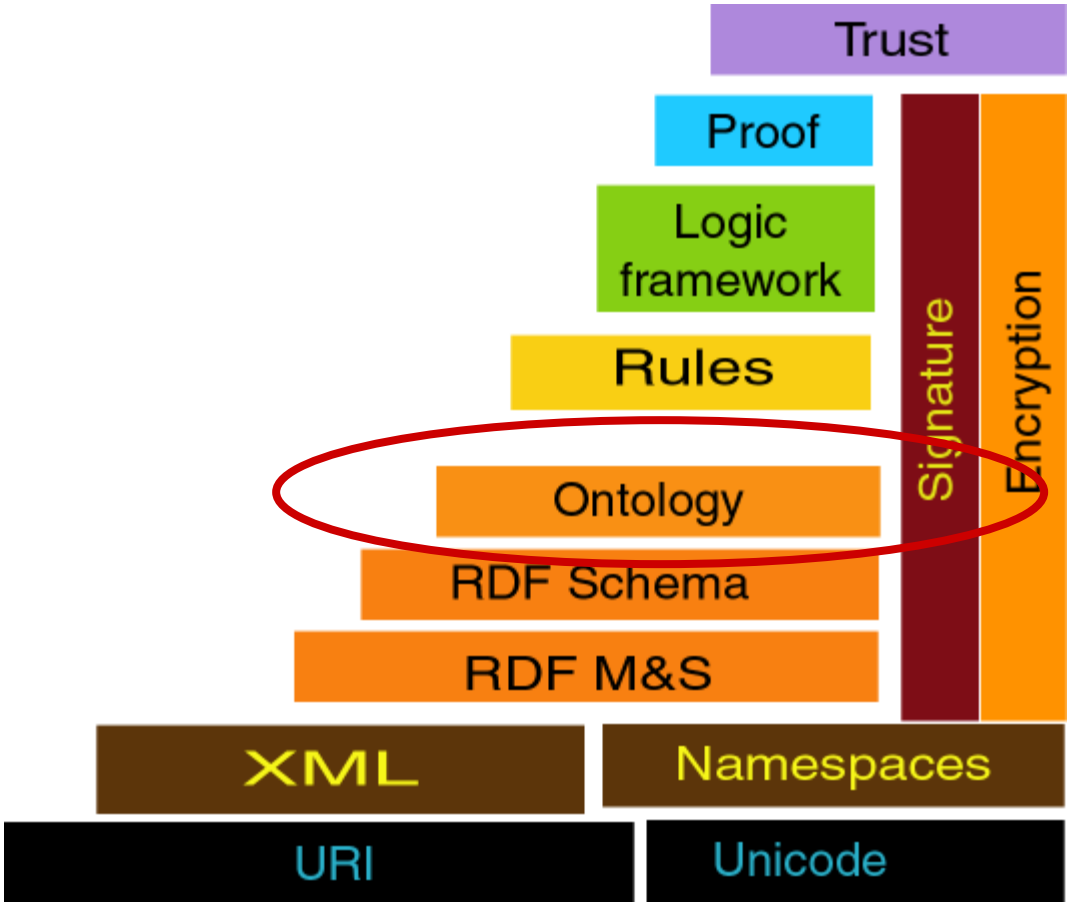
Ontologies et Web sémantique

Cours 7: OWL

Dr. TA Tuan Anh
ttanh@ciid.vast.vn

Rappel


Trust
Proof
Logic framework
Rules
Ontology
RDF Schema
RDF M&S
XML
Namespaces
URI
Unicode
Signature
Encryption

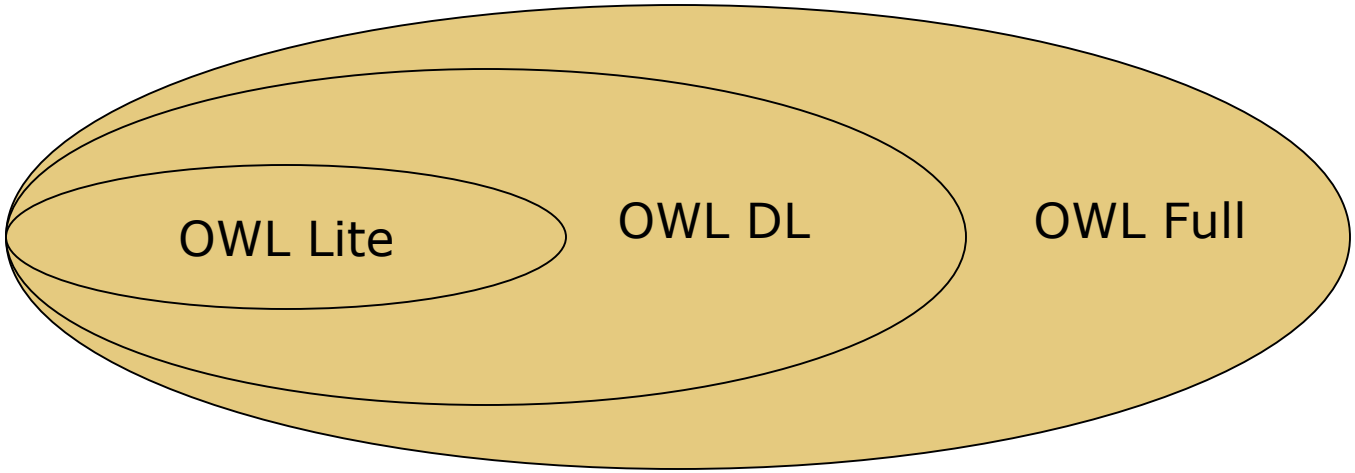
Plan

- Les trois sous langages OWL
- Les constructeurs et axiomes OWL
- Syntaxe en RDF/XML
- Editeur Protége
- Cycle de vie d’une ontologie

OWL : Langage des ontologies

- Offrir un langage standard pour définir des ontologies sur le Web (Ontology Web Language)
- Basé sur RDF(S) pour être compatible avec le Web sémantique
- Etend les constructions de base pour améliorer l'expressivité et le raisonnement a base de la logique de description
- Inspiré de DAML (Darpa) + OIL (EEC)

Les trois sous langages OWL



- OWL DL est nommé pour sa correspondance avec la logique de description
- OWL Lite est un sous ensemble de OWL DL pour faciliter l'implémentation
- OWL Full permet à une ontologie d'augmenter la signification du vocabulaire prédéfini en RDF(S)

Caractéristiques OWL

- OWL LITE :
  - ensemble réduit de constructeurs possibles
  - contraintes simples
  - calculable (NP)
- OWL DL
  - tous les constructeurs avec contraintes d'utilisation
  - assurent l'utilisabilité : complétude, décidabilité
  - correspond à certaines logiques de descriptions
- OWL FULL
  - tous les constructeurs et pas de contrainte
  - une classe peut être considérée comme instance d'une classe
  - indécidable
- Tout document OWL (Lite, DL ou Full) est un document RDF; tout document RDF est un document OWL Full; mais seuls certains documents RDF seront des documents OWL Lite ou OWL DL légaux

Constructeurs OWL

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} ⊔ {mary}
allValuesFrom	$\forall P.C$	∀hasChild.Doctor
someValuesFrom	$\exists P.C$	∃hasChild.Lawyer
maxCardinality	$\leqslant nP$	⩽1hasChild
minCardinality	$\geqslant nP$	⩾2hasChild

Namespace OWL

xmlns:owl ="http://www.w3.org/2002/07/owl#"

Axiomes OWL

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

Classes

- Les classes sont définis sous le type owl:Class
- Les hiérarchies des classes sont décris par rdfs:subClassOf
- owl:Thing est la racine de toutes les classes OWL
- Exemple:

```
<owl:Class rdf:ID="PotableLiquid"/>
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="#PotableLiquid"/>
</owl:Class>
<owl:Class rdf:ID="Grape"/>
<owl:Class rdf:ID="WineGrape">
    <rdfs:subClassOf rdf:resource="#Grape" />
</owl:Class>
```

Propriétés

- Les propriétés sont définies sous le type owl:ObjectProperty

- Exemple:

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
   <rdfs:domain rdf:resource="#Wine"/>
   <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

Caractéristiques de propriété (1)

- TransitiveProperty
  - P(x,y) et P(y,z) implique P(x,z)

```
<owl:TransitiveProperty rdf:ID="locatedIn">
    <rdfs:domain rdf:resource="&owl;Thing" />
    <rdfs:range rdf:resource="#Region" />
</owl:TransitiveProperty>
```

- SymmetricProperty
  - P(x,y) iff P(y,x)

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

Caractéristiques de propriété (2)

- FunctionalProperty
  - P(x,y) et P(x,z) implique y = z

```
<owl:FunctionalProperty rdf:ID="hasVintageYear">
    <rdfs:domain rdf:resource="#Vintage" />
    <rdfs:range rdf:resource="#VintageYear" />
</owl:FunctionalProperty>
```

- inverseOf
  - P1(x,y) iff P2(y,x)

```
<owl:FunctionalProperty rdf:ID="hasMaker"/>
<owl:ObjectProperty rdf:ID="producesWine">
    <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

Restrictions de propriété (1)

- allValuesFrom, someValuesFrom

```
<owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="#PotableLiquid" />
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasMaker" />
            <owl:allValuesFrom rdf:resource="#Winery" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- Pour tous les vins, s'ils ont des producteurs, tous les producteurs sont des caves vinicoles

Restrictions de propriété (2)

- cardinality, minCardinality, maxCardinality

```
<owl:Class rdf:ID="Vintage">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty
                rdf:resource="#hasVintageYear"/>
            <owl:cardinality
                    rdf:datatype="&xsd;nonNegativeInteger">
                    1</owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- owl:maxCardinality peut servir à fixer une borne supérieure et l'élément owl:minCardinality à fixer une borne inférieure

Restrictions de propriété (3)

- hasValue (OWL DL)

```
<owl:Class rdf:ID="Burgundy">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasSugar" />
            <owl:hasValue rdf:resource="#Dry" />
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- Tous les vins de la classe Burgundy sont des vins secs (i.e., leur propriété hasSugar doit avoir au moins une valeur égale à Dry)

Equivalences

- equivalentClass

```
<owl:Class rdf:ID="TexasThings">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#locatedIn" />
            <owl:someValuesFrom
                    rdf:resource="#TexasRegion" />
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

- equivalentProperty

Intersections

- Utilisation du constructeur *intersectionOf*

```
<owl:Class rdf:ID="WhiteWine">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Wine" />
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasColor" />
            <owl:hasValue rdf:resource="#White" />
        </owl:Restriction>
    </owl:intersectionOf>
</owl:Class>
```

Unions

- Utilisation du constructeur *unionOf*

```
<owl:Class rdf:ID="Fruit">
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#SweetFruit" />
        <owl:Class rdf:about="#NonSweetFruit" />
    </owl:unionOf>
</owl:Class>
```

- Remarquez comme cette construction de type union diffère complètement de la suivante

```
<owl:Class rdf:ID="Fruit">
    <rdfs:subClassOf rdf:resource="#SweetFruit" />
    <rdfs:subClassOf rdf:resource="#NonSweetFruit" />
</owl:Class>
```

Complémentaires

- Utilisation du constructeur *complementOf*

```
<owl:Class rdf:ID="NonFrenchWine">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Wine"/>
        <owl:Class>
            <owl:complementOf>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#locatedIn" />
                    <owl:hasValue rdf:resource="#FrenchRegion" />
                </owl:Restriction>
            </owl:complementOf>
        </owl:Class>
    </owl:intersectionOf>
</owl:Class>
```

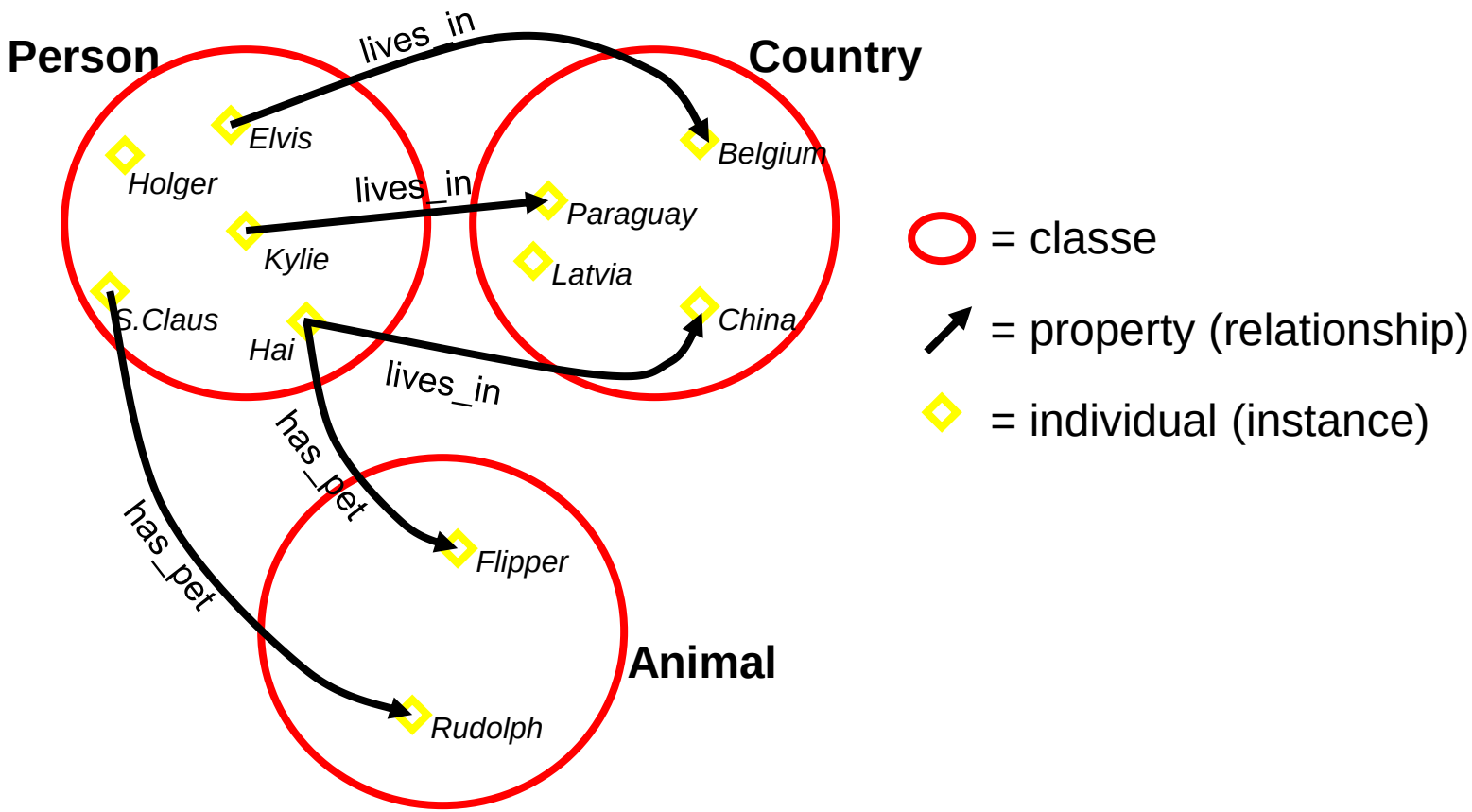
Classes énumérées

- Utilisation du constructeur *oneOf*

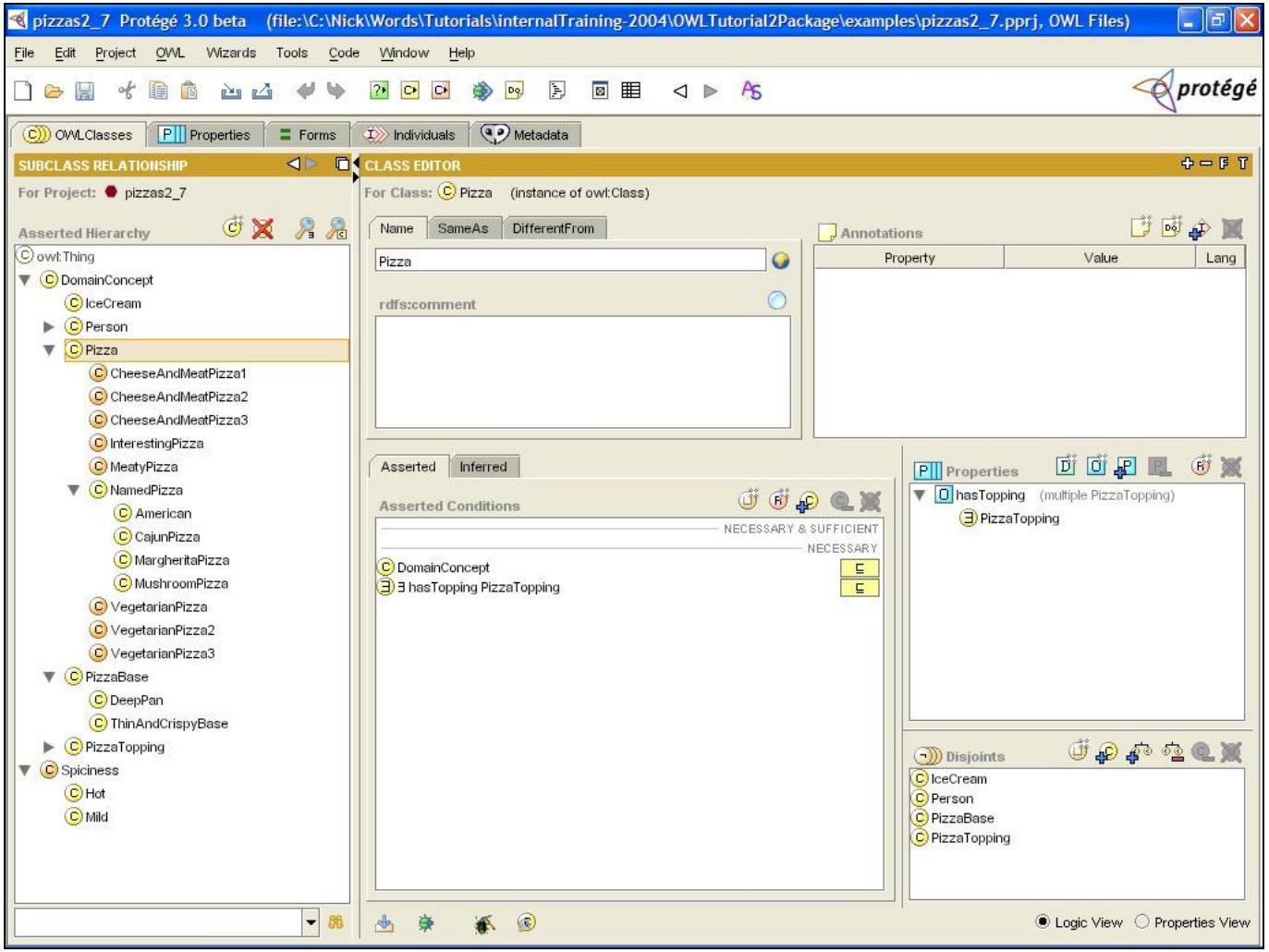
```
<owl:Class rdf:ID="WineColor">
    <rdfs:subClassOf rdf:resource="#WineDescriptor"/>
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#White"/>
        <owl:Thing rdf:about="#Rose"/>
        <owl:Thing rdf:about="#Red"/>
    </owl:oneOf>
</owl:Class>
```

Exercice

- Créer une ontologie en OWL pour le schéma suivant



Editeur Protége (1)

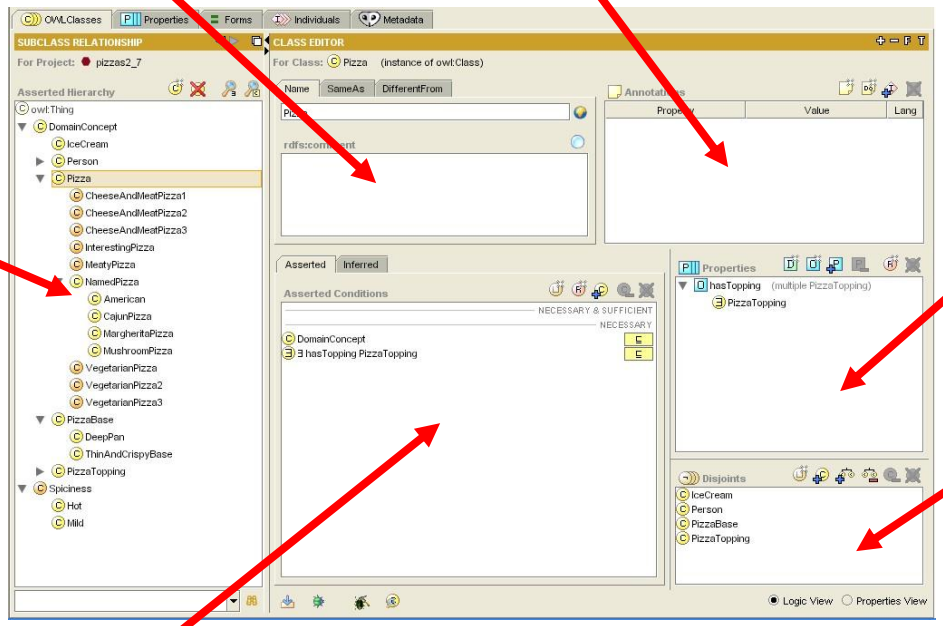


Editeur Protége (2)

Class annotations (for class metadata)

Class name and documentation

Class browser



Properties "available" to Class

Disjoints widget

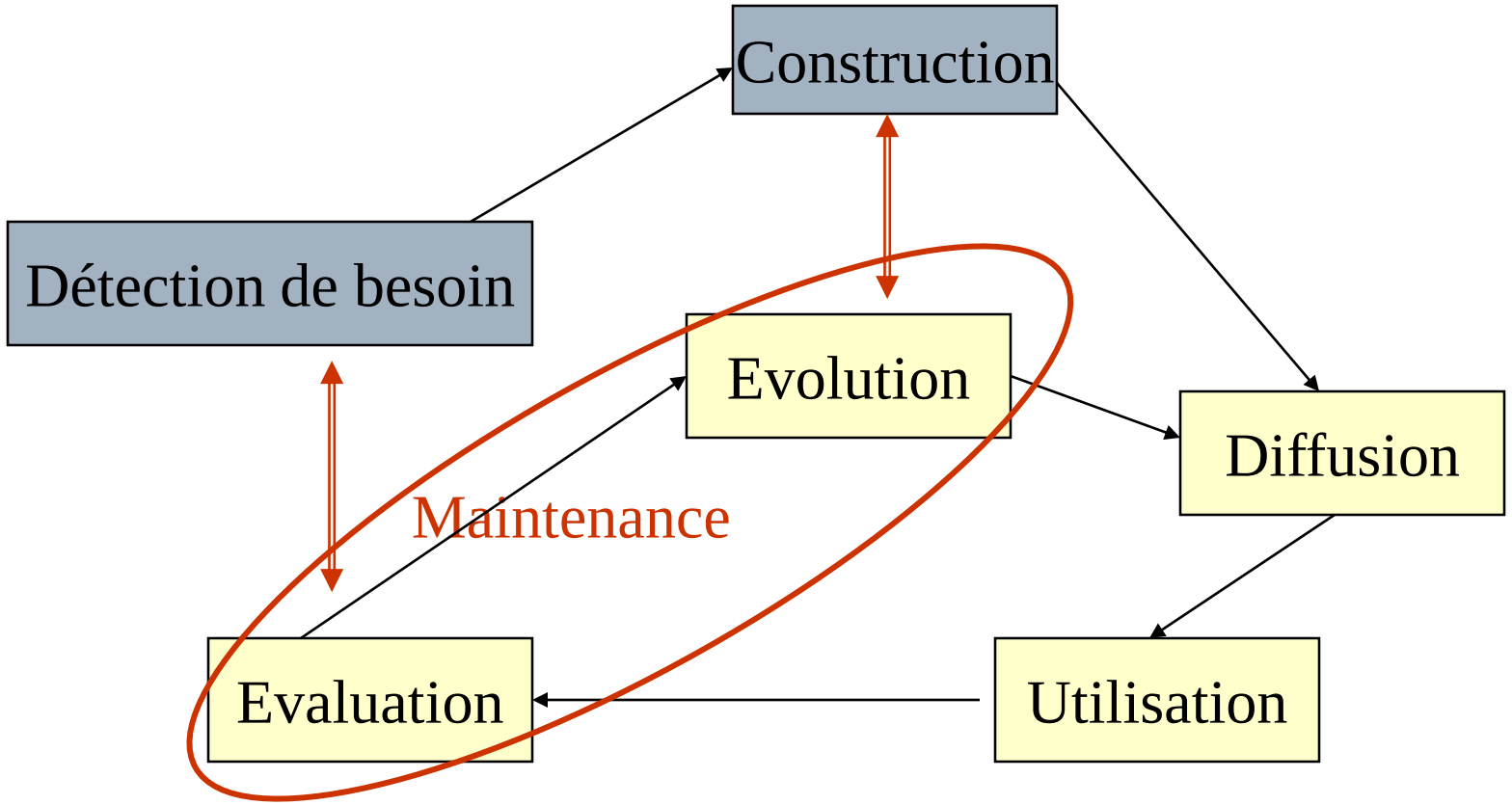
Conditions Widget

Class-specific tools (find usage etc)

Construction des ontologies

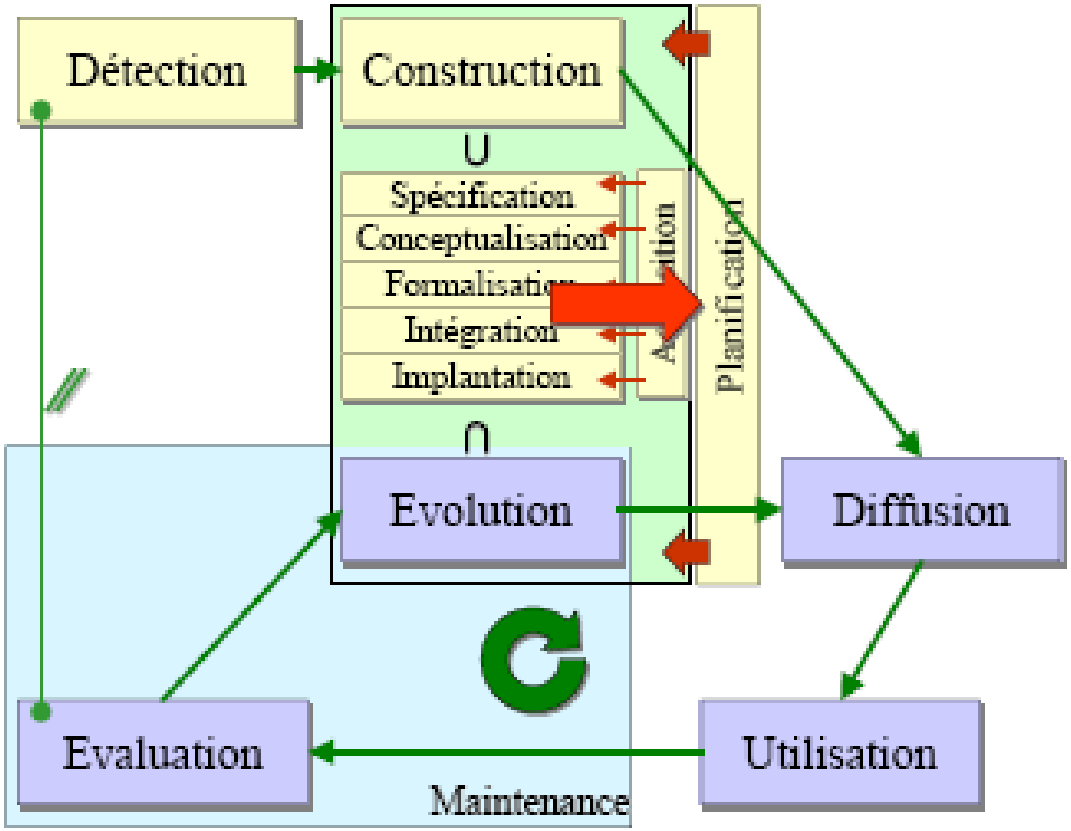
- Une ontologie ne se caractérise pas seulement par sa forme (structure d'ontologie) mais également par la manière d'en élaborer le contenu
  - Cycle de vie des ontologies
  - Des méthodologies pour construction

Cycle globale



Cycle détaillée

Cours de Fabien Gandon



Détection de besoin

- Problèmes dus à un manque de connaissances ou de consensus sur la signification de primitives utilisées pour représenter des connaissances
  - Vérifier la fréquence et les différentes occurrences du besoin
  - Décrire les scénarios problématiques avec les usagers
  - Vérifier que c'est du ressort de l'ontologie
  - Fait un état des lieux de départ pour bâtir une proposition sur ces constatations

Planification

- En informatique, lorsque l'on veut répondre à un problème, on ne commence pas par programmer.
- De même "on ne commence pas par implanter une ontologie"
- La réalisation d'une ontologie est un projet donc les notions de gestion de projet s'appliquent
  - temps & organisation des tâches
  - infrastructures, ressources, main d'oeoe uvre
  - contrôle du coût

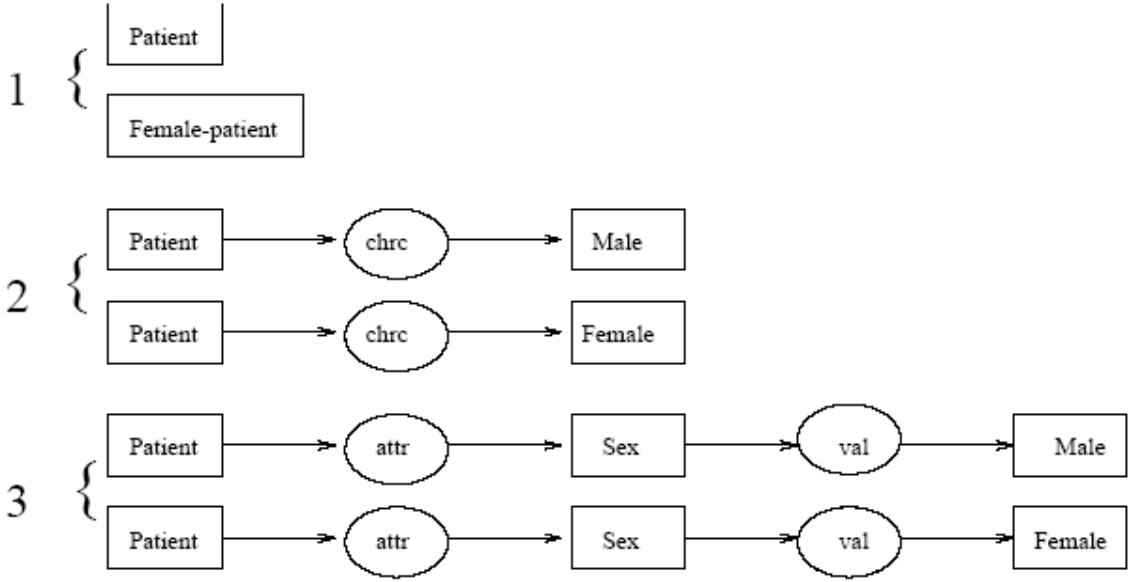
Spécification

- Analyse les objectifs et ambitions du besoin
- Exploiter les différentes ressources avec techniques appropriées
  - Réutilisation d'ontologies
  - Adaptation de terminologies
  - Analyse de données
  - Analyse automatique de documents
  - Entretiens avec les experts

Conception et structuration

- La conception de l'ontologie part systématiquement (actuellement) d'une expression linguistique des connaissances collectées
- La première structure de recueil utilisée après les texte libre est le lexique: liste des notions identifiées et des termes qui les dénotes
- Trois approches de structuration
  - Approche ascendante (bottom-up) : du plus spécifique vers le plus général exemple en programmation / focalisée application
  - Approche descendante (top-down) : du plus général vers le plus spécifique exemple en philosophie avec grandes théories
  - Approche centrifuge (middle-out) : chercher les concepts centraux et organiser des domaines autour

Exemple : le sexe est-il un attribut ?



1. Deux types de concept, *patient* et *patiente* sur lesquels le système ne peut rien « dire » sauf qu'ils sont différents ;
2. Un concept « primitif » (*patient*), deux caractéristiques différentes, différenciant ainsi les deux concepts « définis ».
3. idem 2e cas mais en plus la différence est explicitement liée au sexe puisque c'est une valeur de l'attribut.

Formalisation d'ontologies

- Implanter dans un langage formel les structures issues de la conceptualisation
- Degrés de formalisation
  - Informelle : en langue naturelle
  - Semi-informelle : langue naturelle restreinte, contrôlée, structurée
  - Semi-formelle: dans un langage artificiel définit formellement
  - Formelle: dans un langage avec une sémantique formelle, théorèmes (ex. logique)
- La formalisation ne consiste pas à *remplacer* une version informelle par un version formelle
- La formalisation s'applique à *augmenter* une version informelle avec les pendants formels des aspects pertinents à l'opérationnalisation (i.e., ceux qui sont exploités par les système décrits dans les scénarios)

Intégration & Réutilisation d'ontologies

- Objectifs
  - économiser du temps et des efforts en réutilisant des ontologies existantes
  - améliorer l'interopérabilité
- Réutilisabilité d'ontologies :
  - facilitée si les ontologies sont modulaires, bien délimitées, organisées en hiérarchies
  - facilitée si elles incluent les buts de leur conception et une documentation des choix faits dans sa réalisation
- Difficultés :
  - une ontologie est issue d'une conceptualisation qui reflète un point de vue et demande un réalignement
  - niveau de granularité différent selon les scénarios

Evaluation d’ontologies

- Vérification : évaluation technique et cohérence formelle des ontologies c.f. méthodes de vérification et formalisation
- Validation : évaluation de la réponse aux besoins ayant motivé les spécifications se fait avec les utilisateurs pendant des tests
- La complétude ne peut être prouvée elle doit se démontrer d'elle-même par l'utilisation
- Tout les manques donnent lieu à une nouvelle spécification de besoins dûment motivés par des scénarios applicatifs

Ressources bibliographiques

- Jos de Bruijn: Using Ontologies. Enabling Knowledge Sharing and Reuse on the Semantic Web. DERI Technical Report DERI-2003-10-29, 2003. http://www.deri.org/publications/techpapers/documents/DERI-TR-2003-10-29.pdf
- OWL Guide: http://www.w3.org/TR/owl-guide/
- OWL Reference: http://www.w3.org/TR/owl-ref/
- OWL Abstract syntax and Semantics: http://www.w3.org/TR/owl-semantics/

Projet mini

- A consulter les travaux pratiques sur le site http://www.cs.man.ac.uk/~horrocks/Teaching/cs646/

- Objectives du projet
  - Developer vous-meme une ontologie en OWL
  - Trouver une application au dela de cette ontologie (par exemple pour la recherche d'information)