Application Development with Protégé

Samson Tu
Ray Ferguson

Overview

• Part I - Ray
  • Protégé and Databases
  • Protégé Application Designs
    • API Application Designs
    • Web Application Designs

• Part II - Samson
  • Higher-Level Access to Protégé Knowledge Bases
    • Problem Solving Methods (PSM’s)
    • Reasoning Systems (Algernon, Jess)
    • Scripting Language Interfaces
What does Protégé Do?

Answer: Nothing!

Protégé is a tool.

Allows you to create a model and collect information. Similar to, and just a useless as, a database.

What you probably want is an application that does something useful...

How is Protégé Different from a Database?

- **Emphasis on Model vs. Data**
  - Protege: Model is equal or more interest as data
  - Database: Data is important, model is secondary

- **Emphasis on Expressiveness over Performance**
  - Protégé: Richer modeling language
    - inheritance relationships
    - constraint “overriding”
    - expressing “webs” of relationships
  - Database: Simpler modeling language, optimized for speed
The Misleading Question

Q: What can you do with technology X that you cannot do with related technology Y?
A: (Usually) nothing

Q: What can you do with Protégé that is impossible with a database?
A: Nothing

Q: What can you do with a database that is impossible with a file?
A: Nothing

Q: What can you do with Java that is impossible with assembly language?
A: Nothing

Phrasing the question as "possible vs. impossible" leads nowhere.

The Real Question

When is it easier, clearer, more straightforward to use X instead of Y?

Preferable to have direct rather than simulated support for desired features.
- Simulation reduces clarity and portability
- Some simulation may be necessary, but the less the better

Protégé might be better than a database when:
- Model consists of rich data, with many relationships that are often traversed.
- Requirements and application design are changing and not clearly specified.
- Protege is a good exploratory and experimentation environment.
- Quick iterations are possible between model, data, and application changes.

Oversimplified Answer:
- Simple, flat, fixed model, speed paramount -> Database
- Complex, network-like, changing model with concept hierarchies -> Protégé
It doesn’t have to be Either/Or

- Construct model in Protégé
- Initial implementations with Protégé
- Iterate until requirements/design is firm, initial data is input
- Generate database schema from Protégé model and populate database with Protégé instances

Application Designs

- An Application Design that Doesn’t Work

- Applications Designs that Do Work
  - API level Application designs
  - Web Application designs
An Application Design that Doesn’t Work

- Idea:
  1. Create Protégé Project with database backend
  2. Create the classes and instances
  3. Access the database tables directly with other applications

- Database tables are designed and optimized to work with a particular application in mind.
  - The Protégé database table was designed with the Protégé application in mind
  - The Protégé database table was NOT designed with your application in mind

- Instead access the data though the Protégé API.

Protege API Applications

- Tab as application
- Standalone Application
Protégé Tab as An Application

- Description
  - Create a custom tab plugin
  - Configure Protégé to just display your tab
- Pros
  - Simple
  - Great for few users
  - Iteration (change of model, data, app) is very easy
- Cons
  - Protégé must be installed
  - Difficult to permanently disable standard functions
  - Stuck with Protégé menus, toolbar, etc
  - No security on underlying model and data
  - User really should know something about Protégé

Standalone Application

- Description
  - Write standalone Java Application
  - Call into the Protégé API for knowledge base access
  - Often evolves from a Tab
- Pros
  - No need to install Protégé
  - User doesn’t need to know anything about Protégé
  - Underlying model and data are as secure as you want
  - Can use some or none of the Protégé UI, as desired
    - Forms for classes and instances are available
    - Some tabs will work
- Cons
  - Iteration somewhat more difficult than as Tab
Standalone Application Example

For code see:
http://protege.stanford.edu/conference/2005/slides

Protégé Over the Net

- Applets
- Java WebStart
- Servlets and Java Server Pages
- Protégé RMI server
- Custom server
Applets

- Applets are a standard Web Browser (Internet Explorer, Firefox) plugin for running Java programs inside a browser.
- By default the application runs in a “sandbox”
  - No file system access
- Requires no “application” installation.
- Requires one installation of correct Java version
- Application is only available by going to a web page – no offline capabilities

Example: Protégé Web site demos

Java WebStart

- WebStart is a standard Java mechanism for installing and running Java programs on a client.
- Application is “automatically” installed and started when the user hits a URL.
- Improvement on Applets:
  - Handles Java VM updates
  - Handles application updates
  - Allows off line execution
  - Allows application execution without starting browser
Servlets and Java Server Pages (JSP)

- Servlets are web server plugins written in Java.
  - Called by accessing a particular URL.
  - Control the design and content of the page sent to the caller’s browser
- JSP are code written in a “java-like-language” embedded in a web page. This code can make calls to the web server and typically control the design and/or content of part of the page.

Typically servlets (directly) and JSP’s (indirectly) call into the Protégé API to access knowledge base elements and use this information to influence the design and content of web pages.

Example: “Protege Web Browser”

Remote Method Invocation (RMI) Server

- Standard Java remote procedure call mechanism.
- Used by the Protégé multi-user client.
- Provides programmatic access to Protégé API across the web.
- No need to export project access or database access

Example: Protégé Multiuser Client/Server system.
Custom Server

- Wrap the Protégé API (or the part that you want to export) with your own API and then make it available with whatever network protocol you like.

Example: Protégé CORBA Server

Summary (of Part I)

- Protégé and Databases both have their places
- Standalone applications are easily built on Protégé
  - Using only knowledge base
  - Using also some/all of the Protégé UI
- Web applications are built on top of Protégé in variety of ways
Application Development: Part II

Samson Tu, Ray Ferguson
Stanford Medical Informatics
Stanford University

8th International Protégé Conference
Madrid, Spain, July 2005

With thanks to Monica Crubezy and Olivier Dameron for lending their slides

Overview

• Part I - Ray
  • Protégé and Databases
  • Protégé Application Designs
    • API Application Designs
    • Web Application Designs

• Part II - Samson
  • Higher-Level Access to Protégé Knowledge Bases
    • Problem Solving Methods (PSM’s)
    • Reasoning Systems (Algernon, Jess)
    • Scripting Language Interfaces
Recap: Application Development Architecture

- Protégé knowledge base can be exported to database
- Protégé applications can take different forms
  - Stand-alone application
  - Tab plugin
  - Web-based
    - Applets
    - Java WebStart
    - Servlets and Java Server Pages
    - Protégé RMI server
    - Custom server

Application Development Technology
Problem-Solving Methods (PSMs)

- Standard, explicit algorithms that address stereotypical (Artificial Intelligence) tasks
  - Design, classification, diagnosis, planning
- Domain-independent components that abstract the reasoning process from factual knowledge
  - Reusable for different applications and domains
- Collected and indexed in libraries for reuse

Heuristic classification in MYCIN

- Feature Abstraction
  - Compromised host
  - Immunosuppressed
    - Leukopenia
    - WBC < 2.5
  - Alcoholic
- Solution Refinement
  - Gram-negative infection
  - Pseudomonas
  - E. coli

(After Clancey)
The Heuristic Classification PSM

Heuristic classification for food & wines

(After KSL’s Wine Agent)
To use PSM: Mapping domain to PSM I/O ontology

**Method**

Input Ontology

- observables
- solution space
- abstractors

**Method Output Ontology**

- solution

**Mapping**

- (solution TO wine recommendation)

**Domain Ontology**

- (flavor, color, alcohol content, wine)

Mapping

- (flavor, color, alcohol content TO observables)

Conceptual and syntactic mismatch

<table>
<thead>
<tr>
<th>Conceptual and syntactic mismatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notion of a “Desired features of wine”</td>
</tr>
</tbody>
</table>

Notion of an “Input Case”

<table>
<thead>
<tr>
<th>Symbol</th>
<th>allowed-values=(ADMISSIBLE,OPTIMAL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td>number-of-results</td>
<td>Integer</td>
</tr>
<tr>
<td>observable-list</td>
<td>instance of observable</td>
</tr>
<tr>
<td>space</td>
<td>instance of solution-space</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>allowed-values=(POSITIVE,COMPLETE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
</tr>
<tr>
<td>number-of-results</td>
<td>Integer</td>
</tr>
<tr>
<td>observable-list</td>
<td>instance of observable</td>
</tr>
<tr>
<td>space</td>
<td>instance of solution-space</td>
</tr>
</tbody>
</table>
Specify mappings as a Protégé knowledge base

Each mapping is itself an instance of an ontology of possible mapping types

Mapping Ontology
(flavor, color, area
TO observables)

Domain Ontology
(flavor, color, alcohol content, wine)

Problem-Solving Method

Mapping Ontology
(solution TO wine recommendation)

Method Input Ontology
(observables, solution space, abstractors)

Method Output Ontology
(solution)

Domain Knowledge

Mapping Interpreter

PSM

Input-output Ontology

Domain Ontology

Mapping Ontology
(instances of mapping relations)

PSM input instances

Domain instances
Mapping relations

- Each instance of the PSM input class is calculated from an instance of the domain class
- The slot values of the PSM instance are computed according to slot-mapping expressions that involve the domain instance’s slot values

Instance mapping

Slot mappings

Mapping Wine recommendation query to Input case

Renaming slot mapping

- Number of recommended wines
- Number of recommendations
- Number of results

Slot mappings
- Wine preferences space
- Simple properties space
- Number of recommended wines
- Recommendation coverage
- Number of recommendations space
- Class reference
Slot mappings

- The target slot (tX)
- The slot-value computation expression, possibly involving source slots (s1)
  - local access to (sub)instance slot values:
    *<s1.s11>*
- Different types of slot mappings:
  - renaming: value(tA) = value(s1)
  - constant: value(tC) = constant
  - lexical: value(tB) = “*<s2>* / 20*<s3>*”
  - functional: value(tC) = function()
  - recursive: value(tA) = instance (w/ auxiliary mapping)

Recursive slot mapping

Source Ontology

Class S
slot s1 (S’)

Class S’
slot s’1

Target Ontology

Class T
slot tA (T’)

Class T’
slot t’A slot t’B

Instance mapping

recursive slot mapping

on-demand

Instance mapping

slot mappings
Mapping ontology

- Conditional mapping
  - filtering of source instances
  - one-to-many instance mapping
- Propagation of mappings to source subclasses
- Some degree of many-to-one instance mapping
- Many source KBs to one target KB mapping
- TCL and Python scripting for conditions, functional mappings, and other code
- Hook-ups for custom code
  - at the global level: initialization & cleaning
  - at the instance level: before/after all/each mapping

Results of mapping interpretation

“Wine recommendation query” instance

<table>
<thead>
<tr>
<th>Name</th>
<th>Tannin Level</th>
<th>Area</th>
<th>Alcohol Content</th>
<th>Alcohol Strength</th>
<th>Flavor</th>
<th>Body</th>
<th>Sugar</th>
<th>Color</th>
<th>Mixer</th>
<th>Wine Preferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>query 1</td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Input case” instance

<table>
<thead>
<tr>
<th>Name</th>
<th>Space</th>
<th>Observable-list</th>
</tr>
</thead>
<tbody>
<tr>
<td>input-query 1</td>
<td>Wine</td>
<td>body = MEDIUM</td>
</tr>
</tbody>
</table>

- body = MEDIUM
- color = ROSE
- flavor = DELICATE
- sugar = DRY
- alcohol content = 12

Coverage-type: COMPLETE
Classification-type: OPTIMAL

Number of Recommendations: 5
Number of results: 5
PSM execution support (Demonstration)

What PSMs are available?

- Protégé’s PSM library
  - Heuristic classification
  - Propose and revise
    - Task: Design what satisfy constraints
    - Method: explicit algorithm of iterative constraint-satisfaction problem solving
    - Input: parameters, constraints, fixes
    - Output: valid design
    - Sample problems
      - Elevator design
      - Ribosome structure given NMR data
  - Literature: CommonKADs library

Concluding remarks on PSMs

• Benefits of the PSM approach
  • Clear, systematic paradigm for modeling & annotating methods
  • Support for browsing, selecting, configuring & executing methods
  • Framework for empirical experiments, comparison & reuse

• Future of PSMs
  • Organization of large-scale libraries of distributed PSMs
  • Sharing of scientific data processing methods
  • Framework for Semantic Web Services

Application Development: Technology to Use

Protégé Knowledge Base

- Java
- Problem-Solving Methods
- Rule-Based Systems (JESS, Algernon)
- Scripting Languages

Client
High-level programming tools

- Programming paradigms that have been made interoperable with Protégé
  - JessTab, Algernon: Rule-based programming
  - Prolog tab: Logic-based programming
  - Protégé Script Console, JessTab, Algernon: Scripting environment

- Uses
  - Programmatically modification of Protégé KB
  - Protégé extender
    - e.g., query, enforce relationships
  - Application development

Java Expert System Shell (JESS)

- Developed at Sandia National Laboratory
  - http://herzberg.ca.sandia.gov/jess/
  - Free licensing for non-profit organizations
  - Well supported, active user community

- Features
  - Forward-chaining rule engine
    - matches antecedent facts
    - performs consequent actions
  - Interpretive scripting language
  - Integration with Java
Jess basics

- Collection of facts that can be constrained by templates
  
  (used-car (make toyota) (price 2000)
           (mileage 12000))

- Rules that performs actions based on patterns of existing facts (forward chaining)
  
  (defrule might-buy-car
   ?candidate <- (used-car (mileage ?m : (< ?m 5000)))
   => (assert (candidate ?candidate)))

- Functions for procedural programming
  
  (deffunction greaterThan5 (?x) (return (> ?x 5)))

- Ability to call Java methods from within JESS
  
  (call ?instance getDirectType)
  
  - Allow invocation of Protégé API calls from within JESS

JessTab extensions to Jess and Protégé

- Jess console window in Protégé
- Functions for knowledge-base operations
- Mapping between Jess and Protégé
  
  - Protégé classes mapped to a Jess fact template
  - Protégé instances mapped to Jess facts and Jess facts mapped to instances
  
  - Changes to mapped facts in Jess reflected in Protégé; changes in Protégé reflected in Jess
Jess console window in Protégé

Defining classes and instantiating them

```
Jess> (defclass Person (is-a :THING)
    (slot name (type string))
    (slot age (type integer)))
TRUE
Jess> (make-instance john of Person (name "John") (age 20))
<External-Address:SimpleInstance>
Jess> (mapclass Person)
Person
Jess> (facts)
f-0 (object (is-a Person) (is-a-name "Person")
    (OBJECT <External-Address:SimpleInstance>)
    (age 20) (name "John")
For a total of 1 facts.
```
Modifying slots

```plaintext
Jess> (slot-set john age 21)
Jess> (facts)
f-1 (object (is-a Person) (is-a-name "Person")
(OBJECT <External-Address:SimpleInstance>)
(age 21) (name "John"))
For a total of 1 facts.
```

Creating a second instance

```plaintext
Jess> (make-instance sue of Person (name "Sue") (age 22))
Jess> (facts)
f-1 (object (is-a Person) (is-a-name "Person")
(OBJECT <External-Address:SimpleInstance>)
(age 21) (name "John"))
f-4 (object (is-a Person) (is-a-name "Person")
(OBJECT <External-Address:SimpleInstance>)
(age 22) (name "Sue"))
For a total of 2 facts.
```
Adding a Jess rule

```jess
(defrule twentyone
  (object (is-a Person)
    (name ?n) (age ?a&:(> ?a 21)))
=>
  (printout t "The person " ?n
    " is 21 or older" crlf))
TRUE
Jess> (run)
The person John is 21 or older
The person Sue is 21 or older
2
Jess>
```

JessTab as Protégé extender: Query

- JessTab implements a set of query functions e.g., `(find-all-instances instance-sets query)`

```jess
(deffunction findApplicableGuideline(?pid)
  (return
   (find-all-instances ((?g Guideline))
     (hasApplicableContext ?g ?pid))))
```

```jess
(deffunction hasApplicableContext (…)
```

![JessTab as Protégé extender: Query](image)
JessTab as Protégé extender: Enforcing relationships

- Circumference of a circle = 3.14 * 2 * radius
  (defrule computecircumference
   ?circle <- (object (is-a Circle)(radius ?r&~nil)(circumference nil))
   => (slot-set ?circle circumference (* 3.14 2 ?r))
  )
  (defrule unsetcircumference (object (is-a Circle)(radius nil)(circumference ?c&~nil)(OBJECT ?obj)) => (slot-unset ?obj "circumference")
  )

- Run rules in the background
  (reset)
  (run-until-halt)

Circumference of circle automatically calculated from radius
JessTab as application development tool

- Knowledge base: Algorithm for managing upper respiratory infection

(disclaimer: tutorial example, no medical content)

Jess code to execute the clinical algorithm

- In CommonColdGuideline project, load and run application
- Jess> (batch d:/_ShortCourse/projects/executionEngine.clp) TRUE
- Jess> (runCase Assissi)
- Jess> (runCase "Assissi")
- Processing Context presentation of symptoms
- Processing Inquiry Assess symptoms
- Does patient have symptoms of other illness? (answer for for case Assissi)(yes or no) yes
- Is patient a smoker? (answer for for case Assissi)(yes or no) yes
- Does patient have asthma? (answer for for case Assissi)(yes or no) no
- Does patient have inhalant allergy? (answer for for case Assissi)(yes or no) no
- Processing Decision Home care?
- Processing OrderIntervention Referral
- **Recommendation**: order Referral
Algernon

- Developed by Micheal Hewitt (mhewett@users.sourceforge.net)
  - Under active development
- Features
  - Inference engine with interleaving of forward and backward chaining
  - Operate directly on Protégé frames
  - Access to multiple concurrent KBs
  - Interpretive scripting languages
    - Lisp
    - Unix shell commands
  - Integration with Java

Algernon Protégé console window
Algernon basics

- A path that is a sequence of clauses: (relation frame value), where relation may be a slot or an Algernon macro
  
  
  
  ```
  (date FluorCold_Instance_0 "2005/06/12")
  ```

- Variables that can be bound with values as a result of processing queries or of explicit assignment
  
  ```
  "Find labels (?z) of instances (?y) of subclasses (?x) of Task"
  ```

- Macros provide built-in relations and perform actions
  
  ```
  (ADD-CLASS (?x Concept)(:NAME ?x AlgernonConcept))
  ```

Adding a hierarchy of classes

```
(:TAXONOMY (:THING
  (Plants
    (FloweringPlants
      (Roses)
      (Begonias Moms-Begonia-1)
      (Tulips Tulip-1 Tulip-2 Tulip-3))
    (Animals
      (Reptiles
        (Alligators)
        (Turtles)))
  )))
```
Algernon rules

- Supposed we have a Component class with location, ordered, shipped, status, and last-update slots

Forward chaining rules

- If a component is ordered, set its status as 'Reserved'.

\[
\text{(ordered } ?x \text{ ?date) -> (status } ?x \text{ Reserved)}
\]

\[
\text{(last-update } ?x \text{ ?date)}
\]
Backward chaining rules

- A component is onsite unless it has been sold.

\[
\text{((location } ?x \text{ ONSITE) } \leftarrow \text{ (FAIL (status } ?x \text{ Sold)))}
\]

Query: Is \((\text{status component-1 Sold})\)?

Supposed \((\text{location component-1 ONSITE})\) is true, then conclude \((\text{status component-1 Sold})\) is false

Very good Algernon Tutorial:
http://algernon-j.sourceforge.net/tutorial/
Application Development: Technology to Use

Protégé Knowledge Base

Java
Problem-Solving Methods
Rule-Based Systems (JESS, Algernon)
Scripting Languages

Client

Context / Problems

- Repetitive tasks
  ex: creation of lateralized concepts and their relationships
- Enumerations
  ex: Ribs
- Dependencies between concepts or relationships
  ex: Thorax / Skin of Thorax
- Ontology maintenance
  require ad hoc detection and fixing
Objective: Scripting environment for Protégé

- Create macros
  - repetitive and error-prone tasks
  - formalism for handling intrinsic complexity
  - towards more abstraction
- Code reuse
- User-friendly and powerful
  - simple and intuitive syntax
  - well formalised

Architecture

- Principle
  - Python interpreter in Java: Jython
  - Thread (share address space with Protégé)
- Shared variable: kb
- Compatibility with frames and OWL
  - instance of KnowledgeBase (Frames)
  - instance of OWLKnowledgeBase (OWL)
Architecture

1. Python Code
2. Python Console
3. Jython
4. Protégé

Frames

- Get frame's attributes
- Create frame
- Create instances
• Get classes' attributes
• Create class
• Create relations

OWL
Example of using script to maintain knowledge base: Repetitive tasks

- Creation of a lateraled 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

- 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
Repetitive tasks

- LeftThumb
- LeftIndex
- LeftMiddleFinger
- LeftRingFinger
- LeftLittleFinger

...are LeftFinger

Ontology maintenance

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

After classification:
- LeftThumb
- LeftIndex
- LeftMiddleFinger
- LeftRingFinger
- LeftLittleFinger
Conclusion

- Direct calls to the Protégé API => no limitations
- Jython => power of Python + Java
- Code reuse allow to hide the low-level Protégé API

- ProtegeScript is useful :-)
  - higher level functions
  - from extensional to intentional description