Modélisation de Connaissance et Architectures Logicielles (2eme Partie)

Thèmes du Cours:

- A) Introduction à l'approche logique pour la modélisation des connaissance:
  - Notions de base de la Logique du Premier Ordre: Clauses Logiques (règles, faits), Unification, Résolution Logique (Inférence).
- B) Introduction à la Logique de Description.
- C) Ontologies: Applications dans la Web Sémantique. Inférence sur les Ontologies, Langages de représentation basés sur la combinaison du paradigme Logique et Objets. Présentation de quelques outils logiciels d'édition et d'inférence sur les Ontologies.

A) Logique du Premier Ordre

- Issue de la Logique Mathématique (basée sur: les Clauses logiques de Horn + l'algorithme de l'unification de Robinson (1956) + Le Principe de Résolution.
- Programmation Déclarative.
- Programmation Non-Deterministe.
- Outil de Modélisation des Connaissances et de Programmation de très haut niveau.
- Applications privilégiées: I.A, Ingénierie de Connaissance,…
- Application avec CLP: Planification, Gestion des Ressources…

Déclarativité (Exemple Introductif)

- Déclaration des connaissances sous forme de faits et de règles:
- Exemple:

```
/* derivee(Expression, X, Resultat) */
derivee(X, X, 1).
derivee(sin(X), X, cos(X)).
derivee(cos(X), X, -sin(X)).
……..
derivee(F + G, X, DF + DG):-
    derivee(F, X, DF), derivee(G, X, DG).
derivee(F * G, X, F * DG + DF * G):-
    derivee(F, X, DF), derivee(G, X, DG).
……..
```

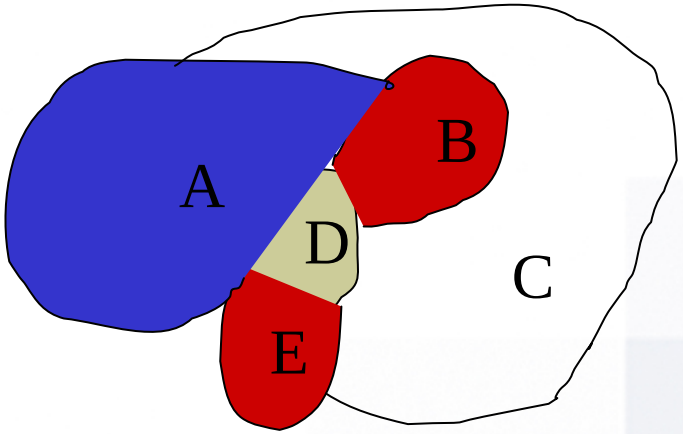
Non Déterminisme (Exemple Introductif) (1)

- Exemple introductif: Problème du Coloriage des cartes avec 4 couleurs.

- Pb: Colorier les régions d'une carte géographique avec 4 couleurs de telle sorte que deux régions voisines n'aient pas la même couleur.

Non Déterminisme (2)

- Exemple d’une carte:



Solution

```
couleur(blanc).
couleur(vert).
couleur(rouge).
couleur(bleu).
colorier(A,B,C,D,E) :- couleur(A), couleur(B), couleur(C),
    couleur(D), couleur(E), not(A,B), not(A,C), not(A,D),
    not(A,E), not(B,C), not(B,D), not(D,E).
```

La requête:

```
-? colorier(A,B,C,D,E).
```

calcule toutes les solutions du problème.

Concepts de base (1)

- 1) Termes de la Logique du Premier Ordre
  - Alphabet de variables V = {A, B, X, Toto, Xyy …}
  - Alphabet de fonctions F = {a, b, x, toto, cDD …}
  - RQ) Par convention: les variables sont des symboles qui commencent par des lettres majuscules, et les fonctions par des lettres minuscules.
  - L'ensemble des *termes* T(V, F) de la logique du premier ordre est défini par:
    - Une variable de V est un terme
    - Si t_1, …, t_n, n >= 0, sont des termes, et f est une fonction d'arité n, alors f(t_1, …, t_n) est un terme de T(V, F).
  - Les termes d'arité 0, f(), sont des *Constantes*. f() est noté f.

Concepts de base (2)

- Exemples de termes:
  - mere(luc, marie).
  - pere(luc, jean).
  - masculin(luc).
  - feminin(marie).
  - plus(X, 0, X).
  - arbraBinaire(a, arbraBinaire(b , nil , nil), nil).
  - personne(luc, age(25), ville(rouen)).

Concepts de base (3)

- Clauses de Horn
  - Une clause de Horn C a la forme:

    A :- B1, … , Bn.

    où A, et les Bi sont des termes de T(V,F), n >= 0.
  - **Sémantique (Déclarative):**

    Pour toutes les variables X1, …, Xk de C, A est vrai si B1, et B2, … et Bn sont vrais.
  - RQ): Si n = 0, C est un *Fait* (noté A.), sinon C est une *Règle*.

Concepts de base (4)

- Exemples:

```
/* Y est parent de X */
parent(X, Y) :- pere(X , Y).
parent(X, Y) :- mere(X, Y).
/* F est fille de P */
fille(F, P) :- feminin(F), parent(F, P).
/* X1 est frère de X2 */
frere(X1, X2) :- masculin(X1),
                 parent(X1, Y),
                 parent(X2, Y).
```

Concepts de base (5)

- Programme Logique
  - Un programme Logique P est une suite de Clauses de Horn
  - Exemple
    - mere(luc, marie).
    - pere(luc, jean).
    - masculin(luc).
    - masculin(jean).
    - feminin(marie).
    - pere(pierre, jean).
    - masculin(pierre).
    - + Les 4 règles précédentes à insérer ici.

Concepts de base (6)

- But
  - Un but B est une conjonction de m termes, m >= 0:

    t1, … , tm.
  - **Sémantique (Déclarative):**

    Existe-t-il des valeurs associées aux variables X1, …, Xk de B, t.q B soit une *Conséquence Logique* du Programme Logique P ?

Concepts de base (7)

- Exemples de buts:
  - ?- pere(luc, jean).
  - == Yes
  - ?- mere(luc, marie).
  - == Yes
  - ?- pere(luc, X). (RQ: X est une Entité à Connaître).
  - == {X = jean}.
  - ?- pere(X, jean).
  - == {X = luc}.
  - == {X = pierre}.
  - ?- pere(luc, X), pere(Y, X).
  - == {X = jean, Y = pierre}
  - == {X = jean, Y = luc}

Concepts de base (7)

- Exemples de buts:
  - ?- pere(luc, jean).
  - == Yes
  - ?- mere(luc, marie).
  - == Yes
  - ?- pere(luc, X). (RQ: X est une Entité à Connaître).
  - == {X = jean}.
  - ?- pere(X, jean).
  - == {X = luc}.
  - == {X = pierre}.
  - ?- pere(luc, X), pere(Y, X).
  - == {X = jean, Y = pierre}
  - == {X = jean, Y = luc}

Exercices

1) Poursuivre le programme logique qui définit les relations familiales en introduisant les notions de: Oncle, Tante, Cousin, Grand Parent, Ancêtre… Avec une base de faits plus riche.
2) Ecrire un Programme logique qui teste l'existence d'un chemin entre 2 nœuds d'un graphe.

Listes

- Les *listes* peuvent être vues comme des termes de la manière suivante:
  - La constante NIL désigne la liste vide, notée **[]**.
  - La liste de termes t_1, …, t_n peut être vue comme un terme liste(t_1, liste(t_2, …, liste(t_n , NIL)…) notée par **[t_1, …, t_n]**.
  - La notation [X1 | Xs] désigne une liste dont X1 est le premier élément, et Xs est le reste, i.e. la **liste** t_2, …, t_n.
  - Le prédicat *liste* peut être définit par le programme suivant:

    liste(nil).

    liste(X , Xs) :- liste(Xs).

Exercices

- Ecrire un programme qui:
- 1) Calcule la longueur d'une liste.
- 2) teste si P est un préfixe de la liste L.
- 3) teste si S est un suffixe de la liste L.
- 4) teste si F est un Facteur de la liste L.
- 5) calcule toutes les permutations P d'une liste L.
- 6) teste si un mot M est reconnu par un automate X.

Concepts de base (7)

- Substitution
  - Une *substitution* α est une application de V dans T(V,F), noté: $\{(X_1, T_1), \ldots (X_n, T_n)\}$ où X_i est une variable et T_i (l'*image* de la variable X_i) est un terme.
  - Par convention, X_i n'a pas d'occurrence dans aucun T_j. Sinon, il suffit de le remplacer par son image T_i.
  - Ex: V = {U, V, X, Y, Z}, F = {a, b, f, g}

    α = {(X, f(a)), (Y, g(b, Z)), (U, f(a))}.

    On a implicitement (V,V) et (Z,Z)

Substitution (suite)

- L'application de α à un terme T est le terme, noté Tα, obtenu en remplaçant chaque variable dans T par son image.
- L'identité, notée *Id*, est l'élément neutre de l'ensemble de toutes les substitutions.
- Exemple:
  - T = g(X, f(Y), Z).
  - α = {(X, f(a)), (Y, g(b, Z))}.
  - Tα = g(f(a), f(g(b, Z)), Z).

Unification

- Problème de l'unification:

1) Soient T_1 et T_2 deux termes. Existe-t-il une substitution α t.q $T_1\alpha == T_2\alpha$? (α est dit *unificateur).*
2) Comment représenter tous les unificateurs dans ce cas ?

- Exemple:

  T_1 = f(X, g(Y), T), T2 = f(f(a,b), g(g(a,Z)), Z).

  α_1= {(X, f(a,b)), (Y, g(a,a)), (T,a), (Z,a)}.

  α_2= {(X, f(a,b)), (Y, g(a,Z)), (T,Z)}.

Unification (Suite)

- Une substitution α est *plus générale* qu'une substitution β s'il existe une substitution γ t.q.

$$\beta = \alpha\gamma$$

- Exemple (suite): α_2 est *plus générale* que α_1 car il existe $\gamma = \{(T, a) , (Z,a)\}$ t.q. $\alpha_1 = \alpha_2\gamma$.
- TH: Lorsque deux termes T1 et T2 de T(V,F) sont unifiables, il existe un seul unificateur *l'unificateur le plus général* u.p.g, (modulo le changement de noms de variables).

Algorithme de l'Unification

- Entrée: Deux termes T1 et T2
- Sortie: Non si T1 et T2 ne sont pas unifiables, et α (l'unificateur le plus générale) sinon.
- Approche:

  On va distinguer 3 cas pour T1 (et T2), où

  - T1 est une constante C1,
  - T1 est une variable V1 et
  - T1 est un terme arborescent A1 d'arité au moins 1.

Algorithme

	C2	X2	A2
C1			
X1			
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2		
X1			
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2	Succès avec {(X2,C1)}	
X1			
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2	Succès avec {(X2,C1)}	Echec
X1			
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2	Succès avec {(X2,C1)}	Echec
X1	Succès avec {(X1,C2)}	Succès avec {(X1,X2)}	
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2	Succès avec {(X2,C1)}	Echec
X1	Succès avec {(X1,C2)}	Succès avec {(X1,X2)}	Succès si A2 n'a pas d'occurrence de X1, avec {(X1, A2)}, Echec sinon
A1			

Algorithme

	C2	X2	A2
C1	Succès si C1==C2	Succès avec {(X2,C1)}	Echec
X1	Succès avec {(X1,C2)}	Succès avec {(X1,X2)}	Succès si A2 n'a pas d'occurrence de X1, avec {(X1, A2)}, Echec sinon
A1	Echec	Cas analogue à X1 avec A2	Cf. Cas de 2 termes arborescents

Unification de A1 et A2

- Succès s'ils ont la même racine, et si forêt(A1) est *unifiable* avec forêt(A2). Echec sinon.
- 2 forêts F1 = *f1, f2, … , fn* et G1 = *g1, g2, …, gm* sont *unifiables* si
  - n = m;
  - f1 et g1 sont unifiables avec α, et
  - les deux forêts f2α, … , fnα, et g2α, …, gmα sont unifiables

Unification et résolution de contraintes

- En programmation logique pure on s'intéresse uniquement aux termes de T(V,F), et l'unification de ces termes.
- En programmation avec contraintes, on s'intéresse aussi à d'autres types de termes et à leurs unifications (résolution de contraintes).
- Ex: Les polynômes linéaires de la forme:

$$a_1z_1+a_2z_2+ ...+a_nz_n+a_{n+1}$$

où z_1, z_2, ..., z_n sont des variables dans N, et les a_i sont des nombres naturels.

Exs: $2x_1$, $x_2 + x_3$, $7x_3 + 9x_4 + x_5 + 77$

Unification et résolution de contraintes (suite)

- L'unification: 2x1 = x2 + x3 dans N est donnée par la *solution générale* représentée par deux solutions *paramétriques* :

  1) x1 --> y1 + y2, x2 --> 2y1, x3 --> 2y

  2) x1 --> y1 + y2 +1, x2 --> 2y1+1, x3 --> 2y+1

- La *solution générale* est obtenue en donnant à y1, y2 (variables libres) toutes les nombres naturels possibles.

Principe de Résolution (un aperçu)

- C'est une méthode d'inférence logique, qui permet (à l'image de *Modus Ponens*) de déduire de nouvelles connaissances.
- Entrée: Un Programme Logique P, un but B.
- Sortie:
  - Une *solution* (si B est une *conséquence logique* de P).
  - Un calcul infini ou Non, si B n'est pas une solution de P.
- Pb. Semi-décidable

Principe de Résolution(2)

- Approche:

  C’est un processus arborescent. On part de (B, Id), où B = b1, …, bn.

  A chaque étape du processus on a un nœud (D, α), où D est un nouveau but, et α est l’unificateur courant.

Principe de Résolution(3)

- Si D est le but vide, α est alors une solution.

- Pour obtenir les fils de (D, α), on considère une *copie* de toutes les clauses qui ont une tête unifiable avec un terme quelconque *bi* de B par l'unificateur α'. Si de telles clauses n'existent pas, on dit que la branche aboutit à l'*échec*.

Principe de Résolution(4)

Pour chaque clause C, on ajoute un nouveau fils (D', $\alpha\alpha$') de (D, α), ou D' est obtenu comme suit:

On remplace dans D le terme choisi *bi* par le corps de C. On obtient D1.

On a alors: D' = D1α'.

Exercice

parent(jean, marie).

parent(marie, luc).

ancêtre(X, Y) :- parent(X, Y).

ancêtre(X, Y) :- parent(X, Z), ancêtre(Z, Y).

Calculer l'arbre de résolution associé au but:

?- ancêtre(X, luc).

Meta-Interprète

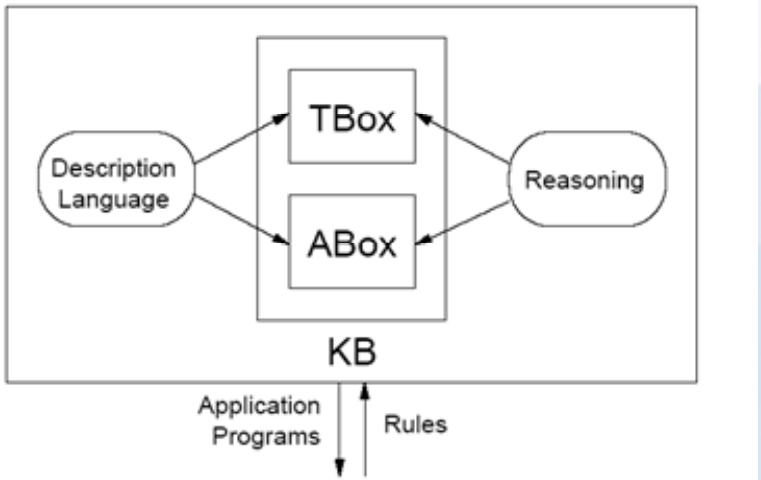
- solve([]).
- solve([X | Xs]) :- solve(X), solve(Xs).
- solve(A) :- clauses(A, Cs), solve(Cs).

Logique du premier ordre (Conclusion)

- Un paradigme majeur en modélisation des connaissances. Cependant:
  - Insuffisant pour la représentation des structures persistantes.
  - Problèmes liés à la semi-décidabilité.

B) La Logique de Description (1)

- Besoin d'une représentation logique des connaissances structurelles (classe d'objets, rôles, relations entre classes…)
- Besoin d'un langage logique plus restrictif (un fragment de la logique du premier ordre) mais décidable.
- Modélisation par T-Box (terminologie pour le description du domaine), et A-Box (assertions sur les objets du domaine):



La Logique de Description (2)

- La T-Box contient des *concepts atomique* (termes d'arité 1), *rôles* (termes d'arité 2), des concepts composites (formules non atomiques):

$$
\begin{array}{lll}
C, D & \longrightarrow & A \mid \quad \text{(atomic concept)} \\
 & & \top \mid \quad \text{(universal concept)} \\
 & & \bot \mid \quad \text{(bottom concept)} \\
 & & \neg A \mid \quad \text{(atomic negation)} \\
 & & C \sqcap D \mid \quad \text{(intersection)} \\
 & & \forall R.C \mid \quad \text{(value restriction)} \\
 & & \exists R.\top \quad \text{(limited existential quantification).}
\end{array}
$$

La Logique de Description (3): Exemple de concepts

- Exemple de concepts:
  - Person, Female: 2 concepts atomiques
  - hasChild: role
  - (Person $\sqcap$ Female) concepts pour designer les personnes du sex féminin.
  - (Person $\sqcap \neg$Female)
  - (Person $\sqcap \exists$ hasChild.$\top$) pour designer toutes les personnes ayant un enfant.
  - (Person $\sqcap \forall$hasChild.Female), pour designer toutes les personnes ayant un enfant du sexe féminin.
  - (Person $\sqcap \forall$hasChild.$\top$) pour designer les personnes sans enfants

La Logique de Description (4): Exemple d'une T-Box

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists \text{hasChild}.\text{Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists \text{hasChild}.\text{Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists \text{hasChild}.\text{Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall \text{hasChild}.\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists \text{hasHusband}.\text{Man}
\end{aligned}
$$

La Logique de Description (5):

- Exemple d'une A-Box:

MotherWithoutDaughter(MARY)
hasChild(MARY, PETER)
hasChild(MARY, PAUL)

Father(PETER)
hasChild(PETER, HARRY)

- Exemple d'inférence en LD:
  - Un concept est-il satisfiable?
  - Un concepts est-il plus général qu'un autre?
  - ….

C) Ontologies (Introduction)

- Definitions
  - *Philo*: La Nature de la Connaissance
  - *Info*: Une Spécification d'une Conceptualisation Partagée
  - *Usage*: Construire une ontologie signifie: Modéliser le domaine, ses concepts, ses objets, ses relations ses structures et ses propriétés essentielles, donner le sens des termes… dans un langage standard (XML, RDF…) qui lui permet la communicabilité à travers le WEB et les applications informatiques.

Ontologies (suite)

- Objectif: Faciliter la réutilisation et le partage des connaissances.
- Approche: Apporter des informations sémantiques accessibles à l'ordinateur et communicable aux agents humains et logiciels.
- Applications: Ingénierie, I.A, Web…

Exemple : Web Sémantique

- Les moteurs de recherche sur le Web sont basés sur une recherche syntaxique.
- Ex1: Chercher sur Google:

  Dupont, thèmes de recherches

  (Mauvaises réponses)
- Ex2: Comment poser des questions telles:
- Quels sont tous les chercheurs travaillant sur le Web Sémantique? (Impossible actuellement)
- Semantic Web: *an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*.

Comment procéder ? (1)

1) Il faut une représentation de l'ontologie du domaine (les termes, leurs sens, tous les concepts: personne, chercheur, thèmes…) dans un langage de représentation d'ontologies

Ex: Class: Thème-de-recherche

Attributs:

Nom, Description, Approche, sous-thèmes

Bibliographies, Thèmes-liés….

Les langages de représentation d'ontologies combinent généralement le paradigme Objet, et le paradigme logique.

Comment procéder ? (2)

2) Il faut une annotation des documents utilisateurs respectant l'ontologie, et accessible aux applications logicielles.

Standards Actuels: XML, RDF, RDFs…

- XML: Langage de représentation des documents avec:
  - marqueurs (tags) associés aux mots-clés,
  - Grammaire du document (DTD),
  - Séparation du fond et de la forme (XSL)...

Comment procéder ? (2, Suite)

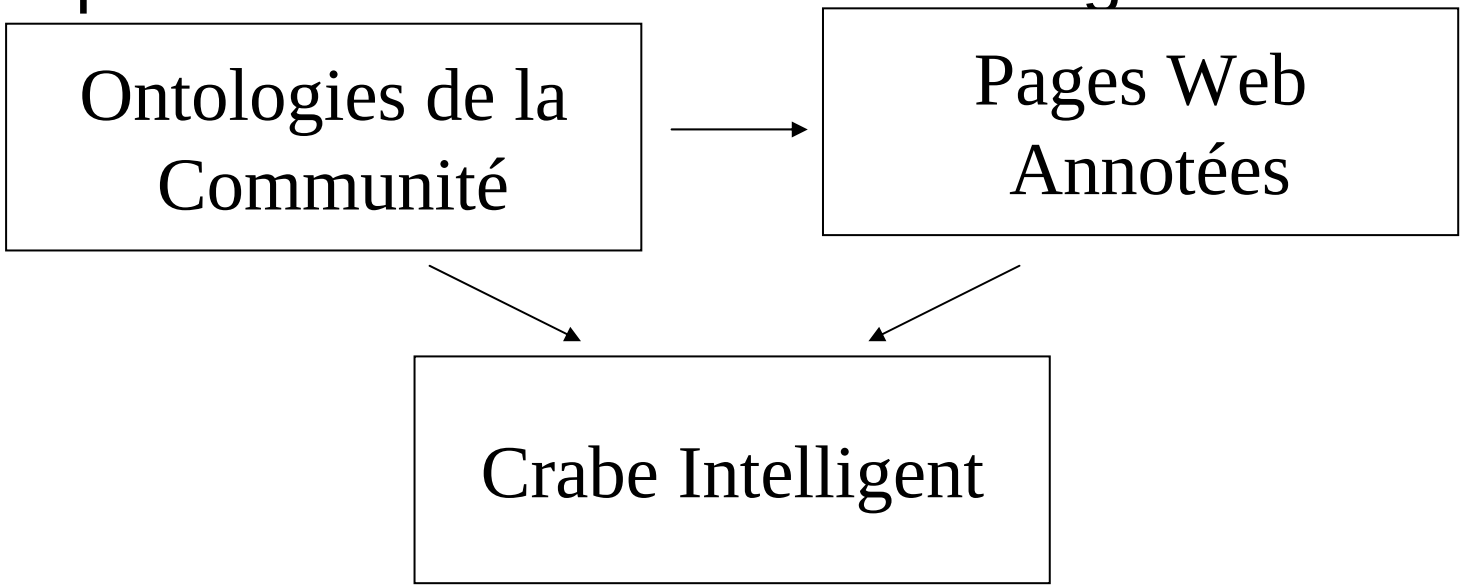
- RDF: une extension de XML, avec:
  - Le triplet (Sujet, Predicat, Objet)
  - Ex: (http//… Auteur Dupont).
- RDFs: une extension de RDF avec
  - Des classes associées à: Ressources, Property type, Object
  - InstancesOf, SubClassOf…
- DAML+OIL au dessus de XML, inspiré du formalisme de la Logique de Description

Comment procéder ? (3)

- 3) Il faut un langage de requêtes, permettant le lien entre les documents et les ontologies, l'extraction des connaissances de haut niveau de ces documents à partir des axiomes et des règles prédéfinies.
- 4) Des outils de collections automatiques de documents (agents…), d'analyse et d'indexation des documents…

L'approche de (KA)2 (1)

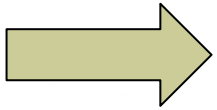
- (KA)2: Knowledge Annot. Of the Knowledge Acquisation Community: http://ontobroker.semanticweb.org/



L'approche de (KA)2 (2)

- L'ontologie de (KA)2 est divisée en 8 sous-otologies: Organisation, Personnes, Projets…
- Elle est formalisée en F-Logic
- C'est un formalisme qui combine PPO et PL:
- C1 :: C2 (C1 est une sous-classe de C2)
- C [a == > r] (C: Class, a: attribut, r: type)
- O : C [a == >> v] (O objet de la classe C. v est la valeur de a)
- …….

Approche générale de gestion d'ontologies

- Modélisation du domaine (UML, Systèmes Spécifique de Modélisation Ontologiques: Protégée2000, Oiled, DL-Workbench…)
-  Génération du Code
- Modélisation des Connaissances par des clauses logiques (SiLRi, Fact, Triple…).
- Puis, Interrogation.

Outil d'édition d'ontologie en formalisme DL

- **DL-workbench : A meta-model based platform for ontology manipulation.**
- **http://www.opencascade.org/dl-workbench/**
- **Un système d'édition d'ontologie, utilisant le formalisme DL**
- **Système ouvert (Plug-in au-dessus de Eclipse, avec Méta-model)**