UNIVERSITE DE LA ROCHELLE – MASTER 2 ICONE
2ère année – ***UE Recherche*** - Romain Raveaux

TP N°1 :

IMAGE ET ONTOLOGIE

1 REALISER UNE ONTOLOGIE POUR L'IMAGE AVEC PROTEGE

Les ontologies sont des outils de description capables de rendre compte de la connaissance *a priori* sur un problème donné (ex. Ontologie Médicale modélisant la connaissance du médecin sur l'interprétation des symptômes d'une population de patients).

Les processus de traitement d'image bénéficieraient grandement d'une logique de description capable de traduire le contenu sémantique d'un ensemble d'images [exemple. Images issu de Radiographies par rayon X]. Les ontologies permettraient alors de combler le fossé sémantique entre des amas de pixels inertes et une information porteuse de sens.

Votre premier travail sera de définir une ontologie décrivant la colorimétrique des objets graphiques ainsi que leurs formes à l'aide des caractéristique suivantes :

Caractéristiques Image :
- Caractéristique de forme :
  - La compacité : Pour chaque région de l'image. : C = 4π(area) / (perimeter)2.
  - L'eccentricité (e).
- Caractéristique couleurs :
  - Valeurs Rouge, Vert, Bleu. Valeurs réelles normalisées entre 0 et 1.

Compacité :
- Cercle : C = 1
- Carré : π / 4
- Une forme fine/étroite : 0

Eccentricité :
- Cercle : e=0
- Ellipse : e : entre 0.6 et 1.5

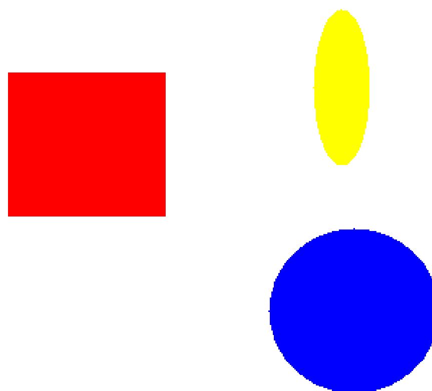

Figure 1 : Exemple d'image que l'ontologie pourra décrire

Les ontologies ne manipulent que des valeurs symboliques [des valeurs nominales], il est donc indispensable de quantifier les valeurs réelles de nos caractéristiques.

Valeurs réelles [0, 1] → {faible, moyen, fort}
Valeurs réelles vers un ensemble fini de valeurs, ceci est une discrétisation.
Valeur >0 mais <=0.3 → faible
Valeur >0.3 mais <=0.8 → moyen
Valeur >0.8 → fort

1°) Créer une classe (entity) « Quantification »
2°) Créer une « DataProperty » → « value » de type *double* appartenant à la classe « Quantification »
Domain : « Quantification »
Range : « value »
3°) Spécialiser la classe « Quantification » entre trois sous-classes : « faible », « moyen », « fort».
4°) Formuler les « DataRestriction»
Cliquer que « moyen », cliquer sur « équivalent classes », dans le « class expression editor » rentrer « valeur some double[> 0.3, <= 0.8] »
5°) Même principe mais pour les autres classes….

On considère que chaque région a une information de type forme et une information de type couleur.
8°) Créer une entité « Color »
9°) Créer trois « objectproperty » hasBlueLevel, hasRedLevel, hasGreenLevel
Domain : Color
Range : Quantification
10°) Spécialiser la classe « Color » en 4 sous-classes.
Yellow,Green,Blue,Red.
11°) Exprimer les contraintes de ces sous-classes :
Exemple pour Yellow : Cliquer sur « equivalent classes », et taper les lignes suivantes dans l'editeur :
hasBlueLevel some Low
and hasGreenLevel some High
and hasRedLevel some High
12°) Même manipulation pour les autres couleurs…..
13°) Créer la classe « Shape ».
14°) Créer trois « objectproperty » hasCompacity, hasEccentricity. Rempliser les range et le domain.
15°) Décliner la classe « Shape » en deux sous classes « Rectangle » et « Ellipse »
16°) Intégrer les contraintes suivantes sur la classe Ellipse :
(hasEccentricity some Low)
or (hasEccentricity some Medium)
and hasCompactness some High
17°) Intégrer les contraintes suivantes sur la classe Rectangle :
18°) Intégrer les contraintes suivantes sur la classe Ellipse :
(hasEccentricity some Low)
or (hasEccentricity some Medium)
and hasCompactness some Medium
19°) Créer une entité « Region »
20°) Créer les « objectproperties » : « hasColor » et « hasShape »
Domain : xxxx
Range : xxxxx

Voilà, la première version de l'ontologie est construite (Un vocabulaire + un ensemble de régles)………

2 OWL API. Utilisation de notre ontologie pour le traitement d'image

L'enseignant vous donnera un « jar » réalisant le traitement d'image, càd, segmentation couleur et extraction des caractéristiques graphiques.

Créer un projet sous votre IDE favori, et intégrer les « lib »……

//// Getting Started ImageProcessing.//

```
SMRSeg SMR = new SMRSeg(imsource);
SMR.WriteSegImage("./OWL/seg.tif");
SMR.RML.WriteSegmentedImageTrueColour("./OWL/segRML.tif");
SMR.WriteLabelImage("./OWL/lab.tif");
ArrayList<RegionLight> ListOfRegions = SMR.getListOfRegions();
```

Chaque RegionLight contient les méthodes suivantes :

```
RegionLight R ;
double e = R.ComputeEccentricity() ;
double c = R.ComputetCompacity() ;
double r R.ComputeRedNorma() ;
double g =R.ComputeGreenNorma() ;
double b =R.ComputeBlueNorma() ;
```

Travail à faire : « sur-coucher » OWL-Api en créant un package « OWL_API_Pkg »
Dans ce package vous allez créer les fonctionnalités suivantes :

1°) Charger et sauvegarder une Ontologie.
2°) Image vers OWL.
  Region vers OWL
    Créer des Individus sans type (on ne préciser pas la classe des individus).
    Chaque individu a un « hasColor » et un « hasShape »
3°) Raisonner sur cette Ontologie générée afin de t yper automatiquement les individus.

/////// OWL API Getting Started ////////////

```
public void Loader(URI owlModel, String nameSpace, String FileName) throws
OWLOntologyCreationException{
              this.NameSpace = nameSpace;
              this.FileName = FileName ;
            manager = OWLManager.createOWLOntologyManager();
              ontology =
manager.loadOntologyFromPhysicalURI(URI.create(owlModel.toString()));
factory = manager.getOWLDataFactory();
      }


public void Saver(String FileName) throws UnknownOWLOntologyException,
OWLOntologyStorageException {
            manager.saveOntology(ontology, new
OWLFunctionalSyntaxOntologyFormat(), URI.create(new
File(FileName).toURI().toString()));

      }
```

Créer un individu « vide » (R est de type RegionLight)

```
	OWLIndividual ind_region = 
factory.getOWLIndividual(URI.create(this.NameSpace+R.Id));
		OWLAxiom axiom_region = 
factory.getOWLDeclarationAxiom(ind_region);
		AddAxiom addAxiomRegion = new AddAxiom(ontology, axiom_region);
		manager.applyChange(addAxiomRegion);
```

Créer et Ajouter une DataProperty à un individu : r est un double dans cet exemple. La valeur r est ajouté à l'individu « ind_r »

```
OWLDataProperty value = 
factory.getOWLDataProperty(URI.create(this.NameSpace+"ImageOntology.owl#val
ue"));


		OWLIndividual ind_r = 
factory.getOWLIndividual(URI.create(this.NameSpace+r));
		OWLAxiom axiom_r = factory.getOWLDeclarationAxiom(ind_r);

		AddAxiom addAxiom_r = new AddAxiom(ontology, axiom_r);
		manager.applyChange(addAxiom_r);
		OWLAxiom axiom_r_real = 
factory.getOWLDataPropertyAssertionAxiom(ind_r, value, r);
		AddAxiom addAxiom_r_real = new AddAxiom(ontology, 
axiom_r_real);
		manager.applyChange(addAxiom_r_real);
```

Créer et Ajouter un ObjectProperty à un individu : r est un double dans cet exemple. L'objet RedObject est ajouté à l'individu « ind_color »

```
OWLObjectProperty RedOject = 
factory.getOWLObjectProperty(URI.create(NameSpace+"ImageOntology.owl#hasRed
Level"));

OWLIndividual ind_color = 
factory.getOWLIndividual(URI.create(this.NameSpace+R.Id+"color"));

		OWLAxiom axiom_color = 
factory.getOWLDeclarationAxiom(ind_color);
		AddAxiom addAxiomcolor = new AddAxiom(ontology, axiom_color);
		manager.applyChange(addAxiomcolor);



		OWLAxiom axiom_RedColor = 
factory.getOWLObjectPropertyAssertionAxiom(ind_color, RedOject, ind_r);
		AddAxiom addaxiom_RedColor = new AddAxiom(ontology, 
axiom_RedColor);
		manager.applyChange(addaxiom_RedColor);
```