



---

*Protégé-2000:  
A Flexible and Extensible  
Ontology-Editing Environment*

---

Natalya F. Noy,  
Monica Crubézy, Ray W. Fergerson,  
Samson Tu, Mark A. Musen

Stanford Medical Informatics  
Stanford University  
Stanford, CA

---

---

# *Facts (maybe)*

---

Fact 1: Ontologies are no longer just for knowledge engineers

Fact 2: A number of new Semantic-Web languages and representation formalisms are emerging; no agreement yet

Fact 3: We are developing ontologies, agents, and applications today, without waiting for the standards

---



# *Facts and Requirements*

## ■ Facts:

- Ontologies are no longer just for knowledge engineers
- A number of new Semantic-Web languages and representation formalisms are emerging; no agreement yet
- We are developing ontologies, agents, and applications today, without waiting for the standards

## ■ Requirements:

- Domain experts need to understand and maintain ontologies
- We need adaptable tools which we can tune to support new languages and formalisms quickly
- We need suites of tools for ontology development and management

# A Solution

---

- Protégé-2000 is an ontology-editing and knowledge-acquisition environment, which has
    - a graphical and easy-to-use interface
    - a flexible knowledge model
    - an extensible plugin architecture
    - an existing set of plugins for
      - ontology merging
      - acquisition of information from online knowledge sources
      - constraint specification and verification
      - .... (we don't even know them all)
-

# *Protégé-2000 Knowledge-Model Components*

- **Classes**  
*concepts in a taxonomic hierarchy*
- **Instances**  
*instances of classes*
- **Slots**  
*first-class objects representing properties of classes and instances*
- **Facets**  
*constraints on allowed slot values, such as cardinality, defaults, allowed classes, and so on.*

# Ontologies in Protégé-2000

The screenshot displays the Protégé-2000 interface for editing an ontology. The main window title is "wines Protégé-2000 [D:\Protege\Tutorial\Wines\wines.ppr]". The menu bar includes "Project", "Edit", "Window", and "Help". The toolbar contains icons for file operations and navigation. The "Classes" tab is selected, showing a hierarchical tree of classes:

- Drink
  - Wine
    - White wine
    - Red wine
      - Beaujolais
      - Red Burgundy
      - Red Zinfandel
      - Red Bordeaux
        - Medoc
          - Pauillac
          - Margaux
          - St. Emillion
          - Graves
        - Pinot Noir
        - Chianti
        - Petit Chablis

The right pane shows the details for the selected "Wine" class. It includes a "Name" field with the value "Wine", a "Documentation" field with the text "A wine class represents all possible wines", and a "Constraints" field. The "Role" is set to "Concrete". Below this is a "Template Slots" table:

Name	Type	Cardinality	Other Facets
S body	Symbol	single	allowed-values={FULL,MEDIUM,LIGHT}
S color	Symbol	single	allowed-values={RED,ROSÉ,WHITE}
S flavor	Symbol	single	allowed-values={DELICATE,MODERATE,STRONG}
S grape	Instance	multiple	classes={Wine grape}
S maker	Instance	single	classes={Winery}
S name	String	single	
S sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}

# Acquiring instance data

The screenshot displays the Protégé-2000 software interface. The main window shows a class hierarchy on the left, with 'Wine' selected. The right pane shows the 'Wine' class details, including its name, documentation, and a list of template slots. A dialog box is open in the foreground, showing the configuration for the instance 'Chateau Lafite Rothschild Pauillac'. The dialog includes fields for Name, Maker, Body, Color, Grape, Flavor, Tannin Level, and Sugar.

**Wine Class Details:**

Name	Documentation	Constraints
Wine	A wine class represents all	

Role
Concrete

Template Slots	
Name	Type
S body	Symbol
S color	Symbol
S flavor	Symbol
S grape	Instance
S maker	Instance
S name	String
S sugar	Symbol

**Chateau Lafite Rothschild Pauillac (Pauillac) Instance Configuration:**

Name	Maker
Chateau Lafite Rothschild Pauillac	Chateau Lafite Rothschild

Body	Color	Grape
FULL	RED	Cabernet Sauvignon grape

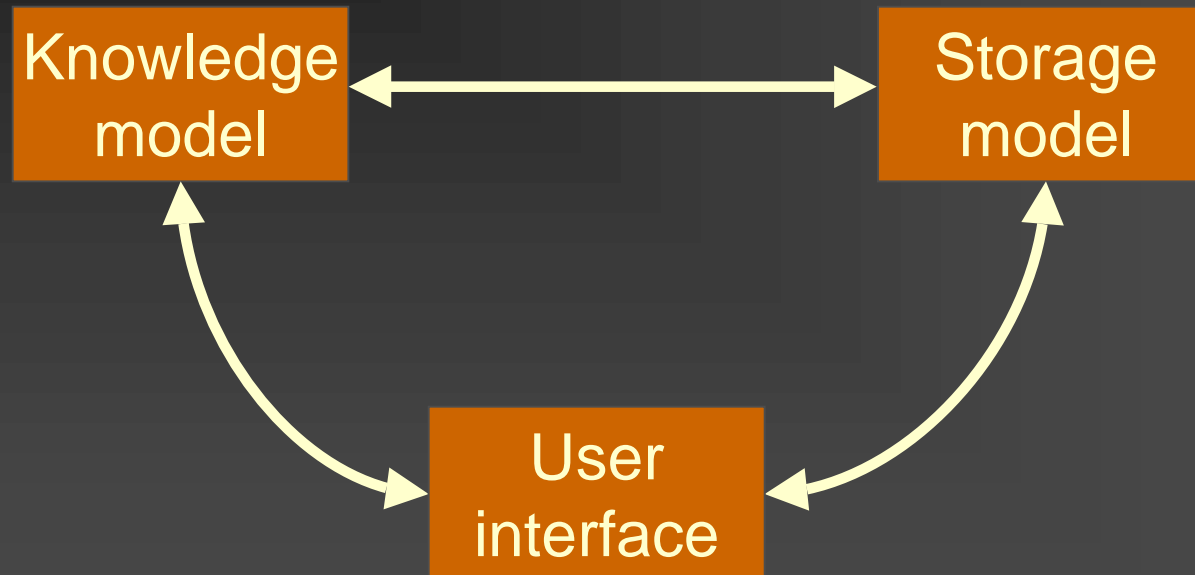
  

Flavor	Tannin Level
STRONG	MODERATE

Sugar: 5

# *Protégé-2000 Architecture*





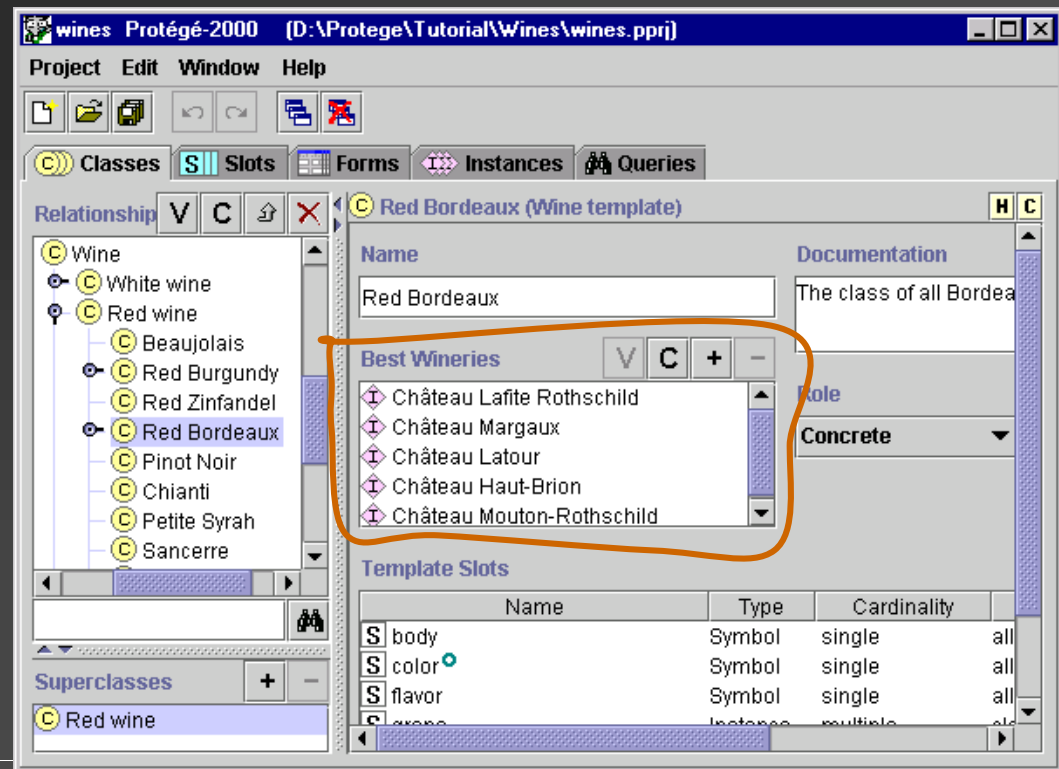
# Changing the Knowledge Model

The screenshot shows the Protégé-2000 interface with the 'wines' project open. The left pane displays a class hierarchy starting with 'Drink' as the superclass, followed by 'Wine', and then various wine types like 'White wine', 'Red wine', 'Beaujolais', etc. The right pane shows the 'Wine' class editor, which is circled in orange. The editor includes fields for 'Name' (Wine), 'Documentation' (A wine class represents all possible wines), and 'Role' (Concrete). Below these is a 'Template Slots' table.

Name	Type	Cardinality	Other Facets
S body	Symbol	single	allowed-values={FULL,MEDIUM,LIGHT}
S color	Symbol	single	allowed-values={RED,ROSÉ,WHITE}
S flavor	Symbol	single	allowed-values={DELICATE,MODERATE,STRONG}
S grape	Instance	multiple	classes={Wine grape}
S maker	Instance	single	classes={Winery}
S name	String	single	
S sugar	Symbol	single	allowed-values={DRY,SWEET,OFF-DRY}

# Changing the Knowledge Model

- Templates for new class-level and slot-level properties
  - metaclasses
  - metaslots



# Changing the Knowledge Model

- Templates for new class-level and slot-level properties
  - metaclasses
  - metaslots

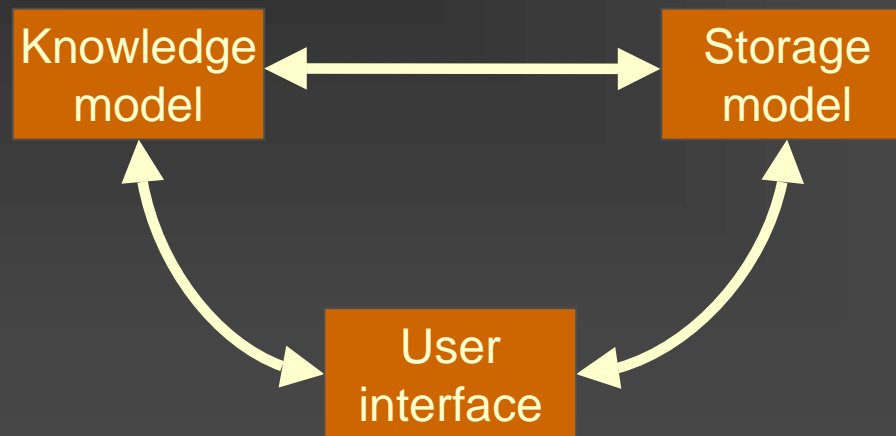
The screenshot shows the Protégé-2000 interface with the following components:

- Project:** wines-oil-food-red-meat Protégé-2000 [D:\Protege\Semantic Web paper\wines-oil-food-red-meat.pprj]
- Menu:** Project Edit Window Help
- Toolbars:** Oil, Classes, Slots, Forms, Instances, Queries
- Relationship Panel:** Shows a class hierarchy starting with :THING, including :SYSTEM-CLASS, Expression, BooleanExpression, OilClass, PropertyRestriction, Axiom, CONSUMABLE-THING, MEAL-COURSE (with subclasses SPICY-RED-MEAT-COURSE, SHELLFISH-COURSE, SEAFOOD-COURSE, OYSTER-SHELLFISH-COURSE), EDIBLE-THING, MEAL, and TASTE.
- Class Editor (SPICY-RED-MEAT-COURSE (OilClass)):**
  - Name:** SPICY-RED-MEAT-COURSE
  - Documentation:** a spicy red-meat course must contain red meat and must contain food that is itself spicy or food containing something that is spicy
  - Role:** Concrete
  - Type:** defined
  - HasPropertyRestriction:** Contains two instances of FOOD.
  - SubClassOf:** Empty.
  - Expressions:** `(has-value FOOD RED-MEAT)`, `(has-value FOOD (or (has-value HAS-TASTE SPICY) (has-value CONTAINS (has-value HAS-TASTE SPICY))))`
  - Template Slots:**

Name	Type	Cardinality	Other Facets
S FOOD	Instance	multiple	classes={EDIBLE-THING}

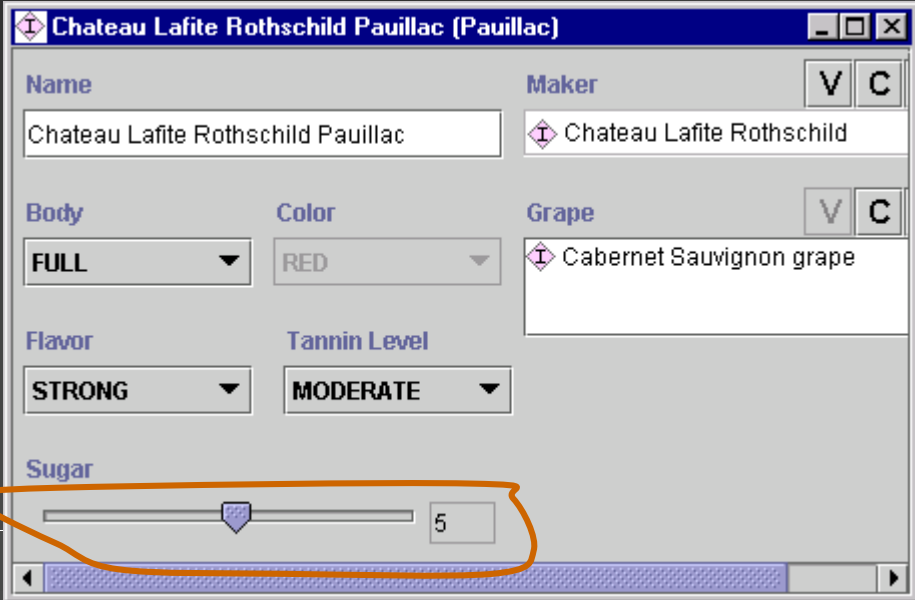
# Protégé-2000 Plugin Architecture

- URIs
- Complex expressions
- Primitive and defined classes
- Transitive, symmetric properties



# Changing The User Interface

- Users can replace any widget on the form with a different one.



The screenshot shows a window titled "Chateau Lafite Rothschild Pauillac (Pauillac)". The form contains several fields:

- Name:** Chateau Lafite Rothschild Pauillac
- Maker:** Chateau Lafite Rothschild
- Body:** FULL
- Color:** RED
- Grape:** Cabernet Sauvignon grape
- Flavor:** STRONG
- Tannin Level:** MODERATE
- Sugar:** A slider control with a value of 5.

An orange oval highlights the Sugar slider control.

# Changing The User Interface

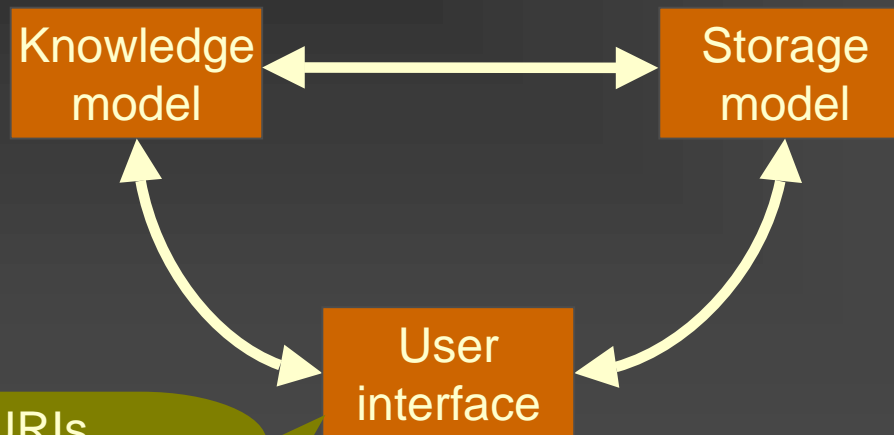
- Users can replace any widget on the form with a different one.

The screenshot shows the Protégé-2000 interface with the class editor for `SPICY-RED-MEAT-COURSE` (OilClass) open. The left pane shows a class hierarchy with `SPICY-RED-MEAT-COURSE` selected. The main editor shows the class details, including the `HasPropertyRestriction` widget, which is highlighted with an orange circle. This widget displays a list of instances (FOOD) and a list of expressions (has-value FOOD RED-MEAT, has-value FOOD, (or (has-value HAS-TASTE SPICY) (has-value CONTAINS (has-value HAS-TASTE SPICY))))). The bottom pane shows a table of instances with columns for Name, Type, Cardinality, and Other Facets.

Name	Type	Cardinality	Other Facets
FOOD	Instance	multiple	classes=(EDIBLE-THING)

# Protégé-2000 Plugin Architecture

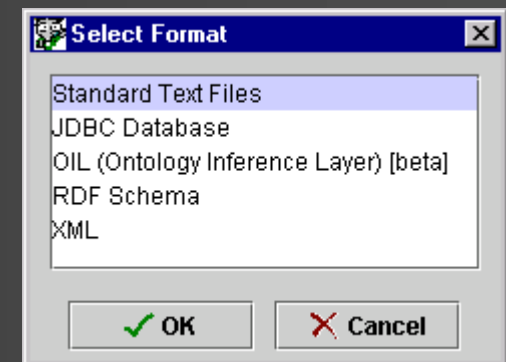
- URIs
- Complex expressions
- Primitive and defined classes
- Transitive, symmetric properties



- Acquire and verify URIs
- Use a structured editor for logical expressions
- Display inferred values for transitive properties

# Changing The Storage Model

- Users can change the output file format - alternative “back ends”
- The back-end code can
  - resolve the remaining differences in the knowledge model,
  - add or remove information,
  - map between Protégé knowledge model and the required knowledge model

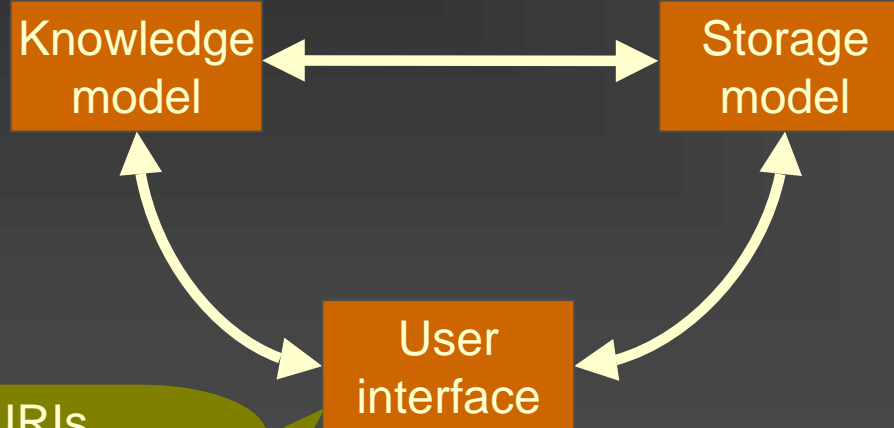




# Protégé-2000 Plugin Architecture

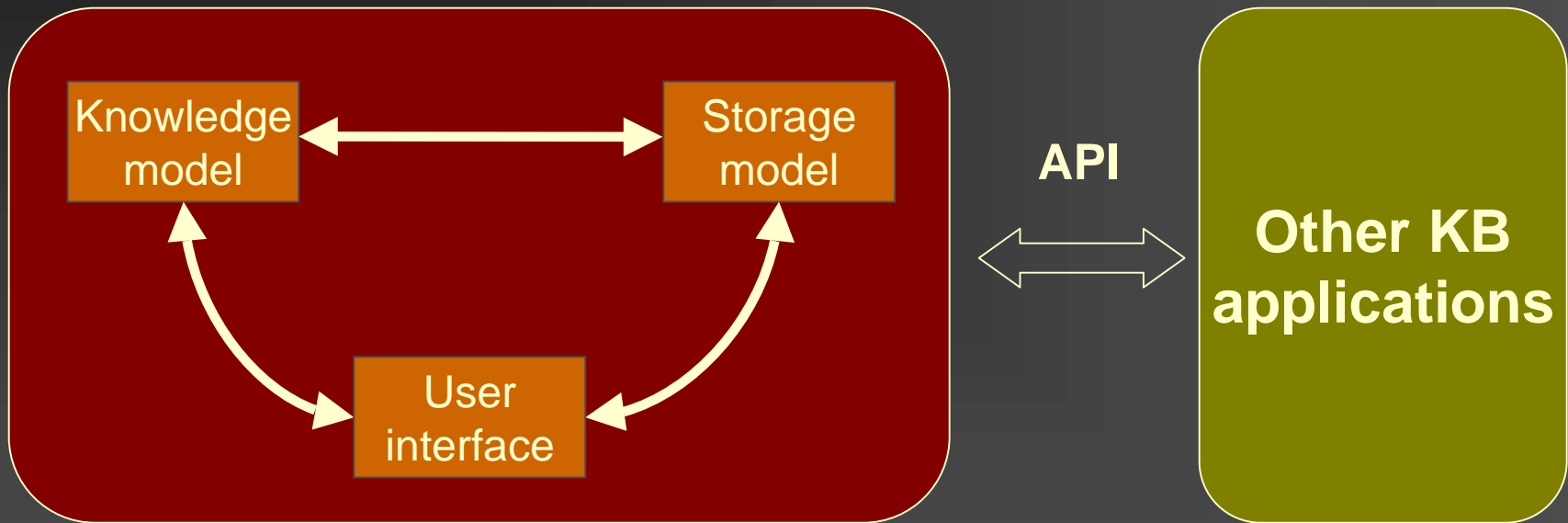
- URIs
- Complex expressions
- Primitive and defined classes
- Transitive, symmetric properties

- RDF Schema
- OIL
- XML
- JDBC database



- Acquire and verify URIs
- Use a structured editor for logical expressions
- Display inferred values for transitive properties

# *Including New Applications*



# Include New Applications

Integration With  
A Description  
Logics Classifier  
(FaCT)

The screenshot shows the Protégé-2000 interface with the following components:

- Project:** wines-oil-food-red-meat Protégé-2000 (D:\Protege\Semantic Web paper\wines-oil-food-re...)
- Menu:** Project Edit Window Help
- Toolbars:** Oil, Classes, Slots, Forms, Instances, Queries
- Relationship Panel:** Superclass view showing a hierarchy:
  - MEAL-COURSE
    - SPICY-RED-MEAT-COURSE
    - SHELLFISH-COURSE
    - SEAFOOD-COURSE
    - OYSTER-SHELLFISH-COURSE
  - EDIBLE-THING
    - SEAFOOD
      - SHELLFISH
        - NON-OYSTER-SHELLFISH
        - OYSTER-SHELLFISH
      - FISH
    - MEAT
    - MEAL
  - TASTE

- Class Editor (Right):** SHELLFISH-COURSE (OilClass)
- Name:** SHELLFISH-COURSE
- Role:** Concrete
- Type:** defined
- HasPropertyRestriction:** FOOD, Expressions: (has-value FOOD SHELLFISH)
- Template Slots:**

Name	Type
S FOOD	Instance

# Using A DL Classifier

The image displays three overlapping screenshots of the Protégé-2000 software interface, illustrating the use of a DL Classifier. The windows show class hierarchies and FaCT interaction logs.

**Left Window:** Shows a class hierarchy with the following structure:

- .THING A
  - :SYSTEM-CLASS A
    - Expression
      - Top
      - Bottom
CONSUMABLE-THING
    - MEAL-COURSE
      - SPICY-RED-MEAT-COURSE
      - SHELLFISH-COURSE
      - SEAFOOD-COURSE
      - OYSTER-SHELLFISH-COURSE
EDIBLE-THING
      - SEAFOOD
        - SHELLFISH
          - NON-OYSTER-SHELLFISH
          - OYSTER-SHELLFISH
FISH

MEAT

- MEAL

TASTE

**Middle Window:** Shows the FaCT Interaction log with the following messages:

```
-> [<PRIMITIVE NAME="CONSUMABLE-THING"/>]
directSupersC(<PRIMITIVE NAME="MEAL-COURSE"/>)
-> [<PRIMITIVE NAME="CONSUMABLE-THING"/>]
directSupersC(<PRIMITIVE NAME="MEAT"/>)
-> [<PRIMITIVE NAME="EDIBLE-THING"/>]
directSupersC(<PRIMITIVE NAME="NON-OYSTER-SHELLFISH"/>)
-> [<PRIMITIVE NAME="SHELLFISH"/>]
directSupersC(<PRIMITIVE NAME="OYSTER-SHELLFISH"/>)
-> [<PRIMITIVE NAME="SHELLFISH"/>]
directSupersC(<PRIMITIVE NAME="OYSTER-SHELLFISH-COURSE"/>)
-> [<PRIMITIVE NAME="SHELLFISH-COURSE"/>]
directSupersC(<PRIMITIVE NAME="RED-MEAT"/>)
-> [<PRIMITIVE NAME="MEAT"/>]
directSupersC(<PRIMITIVE NAME="SALTY"/>)
-> [<PRIMITIVE NAME="SPICY"/>]
directSupersC(<PRIMITIVE NAME="SEAFOOD"/>)
-> [<PRIMITIVE NAME="EDIBLE-THING"/>]
directSupersC(<PRIMITIVE NAME="SEAFOOD-COURSE"/>)
```

**Right Window:** Shows a class hierarchy with the following structure:

- .THING A
  - :SYSTEM-CLASS A
    - Expression
      - Top
CONSUMABLE-THING
    - EDIBLE-THING
      - MEAL
        - MEAL-COURSE
          - SEAFOOD-COURSE
            - SHELLFISH-COURSE
              - OYSTER-SHELLFISH-COURSE
                - Bottom M
SPICY-RED-MEAT-COURSE

TASTE

The Superclasses panel in the right window shows SHELLFISH-COURSE.

# *Other Plugins*

- Diagrammatic knowledge entry
- Ontology visualization
- Ontology merging
- Ontology acquisition from UMLS and WordNet
- Constraint verification

*All these plugins become automatically available for different languages*

# *As A Result, We Get A Tool That*

---

- can be used for ontology development in different (overlapping) representation formalisms
    - translate models from one formalism to another
  - can be easily customized to a new language
    - knowledge model
    - user interface
    - persistent storage
  - can incorporate other applications
-

# *Our Vision*

---

- Complex, distributed systems built from plug-and-play components
- Systems that allow evolution throughout their life cycles via substitution of new components
- Repositories of components for use in new designs and for updating previous applications

*for both ontologies and components  
of knowledge-based systems!*

---

*<http://protege.stanford.edu>*