP vont être activés sur le port 8080 de la machine local (localhost)



D'après les tests que nous avons appliqués à l'ontologie d'application, aucune erreur n'est produite lors du test.





Jusqu'à maintenant, nous avons développé l'ontologie d'application, il nous restera que suivre son évolution, c'est-à-dire les nouveaux concepts à ajouter dans la partie terminologique (TBOX) de l'ontologie. Le résultat de cette étape est une nouvelle ontologie avec une nouvelle hiérarchie de concepts. Pour cela, nous proposons l'utilisation du raisonnement par classification qui est l'un des mécanismes de raisonnements de base pour les logiques de descriptions.

5 Rétablir la communication entre les applications de l'entreprise.

Dans un tel environnement, la modification ou l'ajout de certains composants dans un système a un impact négatif sur son fonctionnement, ce qui conduit généralement à reconcevoir complètement le système ou bien d'en associer d'autres modules afin de garantir sa cohérence et sa flexibilité.

Dans le cadre de notre travail, nous tenons à répondre implicitement dans la suite de cette section aux questions suivantes :

1. Quels composants ont un effet sur le système d'intégration ?
2. Quelles entités sont concernées par le changement ?
3. Comment procéder afin de pouvoir rétablir la communication entre les applications dans le système d'intégration ?

5.1 Pourquoi rétablir ?

L'objectif majeur de l'utilisation des ontologies au sein d'un système d'intégration est de masquer l'hétérogénéité des applications, des données et des modèles d'échanges, en plus de fournir une sémantique explicite aux entités du système pour qu'elles soient compréhensibles et interprétables.

Cependant, les ontologies ont aussi la capacité de faciliter l'interopérabilité entre des applications hétérogènes, et cela en construisant pour chaque application appartenant au système une ontologie qui s'appelle l'ontologie d'application, mais le problème qui se pose dans ce contexte c'est comment l'application se comporte-elle pour qu'elle soit en mesure d'utiliser l'ontologie ? Ce nouveau challenge apparaît car l'application n'a pas été préalablement conçue pour exploiter l'ontologie et de ce fait, on était obligé de rétablir la communication entre les applications à intégrer afin d'en fournir une sémantique interprétable entre elles.

Nous distinguons trois types de communication dans le système d'intégration de l'entreprise (Voir fig. 14)

1. Communication Application-Application
2. Communication Client-Application
3. Communication Partner-Application

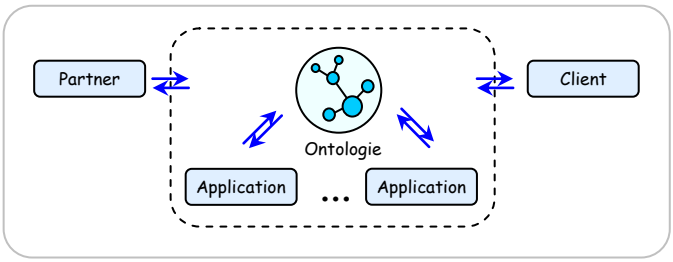


Fig 14: Types de communication dans le système d'intégration.

Nous nous intéresserons dans la suite de notre travail sur les deux premiers types de communication : la communication entre les applications internes de l'entreprise et la communication entre le client et les applications de l'entreprise [74].

5.2 Scénarios de communication entre les applications internes de l'entreprise

Comme nous l'avons déjà mentionné dans la section précédente, les applications de l'entreprise doivent changer leur comportement afin qu'elles répondent aux nouveaux

exigences requis par le système d'intégration. A cette fin, nous montrerons comment ces applications vont se communiquer à travers le vocabulaire produit par l'ontologie d'application.

L'ajout de nouvelles applications dans le système d'intégration n'entraine aucune influence négative sur le système car on pourrait concevoir des applications et d'en associer respectivement leur ontologie de tel sorte que les applications puissent exploiter aisément les ontologies.

Cependant, les applications déjà existantes dans le système n'ont aucun comportement envers les ontologies construites vu que celles-ci n'ont pas été préalablement conçues pour exploiter les ontologies. Pour ce faire, nous avons deux moyens qui servent à corriger le problème :

- Modifier le code des applications pour qu'elles soient en mesure d'exploiter l'ontologie.
- Associer à l'application un module logiciel externe interrogeant l'ontologie et recevant des résultats.

La première solution est rarement utilisée car bien souvent les logiciels ont été achetés et dans ce cas il est hors de question de les modifier [70].

La seconde solution est presque toujours privilégiée et elle conduit au développement d'un module logiciel appelé '*Module_Communication*'

La figure ci-dessous (Fig. 15) présente un scénario de communication entre les applications internes de l'entreprise.

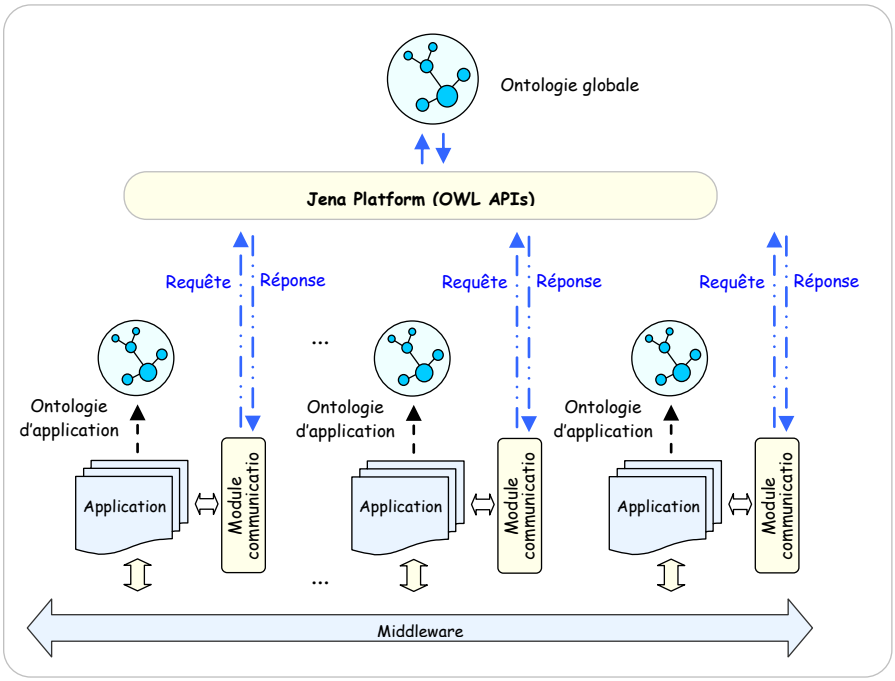


Fig 15: Scénario de communication entre les applications internes de l'entreprise.

L'élément Module_Communication est un composant développé notamment pour permettre à l'application d'interroger l'ontologie et d'en recevoir des résultats, ce module comporte les sous composants : Adaptateur, Requestor, Récepteur et Traitement. (fig. 16)

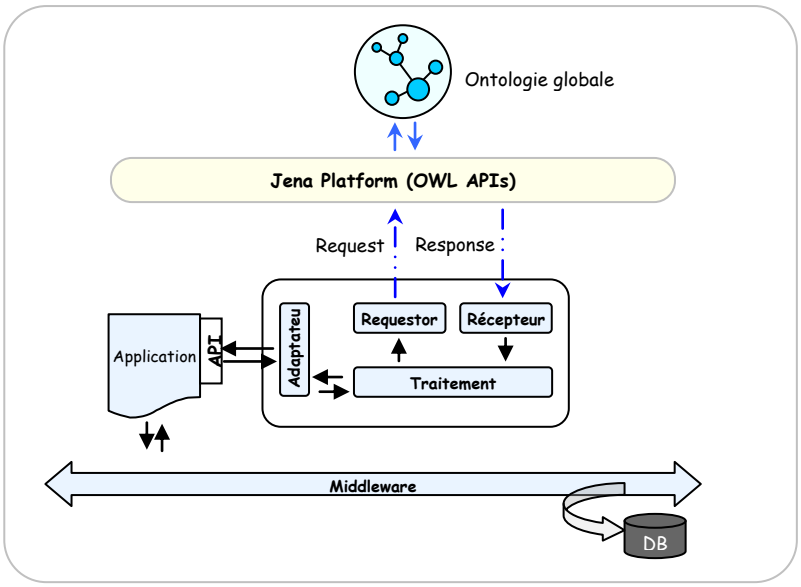


Fig 16: Structure de Module communication.

Adaptateur: le rôle de ce composant est de masquer l'hétérogénéité entre l'application et le Module_Communication, car ces deux entités ont été développées par des langages de programmation différents.

Traitement: ce composant permet à l'application de préparer la requête, c'est à dire de l'initialiser puis il passe le traitement au Requestor.

Requestor : le rôle du composant Requestor est de faire interroger l'ontologie.

Récepteur: ce composant a pour but de recevoir les résultats venant de l'ontologie.

La base de données du middleware *'BD APIs'* renferme une liste de toutes les interfaces des applications disponibles sur le middleware. Cette base de données est mise à jour à chaque fois qu'on ajoute ou qu'on retire des applications depuis le middleware.

Cependant, l'application connectée au middleware a l'aptitude de connaître toutes les applications présentes sur le middleware et cela via sa base de données *'BD APIs'*, c'est-à-dire que l'application dispose d'une API lui permettant d'interroger la base de données et d'en recevoir des résultats afin de connaître toutes les autres applications jointes au middleware.

En revanche, nous voudrions que les applications appartenant au middleware se communiquent à travers les ontologies afin que l'échange de messages entre applications doive avoir une sémantique claire et interprétable. Pour arriver à cette fin, le *'Module communication'* va récupérer l'API de l'application permettant d'interroger la base de données du middleware et plus particulièrement c'est le composant *'Traitement'* qui va utiliser cette API pour interroger l'ontologie. De ce fait, le composant doit préparer la

requête afin d'interroger l'ontologie que ce soit pour connaître les applications présentes dans le système d'intégration ou bien pour interpréter les messages échangés entre les applications. Par exemple, quand l'application reçoit des messages venus des autres applications, le composant doit traiter ces massages grâce à l'ontologie puis il découve leur nature et invoque par la suite l'application pour répondre efficacement aux applications renvoyant ces messages.

Les sous modules du *'Module* communication' ont été implémentés en Java, nous avons utilisé éventuellement les APIs de Jena. De même, nous aurons pu également utiliser les APIs de PROTEGE OWL. Parmi ces APIs on trouve l'API qui sert à interroger l'ontologie, elle s'appelle aussi l'interface Query: `edu.stanford.smi.protege.model.query` et bien d'autres APIs fournies par l'éditeur PROTEGE OWL.

5.3 Scénarios de communication entre le client et les applications de l'entreprise

Nous montrerons dans cette section, et contrairement à ce que nous avons vu auparavant, comment les applications de l'entreprise vont se comporter quand elles reçoivent la requête du client. De ce fait, nous venons de voir quelles sont les entités nécessaires à introduire pour que le changement apporté par le déploiement des ontologies au sein d'un système d'intégration soit robuste et flexible. Pour cela nous montrerons le rôle de chaque entité ajoutée au système et le trajet que va prendre la requête du client depuis sa déposition jusqu'à ce que ce dernier reçoive la réponse.

La figure ci-dessous (Fig. 17) présente un scénario de communication entre le client et les applications de l'entreprise et qui montre les entités ajoutées au système d'intégration de l'entreprise.

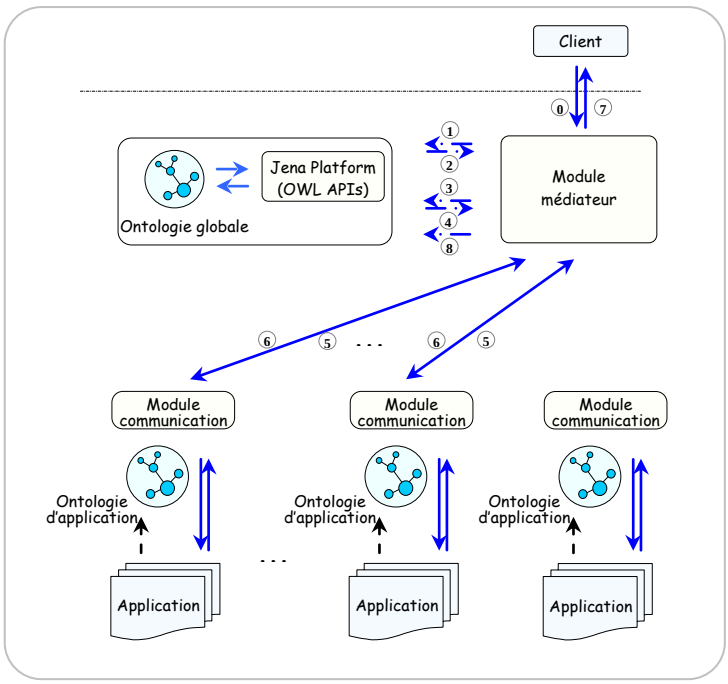


Fig 17: Scénario de communication entre le client et les applications de l'entreprise.

Pour commencer, nous nous attachons d'abord à dégager le rôle de chaque entité introduite dans le système d'intégration, ces entités doivent apporter une solution adéquate de telle sorte que les applications de ce système peuvent aisément manipuler les ontologies. Module médiateur, Module communication et la plateforme Jena y sont.

Module médiateur : le but de ce module est également de recevoir les requêtes provenant des clients, puis les décomposer et les distribuer vers les applications destinatrices appropriées y compris la plateforme Jena. Une fois que les réponses sont renvoyées, ce module doit les composer afin de satisfaire les requêtes des clients.

Module communication : comme son nom l'indique, ce module a pour but d'interpréter les messages venus que ce soit du côté Module médiateur vers l'application destinatrice du message ou bien du côté application vers le module médiateur. Car ces deux derniers ne sont pas forcément sur le même middleware.

Plateforme Jena : offre un certain nombre d'APIs OWL servant à exploiter l'ontologie.

Venons maintenant à découvrir les différentes étapes que doit suivre une requête depuis sa création jusqu'à son arrivé à la destination. Nous distinguons neuf étapes essentielles :

- Etape 0: cette étape se résume par la déposition de la requête du client et son envoi vers le module médiateur afin de la décomposer en sous-requêtes.
- Etape 1 : cette étape sert à découvrir les applications ou bien les activités de l'entreprise à travers les APIs fournies par la plateforme Jena, la recherche peut comporter aussi des restrictions sur le domaine auquel le client s'intéresse et qu'il veut joindre.
- Etape 2 : cette étape représente l'ensemble des activités offertes par l'entreprise.
- Etape 3 : sélectionner un ensemble d'applications ou d'activités parmi les offres dont dispose l'entreprise. Cette sélection doit être accompagnée par la demande de la liste des interfaces de ces applications qui les relient au middleware.
- Etape 4 : retourner toutes les applications ou les activités dont le client a besoin, la réponse doit inclure aussi toutes les informations relatives à la structure de ces applications avec le middleware.
- Etape 5 : invoquer les applications choisies en prenant en compte leur structure qui les relie au middleware, donc le module médiateur a déjà des informations préalables sur le comportement, que ce soit sur la structure de l'application.
- Etape 6 : retourner tous les résultats au module médiateur pour les reconstituer et les composer avant de les passer au client.
- Etape 7 : présenter le résultat au client.
- Etape 8 : mette à jour l'ontologie si c'est nécessaire.

Cet ensemble d'étapes présente un scénario possible de communication que peuvent prendre les applications afin de s'interagir et s'échanger des messages en exploitant évidement le vocabulaire de l'ontologie. Rappelons que les scénarios de communication, alors qu'ils ne sont pas trop détaillés, ont pour but de capturer les concepts de l'ontologie d'application vu l'absence des experts du domaine qui constituent la phase prépondérante pour en arriver.

6 Conclusion

Dans ce chapitre, nous avons présenté une architecture basée sur les ontologies pour l'intégration d'applications de l'entreprise. Nous avons montré la place prépondérante que détiennent les ontologies au seine de cette architecture ainsi que leur effet dans le fonctionnement global du système d'intégration et cela à travers les scénarios de communication que nous avons développés.

Après, nous avons développé une ontologie d'application OWL dans le cadre de l'intégration d'applications de l'entreprise. Nous avons proposé un processus de construction d'une ontologie sur lequel nous nous sommes basés afin de construire l'ontologie d'application. Dans ce processus nous avons utilisé le langage RDF dans la phase de spécification afin d'établir un document formel de spécification des besoins. Nous nous sommes inspirés des différentes phases proposées par la méthode METHONTOLOGY pour la conceptualisation de l'ontologie afin d'atteindre un ensemble de représentations intermédiaires qui facilite sa formalisation ultérieure et cela en adoptant l'approche de la logique de descriptions. L'intérêt de ce formalisme est qu'il est d'une part suffisamment simple pour que des non-spécialistes puissent l'utiliser et, d'autre part, qu'il est plus expressif que les graphes conceptuels et les frames. Basé sur cette formalisation, nous avons choisi, le langage OWL, pour codifier l'ontologie formelle, en utilisant l'éditeur graphique PROTEGE OWL, afin de guider l'implémentation et de produire un document OWL. Par ailleurs, pour vérifier et raffiner l'ontologie OWL au cours du processus de développement, nous utilisons le système RACER. Ce dernier, fournit un support de raisonnement en translatant des expressions OWL à des expressions de la logique de description. Les services d'inférence fournis par RACER incluent le test de satisfiabilité d'un concept et le test de subsumption.

CONCLUSION & PERSPECTIVES

1 BILAN

Le besoin d'intégration des applications dans les entreprises se fait sentir de plus en plus pour accroitre leur réactivité, pour faire face à de nouvelles contraintes,…etc. la refonte complète des systèmes d'information n'est plus au goût du jour. Désormais, le souci des entreprises est de faire relier leurs applications de manière à ce que les échanges inter-applications doivent porter une sémantique interprétable entre des entités communicantes.

Dans ce mémoire, nous avons commencé par la présentation du domaine de l'intégration d'application d'entreprise dans laquelle nous avons présenté quelques définitions sur l'EAI, les différentes approches d'intégrations. Nous avons découvert les diverses typologies d'applications et les principales architectures d'intégration et enfin, nous avons présenté quelques technologies d'intégration à savoir les middlewares orientés services, les middlewares orientés portails,…etc. Après, nous nous sommes attachés à la présentation des ontologies, nous avons commencé par la définition de la notion d'"ontologie". Nous avons présenté les trois principaux formalismes de présentations à savoir les frames, les graphes conceptuels et les logiques de descriptions. Nous avons découvert quelques méthodologies de construction des ontologies, les langages de leur implémentation et enfin certains outils servant leur exploitation.

S'inscrivant dans le champ de l'intégration d'applications d'entreprise, et se voulant essentiellement au développement d'une ontologie d'application, nous avons traité dans ce mémoire le problème de l'hétérogénéité sémantique entre les applications de l'entreprise à travers les propositions suivantes:

1.1 DÉVELOPPEMENT DES SCÉNARIOS DE COMMUNICATION ENTRE LES APPLICATIONS DE L'ENTREPRISE

L'ajout des ontologies au sein du système d'intégration de l'entreprise nous a conduits à en reconcevoir entièrement. Pour cela, nous avons développé des scénarios de communication entre les applications de l'entreprise afin d'exploiter les ontologies construites et d'avoir un échange plus compréhensible entre les applications et cela en leur associant des modules logiciels pour garantir la cohérence et la flexibilité du système.

1.2 PROPOSITION D'UN PROCESSUS DE DÉVELOPPEMENT DES ONTOLOGIES D'APPLICATIONS DANS LE CADRE DE L'EAI

Nous avons proposé un processus de construction d'une ontologie d'application partant de connaissances brutes et arrivant à une ontologie d'application opérationnelle représentée par le langage OWL. Ce processus est composé de cinq phases à savoir: spécification des besoins, conceptualisation, formalisation, implémentation, test & évolution de l'ontologie.

1.3 CONSTRUCTION D'UNE ONTOLOGIE D'APPLICATION DANS LE CADRE DE L'EAI

Nous avons développé l'ontologie d'application en suivant les étapes du processus proposé, nous avons commencé par la phase de spécification qui consiste à établir un document de spécification des besoins servant à décrire l'ontologie à construire à travers les cinq aspects à savoir le domaine de connaissance, l'objectif, les utilisateurs, les sources d'information et la porté de l'ontologie. Ensuite, nous avons organisé et structuré les connaissances acquises en utilisant des représentations intermédiaires semi formels qui sont faciles à comprendre et indépendantes de tout langage d'implémentation. Après, nous avons utilisé le formalisme des logiques de description pour représenter l'ontologie d'application dans un langage formelle et finalement nous avons implémenté et testé l'ontologie en utilisant les outils PROTÉGÉ OWL et le raisonner RACER.

2 EVOLUTION & PERSPECTIVES

Conscient que la spécification et la mise en œuvre d'une architecture basée sur les ontologies afin d'intégrer les applications de l'entreprise est une tâche complexe, nous nous sommes limités seulement, dans ce mémoire, au développement de l'ontologie d'application qui s'inscrit dans le cadre de l'EAI et qui fera l'objet de notre travail. De ce fait, plusieurs limites et perspectives se dégagent de ce mémoire. Nous citons dans la suite, celles que nous croyons les plus importantes.

2.1 EVOLUTION DU SCÉNARIO DE COMMUNICATION PROPOSÉ

Les scénarios de communication que nous avons proposés sont incomplets et dénués de toutes informations détaillées sur l'acheminement des messages entre les applications, que ce soit, sur la structure interne des modules logiciels adjoints aux applications. L'objectif, donc, derrière la proposition des scénarios de communication est de montrer l'effet de l'ajout des ontologies au sein du système d'intégration et nous aider à dégager les termes de l'ontologie afin de faciliter sa construction.

Des enrichissements et améliorations doivent être apportés à ces scénarios afin de garantir la réactivité et flexibilité du système d'intégration, et d'avoir une interopérabilité sémantique entre ses applications.

2.2 EVOLUTION DU PROCESSUS DE DÉVELOPPEMENT D'ONTOLOGIES PROPOSÉ

Etendre le processus de construction d'une ontologie d'application à d'autres usages, c'est-à-dire, en ne se limitant plus du cadre dans lequel le processus sera créé, mais en se basant sur les phases du processus proposé afin d'aboutir à un autre qui sera adéquat et conforme à différents domaines autres que l'EAI.

2.3 EVOLUTION DE L'ONTOLOGIE D'APPLICATION

L'ontologie d'application est construite, il nous restera à suivre son évolution, c'est-à-dire les nouveaux concepts à ajouter dans la partie terminologique (TBOX) de l'ontologie. Le résultat de cette étape sera une nouvelle ontologie avec une nouvelle hiérarchie de concepts. Cet aspect devra être entendu par l'utilisation du raisonnement par classification qui est l'un des mécanismes de raisonnements de base pour les logiques de descriptions.

ANNEXE

```
<?xml version="1.0"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="http://www.owl-ontologies.com/Mémoire.owl#"
 xml:base="http://www.owl-ontologies.com/Mémoire.owl">
 <owl:Ontology rdf:about=""/>
 <owl:Class rdf:ID="Date"/>
 <owl:Class rdf:ID="Domaine-description"/>
 <owl:Class rdf:ID="Activité-choix">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Activité-complexe"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Créateur"/>
 <owl:Class rdf:ID="Application-secteur">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Application-domaine"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Activité-compétitive">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Activité-complexe"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Service-vol">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Service-voyage"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Effet-calculé">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Paramètre-calculé"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Input">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Paramètre"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="XDR-description">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Application-structure"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:about="#Activité-complexe">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Activité"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Application-modèle"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Description-fonctionnelle">
  <rdfs:subClassOf rdf:resource="#Domaine-description"/>
 </owl:Class>
 <owl:Class rdf:ID="Méthode-structure"/>
 <owl:Class rdf:ID="Vente-jeux">
  <rdfs:subClassOf rdf:resource="#Description-fonctionnelle"/>
 </owl:Class>
 <owl:Class rdf:ID="Service-Hotel">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Service-voyage"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Application-information">
  <rdfs:subClassOf rdf:resource="#Application-domaine"/>
 </owl:Class>
 <owl:Class rdf:ID="Vol">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Description-non-fonctionnelle"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Effet">
  <rdfs:subClassOf rdf:resource="#Paramètre"/>
 </owl:Class>
 <owl:Class rdf:ID="Précondition">
  <rdfs:subClassOf rdf:resource="#Paramètre"/>
 </owl:Class>
 <owl:Class rdf:ID="Hotel">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Description-non-fonctionnelle"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Service-véhicule">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Service-voyage"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="E-commerce">
  <rdfs:subClassOf rdf:resource="#Application-secteur"/>
 </owl:Class>
 <owl:Class rdf:ID="Médecine">
  <rdfs:subClassOf rdf:resource="#Application-secteur"/>
 </owl:Class>
 <owl:Class rdf:ID="Femme">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="Personne"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:ID="Hotel-réserv">
  <rdfs:subClassOf rdf:resource="#Service-Hotel"/>
 </owl:Class>
 <owl:Class rdf:ID="Précondition-calculée">
  <rdfs:subClassOf rdf:resource="#Paramètre-calculé"/>
 </owl:Class>
 <owl:Class rdf:ID="Activité-atomique">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Activité"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#Application-modèle"/>
 </owl:Class>
 <owl:Class rdf:ID="Vol-confir">
  <rdfs:subClassOf rdf:resource="#Service-vol"/>
 </owl:Class>
 <owl:Class rdf:ID="Application"/>
 <owl:Class rdf:ID="Transportation"/>
 <owl:Class rdf:ID="Interface-structure"/>
 <owl:Class rdf:ID="Activité-répéter-jusque-à ">
  <rdfs:subClassOf rdf:resource="#Activité-complexe"/>
 </owl:Class>
 <owl:Class rdf:ID="Séquence-activité">
  <rdfs:subClassOf rdf:resource="#Activité-complexe"/>
 </owl:Class>
 <owl:Class rdf:ID="IDL-description">
  <rdfs:subClassOf rdf:resource="#Application-structure"/>
 </owl:Class>
 <owl:Class rdf:ID="Hotel-fairepaym">
  <rdfs:subClassOf rdf:resource="#Service-Hotel"/>
 </owl:Class>
 <owl:Class rdf:ID="Application-comportement"/>
 <owl:Class rdf:ID="Input-calculé">
  <rdfs:subClassOf rdf:resource="#Paramètre-calculé"/>
 </owl:Class>
 <owl:Class rdf:ID="Output-structure"/>
 <owl:Class rdf:ID="Vol-réserv">
  <rdfs:subClassOf rdf:resource="#Service-vol"/>
 </owl:Class>
```

```
<owl:Class rdf:ID="Hotel-confir">
  <rdfs:subClassOf rdf:resource="#Service-Hotel"/>
</owl:Class>
<owl:Class rdf:ID="Education">
  <rdfs:subClassOf rdf:resource="#Application-secteur"/>
</owl:Class>
<owl:Class rdf:about="#Description-non-fonctionnelle">
  <rdfs:subClassOf rdf:resource="#Domaine-description"/>
</owl:Class>
<owl:Class rdf:ID="Vente-livres">
  <rdfs:subClassOf rdf:resource="#Description-fonctionnelle"/>
</owl:Class>
<owl:Class rdf:ID="Output">
  <rdfs:subClassOf rdf:resource="#Paramètre"/>
</owl:Class>
<owl:Class rdf:ID="Output-calculé">
  <rdfs:subClassOf rdf:resource="#Paramètre-calculé"/>
</owl:Class>
<owl:Class rdf:about="#Personne">
  <rdfs:subClassOf rdf:resource="#Description-non-fonctionnelle"/>
</owl:Class>
<owl:Class rdf:ID="Homme">
  <rdfs:subClassOf rdf:resource="#Personne"/>
</owl:Class>
<owl:Class rdf:ID="Produit"/>
<owl:Class rdf:ID="Service-assurance">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Service-voyage"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Activité">
  <rdfs:subClassOf rdf:resource="#Application"/>
</owl:Class>
<owl:Class rdf:ID="Input-structure"/>
<owl:Class rdf:ID="WSDL-description">
  <rdfs:subClassOf rdf:resource="#Application-structure"/>
</owl:Class>
<owl:Class rdf:about="#Service-voyage">
  <rdfs:subClassOf rdf:resource="#Description-fonctionnelle"/>
</owl:Class>
<owl:Class rdf:ID="Niveau"/>
<owl:ObjectProperty rdf:ID="Avoir-secteur">
  <rdfs:range rdf:resource="#Application-secteur"/>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:domain rdf:resource="#Application-domaine"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Dispose-de">
  <rdfs:domain rdf:resource="#E-commerce"/>
  <rdfs:range rdf:resource="#Produit"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="S-éxecute-comme">
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Activité-répéter-jusque-à "/>
        <owl:Class rdf:about="#Activité-choix"/>
        <owl:Class rdf:about="#Séquence-activité"/>
        <owl:Class rdf:about="#Activité-compétitive"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Activité-complexe"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Avoir-output-calculé">
  <rdfs:domain rdf:resource="#Activité-complexe"/>
  <rdfs:range rdf:resource="#Output-calculé"/>
</owl:ObjectProperty>
  <rdfs:range rdf:resource="#Application-information"/>
  <rdfs:domain rdf:resource="#Créateur"/>
<owl:ObjectProperty rdf:ID="Décrit">
  <rdfs:domain rdf:resource="#E-commerce"/>
  <rdfs:range rdf:resource="#Domaine-description"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Avoir-effet">
  <rdfs:range rdf:resource="#Effet"/>
  <rdfs:domain rdf:resource="#Activité"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Ensembe-de">
  <rdfs:domain rdf:resource="#Application-comportement"/>
  <rdfs:range rdf:resource="#Activité"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Description-type">
  <rdfs:domain rdf:resource="#Domaine-description"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Description-fonctionnelle"/>
        <owl:Class rdf:about="#Description-non-fonctionnelle"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Réalise">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="Réalisée-par"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#Activité-complexe"/>
  <rdfs:domain rdf:resource="#Activité-atomique"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Avoir-précondition-calculée">
  <rdfs:domain rdf:resource="#Activité-complexe"/>
  <rdfs:range rdf:resource="#Précondition-calculée"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Se-réfère-à ">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Paramètre"/>
        <owl:Class rdf:about="#Paramètre-calculé"/>
        <owl:Class rdf:about="#Description-fonctionnelle"/>
        <owl:Class rdf:about="#Description-non-fonctionnelle"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Activité"/>
        <owl:Class rdf:about="#Activité-complexe"/>
        <owl:Class rdf:about="#Paramètre"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="produit">
  <rdfs:range rdf:resource="#Application-comportement"/>
  <rdfs:domain rdf:resource="#Application"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="produit-par"/>
  </owl:inverseOf>
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Crée">
  <owl:inverseOf>
    <owl:FunctionalProperty rdf:ID="Créée-par"/>
  </owl:inverseOf>
```

```
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalPro
perty"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-structure">
    <rdfs:range rdf:resource="#Application-structure"/>
    <rdfs:domain rdf:resource="#Application"/>
    <owl:inverseOf>
      <owl:InverseFunctionalProperty rdf:ID="Structure-de"/>
    </owl:inverseOf>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/
>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-info">
    <rdfs:range rdf:resource="#Application-information"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/
>
    <rdfs:domain rdf:resource="#Application-domaine"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-paramètre">
    <rdfs:range rdf:resource="#Paramètre"/>
    <rdfs:domain rdf:resource="#Activité"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-input">
    <rdfs:range rdf:resource="#Input"/>
    <rdfs:domain rdf:resource="#Activité"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-output">
    <rdfs:range rdf:resource="#Output"/>
    <rdfs:domain rdf:resource="#Application"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-input-calculé">
    <rdfs:range rdf:resource="#Input-calculé"/>
    <rdfs:domain rdf:resource="#Activité-complexe"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#produit-par">
    <owl:inverseOf rdf:resource="#produit"/>
    <rdfs:range rdf:resource="#Application"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalPro
perty"/>
    <rdfs:domain rdf:resource="#Application-comportement"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Mode-livraison">
    <rdfs:domain rdf:resource="#E-commerce"/>
    <rdfs:range rdf:resource="#Transportation"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/
>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-paramètre-calculé">
    <rdfs:range rdf:resource="#Paramètre-calculé"/>
    <rdfs:domain rdf:resource="#Activité-complexe"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#Réalisée-par">
    <rdfs:range rdf:resource="#Activité-atomique"/>
    <owl:inverseOf rdf:resource="#Réalise"/>
    <rdfs:domain rdf:resource="#Activité-complexe"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Participe">
    <rdfs:range rdf:resource="#Niveau"/>
    <rdfs:domain rdf:resource="#Application-information"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/
> </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-précondition">
    <rdfs:range rdf:resource="#Précondition"/>
    <rdfs:domain rdf:resource="#Activité"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-domaine">
    <rdfs:range rdf:resource="#Application-domaine"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProp
erty"/>
    <rdfs:domain rdf:resource="#Application"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Avoir-effet-calculé">
    <rdfs:domain rdf:resource="#Activité-complexe"/>
    <rdfs:range rdf:resource="#Effet-calculé"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="Est-un-modèle-de">
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Activité-compétitive"/>
          <owl:Class rdf:about="#Séquence-activité"/>
          <owl:Class rdf:about="#Activité-choix"/>
          <owl:Class rdf:about="#Activité-répéter-jusque-à "/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
    <rdfs:range rdf:resource="#Activité-complexe"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:ID="Jour">
    <rdfs:domain rdf:resource="#Date"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProp
erty"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Text-description">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Application"/>
          <owl:Class rdf:about="#Application-domaine"/>
          <owl:Class rdf:about="#Paramètre"/>
          <owl:Class rdf:about="#Paramètre-calculé"/>
          <owl:Class rdf:about="#Application-structure"/>
          <owl:Class rdf:about="#Domaine-description"/>
          <owl:Class rdf:about="#Créateur"/>
          <owl:Class rdf:about="#Application-comportement"/>
          <owl:Class rdf:about="#Interface-structure"/>
          <owl:Class rdf:about="#Méthode-structure"/>
          <owl:Class rdf:about="#Niveau"/>
          <owl:Class rdf:about="#Output-structure"/>
          <owl:Class rdf:about="#Produit"/>
          <owl:Class rdf:about="#Transportation"/>
          <owl:Class rdf:about="#Input-structure"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:ID="Type">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProp
erty"/>
    <rdfs:domain>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Paramètre"/>
          <owl:Class rdf:about="#Paramètre-calculé"/>
        </owl:unionOf>
      </owl:Class>
    </rdfs:domain>
  </owl:DatatypeProperty>
```

```
owl:DatatypeProperty rdf:ID="Nom">
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   <rdfs:domain>
    <owl:Class>
     <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Application"/>
      <owl:Class rdf:about="#Application-domaine"/>
      <owl:Class rdf:about="#Paramètre"/>
      <owl:Class rdf:about="#Paramètre-calculé"/>
      <owl:Class rdf:about="#Application-structure"/>
      <owl:Class rdf:about="#Domaine-description"/>
      <owl:Class rdf:about="#Créateur"/>
      <owl:Class rdf:about="#Application-comportement"/>
      <owl:Class rdf:about="#Interface-structure"/>
      <owl:Class rdf:about="#Méthode-structure"/>
      <owl:Class rdf:about="#Input-structure"/>
      <owl:Class rdf:about="#Output-structure"/>
      <owl:Class rdf:about="#Transportation"/>
      <owl:Class rdf:about="#Produit"/>
      <owl:Class rdf:about="#Niveau"/>
     </owl:unionOf>
    </owl:Class>
   </rdfs:domain>
 </owl:DatatypeProperty>
 <owl:SymmetricProperty rdf:ID="Avoir-sous-activités">
   <owl:inverseOf rdf:resource="#Avoir-sous-activités"/>
   <rdfs:domain rdf:resource="#Activité"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
   <rdfs:range rdf:resource="#Activité"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProper
ty"/>
 </owl:SymmetricProperty>
 <owl:FunctionalProperty rdf:ID="Date-info">
   <rdfs:domain rdf:resource="#Application-information"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
   <rdfs:range rdf:resource="#Date"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="Avoir-interface-struct">
   <rdfs:range rdf:resource="#Interface-structure"/>
   <rdfs:domain rdf:resource="#IDL-description"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:about="#Créée-par">
   <rdfs:domain rdf:resource="#Application-information"/>
   <owl:inverseOf rdf:resource="#Crée"/>
   <rdfs:range rdf:resource="#Créateur"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="Mois">
   <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
   <rdfs:domain rdf:resource="#Date"/>
 </owl:FunctionalProperty>
 <owl:FunctionalProperty rdf:ID="Année">
   <rdfs:domain rdf:resource="#Date"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
   <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
 </owl:FunctionalProperty>
 <
 <owl:InverseFunctionalProperty rdf:about="#Structure-
de">
   <rdfs:range rdf:resource="#Application"/>
   <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectPro
perty"/>
   <owl:inverseOf rdf:resource="#Avoir-structure"/>
   <rdfs:domain rdf:resource="#Application-structure"/>
 </owl:InverseFunctionalProperty>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 2.1, Build 284)
http://protege.stanford.edu -->
```

RÉFÉRENCES

[1] *« L'accès transparent aux applications : l'eai propose une véritable architecture d'intégration »,* http://www.grd-publications.com/art/ls043/ls043110.htm.

[2] *Panorama d'une infrastructure EAI, Auteur : David ROUSSE (DSI/BEST) Référence : NOT03K012DSI Date : 31/07/2003.*

[3] *R. Hailstone, « Integrations strategies : the start of convergence », IDC, 2003.*

[4] *X. Yang, G. Yu & G. Wang, « Efficient mapping integrity constraints from relational database to XML document », A. Caplinskas and J. Eder édition, ADBIS, LNCS2151, pp 338-351, Springer Verlag Berlin Heidelberg, 2001.*

[5] *D. Serain , « Enterprise application integration », 3ème édition, Dunod, Paris 2001.*

[6] *S. Wong, « Web services: the next evolution of application integration », Web services architecture, 2002.*

[7] *J. Chauvet, « Services Web avec SOAP, WSDL, UDDI, ebXML », editions Eyrolles 2002.*

[8] *R. Altman, « Intégration globale », Gartner Group, février 2001.*

[9] *« EAI au cœur de l'e-business», Octo technology, Edition Eyrolle, Novembre 2000.*

[10] *A. Gruden, P. Strannergrad, « BPM : the next wave », eAI Journal, Janvier, 2003.*

[11] *Dante Consulting, « Web service overview », Dante Consulting, 2003.*

[12] *J. T. Pollok, « The big issue : interoperability vs integration », eAI Journal, 2001.*

[13] *D. S. Linthicum, « Enterprise application integration », Mercator, 2001.*

[14] *J. Schmidt, « Enabling next generation enterprise », eAI Journal,2000.*

[15] *E. Stohr, J. V. Nickerson, « Intra enterprise integration: methods and directions», Addision Wesley, 2001.*

[16] *Panorama EAI : les architectures en concurrence », Dreamsoft,* http://solutions.Journaldunet.com/0202/020211_comparo_eai_archi.shtml.

[17] *Mentions Légales, Le livre blanc EAI est la propriété exclusive de Mediadev. La copie intégrale ou partielle de son contenu, de ses illustrations, références ou index est interdite sans autorisation préalable de Mediadev Page N01,2,7,8,9,10* www.dsi.cnrs.fr/ref-partage/_Documents/EAI/livre_blancMEDIADEV.pdf

[18] *J. C. Lutz, « EAI : architecture paterns », eAI Journal, Mars 2000.*

[19] *N. Erasala, D. C. Yen, T. M. Rajkumer, « Enterprise application integration in the electronic commerce world », Elsevier, CSI, 25 (69-82), 2003.*

[20] *Berners-Lee, T., Hendler, J., et Lasilla, O. (2001). The Semantic Web. Scientific American, 284*

[21] *Ontologies et Web services – Rapport du travail d'intérêt personnel encadré, Hanoï, juillet 2005 Httpwww..com/ tipe-phan_quang_trung_tien*

[22] *Rector A. (1998). Thesauri and formal classi_cations : Terminologies for people and machines. Methods of Information in Medicine, 37(4_5), 501_509.*

[23] *N. Aussenac-Gilles & A. Busnel. "Méthode de construction à partir du texte d'une ontologie du domaine de l'industrie de la fibre de verre". Rapport Interne IRIT/2002-11-R. Avril 2002.*

[24] *James Hendler. 2001. IEEE Intelligent systems. Agents and the Semantic Web.*

[25] *Raul Corazzon. 2003. Descriptive and Formal Ontology. http://www.formalontology.it/.*

[26] *Noy, Natalya, F.; and McGuinness, Deborah, L., "Ontology Development 101: A Guide to Creating Your First Ontology", Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March*

[27] *What is an Ontology?. http://mged.sourceforge.net/ontologies/index.php*

[28] *M. Ikeda, K.Seta et al., Task Ontology Makes It Easier To Use Authoring Tools, Proc. of IJCAI-97, pp.342-347, 1997.*

[29] *F. Furst, "L'ingénierie ontologique". Rapport de recherche N°02-07. 2002.*

[30] *Gomez Pérez A., Benjamins V.R. (1999) "Overview of Knowledge Sharing and Reuse Components : Ontologies and problem-Solving Methods". Proceeding og the IJCAI-99 workshop on Ontologies and problem-Solving Methods (KRR5), Stockholm (Suède), pp. 1.1-1.15.*

[31] *Hernandez, N., Ontologies de domaine pour la modélisation du contexte en recherche d'inrofmation. Thèse de doctorat, Université de Toulouse, 2005.*

[32] *B. Bachimont, Arts et sciences du numérique : Ingénierie des connaissances et critique de la raison computationnelle, Mémoire d'Habilitation à Diriger des Recherches, Université de Technologie de Compiègne, 2004.*

[33] *Terminologie et intelligence artificielle (actes du colloque de Nantes, 10-11 mai 1999)*

[34] *S. Staab, A. Maedche, Axioms are objects too: Ontology engineering beyond the modeling of concepts and relations, Research report 399, Institute AIFB, Karlsruhe, 2000.*

[35] *Guarino, N., « Formal Ontology and Information Systems», Formal Ontology in Information Systems. IOS Press, 1998.*

[36] *M.Uschold & M.Grüninger, "ONTOLOGIES: Principles, Methods and Applications". Knowledge Engineering Review. 1996*

[37] *Gestion de l'évolution d'une ontologie : méthodes et outils pour un référencement sémantique évolutif fondé sur une analyse des changements entre versions de l'ontologie Proposition de recherche doctorale en informatique cognitive (DIC 9410)*

[38] *B. Bachimont, J. Charlet & R. Troncy. "Ontologies pour le Web Sémantique". Action spécifique 32 CNRS / STIC Web sémantique Rapport final. 2003*

[39] *T.R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing". International Journal of Human Computer Studies. 1995*

[40] *Sowa, J.F. "Conceptual structures: information processing in mind and machine", Addison-Wesley, 1984.*

[41] *M. Buchheit, f. Donini, and A. Shaerf" Decidable reasoning in terminological knowledge representation systems". Journal of artificial intelligence research,1: 109-138, 1993.*

[42] *Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., et al. (2001). Ontology -based integration of information - a survey of existing approaches. Paper presented at the IJCAI Workshop on Ontologies and Information Sharing.*

[43] *Uschold, M., et Gruninger, M. (2002). Creating semantically integrated communities on the World Wide Web. Honolulu: Semantic Web Workshop.*

[44] *WebOnt. (2004b). OWL Web Ontology Language Use Cases and Requirements, from http://www.w3.org/TR/2004/REC-owl-test-20040210/*

[45] *Oberle, D., Volz, R., Motik, B., et Staab, S. (2004). An extensible ontology software environment. In S. Staab et R. Studer (Eds.), Handbook on Ontologies (pp. 299-320): Springer Verlag*

[46] *A. Napoli, " une introduction aux logiques de descriptions" N° 3314, Décembre 1997.*

[47] *I. Horrocks "DAML+OIL : A Description Logic for the Semantic Web". IEEE Data Engineering Bulletin Num 1, vol.25, pp 4-9 , 2002*

[48] *O. M. Drews "Raisonnement classificatoire dans une représentation à objets multipoints de vue". Thèse de doctorat, Université Joseph Fourier, Grenoble. pp 30-51. Mars 1993.*

[49] *I. Horrocks, F. Baader, & U. SattlerL. "Description Logics as Ontology Languages for the Semantic Web". In Festschrift in hoor of Jorg Siekmann, LNAI.Springer-Verlag. 2003.*

[50] *Gómez-Pérez Asunción, Fernández-López Mariano, Corcho Oscar, OntoWeb: Ontology -based information exchange for knowledge management and electronic commerce, Deliverable 1.3: A survey on ontology tools, IST-2000-29243, 31 st May, 2002*

[51] *Resource Description Framework (RDF). http://www.w3.org/RDF/.*

[52] *RDF-Schema. http://www.w3.org/TR/rdf-schema/*

[53] *The DARPA Agent Markup Language. http://www.daml.org/.*

[54] *Ian Horrocks. DAML+OIL: a Reason-able Web Ontology Language.*

[55] *Ontology Web Language (OWL). http://www.w3.org/OWL/*

[56] *Bechhofer, S.; Horrocks, I.; Goble, C.; and Stevens, R., "OILEd: a Reason-able Ontology Editor for the Semantic Web", In Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, September 19-21, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.*

[57] *Sure, Y.; Erdmann, M.; Angele, J.; Staab, S.; Studer, R.; and Wenke, D., "OntoEdit: Collaborative Ontology Engineering for the Semantic Web", In Proceedings of the International Semantic Web Conference 2002 (ISWC 2002) , Sardinia, Italia, June 9- 12 2002*

[58] *Farquhar; Fikes, R.; and Rice, J., "The Ontolingua Server: A Tool for Collaborative Ontology Construction", In Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada, 44.1-44.19, 1996, http://www.ksl. stanford.edu/software/ontolingua/].*

[59] *Swartout, B.; Ramesh, P.; Knight, K.; and Russ, T., "Toward Distributed Use of Large-Scale Ontologies", In Symposium on Ontological Engineering of AAAI, Stanford, California, March, 1997.*

[60] *R. Driouche, Z. Boufaida, F. Kordon, "An Ontology Based Architecture for Integrating Enterprise Applications", Modelling, Simulation, Verification and Validation of Enterprise Information Systems'06 Workshop, Cyprus, Paphos, INSTICC Press, pp. 26-37, 20*

[61] *M. Hemam, Z. Boufaida "An Ontology Development Process for the Semantic Web", EKAW'04, 14th International Conference on Knowledge Engineering and Knowledge Management Whittlebury Hall, Northamptonshire, UK, 5-8 October 2004*

[62] *Themistocleous, M., Irani, Z., Kuljis, J., Love, P.: Extending the information system lifecycle through enterprise application integration: a case study experience. In: Pro. of the 37th International Conference on System Sciences, Hawaii (2004)*

[63] *Linthicum, D., S.,. (ed.): Next generation application integration. , Addison-Wesley, (2004)*

[64] *Chauvet, J., (ed.) : Services Web avec SOAP, WSDL, UDDI, ebXML. Eyrolles (2002)*

[65] *Driouche, R Boufaida, Z. Kordon, F. 2006 " An Enterprise application integration architecture supporting ontology based approach for B2B collaboration", soumis à International Journal on interoperability in business information systems.*

[66] *Driouche R, Touati K, Saada-Khelkhal F, Abdemouche H., " Vers une Extension du Scénario de Collaboration EbXML par les Agents Mobiles et les Web Services" Conférence Internationale sur la Productique, IEEE. Tlemcen, 2005.*

[67] *Maedche, A., Motik, B., Silva, N., Volz, R. : MAFRA : A MApping FRAmework for Distributed Ontologies in the Semantic Web. In: Proc. Of the ECAI Workshop Knowledge Transformation, Lyon, France (2002)*

[68] *Linthicum, D., S.,. (ed.): Next generation application integration. , Addison-Wesley, (2004)*

[69] *H. Guergour, R. Driouche, Z. Boufaïda. , "An Approach for Application Ontology Building and Integration Enactment", Semantic Web Applications and Perspectives 3rd Italian Semantic Web Workshop Pisa University 19-20 December, Italy, 2006*

[70] *D. Serain, « Enterprise application integration », 3ème édition, Dunod, Paris 2001.*

[71] *Natalya F. Noy et Deborah L. McGuinness « Développement d'une ontologie 101: Guide pour la création de votre première ontologie » Université de Stanford, Stanford, CA, 94305. Traduit de l'anglais par Anila Angjeli, BnF, Bureau de normalisation document.*

[72] *N. Noy, R. W. Fergeson et M. A. Musen. « The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. » In Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW'00), 2000.*

[73] *Raphaël Troncy « Formalisation des connaissances documentaires et des connaissances conceptuelles à l'aide d'ontologies : application à la description de documents audiovisuels » THÈSE pour l'obtention du Doctorat de l'université Joseph Fourier – Gren*

[74] *R. Driouche, H. Guergour, Z. Boufaïda. , "An Approach Based Ontology for Semantically Enriched Communication Scenario » In Proceedings of the International Symposium on programming and Systems (ISPS'07), USTHB- 7-9 May, Algiers, Algeria 2007.*