ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Introduction to Protégé

Federico Chesani

Department of Computer Science and Engineering
University of Bologna

Ontologies

An ontology is a formal, explicit description of a domain of interest

Allows to specify:

- Classes (domain concepts)
- Semantic relation between classes/properties associated to a concept (slots, and possibly restrictions, facets)
- Possibly, a further logic level (axioms and rules)

- Classes instances

Ontology + Istances = Knowledge Base

Ontologies

- Starting point: a deep analysis, complete(?) and correct(?) of an application domain.

- The knowledge can be then represented/formalized by means of an ontology

- Uses:
  - Export
    - Application export
    - Knowledge export
  - Modelling
    - Fundamental in every step of teh developing process
  - Interoperability between different applications

Ontologis and Semantic Web

Two proposals within the SW Initiative:

- RDF Schema (RDFS), RDF extensions with proper terms for ontological concepts

- OWL (Ontology Web Language), extend RDFS
  - OWL Lite
  - OWL DL
  - OWL Full

  Based on Description Logics

OWL and Description Logic

- Description Logics are a family of logics
- Each fragment depend on which operators are supported in the logic
- More supported operators→ higher the complexity
- OWL-DL supports the following operators:

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	$\top \sqsubseteq \leqslant 1 P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1 P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

Protégé – An ontology editor

- Many different editors:
  - WebODE, http://webode.dia.fi.upm.es/WebODEWeb/index.html
  - ICOM, http://www.inf.unibz.it/~franconi/icom/
  - DOME, http://dome.sourceforge.net/
  - … (many many others)

  - Protégé, http://protege.stanford.edu/
    - Developed at Stanford (US)
    - Open Source, java based, extendible
    - Plug-in architecture, lot of plug-ins available
    - Exports ontology in many formats (rdfs, owl)

Developing an ontology

A possible development process:

1. Analyse the **domain** and the **goal** of the ontology
2. Consider to **reuse** existing ontologies
3. Determine the key **concepts** of the domain
4. Organize concepts in classes and **hierarchies** among classes
5. Determine the **properties** of the classes
6. Add **constraints** (allowed values) on the properties
7. Create the **instances**
8. Assigns **values** to the properties for each instance

Protégé – Developing an ontology

Editing function are split in different tabs

- Tab “Classes”: classes editor
- Tab “Object Properties”
- Tab “Data Properties”
- Tab “Individuals”

- Tab “Forms”: Allows to define forms for the data insertion phase

Protégé – Classes editor

- Assertions
  - **Necessary and Sufficient** (complete definition) (C1 equivalent to C2)
  - **Necessary** (partial definition) (C1 has superclass C2)
  - Inherited
- Assertions are about Properties and Restriction on properties

- Disjoints (individuals of this class cannot be also individuals of another class)
  - By default, OWL classes have some “overlap”!!!!!!

Protégé – Properties

Properties are binary relations between two things.

- Object properties: relation between two individuals
- Datatype properties: relation between an individual and a primitive data type

- Annotation properties (metadata…)

Protégé – Inverse properties

For each object property it is possible to specify the inverse property

E.g.:

individuals: federico, francesco

Properties: hasParent, hasChild

Sentence: “francesco hasParent federico”

Now, define hasChild as inverse property of hasParent

→ It is possible to automatically infer that : “federico hasChild francesco”

Functional Properties

- Functional Properties: a property is functional if, for a given individual, it can be in relation (through such property) with only another individual

- E.g.: “francesco hasFather federico”

- Note: the fact that a property is functional is not used as a constraint, but as axiom for the inference.
- E.g., if we add “francesco hasFather chicco”, we can say:
  1. federico and chicco are the same individual
  2. (provided federico != chicco) the two sentences are inconsistent

- Inverse Functional Properties: the inverse property is functional

Transitive and Symmetric Properties

- Transitive Properties

  E.g., hasAncestor

- Symmetric Properties

  E.g., hasBrother

  "federico hasBrother paolo" allows to infer also

  "paolo hasBrother federico"

- Antisymmetric Properties (if "a rel b", it can never be "b rel a")

  E.g., hasChild: "federico hasChild francesco"

Reflexive Properties

- Reflexive properties: if rel is reflexive, then it always holds “a rel a”, for every indidual a.
  - E.g., knows: “federico knows federico”
- Irreflexive properties (it can never be “a rel a”)
  - E.g., fatherOf

Properties – Domain and Range

Properties relate individuals from a domain to individuals from a range

- **Domain** and **range are not constraints** but, rather, **axioms** used in the inference process

  E.g.: given the classes Pizza and PizzaToppings, the relation hasTopping is defined:

  - Pizza as domain
  - PizzaToppings as range

  If we add “iceCream hasTopping pepperoni”, then the inference engine derives that iceCream is an instance of Pizza

  - Such behaviour can generate problems (initially…)

- Note: for inverse properties, there is the “inversion” of domain and range

Describing and Defining a class

- Description of a class: necessary conditions for an individual to belong to a class

- Definition of a class: necessary and sufficient conditions for an individual to belong to a class

They are descripted/defined by means of

- Expressions (conjunction/disjunction of named/anonymous classes and subClassOf relation)
  - and (intersection), or (union) and not (complement)

- Property Restrictions (in Protégé everything has been reconducted to a restriction)

Property Restrictions

- Quantifier Restrictions
  - Existential Restrictions: individuals that are related through property prop with at least an individual member of the specified class (keyword "**some**", "**someValueFrom**")
  - Universal Restrictions: individuals that are related through property prop only with individuals members of the specified class (keyword "**only**", "**allValuesFrom**")

- Cardinality Restrictions (possibly qualified)
  - min n: individuals that are related through property prop with at least n other individuals
  - exactly n
  - max n

- hasValue Restrictions: individuals that are related through property prop with a certain defined indivdual

Protégé – Reasoning

- Computation of the "inferred ontology"
  - User defined/described ontologies are called "asserted hierarchy". Protégé can invoke a reasoner (a plug-in) to compute the "inferred hierarchy"

- Consistency checking: for each class, can exists at least an individual that could belong to such class?

- Warning: OWL uses the Open World Assumption
  - If needed, Closure Axioms

Protégé - Individuals

- It is possible to define also single individuals
- DL Reasoners supports also the classification task

Protégé – A short exercise

Starts from the Pizza ontology

- Add the description of a new, ChesaniPizza
  - It must have cheese
  - It must have a topping based on meat
  - It must be an "interesting pizza"
  - It must be vegetarian

- Should we make the intersection of these concepts or the union?

- Do you think ChesaniPizza is consistent? Why?

- Transform the description of ChesaniPizza into a definition
  - Which pizza are subsumed by ChesaniPizza?

Protégé – A short exercise

Create an individual, e.g., “pizzaTonight”

- Add some topping
  - Mozz_bufala
  - Sausage
  - Pachino Tomatoes
  - Aubergines
- Create pizzaTonight with these topping

- Try to classify it: to which class this pizza belong? Why?

Protégé – A short exercise

Domain and Range Assertions

- Create the class entity "Cake", as subclass of Food
- Add the property that a Cake hasTopping some FruitTopping
- Invoke the reasoner… what happens? how the cake has been classified?

- Add "disjoint with pizza
- Invoke the reasoner again…

Protégé – A short exercise

OPEN WORLD ASSUMPTION

- Let us analyse the definition of VegetarianPizza
- Create a new pizza Chesani2
- Add Chesani2 hasTopping some CheeseTopping

- Run the reasoner, classification task, and see what happens… why Chesani2 is not classified as VegetarianPizza?

- Modify the property by saying Chesani2 hasTopping only CheeseTopping

Few Links

- Protégé, http://protege.stanford.edu/
- Protégé-OWL, with tutorials, exrcise, howto etc. etc.: http://www.co-ode.org/
- OWL and other standards:

  http://www.w3.org/2001/sw/