Tutorial:
Creating Semantic Web (OWL) Ontologies with Protégé
Holger Knublauch, Mark A. Musen, Natasha F. Noy

Sanibel Island, Florida, USA, October 20-23th, 2003
Creating Semantic Web (OWL) Ontologies with Protégé

Overview

• Ontologies and the Semantic Web
  – The Semantic Web Vision
  – What is an ontology?
  – The Web Ontology Language (OWL)
• Editing OWL Ontologies with Protégé
  – General Introduction to Protégé
  – Editing the basic elements of OWL (Classes, Properties, Individuals)
  – Editing class expressions
  – Using a classifier and PromptDiff
• Advanced Protégé
  – Visual ontology editing with the GraphWidget and ezOWL
  – Metaclasses
  – Understanding the OWL Plugin
  – Developing plugins and applications with the Protégé/OWL API
• Current limitations and outlook

Holger Knublauch
holger@smi.stanford.edu

Mark A. Musen
musen@smi.stanford.edu

Natasha F. Noy
noy@smi.stanford.edu

Stanford Medical Informatics
Semantic Web / Motivation (1)

Semantic Web / Motivation (2)
Semantic Web / Motivation (3)

• Next generation web pages: Machine accessible semantics

• Search engines
  – concepts, not keywords
  – semantic narrowing/widening of queries

• Shopbots
  – semantic interchange, not screenscraping

• E-commerce
  – negotiation, catalogue mapping, personalization

• Web Services
  – need semantic characterizations to find them

• Navigation
  – by semantic proximity, not hardwired links

Semantic Web / Ontologies

• Formal, explicit specification of a shared conceptualization

• What’s inside an ontology
  – Classes + class-hierarchy
  – Properties (Slots) / values
  – Relations between classes (inheritance, disjoint, equivalent)
  – Restrictions on properties (type, cardinality)
  – Characteristics of slots (symmetric, transitive, …)
  – (possibly) Individuals

• Reasoning tasks: classification, subsumption
Tell me what wines I should buy to serve with each course of the following menu.

I recommend Chardonnay or Dry Riesling.

Semantic Web / OWL

- Web Ontology Language (OWL)
- Developed by a World Wide Web Consortium (W3C) working group
- Based on DAML+OIL

- Semantic Web Vision: To enable machines to **comprehend** semantic documents and data
- Habitat for Autonomous Agents
- OWL facilitates greater machine readability of Web content than XML
- Extends RDF and RDF Schema by providing additional vocabulary along with a formal semantics
Semantic Web / OWL / Language Overview

- Classes
- Properties
  - DatatypeProperties (boolean, float, integer, string)
  - ObjectProperties (relationships between classes)
- Individuals

- Built-in ontology mapping support (equivalent classes, sameAs)
- Some other property types (e.g., symmetric, transitive, functional)

- Class Descriptions
  - can be used instead of named classes (e.g., to define superclasses)
  - define classes by the attributes of their members
    - enumerations red, green, or blue
    - restrictions all individuals that have at least 2 children
    - logical statements Person and not Student and not blue eyes

Protégé

- An extensible and customizable toolset for constructing ontologies and for developing applications that use these ontologies
- Outstanding features
  - Automatic generation of graphical-user interfaces, based on user-defined models, for acquiring domain instances
  - Extensible knowledge model and architecture
  - Possible embedding of standalone applications in Protégé knowledge engineering environment and vice versa
  - Scalability to very large knowledge bases
Protégé / Historical background: early days

- ONCOCIN (1980s)
  - Clinical decision-support system (CDSS) for management of patients enrolled in cancer clinical trials

- OPAL (~1985)
  - A graphical user interface to encode cancer clinical trials for ONCOCIN based on a model of cancer trials

- Protégé (Mark Musen dissertation)
  - A system to define model of trials for any domain, to generate OPAL for eONCOCIN (CDSS for any trial domain)

Protégé / Historical background: 1990s – present

- Protégé-II (early 1990s)
  - A knowledge engineering environment (on NeXTStep platform) to define model and generate GUI editor for any domain

- ProtégéWin (mid 1990s)
  - Windows version that emphasized usability
  - External user groups

- Protégé-2000 (late 1990 – present)
  - Java-based version that emphasized formal knowledge model and interoperability
  - Development of extensible plugin architecture
  - Open source
  - Renamed to Protégé in Version 2.0 (Fall, 2003)
  - OWL Support (work in progress, reasonably stable since Summer 2003)
Protégé / General Concepts / Classes

Protégé / General Concepts / Slots (Properties)
Protégé / A world-wide user community

- Stanford offers support to individual users via “protege-help” mailing list
- Users support one another and brainstorm about new ideas using the “protege-discussion” mailing list
- Protégé Web pages provide access to contributed plug-ins, ontologies, help manuals, FAQs, and scientific publications
- Every year, we hold a Protégé users group meeting for both technical discussions and schmoozing

![Protege Cumulative Registration Totals Chart]

Protégé / Lots of user-contributed “plug ins”

- Like Web browsers, Protégé-2000 accepts a wide range of “plug ins” that enhance its functionality
- Many of these plug ins are contributed by members of our user community with little or no discussion with our development team
- It’s absolutely amazing to see what our user community has contributed back to us!

- New visualization systems
- New inferencing systems
- New import and export formats
- New user-interface features
- New means of accessing external data sources
Protégé / Storage Formats

- Users edit and view ontologies in a manner that insulates them from the ultimate storage format.
- Ontologies may be read in from, written out to, and interconverted between a large number of formats:
  - Relational databases (ODBC)
  - CLIPS
  - UML / XMI
  - XML / XML Schema
  - RDF
  - Topic Maps
  - DAML+OIL
  - OWL

Protégé / Role of Protégé in the Semantic Web

Semantic Web

Paradigms | Languages | Tools | Applications
----------|-----------|-------|-------------
Description Logics | OWL | ezOWL, etc | Decision Support
Frames / OO | OKBC | Protégé OWL Plugin | Classification
Protégé | | | Reasoning
Protégé / OWL Plugin

• Extension of Protégé to allow editing OWL ontologies
• Project started April 2003, based on ideas from previous projects (OilTab, RDF and DAML+OIL backends)
• Currently in beta release (http://protege.stanford.edu/plugins/owl)

• Features
  – Loading and saving OWL files
  – Graphical editors for class expressions
  – Access to description logics inference components such as classifiers
  – Powerful platform for hooking in custom-tailored components

Protégé / OWL / Installation

1. Install Java 2 Virtual Machine (SDK version 1.4.2)
2. Install latest version of Protégé 2.0 http://protege.stanford.edu/download/prerelease
3. Download OWL Plugin and unzip it into plugins folder http://protege.stanford.edu/plugins/owl/download/protege-owl.zip (This will create a folder plugins/owl containing two files)

Optional Components
• Install other plugins such as ezOWL
• Install classifiers such as Racer
Protégé / OWL / Ontology Development Process

- Ontology development is an iterative process
- Roughly consists of the following activities

1. Determine scope
2. Consider reuse
3. Enumerate terms
4. Define classes
5. Define properties
6. Create instances
7. Classify ontology

- Ontology development requires
  - some experience and foresight
  - communication between domain experts and developers
  - a tool that is easy to understand, yet powerful
  - a tool that supports ontology evolution

Protégé / OWL / Files and Projects

- An OWL Project typically consists of two files
  - an .owl file (e.g. wine-classified.owl): Contains the ontology itself as RDF
  - a .pprj file (e.g. wine-classified.pprj): Contains project metadata such as layout information, file names
- Project/New... OWL Files – Creates a new (empty) project
- Project/Open... – Loads an existing project
- Project/Save as... – Saves the current project under a given name

Alternative serializations of OWL (e.g. N3)
Protégé / OWL / Files and Projects / Import

- Use Project/Import... to load an existing .owl file that was
  - developed with another tool
  - developed with a previous version of Protégé (.pprj file might be outdated)

Protégé / OWL / Classes / Tab

- Inheritance / Taxonomy
- Metadata & Comments
- Allowed properties
- Logical definition
Protégé / OWL / Classes / Inheritance Tree (1)

- Displays subclass / superclass relationship
- :THING is top-level class (owl:Thing in OWL)

View / Edit class in extra window
Create subclass
Delete class

Protégé / OWL / Classes / Inheritance Tree (2)

- Drag and drop
- Multiple inheritance
- More options in popup menu

View / Edit superclass
Create superclass
Add superclass
Remove superclass
Delete class expression
Protégé / OWL / Classes / Class metadata

- Class name
- Labels in multiple languages
- Documentation and annotations for housekeeping (versioning, author, etc)

Protégé / OWL / Resource Metadata

Standard metadata for all ontology resources

<table>
<thead>
<tr>
<th>Name</th>
<th>OWL Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>rdf:ID</td>
<td>A unique name for the resource</td>
</tr>
<tr>
<td>Documentation</td>
<td>rdfs:comment</td>
<td>A comment describing the resource</td>
</tr>
<tr>
<td>Labels</td>
<td>rdfs:label</td>
<td>Labels in multiple languages (alternative to Name)</td>
</tr>
<tr>
<td>SameAs</td>
<td>owl:sameAs</td>
<td>List of resources that are known to be identical</td>
</tr>
<tr>
<td>DifferentFrom</td>
<td>owl:differentFrom</td>
<td>List of resources that are known to be not identical</td>
</tr>
</tbody>
</table>
Protégé / OWL / Resource Metadata / Names

- Each resource (Class, Property or Individual) must have a unique name (within its namespace)
- Names must be the local part of a URI
- Support for multiple namespaces is in progress, and the editor will reflect this (e.g. displaying namespace placeholders such as wine:Riesling if multiple namespaces are used)
- In Protégé, names
  - must start with a letter or the underscore (_)
  - must only contain letters, digits, _, -, .
  - can not contain spaces
- Protégé will display illegal names in red and replace illegal characters with underscores (_)
Protégé / OWL / Resource Metadata / Annotations

- Annotations are comments on the resource
- Have no formal semantics, are ignored during reasoning
- Must be values of properties that are marked as “Annotation Properties” (type owl:AnnotationProperty)
- Any annotation property can be assigned to any resource
- Some pre-defined annotation properties exist
- Currently only text strings are supported

![Image](user-defined-and-pre-defined-annotation-properties-can-be-used)

Protégé / OWL / Classes / Properties

- List of all properties that can be assigned to individuals of this class (Either explicitly (property has the class as domain) or by use)
- Similar to attributes and relationships in object-oriented languages
- Inherited properties have a white S

![Image](view-edit-property-create-and-add-property-add-existing-property-remove-property)
Protégé / OWL / Properties

- A property can be used to assign values to individuals (and classes and properties) and to define relations between them.

- Properties can have characteristics / attributes [These are defined in the Property metaclass]

- Properties have “global” and “local” characteristics
- Global characteristics define general properties of the property
- Local characteristics define properties when the property is assigned to a certain class
- Local attributes are represented by means of OWL restrictions (below)
- Some characteristics can have both global and local values (i.e. they are “overloaded” for a local class):
  - Range (global) = All Values From (local)
  - FunctionalProperty (global) = Maximum cardinality of 1 (local)

---

Protégé / OWL / Properties / Global Characteristics

<table>
<thead>
<tr>
<th>OWL Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>rdfs:range</td>
</tr>
<tr>
<td>Domain</td>
<td>rdfs:domain</td>
</tr>
<tr>
<td>Domain Defined</td>
<td>(explicit domain)</td>
</tr>
<tr>
<td>Super-properties</td>
<td>rdfs:subPropertyOf</td>
</tr>
<tr>
<td>Equivalents</td>
<td>owl:equivalent-Property</td>
</tr>
<tr>
<td>Inverse Property</td>
<td>owl:inverseOf</td>
</tr>
<tr>
<td>Transitive</td>
<td>owl:Transitive-Property</td>
</tr>
<tr>
<td>Symmetric</td>
<td>owl:Symmetric-Property</td>
</tr>
<tr>
<td>InverseFunctional</td>
<td>owl:Inverse-FunctionalProperty</td>
</tr>
<tr>
<td>Multiple Cardinality</td>
<td>owl:Functional-Property</td>
</tr>
<tr>
<td>Annotation-Property</td>
<td>owl:Annotation-Property</td>
</tr>
</tbody>
</table>
Protégé / OWL / Properties / Range

- Range defines the type of values that can be assigned to the property
- If the range is “Instance” then you also have to specify the list of allowed classes
Protégé / OWL / Properties / Global Cardinality

- The global cardinality states whether multiple values can be assigned to a given property
- Similar to 0..1 or 0..n relationships in relational and object-oriented models
- Properties with global cardinality of 0..1 in OWL are called functional properties
- Default cardinality is usually multiple

Protégé / OWL / Properties / Inverse Properties

- Inverse properties are bi-directional
- If Wine A has maker Winery B, then Winery B produces Wine A
Protégé / OWL / Properties / Transitive

- Typically represent certain part-whole relations
- Example: Because the SantaCruzMountainsRegion is locatedIn the CaliforniaRegion, then it must also be locatedIn the USRegion, since locatedIn is transitive.

Protégé / OWL / Properties / Symmetric

- A symmetric property must always hold in both directions
- MendocinoRegion is adjacent to SonomaRegion and vice-versa.
- Symmetric properties must have equal range and domain
• For inverse functional properties, if two individuals have the same value then we can infer that the individuals are the same, too
• Example: Each Wine has exactly one Winery as producer

Protégé / OWL / Properties / Domain

• The domain of a property is the collection of classes that can have a value for the property
• In Protégé this is traditionally defined from a class point-of-view, i.e. you define a class and then add the properties it can have (the so-called template slots)
• OWL allows to define properties without any restrictions on the domain: This means the property can be used anywhere
  → Difficult to guide form-based knowledge acquisition process
• Option “Domain defined” specifies whether or not there is a domain restriction defined
Protégé / OWL / Properties / Local Characteristics

- Many characteristics can be defined for each class individually
- OWL concept: Property Restrictions
- Sometimes called “facet overloading”
- Switch to “Local at class” tab of property

<table>
<thead>
<tr>
<th>OWL Restriction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum Cardinality</td>
<td>The minimum number of values that the property must have at the given class</td>
</tr>
<tr>
<td>Maximum Cardinality</td>
<td>The maximum number of values that the property can have at the given class</td>
</tr>
<tr>
<td>All Values From</td>
<td>The type (class) that all values of the property must have at the given class</td>
</tr>
<tr>
<td>Some Values From</td>
<td>The type (class) that at least one of the values of the property must have at the given class</td>
</tr>
<tr>
<td>Has Value</td>
<td>The value that at least one of the values of the property must be at the given class</td>
</tr>
</tbody>
</table>

Protégé / OWL / Properties / Cardinality

- Cardinality specifies the number of values that the individuals of the class must and can have for the property
  - Minimum cardinality / number of required values
  - Maximum cardinality / number of allowed multiple values
Protégé / OWL / Properties / All Values From

- AllValuesFrom specifies the types that all values of the property must have for instances of this class
- Mostly used for instance properties
- Overloads the global range
- If there is more than one type, then they are interpreted as union, i.e. all values must have one of the types:

```xml
<owl:allValuesFrom>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#FriendlyPerson"/>
      <owl:Class rdf:about="#HappyPerson"/>
    </owl:unionOf>
  </owl:Class>
</owl:allValuesFrom>
```

Protégé / OWL / Properties / Some Values From

- Specifies the types that at least one of the values of the property must have for instances of this class
- Mostly used for instance properties
- For example, every AcademicPerson must have at least one workplace at a University
Protégé / OWL / Properties / Has Value

- Specifies one value that the individuals of the class must have for the property
- For example, all Sweet Rieslings have white color

Protégé / OWL / Restrictions

- Restrictions are a special type of anonymous class definitions
- A restriction defines an anonymous class
- Members of this class are all individuals that fulfill the restriction
- Restrictions are often used to define necessary conditions for a class
• To add a restriction either
  – Left-click one of the c buttons; this will open the dialog on the right
  – Right-click one of the c buttons, select the property to restrict and then edit the filler in the table

Protégé / OWL / Restrictions / Synchronization (1)

• Local characteristics and restrictions are synchronized
• For example, if you state that SweetRiesling has white color, then the system automatically adds a HasValue restriction on the property for the class
• In this example, multiple allowed classes for the property friends are converted into an allValuesFrom restriction at the class with a union of the two allowed classes.

• Two mechanisms to edit restrictions:

  • Define local characteristics on the property form
    → Bundles all characteristics from a property’s point of view

  • Define restrictions on the class form
    → Displays the class definition
Protégé / OWL / Expressions

- OWL supports class definitions made out of logical combinations of other classes
  - Example: “A GermanWine is a Wine that is made in Germany”

- Example: “A tasty Wine is a Wine that is not a German Wine”

Protégé / OWL / Expression Syntax

- RDF is the official OWL syntax:
  ```xml
  <owl:Class rdf:ID="GermanWine">
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Wine"/>
          <owl:Restriction>
            <owl:onProperty>
              <owl:ObjectProperty rdf:about="#locatedIn"/>
            </owl:onProperty>
            <owl:hasValue rdf:resource="#GermanyRegion" rdf:type="#Region"/>
          </owl:Restriction>
        </owl:intersectionOf>
        <rdfs:subClassOf>
          <owl:Class rdf:about="#Wine"/>
        </rdfs:subClassOf>
      </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Wine"/>
    </rdfs:subClassOf>
  </owl:Class>
  ```

- Intended for machines
- Needed: A user friendly syntax

\[ \text{Wine} \cap (\text{locatedIn} \equiv \text{GermanyRegion}) \]
### OWL Element Syntax in Protégé

<table>
<thead>
<tr>
<th>OWL Element</th>
<th>Symbol</th>
<th>Key</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>allValuesFrom</td>
<td>∀</td>
<td>*</td>
<td>∀ children Male</td>
</tr>
<tr>
<td>someValuesFrom</td>
<td>∃</td>
<td>?</td>
<td>∃ children Lawyer</td>
</tr>
<tr>
<td>hasValue</td>
<td>∋</td>
<td>$</td>
<td>rich ∋ true</td>
</tr>
<tr>
<td>cardinality</td>
<td>=</td>
<td>=</td>
<td>children = 3</td>
</tr>
<tr>
<td>minCardinality</td>
<td>≥</td>
<td>&gt;</td>
<td>children ≥ 3</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>≤</td>
<td>&lt;</td>
<td>children ≤ 3</td>
</tr>
<tr>
<td>complementOf</td>
<td>¬</td>
<td>!</td>
<td>¬ Parent</td>
</tr>
<tr>
<td>intersectionOf</td>
<td>⊓</td>
<td>&amp;</td>
<td>Human ⊓ Male</td>
</tr>
<tr>
<td>unionOf</td>
<td>⊔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>enumeration</td>
<td>{...}</td>
<td>{}</td>
<td>{male female}</td>
</tr>
</tbody>
</table>

### Protégé / OWL / Expression Editor

- **Text field** for keyboard or mouse users
- **Qualified Restrictions** (all, some, has)
- **Cardinality Restrictions**
- **Set operators** (and, or, not)
- **Resources** (Classes, Properties, Individuals)
- **Datatypes** (boolean, float, integer, string)
- **True and False**
- **Brackets** (e.g. sets)
- **Backspace** for mouse
Protégé / OWL / Expression Editor / Synonyms

- Some symbols are not on the keyboard
- There are special keys for them, but also synonyms (after SPACE)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Key</th>
<th>Synonyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀</td>
<td>*</td>
<td>all valuesFrom for all only</td>
</tr>
<tr>
<td>∃</td>
<td>?</td>
<td>some valuesFrom exists</td>
</tr>
<tr>
<td>∋</td>
<td>$</td>
<td>has hasValue value</td>
</tr>
<tr>
<td>=</td>
<td>=</td>
<td></td>
</tr>
<tr>
<td>≥</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>≤</td>
<td>&lt;</td>
<td></td>
</tr>
<tr>
<td>¬</td>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>⊓</td>
<td>&amp;</td>
<td>and</td>
</tr>
<tr>
<td>⊔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{...}</td>
<td>{}</td>
<td></td>
</tr>
</tbody>
</table>

Protégé / OWL / Expression Editor / Syntax Checking

- You can only assign valid OWL expressions
- The Editor checks the syntax while you type
- The “nerd” icon will get a red face if something is wrong
- Pressing ENTER then displays an error message
Protégé / OWL / Expression Editor / Auto-Completion

- Keyboard users only need to type the first few letters of a class/property/individual name
- Pressing TAB or CTRL+SPACE opens Auto-Completion box
- If there is only one choice, the name is inserted directly
- Select an existing name or continue typing to narrow/widen the choice
- UP/DOWN to select, ENTER to insert, ESCAPE to cancel

Protégé / OWL / Classes / Equivalent Classes

- List of classes that are known to have exactly the same instances
- Equivalent classes are super/sub classes of each other
- Can contain named classes or class expressions
- Defines necessary and required conditions for class membership
- Can be used as logical definition of the class
Protégé / OWL / Classes / Moving Classes

- Superclasses / Restrictions define necessary conditions
- Equivalent classes define necessary and sufficient conditions
- Sometimes during ontology evolution, users may want to switch
- Equivalent classes are moved into
  - Restrictions: if they are restrictions
  - “Plain” superclasses: otherwise

Protégé / OWL / Classes / Disjoint Classes

- In OWL, classes may have shared instances
- If two classes are defined as disjoint then there is no individual that is member of both at the same time
- Protégé usually adds both directions of this symmetric relationship
Protégé / OWL / Individuals

- Individuals are specific instances of the classes from the ontology
- Each individual can have multiple classes as “types”
- Protégé automatically creates forms for the acquisition of individuals
Enumerated classes are defined by listing all its instances exhaustively. Often used in an equivalent class statement.

To define an enumerated class:
1. Define the class
2. Create the individuals (e.g. White) using the Individuals Tab
3. Create an equivalent enumeration using { ... }

Protégé automatically generates forms for all classes. Each property is represented by one widget. Default widgets are selected for the value types. Default layout heuristics are applied for user-defined classes. The default layout may be sufficient in most cases but often applications want to make custom-tailored adaptations. The Forms Tab can be used:
- to modify the layout of the forms
- to select alternative widgets for selected properties
Protégé / OWL / Forms / Editing the class form

- Two alternative views of properties
- Configure owl:Class
- Double-click on widget

Protégé / OWL / Forms / Adding properties quickly

- The small buttons allow users to change the type of a resource, and its form, rapidly, without having to switch tabs
- Allow simple building of “anonymous types” to ask queries

Create new property for type
Add existing property to type
Remove property from type
Edit type (opens class form)
Edit form in Forms Tab
OWL by default assumes that two individuals could represent the same thing. AllDifferent explicitly defines sets of individuals that are not the same (abbreviation for multiple owl:differentFrom statements).

Protégé / OWL / Species Validator

- The OWL Language is divided into three dialects
- OWL Lite:
  - Classification hierarchy
  - Simple restrictions
- OWL DL:
  - Maximal expressiveness while maintaining tractability
  - Standard formalization
- OWL Full:
  - Very high expressiveness (e.g. metaclasses, classes as values)
  - All syntactic freedom of RDF (self-modifying)
  - Loosing tractability (reasoning algorithms become inefficient)

Protégé supports OWL DL and parts of OWL Full
Protégé / OWL / Species Validator

• The Species Validator determines whether an ontology uses constructs from OWL Lite, OWL DL or OWL Full

Protégé / OWL / Source Code Viewer

Re-assign from source code
Protégé / OWL / Classification

- OWL’s background of description logics allows to use DL reasoners
- Classification is a reasoner that computes the subsumption relationships (inheritance) between classes based on their logical definition: Result is a new class hierarchy of the existing classes

Protégé / OWL / Classification / Getting Started

- Protégé allows the user to plug in external classifiers
- We use the DIG interface to access them (e.g. Racer)
- Classifier must execute as a server process
  - Download Racer
  - Execute it together with Protégé
  - If necessary, configure the classifier’s URI (required for Mac users)
PromptDiff displays the changes in the inheritance hierarchy using color coding and comments.
Protégé / OWL / Classification / Example / Result

The Classifier has found out that Sauterne is a WhiteBordeaux because:
- SauterneRegion is part of BordeauxRegion
- Sauterne has white color

Protégé / OWL / Metaclasses

- Protégé classes and properties are instances of metaclasses.
- Metaclasses thus defines the characteristics of other classes.
- Traditional object-oriented systems have a similar structure:

  - Protégé provides its own metamodel as a Protégé ontology.
  - The metamodel can be extended by plugins and by users.
Protégé / OWL / Metaclasses / Creating Metaclasses

Protégé / OWL / Metaclasses / Creating Classes
Protégé / OWL / Metaclasses / Usage

- Use of metaclasses means your ontology is in OWL Full
  → Limited reasoning support due to intractability

- Alternative: Annotation properties
  - Can be assigned to any ontology resource
  - Free choice of properties at any time

- But...
  - Metaclasses add widgets to the class editing form
  - Make explicit which values should be entered
  - Guide the knowledge-acquisition process
Protégé / OWL / API

- OWL is a relatively new language
- Few tools and libraries exist to parse and manipulate OWL files
  - Jena API (HP Labs, Bristol)
    - RDF parser library
    - Focus on RDF syntax trees
    - Large user community
    - Insufficient event support
  - OWL API (Manchester, Karlsruhe)
    - Pure OWL API
    - Focus on OWL Abstract syntax
    - Very new
  - Protégé/OWL API
    - Optimized for component development for Protégé
    - Uses Jena API and OWL API for special services
Protégé / OWL / API / Writing Applications

```java
OWLKnowledgeBase okb = ...

NamedCls personCls = okb.createNamedCls("Person");
NamedCls aCls = okb.createNamedCls("HappyPerson");
NamedCls bCls = okb.createNamedCls("OtherPerson");
aCls.addDirectSuperclass(personCls);
bCls.addDirectSuperclass(personCls);

OWLSlot slot = okb.createOWLSlot("children");
slot.setValueType(ValueType.INSTANCE);
slot.setAllowedClses(Collections.singleton(personCls));
personCls.addDirectTemplateSlot(slot);

SomeRestriction aRestriction =
    okb.createSomeRestriction(slot, aCls);
personCls.addDirectSuperclass(aRestriction);
```

Protégé / OWL / API / Plugins

- Protégé is an open platform that allows extensions (plugins)
  - Storage Plugins: Load and save ontologies in various formats
  - Tab Plugins: Provide a top-level user interface component
  - Slot Widgets: Customized components for editing property values
- Developing plugins is really simple!
Protégé / OWL / Graph Widget

The GraphWidget can be used to edit instance properties visually.

Protégé / OWL / API / Plugins / Installation

- Many Plugins can be downloaded from Protégé web site
- They usually consist of one or more .jar files
- Copy these .jar files into the plugins folder of your Protégé installation
- Tab plugins usually have to be activated before you see them (Project/Configure...)

Configure file://C:/projects/owl4wine:classified.pprj
Protégé / OWL / Limitations and Future Work

- Major limitation: Only one namespace per project
  - Import currently not supported
  - Will become available with Protégé 2.1
- Very few OWL Language features are missing
  - Enumerated datatypes
- A-Box Reasoning: Given the following properties, to what classes does this individual belong
- Alternative syntaxes (e.g. “OWL for Dummies” formats)
- Database backend and Multi-User support are almost finished
Protégé / OWL / Summary

- Protégé has a long history of success in ontology development
- Protégé has an open, extensible platform for tools and applications
- The large user community ensures support and future evolution

- The OWL Plugin is an extension of Protégé with editors, storage mechanisms, and access to reasoners
  - Integrated in the look-and-feel of Protégé
  - Providing optimized editing support for OWL specific language elements

- Third-party extensions exist (e.g. for visual editing)
- Many people are already using Protégé/OWL

→ Get involved and contribute your ontologies and plugins!

Protégé / OWL / More Information

- Protégé web site: http://protege.stanford.edu
- OWL Plugin: http://protege.stanford.edu/plugins/owl
- Getting help: protege-discussion list