Introduction au langage OWL (Ontology Web Language)

OWL1

Bernard ESPINASSE

Aix-Marseille Université
LIS UMR CNRS 7020

Novembre 2017

- **Des Logiques de Description à OWL**
- **OWL-Lite**
- **OWL-DL**
- **Traduction d'une ontologie en OWL-DL et RDFS/XML**

Références

- **Livres, articles et rapports :**
  - W3C, « OWL Web Ontology Language Semantics and Abstract Syntax »
  - G. Antoniou, Van Harmelen F., A Semantic Web Primer, The MIT Press Cambridge, Massachusetts London, England, 1999.
  - X. Lacot, « Introduction à OWL, un langage XML d'ontologies Web », 2005.
  - F. Lapique, « Le langage d'ontologie Web OWL », EPFL, 2006.
  - …
- **Web W3C :**
  - Page du W3C : http://www.w3.org/2004/OWL/
  - Référence : http://www.w3.org/TR/owl-ref/
  - Guide : http://www.w3.org/TR/owl-guide/
- **Cours/tutoriaux :**
  - Cours de M. Gagnon, Ecole Polytechnique de Montréal, 2007.
  - Cours de S. Staab, ISWeb – Lecture « Semantic Web », Univ. de Koblenz-Landau.
  - Cours de D. Genest, « Web Sémantique », Univ. d'Angers, 2007.
  - Cours de O. Corby, « OWL : W3C Ontology Web Language », INRIA-ESSI, 2005.
  - …

Plan

1. Introduction à OWL

- **Bref historique de OWL (OWL1, OWL2)**
- **Limites de RDF/RDF-S**
- **XML, RDF et OWL les 3 couches du Web sémantique**
- **Des Logiques de Description à OWL : OWL1**
- **Les 3 sous-langages (profils) de OWL1: OWL1-Lite, OWL1-DL et OWL1-Full**

Bref historique de OWL

- Il existe **plusieurs langages informatiques** spécialisés dans la **création** et la **manipulation d'ontologies :**
  - **OKBC** (Open Knowledge Base Connectivity-1997) : API permettant d'accéder à des bases de connaissance
  - **KIF** (Knowledge Interchange Format-1998) : langage destiné à faciliter des échanges de savoirs entre systèmes informatiques hétérogènes.
  - **LOOM** : langage de représentation des connaissances dont le but avoué est de « permettre la construction d'applications intelligentes »
  - **DAML-ONT** (2000) : fondé sur XML, résulte d'un effort du DARPA (Defense Advanced Research Projects Agency) pour l'expression de classes plus complexes que le permet RDF-S
- **Nov 2001 : W3C** (World Wide Web Consortium) lance le GT « WebOnt », pour la création d'un langage standard de manipulation d'ontologies Web
- **Juil 2002** : **1°Working Draft** « OWL Web Ontology Language 1.0 Abstract Syntax »
- **Fév 2004** : **OWL** devient **recommandation** du **W3C : OWL 1**
- **2008/2011 : Définition de OWL 2**

Seul traité ici OWL1

Limitations de RDF/RDF-S

RDF/RDF-S présente des limitations pour représenter des ontologies :

- **rdfs:range** définit le domaine de valeurs d'une propriété quelle que soit la classe concernée :
  Ex : il ne permet pas d'exprimer que les vaches ne mangent que de l'herbe alors que d'autres sortes d'animaux mangent également de la viande.
- **RDF-S** ne permet pas d'exprimer que **2 classes sont disjointes** :
  Ex : les classes des hommes et des femmes sont disjointes.
- **RDF-S** ne permet pas de **créer des classes par combinaison ensembliste d'autres classes** (intersection, union, complément) :
  Ex : on veut construire la classe Personne comme l'union disjointe des classes des hommes et des femmes.
- **RDF-S** ne permet pas de **définir de restriction sur le nombre d'occurrences de valeurs** que peut prendre une **propriété** :
  Ex : on ne peut pas dire qu'une personne a exactement 2 parents.
- **RDF-S** ne permet pas de **caractériser des propriétés** notamment :
  - ***transitivité*** : Ex : estPlusGrandQue
  - ***unicité*** : Ex : estLePèreDe
  - ***propriété inverse*** : Ex : « mange » = propriété inverse de « estMangéPar »

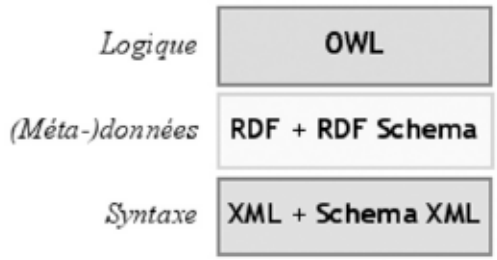
Introduction à OWL

- **OWL (Web Ontology Language)** est un **langage de description d'ontologies** conçu pour la **publication** et le **partage d'ontologies** sur le **Web sémantique**
- **OWL** s'inspire fortement de **DAML** (projet US) **+ OIL** (projet Européen)
- **OWL** (comme RDF) est **un langage XML** (universalité syntaxique)
- **OWL permet aux hommes et aux machines :**
  - **une riche représentation des connaissances :** propriétés, classes avec identité, équivalence, contraire, cardinalité, symétrie, transitivité,…
  - **de raisonner** sur ces connaissances en s'appuyant sur une **axiomatique formelle (Logique de Description)**
- **Références OWL :**
  - Page du W3C : http://www.w3.org/2004/OWL/
  - Référence : http://www.w3.org/TR/owl-ref/
  - Guide : http://www.w3.org/TR/owl-guide/

XML, RDF et OWL : les 3 couches du WS

XML, **RDF** et **OWL** constituent les 3 couches de base du Web Sémantique:

- **XML** : support de sérialisation sur lequel s'appuient **RDF/RDF-S** et **OWL**
- **RDF/RDF-S** et **OWL** : permettent de définir des structures de données et les relations logiques qui les lient :



OWL : particularités

OWL est un **langage d'ontologies** pour le **Web Sémantique** devant :

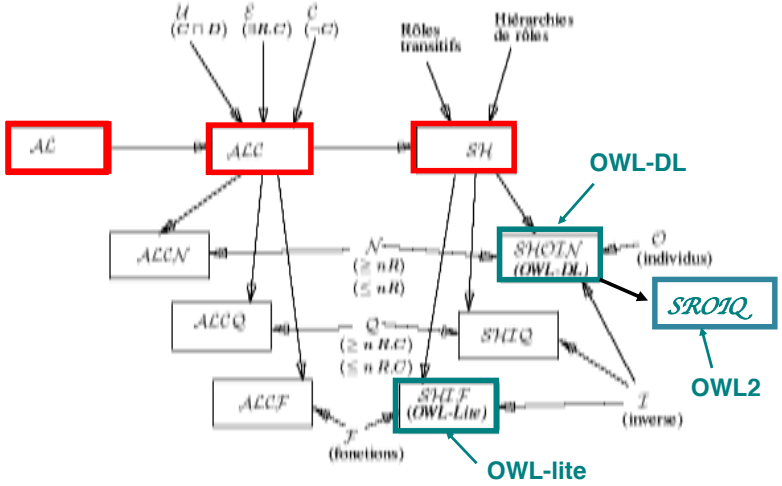
- **faciliter le raisonnement** d'**agents logiciels** qui accèdent à des **ressources** sur le **Web**
- permettre **l'utilisation de plusieurs ontologies** pouvant être **reliées**

=> caractéristiques majeures de OWL :

- **monde ouvert** : une ontologie décrit certaines ressources (ex : B est une sous-classe de A dans une ontologie O), **mais ces ressources peuvent être étendues** (ex : dans une ontologie O' qui étend O, B est aussi une sous-classe de C).
- **monotonie** :
  - **on peut rajouter** de **nouvelles informations**, mais **pas en enlever,**
  - ce qui est rajouté peut **permettre de nouvelles inférences**, mais **pas annuler les inférences de l'ontologie initiale**.

OWL et les logiques de description - LD

« **Photo** » **de famille des LD $\mathcal{AL}$, $\mathcal{ALC}$ et $\mathcal{SH}$** (d'après Gagnon) :



OWL et moteurs d'inférences dans les LD

Comparaison de moteurs d'inférence pour les LD Expressives (d'après Fournier-Viger) : **FaCT** [Horrocks, 1998], **Racer** [Haarslev & Möller, 2001], **Pellet** [Sirin & Parsia, 2004], **FaCT++** [Tsarkov & Horrocks, 2004], **F-OWL** [Zou & al., 2004], … :

Moteur	*Racer*	*FaCT*	*Pellet*	*FaCT++*
LD	$\mathcal{SHIQ}(\mathcal{D})-$	$\mathcal{SHIQ}$, $\mathcal{SHF}$	$\mathcal{SHIN}(\mathcal{D})$, $\mathcal{SHON}(\mathcal{D})$	$\mathcal{SHIF}(\mathcal{D})$
Implantation	*C++*	*Common Lisp*	*Java*	*C++*
Inférence	*TBox*/*ABox*	*TBox*	*TBox*/*ABox*	*TBox*
API *Java*	oui	oui	natif	oui
Mise-à-échelle	bonne	bonne	bonne	bonne
OWL	*OWL-DL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-LITE*
Décidabilité	oui (*OWL-LITE*)	oui	oui (*OWL-LITE*)	oui
DIG	oui	oui	non	?

Moteur	*Surnia*	*Hoolet*	*F-OWL*	
LD	logique prédicats	logique prédicats	$\mathcal{SHIQ}(\mathcal{D})$ et *RDF*	
Implantation	*Python*	*Java*	*Java*	
Inférence	*TBox*/*ABox*	*TBox*/*ABox*	*TBox*/*ABox*	
API *Java*	?	oui	oui	
Mise-à-échelle	médiocre	médiocre	médiocre	
OWL	*OWL-FULL*$\sim^{\dagger}$	*OWL-DL*$\sim^{\dagger}$	*OWL-FULL*$\sim^{\dagger}$	
Décidabilité	non	non	non	
DIG	non	non	non	

(∼† = approximativement)

Syntaxe OWL : conventions adoptées

- **Notation BNF** : similaire à celles du W3C (« OWL Web Ontology Language Semantics and Abstract Syntax »)
- **Conventions adoptées** :
  - **symboles terminaux** : représentés en ***italique***
  - **symboles non-terminaux** : représentés en **caractères gras**
  - **alternatives** : séparées par des barres (**|**) ou sont données par des règles différentes
  - **éléments facultatifs** (pouvant apparaître au plus une fois) englobés par des crochets ([. . .])
  - **éléments répétitifs** (pouvant apparaître un nombre illimité de fois, incluant zéro) : englobés par des accolades ({. . .})
- On fera plusieurs simplifications dans la définition du langage, pour des raisons de clarté (ainsi OWL permet d'ajouter des annotations dans les ontologies)
- On se rapportera aux documents du W3C pour une description exhaustive du langage.

Une ontologie en OWL (1)

Tout document OWL est une ontologie :

- qui peut avoir un **identificateur unique** représenté par une **URI**
- qui contient :
  - des **faits** qui sont des descriptions d'individus
  - des **axiomes** qui fournissent les descriptions de concepts.

- Un document OWL a la forme suivante :

  ontologie ::= *Ontology(* [ontologieID] { directive } *)*

  directive ::= axiome I fait

- L'ontologie OWL la plus simple que l'on peut écrire est :

  ontology()

- Notons, que OWL comprend les 2 classes pré-définies :

  owl:Thing : correspondant à $\top$

  owl:Nothing : correspondant à $\bot$.

Une ontologie en OWL (2)

Ontologie en LD:

$$\text{Personne} \sqcap \forall \text{aEnfant.}(\text{Docteur} \sqcup \exists \text{aEnfant.Docteur})$$

Ontologie en OWL (en syntaxe RDF/XML) :

```
<owl:Class>
    <owl:intersectionOf rdf:parseType="collection">
        <owl:Class rdf:about="#Personne"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#aEnfant"/>
            <owl:toClass>
                <owl:unionOf rdf:parseType="collection">
                    <owl:Class rdf:about="#Docteur"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#aEnfant"/>
                        <owl:hasClass rdf:resource="#Docteur"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:toClass>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```


Différentes syntaxes de OWL

Il y a différentes syntaxes pour stocker, partager, éditer des ontologies OWL :

- la **syntaxe d'échange RDF / XML officiellement recommandée**, et que tout outil compatible OWL doit prendre en charge
- des **syntaxes standard** que tous les outils OWL et API prennent en charge
- des **syntaxes spécialement conçues** pour des applications et buts particuliers.

Quelle que soit la syntaxe utilisée, le langage OWL n'est pas défini à l'aide d'une **syntaxe concrète particulière**, mais est défini par une **spécification structurelle abstraite** de haut niveau, qui est **ensuite traduite** dans **diverses syntaxes concrètes**.

- **Principales syntaxes concrètes :**
  - *Fonctionnelle*
  - *RDF/XML*
  - *Turtle*
  - *OWL/XML*
  - *Manchester*

Syntaxe OWL *fonctionnelle*

- **Première étape vers des syntaxes concrètes**
- C'est une **syntaxe simple de base** qui sert de pont entre la spécification abstraite/structurelle et diverses syntaxes concrètes.
- Pas destinée à être utilisée comme une syntaxe d'échange, mais est pour traduire la spécification structurelle dans d'autres syntaxes concrètes.
- ***Exemple:***

  un axiome de classes equivalents spécifiant que "Teenager" équivaut à une personne dont l'âge est compris entre 12 et 20 ans

```
EquivalentClasses(:Teenager
ObjectIntersectionOf(:Person
          DataSomeValuesFrom(:hasAge
          DatatypeRestriction(xsd:integer
              xsd:maxExclusive "20"^^xsd:integer
              xsd:minExclusive "12"^^xsd:integer))))
```

Syntaxe OWL *RDF/XML*

- Syntaxe concrète RDF **très verbeuse** et **difficile à lire** pour un **humain**
- Utilisée par la plupart des **outils OWL** comme **syntaxe par défaut** pour enregistrer les ontologies

- ***Exemple:***

```
<owl:Class rdf:about="http://www.semanticweb.org/ontologies/ontogenesi
    <owl:equivalentClass>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <rdf:Description rdf:about="http://www.semanticweb.org
                <owl:Restriction>
                    <owl:onProperty rdf:resource="http://www.semanticw
                    <owl:someValuesFrom>
                        <rdfs:Datatype>
                            <owl:onDatatype rdf:resource="http://www.w
                            <owl:withRestrictions rdf:parseType="Colle
                                <rdf:Description>
                                    <xsd:maxExclusive rdf:datatype="ht
                                </rdf:Description>
                                <rdf:Description>
                                    <xsd:minExclusive rdf:datatype="ht
                                </rdf:Description>
                            </owl:withRestrictions>
                        </rdfs:Datatype>
                    </owl:someValuesFrom>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </owl:equivalentClass>
</owl:Class>
```


Syntaxe OWL *Turtle*

- Autre syntaxe concrète RDF **moins verbeuse** et **un plus lisible** que la syntaxe RDF/XML
- Reste encore **difficile à lire** pour un **humain**

- ***Exemple:***

```
:Teenager rdf:type owl:Class ;
      owl:equivalentClass [ rdf:type owl:Class ;
          owl:intersectionOf ( : Person
              [ rdf:type owl:Restriction ;
                owl:onProperty :hasAge ;
                owl:someValuesFrom [ rdf:type rdfs:Datatype ;
                    owl:onDatatype xsd:integer ;
                    owl:withRestrictions (
] ]

) ].

[ xsd:maxExclusive "20"^^xsd:integer ]
[ xsd:minExclusive "12"^^xsd:integer]
)
```


Syntaxe OWL *OWL/XML*

- Syntaxe **plus régulière et plus simple** mais reste encore **verbeuse**
- **Format de représentation concret** pour les ontologies OWL
- **Dérivé** directement de la **syntaxe fonctionnelle**

- ***Exemple:***

```
<EquivalentClasses>
       <Class IRI="#Teenager"/>
       <ObjectIntersectionOf>
           <Class IRI="#Person"/>
           <DataSomeValuesFrom>
              <DataProperty IRI="#hasAge"/>
              <DatatypeRestriction>
                  <Datatype abbreviatedIRI="xsd:integer"/>
                  <FacetRestriction facet="http://www.w3.org/2001/XMLSch
                      <Literal datatypeIRI="http://www.w3.org/2001/XMLSc
                  </FacetRestriction>
                  <FacetRestriction facet="http://www.w3.org/2001/XMLSch
                      <Literal datatypeIRI="http://www.w3.org/2001/XMLSc
                  </FacetRestriction>
              </DatatypeRestriction>
          </DataSomeValuesFrom>
       </ObjectIntersectionOf>
  </EquivalentClasses>
```


Syntaxe OWL *Manchester*

- Fournit une **représentation compacte** pour les ontologies OWL, **facile à lire et à écrire**
- Principale motivation : **pouvoir être utilisé pour la modification des expressions de classe** dans des **outils** tels que **Protege 3 et 4, Top Braid, …**
- Elle a été étendue pour **représenter des ontologies complètes**
- Elle est maintenant **spécifiée dans une note W3C**

- ***Exemple:***

```
Class: Teenager
  EquivalentTo: Person and (hasAge some integer[> 12 , < 20])
```

Les 3 profils de OWL1 (1)

- Première version du **W3C de OWL (2004) = OWL1**
- **OWL1 a 3 profils :**
  - **OWL1-Lite** : correspondant à la LD $\mathcal{SHIF}(\mathcal{D})$
  - **OWL1-DL** : *(DL pour Logique de Description)* associée à la LD $\mathcal{SHOIN}(\mathcal{D})$

    $(\mathcal{D})$ pour « Data property » car sont distingués 2 types de rôles : ceux liant 2 individus, et ceux associant un individu à un littéral.
  - **OWL1-Full** : comprend tout **OWL1-DL**, avec **en plus tout RDF**
- **OWL1-Lite** et **OWL1-DL** :
  - ne sont **pas des extensions de RDF** : un triplet RDF n'est pas nécessairement valide dans ces 2 sous-langages : raison pour laquelle on a le sous-langage OWL-Full
  - possèdent **une syntaxe abstraite définie** : correspondance directe entre cette syntaxe et le formalisme utilisé en LD (syntaxe traduisible en RDF/XML)

Les 3 profils de OWL1 (2)

- **OWL1 Lite :**
  - **simple, facile à programmer,**
  - **expressivité limitée** à des hiérarchies de classes et des contraintes simples de cardinalité 0 ou 1 (relations fonctionnelles)
    Ex : une personne a une seule adresse, mais peut avoir un ou plusieurs prénoms, OWL Lite ne le permet pas
  - **raisonnements complets et rapides**
- **OWL1 DL :**
  - **DL pour « Logique de Description »**
  - **expressivité élevée** : contient tout OWL avec certaines restrictions
    Ex : une classe ne peut pas aussi être un individu ou une propriété
  - **raisonnements :**
    - **ceux pouvant être faits dans une LD**
    - **plus lents que dans OWL-Lite**
    - **complétude du calcul** : toutes les inférences seront assurées d'être prises en compte
    - **décidabilité** : tous les calculs seront terminés dans un temps fini

Les 3 profils de OWL (3)

- **OWL1 Full :**
  - **expressivité maximale**
  - **compatibilité complète avec RDF/RDFS**
  - **raisonnements,** souvent :
    - très **complexes**
    - très **lents**
    - **incomplets**
    - **indécidables**

Les 3 profils de OWL1 (4)

Expressivité :

On a : OWL1 Lite ⊂ OWL1 DL ⊂ OWL1 Full

Raisonnements [Staab 2007] **:**

Sous langage OWL	Complexité des données	Complexité combinée
OWL1 Lite	NP	Exptime
OWL1 DL	Inconnu	NExptime
OWL1 Full	indécidable	indécidable

- *Complexité des données : en supposant une T-Box donnée, complexité par rapport à la taille de la ABox*
- *Complexité combinée : par rapport à des tailles combinées de ABox et deTBox*

La plupart des systèmes fournissent une implémentation de **OWL1 Lite** ou de **OWL1 DL**

2. Profil OWL1-Lite

- **Syntaxe des axiomes « classe »**
- **Syntaxe des restrictions**
- **Syntaxe des types de données**
- **Syntaxe axiomes « propriété » et « équivalence de classes »**
- **Exemple d'ontologie en OWL-Lite**

OWL1-Lite : Axiomes « classe »

En OWL-Lite, une **classe** est définie ainsi **:**

> **axiome** ::= *Class(* [**classeID**] **modalité** { **superclasse** } *)*
> **modalité** ::= *complete* | *partial*
> **superclasse** ::= **classeID** | **restriction**

la **modalité est obligatoire** :

- *complete* indique qu'il s'agit d'une définition (≡),
- *partial* indique qu'il s'agit d'un axiome de spécialisation (⊑).

- Exemple simple d'ontologie avec un seul axiome :
  - En LD :
    Chat ⊑ Animal
  - En OWL-Lite :
    Namespace(local = <http://www.lsis.org#>)
    Ontology(
    Class(local:Chat partial local:Animal))
- En OWL-Lite, possible de spécifier que 2 ou plusieurs classes sont équivalentes, mais ssi ces classes ont un identificateur :
  axiome ::= EquivalentClasses(classeID classeID {classeID})

OWL1-Lite : Restrictions

Types de restriction déclarables en OWL-Lite sont d'une des formes:

- ***∀R.C :*** quantifications universelle
- ***∃R.A*** : quantification existentielle, mais avec **A = concept atomique**
- *de* ***cardinalité*** : les seules valeurs permises sont 0 et 1

Syntaxe OWL-Lite pour les restrictions :

- **restriction** ::= *restriction(* propriétéIndividuID élémentRestrictionIndividu *)* | *restriction(* propriétéLitteralID élémentRestrictionLittéral *)*
- **élémentRestrictionIndividu** ::= *allValuesFrom(* classeID *)* | someValuesFrom(classeID) | cardinalité
- **élémentRestrictionLittéral** ::= *allValuesFrom(* dataRange *)* | someValuesFrom(dataRange) | cardinalité
- **cardinalité** ::= minCardinality(0) | minCardinality(1) | maxCardinality(0) | maxCardinality(1) | cardinality(0) | cardinality(1)
- **dataRange** ::= typeLittéralID | *rdfs :Literal*

OWL1-Lite : Types de données

On peut utiliser **plusieurs types de données pour qualifier les restrictions de littéral**, notamment :

- xsd:string,
- xsd:boolean,
- xsd:decimal,
- xsd:float,
- xsd:double,
- xsd:time,
- xsd:date,
- xsd:int ,
- xsd:positiveInteger,
- …

OWL1-Lite : exemple 1 d'ontologie

soit à définir les concepts « **célibataire** » et « **végétarien** » :

- **En LD :**

  Célibataire ≡ Personne ⊓ ≤ 0 mariéAvec

  Végétarien ≡ Personne ⊓ ∀mange.Vegetal

- **En OWL-Lite :**

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    Class(local:Célibataire complete
        local:Personne
        restriction(local:mariéAvec maxCardinality(0)))
    Class(local:Végétarien complete
        local:Personne
        restriction(local:mange allValuesFrom(local:Vegetal))))
```


OWL1-Lite : Axiome « propriété » (2)

Syntaxe des axiomes « propriété » :

axiom ::= *DatatypeProperty(* **propriétéLittéralID**
*{super(***propriétéLittéralID** *)}*
[Functional]
{domain(**classeID** *)}*
{range(**dataRange** *)}*)
| *ObjectProperty(* **propriétéIndividuID**
*{super(***propriétéIndividuID** *)}*
*{inverseOf(***propriétéIndividuID** *)}*
[Functional | InverseFunctional | Functional InverseFunctional | Transitive]
{domain(**classeID** *)}*
{range(**classeID** *)}*)

OWL1-Lite : Axiomes « propriété » (1)

- Dans OWL-Lite on peut par un axiome :
  - **déclarer** une **propriété**
  - **définir** son **domaine** et son **image**
  - la déclarer **fonctionnelle**.
  - déclarer une propriété comme **sous-propriété d'une autre propriété**,
- Dans le cas de propriétés établissant une relation avec un autre individu, on peut de plus la déclarer :
  - **symétrique**
  - **transitive**
  - **injective** (il ne peut y avoir 2 individus du domaine qui ont la même valeur dans l'image de la fonction)

    En OWL, on utilise la **terminologie « inverse fonctionnelle »** pour désigner une **relation injective**

OWL1-Lite : Axiomes « propriété » (3)

- La spécification de **l'image** ou du **domaine** d'une **propriété** dans OWL-Lite ne demande pas une LD plus expressive que $\mathcal{AL}$
- **Pour exprimer que :**
  - **un rôle R ne peut s'appliquer qu'à des individus de la classe C** , on écrit l'axiome : **∃R.⊤ ⊑ C**
  - **tous les individus du domaine d'un rôle R appartiennent à la classe C**, on écrit l'axiome : **⊤ ⊑ ∀R.C**

Remarque : ne pas confondre ces **axiomes** avec des **restrictions de rôle** :

- supposons un axiome de la forme : $C_1 \equiv C_2 \sqcap \forall R.C_3$
- cet axiome n'impose pas que tous les éléments du domaine de R appartiennent à la classe C_3
- on peut ajouter à l'ontologie l'axiome : $C_4 \equiv C_5 \sqcap \forall R.C_6$

OWL1-Lite : Axiomes « équivalence de classes »

OWL-Lite permet aussi d'écrire des axiomes pour :

- spécifier des **équivalences de classe**
- déclarer qu'une **propriété est sous-propriété d'une autre propriété** (redondant par rapport à l'élément *super(…)* d'axiome de propriété) :

Syntaxe des axiomes d'équivalence :

axiom ::= *EquivalentProperties(* **propriétéLittéralID propriétéLittéralID** {**propriétéLittéralID**} *)*
| *SubPropertyOf(* **propriétéLittéralID propriétéLittéralID** *)*
| *EquivalentProperties(* **propriétéIndividuID propriétéIndividuID** {**propriétéIndividuID**} *)*
| *SubPropertyOf(* **propriétéIndividuID propriétéIndividuID** *)*

OWL1-Lite : exemple 2 d'ontologie (1)

En Logique Descriptive :

Célibataire ≡ Personne ⊓ ≤ 0 mariéAvec
Personne ⊑ ∃nom.`xsd:string` ⊓ ∃age.`xsd:int`
Homme ⊑ Personne
Femme ⊑ Personne
Père ≡ Personne ⊓ ∃aEnfant.Personne
PèreDeFilles ≡ Père ⊓ ∀aEnfant.Femme
⊤ ⊑ ∀aEnfant.Personne (image)
∃aEnfant.⊤ ⊑ Personne (domaine)
aEnfant ≡ aParent^{-} (rôle inverse)
⊤ ⊑ ≤ 1 estConjointDe (rôle fonctionnel)
⊤ ⊑ ≤ 1 estConjointDe^{-} (rôle injectif)
estConjointDe ≡ estConjointDe^{-} (rôle symétrique)
⊤ ⊑ ∀estConjointDe.Personne (image)
∃estConjointDe.⊤ ⊑ Personne (domaine)
mariéAvec ⊑ estConjointDe
⊤ ⊑ ∀age.`xsd:int` (image)
∃age.⊤ ⊑ Personne (domaine)
⊤ ⊑ ∀nom.`xsd:string` (image)
∃nom.⊤ ⊑ Personne (domaine)

OWL1-Lite : exemple 2 d'ontologie (2)

En OWL-Lite :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    Class(local:Célibataire complete
        local:Personne
        restriction(local:mariéAvec maxCardinality(0)))
    Class(local:Personne partial
        restriction(local:nom someValuesFrom(xsd:string))
        restriction(local:age someValuesFrom(xsd:int)))
    Class(local:Homme partial local:Personne)
    Class(local:Femme partial local:Personne)
    Class(local:Père complete
        local:Personne
        restriction(local:aEnfant someValuesFrom(local:Personne)))
    Class(local:PèreDeFilles complete
        local:Père
        restriction(local:aEnfant allValuesFrom(local:Femme)))
```


OWL1-Lite : exemple 2 d'ontologie (3)

(suite OWL-Lite) :

```
    ObjectProperty(local:aEnfant
        domain(local:Personne)
        range(local:Personne)
        inverseOf(local:aParent))
    ObjectProperty(local:estConjointDe
        Functional
        InverseFunctional
        Symmetric
        domain(local:Personne)
        range(local:Personne))
    ObjectProperty(local:mariéAvec super(local:estConjointDe))
    DatatypeProperty(local:age
        domain(local:Personne)
        range(xsd:int))
    DatatypeProperty(local:nom
        domain(local:Personne)
        range(xsd:string)))
```

OWL1-Lite : exemple 2 d'ontologie (4)

En Logique Descriptive	En OWL-Lite
Célibataire ≡ Personne ⊓ ≤ 0 mariéAvec Personne ⊑ ∃nom.xsd:string ⊓ ∃age.xsd:int Homme ⊑ Personne Femme ⊑ Personne Père ≡ Personne ⊓ ∃aEnfant.Personne PèreDeFilles ≡ Père ⊓ ∀aEnfant.Femme ⊤ ⊑ ∀aEnfant.Personne (image) ∃aEnfant.⊤ ⊑ Personne (domaine) aEnfant ≡ aParent$^-$ (rôle inverse) ⊤ ⊑ ≤ 1 estConjointDe (rôle fonctionnel) ⊤ ⊑ ≤ 1 estConjointDe$^-$ (rôle injectif) estConjointDe ≡ estConjointDe$^-$ (rôle symétrique) ⊤ ⊑ ∀estConjointDe.Personne (image) ∃estConjointDe.⊤ ⊑ Personne (domaine) mariéAvec ⊑ estConjointDe ⊤ ⊑ ∀age.xsd:int (image) ∃age.⊤ ⊑ Personne (domaine) ⊤ ⊑ ∀nom.xsd:string (image) ∃nom.⊤ ⊑ Personne (domaine)	Namespace(local = <http://www.lsis.org#>) Ontology(Class(local:Célibataire complete local:Personne restriction(local:mariéAvec maxCardinality(0))) Class(local:Personne partial restriction(local:nom someValuesFrom(xsd:string)) restriction(local:age someValuesFrom(xsd:int))) Class(local:Homme partial local:Personne) Class(local:Femme partial local:Personne) Class(local:Père complete local:Personne restriction(local:aEnfant someValuesFrom(local:Personne))) Class(local:PèreDeFilles complete local:Père restriction(local:aEnfant allValuesFrom(local:Femme))) ObjectProperty(local:aEnfant domain(local:Personne) range(local:Personne) inverseOf(local:aParent)) ObjectProperty(local:estConjointDe Functional InverseFunctional Symmetric domain(local:Personne) range(local:Personne)) ObjectProperty(local:mariéAvec super(local:estConjointDe)) DatatypeProperty(local:age domain(local:Personne) range(xsd:int)) DatatypeProperty(local:nom domain(local:Personne) range(xsd:string)))

OWL1-Lite : Faits (1)

Les faits permettent de fournir des informations sur des entités spécifiques (appelées *individus*), notamment :

- les **classes auxquelles l'individu appartient**,
- les **valeurs des propriétés de l'individu** (valeurs qui sont des individus ou des littéraux selon des propriétés)

fait ::= individu
individu ::= *Individual(* [**individuID**] { *type(* type *)* } { valeur} *)*
valeur ::= *value(* **propriétéIndividu IndividuID** *)*
| *value(* **propriétéIndividu Individu***)*
| *value(* **propriétéLittéral Littéral** *)*

- Le type d'un individu est soit un **concept atomique**, soit une **restriction** :

type ::= classeID
| restriction

- Autre type de fait en OWL-Lite pour spécifier que plusieurs identificateurs d'individus représentent le même individu ou des individus différents :

fait ::= *SameIndividual(* **individuID individuID** { **individuID** }*)*
| *DifferentIndividuals(* **individuID individuID** { **individuID** }*)*

OWL1-Lite : exemple de faits (1)

- A l'ontologie précédente, rajoutons les faits suivants :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
...
Individual(local:Valentin
        type(local:Homme)
        type(local:Célibataire)
        value(local:nom "Valentin"^^xsd:string)
        value(local:age "10"^^xsd:int))
Individual(local:Bernard
        type(local:Homme)
        value(local:mariéAvec
            Individual(
                type(local:Femme)
                    value(local:nom "Sabine"^^xsd:string)
                    value(local:age "46"^^xsd:int)))
                    value(local:nom "Bernard"^^xsd:string)
                    value(local:age "54"^^xsd:int)
                    value(aEnfant local:Valentin))
...)
```

Ces faits représentent 3 individus (2 hommes et une femme) : Bernard est le père de Valentin et est marié avec une femme Sabine

Remarque : Aucune URI donné pour identifier Sabine, sa description est faite à l'endroit même où la valeur du rôle estMariéAvec est spécifiée pour Bernard.

OWL1-Lite : exemple de faits (2)

Si on connaît l'URI de l'individu Sabine, soit local:Sabine, on peut écrire les mêmes faits en séparant complètement les descriptions des 3 individus :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    ...
    Individual(local:Valentin
        type(local:Homme)
        type(local:Célibataire)
        value(local:nom "Valentin"^^xsd:string)
        value(local:age "10"^^xsd:int))
    Individual(local:Bernard
        type(local:Homme)
        value(local:mariéAvec local:Sabine)
        value(local:nom "Bernard"^^xsd:string)
        value(local:age "54"^^xsd:int)
        value(aEnfant local:Valentin))
    Individual(local:Sabine
        type(local:Femme)
        value(local:nom "Sabine"^^xsd:string)
        value(local:age "46"^^xsd:int))
    ...)
```

OWL1-Lite : exemple de faits (3)

- Autre représentation de ces faits :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    ...
    Individual(local:Bernard
        type(local:Homme)
        value(local:mariéAvec
        Individual(local:Sabine
            type(local:Femme)
            value(local:nom "Sabine"^^xsd:string)
            value(local:age "46"^^xsd:int)))
        value(local:nom "Bernard"^^xsd:string)
        value(local:age "54"^^xsd:int)
        value(aEnfant
            Individual(local:Valentin
                type(local:Homme)
                type(local:Célibataire)
                value(local:nom "Valentin"^^xsd:string)
                value(local:age "10"^^xsd:int))))
...)
```


3. Profil OWL1-DL

- **Syntaxe des axiomes**
- **Syntaxe des restrictions**
- **Syntaxe des types de données**
- **Syntaxe axiomes propriété et équivalence de classes**
- **Exemple d'ontologie en OWL-DL**

DL $\mathcal{SHOIN}(\mathcal{D})$ (1)

(d'après STAAB)

	Concepts	
ALC	Atomic	A, B
	Not	¬C
	And	C ⊓ D
	Or	C ⊔ D
	Exists	∃R.C
	For all	∀R.C
Q(N)	At least	≥n R.C (≥n R)
	At most	≤n R.C (≤n R)
O	Nominal	{i_1,…,i_n}

	Roles	
I	Atomic	R
	Inverse	R^-

Ontology (=Knowledge Base)

Concept Axioms (TBox)	
Subclass	C ⊑ D
Equivalent	C ≡ D

	Role Axioms (RBox)	
H	Subrole	R ⊑ S
S	Transitivity	Trans(S)

Assertional Axioms (ABox)	
Instance	C(a)
Role	R(a,b)
Same	a = b
Different	a ≠ b

- **OWL-DL** : correspond la **DL** $\mathcal{SHOIN}(\mathcal{D})$
- ($\mathcal{D}$) « Data property »

DL $\mathcal{SHOIN}(\mathcal{D})$ (2)

(d'après STAAB)

Exemple :

Terminological Knowledge (*TBox*):

Humain ⊑ ∃ parentDe.Humain

Orphelin ≡ Humain ∪ ¬∃ enfantDe.Vivant

Knowledge about Individuals (*ABox*):

Orphelin (harrypotter)

parentDe(jamespotter, harrypotter)

OWL1-DL / DL $\mathcal{SHOIN}(\mathcal{D})$: constructeurs de classes

- **OWL-DL** : correspond la **DL $\mathcal{SHOIN}(\mathcal{D})$**
- emboitement d'expressions autorisé : Person ⊓ ∀ hasChild.(Doctor ⊔ ∃hasChild.Doctor)

Constructeur de OWL DL	Constructeurs de $\mathcal{SHOIN}(\mathcal{D})$	DL Exemple
Thing	⊤	⊤
Nothing	⊥	⊥
intersectionOf(C_1… C_n)	$C_1 \sqcap \ldots \sqcap C_n$	Humain ⊓ Male
unionOf(C_1… C_n)	$C_1 \sqcup \ldots \sqcup C_n$	Docteur ⊔ Avocat
complementOf(C)	¬C	¬Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{Paul} ⊔ {Marie}

Avec :

- ⊤ est le **concept universel** (top concept) A ⊔ ¬A qui contient tous les individus
- **C ≡ D ⇔ (C ⊑ D et D ⊑ C)** (≡ axiome d'équivalence ; ⊑ d'inclusion)

OWL1-DL / DL $\mathcal{SHOIN}(\mathcal{D})$: constructeurs de rôles

Constructeur de OWL DL	Constructeurs de $\mathcal{SHOIN}(\mathcal{D})$	DL Exemple
Rôle P		
P **allValuesFrom** *(C)*	∀P.C	∀aEnfant.Docteur
P **someValuesFrom** *(C)*	∃P.C	∃aEnfant.Avocat
P **hasValue** *(x)*	∃P.{x}	∃aEnfant.Alice
P **maxCardinality** *(n)*	≤ n P	≤ 1 aEnfant
P **minCardinality** *(n)*	≥ n P	≥ 2 aEnfant

- **Remarque** : dans **DL $\mathcal{SHOIN}(\mathcal{D})$** le ($\mathcal{D}$) « Data property » signifie que 2 types de rôles sont distingués : ceux liant 2 individus, et ceux associant un individu à un littéral)

OWL1-DL / DL $\mathcal{SHOIN}(\mathcal{D})$: axiomes

Axiomes de OWL DL	Axiomes de $\mathcal{SHOIN}(\mathcal{D})$	DL Exemple
Class(C **partial** C_1 … C_n)	$A \sqsubseteq C_1 \sqcap \ldots C_n$	Humain ⊑ Animal ⊓ Bipede
Class(A **complete** C_1 … C_n)	$A \equiv C_1 \sqcap \ldots C_n$	Homme ≡ Humain ⊓ Male
EnumeratedClass(A x_1 … x_n)	$A \equiv \{x_1, \ldots, x_n\}$	Male ⊑ ¬Femelle
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$	Humain ⊑ Animal ⊓ Bipede
EquivalentClass(C_1 C_2)	$C_1 \equiv C_2$	Homme ≡ Humain ⊓ Male
DisjointWith(C_1 C_2)	$C_1 \sqsubseteq \neg C_2$	Male ⊑ ¬Femelle
sameIndividualAs(x_1 x_2)	$\{x_1\} \equiv \{x_2\}$	{Brigitte Bardot } ≡ {BB}
differentFrom(x_1 x_2)	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{Paul} ⊑ ¬{Pierre}
SubPropertyOf(P_1 P_2)	$P_1 \sqsubseteq P_2$	aFille ⊑ aEnfant
EquivalentProperties(P_1 P_2)	$P_1 \equiv P_2$	Cout ≡ Prix
inverseOf(P_2)	$P_1 \equiv P_2^-$	aEnfant ≡ aParent $^-$
transitivePropertyOf(P)	$P^+ \equiv P^+$	ancêtre$^+$ ⊑ ancêtre
functionalpropertyOf(P)	$\top \sqsubseteq \leq 1\ P$	T ⊑ ≤1hasMère

inverseFunctionalProperties(P)	$\top \sqsubseteq \leq 1\ P^-$	T ⊑ ≤1hasSSN $^-$

OWL1 - syntaxe de RDF Schema

Exemple :

Personne ⊓ ∀aEnfant.(Docteur ⊔ ∃aEnfant.Docteur)

```
<owl:Class>
    <owl:intersectionOf rdf:parseType="collection">
        <owl:Class rdf:about="#Personne"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#aEnfant"/>
            <owl:toClass>
                <owl:unionOf rdf:parseType="collection">
                    <owl:Class rdf:about="#Docteur"/>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#aEnfant"/>
                        <owl:hasClass rdf:resource="#Docteur"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:toClass>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

OWL1-DL versus OWL1-Lite

- **OWL1 Lite ⊂ OWL1 DL**
- La **plupart des systèmes** fournissent une implémentation de **OWL Lite** ou **OWL DL**
- **Dans OWL1 Lite :**
  - **constructeurs de OWL1 DL ne pouvant pas être utilisés :**
    - **unionOf**
    - **complementOf**
    - **oneOf**
    - **hasValue**
    - **disjointWith**
  - **constructeurs de OWL1 DL dont l'applicabilité est restreinte :**
    - **intersectionOf**
    - **minCardinality**
    - **maxCardinality**
    - **cardinality**
- **OWL1-Full comprend tout OWL-DL, avec en plus tout RDF**

OWL-DL : Axiomes « classe » (1)

- **OWL1-Lite** permet des définitions contenant seulement des concepts atomiques ou des restrictions limitées
- **OWL1-DL** permet des définitions avec des descriptions plus complexes
- **Syntaxe des axiomes en OWL1-DL :**

axiome ::= *Class(* [**classeID**] **modalité** { **description** } *)*
modalité ::= *complete* | *partial*
description ::= **classeID**
| **restriction**
| *unionOf(* {**description** } *)*
| *intersectionOf(* {**description** } *)*
| *complementOf(* **description** *)*
| *oneOf(* {**individuID** } *)*

- OWL1-DL permet de **déclarer des classes équivalentes, des classes disjointes, et des axiomes de spécialisation** :

axiome ::= *DisjointClasses(* **description description** {**description** } *)*
| *EquivalentClasses(* **description** { **description** } *)*
| *SubClassOf(* **description description** *)*

OWL1-DL : Axiomes « classe » (2)

- Les classes impliquées dans les axiomes peuvent avoir des descriptions complexes
- La déclaration de classes équivalentes peut ne contenir qu'une seule classe, ce qui revient à déclarer une classe
- On peut faire la même chose avec un axiome de la forme :

Class(**classeID** *partial).*

- OWL-DL offre un autre type d'axiome pour définir une classe par énumération, en fournissant la liste de tous les individus qui la composent :

axiome ::= *EnumeratedClass(* **classeID** { **individuID** } *)*

OWL1-DL : Axiomes « classe » (3)

Souvent plusieurs manières de définir un même axiome : Ex : région PACA (Provence Alpes Côte d'Azur) de France composée des départements : {04, 05, 06, 13, 83, 84} : DepartementMembrePaca ≡ {04, 05, 06, 13, 83, 84}

- Peut être définit de **2 façons équivalentes** en OWL-DL :

1 : Namespace(local = <http://www.lsis.org#>)
Ontology(
Class(local:DepartementMembrePaca complete
oneOf(local:04 local:05 local:06 local:13 local:83 local:84)))
ou
2 : Namespace(local = <http://www.lsis.org#>)
Ontology(
EnumeratedClass(local:DepartementMembrePaca
local:04 local:05 local:06 local:13 local:83 local:84))

- **1° forme** : la classe décrite par énumération demeure anonyme, et est ainsi locale à l'axiome.
- **2° forme** : permet de fournir un identificateur à un concept défini par énumération, ce qui permettra de le réutiliser à plusieurs endroits.

OWL1-DL : exemple (1)

Exemple d'ontologie en OWL-DL, avec les axiomes précédents :

En Logique de Description :

AnimalDeCompagnie ≡ AnimalDomestique ⊓ (Chien ⊔ Chat)
(Chien ⊓ Chat) ⊑ ⊥
AnimalDomestique ≡ Animal ⊓ ¬AnimalSauvage

En OWL1-DL :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    Class(local:Animal partial)
    Class(local:AnimalSauvage partial)
    Class(local:AnimalDeCompagnie complete
        local:AnimalDomestique
        unionOf(
            local:Chien
            local:Chat))
    DisjointClasses(local:Chien local:Chat)
    Class(local:AnimalDomestique complete
        local:Animal
        complementOf(local:AnimalSauvage)))
```


OWL-DL : exemple (2)

Une autre forme équivalente en OWL-DL :

```
Namespace(local = <http://www.lsis.org#>)
Ontology(
    Class(local:Animal partial)
    Class(local:AnimalSauvage partial)
    Class(local:AnimalDeCompagnie complete
        intersectionOf(
            local:AnimalDomestique
            unionOf(
                local:Chien
                local:Chat)))
    DisjointClasses(local:Chien local:Chat)
    Class(local:AnimalDomestique complete
        intersectionOf(
            local:Animal
            complementOf(local:AnimalSauvage))))
```


OWL1-DL : Restrictions

Restrictions dans OWL-DL = celles d'OWL-Lite plus :

- de descriptions complexes,
- de n'importe quelle valeur entière dans une restriction de cardinalité,
- de la spécification de l'individu pour remplir un rôle

restriction ::= *restriction(* **propriétéIndividuID élémentRestrictionIndividu** *)*
| *restriction(* **propriétéLitteralID élémentRestrictionLittéral** *)*
élémentRestrictionIndividu ::= *allValuesFrom(* **description** *)*
| *someValuesFrom(* **description** *)*
| *value(* **individuID** *)*
| cardinalité
élémentRestrictionLittéral ::= *allValuesFrom(* **dataRange** *)*
| *someValuesFrom(* **dataRange** *)*
| *value(* **littéral** *)*
| **cardinalité**
cardinalité ::= *minCardinality(***entierNonNégatif***)*
| *maxCardinality(***entierNonNégatif***)*
| *cardinality(***entierNonNégatif***)*
dataRange ::= **typeLittéralID** | *rdfs :Literal* | *oneOf(* { **littéral** } *)*

OWL1-DL : Axiomes « propriété »

Axiomes propriétés : idem OWL-Lite, mais pas limités à des concepts atomiques (permettent n'importe quelle description complexe de concept) :

axiom ::= *DatatypeProperty(* **propriétéLittéralID**
*{super(***propriétéLittéralID** *)}*
[Functional]
{domain(**description** *)}*
{range(**dataRange** *)})*
| *ObjectProperty(* **propriétéIndividuID**
*{super(***propriétéIndividuID** *)}*
*{inverseOf(***propriétéIndividuID** *)}*
[Functional|InverseFunctional|Functional
InverseFunctional|Transitive]
{domain(**description** *)}*
{range(**description** *)})*

OWL1-DL : Axiomes « équivalence propriétés »

Axiomes équivalences : idem OWL-Lite :

axiom ::= *EquivalentProperties(* **propriétéLittéralID propriétéLittéralID**
{**propriétéLittéralID**} *)*
| *SubPropertyOf(* **propriétéLittéralID propriétéLittéralID** *)*
| *EquivalentProperties(* **propriétéIndividuID propriétéIndividuID**
{**propriétéIndividuID**} *)*
| *SubPropertyOf(* **propriétéIndividuID propriétéIndividuID** *)*

OWL1-DL : Faits

- En **OWL-Lite**, la classe d'un individu dans la ABox ne peut être qu'une intersection de concepts atomiques ou de restrictions simple.
- En **OWL-DL**, elle peut être n'importe quelle description complexe :

fait ::= individu
| *SameIndividual(* individuID individuID { individuID }*)*
| *DifferentIndividuals(* individuID individuID { individuID }*)*
individu ::= *Individual(* [individuID] { *type(* description *)* } { valeur} *)*
valeur ::= *value(* propriétéIndividu IndividuID *)*
| *value(* propriétéIndividu Individu*)*
| *value(* propriétéLittéral Littéral *)*

4. Traduction d'une ontologie en OWL1-DL et en RDFS/XML

- **Traduction des identificateurs, déclarations de classes, descriptions de propriétés, des axiomes sur propriétés et descriptions d'individus**
- **Exemple : expression d'une ontologie en LD, traduction en OWL1-DL (syntaxe abstraite), et traduction en RDF/S-XML**

Traduction d'une ontologie OWL en RDF

- On utilise une fonction de traduction T qui, appliquée récursivement à l'ontologie, transforme ses composantes en **triplets RDF** :
- **Traduction des identificateurs :**

OWL	RDF
classeID	classID rdf: type owl:Class
individuID	individuID
propriétéIndividuID	propriétéIndividuID rdf:type owl:ObjectProperty
propriétéLittéralID	propriétéIndividuID rdf:type owl:DatatypeProperty

- **Traduction d'une ontologie :**

Ontology(ID directive$_1$ … directive$_n$)	ID rdf:type owl:Ontology T(directive$_1$) … Tdirective$_n$)
Ontology(directive$_1$ . . . directive$_n$)	T(directive$_1$) … Tdirective$_n$)

Traduction en RDF (2)

- **Traduction des déclarations de classes** *(LIST = fonction de traduction intermédiaire retournant une liste d'items):*

OWL	RDF
Class(ID partial description$_1$ … description$_n$)	ID rdf:type owl:Class ID rdfs:subClassOf T(description$_1$) … ID rdfs:subClassOf T(description$_n$)
Class(ID complete description$_1$ … description$_n$)	ID rdf:type owl:Class ID owl:intersectionOf *LIST*(description$_1$ … description$_n$)
Class(ID complete description)	ID rdf:type owl:Class ID owl:equivalentClass T(dimension)
Class(ID complete unionOf(description$_1$ … description$_n$)	ID rdf:type owl:Class ID owl:unionOf *LIST*(description$_1$ … description$_n$)
Class(ID complete intersectionOf(description$_1$ … description$_n$)	ID rdf:type owl:Class ID owl:intersectionOf *LIST*(description$_1$ … description$_n$)
Class(ID complete complementOf(description))	ID rdf:type owl:Class ID owl:complementOfClass T(dimension)
EnumeratedClass(ID complete individu$_1$ … individu$_n$)	ID rdf:type owl:Class ID owl:oneOf *LIST*(description$_1$ … description$_n$)

Traduction en RDF (3)

(LIST = fonction de traduction intermédiaire retournant une liste d'items)

- **Traduction des restrictions :**

…

- **Traduction des descriptions de propriétés :**

…

- **Traduction des axiomes sur propriétés et descriptions d'individus :**

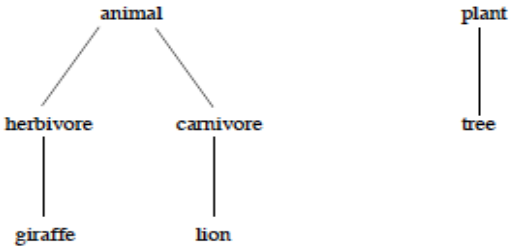
…

Exemple : l'ontologie « African Wildelife Ontology »

(Source : G. Antoniou)

On s'intéresse à l'ontologie « African Wildelife Ontology »

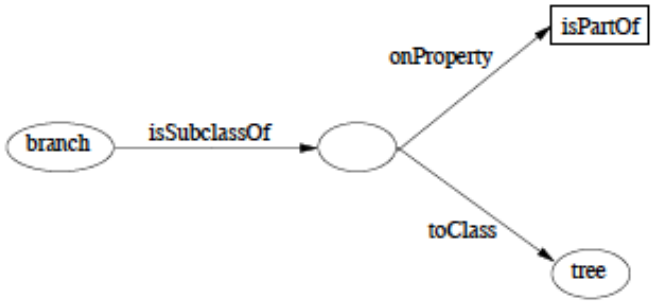
- Cette ontologie décrit une partie de la vie sauvage africaine
- On s'intéresse ici à un sous-ensemble de cette ontologie, centré sur les classes et sous-classes représentées par les sous-graphes suivants (class hierarchy) :



- Remarque : le graphe complet associé à l'ontologie est bien plus grand.

African Wildelife Ontology

Les graphes suivants illustrent le fait que les branches (branch) sont des parties de (is-part-of) d'arbre (tree) :



On aurait une représentation similaire pour indiquer que les feuilles (leaf) sont sont des parties de (is-part-of) d'arbre (tree)

African Wildelife Ontology : expression en LD

(Source : G. Antoniou)

Cette partie de l'ontologie « African Wildelife Ontology » pourrait s'exprimer en LD ainsi :

$eats \equiv eaten\text{-}by^{-}$
$is\text{-}part\text{-}of \equiv has\text{-}part^{-}$
$is\text{-}part\text{-}of \equiv is\text{-}part\text{-}of^{+}$
$Plant \sqcup \exists is\text{-}part\text{-}of.Plant \sqsubseteq \neg Animal$
$Tree \sqsubseteq Plant$
$Branch \sqsubseteq \exists is\text{-}part\text{-}of.Tree$
$Leaf \sqsubseteq \exists is\text{-}part\text{-}of.Branch$

$Carnivore \equiv Animal \sqcap \forall eats.Animal$
$Herbivore \equiv Animal \sqcap \forall eats.(Plant \sqcup \exists is\text{-}part\text{-}of.Plant)$
$Giraffe \sqsubseteq Animal \sqcap \forall eats.Leaf$
$Lion \sqsubseteq Animal \sqcap \forall eats.Herbivore$
$TastyPlant \equiv Plant \sqcap \exists eaten\text{-}by.Herbivore \sqcap \exists eaten\text{-}by.Carnivore$

African Wildelife Ontology : quelques déductions possibles en LD

Quelques inférences pouvant être faites sur la LD précédente :

- **Lion is a subclass of Carnivore**
- **Giraffe is a subclass of Herbivore**
- **Herbivore and Carnivore are disjoint**
- **Leaf is-part-of Tree**
- **Tasty-Plant is inconsistent**

African Wildelife Ontology : expression en syntaxe abstraite OWL1-DL (1)

(Source : G. Antoniou)

```
ObjectProperty (eats
      InverseOf (is-eaten-by))
ObjectProperty (has-part
      InverseOf(is-part-of)
      transitive)
Class(Animal)
DisjointClasses (Animal, unionOf(Plant
                                 restriction (is-part-of
                                 SomeValuesFrom Plant)))
Class (Tree partial Plant)
Class (Branch partial
      restriction (is-part-of SomeValuesFrom Tree))
Class (Leaf partial
      restriction (is-part-of SomeValuesFrom Branch))
```


African Wildelife Ontology : expression en syntaxe abstraite OWL1-DL (2)

(Source : G. Antoniou)

```
Class (Carnivore complete
      Animal
      restriction (eats AllValuesFrom Animal))
Class (Herbivore complete
      Animal
            restriction (eats
                  AllValuesFrom unionOf(Plant
                        restriction (is-part-of
                        SomeValuesFrom Plant)))))
Class (Giraffe partial
      Animal
      restriction(eats SomeValuesFrom Leaf))
Class (Lion partial
      Animal
      restriction(eats AllValuesFrom Herbivore))
Class (Tasty Plant complete
```

Plant
restriction(eaten-by SomeValuesFrom Herbivore)
restriction(eaten-by SomeValuesFrom Carnivore))

African Wildelife Ontology : expression en syntaxe RDFS-XML (1)

(Source : G. Antoniou)

Entête:

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:owl ="http://www.w3.org/2002/07/owl#"
        xmlns="http://www.mydomain.org/african">
<owl:Ontology rdf:about="">
    <owl:VersionInfo>
    My example version 1.2, 17 October 2002
    </owl:VersionInfo>
</owl:Ontology>
```

Animal:

```
<owl:Class rdf:ID="animal">
```



```
<rdfs:comment>Animals form a class</rdfs:comment>
</owl:Class>
```

African Wildelife Ontology : expression en syntaxe RDFS-XML (2)

Properties:

```
<owl:TransitiveProperty rdf:ID="is-part-of"/>
<owl:ObjectProperty rdf:ID="eats">
    <rdfs:domain rdf:resource="#animal"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="eaten-by">
    <owl:inverseOf rdf:resource="#eats"/>
</owl:ObjectProperty>
```

Plants and Trees:

```
<owl:Class rdf:ID="plant">
    <rdfs:comment>Plants are disjoint from animals. </rdfs:comment>
    <owl:disjointWith="#animal"/>
</owl:Class>
<owl:Class rdf:ID="tree">
    <rdfs:comment>Trees are a type of plant. </rdfs:comment>
```



```
    <rdfs:subClassOf rdf:resource="#plant"/>
</owl:Class>
```


African Wildelife Ontology : expression en syntaxe RDFS-XML (3)

Branches:

```
<owl:Class rdf:ID="branch">
    <rdfs:comment>Branches are parts of trees. </rdfs:comment>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#is-part-of"/>
            <owl:allValuesFrom rdf:resource="#tree"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Leaves:

```
<owl:Class rdf:ID="leaf">
    <rdfs:comment>Leaves are parts of branches. </rdfs:comment>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#is-part-of"/>
            <owl:allValuesFrom rdf:resource="#branch"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

African Wildelife Ontology : expression en syntaxe RDFS-XML (4)

Carnivores:

```
<owl:Class rdf:ID="carnivore">
    <rdfs:comment>Carnivores are exactly those animals
    that eat also animals.</rdfs:comment>
    <owl:intersectionOf rdf:parsetype="Collection">
    <owl:Class rdf:about="#animal"/>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:someValuesFrom rdf:resource="#animal"/>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```


African Wildelife Ontology : expression en syntaxe RDFS-XML (5)

Herbivores:

```
<owl:Class rdf:ID="herbivore">
        <rdfs:comment>
        Herbivores are exactly those animals that eat only plants or parts of plants.
        </rdfs:comment>
        <owl:intersectionOf rdf:parsetype="Collection">
            <owl:Class rdf:about="#animal"/>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#eats"/>
                <owl:allValuesFrom>
                <owl:unionOf rdf:parsetype="Collection">
                    <owl:Class rdf:about="#plant"/>
                    <owl:Restriction>
                      <owl:onProperty rdf:resource="#is-part-of"/>
                      <owl:allValuesFrom rdf:resource="#plant"/>
                    </owl:Restriction>
                </owl:unionOf>
            </owl:allValuesFrom>
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```


African Wildelife Ontology : expression en syntaxe RDFS-XML (6)

Giraffes:

```
<owl:Class rdf:ID="giraffe">
    <rdfs:comment>Giraffes are herbivores, and they
    eat only leaves.</rdfs:comment>
    <rdfs:subClassOf rdf:type="#herbivore"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:allValuesFrom rdf:resource="#leaf"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```


African Wildelife Ontology : expression en syntaxe RDFS-XML (7)

Lions:

```
<owl:Class rdf:ID="lion">
    <rdfs:comment>Lions are animals that eat only herbivores.</rdfs:comment>
    <rdfs:subClassOf rdf:type="#carnivore"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#eats"/>
            <owl:allValuesFrom rdf:resource="#herbivore"/>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Tasty Plants:

```
owl:Class rdf:ID="tasty-plant">
    <rdfs:comment>Plants eaten both by herbivores and carnivores
    </rdfs:comment>
    <rdfs:comment>
        Try it out! See book for code.
    <rdfs:comment>
</owl:Class>
```

Environnement et outils pour OWL

OWL-API : interface Java

Éditeur d'ontologie :

- **Protégé** (Univ. Stanford) et plug-ins associés : Visualisation avancée (Graphviz), Jambalaya, OWL-S plugin pour la modélisation et l'exécution de processus, Interface DIG pour moteur de raisonnement …
- **SWOOP** (repris par Google en 2007)
- **POWL** et **Ontowiki** (en PHP sur le WEB – manque de maturité mais en progression)
- TopBraid Composer (Commercial)
- ..

Moteurs d'inférences (ou de raisonnement) :

- **Pellet**
- Fact++ (Manchester)
- KAON2
- **RacerPro** (propriétaire et très cher)
- **Jess** (propriétaire mais licences académiques)
- Bossam
- …