الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 – Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de Fin d'études Master

Filière : Informatique

Option : Informatique Académique

Thème :

La Génération Automatique des Ontologies à partir des Diagrammes de classes UML

Encadré Par :

Mme : Souiou Wafa

Présenté par :

Lezghed Amir

Juin 2017

« Au nom d'Allah, le Clément, le Miséricordieux »

Résumé

Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle ***modélisation*** ; son produit est un ***modèle.***

UML est un langage standardisé par l'OMG qui est devenue nécessaire pour la modélisation, il permet de modéliser des choses qui n'auraient pas pu l'être avant. Les utilisateurs de la plupart des autres méthodes et langages de modélisation auront avantage à utiliser UML, puisqu'elle supprime toutes les différences non nécessaires de notation et de terminologie qui obscurcissent les similarités de bases de ces différentes approches. Cependant, UML ne permet pas la définition d'une sémantique formelle ce qui entrave l'inférence automatique sur des modèles UML.

Dans l'autre coté les ontologies fournissent la base de la sémantique critique, dans le contexte de l'information et la science d'information. L'ontologie définit un ensemble de primitives représentatives en utilisant un domaine de connaissance qui sera modélisé.

Pour cela, les travaux sont orientés vers les ontologies qui sont devenues l'épine dorsale du web sémantique qui décrit formellement en utilisant un langage standard appelé OWL.

L'objectif de notre travail consiste à implémenter un modèle qui assure la génération automatique des ontologies a partir d'un diagramme de classe d'un système d'information a fin de tirer profit de l'expressivité visuelle du langage de notation UML et la puissance des ontologies.

Mots clés : modélisation, UML, diagrammes de classe, ontologie, Système d'information.

TABLE DES MATIERES

Liste des figures

Liste des tableaux

Liste des abréviations et acronymes

OMG : Object Management Group

OMT : object modeling technique

OOD : Object Oriented Development

OOSE : Object-oriented software engineering

KIF : Knowledge Interchange Format

OIL : Ontology Interchange Language

OWL : Ontology Web Language

RDF : Resource Description Framework

RDF(S) : Resource Description Framework Schema

SKOS : Simple Knowledge Organization System

W3C : World Wide Web Consortium

XMI : XML Metadata Interchanger

XML : Extensible Markup Language

XSD : XML Schema Definition

API : Application programming interface

CVS : Concurrent Versioning System

HTML : Hypertext Markup Language

IDE : Integrated Development Environment

JVM : Java virtual machine

PHP : Personal Home Page

Introduction Générale

1. Contexte Général :

Dans de nombreux domaines d'activité utilisant des systèmes d'information, la connaissance est souvent représentée par des modèles UML notamment par des diagrammes de classes modélisant les entités propres au domaine.

La méthodologie que nous avons défini permet d'alléger et de guider le travail important que constitue la construction complète d'une ontologie a partir d'un diagramme de classe.

Notre premier objectif consiste à capturer ce savoir-faire par la construction d'ontologie permettant la modélisation des concepts UML ce qui permettra de :

- Profiter de l'expressivité visuelle du langage de notation UML et la puissance des ontologies;
- l'intégration Entre les langages de modélisation et l'obtention d'une plus grande valeur des captures d'information;
- bénéficier d'UML Afin d'avoir des modèles sur les ontologies pour faire des analyses et implémentations OWL pour tester la consistance des ontologies.

Pour y arriver, nous avons réalisé une étude générale sur les différents diagrammes UML et sur les ontologies, cela nous a permis de collecter les informations nécessaires concernant chacun d'eux et de connaitre toutes les possibilités et les concepts qu'on peut les transformer entre eux, afin de choisir une méthode pour faire la meilleure transformation.

2. Problématique :

UML est le langage orienté objet de modélisation unifié qui est devenu un standard important, dans l'autre coté les ontologies sont devenues l'épine dorsale du

web sémantique. L'utilisation des ontologies est importante pour de nombreuse technologies en pleine expansion car ils fournissent la base sémantique critique. Dans le contexte de l'informatique et sciences de l'information.

3. Motivation

Vue les avantages que présentent les ontologies par rapport à UML en terme d'extensibilité, de raisonnement et de formalisme, les travaux récents montrent qu'il est nécessaire de se diriger vers les ontologies afin d'intégrer une nouvelle vision de représentation formelle et partage de connaissances.

4. **Objectif** :

L'objectif de Notre travail consiste à implémenter un modèle qui assure la génération automatique des ontologies a partir d'un diagramme de classe d'un système d'information afin de tirer profit de l'expressivité visuelle du langage de notation UML et la puissance des ontologies.

5. Plan du mémoire :

Pour évoquer la problématique posée nous proposons d'organiser notre mémoire en quatre chapitres :

Partie 1 : Etat de l'art

Le chapitre 1 est consacré aux définitions des concepts fondamentaux relatifs aux entrepôts de données, tel que la modélisation multidimensionnelle.

Le chapitre 2 Ce chapitre comportera des généralités sur l'ontologie telle que les définitions de la notion d'ontologie, les différents composants d'une ontologie.

Partie 2 : Contribution

Le chapitre 3 présente notre conception en présentant notre architecture de système ainsi que notre algorithme de transformation des diagrammes UML (classe et séquence) vers l'ontologie.

Le chapitre 4 propose une mise en œuvre de notre travail, où l'on décrit L'environnement, le langage et les outils de développement ainsi que les interfaces du notre application.

Enfin, le mémoire s'achève sur une conclusion dans laquelle, nous résumons les travaux et les contributions détaillés dans ce mémoire.

Chapitre 1

UML

1 Introduction

La description de la programmation par objets a fait ressortir l'étendue du travail conceptuel nécessaire : définition des classes, de leurs relations, des attributs et méthodes, des interfaces, etc.

Pour programmer une application, il ne convient pas de se lancer tête baissée dans l'écriture du code : il faut d'abord organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle ***modélisation*** ; son produit est un ***modèle.***

Les spécifications fournies par la maîtrise d'ouvrage en programmation impérative étaient souvent floues : les articulations conceptuelles (structures de données, algorithmes de traitement) s'exprimant dans le vocabulaire de l'informatique, le modèle devait souvent être élaboré par celle-ci. L'approche objet permet en principe à la maîtrise d'ouvrage de s'exprimer de façon précise selon un vocabulaire qui, tout en transcrivant les besoins du métier, pourra être immédiatement compris par les informaticiens. En principe seulement, car la modélisation demande aux maîtrises d'ouvrage une compétence et un professionnalisme qui ne sont pas aujourd'hui répandus.

2 Histoire des modélisations par objets

Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements. Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (**Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE**, etc.), mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes :

- **OMT**[(1)] de James Rumbaugh (*General Electric*) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- **OOD**[(2)] de Grady Booch, définie pour le *Department of Defense*, introduit le concept de paquetage (*package*) ;

- **OOSE**[3] d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou *use cases*).**[1]**

Chaque méthode avait ses avantages et ses partisans. Le nombre de méthodes en compétition s'était réduit, mais le risque d'un éclatement subsistait : la profession pouvait se diviser entre ces trois méthodes, créant autant de continents intellectuels qui auraient du mal à communiquer.

Événement considérable et presque miraculeux, les trois gourous qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »). UML (***Unified Modeling Language***) est né de cet effort de convergence. L'adjectif *unified* est là pour marquer qu'UML unifie, et donc remplace**. [2]**

En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage. L'unification a progressé par étapes. En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, ***Unified Method*** *0.8 ;* en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot *méthode* par le mot *langage*, plus modeste). Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys, etc.) et UML 1.0 est soumis à **l'OMG**[4]. L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version d'UML en cours en 2008 est UML 2.1.1 et les travaux d'amélioration se poursuivent. **[1]**

UML est donc non seulement un outil intéressant, mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs du domaine, acteurs qui ont d'ailleurs contribué à son élaboration.

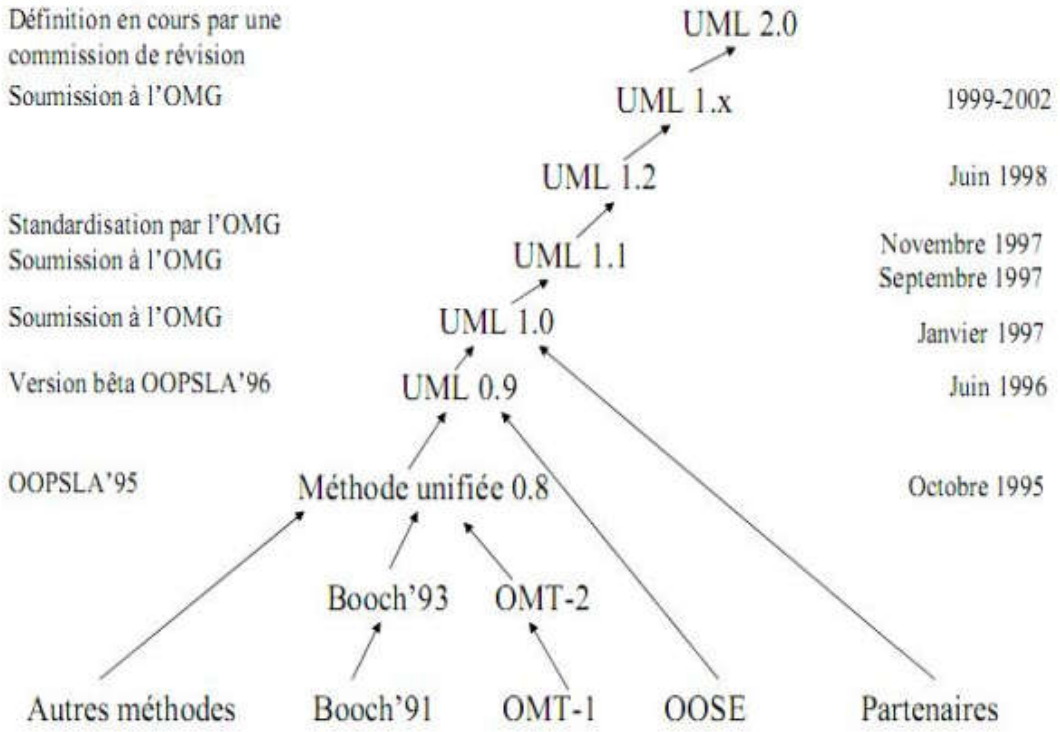


Figure 1.1 : l'évolution de l'UML **[3]**

3 UML

3.1 Définition

UML (le langage de modélisation unifie) est une notation permettent de modéliser un problème de façon standard.

Donc UML est un langage graphique du monde du génie logiciel a comme but la modélisation des données et des traitements.

UML est à présent un standard défini par Objet Management Groupe qui diffuse depuis novembre 2007 la version UML 2.1.2, et travaille a présent sur la version 2.2 [4].

3.2 Pourquoi l'UML ?

Après l'étude de quelques méthodes. On a trouvé que « UML » est la meilleure. Elle offre plusieurs avantage parmi nous citons :

- UML n'est pas une méthode ou un processus !
  - Si l'on parle de méthode d'objet pour UML, c'est par abus du langage !

- UML a été pensé pour permettre de modéliser les activités de l'entreprise.

➢ UML est support de communication (un langage universel).

- Sa notation graphique permet d'exprimer visuellement une solution objet.
- L'aspect formel de sa notation limite les ambigüités et l'incompréhension.
- Son aspect visuel facilite la comparaison et l'évaluation de solution.//

➢ Penser objet avec UML, pour concevoir objet.

- Pour penser et concevoir objet, il faut savoir prendre de la hauteur, jonglé avec des concepts abstraits indépendante du langage d'implémentation et des contraintes purement technique.
- UML comble une lacune importante des technologies. Il permet d'exprimer et d'élaborer des modèles objet, indépendamment de tous les langages de programmation.//

➢ UML : le chemin vers l'unification des processus.

- Guidée par les besoin des utilisateurs de système.
- Centre sur l'architecture logicielle.
- Interactive et incrémentation.//

3.3 Les points forts et faibles d'UML

3.3.1 Les points fort d'UML

➢ UML est un langage formel et normalisé.

- Gain de précision.
- Gage de stabilité.
- Encourage l'utilisation d'outils.

➢ UML et un support de communication performent.

- Il cadre l'analyse.
- Il facilite la compréhension des représentations abstraites complexes.
- Son caractère polyvalent et sa souplesse en font un langage universel **[5]**.

3.3.2 Les points faibles d'UML

- La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation :

  Même si l'Espéranto est une utopie, la nécessite de s'accorder sur des modes d'expression communs est vitale en informatique. UML n'est pas à l'origine des concepts objets, mais il constitue une étape majeure, car il unifie les différentes approches et donne une définition plus formelle.

- Le processus (non couvert par UML) est une autre clé de la réussite d'un projet Or, l'intégration d'UML dans un processus n'est pas triviale et améliorer un processus est une tache complexe et longue.

  Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais l'acceptabilité industrielle sont tout de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse objet performant et standard **[5]**.

3.4 Vue diagramme d'UML

UML propose une manière de décrire, le plus souvent graphiquement, le résultat des déférentes étapes de développement d'une applications, donc UML fournit un grand nombre de diagrammes pour représenter le système étude, selon déférentes points de vue.

Une vue est constitué d'un ou plusieurs diagrammes. On distingue deux types de vue :

3.4.1 **La vue structurel ou statique :** c'est-à-dire représentant le système physiquement. Elle a les diagrammes suivants :

- Diagramme de cas d'utilisation.
- Diagramme de classe.
- Diagramme d'objet.
- Diagramme de composant.
- Diagramme déploiement.

3.4.2 **La vue comportemental ou dynamique :** montrant le fonctionnement du système. Elle a les diagrammes suivants :

- Diagramme d'état-transition.
- Diagramme d'activité.

3.4.3 **Diagrammes d'interaction :** utilisé pour rendre compte de l'organisation spatiale des participants à l'interaction.

- Diagramme de séquence.
- Diagramme de collaboration.

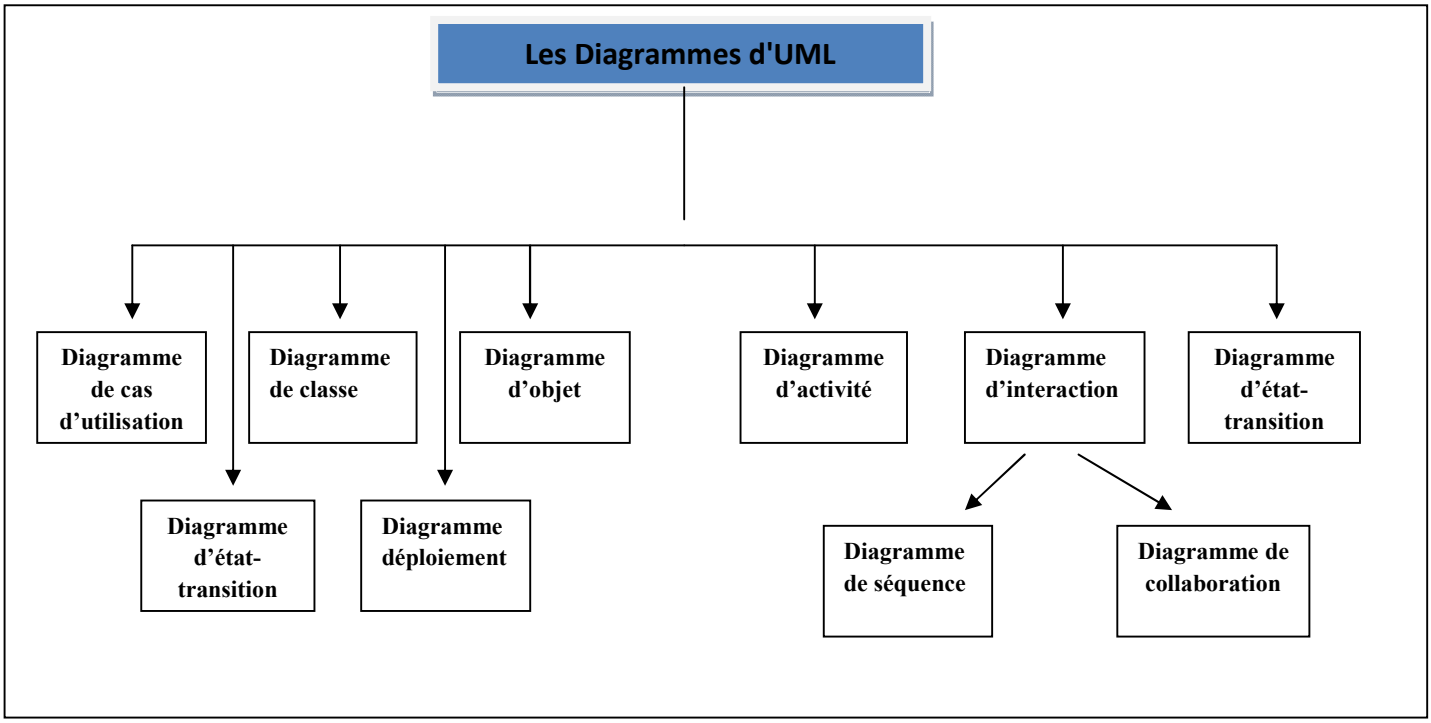


Figure 1.2 : les principaux vues d'UML

4 Les diagrammes d'UML

4.1 Diagramme de cas d'utilisation

Formalisé par « IVAR JACOBSON », un cas d'utilisation en anglais « use case » permet de mettre en évidence les relations fonctionnelles entre les acteurs et le système étudié. Le format de représentation d'un cas d'utilisation est complètement libre, mais UML propose un formalisme et des concepts issus de bonne pratique. Une séquence d'action réalisée par le système, représentée par une boite rectangulaire, produisant un résultat sur un acteur, appelé acteur principal, et ceci indépendamment de son fonctionnement interne **[6]**.

Il existe 3 types de relations standardisées entre cas d'utilisation :

- Relation d'inclusion : le cas d'utilisation de base incorpore un autre de façon obligatoire, le mot clé est « inclus »
- Relation d'extension :. Cette relation est utilisée pour indiquer que le cas d'utilisation source (à l'origine de la flèche), le mot clé est « extend ».

- Relation de généralisation / spécialisation : les cas d'utilisation descendant héritent de la description de leur parents communs.

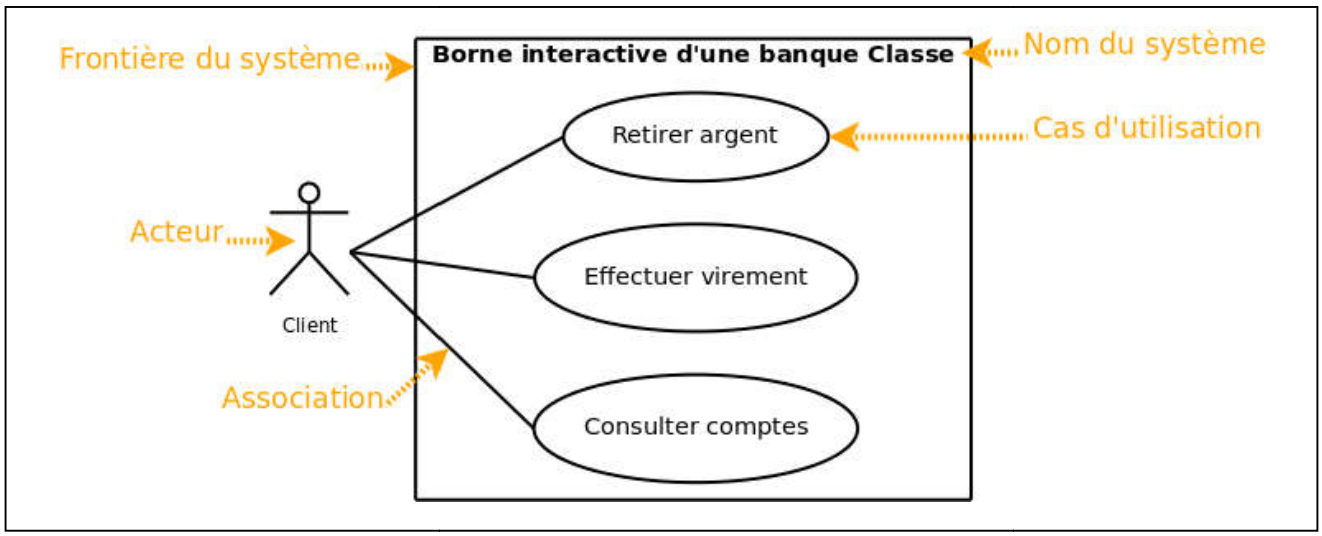


Figure 1.3 : exemple d'un diagramme de cas d'utilisation

4.2 Diagramme d'objet

Permet de représenter les instances des classes, c'est-à-dire des objets. Comme le diagramme de classes, il exprime les relations qui existent entre les objets, mais aussi l'état des objets, ce qui permet d'exprimer des contextes d'exécution. En ce sens, ce diagramme est moins général que le diagramme de classes.

Les diagrammes d'objets s'utilisent pour montrer l'état des instances d'objet avant et après une interaction, autrement dit c'est une photographie à un instant précis des attributs et objet existant. Il est utilisé en phase exploratoire **[8]**.

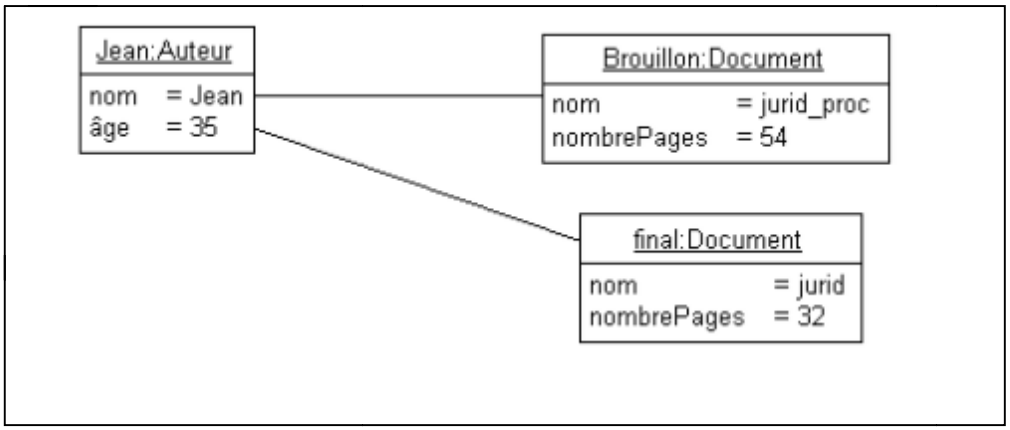


Figure 1.4 : exemple d'un diagramme d'objet

4.3 Diagramme de dépoilement

Est une vue statique qui sert à représenter l'utilisation de l'infrastructure physique par le système et la manière dont les **composants** du système sont répartis ainsi que leurs relations entre eux. Les éléments utilisés par un **diagramme de déploiement** sont principalement les **nœuds**, les **composants**, les **associations** et les **artefacts**. Les caractéristiques des ressources matérielles physiques et des supports de communication peuvent être précisées par stéréotype **[8]**.

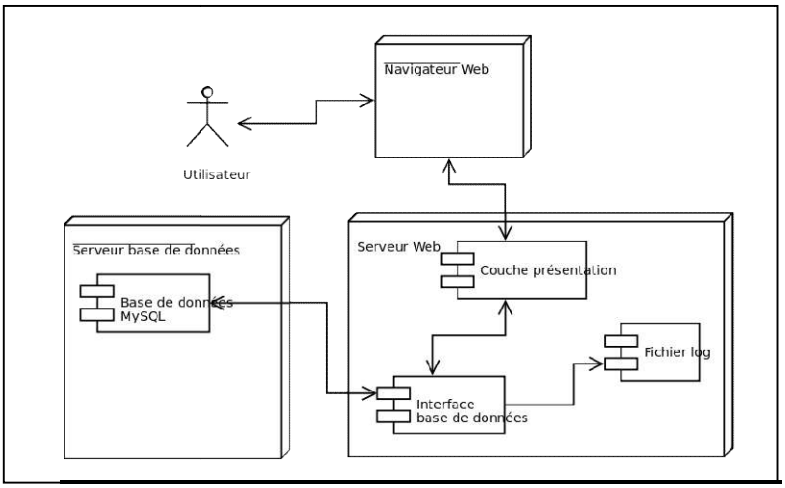


Figure 1.5 : exemple d'un diagramme de dépoilement

4.4 Diagramme de composant

Les diagrammes de composants permettent de décrire l'architecture physique et statique d'une application en termes de modules : fichiers sources, librairies, exécutables, etc. Ils montrent la mise en œuvre physique des modèles de la vue logique avec l'environnement de développement.

Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation et de mettre en évidence la réutilisation de composants.

Les composants peuvent être organisés en paquetages, qui définissent des sous-systèmes. Les sous-systèmes organisent la vue des composants (de réalisation) d'un système. Ils permettent de gérer la complexité, par encapsulation des détails d'implémentation **[5].**

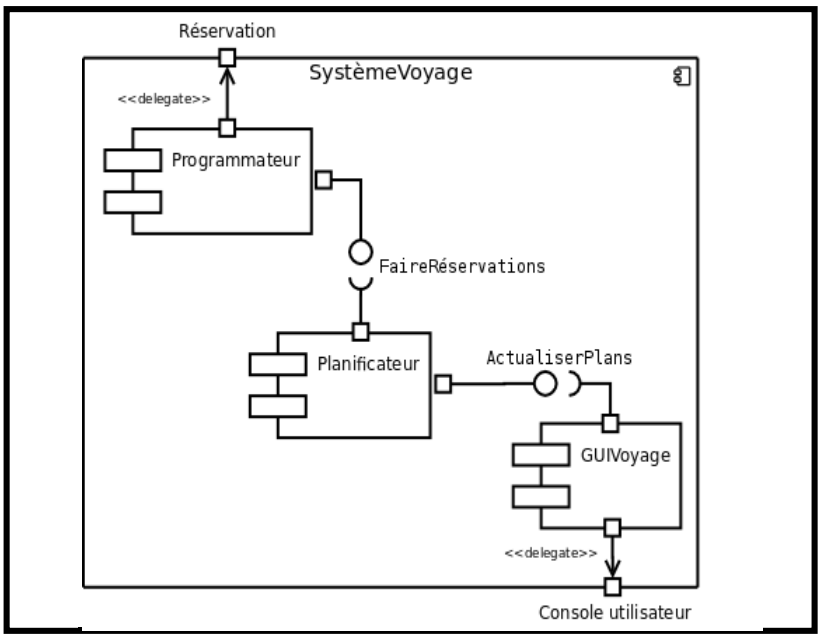


Figure 1.6 : exemple d'un diagramme de composant

4.5 Diagramme d'état transition

Est un schéma utilisé en génie logiciel pour représenter des automates déterministes. Il fait partie du modèle UML et s'inspire principalement du formalisme des "statecharts" et rappelle les "grafcets" des automates. S'ils ne permettent pas de comprendre globalement le fonctionnement du système, ils sont directement transposables en algorithme. Tous les automates d'un système s'exécutent parallèlement et peuvent donc changer d'état de façon indépendante **[9].**

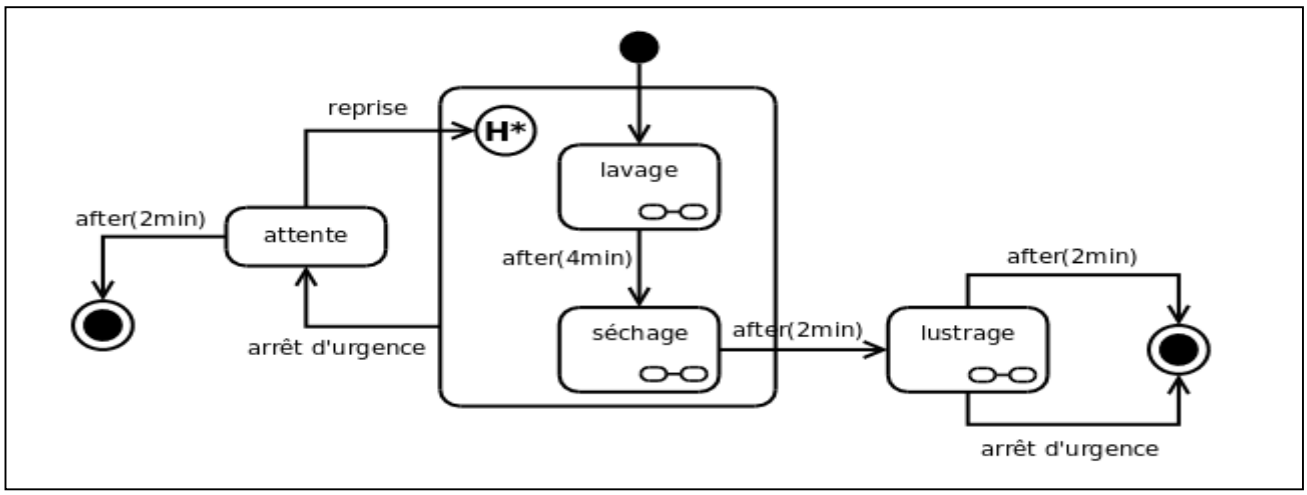


Figure 1.7 : exemple d'un diagramme d'état transition

4.6 Diagramme de collaboration

Un diagramme de collaboration est un diagramme d'interactions UML 1.x, représentation simplifiée d'un diagramme de séquence se concentrant sur les échanges de messages entre les objets, et où la chronologie n'intervient que de façon annexe.

Il consiste en un graphe dont les nœuds sont des objets et les arcs (numérotés selon la chronologie) les échanges entre ces objets.

Les diagrammes de collaboration ont été remplacés en UML 2.x par les diagrammes de communication **[8]**.

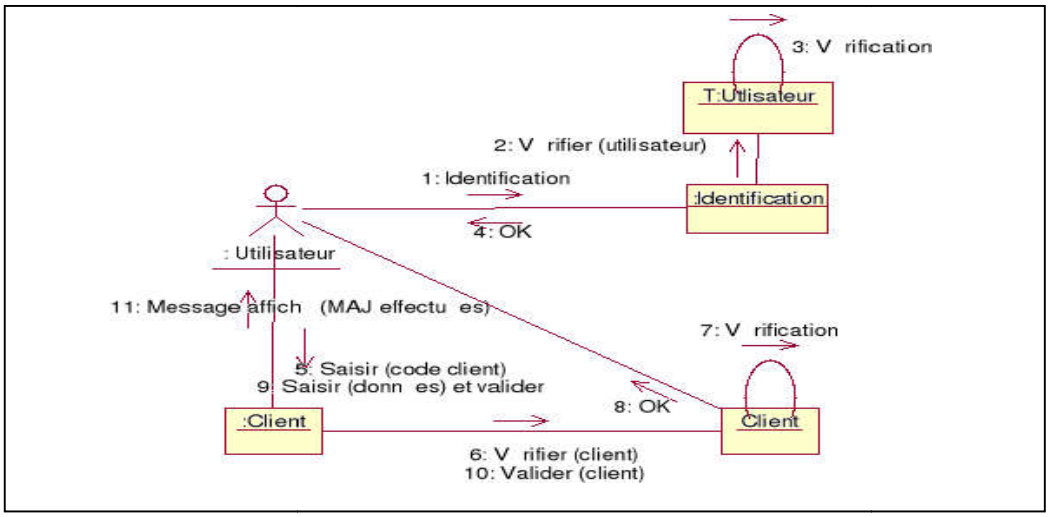


Figure 1.8 : exemple d'un diagramme de collaboration

4.7 Diagramme d'activité

La fiche descriptive d'un cas d'utilisation peut contenir plusieurs scénarios alternatifs et/ou d'exception. Il est alors difficile d'avoir une vision de l'ensemble des actions. Le diagramme d'activité est un moyen graphique pour donner cette vision d'ensemble.

Certaines personnes préfèrent le diagramme d'activités à la description textuelle. Pour ma part, je préfère commencer par une description textuelle qui donne des précisions que nous n'aurons pas dans le diagramme d'activité (des informations telles que les pré-conditions, le démarrage, les post-conditions, etc.).

Ensuite, pour les cas d'utilisation les plus complexes, un diagramme d'activité peut aider à y voir un peu plus clair. Cela peut même aider à trouver de nouvelles questions auxquelles on n'avait pas pensé jusque-là. **[7]**

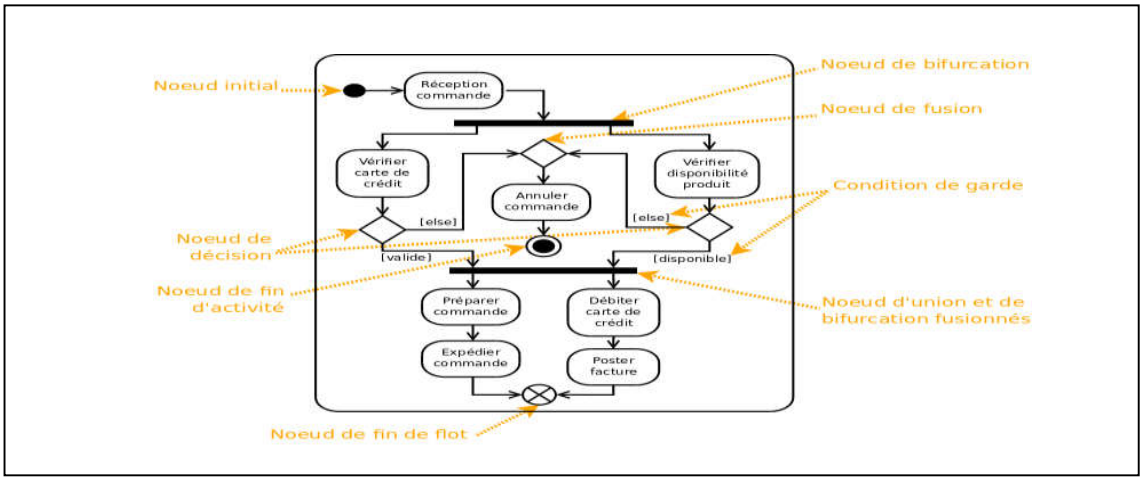


Figure 1.9 : exemple de diagramme d'activité

4.8 Diagramme de classes

4.8.1 Définition

Les diagrammes de classe sont les diagrammes UML les plus utilisés. Ils ont pour but de mettre en évidence les classes d'un système et leurs relations **[5].**

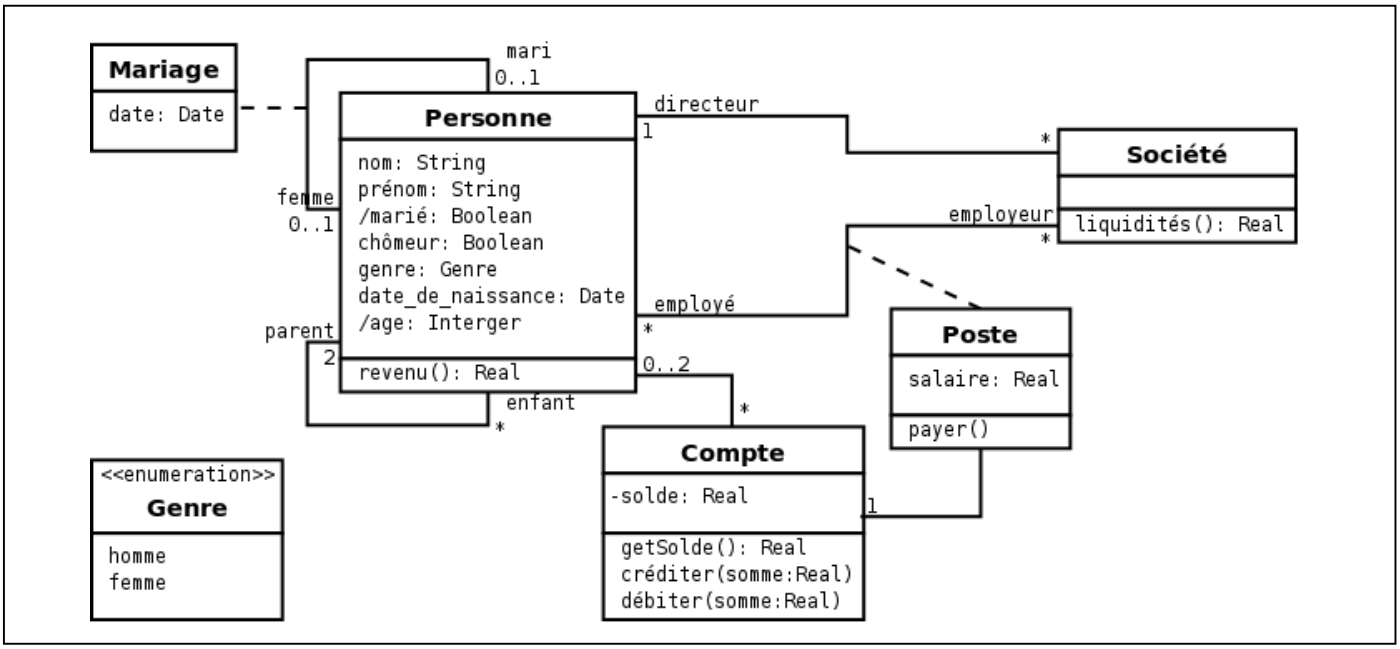


Figure 1.10 : exemple d'un diagramme de classes

4.8.2 Schéma des classes

De manière générale, les classes peuvent être représentées par un rectangle avec trois compartiments, comme indiqué ci-dessous :

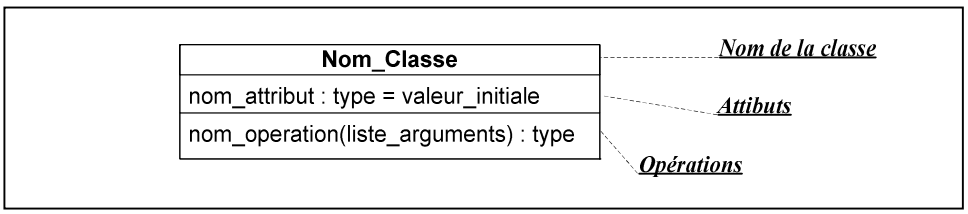


Figure 1.11 : exemple d'une classe

4.8.3 **Les attributs**

La déclaration d'un attribut a la syntaxe suivante :

Nom_Attribut : Type_attribut = Valeur_Initiale

UML permet de définir les attributs dérivés (attributs qui peuvent être calculés par d'autres éléments déjà existants). Ces attributs sont marqués par le symbole " / ". Par exemple, pour une classe rectangle, on connaît les attributs longueur et largeur. L'attribut dérivé serait la surface. **[10]**

4.8.4 **Les opérations**

La déclaration d'une opération a la syntaxe suivante :

Nom_Opération (Nom_Argument : Type_Argument = Valeur_par_defaut...) : type_Retourné

UML permet de définir des opérations qui retourne une valeurs ou ne retournant rien (void) .

4.8.5 **La visibilité**

UML définit trois niveaux de visibilité des attributs et des opérations **: [10]**

	Niveaux de visibilité
-	**Privé** : l'élément n'est visible que dans la classe.
+	**Public** : l'élément est visible par toutes les autres classes.
#	**Protégé** : visible par la classe et ses sous classes.

Tableau 1.1 : niveau de visibilité

4.8.6 **Les identifiants**

La notation UML prévoit aussi les variables et les opérations de classes. Ces éléments sont identifiés par leur nom souligné dans la représentation de la classe. **[10]**

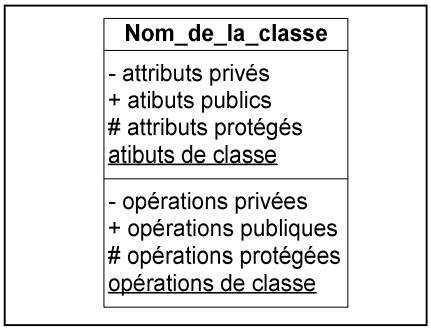


Figure 1.12 : exemple d'une classe avec identifiant

4.8.7 **Les associations**

- Les associations en UML sont représentées par des traits continus entre les

classes . Pour plus de renseignements, il est utile de nommer les associations. Pour cela, le nom de ces dernières se met au milieu de la ligne qui symbolise l'association, en italique sous forme verbale en général.

- Le sens de lecture de l'association peut être précisé au moyen des symboles " < " et " > ". De même, pour plus de compréhension de l'association, on peut indiquer à chaque extrémité le rôle qui décrit comment une classe voit une autre classe au travers d'une association. Le nommage des rôles est sous forme nominale. **[11]**

4.8.7.1 Les cardinalités

On peut préciser la multiplicité des associations. Ceci indique combien d'objets de la classe considérée participent à la relation et peuvent être liés avec un objet de l'autre classe. UML définit la syntaxe suivante : **[11]**

	Cardinalité
1	**Un et un seul**
0..1	**Zéro ou un**
x..y	**De x à y (x et y remplaçant ici un nombre quelconque)**
*****	**Plusieurs, en quantité indéfinie**
0..*	**De zéro à plusieurs**
1..*	**De un à plusieurs**
N	**Plusieurs, en quantité définie**

Tableau 1.2 : Les différentes cardinalités

4.8.7.2 Classes associé

UML définit des classes associées qui ont pour but d'ajouter des attributs et des opérations à une association de deux (ou plusieurs) classes. UML représente la classe associée par une classe reliée par un trait en pointillé à l'association concernée. **[12]**

Ci-dessous un exemple (extrait du football) de classe associée :

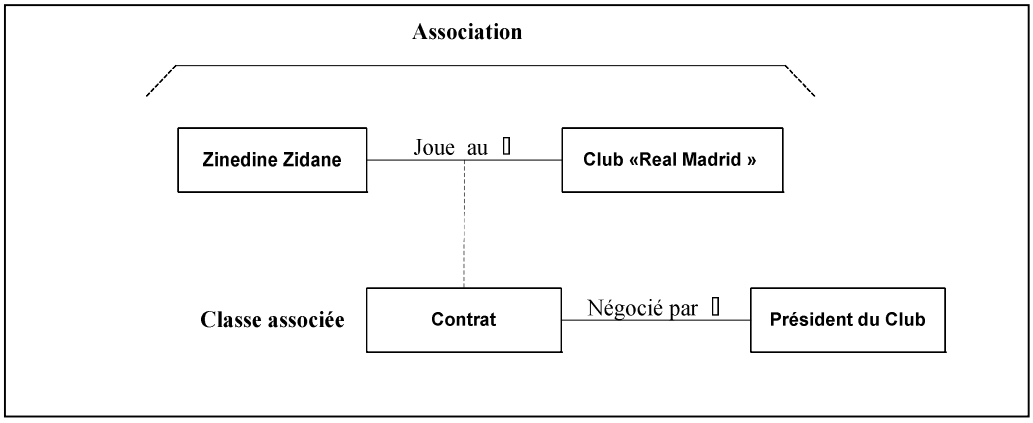
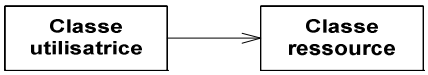


Figure 1.13 : exemple d'une association

4.8.8 Les relations

4.8.8.1 La dépendance

Une relation de dépendance indique un "lien sémantique" entre deux (ou plusieurs) classes. Elle se note par une flèche partant des classes dite *classes utilisatrices*, dirigée vers les classes dites *classes ressources*, comme suit :



4.8.8.2 Agrégation et composition

- Une ***agrégation*** représente une association non symétrique dans laquelle une extrémité de l'association joue un rôle prédominant par rapport à l'autre extrémité (par exemple, une classe qui fait partie d'une autre, une action d'une classe impliquant une action dans une autre classe...) L'agrégation en UML se représente par un petit losange blanc du côté de l'agrégat. **[11]**

- Il existe un cas particulier de l'agrégation qui s'appelle la ***composition*** ; il s'agit de l'association correspondant à la contenance. UML représente la composition par un losange noir. **[11]**

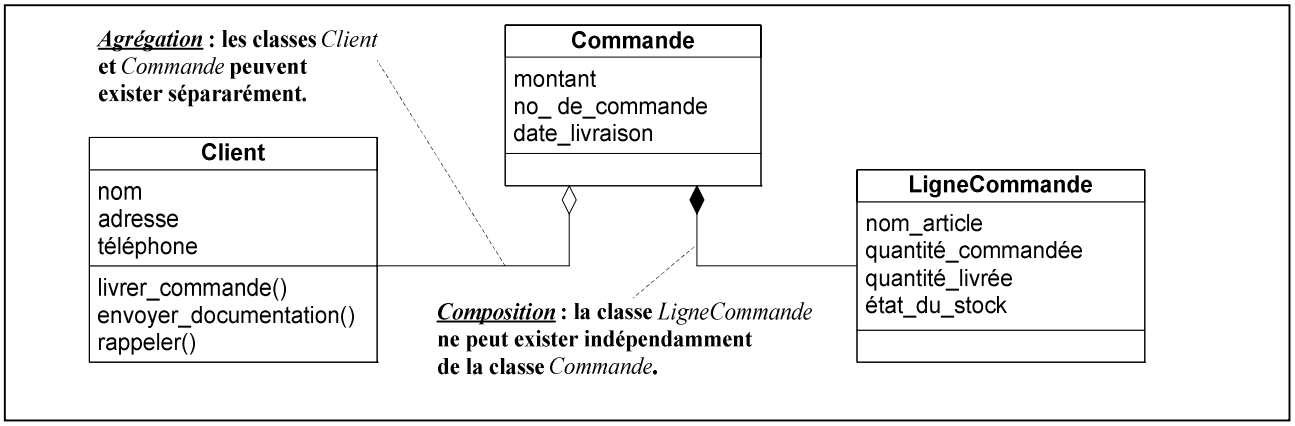


Figure 1.14 : exemple d'utilisation des agrégats et composition

4.8.8.3 La généralisation (héritage) et la spécialisation

- La ***généralisation*** est une technique permettant de mettre en commun, à l'intérieur d'une superclasse, certaines caractéristiques communes à plusieurs classes (dites ***sous-classes*** ou ***classe de base***). Elle exprime ainsi le fait que les éléments d'une classe puissent être décrits dans une autre classe, les attributs, opérations et associations de la superclasse étant hérités par les sous-classes.
- On représente la généralisation par une flèche creuse, pointant de la classe la plus spécialisée vers la classe plus générale. Le contraire de la généralisation se nomme la ***spécialisation***. La spécialisation sert à préciser de nouvelles contraintes à partir d'un modèle de base. **[12]**

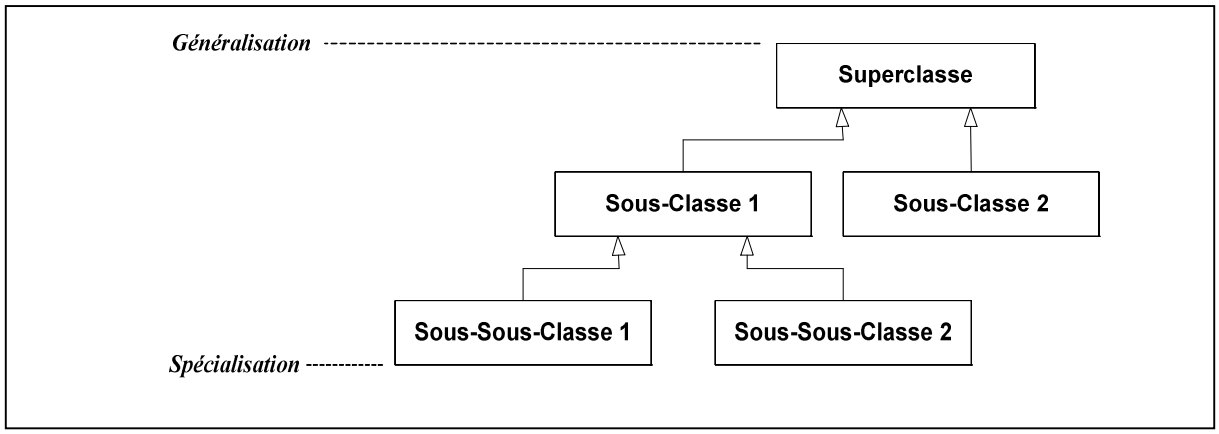


Figure1.15 : exemple d'utilisation des générations

4.9 Diagramme de séquence

Il s'agit de la représentation des interactions entre les objets selon un point de vue temporel. Le diagramme est structuré de la manière suivante :

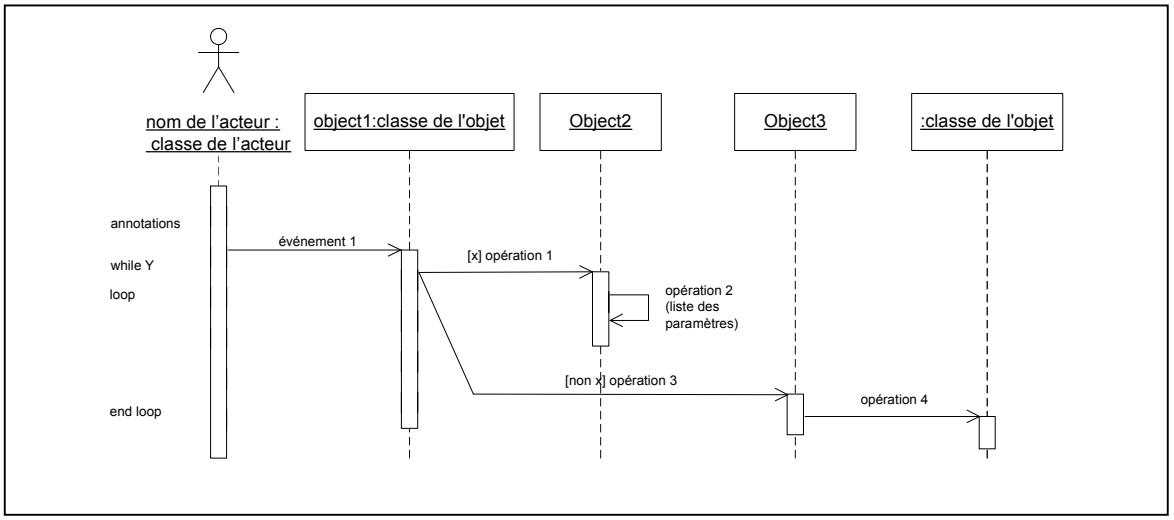


Figure 1.16 : exemple d'un diagramme de séquence

Les objets étudiés sont placés sur la première ligne et pour chaque objet, on lui associe une barre verticale en pointillée appelée " ligne de vie " de l'objet. Le diagramme possède un axe des temps dirigé du haut vers le bas. Les messages sont représentés par des flèches horizontales orientées de l'émetteur vers le destinataire. Lorsque le message possède un temps de propagation non négligeable, les flèches sont alors obliques. **[13]**

La représentation des périodes d'activité des objets est possible à l'aide de bande rectangulaire le long des lignes de vie des objets et dont les extrémités représentent respectivement le début et la fin de l'activité. **[13]**

Dans le diagramme, on peut indiquer les branchements conditionnels par du pseudo code placé le long de la ligne de vie ou alors entre crochets sur le message à conditionner. De même, on peut placer en pseudo code les boucles d'itération (while, for).

5 Conclusion

Dans cette partie nous avons présenté le langage de modélisations UML et ces différents diagrammes, en discutant les points forts et les points faibles de l'UML.

Chapitre 2

L'Ontologie

1 Introduction

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. 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. Plusieurs définitions du terme ontologie ont été proposées selon les courants et les communautés de pensée. Nous en reprenons les principales dans cette section.

2 Définitions

2.1 Du point de vue de la Métaphysique

Historiquement, l'Ontologie (avec "O") est d'abord une notion philosophique. Nous pouvons en trouver une définition succincte dans les dictionnaires :

- **Le Petit Robert (Ed. 1983) :** n.f., (1692 ; lat. philo. Ontologia, 1646).

Philo. Partie de la métaphysique qui s'applique à l'être en tant qu'être, indépendamment de ses déterminations particulières.

- **Le Petit Larousse (Ed. 1998) :** n.f. (gr. ontos, être, et logos, science).

PHILOS. 1. Etude de l'être en tant qu'être, de l'être en soi. 2. Etude de l'existence en général, dans l'existentialisme.

- **Encyclopaedia Universalis (Ed. 2006) :** "Ontologie" veut dire : doctrine ou théorie de l'être. Cette simple définition, toute nominale d'ailleurs, propose une petite énigme de lexique : le mot "ontologie" est considérablement plus récent que la discipline qu'il désigne ; ce sont les Grecs qui ont inventé la question de l'être, mais ils n'ont pas appelé ontologie la discipline qu'ils instituaient.

- **Aristote :** désigne de façon indirecte comme "la science que nous cherchons" la théorie de l'être en tant qu'être donc l'origine de la notion d'ontologie remonte à Aristote. **[14]**

2.2 Du point de vue de l'ingénierie des connaissances

Dans l'univers de l'informatique le terme "ontologie" a été introduit dans les années 1990 avec les travaux de **Grüber** et son équipe à Stanford **[15]**. Où il écrit : "**une ontologie**

est une spécification formelle, explicite d'une conceptualisation partagée". Il précise que cette définition est faite dans le contexte du partage et de la réutilisation des connaissances, et que ce qui importe est ce pourquoi une ontologie est faite**.**

- **Explicite** signifie que le type des concepts utilisés et les contraintes sur leur utilisation sont explicitement définis.
- **Formelle** se réfère au fait que l'ontologie doit être compréhensible par les machines.
- **Partagée** reflète la notion de connaissance consensuelle décrite par l'ontologie, c'est-à-dire qu'elle n'est pas restreinte au point de vue de certains individus seulement, mais reflète un point de vue plus général, partagé et accepté par un groupe.

Parmi les nombreuses autres définitions, citons :

- **Welty :** une ontologie est la définition des classes, relations, contraintes et règles d'inférence qu'une base de connaissances peut utiliser **[16];**
- **Guarino** : une ontologie est un vocabulaire partagé, plus une spécification (en fait une caractérisation) du sens "convenu" de ce vocabulaire **[17];**
- **Le W3C4 :** une ontologie définit les termes utilisés pour décrire et représenter un domaine de la connaissance.

3 Les composants de l'ontologie

Les ontologies rassemblent les connaissances propres à un domaine donné. En représentation des connaissances, ces connaissances sont véhiculées à l'aide d'un certain nombre de constituants ou briques de base et qui sont principalement des: 1) Concepts, 2) Relations, 3) Fonctions, 4) Axiomes, 5) Instances **[15], [18]**

Le détail de ces constituants est comme suit :

- **Les concepts**, aussi appelés termes ou classes de l'ontologie, constituent les objets de base manipulés par les ontologies. Ils correspondent aux abstractions pertinentes du domaine du problème.
- **Les relations** traduisent les interactions existant entre les concepts présents dans le domaine analysé. Ces relations sont formellement définies comme tout sous- ensemble d'un

produit cartésien de n ensembles,

Il existe différents types de relations :

- la relation de spécialisation (subsomption),
- la relation de composition (méronymie),
- la relation d'instanciation, etc.

Ces relations nous permettent de capturer, la structuration ainsi que l'interaction entre les concepts, ce qui permet de représenter une grande partie de la sémantique de l'ontologie.

- **Les fonctions :**constituent des cas particuliers de relations, dans laquelle un élément de la relation, (le nième) est défini en fonction des N-1 éléments précédents. **[18]**
- **Les axiomes :** constituent des assertions, acceptées comme vraies, à propos des abstractions du domaine traduites par l'ontologie.
- **Les instances :**constituant la définition extensionnelle de l'ontologie ; ces objets véhiculent les connaissances (statiques, factuelles) à propos du domaine du problème. **[19]**

3.1 Propriétés des concepts :

Il est possible d'associer aux concepts un certain nombre de propriétés qui peuvent porter aussi bien sur l'extension que sur l'intention et dont on peut citer principalement et sans prétendre à aucune exhaustivité celles présentées dans **[19]**:

- **la généricité :** un concept est générique s'il n'admet pas d'extension. Par exemple, la vérité est concept générique;
- **l'identité :** un concept porte une propriété d'identité si cette propriété permet d'identifier les instances de ce concept. Par exemple, le numéro de l'étudiant constitue une propriété d'identité pour le concept d'étudiant;
- **la rigidité :** un concept est rigide si toute instance de ce concept en reste instance dans tous les mondes possibles. Par exemple, humain est un concept rigide, étudiant est un concept non rigide;
- **l'anti-rigidité:** un concept est anti-rigide si toute instance de ce concept est essentiellement définie par son appartenance à l'extension d'un autre concept. Par exemple, étudiant est un concept anti-rigide car l'étudiant est avant tout un humain.
- **l'unité :** un concept est un concept unité si, pour chacune de ses instances, les différentes parties de l'instance sont liées par une relation qui ne lie pas d'autres

instances de concepts.

En plus de ces propriétés intrinsèques, nous pouvons aussi citer les propriétés inter-concepts qui portent sur des propriétés extrinsèques aux concepts. Il s'agit principalement de **[20] [21]** :

- **l'équivalence :** deux concepts sont équivalents s'ils ont la même extension. Par exemple, le concept salarié est équivalent au concept élève dans le cadre d'une entreprise qui dispense des formations à tous ses salariés;
- **la disjonction :** deux concepts sont disjoints si leurs extensions sont disjointes. Par exemple, homme et femme sont disjoints dans la mesure où aucune instance de l'un ne peut être à la fois un homme et une femme;

3.2 Propriétés des relations :

Tout comme pour les concepts, il existe aussi pour les relations un certain nombre de propriétés, dont principalement on peut distinguer des propriétés ***inter-relations***, et des propriétés liant une relation et des concepts:

- Les propriétés nécessaire à une relation permettent de définir la relation, et on distingue principalement:

- **les propriétés algébriques :** symétrie, réflexivité, transitivité;
- **la cardinalité :** nombre de participations possibles d'une instance de concept dans une relation.

- Les propriétés ***inter-relations*** portant sur plusieurs relations et qui peuvent être:

- **l'incompatibilité :** deux relations sont incompatibles si elles ne peuvent lier les mêmes instances de concepts. Par exemple, les relations "être rouge" et "être vert" sont incompatibles;
- **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. Par exemple, les relations "a pour père" et "a pour enfant" sont inverses l'une de l'autre
- **l'exclusivité :** deux relations sont exclusives si, quand l'une lie des instances de concepts, l'autre ne lient pas ces instances, et vice-versa. L'exclusivité entraîne l'incompatibilité. Par exemple, l'appartenance et le non appartenance sont exclusifs.

Les propriétés liant une relation et des concepts et qui sont **[22] :**

- **Le lien relationnel :** il existe un lien relationnel entre une relation R et deux concepts C1 et C2 si, pour tout couple d'instances des concepts C1 et C2, il existe une relation de type R qui lie les deux instances de C1 et C2.
- **La restriction de relation :** pour tout concept de type C1, et toute relation de type R liant C1, les autres concepts liés par la relation sont d'un type imposé.

4 Classification des ontologies

En fonction de leur usage, on distingue classiquement cinq catégories d'ontologies **[23]** : génériques, de domaine, d'application, de représentation et de méthodes.

4.1 L'ontologie du domaine :

Spécifient un point de vue sur un domaine particulier. Le vocabulaire est généralement lié à un domaine de connaissances générique comme la médecine ou la loi. Les concepts d'une ontologie de domaine sont souvent définis comme une spécialisation des concepts des ontologies génériques. Enterprise est un exemple d'ontologie décrivant le domaine de l'entreprise **[24]**.L'ontologie de domaine est constituée de la terminologie du domaine d'application et d'un ensemble de relations et de types de base comme la relation classe/super-classe. Une ontologie de domaine et alors constituée de **[25]**:

1. une description en extension du vocabulaire du domaine,
2. une typologie,
3. une hiérarchie ou un treillis de classes.

4.2 L'ontologie générique :

Décrivent des concepts généraux, indépendants d'un domaine ou d'un problème particulier. Il s'agit par exemple des concepts de temps, d'espace, d'événement. Elles sont prévues pour être utilisées dans des situations diverses, et pour servir une large communauté d'utilisateurs. Les ontologies génériques sont aussi appelées Modèles de haut niveau ou ontologies TOP. L'ontologie de Sowa **[26]** et CYC **[27]** sont des ontologies génériques.

4.3 L'ontologie d'application :`

Une ontologie d'application décrit la structure des connaissances nécessaires à la réalisation d'une tâche particulière **[27]**. Elle permet aux experts du domaine d'utiliser le même langage que celui de l'application. Elle réalise trois objectifs :

1. faciliter le processus d'acquisition des connaissances de l'application ;
2. permettre l'intégration avec les serveurs existants dans l'environnement de l'application
3. sélectionner les structures de données appropriées au modèle computationnel .

4.4 L'ontologie de représentation:

spécifient un formalisme de description qui fournit une structure de représentation et des primitives pour décrire les concepts des ontologies de domaine et des ontologies génériques. La Frame ***Ontology*** en est un exemple type **[28]**. La Frame ***Ontology*** est associée à ***Ontolingua,*** un système de description et de traduction d'ontologies dont la syntaxe et la sémantique s'appuient sur le langage **KIF**[(5)] **[28]**.

4.5 L'ontologie de méthode:

Décrivent le processus de raisonnement d'une façon indépendante d'un domaine et d'une implémentation donnée. Elles spécifient des entités qui relèvent de la résolution d'un problème telles que l'abduction, la déduction ou l'observation **[29]**. Elles fournissent les définitions des concepts et relations utilisés pour spécifier un processus de raisonnement lors de la réalisation d'une tâche particulière.

En plus de cette classification, l'institut AIFB de l'université de Karlsruhe distingue les ontologies légères Light-weight ontology et les ontologies riches Heavy-weight ontology **[30].**

- **Une ontologie légère** comprend des concepts, des types atomiques, une hiérarchie IS-A entre les concepts, et des relations entre les concepts. C'est le type d'ontologie le plus courant.
- **Une ontologie riche** comprend des contraintes de cardinalité, une taxonomie de relations, des axiomes/héritages sémantiques (logique de description, logique de propositions, clauses de Horn, logiques d'ordre plus élevé) et suppose l'existence d'un système d'inférence.

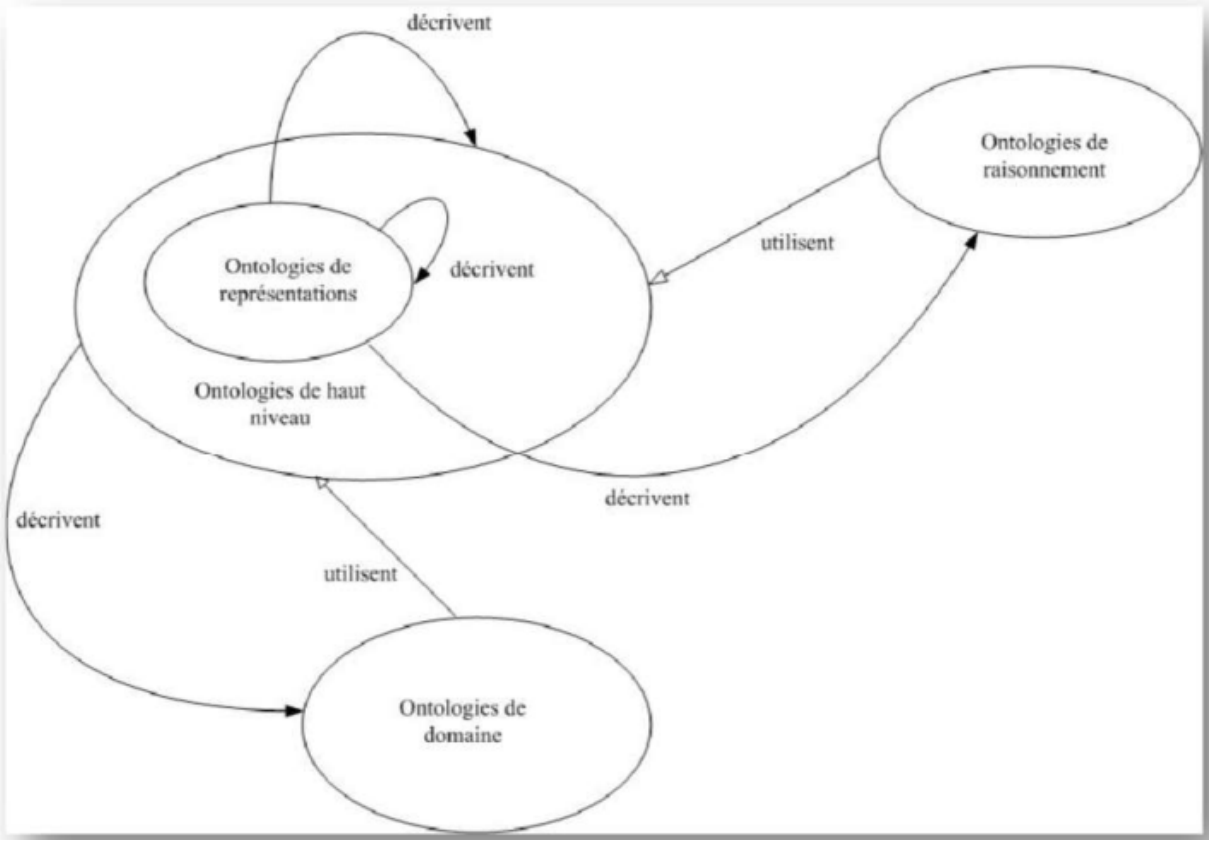


Figure 2.1 : Différents types d'ontologies

5 Le rôle des ontologies

Fondamentalement, le rôle des ontologies est d'améliorer la communication entre humains, mais aussi entre humains et ordinateurs et finalement entre ordinateurs.

5.1 Communication :

Les ontologies peuvent intervenir dans la communication entre humains. Elles servent par exemple, à créer au sein d'un groupe ou d'une entreprise un «vocabulaire» standardisé. Pour cette raison, on utilise les ontologies informelles. L'existence de vocabulaires différents au sein d'une entreprise (ex : bureau d'études, bureau des méthodes) ou d'une industrie (ex : constructeur automobile, équipementier) constitue un frein à la collaboration et aux partenariats. Les enjeux touchent donc directement la compétitivité de l'entreprise. Pour l'entreprise, l'ontologie sert à :

- améliorer la compréhension entre les employés,
- favoriser la diffusion des informations et leur exploitation,
- Promouvoir une nouvelle approche de conception des systèmes d'information (réutilisation et interopérabilité de logiciels).

Pour ces besoins de standardisation du vocabulaire, une terminologie ou une ontologie informelle peuvent suffire.

5.2 L'interopérabilité :

L'interopérabilité est une spécialisation de la communication, dans ce cas vue entre deux ordinateurs. L'ontologie répertorie alors les concepts que des applications peuvent s'échanger même si elles sont distantes et développées sur des bases différentes. Cette interopérabilité est l'interopérabilité sémantique.

5.3 Interface Homme-Machine :

La visualisation de l'ontologie permet à l'utilisateur de comprendre le vocabulaire utilisé par le SI et de mieux formuler ses requêtes.

6 Processus de construction d'une ontologie

Le processus de construction d'ontologies repose sur un enchaînement de trois étapes (conceptualisation, ontologisation, opérationnalisation) permettant de passer des données brutes à l'ontologie opérationnelle.

6.1 Conceptualisation

Cette étape consiste à identifier des connaissances contenues dans un corpus représentatif du domaine. Ce travail doit être mené par un expert du domaine, assisté par un ingénieur de la connaissance. Cette étape permet d'aboutir à un modèle informel, sémantiquement ambiguë et généralement exprimé en langage naturel.

6.2 Ontologisation:

C'est une formalisation, autant que possible, sans perte d'information, du modèle conceptuel obtenu à l'étape précédente. Ce travail doit être mené par l'ingénieur de la connaissance, assisté de l'expert du domaine. Cette étape Facilite sa représentation ultérieure dans un langage complètement formel et opérationnel **[20].**

L'Ontologisation peut être complétée par une étape d'intégration au cours de laquelle une ou plusieurs ontologies vont être importées dans l'ontologie à construire **[21].**

6.3 Opérationnalisation:

C'est une transcription de l'ontologie dans un langage formel (i.e.possédant une syntaxe et une sémantique) et opérationnel (i.e. doté de services inférentiels permettant de mettre en œuvre des raisonnements) de représentation de connaissances. Ce travail doit être mené par

l'ingénieur de la connaissance.

6.4 Les ontologies et le raisonnement

La représentation des connaissances par les ontologies peut s'accompagner par des mécanismes de raisonnement. Le raisonnement concerne la manipulation des connaissances déjà acquises pour produire de nouvelles connaissances. Il utilise des mécanismes d'inférence qui permettent la résolution des problèmes pour lesquels il n'existe pas de procédures explicites dans le programme. L'opération d'inférence consiste à admettre une proposition préalable qui a la valeur Vrai. Le raisonnement, déduction, induction, etc. Sont des cas spéciaux de l'inférence. Différents mécanismes de raisonnement sont utilisés selon les objectifs du système à mettre en place: raisonnement logique, raisonnement par classification, le filtrage, l'héritage et le raisonnement à base de règles **[32].**

6.4.1 Le filtrage

Utilisé par la plupart des réseaux sémantiques, il consiste à parcourir le graphe et à chercher tous les sous-graphes ayant des propriétés ou une structure commune avec un graphe cible.

6.4.2 L'héritage

L'héritage est un mécanisme de raisonnement qui consiste à récupérer des informations des classes représentant des concepts plus généraux, pour les utiliser dans des classes plus spécialisées, cette récupération se fait en suivant les liens de spécialisation « ***est-un*** ». La relation de spécialisation représente l'inclusion ensembliste, toutes les instances d'une classe le sont aussi pour ses superclasses. Une classe doit donc pouvoir récupérer l'information de ses superclasses. Le mécanisme d'héritage de propriété permet la récupération de cette information à travers les liens de spécialisation et évite ainsi d'avoir à recopier les attributs des superclasses dans la sous-classe. L'héritage est dynamique, et il veut dire que l'information héritée d'une classe n'est pas stockée dans la sous classe mais récupérée chaque fois que le système accède à la sous-classe. Cela garantit que toute modification faite à une classe est prise en compte par ses classes. Le mécanisme d'héritage est plus un raccourci d'écriture (car il évite à recopier l'information) qu'un réel mécanisme d'inférence de nouvelles connaissances **[33].**

6.4.3 Le raisonnement à base de règles

Le raisonnement à base de règles est également un mécanisme de raisonnement sur les

connaissances. L'élément de base des systèmes à base de règles est la règle d'inférence, la règle a la forme suivante :

Si < ***condition***> *Alors* < ***Action*** >

La partie condition est exprimée par un prédicat logique correspondant à une affirmation sur la base de connaissance qui doit être vraie au moment de valider la règle pour que l'action soit déclenchée, la partie action, qui est la partie exécutable de la règle indique des ajouts ou modification à faire à la base. Un système à base de règles comporte trois parties :

Une base de règle, une base de fait et un moteur d'inférence. Les systèmes à base de règles permettent en général de bien résoudre les problèmes de diagnostic ; ils offrent un cadre déclaratif pour exprimer des connaissances procédurales de « ***savoir-faire*** », ce qui permet de voir clairement les conditions dans lesquelles une règle est applicable**[33].**

7 Langages de représentation d'ontologies

Penser une ontologie ne peut se faire sans un formalisme pour la représenter. Le langage utilisé pour décrire les termes d'une ontologie a un impact direct sur le niveau de formalisme de l'ontologie. Ainsi, on distingue les ontologies informelles, qui utilisent le langage naturel et qui peuvent coïncider avec les terminologies, les ontologies semi-formelles, qui fournissent une faible axiomatisation, comme les taxonomies, et enfin les ontologies formelles, qui définissent la sémantique des termes par une axiomatisation complète et rigoureuse.

Un langage d'ontologie est un langage formel permettant de représenter une ontologie, décrire les annotations, exploiter et raisonner sur les contenus des ressources

Plusieurs langages, dont la syntaxe est basée sur le langage **XML**[(6)], ont été conçus pour une utilisation des ontologies dans le cadre du WS, notre centre d'intérêt. Parmi eux, les plus importants sont **RDF**[(7)]**/RDF(S)**[(8)] **[34][35],** qui est la recommandation du **W3C**[(9)] pour représenter les métadonnées, **OIL**[(10)] **[36],** DAML+OIL, **OWL**[(11)] qui est la recommandation du W3C pour représenter des ontologies. Ajoutons à cette liste **SKOS**[(12)]**,** un langage plus récent, qui est appelé à jouer un rôle important pour la représentation en particulier des terminologies. Nous allons rapidement dans la section suivante faire une revue des langages RDF/RDF(S) **[34],** OWL et SKOS **[37],** utilisés dans la cadre du Web Sémantique.

.

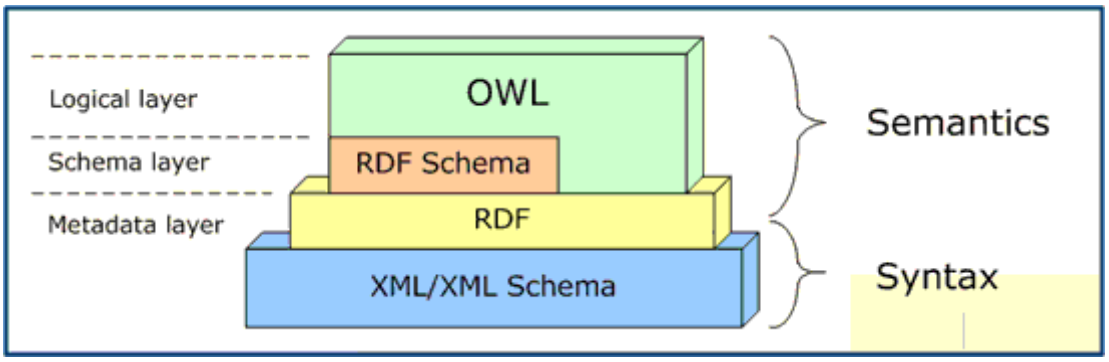


Figure 2.2 : les différents couches d'ontologie

7.1 RDF (Resource Description Framework)

RDF (Resource Description Framework) **[34]** est une recommandation du W3C pour décrire des ressources. C'est un modèle de graphe pour décrire les (méta-)données en permettant leur traitement automatisé. A l'origine, il a été déni pour décrire des ressources du Web telles que les pages Web ; cependant une ressource peut être toute chose ayant une identité (objet physique, concept abstrait, etc.). RDF décrit les ressources en exprimant des propriétés et en leur attribuant des valeurs. Il utilise pour cela le vocabulaire déni par RDF Schéma noté RDF(S)**[34]**

RDF(S) fournit un ensemble de primitives simples, mais puissantes, pour la structuration de la connaissance d'un domaine en classes et sous-classes, propriétés et sous-propriétés avec la possibilité de restreindre leur domaine d'origine (***rdf :domain***) et leur domaine d'arrivée (***rdf :range***).

L'élément de base d'un document RDF est la ressource, correspondant à la représentation conceptuelle d'une entité. Une ressource est identifiée par son URI (Uniform Resource Identifier), de la forme ***http ://uri/du/document#courant***. Un document structuré en RDF est un ensemble de triplets. Un triplet RDF est une association < ***sujet, objet, prédicat*** > appelée assertion. Par exemple, le sujet peut être une page Web donnée, l'objet, une propriété de cette page comme son titre, et le prédicat, la valeur de cette propriété. Une des syntaxes (sérialisation) de ce langage (en plus de la forme graphique) est RDF/XML dont nous fournissons un exemple au Figure 2.3**[38]**

```
<rdf:RDF
  xmlns: rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"/>
  </rdf:Description>

  <rdf:Description about="http://www.w3.org/staffId/85740">
    <rdf:type resource="http://description.org/schema/Person"/>
    <v:Name>Ora Lassila</v:Name>
    <v:Email>lassila@w3.org</v:Email>
  </rdf:Description>
</rdf:RDF>
```

Figure 2.3 : Exemple d'un document RDF

RDF(S) est indiqué pour la description de ressources, cependant il présente assez rapidement des limites lorsqu'il s'agit de l'utilisation comme langage de représentation d'ontologies ayant de fortes contraintes **[38]** :

1. la portée locale des propriétés : rdf :range permet de dénir les domaines de valeurs (range) des propriétés. Cependant, il n'est pas possible avec RDF(S) de limiter leur application à un certain nombre de classes seulement. Par exemple, pour la propriété mange de la classe Etre Vivant, on peut lui assigner comme domaine de valeur nourriture, mais il est par la suite impossible de déclarer que certains êtres vivants mangent des plantes et que d'autres mangent de la viande;
2. l'expression de la disjonction de classes : il arrive souvent que l'on veuille déclarer que deux classes sont disjointes (l'intersection de leurs instances est vide). Par exemple, les classes Homme et Femme sont disjointes. Dans RDF(S), il n'est pas possible de l'exprimer; on ne peut définir que des relations de type "sous-classes" : (e.g., Femme est sous-classe de Humain);
3. l'expression de la combinaison booléenne de classes : on aimerait parfois construire des classes à partir de la combinaison d'autres classes en utilisant les opérateur d'Union, d'Intersection et de Complément. Par exemple on peut vouloir définir la classe Humain comme l'union disjonctive des classes Femme et Homme. RDF(S) ne permet pas de le faire ;
4. l'expression de la restriction de cardinalités : parfois on voudrait définir le nombre exact de valeurs qu'une propriété donnée peut avoir. Par exemple, on peut vouloir dire qu'une personne a exactement deux parents ou qu'un cours est enseigné par au moins

un professeur. Ce genre de restriction est impossible à définir dans RDF(S) ;

5. l'expression de certaines caractéristiques de propriétés : on aimerait exprimer les caractéristiques de certaines propriétés telle que la transitivité (e.g., un grand-parent est le parent d'un parent), l'inverse (Est-Parent est l'inverse de Est-Enfant), etc. De telles caractéristiques sont impossibles à exprimer avec RDF(S).

7.2 Ontology Web Language (OWL)

Le langage OWL fournit des mécanismes pour créer tous les composants d'une ontologie : classes, instances, propriétés et axiomes. OWL repose également sur la syntaxe des triplets RDF et réutilise certaines des constructions RDFS. Comme en RDFS, les classes peuvent avoir des sous-classes, fournissant ainsi un mécanisme pour le raisonnement et l'héritage des propriétés **[39].** Par contre, en OWL, on distingue :

1) les propriétés objet (***object property***) : les relations, qui relient des instances de classes à d'autres instances de classes. C'est l'équivalent des triplets RDF dont l'objet est une ressource.

2) les propriétés type de données (***datatype property***) : les attributs, qui relient des instances de classes à des valeurs de types de données (nombres, chaînes de caractères,…). C'est l'équivalent des triplets RDF dont l'objet est une valeur littérale.

Les axiomes fournissent de l'information au sujet des classes et des propriétés, spécifiant par exemple l'équivalence entre deux classes. Donc OWL permet de définir des ontologies comme un jeu de définition de classes, de propriétés et de contraintes.

Toute classe définie dans une ontologie OWL est une sous-classe de ***owl:Thing***.

OWL a été fractionné en trois langages distincts chacune étant une extension de la précédente :

- **OWL Lite :** Convient aux utilisateurs qui ont principalement besoin d'une hiérarchie de classification et de contraintes simples. Ce sous langage reprend tous les constructeurs de RDF (c'est-à-dire fournit des mécanismes permettant de définir un individu comme instance d'une classe, et de mettre des individus en relation), il utilise les mots-clés de RDFS (***rdfs:subClassOf, rdfs:Property, rdfs:subPropertyOf, rdfs:range, rdfs:domain***), avec la même sémantique, et il supporte les contraintes de cardinalité, mais ne permet d'utiliser que les valeurs 0 ou 1. **[40]**

➢ **OWL LD (OWL Description Logic):** Convient aux utilisateurs qui veulent le maximum d'expressivité, ce sous langage reprend tous les constructeurs d'OWL LITE, il permet tout entier positif dans les contraintes de cardinalité, et le tire son nom de sa correspondance avec les logiques de descriptions.

➢ **OWL FULL :** ce sous langage reprend tous les constructeurs d'OWL DL, et tous les constructeurs de RDF Schéma. Il permet d'utiliser une classe en position d'individu dans les constructeurs.

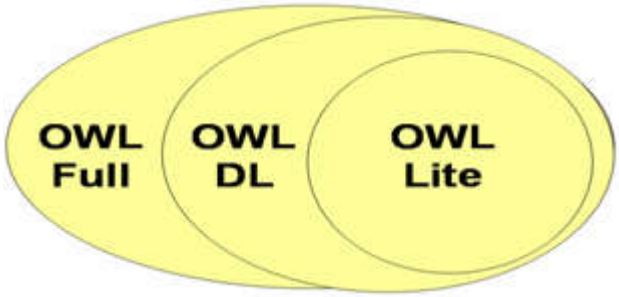


Figure 2.3 Les types de langage OWL

Il y a une stricte compatibilité ascendante de ces trois langages : toute ontologie OWL Lite valide est une ontologie OWL DL valide, et toute ontologie OWL DL valide est une ontologie OWL Full valide. Ainsi, toute conclusion d'une ontologie OWL Lite est une conclusion valide de OWL DL, et toute conclusion OWL DL est une conclusion valide de OWL Full. **[40]**

8 Les domaines d'ontologie

Les ontologies sont utilisées dans plusieurs domaines, le domaine de l'eau, le web sémantique, linguistique et le domaine médicale...etc.

Vue les avantages que présentent les ontologies par rapport à UML en terme d'extensibilité, de raisonnement et de formalisme, les travaux récents montrent qu'il est nécessaire de se diriger vers les ontologies afin d'intégrer une nouvelle vision de représentation formelle et partage de connaissances.

9 Approches de transformation d'UML vers Ontologie

Il existe plusieurs approches dans la littérature sur la façon de transformer les diagrammes de classes UML vers ontologie OWL. **Dans[41] les auteurs** les auteurs proposent une méthode qui convertit les diagrammes de classes UML à une ontologie OWL en utilisant le langage OWL / XML au niveau schéma et en visant à maintenir quelques caractéristiques

sémantiques des diagrammes de classe tel que l'héritage, l'encapsulation, types d'association (composition, agrégation et association simple), les contraintes d'intégrité et l'identifiant d'une classe.

Dans [42] les auteurs proposent une approche basée sur l'utilisation combinée de La méta-modélisation et les grammaires de graphe pour générer automatiquement des ontologies OWL à partir des diagrammes de classes UML. ils ont utilisé L'outil de méta-modélisation **AToM3** pour proposer et mettre en œuvre un ***Méta-modèle*** de diagramme de classe, puis la génération automatique d'un outil de modélisation visuelle pour traiter les diagrammes de classes. A la fin il ont défini une grammaire graphique pour traduire les modèles créés dans l'outil généré aux ontologies OWL au format RDF / XML. **Dans [43]** : les auteurs présentent une Approche qui consiste à préparer le diagramme de classes pour le traitement ultérieur Comme **XMI**(13) document. Puis utiliser des règles de transformation pour les connaissances UML suivantes:(***les classes, attributs, associations et l'héritage entre les classes***) vers une ontologie OWL 2, puis la visualiser avec protégé.

10 conclusion

Dans ce chapitre, nous avons présenté les différentes définitions de la notion d'ontologie. Nous avons aussi souligné les différents éléments dont elle est constituée, ainsi que les différents domaines d'applications en mettant l'accent sur quelques travaux connexes.

Chapitre 3

Conception

1 Introduction

Après avoir présenté dans l'état de l'art le langage de modélisation UML et les ontologies, nous présentons dans cette section une approche qui consiste à générer automatiquement des ontologies a partir des diagrammes UML (Diagramme de classe , séquence).

Notre objectif est de faire un passage d'un diagramme UML, exactement les **Diagrammes de classe et de séquence** vers une ontologie. Cet objectif va être réalisé par la conversion des termes de chaque diagramme vers des concepts d'ontologie OWL. Cette ontologie permet de représenter et intégrer d'une façon cohérente les connaissances des diagrammes UML.

Nous montrons dans ce chapitre les différentes phases de construction d'ontologie à partir de diagramme de classe ou de séquence. Nous commençons par la phase de modélisation des diagrammes en utilisant l'outil suivant : ArgoUML. Ensuite, nous passons à la phase d'exportation des diagrammes dessinés sous forme d'un fichier XMl. Par la suite, nous présentons la construction des concepts et enfin nous terminons par la transformation vers une ontologie OWL et pour tester la consistance sémantique de notre ontologie résultante, nous la chargeons sous Protegé.

2 Approche générale

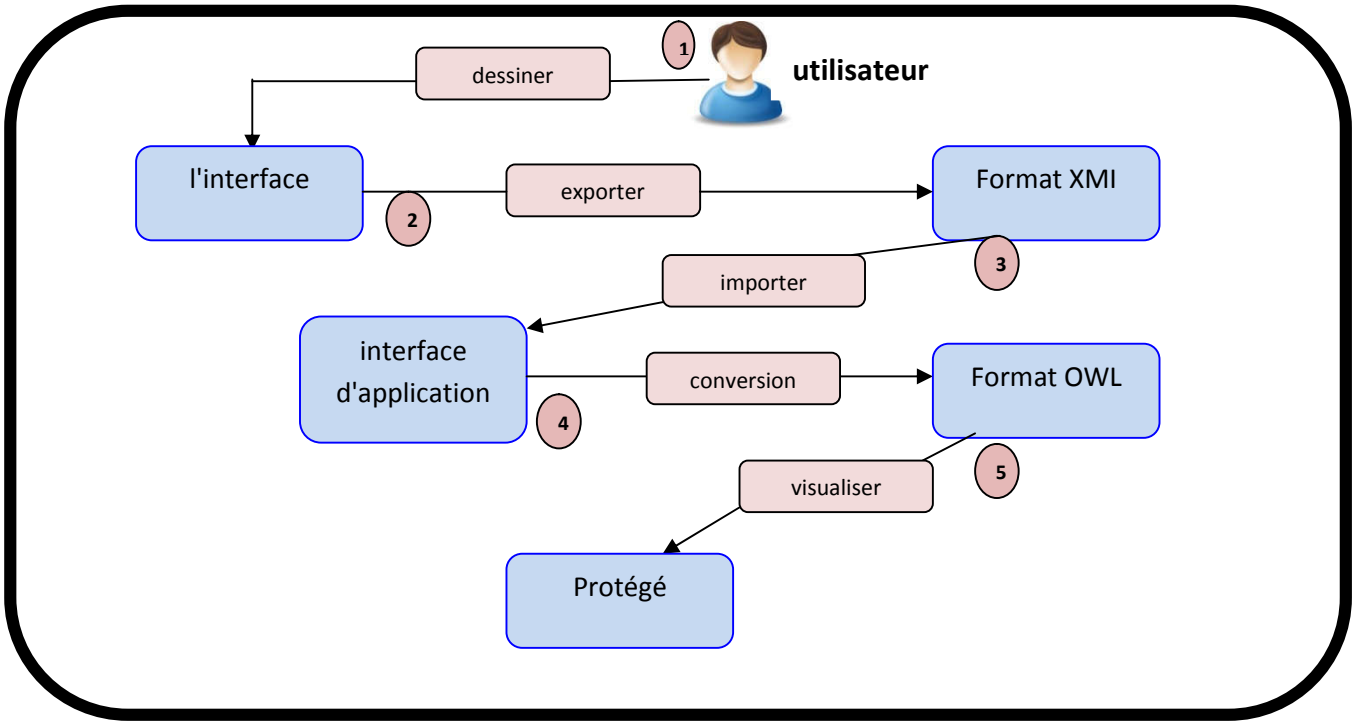


Figure 3.1 : Processus générale de la conversion des diagrammes UML vers Ontologie OWL

2.1 Phase 1 : Dessiner les diagrammes

Pour commencer la construction de notre ontologie, il faut d'abord dessiner le diagramme pour extraire les connaissances importantes à représenter dans l'ontologie. Ces connaissances sont classifiées dans plusieurs types : concept (classe), attribut (propriété), association (relation) ou bien opération pour les diagrammes de classe, et acteur, les objets ou opération (action) pour les diagrammes de séquence.

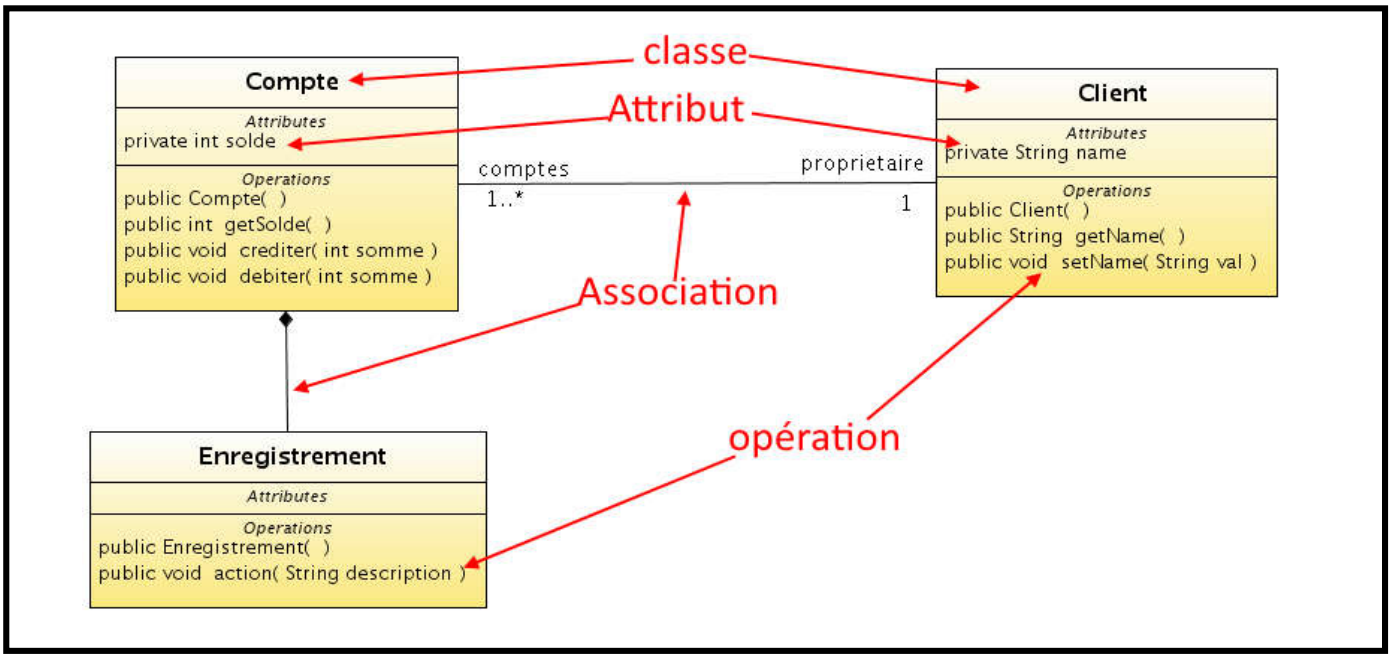


Figure 3.2 : les différentes connaissances de diagramme de classes

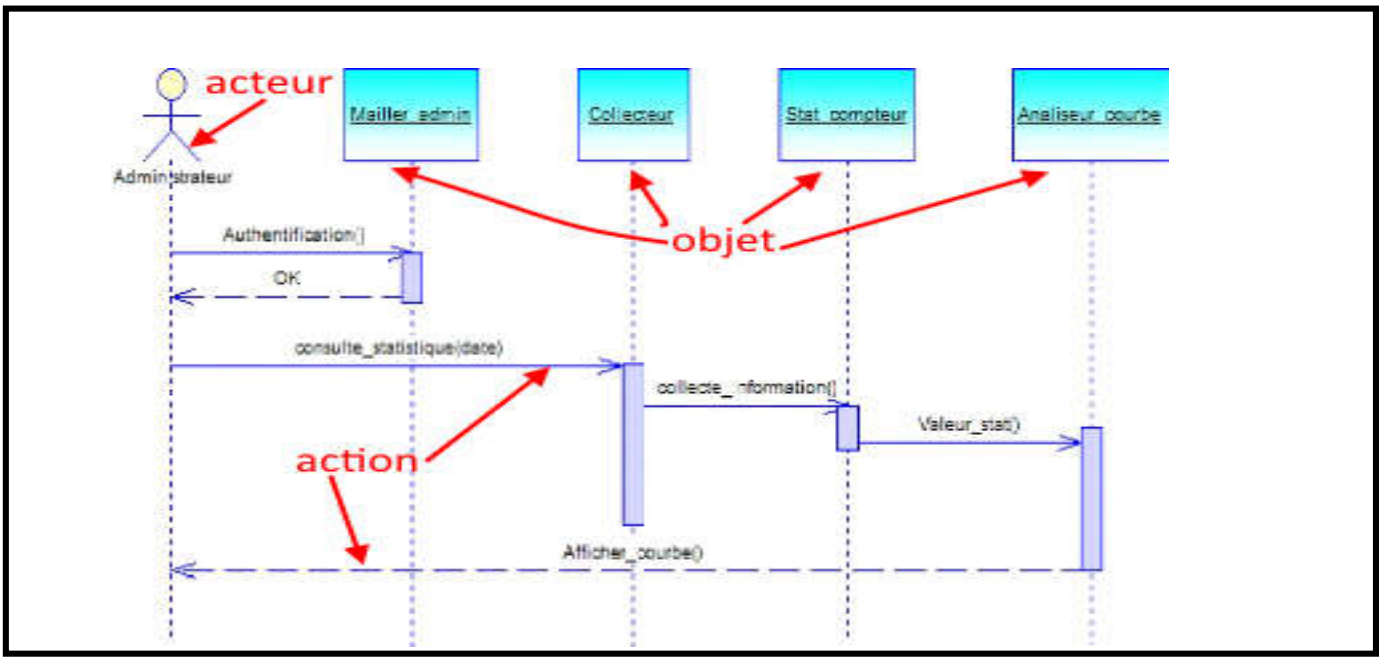


Figure 3.3 : les différentes connaissances de diagramme de séquence

2.2 Phase 2 : La phase de préparation des connaissances

Pour préparer les connaissances de l'étape précédentes pour qu'elles soient utilisables dans le processus de génération (conversion), nous allons exporter nos diagrammes dessinés au format XML.

Cette format facilite le travail et donne une très bonne structure pour décrire les informations sous forme de texte, On utilisant des **balises** pour délimiter les informations.

Figure 3.4 : exemple d'un fichier XMI

2.3 Phase 03 : Génération du fichier XMI

C'est la plus importante phase dans notre travail, elle consiste a stocker les différentes connaissances dans des listes d'objets. Nous avons plusieurs types d'objets créés (classe, attribut , association ...etc.).

Ces types sont créés par rapport a leur forme dans le format XMI, pour contenir les informations nécessaires pour la conversion.

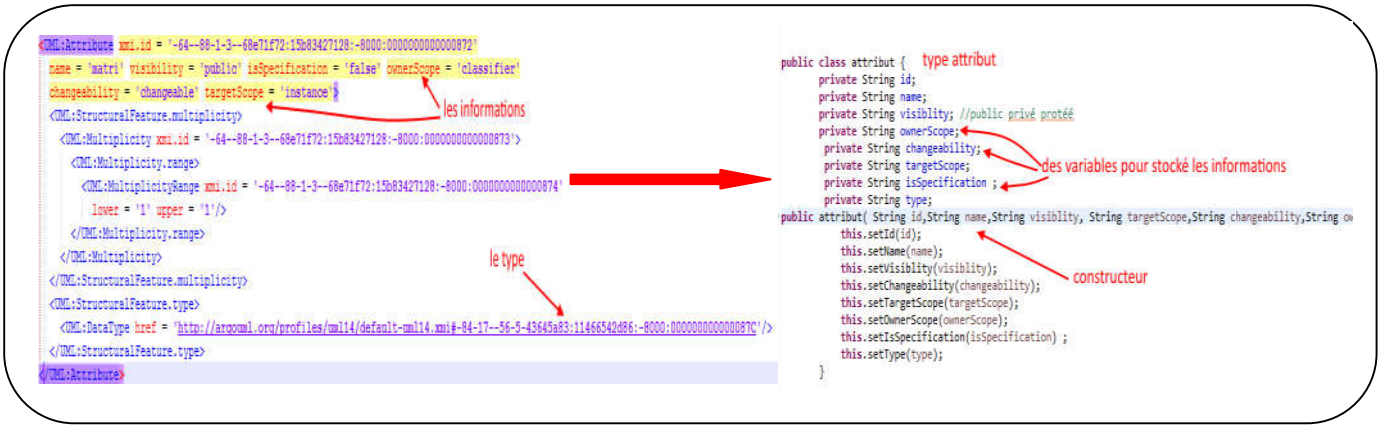


Figure 3.5 : Exemple de créations de type attribut.

De la méme facon on a créé des types pour toutes les différentes connaissences des diagrammes comme le montre les tableaux suivants:

Les types d'objets
Classe
Attribut
Assossiation
Généralisation
Opération
AssossiationEnd

Tableau 3.1 : listes des types dobjets pour un diagramme de classe

Les types d'objets
Acteur/objet
Action

Tableau 3.2 : listes des types dobjets pour un diagramme de séquence

Pour créer un objet de chaque types précedants, il faut déclancher un **constructeur** qui permet de réserver de la mémoire pour notre futur objet et donc, par extension, d'en réserver pour toutes ses variables**[44]**.

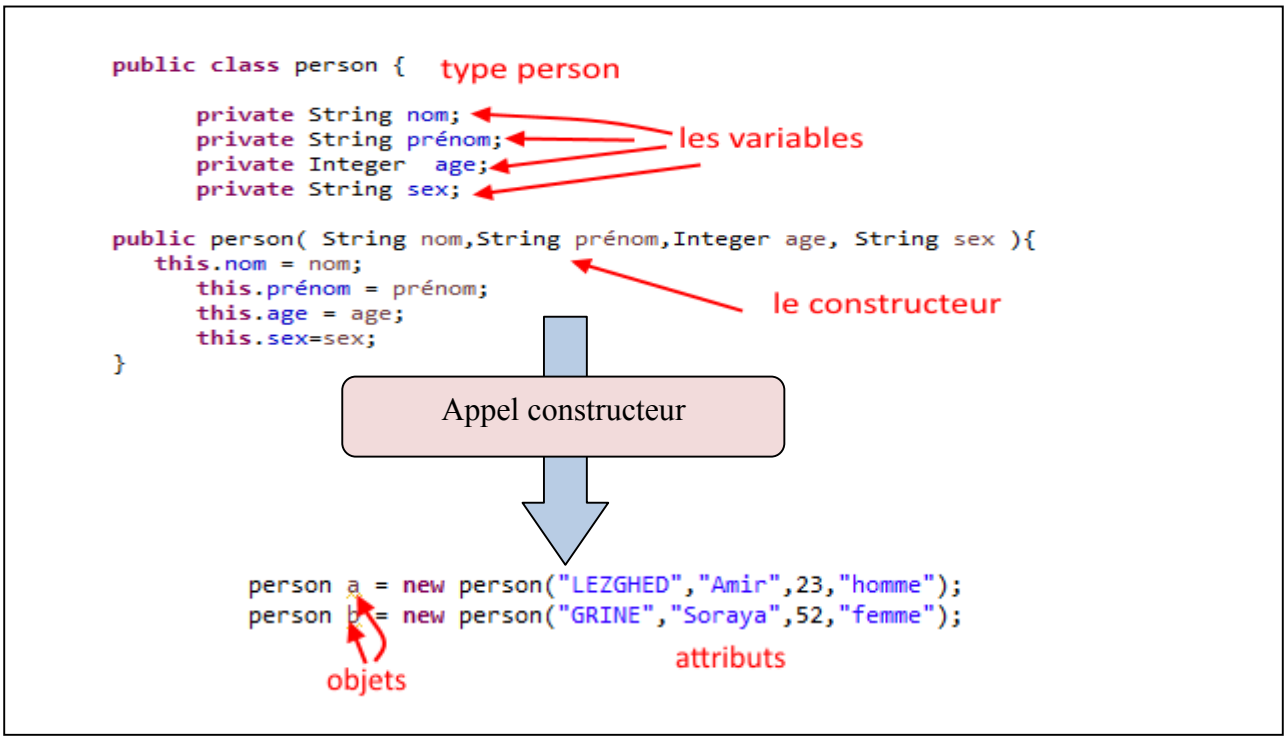


Figure 3.6 : exemple d'utilisation d'un constructeur

2.4 Phase 04 : Conversion et Transformation vers ontologie :

Dans cette phase nous mettons un ensemble des règles de mapping (conversion) pour transformer ***les objets stockés*** des diagrammes vers une ontologie OWL.

Les règles de transformation sont :

2.4.1 Diagramme de classe :

2.4.1.1 Régle1: les classes

Avant de commencer la conversion, nous créons une classe OWL appelé Objet (*OWL: thing*) pour représenter la superclasse Objet de UML.

Rappelons que toutes les classes de diagramme UML héritent par défaut de la classe d'objet. Ensuite, chaque classe de diagramme est converti en un concept OWL avec le même nom **[45],** donc : chaque **classe** sera transformé a un concept owl: Class ;

*<owl:Class rdf:about= "#**nom_de_la_classe**"/>*

2.4.1.2 Règle 2 : L'héritage entre les classes

L'héritage est l'une des notions fondamentales dans UML dont la conservation après la cartographie est une utilité majeure. Ainsi, nous proposons d'utiliser la hiérarchie des concepts fournis par l'ontologie pour représenter cette notion **[46],**.

Notre proposition consiste à représenter chaque sous-classe du diagramme de classe par une *subclasse* qui hérite nécessairement de sa classe mère qui hérite de la classe Objet (*OWL: thing*) .

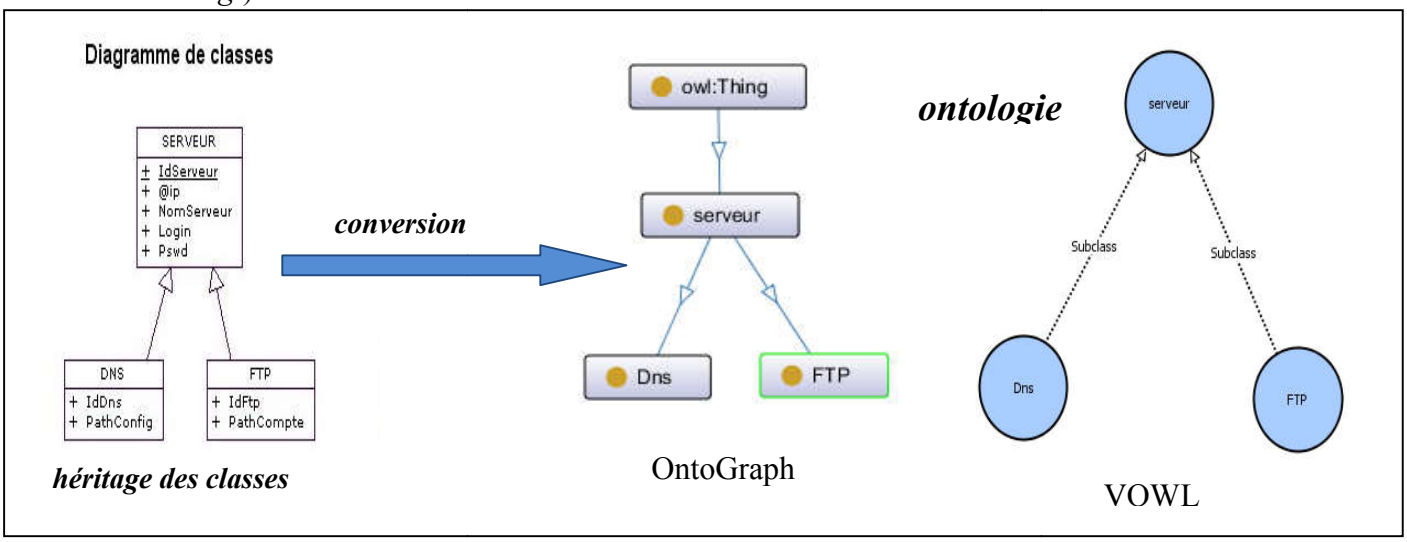


Figure 3.7 : visualisation d'héritage avec l'extension OntoGraph et VOWL

- le code, sera comme suit :

<owl:Class rdf:about="#***nom_de_la classe_mére***">

<rdfs:subClassOf rdf:resource="#***nom_du_sousclasse***"/>

</owl:Class> // balise de fermeture

2.4.1.3 Règle 3 : Les attributs

Un attribut de classe avec un type primitif UML est mappé au type de données de propriété défini à l'aide du type de données Classe de propriété (***DatatypeProperty***);

Les types de données UML sont transformés en schéma XML (**XSD**(14)) types de données car OWL utilise la majorité des types de données intégré au schéma XML. Les instances de la primitive, les types utilisés dans UML comprennent eux-mêmes: ***Boolean***, ***Integer*** et ***String*** **[47].** Le tableau suivant présente la primitive UML, les types de données et leurs transformations.

Les types de données	
UML	XSD
Integer	xsd:integer
Boolean	xsd:boolean
String	xsd:String

Tableau 3.3 : les différents types UML et leur transformation

L'UML spécifie deux types de champs pour les attributs: instance et l'identifiant de la classe, et ce dernier est représenté par des noms soulignés. **[48]**

Les membres d'instance sont regroupés dans une instance spécifique. Les valeurs des attributs peuvent varier entre les instances, et l'invocation de la méthode peut affecter l'état de l'instance (c'est-à-dire changer les attributs de l'instance). **[8]**

- pour le code, il sera comme suit :

owl:DatatypeProperty rdf:about="#***nom_attribut***">

<rdfs:domain rdf:resource="#***nom_de_la_classe***"/>

<rdfs:range rdf:resource="XMLSchema#***string***"/> // le type

</owl:DatatypeProperty> // balise de fermeture

L'identifiant de la classe est considéré comme un simple attribut; Par conséquent, il est converti en une propriété de type de données (***DatatypeProperty***). **[8]** Et parce que l'identifiant doit avoir une valeur unique et différente, nous déclarons la propriété comme une fonction Propriété avec classe ***FunctionalProperty*** comme suit:

owl:DatatypeProperty rdf:about="#***nom_attribut***">

<rdf:type rdf:resource="#FunctionalProperty"/>

<rdfs:domain rdf:resource="#***nom_de_la_classe***"/>

<rdfs:range rdf:resource="XMLSchema#***string***"/> // le type

</owl:DatatypeProperty> // balise de fermeture

2.4.1.4 Règle 4 : Encapsulation (la visibilité)

La solution que nous proposons pour garder la sémantique de L'encapsulation est de marquer le début des attributs avec des symboles, le tableau suivant représente les différentes visibilités et leurs symboles :

Visibilité	Symbole
Privé	(-)
Protégé	(~)
Public	(+)

Tableau 3.4 : les niveaux de visibilité et leurs représentations

- le code sera comme suit :

- **visibilité privé :**

owl:DatatypeProperty rdf:about="#-***nom_attribut***">

- **visibilité protégé :**

owl:DatatypeProperty rdf:about="#~***nom_attribut***">

- **visibilité public :**

owl:DatatypeProperty rdf:about="#+***nom_attribut***">

2.4.1.5 Règle 5 : les opérations

La solution que nous proposons pour représenter les opérations est parmi les atouts de notre méthode. Effectivement, nous proposons de créer pour chaque classes qui contient des relations, une «sous-classes» nommée "relations", puis créez des sous-classes de la classe "relations", ces sous classes représentent les relations.

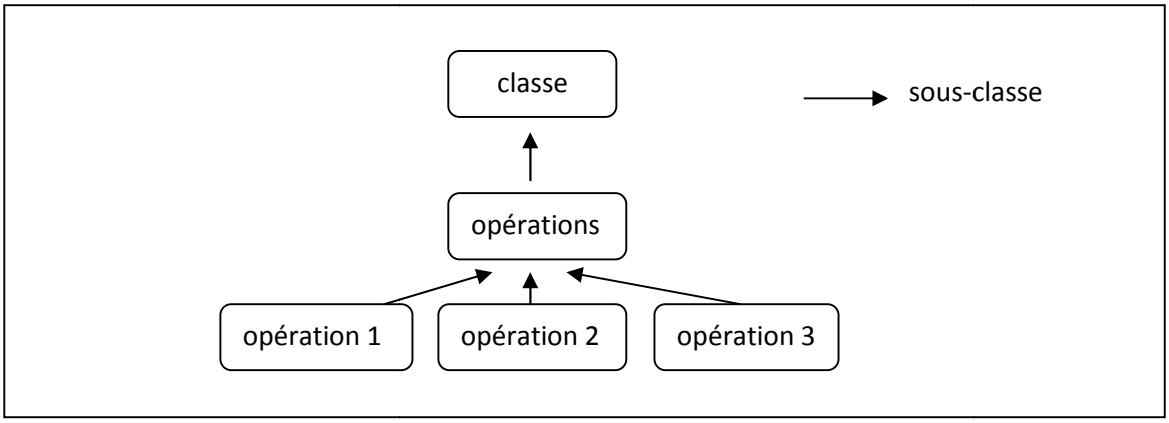


Figure 3.8 : hiérarchie de représentation des relations

Il ya deux types d'opérations, le premier possède des paramètres et le deuxième retourne une valeur.

- **Les opérations (return) :**

+capacité (): Integer;

Pour ce type d'opérations nous avons proposé pour les représenter cette structure, qui consiste a convertir l'opération en (***Owl class***) qui hérite de la classe "opération" et pour la valeur retourné pour ces opérations elle est converti en propriété de type de données (***DatatypeProperty***), nommé nom de la classe plus return et le type de cette propriété est le type retourné.

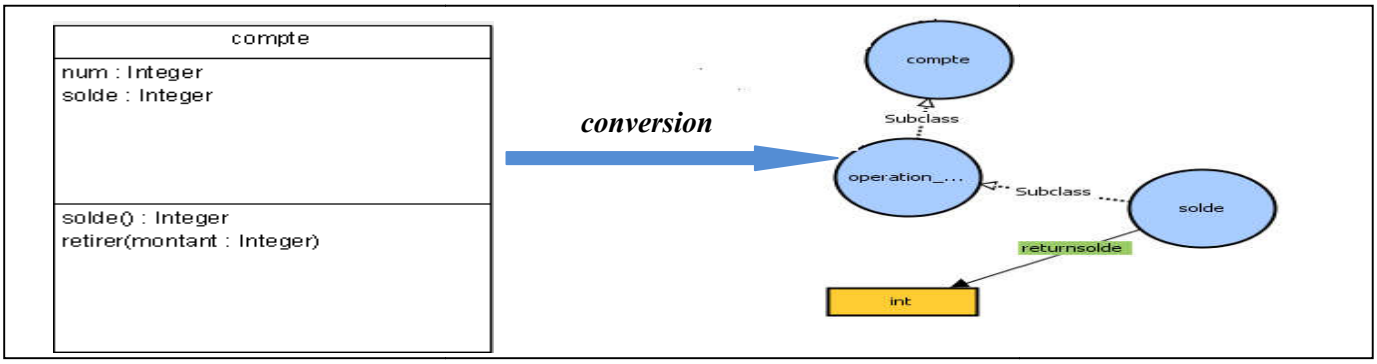


Figure 3.9 : représentation d'opération return

➢ le code sera comme suit :

- **création du sous classe opérations :**

<owl:Class rdf:about="#***nom_de_la_classe***">

<rdfs:subClassOf rdf:resource="#***operation_ nom_de_la_classe*** "/>

</owl:Class>

- **création du sous classe nommé par le nom d'opération :**

<owl:Class rdf:about="# ***operation_ nom_de_la_classe*** ">

<rdfs:subClassOf rdf:resource="# ***nom_d'opération*** "/>

</owl:Class>

- **création de la valeur retournée :**

<owl:DatatypeProperty rdf:about="# ***nom_d'opération _return***">

<rdfs:domain rdf:resource="# ***nom_d'opération*** "/>

<rdfs:range rdf:resource="XMLSchema#***int***"/>

</owl:DatatypeProperty>

- **les opérations avec paramètre:**

Pour ce type d'opérations nous avons proposé pour les représenter cette structure, qui consiste a convertir l'opération en (***Owl class***) qui hérite de la classe "opération" et aussi (***Owl class***) les paramètres qui hérite de la classe créée (à refaire), puis le paramètre est converti en propriété de type de données (***DatatypeProperty***).

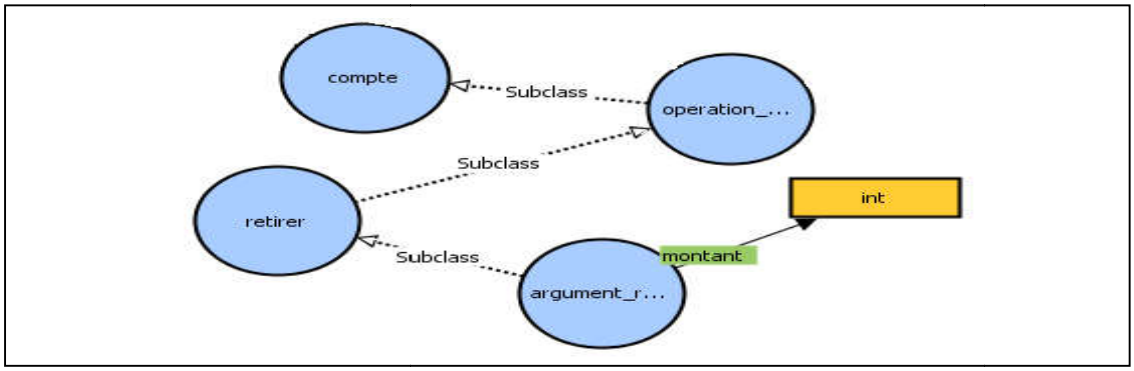


Figure 3.10 : représentation d'opération avec arguments

➢ le code sera comme suit :

- **création du sous classe opérations :**

<owl:Class rdf:about="#***nom_de_la_classe***">

<rdfs:subClassOf rdf:resource="#***operation_ nom_de_la_classe*** "/>

</owl:Class>

- **création du sous classe nommé par le nom d'opération :**

<owl:Class rdf:about="# ***operation_ nom_de_la_classe*** ">

<rdfs:subClassOf rdf:resource="# ***nom_d'opération*** "/>

</owl:Class>

- **création sous classe nommé argument (paramètre) :**

<owl:Class rdf:about="# ***nom_d'opération*** ">

<rdfs:subClassOf rdf:resource="# ***argument_nom_d'opération*** "/>

</owl:Class>

- **création de la valeur du paramètre :**

<owl:DatatypeProperty rdf:about="# ***nom_du_ paramètre*** ">

<rdfs:domain rdf:resource="# ***argument_nom_d'opération*** "/>

<rdfs:range rdf:resource="XMLSchema#***int***"/>

</owl:DatatypeProperty>

2.4.1.6 Les relations:

- **Relation unidirectionnelle (dépendance):**

Ce type d'association est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriété d'objet ***(ObjectProperty).***

➢ le code sera comme suit :

<owl:ObjectProperty rdf:about="#***nom_d'assiociation***">

<rdfs:domain rdf:resource="#***classe_émettrice***"/>

<rdfs:range rdf:resource="#***classe_cible***"/>

</owl:ObjectProperty>

- **Les associations:**

Une association est considérée comme deux rôles, ces rôles peuvent être mappé intuitivement par deux propriétés d'objet ***(ObjectProperty)***, une des deux classes est l'inverse de l'autre en utilisant la classe ***(ObjectProperty).***

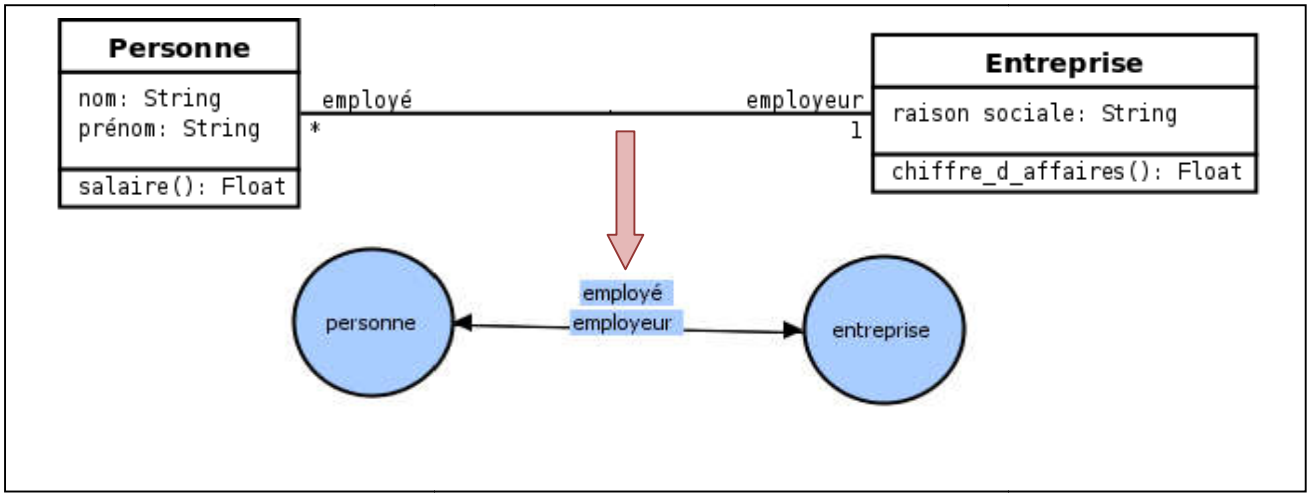


Figure 3.11 : représentation d'une association

- le code sera comme suit :

<owl:ObjectProperty rdf:about="#***nom_d'assiociation***">

<rdfs:domain rdf:resource="#***classe_émettrice***"/>

<rdfs:range rdf:resource="#***classe_cible***"/>

<Owl: inverseOf rdf: resource = "# ***nom_d'assiociation _inverse***" />

</ Owl: ObjectProperty>

- **Les types d'associations:**

Parmi les difficultés qui se posent à ce niveau, la représentation de ces types de relations (agrégation, composition, ou association simple). Pour remédier à ce problème, plusieurs

solutions ont été proposées, parmi lesquelles, ajoutant propriétés d'annotation à la propriété d'objet qui représente la relation. Cette possibilité est offerte par OWL DL **.[49]**

Cependant, ces propriétés n'ont pas de valeur sémantique; ils ne sont pas utilisés par les raisonnants dans le processus de raisonnement, qui conduit à une imperfection sémantique des diagrammes convertis. Notre proposition consiste de les structurer comme des propriétés d'objet ***(ObjectProperty),*** et ajouter pour chaque type une fonction de propriété comme mot clé, cela pour les séparer et pour garder la sémantique des diagrammes convertis.

- **L'agrégation**

Ce type d'association est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriétés d'objet ***(ObjectProperty).*** Et on a ajouté une fonction de propriété (***transitive*** **)** pour garder la sémantique**.**

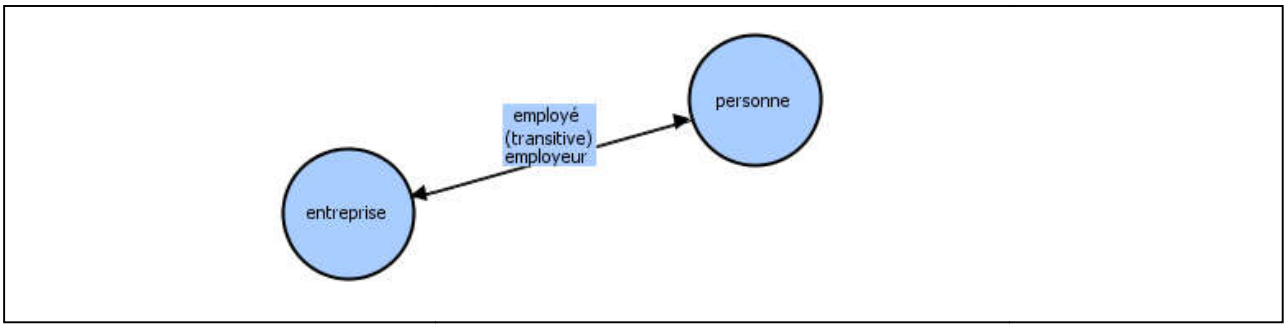


Figure 3.12 : représentation d'une association d'agrégation

➢ le code sera comme suit :

<owl:ObjectProperty rdf:about="#***nom_d'assiociation***">

<rdf:type rdf:resource="#***TransitiveProperty***"/>

<rdfs:domain rdf:resource="#***classe_émettrice***"/>

<rdfs:range rdf:resource="#***classe_cible***"/>

</owl:ObjectProperty>

- **La composition**

Ce type d'association est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriété d'objet ***(ObjectProperty).*** On a ajouté deux fonctions de propriété (***transitive*** **) et (** ***asymetrique*****)** pour garder la sémantique**.**

- le code sera comme suit :

<owl:ObjectProperty rdf:about="#***nom_d'assiociation***">

<rdf:type rdf:resource="#***AsymmetricProperty***"/>

<rdf:type rdf:resource="#***TransitiveProperty***"/>

<rdfs:domain rdf:resource="#***classe_émettrice***"/>

<rdfs:range rdf:resource="#***classe_cible***"/>

</owl:ObjectProperty>

➢ Les cardinalités:

OWL Classes: cardinalité, ***MaxCardinalité***, ***MinCardinality***, permettent d'appliquer les restrictions de cardinalité et de valeur à la propriété reliant deux concepts d'ontologie. Nous utilisons ces classes pour représenter UML Cardinalités en appliquant des restrictions de cardinalité à l'objet Propriétés représentant les relations du diagramme. Si L'association est une contrainte d'intégrité (cardinalité 1..1), nous proposons d'utiliser une propriété fonctionnelle pour la liaison de la Restriction de cardinalité à la valeur 1.

cardinalités	Restriction
0..1	<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:maxCardinality>
1..1	<rdf:type rdf:resource="&owl;FunctionalProperty" />
1..*	<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger"> 1 </owl:minCardinality>
*	no restriction

Tableau 3.5 : règles de conversion des cardinalités

2.4.2 Diagramme de séquence :

2.4.2.1 Régle1: (objets et acteurs)

Avant de commencer la conversion, nous créons une classe OWL appelé Objet (*OWL: thing*) pour représenter la superclasse Objet de UML.

Rappelons que toutes les classes de diagramme UML héritent par défaut de la classe d'objet. Ensuite, chaque objet et acteur du diagramme de séquence est converti en un concept OWL avec le même nom 11,donc : ils seront transformé a un concept owl: Class ;

*<owl:Class rdf:about= "#**nom_d'acteur**"/>*

*<owl:Class rdf:about= "#**nom_d'objet**"/>*

2.4.2.2 régle2: (messages)

- **Les messages synchronisés**

Un message synchronisé est considéré comme deux rôles, ces rôles peuvent être mappé intuitivement par deux propriétés d'objet ***(ObjectProperty)***, une des deux classes est l'inverse de l'autre. L'inverse est un message de réponse. Et on a ajouté une fonction de propriété (***reflexive*** **)** pour garder la sémantique**.**

- Les messages de réponse : chaque message sera mappé par une propriété d'objet ***(ObjectProperty)***

➢ le code sera comme suit :

Pour le message synchronisé :

<owl:ObjectProperty rdf:about="#***message***">

<rdf:type rdf:resource="#***ReflexiveProperty***"/>

<rdfs:domain rdf:resource="# ***objet_ou_acteur_émetteur***"/>

<rdfs:range rdf:resource="# ***objet_ou_acteur_cible***"/>

</owl:ObjectProperty>

Pour le message de réponse :

<owl:ObjectProperty rdf:about="#***message***">

<rdfs:domain rdf:resource="#***objet_ou_acteur_émetteur***"/>

<rdfs:range rdf:resource="#***objet_ou_acteur_cible***"/>

</owl:ObjectProperty>

- **Les messages désynchronisés :**

Ce type de message est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriétés d'objet ***(ObjectProperty)*** Et on a ajouté une fonction de propriété (***irreflexive*** **)** pour garder la sémantique**..**

- le code sera comme suit :

<owl:ObjectProperty rdf:about="#***message***">

<rdf:type rdf:resource="#***IrreflexiveProperty***"/>

<rdfs:domain rdf:resource="# ***objet_ou_acteur_émetteur*** "/>

<rdfs:range rdf:resource="# ***objet_ou_acteur _cible***"/>

</owl:ObjectProperty>

- **Les messages de création :**

Ce type de message sert à lancer la création des objets, il est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriétés d'objet ***(ObjectProperty)*** et on a ajouté le mot "***create***" plus le nom du message pour garder la sémantique**.**

- le code sera comme suit :

<owl:ObjectProperty rdf:about="#***create_message***">

<rdfs:domain rdf:resource="#***objet_ou_acteur_émettrice***"/>

<rdfs:range rdf:resource="# ***objet _cible***"/>

</owl:ObjectProperty>

- **Les messages de destruction :**

Ce type de message sert à lancer la destruction des objets, il est considéré comme un seul rôle, ce rôle peut être mappé intuitivement par une propriétés d'objet ***(ObjectProperty)*** et on a ajouté le mot "***supp***" plus le nom du message pour garder la sémantique**.**

- le code sera comme suit : `

<owl:ObjectProperty rdf:about="#***supp_message***">

<rdfs:domain rdf:resource="#***objet_ou_acteur_émettrice***"/>

<rdfs:range rdf:resource="# ***objet _cible***"/>

</owl:ObjectProperty>

- voici l'algorithme de conversion :

1: Entrée : diagramme de classe ou de séquence

2:SWITCH (choix) **{**

Créer la Classe "Object"(OWL thing)

choix1: //diagramme de classe

POUR chaque c.classe du diagramme **Faire**

Créer une classe "C.classeName" sous-classe de "C.classeParent"

Pour chaque A de C.attributList **Faire**

Créer DataTypeProperty "A.attVisibility +A.attributeName"

Avec le domaine "C.ClasseName"

Avec Range "getType (A.attributType)"

Si A.is_id **Alors**"A"

est InverseFunctionalProperty

Finsi

Finpour

Pour chaque O de C.opérationListe **Faire**

Créer une classe "opération+C.classeName" sous-classe de "C.classeParent"

Créer une classe "O.opérationVisibility+O.opérationName"sous-classe de "opération+C.classeName"

Si o.type == "return" **Alors**

Créer DataTypeProperty "return+O.opérationName"

Avec le domaine " O.opérationVisibility+ O.opérationName "

Avec Range "getType (O.returnType)"

Sinon

Créer une classe "O.opérationVisibility+O.opérationName"sous-classe de "opération+C.classeName"

Créer une classe " argument+ O.opérationName" sous-classe de " opération+C.classeName "

Créer DataTypeProperty "O.opérationArgument"

Avec le domaine " O.opérationVisibility+ O.opérationName "

Avec Range "getType (O.returnType)"

Finsi

Finpour

Pour chaque R de C.relationListe **Faire**

Créer une propriété "R" ObjectProperty"

Avec le domaine "C.ClasseName"

Avec Range "R.classCible" InverseOf"GetRelationName (C.className, R.classCible)"

Switch (cas) {

Cas 1: R.type =="agrégation"

"R" est TransitiveProperty

Cas 2: R.type =="composition"

"R" est TransitiveProperty && "R" est AsymetricProperty

}

RestrictionCardinalities (C.R.cardCible)

Fin pour

FIN POUR //Fin diagramme de classe

choix2: //diagramme de séquence

Pour chaque O.Objet du diagramme **Faire**

Créer une classe "O.ObjetName"

Finpour

Pour chaque A.Acteur du diagramme **Faire**

Créer une classe "A.ActeurName"

Finpour

Pour chaque M de M.MessageListe **Faire**

Créer une propriété "M" ObjectProperty"

Avec le domaine "C.ClasseName"

Avec Range"M.classCible"

Switch (cas) {

Cas 1: M.type =="synchronisé"

"M" est ReflexiveProperty

Cas 2: M.type =="a synchronisé"

"M" est IrreflexiveProperty

}

Cas 3: M.type =="create"

M."create+M.MessageName"

}

Cas 4: M.type =="supp"

M."supp+M.MessageName"

}

Fin pour

Figure 3.11 : Algorithme de conversion vers ontologie

3 Conclusion:

Après avoir présenté dans l'état de l'art quelques travaux de transformation de diagramme de classe du profil UML vers Ontologie, nous considérons que ces travaux ne prennent pas en compte la transformation des méthodes. Cependant, nous avons présenté dans cette section une approche qui consiste à convertir les diagrammes UML (classe et séquence) vers l'ontologie en se basant sur la conversion des méthodes et cela fait en proposant un

algorithme qui traite des règles pour transformer chaque catégorie de connaissances UML avec une propriété OWL d'ontologie.

Chapitre 4

Implémentation

1. Introduction

Dans le but d'une démonstration qui concrétise notre étude, nous présenterons dans ce chapitre une implémentation qui traduit le passage du modèle conceptuel décrit dans le chapitre précédent vers un produit informatique.

Pour l'implémentation de notre logiciel, il est tout à fait naturel d'utiliser un ensemble d'outils (plateforme, éditeurs..) ainsi que différents langages pour atteindre notre produit final.

2. Implémentation

L'implémentation consiste à traduire le résultat obtenu lors de l'étape de conception en un programme ou logiciel informatique exécuté sur machine en utilisant les outils de programmation adaptés au problème à traiter.

3. Langage de développement :

3.1 Le langage JAVA

Java est un langage de programmation orienté objet développé par Sun Microsystems (aujourd'hui racheté par Oracle).

Le choix du langage JAVA pour le développement de ce projet a été motivé pas les point suivants :

- **Portabilité** : En effet un programme développé en java peut s'exécuter sur n'importe quelle machine c'est-à-dire tout environnement (Windows, Unix et Mac), à condition bien sur que celle-ci dispose d'une machine virtuelle java (**JVM**(14) en anglais). **[50]**
- **Approche objet** : Elle est tout à fait adaptée à la manipulation de plusieurs types de documents, ou structures de données en tant qu'objet (ou même plusieurs versions du format d'un document peuvent être considérées comme des objets).
- **Fiabilité** : Java est très récent (1995), ce qui lui permet d'éviter les écueils de ses concurrent et les nombreux bugs souvent présent dans les anciens langages. **[51]**
- **Manipulation** : Java permet de manipuler les ontologies OWL à l'aide de l'API Jena En effet :
    - Il dispose d'une bibliothèque de classe très riches.

- Il est robuste : il permet bonne gestion de mémoire (pas d'accès directe à la mémoire) et des exceptions **[52]**

On peut faire de nombreuses sortes de programmes avec Java :

- Des applications, sous forme de fenêtre ou de console ;
- Des applets, qui sont des programmes Java incorporés à des pages web ;
- Des applications pour appareils mobiles, avec J2ME ;
- et bien d'autres ! J2EE, JMF, J3D pour la 3D. **[53]**

3.2 Le langage OWL

OWL (Ontologie Web Language) Le langage OWL est basé sur la recherche effectuée dans le domaine de la logique de description. Il peut être vu en quelque sorte comme un format de fichier pour certaines logiques de description. Il 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. **[53]**

3.3 RDF

RDF (Resource Description Framework)est un modèle de graphes pour décrire les (méta-) données et permettre un certain traitement automatique des métadonnées. Une des syntaxes (sérialisation) de ce langage est RDF/XML. Il s'agit d'un dialecte XML développée par le consortium W3C .

En annotant des documents non structurés et en servant d'interface pour des applications et des documents structurés (par ex . bases de données, GED , etc.) RDF permet une certaine interopérabilité entre des applications échangeant de l'information non formalisée et non structurée sur le Web. **[54]**

3.4 CSS :

Les feuilles de styles en cascade (CSS, pour *Cascading Style Sheets*) décrivent l'apparence des divers éléments d'une page web par le biais de couples *propriété* / *valeur*. Étant distinctes du code de la page (**HTML**(15) ou XML), elles constituent un moyen pour séparer structure et mise en page d'un site web ou d'une interface graphique . En tant que spécification du W3C.

Si l'on utilise le HTML pour déterminer la présentation dans un navigateur graphique, au lieu de se limiter à structurer le document, CSS c'est la beauté de l'informatique. **[55]**

4. Les outils de développement :

4.1 NetBeans

Cet **IDE**(16) a été créé à l'initiative de Sun Microsystems. Il présente toutes les caractéristiques indispensables à un environnement de qualité, que ce soit pour développer en Java, Ruby, C/C++ ou même **PHP**(17).

NetBeans est sous licence OpenSource, il permet de développer et déployer rapidement et gratuitement des applications graphiques Swing, des Applets, des JSP(15)/Servlets, des architectures J2EE, dans un environnement fortement personnalisable.**[56]**

L'IDE NetBeans repose sur un noyau robuste, la plateforme NetBeans, que vous pouvez également utiliser pour développer vos propres applications Java, et un système de plugins performant, qui permet d'avoir un IDE modulable.

A coté de la version complète de l'IDE NetBeans, il existe différentes déclinaisons qui se concentrent sur une plateforme ou un langage précis (Java ME, Java : SE + ME + EE, Ruby, C/C++, PHP).

NetBeans contient, en plus du support pour **CVS**(18) et SubVersion, un support pour ClearCase, mais aussi pour Mercuriale.

NetBeans détient un support de développement d'applications Web avec des améliorations pour l'édition des JSP, la gestion serveur et le support des dernières versions de Tomcat. **[56]**

Enfin cet IDE possède un débogueur de grande qualité ainsi qu'une interface graphique améliorée.

4.2 JAVA FX

JavaFX est une technologie créée par Sun Microsystems qui appartient désormais à Oracle, à la suite du rachat de Sun Microsystems par Oracle le 20 avril 2009.

Avec l'apparition de Java 8 en mars 2014, JavaFX devient la bibliothèque de création d'interface graphique officielle du langage Java, pour toutes les sortes d'application (applications mobiles, applications sur poste de travail, applications Web), le développement de son prédécesseur Swing étant abandonné (sauf pour les corrections de bogues). **[57]**

JavaFX est désormais une pure **API**(19) Java (le langage de script spécifique qui a été un temps associé à JavaFX est maintenant abandonné).

JavaFX contient des outils très divers, notamment pour les médias audio et vidéo, le graphisme 2D et 3D, la programmation Web, la programmation multi-fils etc.

Le SDK de JavaFX étant désormais intégré au JDK(17) standard Java SE, il n'y a pas besoin de réaliser d'installation spécifique pour JavaFX. **[57]**

Des projets libres complètent JavaFX en fournissant des composants de haute qualité absents de JavaFX proprement dit, voir par exemple : JFXtras et ControlsFX.

4.3 ArgoUML

ArgoUML est une application de diagramme UML écrite en Java et publiée sous la licence publique Eclipse open source. En vertu d'une application Java, elle est disponible sur n'importe quelle plate-forme prise en charge par Java SE. **[58]**

Selon la liste officielle des fonctions, **[59]** ArgoUML est capable de:

- Tous les 9 diagrammes UML 1.4 sont pris en charge. Sous les norme UML.
- Standard UML 1.4 Métamodèle.
- Prise en charge de XMI.
- Exportez les diagrammes en tant que GIF, PNG, PS, EPS, PGML et SVG.
- Importation inversée / importation JAR / classe.

4.4 Protégé

PROTEGE-OWL est une interface modulaire, développée au (*Stanford Medical Informatics*)de l'Université de Stanford, permettant l'édition ,la visualisation ,la contrôle

textuelles, et la fusion semi-automatique d'ontologies .Le modèle de connaissances de (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.**[60]**

L'interface, très bien conçue, et l'architecture logicielle permettant l'insertion de plug-ins apporter de nouvelles fonctionnalités (par exemple, la possibilité d'importer et d'exporter les ontologies construites dans divers langages opérationnels de représentationnels 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'autre outils.

Les points forts de Protégé:

- construire des ontologies.
- personnaliser des formulaires d'acquisition des connaissances.
- transférer la connaissance de domaine.
- faire des contrôles de cohérence de l'ontologie.

5. Présentation de l'application :

Cette partie est consacrée à la description de phase de réalisation et d'implémentation de ce projet, on va présenter quelques interfaces afin d'illustrer plus clairement les diverses utilisation de l'application.

La page d'accueil représente la première page de notre application. Elle sert généralement de présentation et contient trois boutons, Dessiner, Importer et quitter. Ce dernier et pour sortir de l'application. La figure suivante montre la page d'accueil de l'application « UML vers OWL converter » :

Figure 4.1 la page d'accueil de l'application

Par la suite, la figure suivante montre l'interface ArgoUML, après avoir cliquer sur le bouton dessiner :

Figure 4.2 : interface ArgoUML

Après avoir dessiner notre diagramme et l'exporter sous format XMI, nous allons l'importer on cliquant sur le bouton Importer :

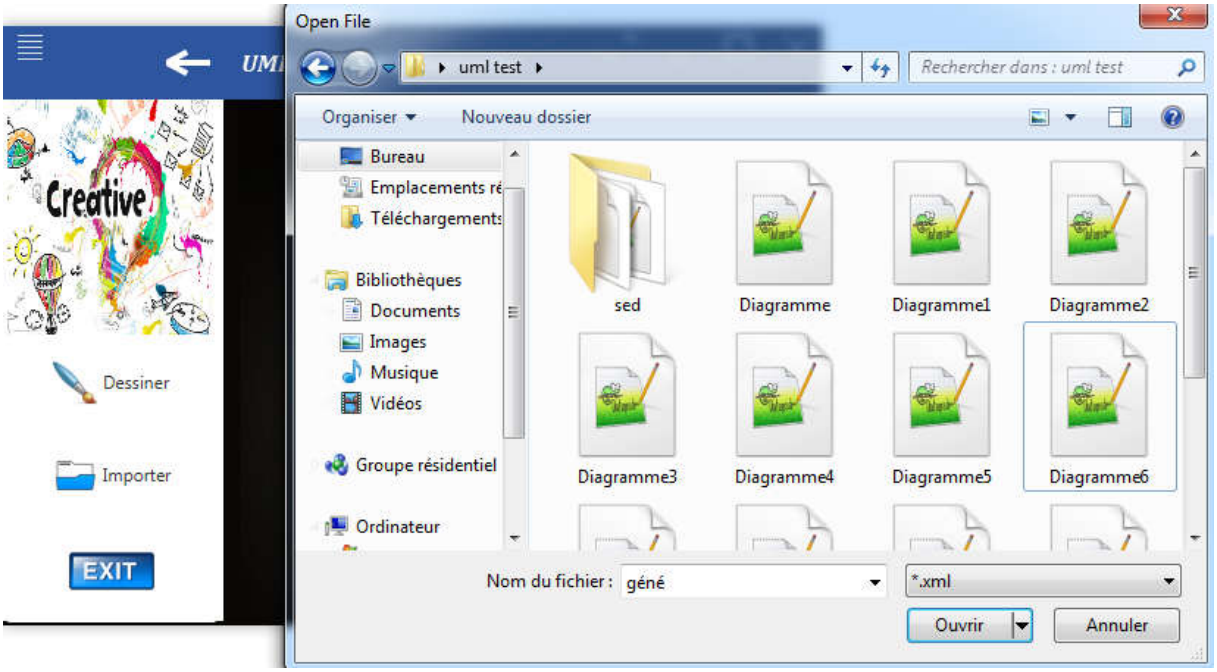


Figure 4.3 : Importation du diagramme dessiné

Une nouvelle fenêtre sera afficher avec deux onglets, le premier pour afficher le code importer comme le montre la figure suivante :

Figure 4.4 : Affichage du fichier XMI

Mais le deuxième onglet a trois boutons, converter, sauvegarder et visualiser. Les deux derniers sont désactivés, il seront actifs après la conversion, tous sa est présenté par la figure suivante :

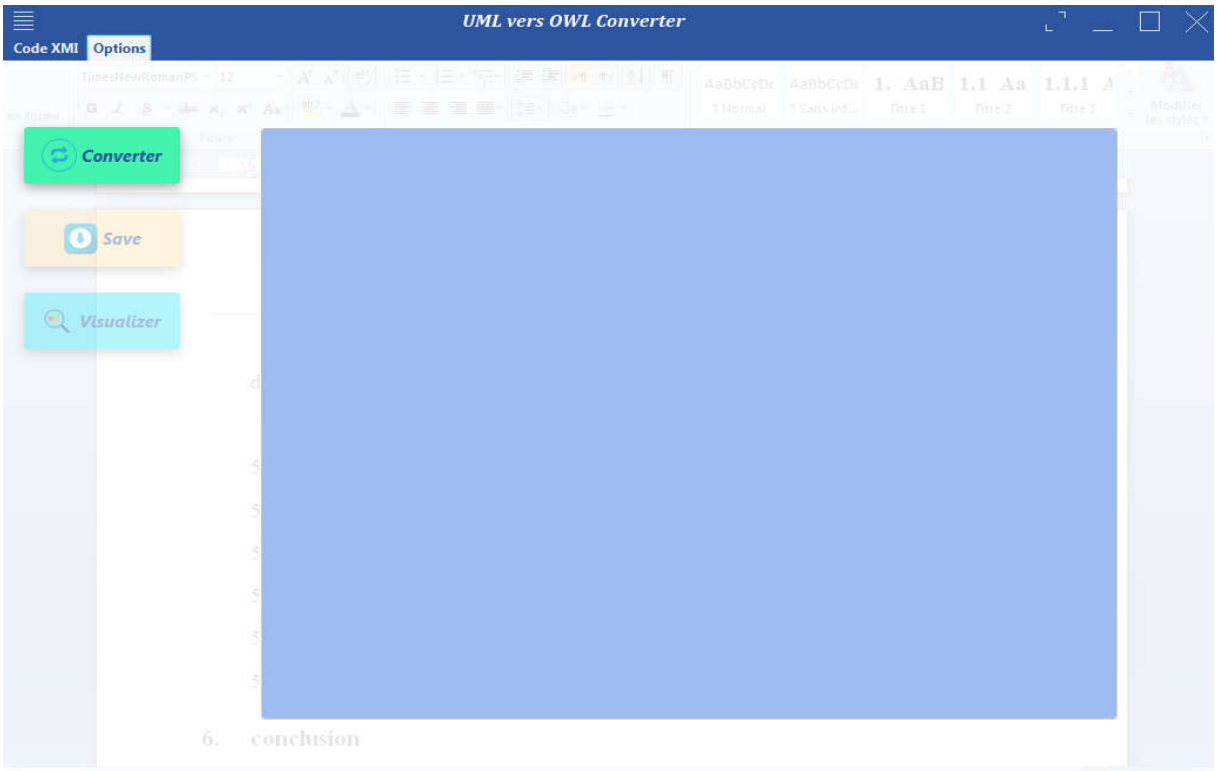


Figure 4.5 : Interface du transformation

Pour lancer la Transformation il faut cliquer sur le bouton ***converter***, les autres boutons seront activés et le code sera afficher comme le montre la figure suivante :

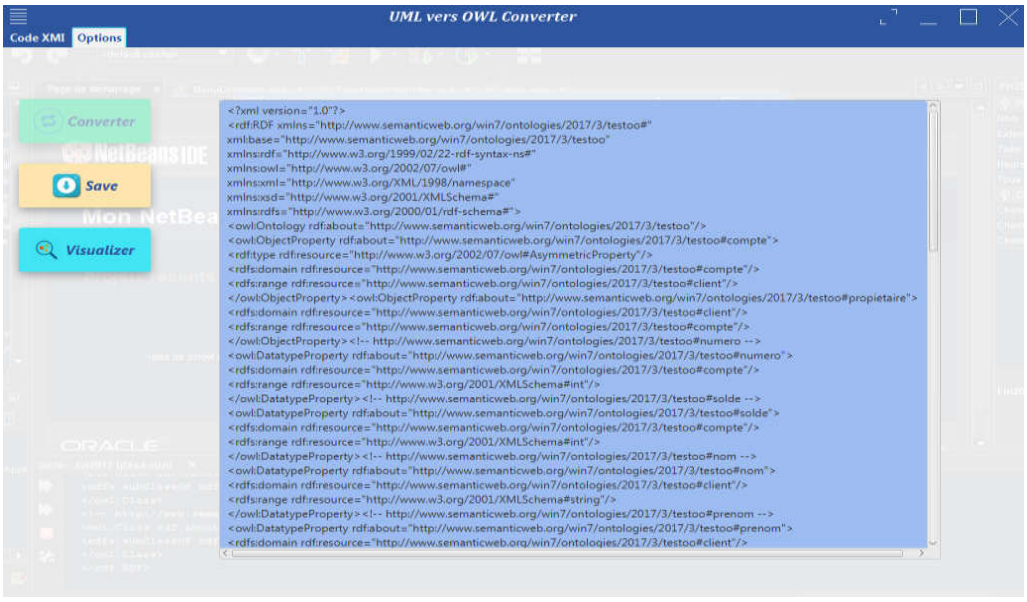


Figure 4.6 : Affichage du code OWL

La figure suivante montre l'enregistrement du code transformé dans un fichier OWL en cliquant sur le bouton ***sauvegarder*** :

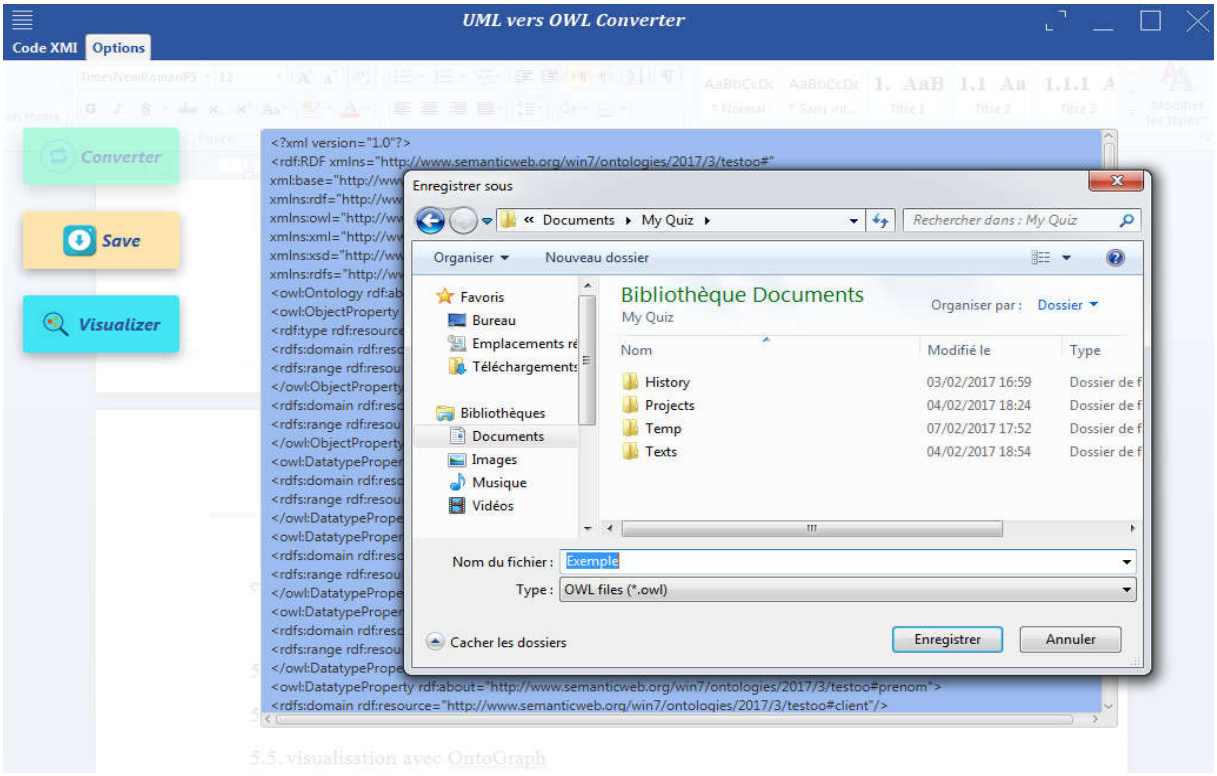


Figure 4.7 : Fenêtre d'enregistrement du code OWL

La visualisation du fichier sera effectué en cliquant sur le bouton ***visualiser,*** en utilisant Protégé. Ce dernier peut visualiser notre ontologie avec plusieurs bibliothèques, pour notre cas on va utilisé "OntoGraph" et "VOWL", comme montrent les figures suivantes :

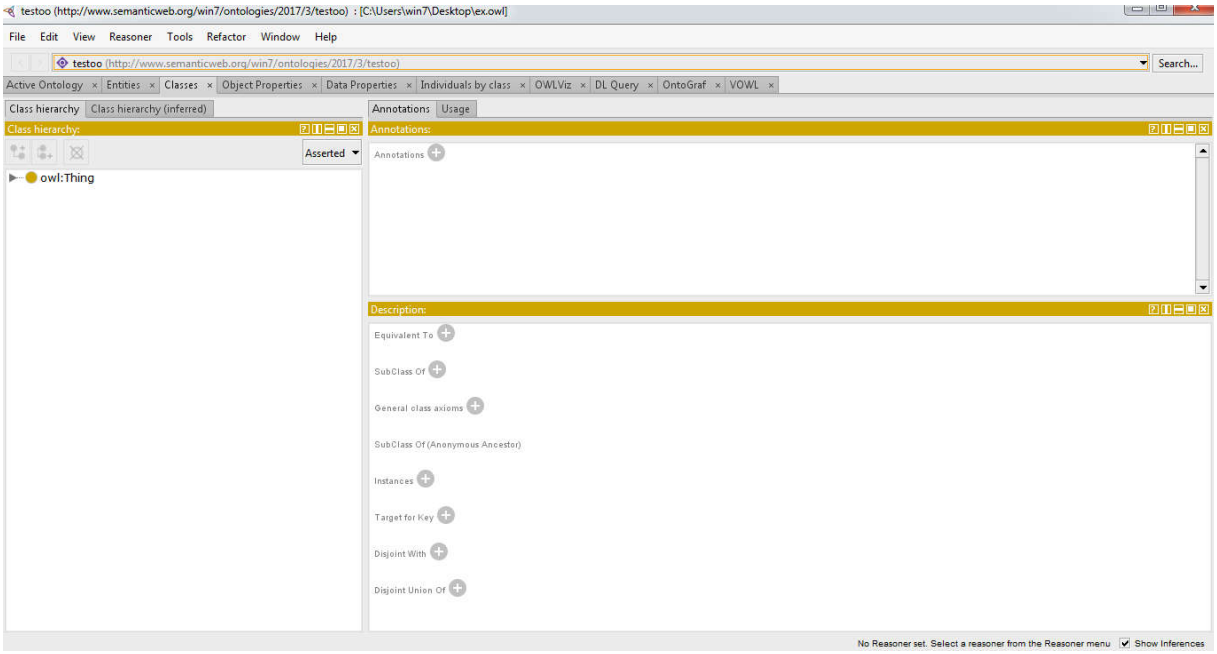


Figure 4.8 : L'interface du Protégé

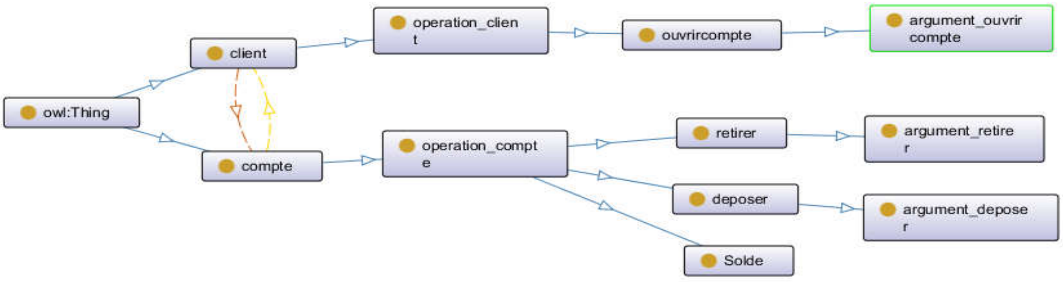


Figure 4.9 : visualisation avec OntoGraph

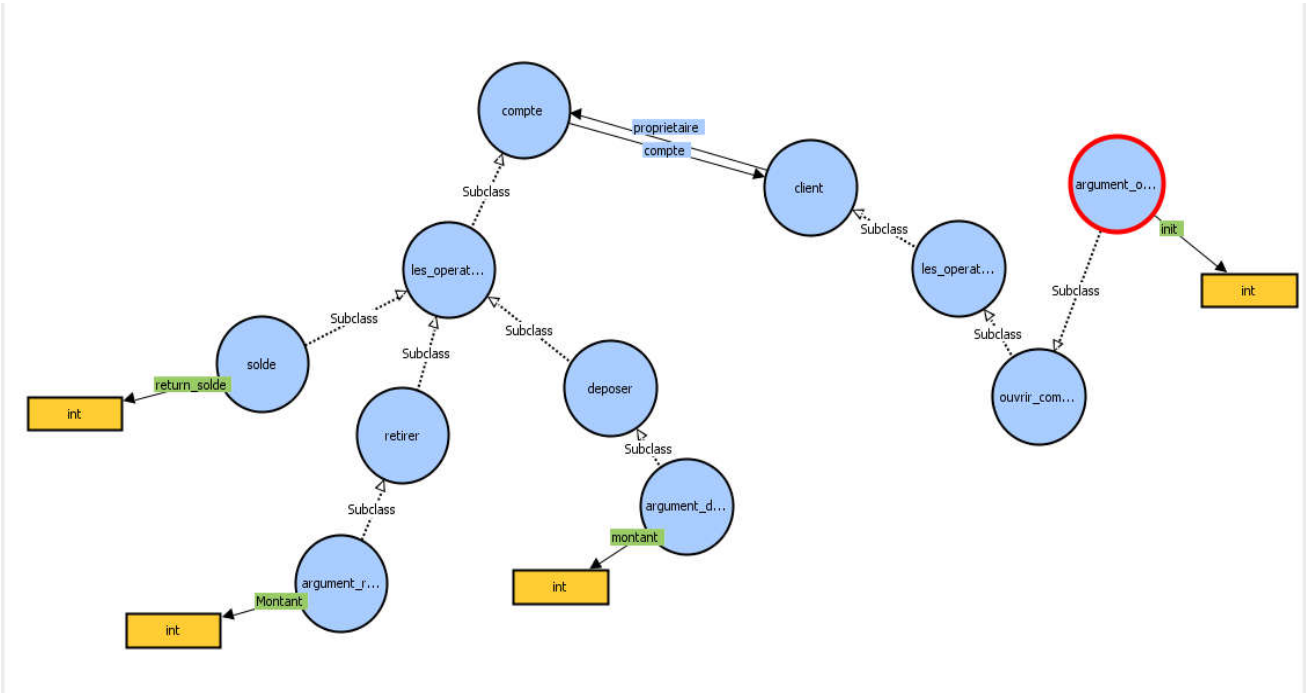


Figure 4.10 : visualisation avec OntoGraph

6. conclusion

Dans ce chapitre, nous avons présenté une implémentation qui traduit le passage du modèle conceptuel décrit dans le chapitre précédent vers un produit informatique. Nous avons présenté les outils et environnements de développement utilisés, ainsi que la création de

l'ontologie a partir des diagramme UML (classe et séquence) basé sur les règles de transformation et un algorithme précis.

Conclusion Générale

Le présent mémoire est le fruit du travail réalisé dans le cadre de fin d'études visant l'obtention du diplôme de master en informatique académique. Dans ce contexte, nous nous intéressons au domaine de l'ingénierie des systèmes d'information, et en particulier à la modélisation et l'implémentation des systèmes d'information.

Pour programmer une application, il faut modéliser, organiser ses idées, les documenter, puis organiser la réalisation en définissant les modules et étapes de la réalisation. C'est cette démarche antérieure à l'écriture que l'on appelle ***modélisation***.

L'objectif de ce travail était l'implémentation d'un algorithme qui assure la génération automatique des ontologies a partir d'un diagramme de classe d'un système d'information a fin de tirer profit de l'expressivité visuelle du langage de notation UML (***Unified Modeling Language***) et la puissance des ontologies.

Dans notre travail nous avons concentré d'une part sur la modélisation des diagrammes UML (classe et séquence) en utilisant l'outil **ArgoUML** version **0.34** développée par Jason E. Robbins, mais maintenant il est «**open source**» hébergé par **Tigris.org.**

Tout d'abord, la modélisation avec **ArgoUML** nous a permis de modéliser les diagrammes UML (classe et séquence), il permet de définir et de dessiner les diagrammes d'une manière simple, ainsi qu'il est moins complexe et efficace que les autres langages de modélisation, comme il est une application Java, ce qui facilite notre travail.

En générale, l'utilisation d'**ArgoUML** pour la modélisation conduit à avoir un modèle en format **XMI** (***XML Metadata Interchange***) qu'on peut le comprendre et l'utiliser pour faire la transformation vers l'ontologie.

Certainement, cette implémentation nous a permis de créer une ontologie faisant appel à plusieurs méthodes et fonctions en suivant des règles précises, cette application réalise l'objectif de notre thématique, elle assure la génération automatique des

ontologies a partir d'un diagramme de classe ou de séquence, en exécutant un algorithme très efficace.

En conclusion, Cette transformation permet de construire une ontologie qui définie 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. Pour tester la consistance sémantique de notre ontologie résultante, nous l'avons chargé sous Protégé.

Liste des références

[1] laurent-audibert, «Cours-UML ». Disponible sur : http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-3-5

[2] Laurent Perochon , «Unified Modeling Langage : UML ». Disponible sur :https://projet-plume.org/ressource/uml

[3] Boubker Sbihi, «informatisation d'une médiathèque à travers la norme UML». Disponible sur : https://www.epi.asso.fr/revue/articles/a0509b.htm

[4] Pascal Roques, «pourquoi UML ». Disponible sur http://www.umlsysml.org/modelisation-objet/pourquoi-uml

[5] «Les méthodes objet et la genèse d'UML». Disponible sur : http://uml.free.fr/cours/i-p5.html

[6] Tho-Hau Nguyen, «Modèle des Cas d'utilisation ». Disponible sur : http://www.grosmax.uqam.ca/nguyen_tho/INF7215/PDF/Le%20mod%C3%A8le%20des%20Use%20Cases.pdf

[7] Michel VOLLE, «Références sélectives sur UML ». Disponible sur : http://www.volle.com/travaux/umlimp.htm

[8] «diagramme d'objets ». Disponible sur : https://fr.wikipedia.org/wiki/Diagramme_d%27objets

[9] Delphine Longuet, «Diagrammes états-transitions». Disponible sur : https://www.lri.fr/~longuet/Enseignements/14-15/Et3-UML/Et3-DiagEtatsTransitions.pdf

[10] James Sugrue, «Getting Started with UML». Disponible sur : https://dzone.com/refcardz/getting-started-uml

[11] James, «UML 2.0 Cheatsheet ». Disponible sur : https://fr.scribd.com/doc/399157/UML-2-0-Cheatsheet

[13] Allen I. Holub, «UML REFERENCE CARD ». Disponible sur : http://www.digilife.be/quickreferences/QRC/UML%20Reference%20Card.pdf

[14] Gayo Diallo, «Une Architecture a base d'Ontologies». Disponible sur : https://tel.archives-ouvertes.fr/tel-00221392/document

[15] Gruber, T. R. (1993). A translation approach to portable ontologies. Knowledge Acquisition, 5(2):199-220

[16] Welty, C. et Nancy, I. (1999). Using the right tools : enhancing re-trieval from marked-up documents. J. Computers and the Humanities, 33(10):59-84.

[17] Guarino, N. (1998a). Formal ontologies and information systems. In FOIS'98,Trento, Italy, IO Press.

[18] Gomez-Perez, A. et Rojas, M. (1999). Ontological reengineering and reuse. In European Knowledge Acquisition Workshop (EKAW).

[19] Guarino, N. (1998b). Some ontological principles for designing upper level lexical resources. CoRR, cmp-lg/9809002.

[20] Fürst,f (2009) ,Axiom-based ontology matching.

[21] Guarino, N. (1999). Ontoseek : Content-based acess to the web. IEEE Intelligent Systems

[22] Jarrar, M. et Meersman, R. (2002). Formal ontology engineering in the dogma approach. Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics (ODBase 02), LNCS 2519:1238 -1254

[23] van Heijst, G., Schreiber, A. et Wielinga, B. (1997). Using explicit ontologies in kbs development. Int. J. of Human-Computer Studies, 46(2/3):183292.

[24] laurent-audibert, «Cours-UML ». Disponible sur : http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-3-5

[25] «enterprise ontology ». Disponible sur :http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html

[26] Wielinga, B. J., Schreiber, A. et Breuker, J. (1992). Kads : A modelling approach to knowledge engineering. Knowledge Acquisition, Special issue "The KADS Approach to Knowledge Engineering", 4(1):5-53.

[27] Sowa, J. (1984). Conceptual structures : Information processing in mind and machine. Addison Wesley.

[28] Lenat, D. B. et Guha, R. V. (1990). Building large knowledge based systems. Reading, Massachusetts : Addison Wesley.

[29] Gruber, T. R. (1992). Ontolingua : A mechanism to support portable ontologies. tech. report. Rapport technique, Stanford University, Knowledge Systems Laboratory.

[30] Genesereth, M. et Fikes, R. (1992). Knowledge interchange format - version 3 - reference manual. Rapport technique, Stanford University, Logic Group, Logic-92-1, Stanford, CA.

[31] Chandrasekaran, B., Josephson, J. et Benjamins, V. R. (1998). The ontology of tasks and methods. Knowledge Acquisition Workshop, KAW98.

[32] Studer, R., Benjamins, V. R. et Fensel, D. (1998). Knowledge engineering : Principles and methods. Data Knowl. Eng., 25(1-2):161-197.

[33] Klai Sihem, "Construction d'une ontologie à partir de bases de donnees pour l'aide à la maintenance industrielle application ;turbine à vapeur ", A Thesis, 20 aout 1955 skikda University ,skikda,Algeria,2009

[34] Rilling, Juergen, et al. "A unified ontology-based process model for software maintenance and comprehension." Models in Software Engineering. Springer Berlin Heidelberg, 56-65.2006 .

[35] Lassila, O. et Swick, R. R. (1999). Resource description framework (rdf) model and syntax specication. World Wide Web Consortium W3C Recommendation 22 February 1999 / W3C /.

[36] Decker, S., Mitra, P. et Melnik, S. (2000). Framework for the semantic web : An rdf tutorial. 4(6):68-73.

[37] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L. et Patel-Schneider, P. F. (2001). Oil : An ontology infrastructure for the semantic web. In IEEE Intelligent Systems, volume 16, pages 38-45.

[38] Miles, A. et Brickley, D. (2005). Skos core guide. Rapport technique, 2nd W3C Public Working Draft 2 November 2005.

[39] McBride, B. (2004). The resource description framework (rdf) and its vocabulary description language rdfs. Handbook on Ontologies, pages 51-66.

[40] McGuinness, D. L., Fikes, R., Hendler, J. A. et Stein, L. A. (2002). Daml+oil : An ontology language for the semantic web. IEEE Intelligent Systems, 17(2)(5):72-80.

[41] Nardi, D. et Brachman, R. J. (2003). An introduction to description logics. pages 1-40.

[42] Mohamed Bahaj, Jamal Bakkas, « Automatic Conversion Method of Class Diagrams to Ontologies Maintaining Their Semantic Features ». Disponible sur : http://www.dl.icdst.org/pdfs/files/3da90e7961198c07ed0020085450545a.pdf

[43] Aissam Belghiat, Mustapha Bourahla, «Automatic Generation of OWL Ontologies from UML Class Diagrams Based on MetaModelling and Graph Grammars». Disponible sur : http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-3-5

[44] Imants Zarembo, Sergejs Kodors «Automatic Transformation of UML Geospatial Profile to OWL Ontologies». Disponible sur : http://www.quaesti.com/archive/?vid=1&aid=3&kid=160101-45&q=f6

[45] « Ontology Classes ». Disponible sur : http://mappings.dbpedia.org/server/ontology/classes/

[46] K. Kiko and C. Atkinson, "A Detailed Comparison of UML and OWL", 2008.

[47] OMG, "Unified Modeling Language (OMG UML) Superstructure", version 2.3, Disponible sur : http://www.omg.org/spec/UML/2.3/

[48] «UML tutorial ».: https://www.tutorialspoint.com/uml/

[49] M. R. Jensen, T. H. Møller Torben, B. Pedersen "Converting XML Data to UML Diagrams For Conceptual Data Integration". Data & Knowledge Eng., vol. 44, no. 3, pp. 323-346, 2003

[50] David J. Eck, « Introduction to Programming Using Java». Disponible sur : math.hws.edu/eck/cs124/downloads/javanotes6-linked.pdf

[51] «Java Glossary ». Disponible sur : web.cerritos.edu/jwilson/SitePages/java_language_resources/Java_Glossary.pdf

[52] Bernard Espisse: Introduction aux Ontologie « Explicitation de la sémantique dans les bases de données » : thèse magistère (ENSMA 2002 présenté Hondjack Dehainsala.

[53] Robert Sedgewick and Kevin Wayne, «Programming in java ». Disponible sur : introcs.cs.princeton.edu/home/chapter1.pdf

[54] Rmoul Asma, «Fusionnement des sites de réseaux sociaux dans une ontologie par rétro-ingénierie et AFC»./université de Skikda 2015.

[55] «CSS3 Introduction». Disponible sur : https://www.w3schools.com/css/css3_intro.asp

[56] «NetBeans ». Disponible sur : http://www.oracle.com/ Developer Tools/ NetBeans

[57] «JavaFX ». Disponible sur : https://fr.wikipedia.org/wiki/JavaFX

[58] Florent de LAMOTTE, «Présentation d'ArgoUML ». Disponible sur : http://lootre.free.fr/argopno/doc/presentation/Presentation.html

[59] Issue 1834 (May 5th, 2003): Implement undo. http://argouml.tigris.org/issues/show_bug.cgi?id=1834

[60] «Péotégé ». Disponible sur : protege.stanford.edu