Introduzione a Protégé

Federico Chesani, 9 Giugno 2009

Ontologie

Una ontologia è una descrizione formale esplicita di un dominio di interesse

Permette di specificare:

- Classi (cioè concetti del dominio)
- Relazioni semantiche tra classi
- Proprietà associate ad un concetto (slots, ed eventuali restrizioni, facets)
- Eventuale livello logico (assiomi, regole di inferenza)

- Istanze delle classi

Ontologia + Istanze = Knowledge Base

Ontologie

- Il punto di partenza è un'analisi approfondita, completa(?) e corretta(?) di un dominio applicativo.
- La conoscenza così individuata può essere rappresentata/ formalizzata tramite una ontologia
- Usi:
  - Esportabilità
    - Esportabilità dell'applicazione
    - Esportabilità della conoscenza
  - Modellazione
    - Utile in tutte le fasi di sviluppo di un progetto dall'analisi dei requisiti alla documentazione
  - Interoperabilità fra applicazioni differenti

Ontologie e Semantic Web

Due proposte supportano la definizione di ontologie in SW:

- RDF Schema (RDFS), estensione di RDF con termini appositi per rappresentare concetti ontologici

- OWL (Ontology Web Language), riprende le idee di RDFS e ne propone una implementazione a livello di linguaggio stesso
  - OWL Lite
  - OWL DL
  - OWL Full

  Formalmente, basato sulle Description Logics

OWL e Description Logic

- Le Description Logics sono una famiglia di logiche
- Si distinguono tra di loro a seconda degli operatori supportati
- Più sono gli operatori supportati → maggiore è la complessità
- Per OWL DL sono supportati i seguenti operatori:

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$1hasSSN$^-$

Protégé – Un editor per ontologie

- Esistono diversi editor per ontolgie:
  - WebODE, http://webode.dia.fi.upm.es/WebODEWeb/index.html
  - ICOM, http://www.inf.unibz.it/~franconi/icom/
  - DOME, http://dome.sourceforge.net/
  - … (molti altri)

  - Protégé, http://protege.stanford.edu/
    - Sviluppato a Stanford (US)
    - Open Source, java based, estendibile
    - Ambiente a plug-in, ricchissimo di plug-ins
    - Esporta ontologie in diversi formati (tra qui rdfs e owl)

Progettare un'ontologia

Protégé risulta essere di aiuto in alcune fasi della progettazione di una ontologia:

1. Determinare il **dominio** e lo **scopo** dell'ontologia
2. Considerare la possibilità di **riusare** ontologie esistenti
3. Individuare i **concetti** chiave del fenomeno da descrivere
4. Organizzare i concetti in classi e **gerarchie** tra le classi
5. Definire le **proprietà** delle classi
6. Definire "***vincoli***" (valori leciti)sulle proprietà
7. Creare le **istanze**
8. Attribuire i **valori** alle proprietà per tutte le istanze create

Protégé – Progettare un'ontologia

Le funzioni di editing sono divise, in base all'argomento, tramite tabs

- Tab "Classes": editor delle classi
- Tab "Object Properties"
- Tab "Data Properties"
- Tab "Individuals"

- Tab "Forms": permette la definizione di forms per l'inserimento dei dati (delle istanze)

Protégé – Editor delle classi

Per ogni classe, si possono specificare e/o visualizzare:

- Assertions (caretteristiche della classe/concetto)
  - Necessary and Sufficient (definizione completa) (C1 equivalnet to C2)
  - Necessary (definizione parziale) (C1 has superclass C2)
  - Inherited (ereditate da superclassi)
- Le asserzioni riguardano Properties e Restriction on properties
- Disjoints (gli individui di questa classe non posso essere anche individui di altre classi/concetti semplici e/o composti)
  - A default, le classi OWL hanno overlap!!!!!!

Protégé - Proprietà

Le proprietà sono relazioni binarie tra due cose ("things").

Tipi principali di proprietà:

- Object properties (relazione tra due individui): le classi (i concetti) sono caratterizzati dalle relazioni (proprietà) che coinvolgono gli individui di quella classe
- Datatype properties (relazione tra un individuo ed un tipo di dato primitivo)

- Annotation properties (metadata…)

Protégé – Proprietà inverse

Per ogni object property è possibile specificare anche la inverse property

E.g.:

Individui: federico, francesco

Proprietà: hasParent, hasChild

Sentence: "francesco hasParent federico"

Supponiamo di definire hasChild come la inverse property di hasParent

→ È possibile inferire automaticamente che:

"federico hasChild francesco"

Proprietà - Caratteristiche

- Functional Properties: una proprietà è funzionale se, per un dato individuo, esso può essere in relazione (tramite tale proprietà) con un solo altro individuo

- E.g.: "francesco hasFather federico"
- Nota: il fatto che una proprietà sia funzionale non è usato come vincolo, ma come assioma per l'inferenza. E.g., se asserisco anche "francesco hasFather chicco", posso dedurre che:
  1. federico e chicco sono lo stesso individuo, oppure
  2. (ammesso che sia stato dichiarato esplicitamente federico != chicco) le due frasi sono inconsistenti!

- Inverse Functional Properties: cioè la inverse property è functional

Proprietà - Caratteristiche

- Transitive Properties
  E.g., hasAncestor

- Symmetric Properties
  E.g., hasBrother
  "federico hasBrother paolo" permette di inferire anche
  "paolo hasBrother federico"
- Antisymmetric Properties (se "a rel b", allora non potrà mai essere "b rel a")
  E.g., hasChild: "federico hasChild francesco"

Proprietà - Caratteristiche

- Reflexive properties (se rel è riflessiva, allora vale sempre "a rel a", per ogni individuo a.
  E.g., knows: "federico knows federico", "federico knows andrea"

- Irreflexive properties (non può mai valere "a rel a")
  E.g., fatherOf

Proprietà – Domain e Range

Le proprietà mettono in relazione individui appartenenti ad un domain con individui appartenenti ad un range

- La definizione di domain e range non comporta un controllo di consistenza, bensì viene utilizzata come assioma per inferire nuova conoscenza

  E.g.: date le classi Pizza e PizzaToppings, la relazione hasTopping ha:
  - Pizza come domain
  - PizzaToppings come range

  Se scrivo "iceCream hasTopping pepperoni", allora viene inferito che iceCream è una istanza di Pizza
  - Questo comportamento può essere causa di problemi…

- Nota: Inversione di domain e range per le inverse properties

Descrivere e Definire le classi

- Descrizione di una classe: condizioni necessarie perché un individuo appartenga a quella classe
- Definizione di una classe: condizioni necessarie e sufficienti perché un individuo appartenga a quella classe

Le classi possono essere descritte/definite tramite:

- Espressioni (congiunzione/disgiunzione di classi named/ anonymous e relazione subClassOf)
  - congiunzione di and (intersezione), or (unione) e not (complemento)
- Restrizioni sulle proprietà (in Protégé 4.0 tutto è stato ridotto a restrizioni)

Descrivere classi tramite Restrizioni sulle proprietà

Le restrizioni possono essere:

- Quantifier Restrictions
  - Existential Restrictions: la classe i cui individui sono in relazione tramite la proprietà prop con almeno un individuo membro della classe specificata (keyword "some", "someValueFrom")
  - Universal Restrictions: la classe i cui individui sono in relazione tramite la propritetà prop con solo individui membri della classe specificata (keyword "only", "allValuesFrom")
  - (la semantica è data come implicazione logica… comprende anche gli individui il cui antecedente dell'implicazione è falso)
- Cardinality Restrictions (possibly qualified)
  - min n: gli individui di questa classe sono in relazione tramite la proprietà prop con almeno n individui
  - exactly n
  - max n
- hasValue Restrictions: individui che sono in relazione prop con un certo individuo

Protégé – Qualche ragionamento

- Calcolo della "inferred ontology"
  - Le classi definite/descritte dall'utente sono dette "asserted hierarchy". Tramite il meccanismo della sussunzione, Protégé (FACT++ reasoner e altri) offre il calcolo automatico della "inferred hierarchy" (comando "classify")
- Consistency checking: per ogni classe, può esistere almeno un individuo appartenente a tale classe?
- Attenzione: si fa assunzione, in OWL, di ipotesi di mondo aperto (Open World Assumption)
  - Se necessario, si introducono Closure Axioms

Protégé - Individui

- E' poi possibile specificare anche singoli individui
- Protégé offre la possiblità di esportare la knowldege base così definita

- I reasoner DL supportano ovviamente anche il task di classificazione dei singoli individui

Protégé – Breve esercizio

Consideriamo l'ontologia Pizza

- Descriviamo una nuova classe, ChesaniPizza
  - Deve avere il formaggio
  - Deve avere un topping a base di carne
  - Deve essere una pizza "interessante"
  - Deve essere vegetariana
  - Devo fare l'intersezione di questi concetti o l'unione?
- La ChesaniPizza è consistente? Perché?

- Trasformiamo la descrizione di ChesaniPizza in definizione
  - Quali pizze sono sussunte da ChesaniPizza?

Protégé – Breve esercizio

Creiamo un individuo, ad es. "pizzaStasera"

- Aggiungiamo prima alcuni ingredienti
  - Mozz_bufala
  - Luganega (o lucaneca)
  - Pomodoro_di_pachino
  - Melanzane
- Creiamo la pizzaStasera con questi ingredienti
- Invochiamo la classificazione: a che classe appartiene questa pizza? Perché?

Qualche link

- Protégé, http://protege.stanford.edu/
- Protégé-OWL, con tutorial, esercizi, spiegazioni, etc. etc.: http://www.co-ode.org/
- OWL e altri standard: http://www.w3.org/2001/sw/