JOT: a Protégé Scripting Environment for Creating and Managing Ontologies

Olivier Dameron

SMI - Stanford University

Protégé Conference 2004
1. Repetitive tasks
   ex: creation of lateralized concepts and their relationships
2. Enumerations
   ex: Ribs
3. Dependancies between concepts or relationships
   ex: Thorax / Skin of Thorax
4. Ontology maintenance
   require adhoc detection and fixing
Objective: Scripting environment for Protégé

1. Create macros
   1. repetitive and error-prone tasks
   2. formalism for handling intrinsic complexity
   3. towards more abstraction

2. Code reuse

3. User-friendly and powerful
   1. simple and intuitive syntax
   2. well formalised
Architecture

1. Principle
   1. Python interpreter in Java: Jython
   2. Thread (share address space with Protégé)

2. Shared variable: \textbf{kb}

3. Compatibility with frames and OWL
   1. instance of KnowledgeBase (Frames)
   2. instance of OWLKnowledgeBase (OWL)
1. Python Code
1. Python Code

2. Python Console
A r c h i t e c t u r e

1. Python Code

2. Python Console

3. Jython

1. Python Code

2. Python Console

3. Jython
1. Python Code

2. Python Console

3. Jython

4. Protégé
Frames

1. Get frame's attributes
2. Create frame
3. Create instances
F r a m e s

Classes
- Thing
- System-Class
- Winery (42)
- Wine region
- Consumable thing
- Food (1)
  - Meat
  - Fowl
  - Seafood
  - Pasta
  - Tomato-based food (1)
  - Dessert
  - Fruit
- Drink
- Meal course (5)
- Wine grape (15)

Python Console Tab - Protégé

```
>>> kb.getClass()  
<jclass edu.stanford.smi.protege.model.DefaultKnowledgeBase at 24103634>  
>>> kb.getClss("Food")  
Cls(Food, FrameID(1:10014))  
>>> kb.getClss("Food").getDirectSubclasses()  
[Cls(Meat, FrameID(1:10055)), Cls(Fowl, FrameID(1:10060)), Cls(Seafood, FrameID(1:10063)), Cls(Pasta, FrameID(1:10068)), Cls(Tomato-based food, FrameID(1:10075)), Cls(Dessert, FrameID(1:10076)), Cls(Fruit, FrameID(1:10079))]  
>>> kb.getClss("Food").getDirectInstances()  

>>> kb.getClss("Food").createDirectInstance("myNewFoodInstance")  
SimpleInstance(myNewFoodInstance of [Cls(Food, FrameID(1:10014))])
```
1. Get classes' attributes
2. Create class
3. Create relations
OWL

```python
>>> kb.getClass()
<clazz edu.stanford.smi.protege.owl.jena.JenaOWLKnowledgeBase at 13572454>

>>> kb.getNamedCls("AnatomicalConcept")
Cls(AnatomicalConcept, FrameID(1:10075))

>>> kb.getSlot("hasDirectPart")
Slot(hasDirectPart)
```
OWL

AnatomicalConcept

- LeftAnatomicalConcept
- RightAnatomicalConcept
- Heart
- Side

Heart

rdfs:comment

 asserted Conditions

- AnatomicalConcept

hasPart (multiple AnatomicalConcept)

hasSide (multiple Side)

Python Console Tab - Protégé

Jython 2.1 on platform java 5.0-beta2

>>> kb.getClass()
<jclass edu.stanford.smir.ontology.owl.jena.JenaOWLKnowledgeBase at 13572454>

>>> kb.getNamedCls("AnatomicalConcept")
Cls(AnatomicalConcept, FrameID(1:10075))

>>> kb.getSlot("hasDirectPart")
Slot(hasDirectPart)

>>> kb.createNamedSubClass("Heart", kb.getNamedCls("AnatomicalConcept"))
Cls(Heart, FrameID(1:10093))

>>>
Repetitive tasks

Creation of a lateralized anatomical concept: Hand
- create Hand
- create subconcepts LeftHand and RightHand
- define LeftHand = Hand on the LeftSide
- Hand: either LeftHand or RightHand
- LeftHand and RightHand are disjoint
Repetitive tasks

Create clone
Create subclass
Create subclass using metaclass
Delete selected class
Change metaclass
Change metaclass of subclasses
Hide class
Expand
collapse
Sort direct subclasses
Sort all subclasses
Check Consistency
Classify sub-tree
Extract sub-ontology to file
Add covering axiom
Convert to defined class
Search subclass by property value
Set all subclasses disjoint
Set deprecation flag
Necessary and Sufficient Conditions:
- hand
- right anatomical concept
Necessary and Sufficient Conditions:
- left hand
- right hand

Necessary Conditions:
- anatomical concept
createLateralizedConcept(“Hand”, “Anat”):
  c = createConcept(“Hand”, “AnatomicalConcept”)
  lc = createConcept(“LeftHand”, “Hand”)
  rc = createConcept(“RightHand”, “Hand”)
  define c = lc or rc
  define lc = c and LeftAnatomicalConcept
  define rc = c and RightAnatomicalConcept
  make lc and rc disjoint
Necessary and Sufficient Conditions:
- left finger
- right finger

Necessary Conditions:
- anatomical concept
Repetitive tasks

Necessary and Sufficient Conditions:
- finger
- left anatomical concept
After classification:
- LeftThumb
- LeftIndex
- LeftMiddleMiddleFinger
- LeftRingFinger
- LeftLittleFinger

... are LeftFinger
Enumerations

1. Vertebrae
2. Ribs (lateralized !)
3. Muscles
Dependencies

1. Ex: Wall of Heart
   - Heart = WallOfHeart, Septum, 4 cavities
   - Heart = LeftAtrium, RightAtrium, LeftVentricle, RightVentricle

2. WallOfLeftAtrium
   - constitutionalPartOf LeftAtrium
   - regionalPartOf WallOfHeart

3. Epicardium, Myocardium, Endocardium
   - constitutionalPartOf WallOfLeftAtrium
   - regionalPartOf Epicardium
Dependencies

- 16 composed concepts
- 32 relations of direct composition

- You are lucky if you don't forget one
- If you do, enjoy the debugging
- Not scalable
  - no neighborhood relationships
  - \{anterior, posterior, lateral\} + \{inferior, superior\} parts of LeftAtrium
Ontology maintenance

1. Make specific functions on the fly
2. Reuse functions
3. Dynamically insert / remove java listeners
4. Take advantage of all the existing Java libraries (web services, ...)

Lessons learned

- JOT is useful :-)
  - higher level functions
  - from extensional to intensional description
- Domain-independent but language-specific macros
- Domain-dependant but language independent functions
  (reuse functions from 1 according to the language)
1. Direct calls to the Protégé API => no limitations
2. Jython => power of Python + Java
3. Code reuse allow to hide the low-level Protégé API