

CHAPITRE 3

CONCEPTION ET IMPLEMENTATION D'UNE ONTOLOGIE FLOUE SUR LES VENTES DES VOITURES

1 Introduction

Dans ce chapitre nous allons étudier le problème de ce mémoire qui est la construction de l'ontologie floue. Pour y arriver, nous devons d'abord définir le langage de descriptions d'ontologies OWL. Nous avons choisi afin de codifier notre ontologie l'éditeur d'ontologies Protégé-OWL. Finalement, le plugin de fuzzy owl est utilisé afin d'ajouter la partie floue et de tester la consistance de l'ontologie.

2 Le langage OWL

Cette section constitue l'introduction à OWL (Web Ontology Language) proprement dite. Elle s'appuie grandement sur les Recommandations du W3C du 10 février 2004.

2.1 Objectifs et motivation

OWL est destiné à être utilisé quand l'information contenue dans les documents doit être traitée par des applications, par opposition aux situations où le contenu doit seulement être présenté (i.e. la forme). OWL est employé pour représenter explicitement la signification (i.e. le fond) des termes sans les vocabulaires et les relations entre ces termes. Cette représentation des termes et de leurs corrélations s'appelle une ontologie. OWL offre plus de facilités pour exprimer la signification et la sémantique que XML, RDF et RDFS. OWL va ainsi au-delà de ces langages dans sa capacité de représenter le contenu compréhensible par une machine sur le Web. OWL est une révision du langage d'ontologie du web DAML+OIL [40], intégrant les leçons apprises de la conception et de l'application de DAML +OIL [40].

2.2 Structure d'une ontologie OWL

2.2.1 Espaces de nommage

Afin de pouvoir employer des termes dans une ontologie, il est nécessaire d'indiquer avec précision de quels vocabulaires ces termes proviennent. C'est la raison pour laquelle, comme tout autre document XML, une ontologie commence par une déclaration d'espace de nom (parfois appelée « de nommage ») contenue dans une balise ***rdf:RDF***. Supposons que nous souhaitons écrire une ontologie sur une population de personnes ou, d'une manière plus générale, sur l'humanité. Voici la déclaration d'espace de nom qui pourrait être employée :

```
<rdf:RDF  
  xmlns = "http://domain.tld/path/humanite#"  
  xmlns:humanite= "http://domain.tld/path/humanite#"  
  xmlns:base = "http://domain.tld/path/humanite#"  
  xmlns:vivant = "http://otherdomain.tld/otherpath/vivant#"  
  xmlns:owl = "http://www.w3.org/2002/07/owl#"  
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"  
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
```

Les deux premières déclarations identifient l'espace de nommage propre à l'ontologie que nous sommes en train d'écrire. La première déclaration d'espace de nom indique à quelle ontologie se rapporter en cas d'utilisation de noms sans préfixe dans la suite de l'ontologie. La troisième déclaration identifie l'URI de base de l'ontologie courante.

La quatrième déclaration signifie simplement que, au cours de la rédaction de l'ontologie humanité, on va employer des concepts développés dans une ontologie vivant, qui décrit ce qu'est un être vivant. Les quatre dernières déclarations introduisent le vocabulaire d'OWL et les objets définis dans l'espace de nommage de RDF, du schéma RDF et des types de données du Schéma XML. Afin de simplifier l'écriture des URI dans la déclaration d'espace de nom et, surtout, dans les valeurs des attributs de l'ontologie, il est conseillé de définir des abréviations au moyen d'entités de type de document :

```
<!DOCTYPE rdf:RDF [
    <!ENTITY humanite "http://domain.tld/path/humanite#" >
    <!ENTITY vivant "http://otherdomain.tld/otherpath/vivant#" >
]>
```

Ainsi, la déclaration d'espace de nom initiale devient :

```
<rdf:RDF
    xmlns = "&humanite;"
    xmlns:humanite= "&humanite;"
    xmlns:base = "&humanite;"
    xmlns:vivant = "&vivant;"
    xmlns:owl = "http://www.w3.org/2002/07/owl#"
    xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
```

2.2.2 En-têtes d'une ontologie

Tout comme il existe une section d'en-tête <head>...</head> en haut de tout document XHTML bien formé, on peut écrire, à la suite de la déclaration d'espaces de nom, un en-tête décrivant le contenu de l'ontologie courante. C'est la balise **owl:Ontology** qui permet d'indiquer ces informations :

```
<owl:Ontology rdf:about="">
    <rdfs:comment>Ontologie décrivant l'humanité</rdfs:comment>
    <owl:imports
        rdf:resource="http://otherdomain.tld/otherpath/vivant"/>
    <rdfs:label>Ontologie sur l'humanité</rdfs:label>
    ...
```

- L'élément ***owl:Ontology*** rassemble la plupart des métadonnées OWL du document peut être nécessaire pour enregistrer les informations de version et importer les définitions dont le document dépend.
- L'attribut ***rdf:about*** fournit le nom ou une référence de l'ontologie. Lorsque sa valeur est " ", le cas habituel, le nom de l'ontologie correspond à l'adresse URI de base de l'élément ***owl:Ontology***.
- L'attribut ***owl:imports*** admet un seul argument, identifié par l'attribut ***rdf:resource***. L'importation d'une autre ontologie reporte le jeu entier de ses assertions dans l'ontologie courante. Pour la bonne utilisation de cette ontologie importée, elle devrait normalement être coordonnée à une déclaration d'espace de nommage.

2.3 Eléments du langage

Cette partie ne va pas reprendre toutes les finesses d'OWL, mais uniquement les plus importantes. Pour des explications exhaustives du rôle de chacun des éléments composant le vocabulaire d'OWL, il est recommandé de se reporter à la Recommandation du W3C [41].

2.3.1 Les classes

Une classe définit un groupe d'individus qui sont réunis parce qu'ils ont des caractéristiques similaires. L'ensemble des individus d'une classe est désigné par le terme « extension de classe », chacun de ces individus étant alors une « instance » de la classe.

2.3.1.1 Déclaration de classe

Les concepts les plus élémentaires dans un domaine devraient correspondre aux classes racines des divers arbres taxonomiques. Chaque individu du monde OWL est membre de la classe ***owl:Thing*** (Thing : la classe universelle). Chaque classe définie par l'utilisateur est donc implicitement une sous-classe de ***owl:Thing***. On définit les classes racines spécifiques d'un domaine en déclarant simplement une classe nommée. Le langage OWL définit également une classe vide : ***owl:Nothing***. Une classe peut ainsi se déclarer de manières différentes :

- l'indicateur de classe : La description de la classe se fait, dans ce cas, directement par le nommage de cette classe. Une classe « humain » se déclare de la manière suivante :

```
<owl:Ontology rd<owl:Class rdf:ID="Humain" />f:about="">
```

Il est à noter que ce type de description de classe est le seul qui permette de nommer une classe. Dans les cinq autres cas, la description représente une classe dite « anonyme », créée en plaçant des contraintes sur son extension.

- l'énumération des individus composant la classe : Ce type de description se fait en énumérant les instances de la classe, à l'aide de la propriété *owl:oneOf* :

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Damien" />
    <owl:Thing rdf:about="#Olivier" />
    <owl:Thing rdf:about="#Philippe" />
    <owl:Thing rdf:about="#Xavier" />
    <owl:Thing rdf:about="#Yves" />
  </owl:oneOf>
</owl:Class>
```

2.3.1.2 Héritage

Il existe dans toute ontologie OWL une superclasse, nommée Thing, dont toutes les autres classes sont des sous-classes. Ceci nous amène directement au concept d'héritage, disponible à l'aide de la propriété *subClassOf* :

```
<owl:Class rdf:ID="Humain">
  <rdfs:subClassOf rdf:resource="&etre;#EtreVivant" />
</owl:Class>
<owl:Class rdf:ID="Homme">
  <rdfs:subClassOf rdf:resource="#Humain" />
</owl:Class>
<owl:Class rdf:ID="Femme">
  <rdfs:subClassOf rdf:resource="#Humain" />
</owl:Class>
```

Enfin, il existe également une classe nommée `noThing`, qui est sous-classe de toutes les classes OWL. Cette classe ne peut avoir aucune instance.

2.3.2 Les instances de classe

2.3.2.1 Définition d'un individu

La définition d'un individu consiste à énoncer un « fait », encore appelé « axiome d'individu ».

On peut distinguer deux types de faits :

- les faits concernant l'appartenance à une classe : La plupart des faits concerne généralement la déclaration de l'appartenance à une classe d'un individu et les valeurs de propriété de cet individu. Un fait s'exprime de la manière suivante :

```
<Humain rdf:ID="Pierre">
  <aPourPere rdf:resource="#Jacques" />
  <aPourFrere rdf:resource="#Paul" />
</Humain>
```

Le fait écrit dans cet exemple exprime l'existence d'un Humain nommé « Pierre » dont le père s'appelle « Jacques », et qu'il a un frère nommé « Paul ». On peut également instancier un individu anonyme en omettant son identifiant :

```
<Humain >
  <aPourPere rdf:resource="#Jacques" />
  <aPourFrere rdf:resource="#Paul" />
</Humain>
```

Ce fait décrit, dans ce cas, l'existence d'un Humain dont le père se nomme « Jacques » et qui a un frère nommé « Paul ».

- les faits concernant l'identité des individus : Une difficulté qui peut éventuellement apparaître dans le nommage des individus concerne la non-unicité éventuelle des noms attribués aux individus. Par exemple, un même individu pourrait être désigné de plusieurs façons différentes.

L'exemple suivant permet de déclarer que les noms « Louis_XIV » et « Le_Roi_Soleil » désignent la même personne :

```
<rdf:Description rdf:about="#Louis_XIV">
  <owl:sameAs rdf:resource="#Le_Roi_Soleil" />
</rdf:Description>
```

2.3.2.2 Affecter une propriété à un individu

Une fois que l'on sait écrire une classe en OWL, la création d'une ontologie se fait par l'écriture d'instances de ces objets, et la description des relations qui lient ces instances.

2.3.3 Les propriétés

Maintenant que l'on sait écrire des classes OWL, il ne manque plus que la capacité à exprimer des faits au sujet de ces classes et de leurs instances. C'est ce que permettent de faire les propriétés OWL. OWL fait la distinction entre deux types de propriétés :

- les propriétés d'objet permettent de relier des instances à d'autres instances
- les propriétés de type de donnée permettent de relier des individus à des valeurs de données.

Une propriété d'objet est une instance de la classe *owl:ObjectProperty*, une propriété de type de donnée étant une instance de la classe *owl:DatatypeProperty*. Ces deux classes sont-elles même sous-classes de la classe RDF *rdf:Property*.

La définition des caractéristiques d'une propriété se fait à l'aide d'un axiome de propriété qui, dans sa forme minimale, ne fait qu'affirmer l'existence de la propriété :

```
<owl:ObjectProperty rdf:ID="aPourParent" />
```

2.3.3.1 Définition d'une propriété

Afin de spécifier une propriété, il existe différentes manières de restreindre la relation qu'elle symbolise. Par exemple, si on considère que l'existence d'une propriété pour un individu donné de l'ontologie constitue une fonction faisant correspondre à cet individu un autre individu ou une valeur de donnée, alors on peut préciser le domaine et l'image de

la propriété. Une propriété peut également être définie comme la spécialisation d'une autre propriété [43].

```
<owl:ObjectProperty rdf:ID="habite">
  <rdfs:domain rdf:resource="#Humain" />
  <rdfs:range rdf:resource="#Pays" />
</owl:ObjectProperty>
```

Dans l'exemple ci-dessus, on apprend que la propriété habite a pour domaine la classe Humain et pour image la classe Pays : elle relie des instances de la classe Humain à des instances de la classe Pays. Dans le cas d'une propriété de type de donnée, l'image de la propriété peut être un type de donnée, comme défini dans le Schéma XML. Par exemple, on peut définir la propriété de type de données anneeDeNaissance :

```
<owl:Class rdf:ID="dateDeNaissance" />
<owl:DatatypeProperty rdf:ID="anneeDeNaissance">
  <rdfs:domain rdf:resource="#dateDeNaissance" />
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

Dans ce cas, anneeDeNaissance fait correspondre aux instances de la classe de dateDeNaissance des entiers positifs. On peut également employer un mécanisme de hiérarchie entre les propriétés, exactement comme il existe un mécanisme d'héritage sur les classes *rdfs:subPropertyOf* :

```
<owl:Class rdf:ID="Humain" />
  <owl:ObjectProperty rdf:ID="estDeLaFamilleDe">
    <rdfs:domain rdf:resource="#Humain" />
    <rdfs:range rdf:resource="#Humain" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="aPourFrere">
    <rdfs:subPropertyOf rdf:resource="#estDeLaFamilleDe" />
    <rdfs:range rdf:resource="#Humain" /> .....
  </owl:ObjectProperty>
</owl:Class>
```


La propriété `aPourFrere` est une sous-propriété de `estDeLaFamilleDe`, ce qui signifie que toute entité ayant une propriété `aPourFrere` d'une certaine valeur a aussi une propriété `estDeLaFamilleDe` de même valeur.

2.3.3.2 Caractéristiques des propriétés

En plus de ce mécanisme d'héritage et de restriction du domaine et de l'image d'une propriété, il existe divers moyens d'attacher des caractéristiques aux propriétés, ce qui permet d'affiner grandement la qualité des raisonnements liés à cette propriété.

Parmi les caractéristiques de propriétés principales, on trouve :

- *TransitiveProperty*: si on définit une propriété `P` comme transitive, alors pour tout $x, y, z : P(x, y)$ et $P(y, z)$ implique $P(x, z)$.

Par exemple, on pourrait indiquer que la propriété `sousRégionDe` entre des régions est transitive :

```
<owl:TransitiveProperty rdf:ID="sousRégionDe">
  <rdfs:domain rdf:resource="#Région"/>
  <rdfs:range rdf:resource="#Région"/>
</owl:TransitiveProperty>
```

- *SymmetricProperty* : si une propriété `P` est symétrique, alors pour tout $x, y : P(x, y)$ iff $P(y, x)$.

Un exemple à la mode de propriété symétrique est la relation `estAmiDe` :

```
<owl:SymmetricProperty rdf:ID="estAmiDe">
  <rdfs:domain rdf:resource="#Humain"/>
  <rdfs:range rdf:resource="#Humain"/>
</owl:SymmetricProperty>
```

- *FunctionalProperty* : si une propriété `P` est marquée comme fonctionnelle, alors pour tout $x, y, z : P(x, y)$ et $P(x, z)$ implique $y = z$

```
<owl:ObjectProperty rdf:ID="aPourEpoux">
  <rdf:type rdf:resource="&owl;FunctionalProperty">
  <rdfs:domain rdf:resource="#Femme"/>
  <rdfs:range rdf:resource="#Humain"/>
</owl:ObjectProperty>
```

Cet axiome affirme que la propriété *aPourEpoux* est fonctionnelle, c'est-à-dire qu'une femme ne peut avoir qu'un époux au plus (ce qui est un bon exemple de la dépendance culturelle des ontologies).

- *InverseOfProperty* : si une propriété *P1* est marquée comme étant l'inverse de *P2*, alors pour tout *x, y*: $P1(x, y) \text{ iff } P2(y, x)$.

```
<owl:ObjectProperty rdf:ID="aPourEnfant">
  <owl:inverseOf rdf:resource="aPourParent"/>
</owl:ObjectProperty>
```

- *InverseFunctionalProperty* : si une propriété *P* est marquée comme fonctionnelle symétrique, alors pour tout *x, y, z*: $P(y, x)$ et $P(z, x)$ implique $y = z$.

```
<owl:InverseFunctionalProperty rdf:ID="est LaMèreBiologiqueDe">
  <rdfs:domain rdf:resource="#Femme"/>
  <rdfs:range rdf:resource="#Humain"/>
</owl:InverseFunctionalProperty>
```

2.3.4 Les contraintes de cardinalité

Dans le langage OWL comme dans RDF, toute instance d'une classe peut avoir un nombre arbitraire de valeurs (zéro à plusieurs) pour une propriété particulière. Pour imposer une propriété (au moins une), ou lui autoriser un nombre spécifique de valeurs, ou en interdire l'utilisation, on peut utiliser des contraintes de cardinalité. Le langage OWL offre trois structures pour restreindre localement la cardinalité des propriétés dans le contexte d'une classe.

- *owl:maxCardinality* : Une restriction avec une contrainte *owl:maxCardinality* décrit la classe de tous les individus au plus *N* valeurs sémantiquement distinctes (des individus ou des valeurs de données) pour la propriété concernée, où *N* est la valeur de la contrainte de cardinalité.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#aPourParent"/>
  <owl:maxCardinality
    rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>
```

Cet exemple décrit la classe des individus ayant au plus deux parents.

- *owl:minCardinality* : Une restriction avec une contrainte *owl:minCardinality* décrit la classe de tous les individus ayant au moins N valeurs sémantiquement distinctes (des individus ou des valeurs de données) pour la propriété concernée, ou N est la valeur de la contrainte de cardinalité.

L'exemple suivant décrit la classe des individus ayant au deux parents :

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#aPourParent"/>
  <owl:minCardinality
    rdf:datatype="&xsd;nonNegativeInteger">2</owl:minCardinality>
</owl:Restriction>
```

- *owl:cardinality* : Une restriction avec une contrainte *owl:Cardinality* décrit la classe de tous les individus ayant exactement N valeurs sémantiquement distinctes pour la propriété concernée, ou N est la valeur de la contrainte de cardinalité.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#aPourParent"/>
  <owl:Cardinality
    rdf:datatype="&xsd;nonNegativeInteger">2</owl:Cardinality>
</owl:Restriction>
```

Cet exemple décrit la classe des individus ayant exactement deux parents.

2.3.5 Les classes disjointes

On peut définir le caractère disjoint d'un ensemble de classes au moyen du constructeur **owl:disjointWith**. Il garantit que l'individu membre d'une classe ne puisse pas être simultanément l'instance d'une autre classe définie.

```
<owl:Class rdf:ID="Pasta" />
  < rdfs:subClassOf rdf:resource="#EdibleThing"/>
  < owl:disjointWith rdf:resource="#Meat" />
  < owl:disjointWith rdf:resource="#Dessert" />
  < owl:disjointWith rdf:resource="#Fruit" />
</owl:Class>
```

L'exemple de la classe des pâtes (Pasta) montre plusieurs classes disjointes. Remarquez que la définition fait seulement valoir que la classe pasta est disjointes par rapport à toutes les autres classes.

3. Editeur d'ontologies PROTÉGÉ

Protégé est un outil libre et open source qui a été développé par l'université de Stanford. Il permet de définir des vocabulaires ou des thésaurus et de représenter des connaissances par le biais d'ontologies. Protégé supporte les langages standards du W3C tels que XML, RDF et OWL. Le support d'OWL, comme de nombreux autres formats, est possible dans Protégé grâce à un plugin dédié.

Grace à toutes ces spécificités que dispose Protégé, notre choix portera sur cet outil pour faciliter l'implémentation de notre ontologie [40].

3.1 Création d'un projet PROTÉGÉ

A l'ouverture du l'éditeur protégé la fenêtre suivante s'affiche. C'est commandes de MS-DOS représente les appels à l'ensemble de plugins java que le protégé va les utiliser.

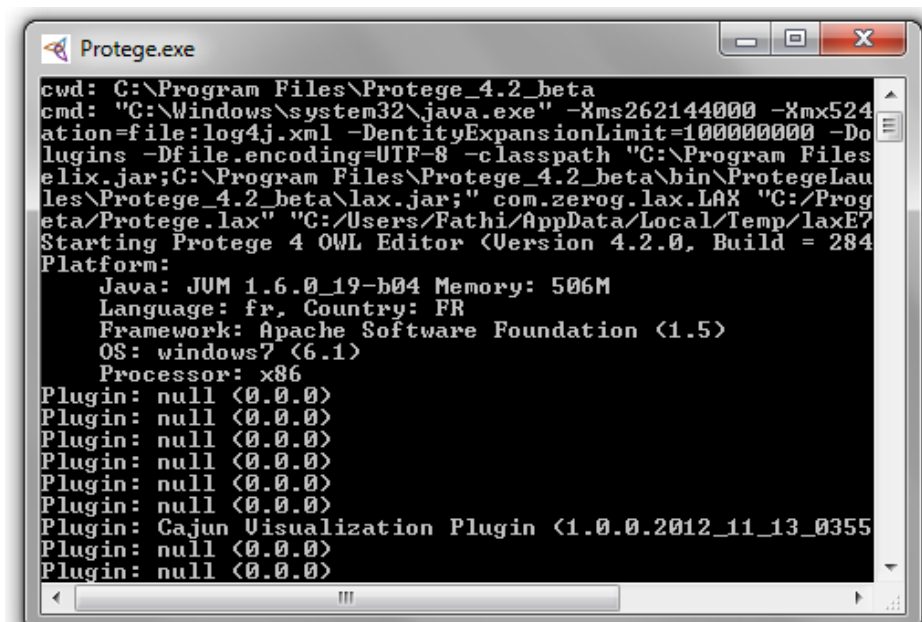


Figure 3.1 les appels à l'ensemble de plugins java

Pour la création d'un nouveau projet ou l'ouverture d'un projet déjà existant, on choisissant le sous menu File, et après *New* ou *Open*.

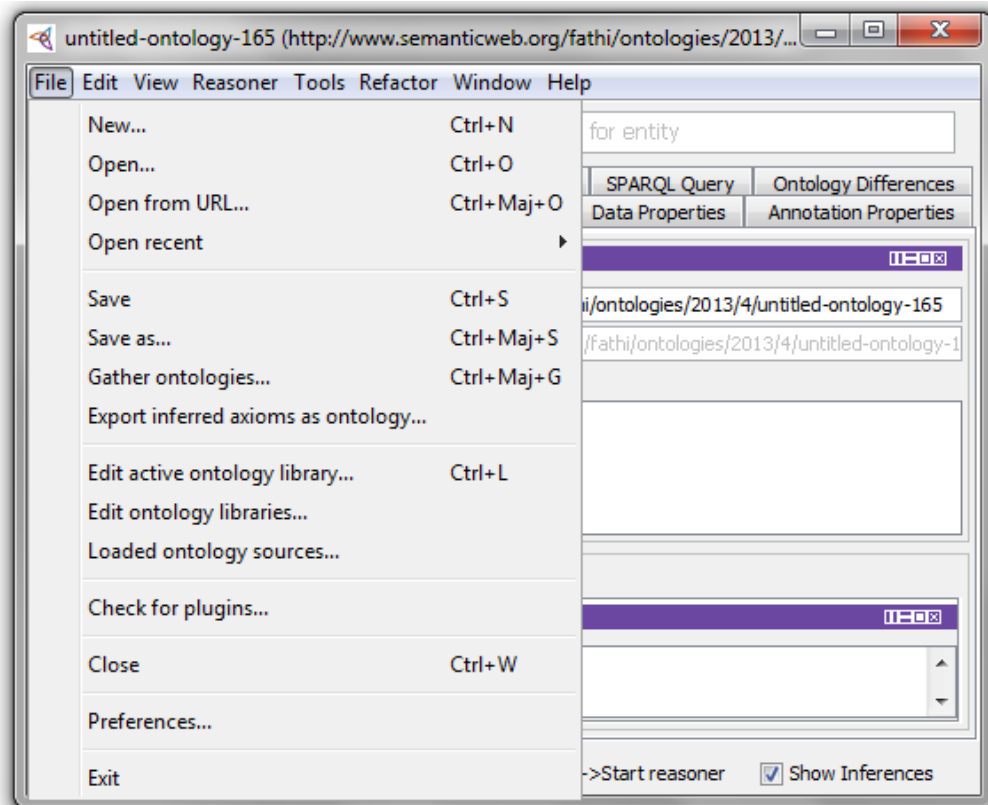


Figure 3.2 Création d'un nouveau projet

Pour enregistré le projet on choisisant *Save* dans le sous menu *File*, La fenêtre suivante apparaître qui permet d'enregistré l'ontologie dans plusieurs format. RDF/XML est le format par default pour les ontologies crée par protégé et enfin en clique sur le bouton Ok.

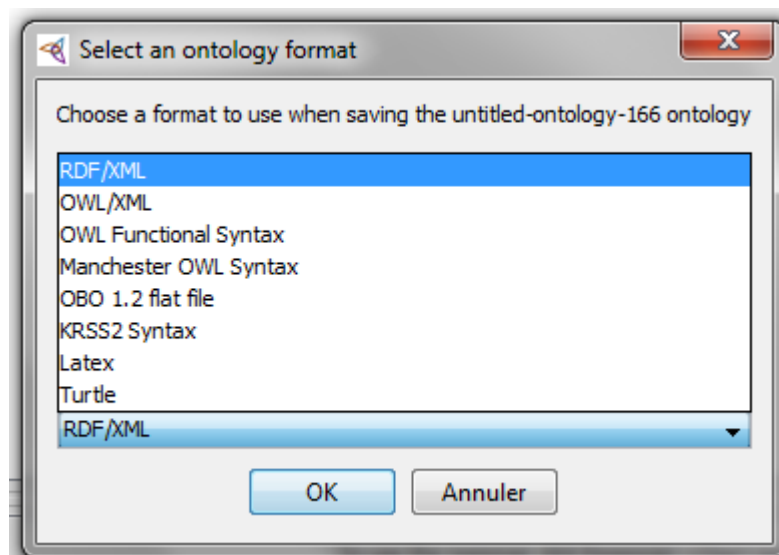


Figure 3.3 Sélection type de projet

3.2 Définition des classes et des propriétés

L'interface de PROTEGE est assez simple, l'ensemble des fonctionnalités de l'éditeur étant regroupé en onglets. Le premier onglet présente les classes de l'ontologie. Il est possible de créer, modifier, supprimer une classe, et de lui attacher des propriétés. Ces propriétés peuvent elles-mêmes être caractérisées. La capture d'écran de la figure 3.4 : Capture de l'écran principal de PROTEGE, onglet « classes ».

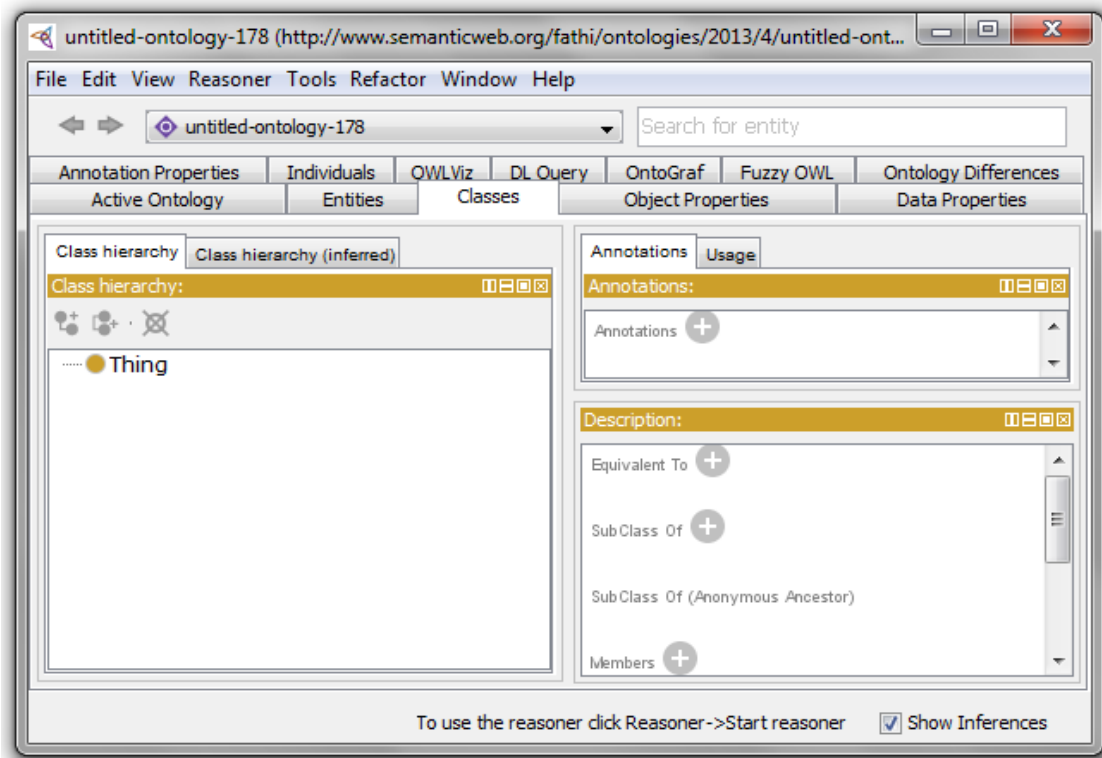


Figure 3.4 Capture de l'écran principal de Protégé, onglet "classes"

3.3 Gestion des instances de classe et leurs propriétés

Sur l'écran présenté ci-dessous, il est possible d'éditer les informations concernant n'importe individu. Il est important de comprendre que, dans l'Instance Browser, un individu est désigné par son identifiant (son « rdf : ID »), et non par son prénom. En ce sens, le nommage des individus de l'ontologie écrite en exemple est sans doute un peu ambigu, bien que tout à fait correct. Pour des raisons pédagogiques, il aurait été plus indiqué de nommer ces instances « homme1 », « homme2 », etc.

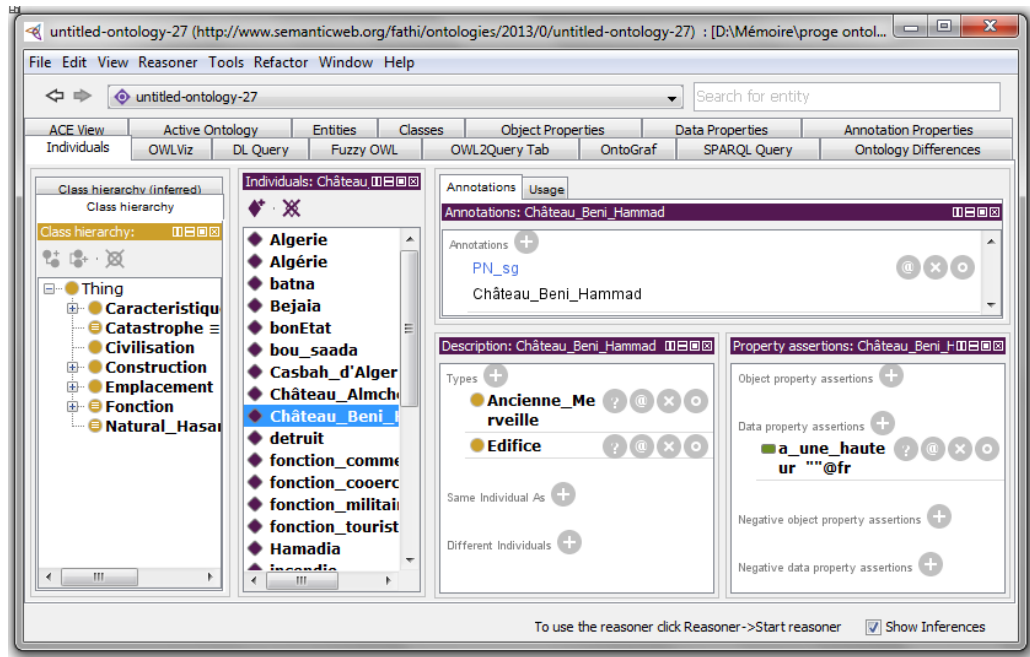


Figure 3.5 Editeur d'instances intégré à protégé

3.4 Possibilité d'effectuer des requêtes

Enfin, une fonctionnalité intéressante de PROTEGE concerne la possibilité d'effectuer des requêtes sur l'ontologie en cours d'édition [42]. L'onglet DL Query permet aux utilisateurs de tester rapidement les définitions des catégories de voir qu'ils subsument les sous-classes appropriées. On peut vérifier l'appartenance de classe des descriptions arbitraires sans avoir à créer des espaces réservés nom de classe.

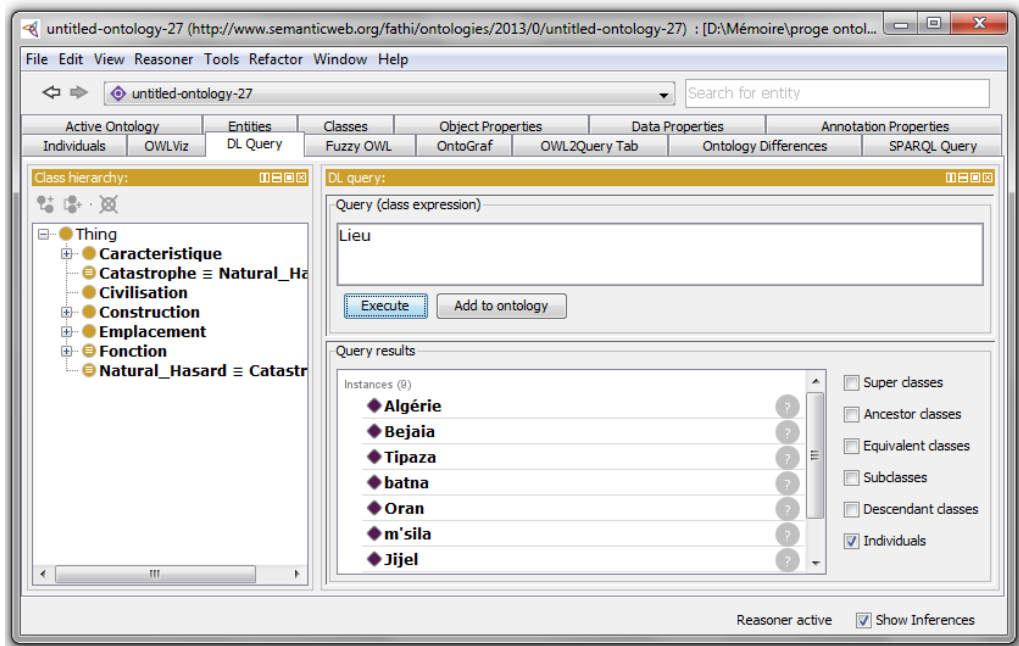


Figure 3.6 Gestionnaire de requetes de protégé

4. Conception détaillée de l'ontologie

Comme la construction d'une ontologie d'un domaine nécessite toute une équipe de personnel informatique et une équipe de spécialiste du domaine étudié ainsi qu'un temps énorme pour aboutir à une ontologie complète dans le sens où elle répond aux objectifs déjà prévus, nous avons décidé de construire une ontologie floue qui sera utilisée dans un système de recherche d'information afin d'exploiter son potentiel sémantique.

Dans notre travail nous avons fixé comme objectif principal la construction d'une ontologie floue, nous avons choisi comme domaine d'application le vente des voitures c'est-à-dire une ontologie qui regroupe les concepts fondamentaux des voitures et l'ensemble des relations de ces derniers.

4.1 Description de l'ontologie ventes des voitures :

Notre exemple est une version modifiée de l'ontologie de ventes des voitures qui est présentée dans [38], supposons qu'un vendeur vend une voiture berline. Un acheteur est à la recherche d'une deuxième voiture de tourisme. Ainsi que l'acheteur et le vendeur ont des préférences (restrictions). Voyons maintenant comment représenter les connaissances de domaine. Un concept "Acheter" regroupe toutes les préférences de l'ensemble d'acheteurs de telle sorte que le plus élevé est le degré maximal de satisfiabilité d'achat. Par exemple, l'acheteur dispose de 5 préférences A1-A5. Certaines préférences peuvent être plus importantes que les autres, donc nous utilisons un ensemble des poids dans le concept "préférences_des_acheteurs", nous décrivons ces poids plus tard.

Comme le concept "Acheter", le concept "Vendre" regroupe toutes les préférences de l'ensemble des vendeurs de telle sorte que le plus élevé est le degré maximal de satisfiabilité de vente, Les préférences des vendeurs sont représentées par un ensemble des poids combinant les 5 préférences. Ces derniers peuvent être codés comme V1-V5. L'ontologie inclut également des informations de fond sur le domaine des véhicules, ce que n'est pas montré dans cet exemple.

Enfin, il est clair que le meilleur accord entre l'acheteur et le vendeur est déterminé par le degré maximal de satisfiabilité de la conjonction Acheter et Vendre en utilisant la logique floue.

4.1 Construction d'un glossaire des termes

Une ontologie floue est une simple représentation d'une branche. L'ontologie en questions, regroupe les concepts de la branche, donc la recherche d'information au quel l'ontologie est destinés sera effectuée sur les concepts et leurs instances. Nous avons construit un glossaire des termes de notre ontologie.

Dans ce glossaire nous n'avons pas pris beaucoup attention à la redondance et à l'appartenance des concepts à la branche mais nous avons pris beaucoup plus d'intérêt à regrouper tout ce qu'est en relation avec le domaine pour pouvoir le délimiter.

Le tableau qui suit représente les termes regroupés dans le glossaire.

Terme	Description	Type
Véhicule	Un moyen de transport qui permet de déplacer des personnes ou des charges d'un point à un autre.	Classe
Voiture	Est une véhicule sa capacité est généralement de deux à cinq personnes, mais peut varier de une à neuf places.	Classe
Preference_acheteur	Représente l'ensemble des préférences des acheteurs	Classe
Preference_vendeur	Représente l'ensemble des préférences des vendeurs	Classe
Detail_voiture	Est l'ensemble de caractéristiques de voitures	Classe
A1...A5	Est une parmi les préférences des acheteurs	Classe
V1...V5	Est une parmi les préférences des vendeurs	Classe
Assurance	Un service qui fournit une prestation lors de la survenance d'un risque	Classe
.....

Table 3.1 Extrait du glossaire des termes

4.2 Construction de la taxonomie d'ontologie

En raffinant le glossaire des termes présentés précédemment nous allons à présent construire une taxonomie.

La hiérarchie de classification de concepts démontre l'organisation des concepts de l'ontologie en un ordre hiérarchique qui exprime les relations sous-classe, super-classe. En utilisant la relation "sous classe de" entre les classes pour définir leurs classifications, la classe C1 est une sous classe de la classe C2 si et seulement si toute instance de la classe C1 est une instance de la classe C2, par exemple la classe voiture est une sous classe de véhicule parce que tous les voitures sont des véhicules.

Pour la construction d'une taxonomie nous suivons un procédé de développement de haut en bas en commençant par une définition des concepts les plus généraux du domaine et en poursuivant par la spécialisation des concepts.

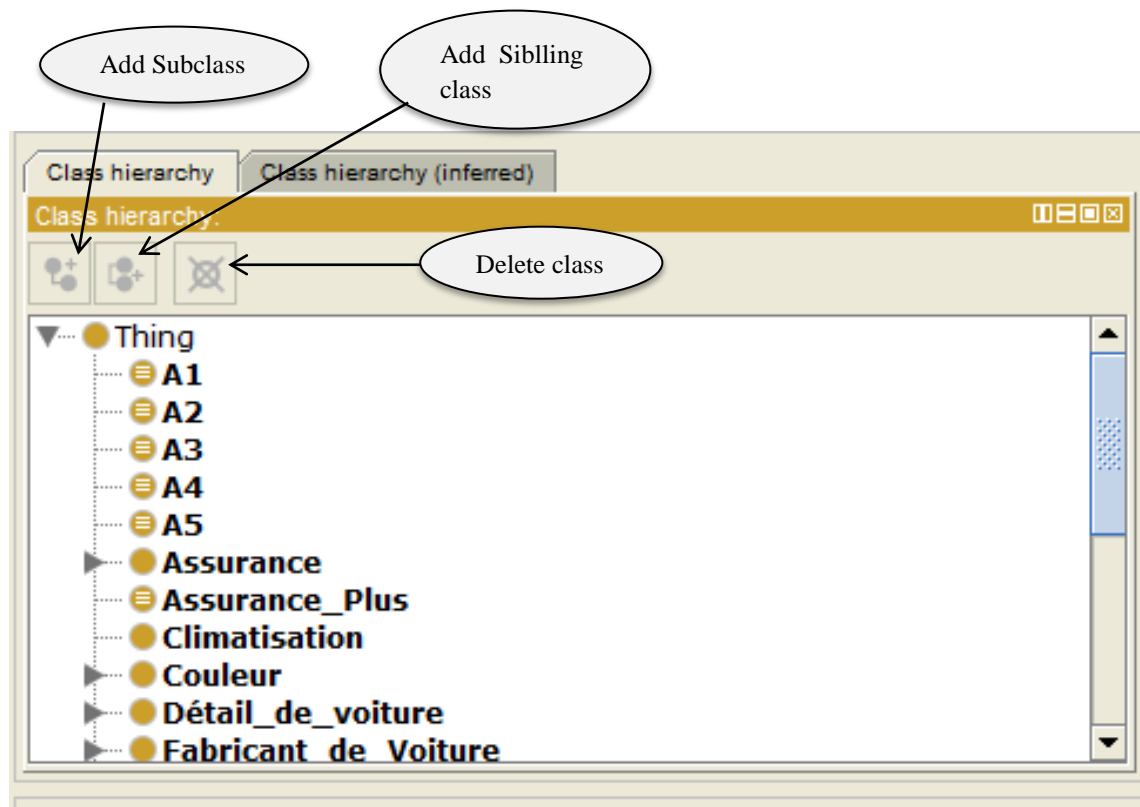


Figure 3.7 Construction de la taxonomie des concepts

4.3 Construction du diagramme de relation binaire:

Dans cette étape, nous représentons les relations binaires par un diagramme. Dans ce diagramme, les classes sont représentées par les rectangles et les relations par des arcs orientés (du domaine vers le co-domaine) et étiquetés par le nom de la relation. Le diagramme de la figure 3.8 représente les relations binaires que nous avons associées à notre ontologie.

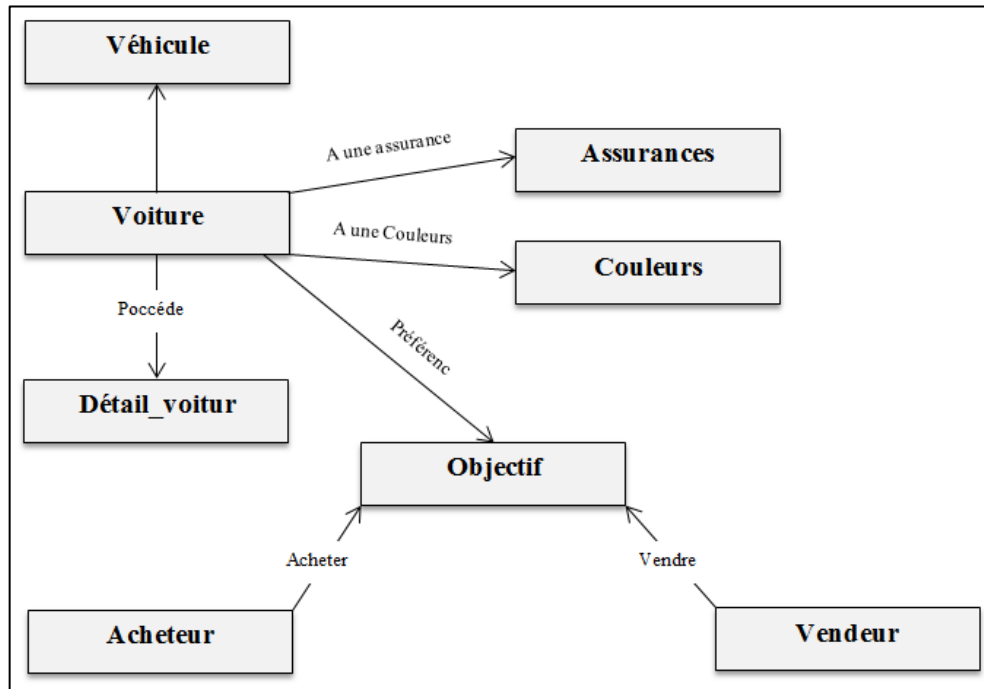


Figure 3.8 Diagramme des relation binaires

4.4 Construction de dictionnaire des concepts

Dans cette étape nous allons décrire formellement des concepts qui ont été présentés dans la hiérarchie des classes, ce processus correspond à la création des descriptions pour chaque concept: les attributs et les relations de ce concept.

Nom du concept	Attributs des classes	Relation
Voiture	a_une_année a_un_prix a_une_securite	A_une_couleur A_une_assurance Dispose_systeme_alarme
Detail_voiture	a_capacite_de_chargement a_performance_et_manipulation a_un_espace_de_passager	Voiture_procède_détails
Assurance	Type_assurance Année_assurance	Voiture_A_une_assurance
.....

Table 3.2 Extrait du dictionnaire des concepts

4.5 Description des relations binaires

Pour chaque relation dont la source est dans l'arbre de classification de concepts, nous définissons: son nom, le nom du concept source, le nom du concept cible, et la cardinalité.

Le concept (Thing) représente la classe universelle.

Nom de la relation	Concept source	Concept cible	Cardinalité
a_une_couleur	Thing	Couleur	(1, N)
a_une_assurance	Voiture	Assurance	(0, N)
Dispose_systeme_alarme	Voiture	Systeme_alarme	(0, N)
Voitur_pocède_détail	Voiture	Detail_voiture	(1, 1)
a_un_fabricant	Voiture	Fabricant	(1, N)

Table 3. 3 Tableau des relations binaires

4.6 Description des attributs des classes

Pour chaque attribut apparaissant dans le dictionnaire de concepts nous spécifions: son nom, type de la valeur.

Nom d'attribut	Type de la valeur
a_une_année	Int
a_une_pris	Int
Type_assurance	String
.....

Table 3.4 Tableau d'attributs de classes

5. La description de la partie floue

Dans cette partie nous allons décrire la partie floue dans notre ontologie, nous expliquons la définition des types de données floues, et les annotations ajoutées pour définir les concepts flous.

5.1 Description des types de données floues

Dans notre exemple on a utilisé 7 types de données floues qui représentent les classes de prix des voitures et les types de garanties offre par le vendeur avec la plus et la petite valeur de chaque type de données floues.

Le tableau qui suit représente les types de données floues utilisé dans l'ontologie.

Types de données	Description de type de données	Annotations utilisée
ls22000_24000	le prix de voiture est entre [22000,24000]	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="leftshoulder" a="22000.0" b="24000.0" /> </fuzzyOwl2>
rs15000_175000	le prix de voiture est entre [22000,24000]	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="rightshoulder" a="15000.0" b="175000.0" /> </fuzzyOwl2>
sellerM_GR	La garantie par miles	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="leftshoulder" a="60.0" b="72.0" /> </fuzzyOwl2>
rs22500_22750	le prix de voiture est entre [22500,22750]	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="rightshoulder" a="22500.0" b="22750.0" /> </fuzzyOwl2>
leq26000	le prix de voiture inférieur à 26000	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="leftshoulder" a="26000.0" b="26000.0" /> </fuzzyOwl2>
sellerKm_GR	La garantie par kilomètre	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="leftshoulder" a="100000.0" b="125000.0" /> </fuzzyOwl2>
geq22000	Le prix de voiture supérieur à 22000	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="rightshoulder" a="22000.0" b="22000.0" /> </fuzzyOwl2>
ls22300_22750	le prix de voiture est entre [22300,22750]	<fuzzyOwl2 fuzzyType="datatype"> <Datatype type="leftshoulder" a="22300.0" b="22750.0" /> </fuzzyOwl2>

Table 3.5 les types de données floues

5.2 Description des concepts flous

L'objectif de notre ontologie est de trouver le meilleur accord entre l'acheteur et le vendeur, pour cette raison on a ajouté des préférences de l'acheteur et des préférences de vendeur.

L'acheteur dispos de 5 préférences qui sont A1-A5 donc les concepts sont codés comme suit:

Acheter \equiv Beta \cap Préférences_des_Acheteurs

Beta \equiv Voiture_Tourisme $\cap \exists$ a_un_prix. leq26000

A1 $\equiv \neg (\exists$ a_une_Systeme_alarme .Systeme_d_alarme) $\cup \exists$ a_un_prix. ls22300_22750)

A2 $\equiv (\exists$ a_une_Assurance some Assurance_Incendie) $\cap \exists$ a_une_Assurance.

(Assurance_contre_vol \cup Assurance_Conducteur)

A3 $\equiv (\exists$ Dispose_Climatisation. Climatisation) $\cap \exists$ a_une_couleur. (Blanc \cup Gris \cup Noir)

A4 $\equiv \exists$ a_un_prix. ls22000_24000

A5 $\equiv \exists$ a_KM_Garantie . rs15000_175000

Certaines préférences peuvent être plus importantes que les autres, donc nous ajoutons la propriété d'annotation suivante dans le concept de préférences des acheteurs.

```
<fuzzyOwl2 fuzzyType = " concept">
  <Concept type = " weightedSum ">
    <Concept type = " weighted" value ="0.1" base ="A1" />
    <Concept type = " weighted" value ="0.2" base ="A2" />
    <Concept type = " weighted" value ="0.1" base ="A3" />
    <Concept type = " weighted" value ="0.2" base ="A4" />
    <Concept type = " weighted" value ="0.4" base ="A5" />
  </Concept >
</fuzzyOwl2 >
```

Comme l'acheteur, le vendeur dispose 5 de préférences qui sont V1-V5 donc les concepts sont codés comme suit:

Vendre \equiv Sigma \cap Préférences_Vendeur

Sigma \equiv Voiture_Berline $\cap \exists$ a_un_prix some .geq22000

V1 $\equiv \neg (\exists$ a_navigateur.Paquet_Navigator) $\cup \exists$ a_un_prix.rs22500_22750

V2 $\equiv \exists$ a_une_Assurance . Assurance

V3 $\equiv \exists$ a_KM_Garantie . sellerKm_GR

V4 $\equiv \exists$ a_M_Garantie . sellerM_GR

V5 $\equiv \neg (\exists$ a_une_couleur.Noir) $\cup \exists$ Dispose_Climatisation.Climatisation

L'annotation suivante représente l'ensemble des poids de préférences des vendeurs.

```
<fuzzyOwl2 fuzzyType = "concept">
  <Concept type = "weightedSum ">
    <Concept type = "weighted" value = "0.3" base = "V1" />
    <Concept type = "weighted" value = "0.1" base = "V2" />
    <Concept type = "weighted" value = "0.3" base = "V3" />
    <Concept type = "weighted" value = "0.1" base = "V4" />
    <Concept type = "weighted" value = "0.2" base = "V5" />
  </Concept >
</fuzzyOwl2 >
```

6. Formalisation

Cette étape consiste en la représentation de connaissances extraites dans un langage formel tel que les graphes conceptuels. Cette partie stipule la création d'un model formel avant de générer le model computable mais grâce aux éditeurs d'ontologie cette phase se fait implicitement et c'est au logiciel de créer et vérifier le model formel.

7. L'implémentation

Pour la construction de la partie classique de l'ontologie, on a utilisé l'éditeur owl Protégé, ce qui permet de raisonner sur notre ontologie à l'aide des raisonneurs de l'ontologie standard. Ensuite, nous pouvons ajouter la partie floue de l'ontologie en utilisant les propriétés d'annotation.

La représentation des informations floues en utilisant les annotations OWL 2 peut également être faite avec protégé. Cependant, la création des annotations est une tâche fastidieuse et source d'erreurs, pour cette raison nous avons utilisé un plug-in Protégé qui fait la syntaxe des annotations transparente pour les utilisateurs.

Le Fuzzy OWL 2 plug-in est disponible sur le web. Une fois le plugin installé, un nouvel onglet OWL flou apparaitre qui permet d'utiliser le plug-in.

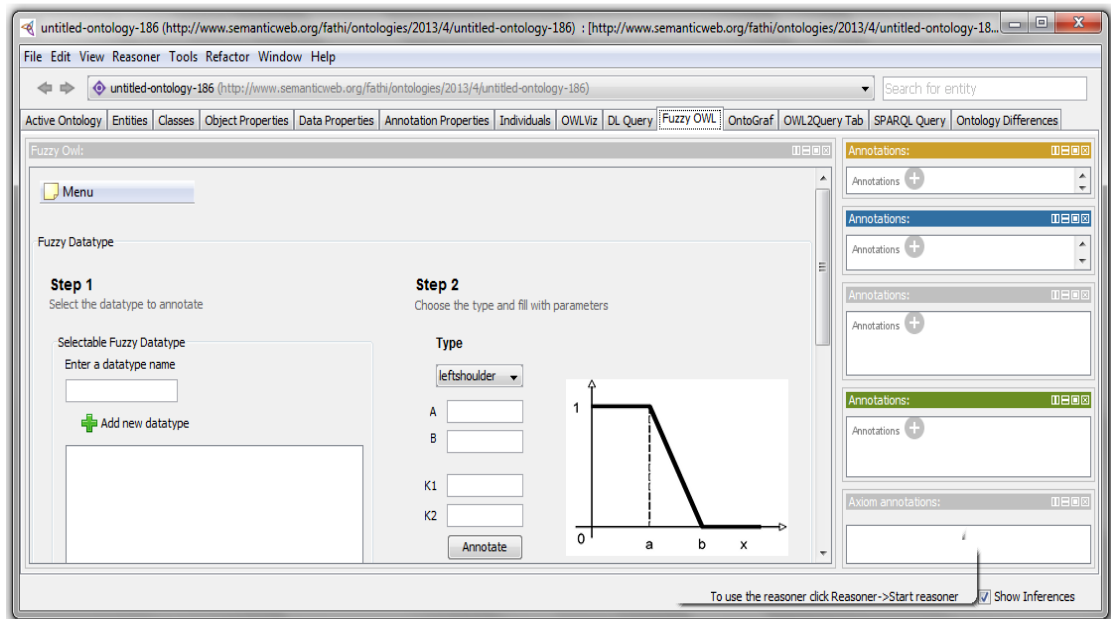


Figure 3.9 L'onglet fuzzy owl de plugin

Le plug-in dispose d'un menu avec les options disponibles. L'utilisateur a un choix de définir les éléments flous dans l'ontologie (types de données floue, des concepts, des concepts pondérés, les modificateurs flous,...etc), il nous permet de spécifier la logique floue utilisée dans l'ontologie.

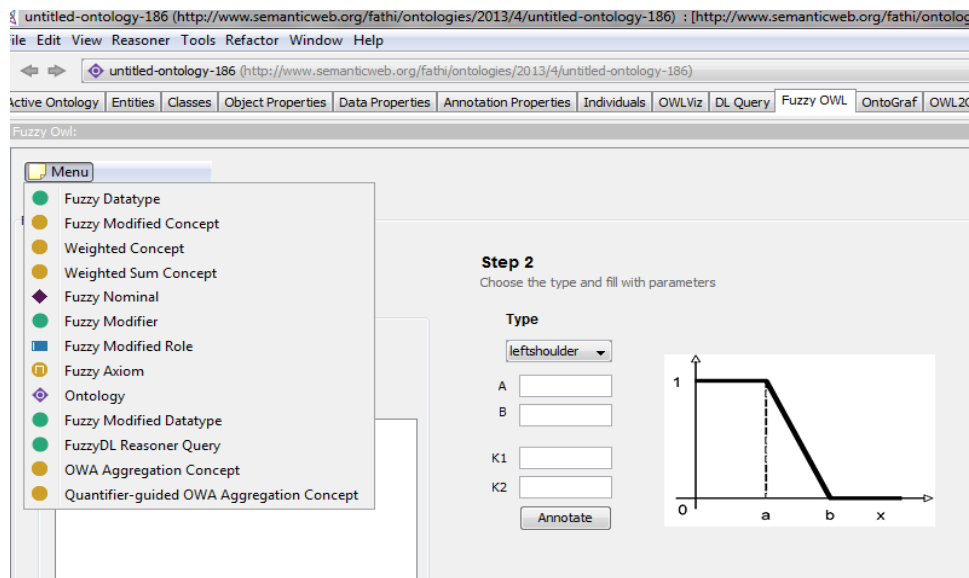


Figure 3.10 Menu des options de plugin

Figure 3.11 illustre comment le plug-in fonctionne en montrant comment créer un nouveau type de données floues. L'utilisateur indique le nom du type de données, et le type de la fonction d'appartenance. Ensuite, le plug-in demande les paramètres nécessaires selon le type. Une image est affichée pour aider l'utilisateur à rappeler la signification des paramètres. Puis,

après quelques vérifications d'erreur de base, le nouveau type de données est créé et peut être utilisé dans l'ontologie.

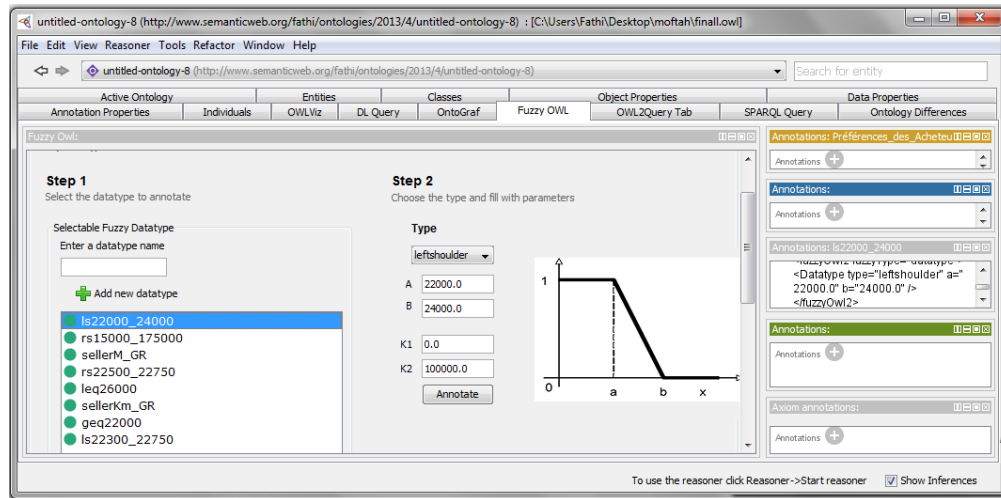


Figure 3.11 Création d'un type de donnée avec fuzzy owl

En outre, le plug-in intégré un raisonneur fuzzyDL qui permet de soumettre des requêtes. Pour l'instant, ces requêtes doivent être exprimées en utilisant la syntaxe particulière supports par fuzzyDL. Cela permet d'utiliser le raisonneur sans quitter Protégé et de traduire les annotations ontologie OWL 2 en syntaxe fuzzyDL.

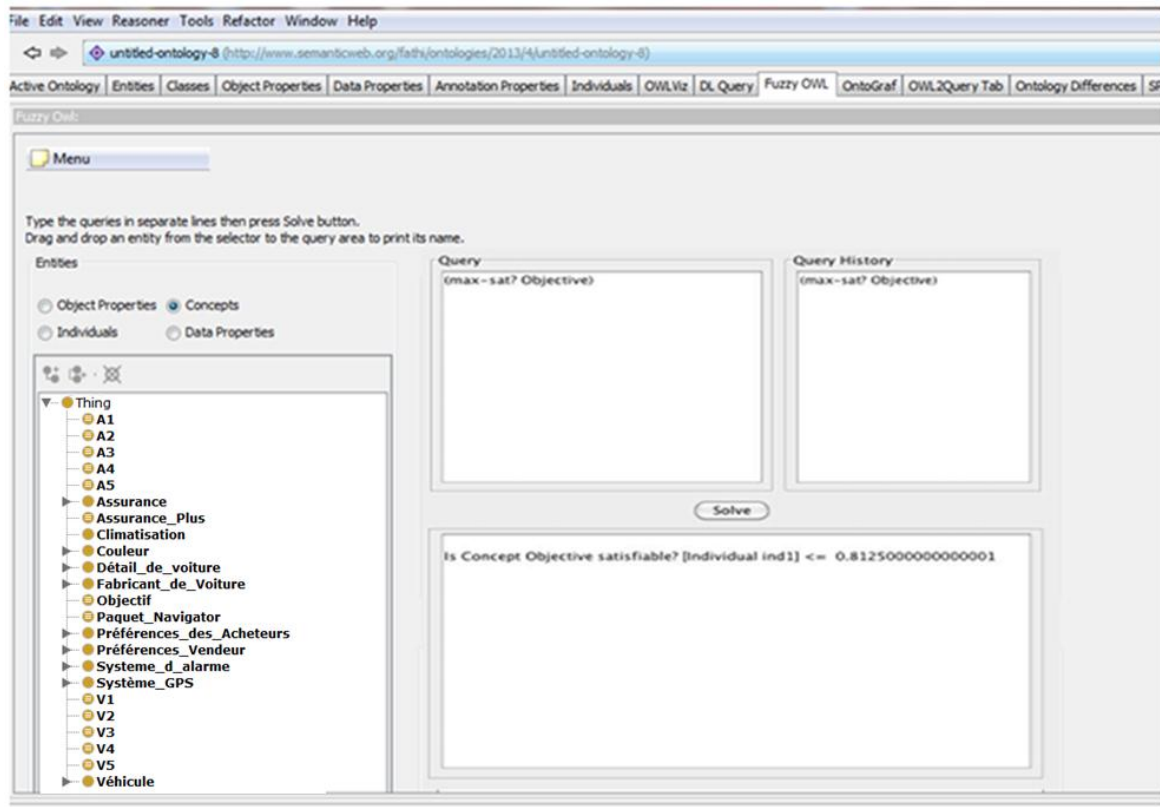


Figure 3.12 Test d'ontologie avec fuzzyDL Reasoner Query

Nous donnons dans ce qui suit des figures qui représentent un extrait de l'ontologie

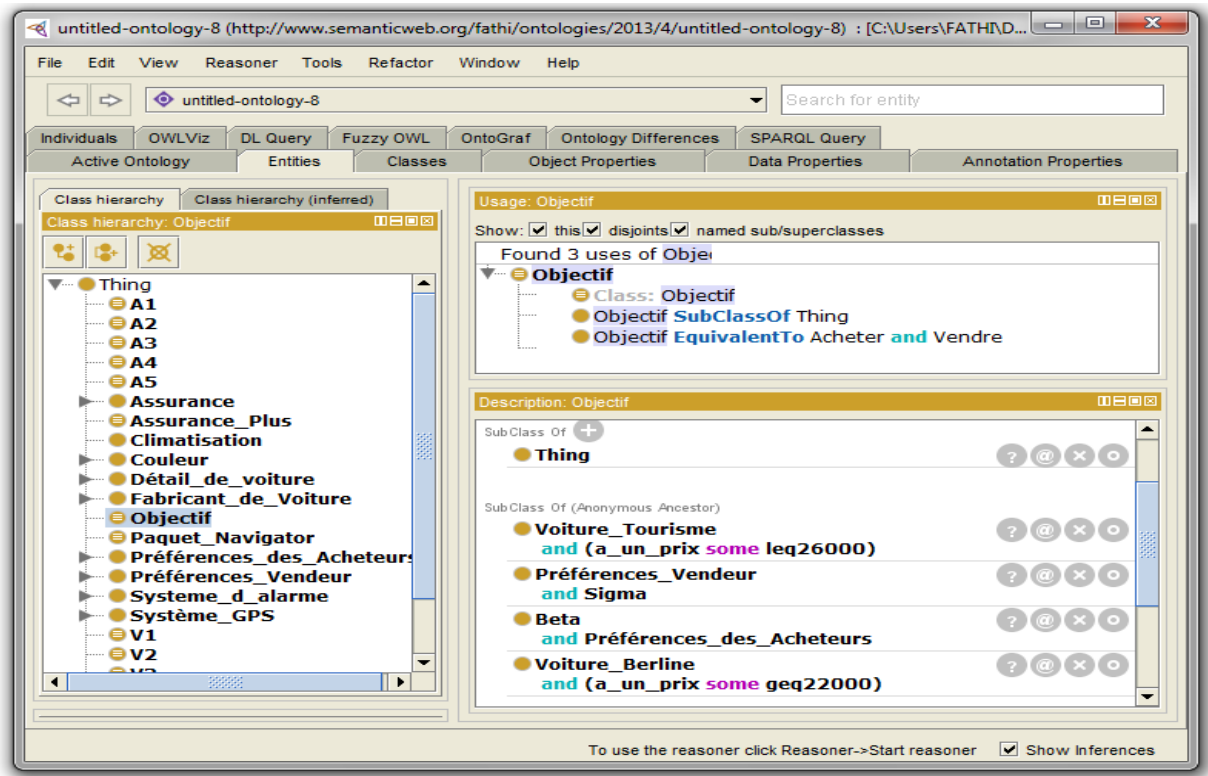


Figure 3.13 Extrait de l'ontologie ventes des voitures

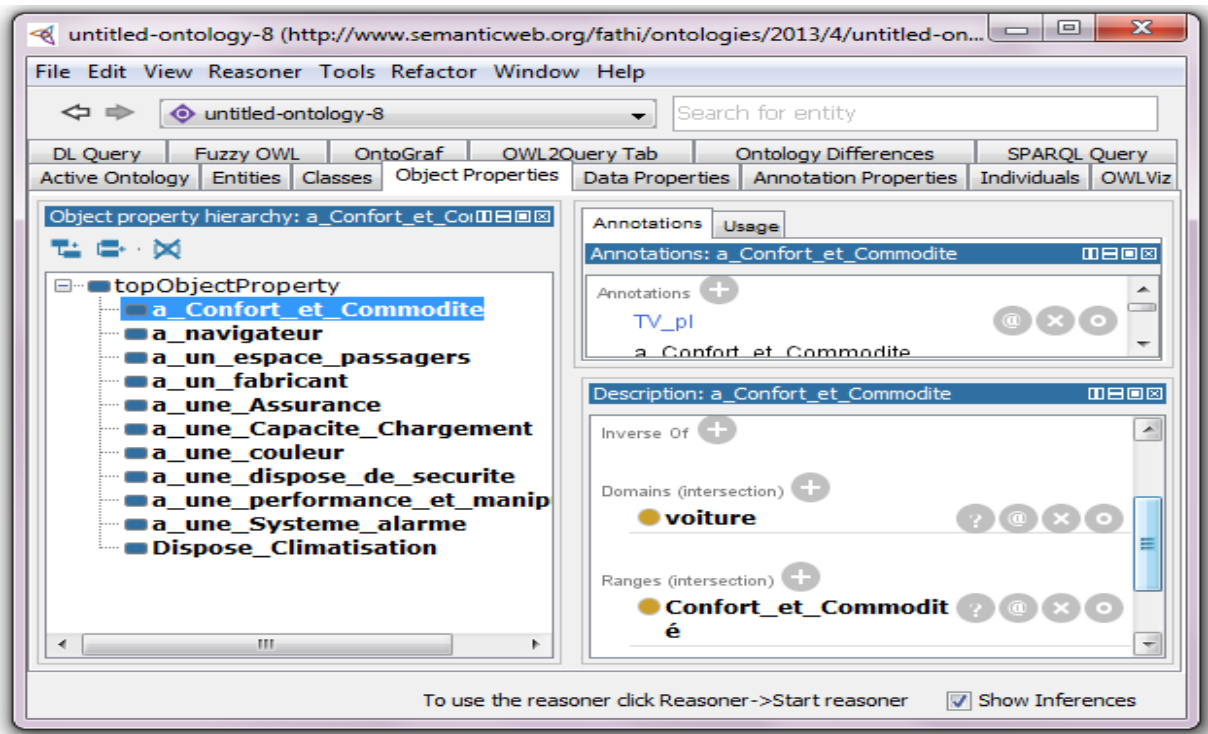


Figure 3.14 Les relations de l'ontologie

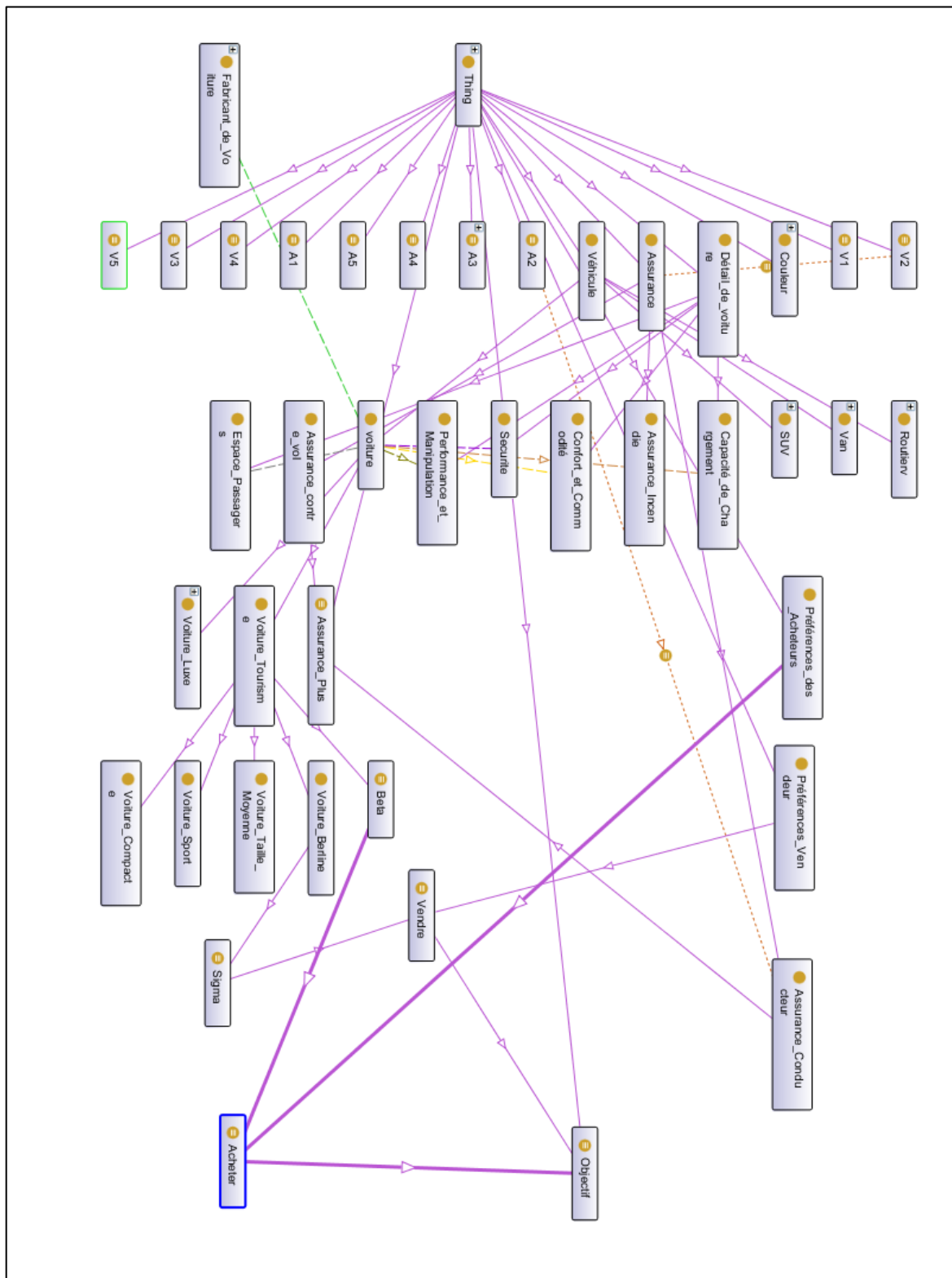


Figure 3.15 la hiérarchie des classes avec OntoGraf

8. conclusion

Dans ce chapitre, nous avons présenté les détails de construction d'ontologie floue. Nous avons choisi, le langage OWL, pour codifier l'ontologie formelle, en utilisant l'éditeur Protégé-owl afin de guider l'implémentation et de produire un document OWL. Par ailleurs nous avons ajouté un plugin à protégé pour le raisonnement de la partie floue.

Contenu

1 Introduction	44
2 Le langage OWL	44
2.1 Objectifs et motivation	44
2.2 Structure d'une ontologie OWL.....	45
2.2.1 Espaces de nommage	45
2.2.2 En-têtes d'une ontologie	46
2.3 Eléments du langage.....	47
2.3.1 Les classes	47
2.3.1.1 Déclaration de classe	47
2.3.1.2 Héritage	48
2.3.2 Les instances de classe	49
2.3.2.1 Définition d'un individu	49
2.3.2.2 Affecter une propriété à un individu	50
2.3.3 Les propriétés	50
2.3.3.1 Définition d'une propriété.....	50
2.3.3.2 Caractéristiques des propriétés	52
2.3.4 Les contraintes de cardinalité	53
2.3.5 Les classes disjointes.....	54
3. Editeur d'ontologies PROTÉGÉ	55
3.1 Création d'un projet PROTÉGÉ	55
3.2 Définition des classes et des propriétés.....	57
3.3 Gestion des instances de classe et leurs propriétés.....	57
3.4 Possibilité d'effectuer des requêtes	58
4. Conception détaillée de l'ontologie.....	59
4.1 Description de l'ontologie ventes des voitures :	59
4.1 Construction d'un glossaire des termes	60
4.2 Construction de la taxonomie d'ontologie	61
4.4 Construction de dictionnaire des concepts	62
4.5 Description des relations binaires	63
4.6 Description des attributs des classes	64
5. La description de la partie floue	64
5.1 Description des types de données floues.....	64
5.2 Description des concepts flous.....	65

6. Formalisation.....	67
7. L'implémentation.....	67
8. conclusion	72

Les Figure

Figure 3.1 les appels à l'ensemble de plugins java	55
Figure 3.2 Création d'un nouveau projet	56
Figure 3.3 Sélection type de projet.....	56
Figure 3.4 Capture de l'écran principal de Protégé, onglet "classes"	57
Figure 3.5 Editeur d'instances intégré à protégé	58
Figure 3.6 Gestionnaire de requetes de protégé	58
Figure 3.7 Construction de la taxonomie des concepts	61
Figure 3.8 Diagramme des relation binaires	62
Figure 3.9 L'onglet fuzzy owl de plugin	68
Figure 3.10 Menu des options de plugin	68
Figure 3.11 Création d'un type de donnée avec fuzzy owl.....	69
Figure 3.12 Test d'ontologie avec fuzzyDL Reasoner Query.....	69
Figure 3.13 Extrait de l'ontologie de ventes des voitures.....	70
Figure 3.14 Les relations de l'ontologie	70
Figure 3.15 la hiérarchie des classes avec OntoGraf	71

Les Tableaux

Table 3.1 Extrait du glossaire des termes.....	60
Table 3.2 Extrait du dictionnaire des concepts.....	63
Table 3. 3 Tableau des relations binaires	63
Table 3.4 Tableau d'attributs de classes	64
Table 3.5 les types de données floues	65

Conception et Implémentation d'une Ontologie Floue sur les Ventes des Voitures