

Ontology Design and Software Technology

Protege and Java, UML & Model-Driven Architecture



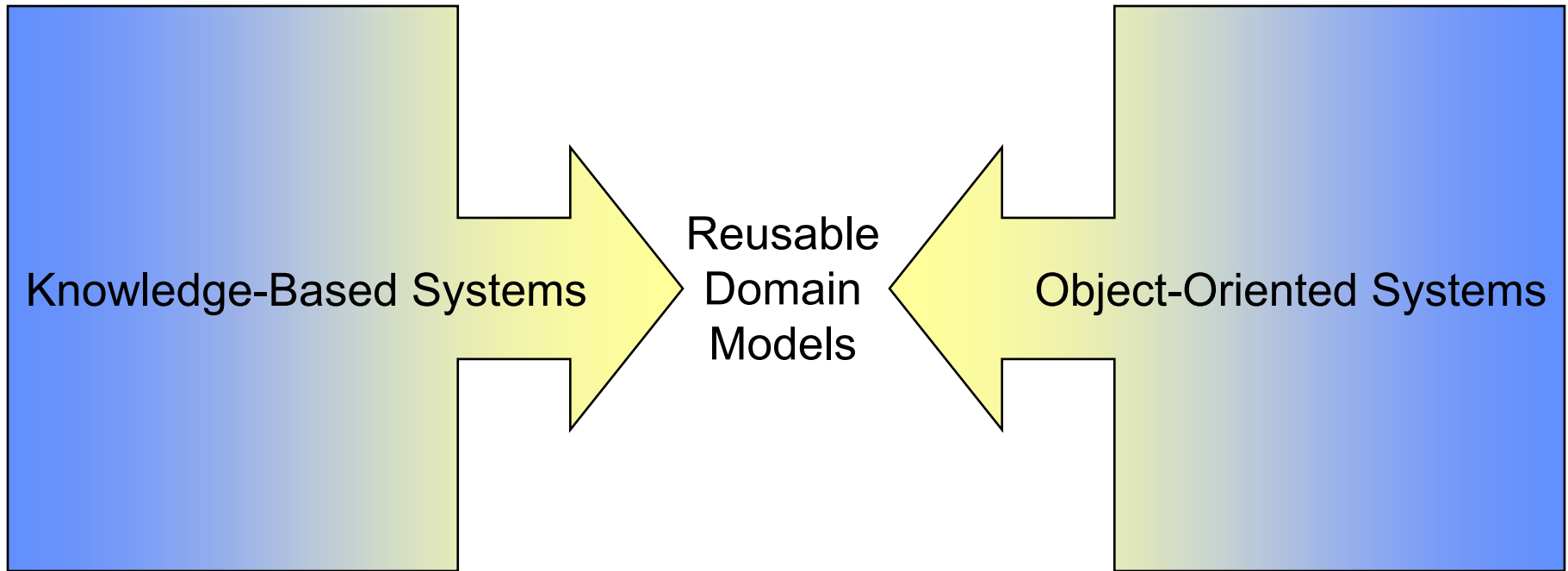
Holger Knublauch

holger@smi.stanford.edu

Colloquium - Stanford Medical Informatics

June 12, 2003

Between Ontology Design and Software Technology

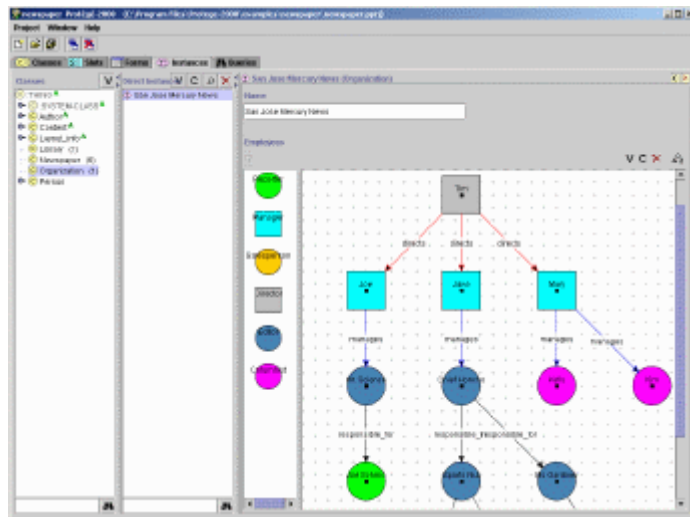


Example Scenario of a Knowledge-Based System

- Clinical Guideline: Cookbook for diagnosis and treatment processes
- Goal:
 - Application that assists clinical staff in following a Clinical Guideline
 - "Expert system" (I observe the following, what could be the cause?)
 - Watchdog mechanisms (The blood pressure has not been checked yet!)
 - Automatic transmission of patient and process data between staff
- Foundation:
 - Correct formal model of the Clinical Guideline

Example Scenario of a Knowledge-Based System

- Knowledge modeling tools like Protege help build these formal models
 - Structure knowledge
 - Easy to use editor for concepts and relationships
 - Automatically generated forms and graphs to acquire instances
 - Constraints to ensure consistency
 - Logical statements to make implicit knowledge explicit



The screenshot shows the Protege 2000 interface with a query window. The query is: (findall ?employee (exists ?editor (and (responsible_for ?editor ?employee) (> (number-of-slot-values (responsible_for ?editor) 1))))). The description is: All employees who are in a team (i.e., whose responsible editor is responsible for more than 1 employee).

Query	Query Responses
three-articles-not-written-by-person	7employees
editors highly paid	Mr. Science
employees paid more than their editor	Sports Nut
employees-in-team	Ms. Gardiner
unsupervised employees	Larry Tennis-rut
	Anne BasketballHead

Example Scenario of a Knowledge-Based System

- So now we got that correct formal model... what next?
- Knowledge model must be integrated into executable system
- Basically a mapping task:
 - Structural knowledge → Software architecture
 - Process knowledge → Communication infrastructure
 - Medical knowledge → Terminology / Classes
 - Diagnostic knowledge → Inference engine
 - Action knowledge → Procedural code
- Knowledge modeling tools are not sufficient for these tasks

Modeling Tools

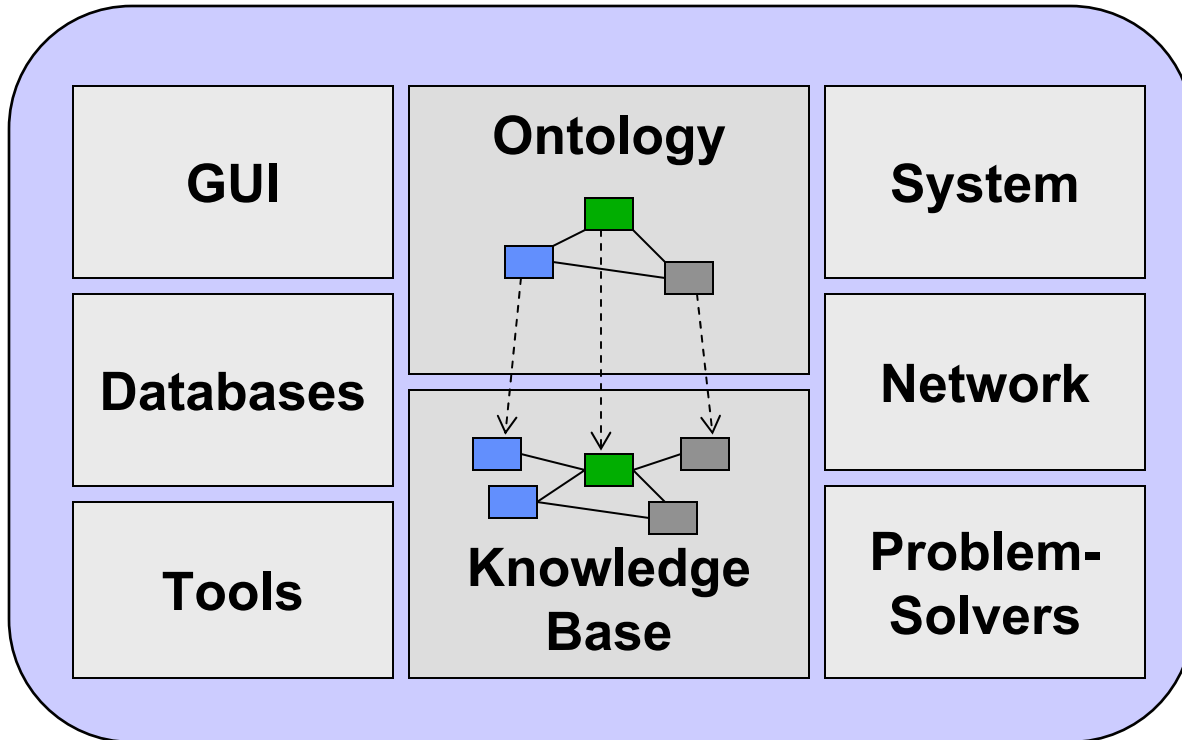
- “Knowledge modeling” tools are academic niche products
- Few of them are used in real-world systems
 - Protégé: 10,000 users
- Different terminology (slot \approx attribute, ontology \approx domain model)

- Mainstream “modeling” tools are
 - Visio/Powerpoint: Millions of users
 - Database tools (e.g., Access/Visual Basic)
 - UML tools (Poseidon: 300,000 users, Together/J)
 - IDE’s (IntelliJ, Eclipse, NetBeans, JBuilder)

- Most projects that we call “knowledge based” are developed with standard tools
- Many projects use tools like Protégé for early requirements and analysis activities only

Ontologies in Real-World Applications

Knowledge model is only one part of a greater system



Meanwhile...

- Mainstream software community recognizes that reusability of object-oriented models is very limited
 - Models are not build on the right level of abstraction
 - Too much focused on specific application
- Trend to increase the complexity of technologies
 - Enterprise JavaBeans
 - SAP / PeopleSoft
- Trend to increase the semantics of object-oriented systems
 - Technology neutrality
 - UML 2.0
 - Executable UML
 - Model-Driven Architecture
 - Increased interest in ontologies

Ontologies in Mainstream Software Technology

- Communication with domain experts
 - Formal reasoning / constraint checking
 - Requirements engineering
 - Domain modeling / alternative to UML
 - Model reuse / sharing / merging
 - Populating data bases (instances are usually not modeled in SE)

 - Model-Oriented Software Development
 - Integrate ontology design into mainstream software projects
 - Richer semantics of models
 - Better model exchange and reuse
 - Escaping the AI niche
- Integrated Development Processes
need Integrated Tools and Languages**

Accessing Protege Ontologies in Java Programs

```
Cls articleCls = kb.getCls("Article");
Iterator articles = articleCls.getInstances().iterator();
while (articles.hasNext()) {
    Instance article = (Instance) articles.next();
    String title = (String) article.getOwnSlotValue(kb.getSlot("title"));
    System.out.println("- " + title);
    Collection topics = article.getOwnSlotValues(kb.getSlot("topics"));
    Iterator it = topics.iterator();
    while(it.hasNext()) {
        Cls topic = (Cls) it.next();
        System.out.println("    topic: " + topic.getName());
    }
}
```

- Weak link between Protege objects and Java objects
- Error-prone references by name
- Parallel model evolution / model drift
- Error messages only at run-time

Accessing Protege Ontologies in Java Programs

```
Cls articleCls = kb.getArticleCls();
Iterator articles = articleCls.getInstances().iterator();
while (articles.hasNext()) {
    Article article = (Article) articles.next();
    String title = article.getTitle();
    System.out.println("- " + title);
    Collection topics = article.getTopics();
    Iterator it = topics.iterator();
    while(it.hasNext()) {
        Cls topic = (Cls) it.next();
        System.out.println("    topic: " + topic.getName());
    }
}
```

- Error messages at compile-time
- Needed:
 - Code generators such as JSave
 - Generator of ontology "loader" classes

BeanGenerator

The screenshot shows the Protégé-2000 ontology editor interface. The main window displays the 'IdentifyParty' class definition. The left pane shows a hierarchical tree of classes, with 'IdentifyParty' selected. The right pane shows the class details, including its name, role, and template slots.

Relationship V C X

IdentifyParty C X

Name
IdentifyParty

Role
Concrete

Template Slots

Name	Type	Ca
S licenseplate	String	require
S policynumber	String	require

Superclasses + -

```
IdentifyParty
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import java.io.Serializable;
import jade.content.*;
import jade.core.*;
import jade.util.leap.*;

public class IdentifyParty implements AgentAction, Seri
    public String getLicenseplate() {
        return this.licenseplate;
    }
    public String getPolicynumber() {
        return this.policynumber;
    }
    public void setLicenseplate(String value) {
        pcs.firePropertyChange("licenseplate", (this.license
        this.licenseplate = value;
    }
    public void setPolicynumber(String value) {
        pcs.firePropertyChange("policynumber", (this.policyn
        this.policynumber = value;
    }
    public void addPropertyChangeListener(PropertyChangeLi
        pcs.addPropertyChangeListener(pcl);
    }
    public void removePropertyChangeListener(PropertyChan
        pcs.removePropertyChangeListener(pcl);
    }
    protected PropertyChangeSupport pcs = new PropertyChar
    private String licenseplate;
```

Ontology to Implementation Language Mapping

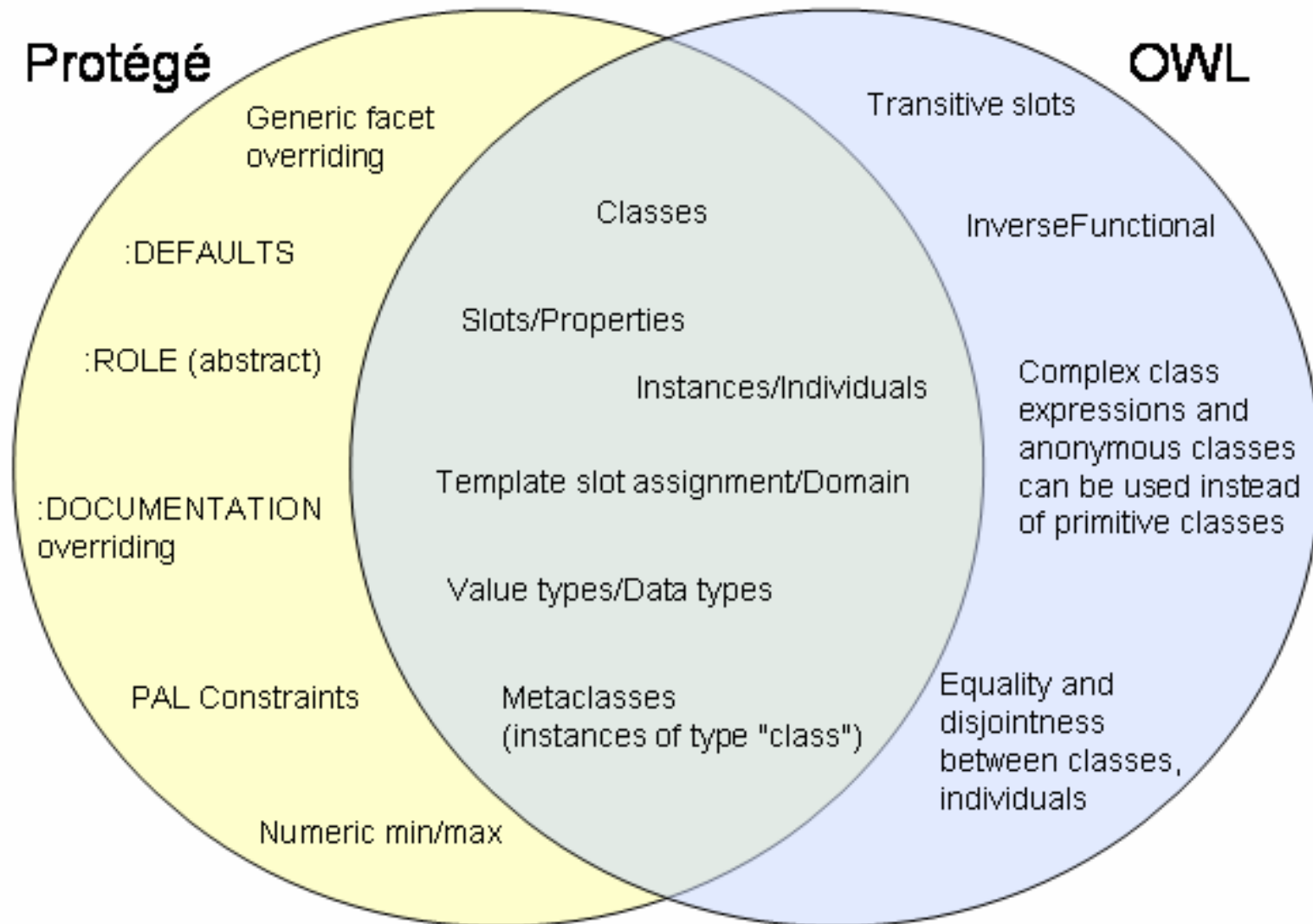
- A step in the right direction
- Protege plus better Java mapping would be very valuable

- But low-level mapping
- Enforces direct correspondence between knowledge model and object-oriented concepts
- No real round-tripping possible (lack of tool integration)

Modeling Language Space



Comparison: Protégé vs. OWL



[Advertisement: Protégé/OWL Edition coming July 7]

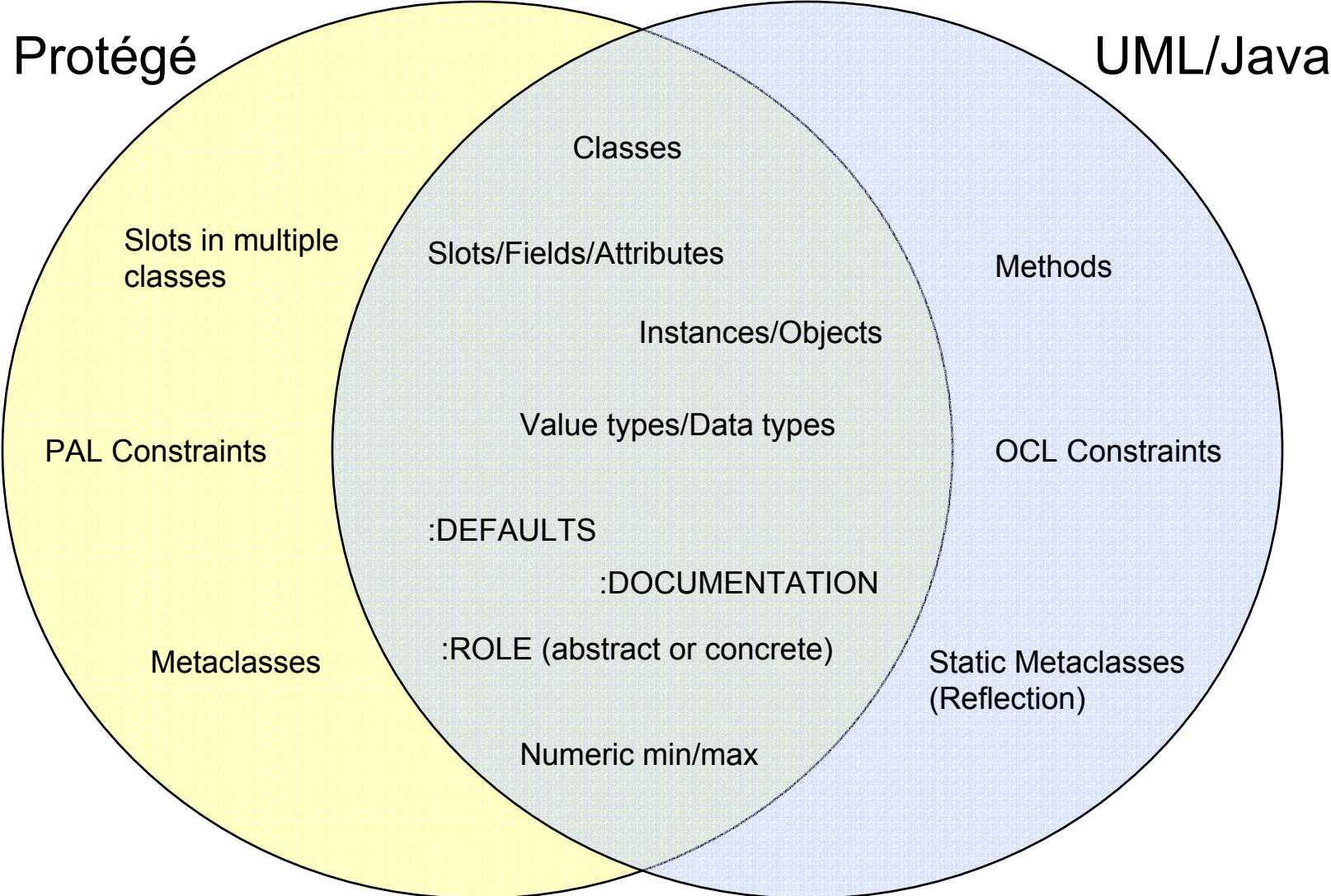
The screenshot displays the Protégé OWL Editor interface for a project named 'OWL-Person Protégé'. The main window shows the configuration for the 'HappyPerson' class, which is of type 'OWL-NAMED-CLASS'. The interface is divided into several panes:

- Left Pane (Class Hierarchy):** Shows a tree view of classes. The hierarchy is: `THING` (Axiom) \supset `SYSTEM-CLASS` (Axiom) \supset `Person` (M) \supset `MalePerson` (M), `PostDoc` (C), `Professor` (C), `HappyPerson` (M), `RichPerson` (C), `Gender` (C), `Color` (C). The 'HappyPerson' class is currently selected.
- Top Pane (Class Properties):** Shows the 'Name' field set to 'HappyPerson', a 'Role' dropdown set to 'Concrete', and empty fields for 'Documentation' and 'Constraints'.
- Restrictions Pane:** Contains a table of restrictions for the 'HappyPerson' class.

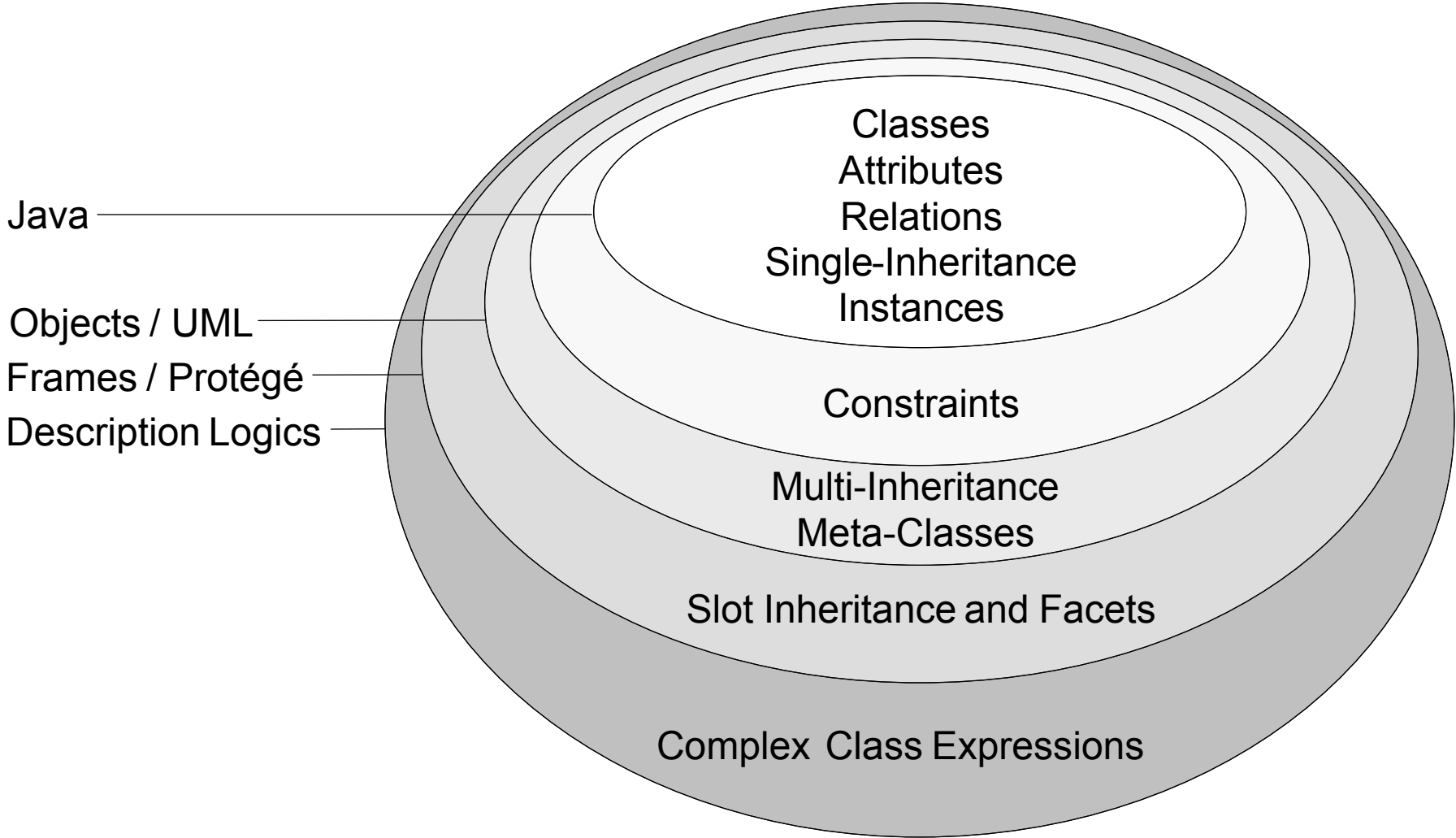
Slot	Restriction	Filler
children	minCardinality	2
children	allValuesFrom	MalePerson \sqcap (rich \supset true) \sqcap Pe
children	maxCardinality	
eyeColor	allValuesFrom	
- Bottom Left Pane (Superclasses):** Shows a list of superclasses with the expression 'Person'.
- Bottom Middle Pane (Equivalent classes):** Shows a list of equivalent classes with the expression 'PostDoc \sqcap (\exists children \neg (rich \supset true))'.
- Bottom Right Pane (Disjoint classes):** Shows a list of disjoint classes with the expression 'Professor'.

An error dialog box is visible in the foreground, displaying the message: "Error: Class or slot name expected at 'Pe'".

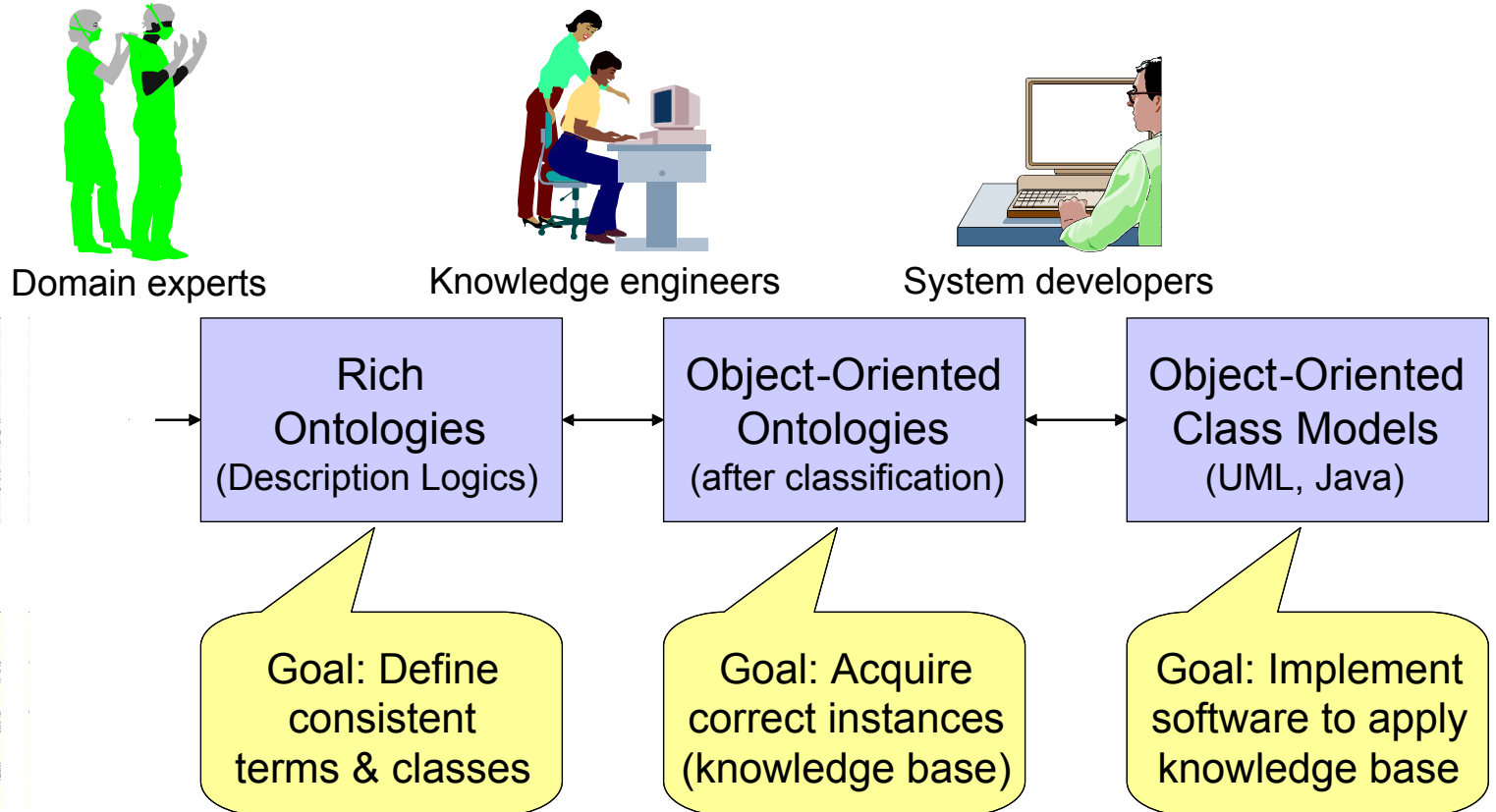
Protégé vs. Object-Oriented Languages



Extended Declarative Power of Ontology Languages

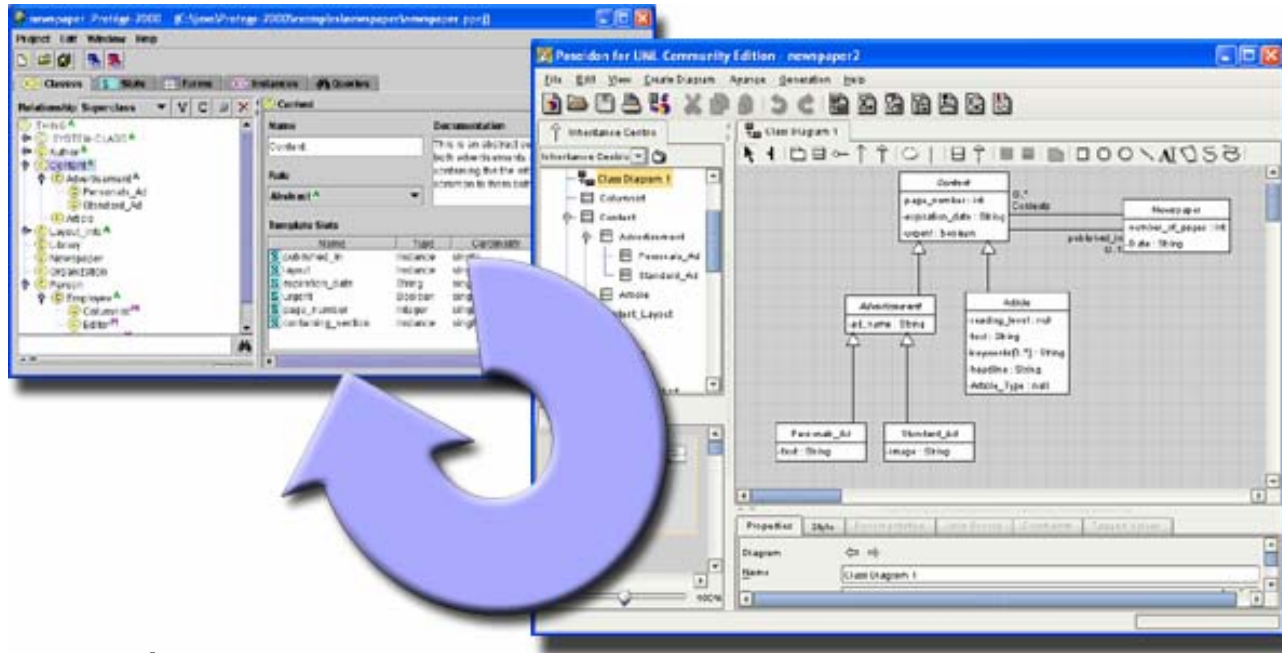


Vision: Synchronized Model-Oriented Development



With each translation: More and better optimized tools

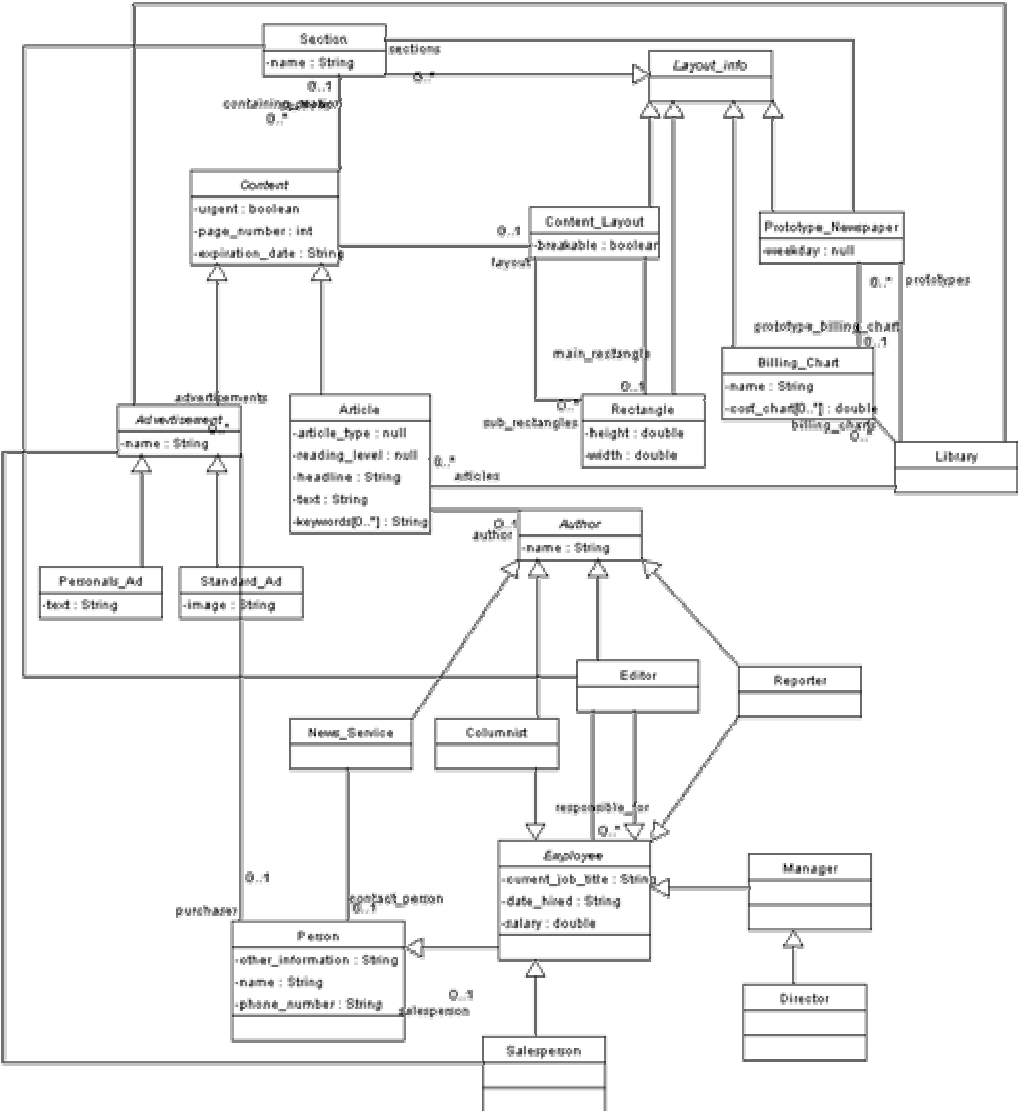
UML Backend



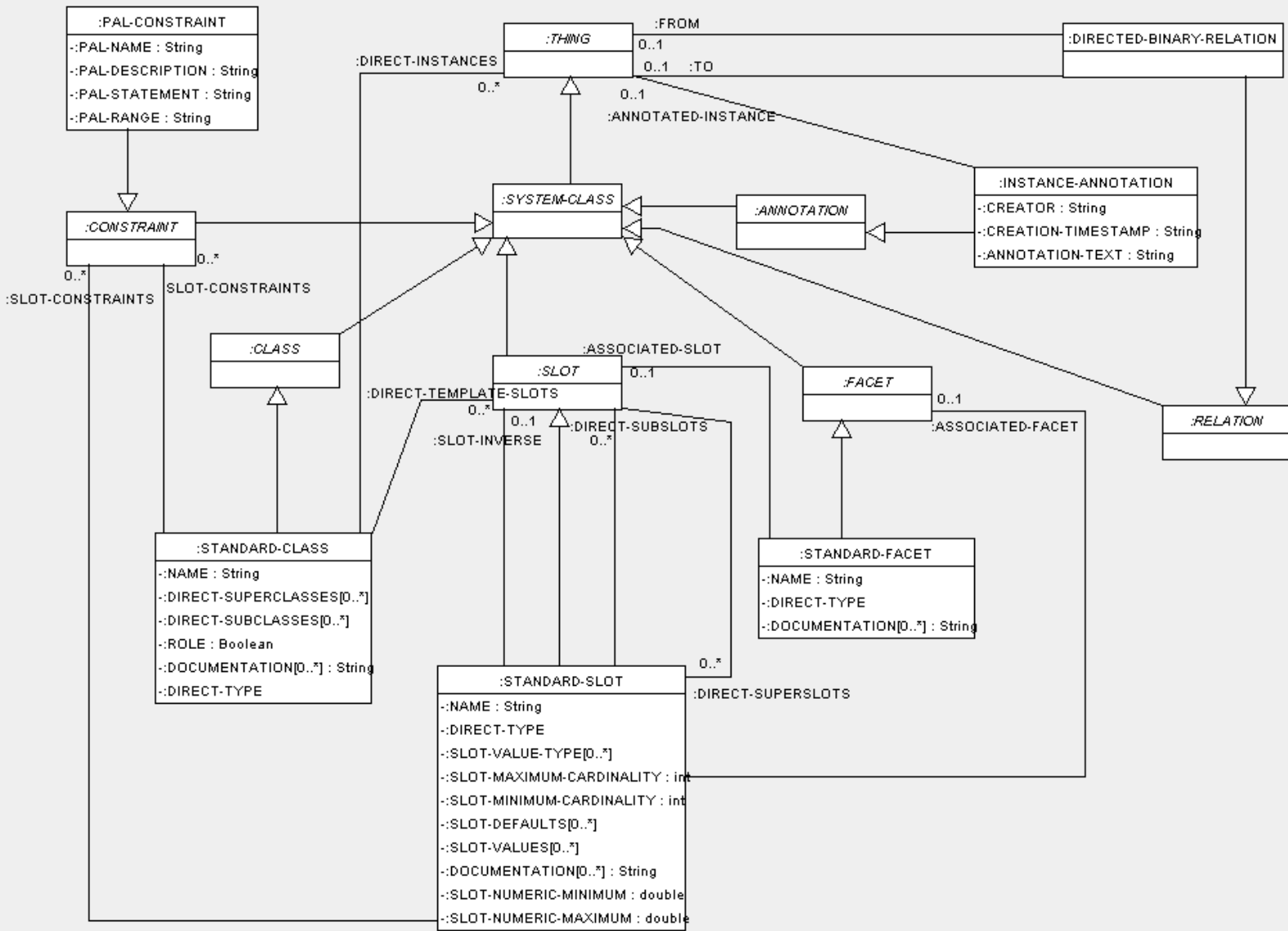
Simple mapping:

- Classes – Classes
- Inheritance – Inheritance
- Primitive Slots – Attributes
- Instance Slots – Associations

Example: Newspaper Ontology in UML



Example: Protégé Metamodel Reengineered



We need more than this

- Simple import / export leads to bumpy transitions
 - Tool and file switching
 - Models are always overwritten and need to be re-adapted each time
 - Only the shared subset of language elements is converted

- Needed:
 - Tool Integration
 - **Protégé as a part of Integrated Development Environments**
 - Synchronization of changes / Round-Trip Engineering
 - **Clean metamodel mapping, Standards**

Object Management Group (OMG)

- World's largest software consortium (established 1989)
- Goal: Define open, vendor-neutral specifications
 - IDL, CORBA (Object-oriented data exchange)
 - UML (Unified Modeling Language)
 - MOF (Meta Object Facility)
 - XMI (XML Metadata Interchange)
 - CWM (Common Warehouse Metamodel)
- Specification Process: Request for Proposals, Submissions, Final Docs
- Similar to World Wide Web Consortium (W3C)
- Tries to build bridges into the W3C universe
- Gradual move to more complete semantic models (e.g. in UML 2)

Platform-Specific Layer

(Java, C#, Smalltalk)

Platform-Independent Layer

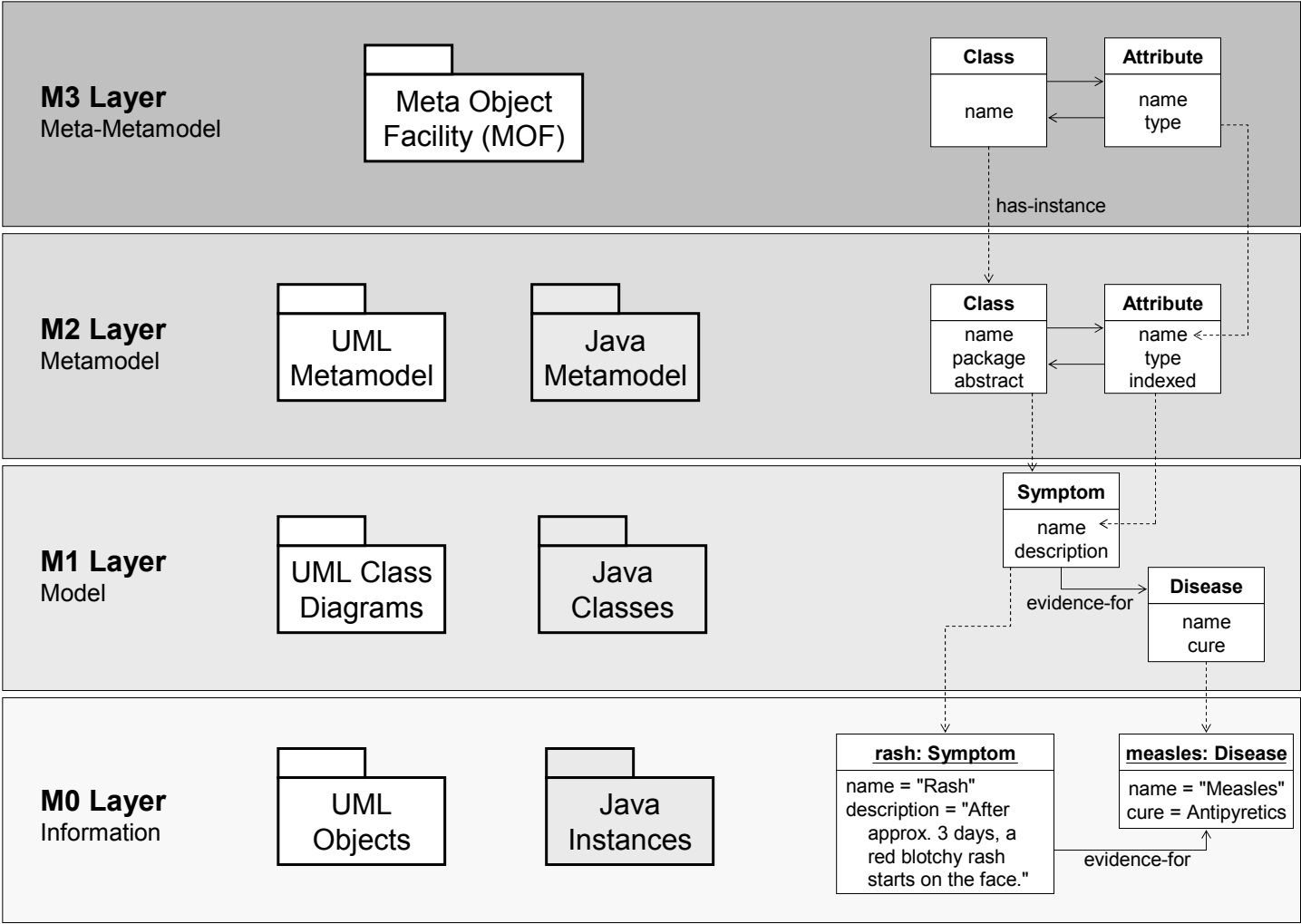
(UML models, CORBA)

Software-Independent Layer

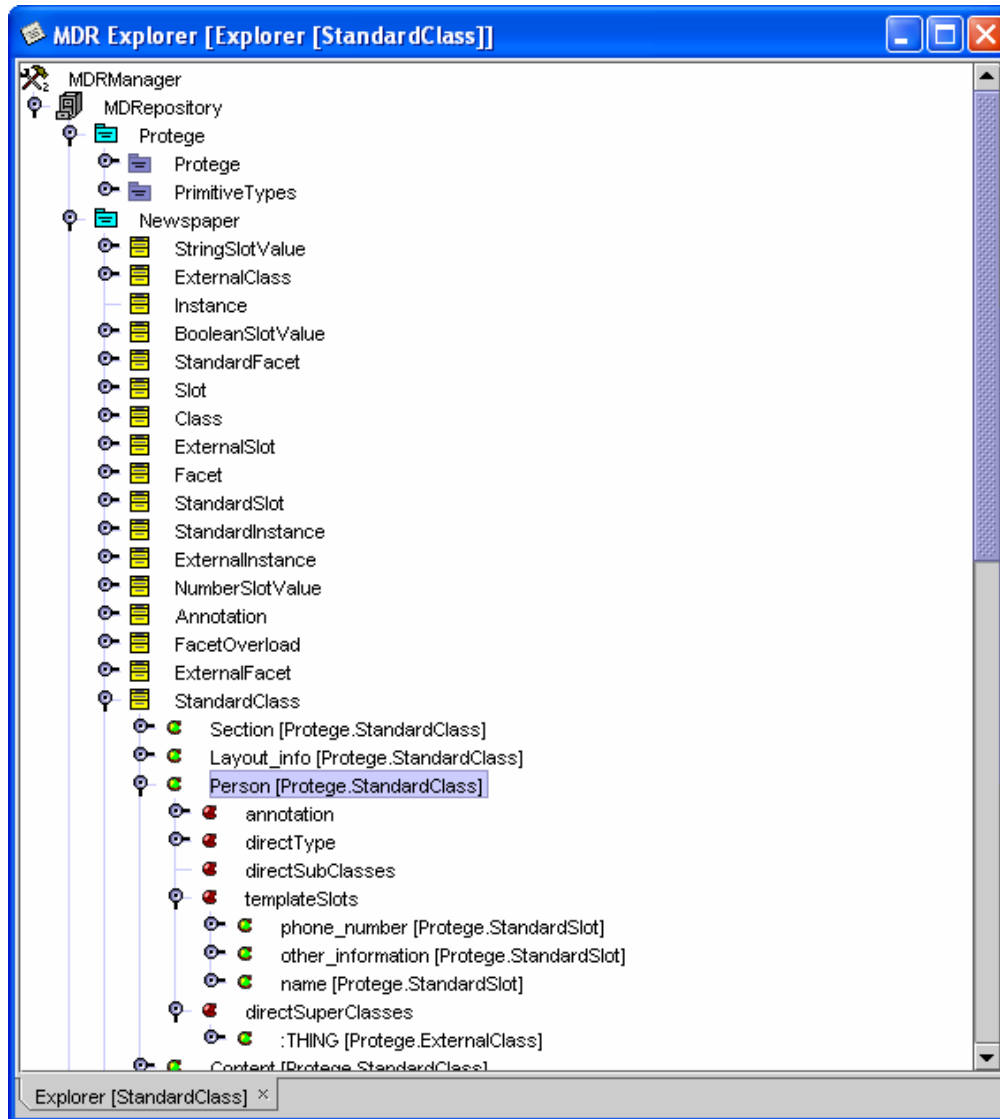
(Reusable Domain Models, Use Cases)



OMG Metamodel Hierarchy



MOF, XMI, JMI, MDR, and all the Rest



(MOF)

Generic services:

- Model management
- XMI reader / writer
- Editors (tree view, etc)
- Mappings
- Constraint checks
- Cascading delete, etc.
- Reflection

Protege's XMI Backend

- Developed in March 2003
- Goals
 - Solid XML storage format
 - Compatible MOF format
 - Proof of concept of tool integration
 - Base technology for future work
- Consists of
 - MOF metamodel of Protégé metamodel (OKBC)
 - Java (JMI) classes that represent this metamodel
 - Mapping implementation between Protégé objects and JMI classes

XMI Backend: Automatically Generated JMI classes

```
package edu.stanford.smi.protege.xmlmodel;

public interface StandardSlot extends Pslot
{
    public java.lang.String getValueType();
    public void setValueType(java.lang.String newValue);
    public int getMaxCardinality();
    public void setMaxCardinality(int newValue);
    public int getMinCardinality();
    public void setMinCardinality(int newValue);
    ...
}
```

XMI Backend: Storage Format

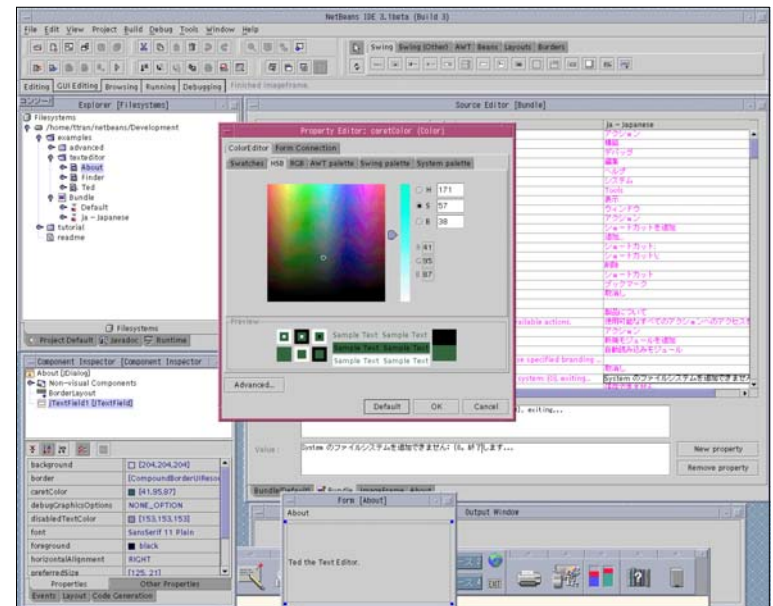
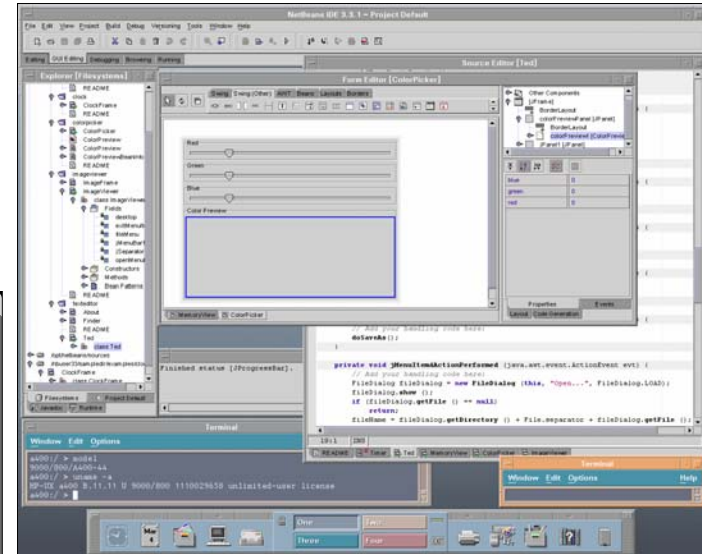
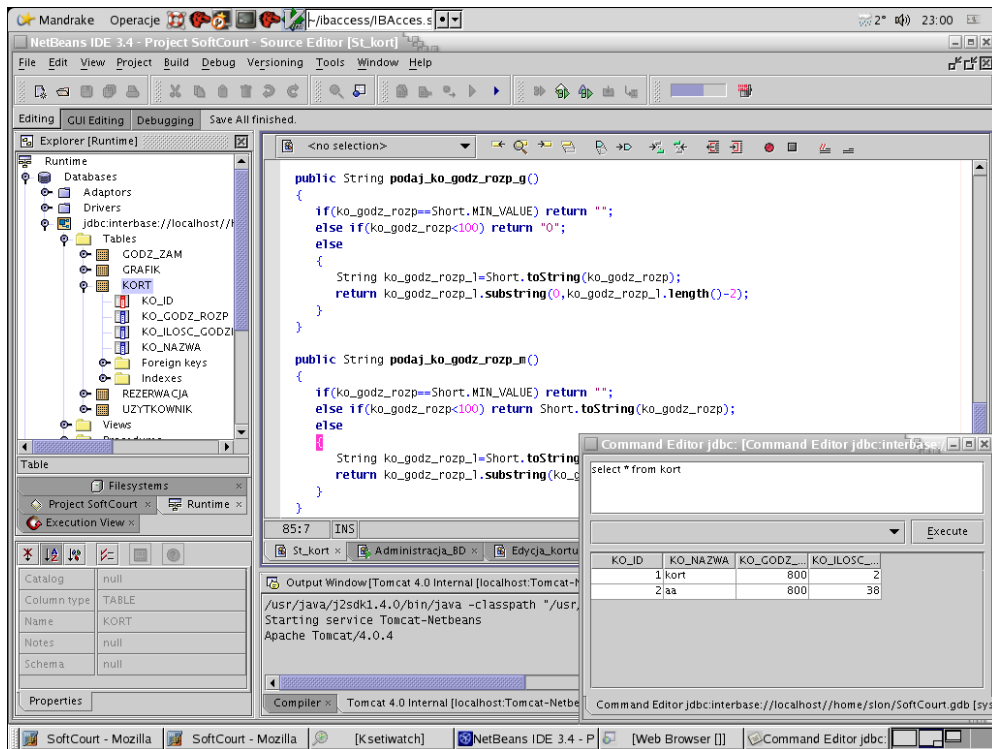
```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<XMI xmi.version = '1.2' timestamp = 'Wed Feb 26 17:38:22 PST 2003'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <Protege.StandardSlot xmi.id = 'a5' name = 'width'
      valueType = 'Float' maxCardinality = '1' minCardinality = '0'>
      <Protege.StandardInstance.directType>
        <Protege.ExternalClass xmi.idref = 'a2' />
      </Protege.StandardInstance.directType>
      <Protege.StandardSlot.templateSlotClasses>
        <Protege.StandardClass xmi.idref = 'a6' />
      </Protege.StandardSlot.templateSlotClasses>
    </Protege.StandardSlot>
    <Protege.StandardSlot xmi.id = 'a7' name = 'current_job_title'
      valueType = 'String' maxCardinality = '1' minCardinality = '0'>
      <Protege.StandardInstance.directType>
        <Protege.ExternalClass xmi.idref = 'a2' />
      </Protege.StandardInstance.directType>
      <Protege.StandardSlot.templateSlotClasses>
        <Protege.StandardClass xmi.idref = 'a8' />
      </Protege.StandardSlot.templateSlotClasses>
    </Protege.StandardSlot>
  </XMI.content>
</XMI>
```

Parser into handy
objects is included

Tool Integration

- Many Java-based tools provide an open API / plugin-mechanism
 - IntelliJ
 - JBuilder
 - Poseidon for UML
 - Argo UML
 - TogetherJ
 - Protégé
- Some of them are general-purpose tool platforms
 - NetBeans
 - Eclipse

Tool Integration / NetBeans Tools Platform



Tool Integration / NetBeans / Services

- Window Management
- Plugin-Architecture
- Action / Toolbar Management
- File / Data Access
- Wizard Frameworks
- Settings Management
- Hierarchical Data Management & Presentation
- Auto Update
- Editor for multiple languages
- CVS / Versioning Control
- Database support
- Scripting Support
- Metadata Management

Tool Integration / NetBeans / Simple Plugins

The screenshot displays the NetBeans IDE interface. The main window is titled "Forte for Java 4, Community Edition [Project Default]". The menu bar includes File, Edit, View, Project, Build, Debug, Versioning, Tools, Window, and Help. The toolbar contains various icons for file operations and development actions. The Explorer on the left shows a project structure with folders like "knublauch (->)", "protege", "beans2protege", and "sample". A context menu is open over the "Person" class, listing actions such as "Open", "Customize Bean", "Compile", "Build", "Execute", "Cut", "Copy", "Paste", "Add", "Delete", "Rename...", "Save As Template...", "Tools", and "Properties". The "Tools" submenu is expanded, showing options like "Internationalization", "JUnit Tests", "Beans2Protege", "Mark as Servlet", "Set as Project Main Class", "Add to Project", "Create Group...", "Synchronize", "Import Management Tool", "Add to Component Palette...", "Generate Javadoc", "Auto Comment...", "Show Javadoc", "Copy File", "Explore node...", and "Bean Browse...". The Source Editor on the right shows the following Java code:

```
package com.knublauch.protege.beans2protege.sample;

import java.beans.*;

/**
 * A simple test JavaBean.
 *
 * @author Holger Knublauch
 */
public abstract class Person extends Object implements java.io.Serializable {

    private static final String PROP_ADDRESS_PROPERTY = "address";
    private static final String PROP_NAME_PROPERTY = "name";

    private Address address;

    private String name;

    private
```

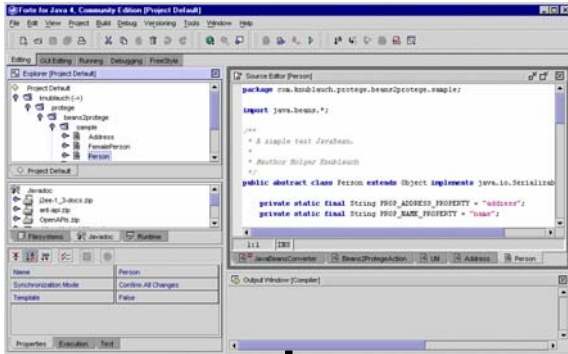
A dialog box titled "<new> Protégé-2000" is open in the foreground. It shows a class hierarchy with "Person" selected. The "Person" class is shown as an abstract class with the following properties:

Name	Type	Cardinality	Oth
address	Instance	single	classes=(Address)
name	String	single	

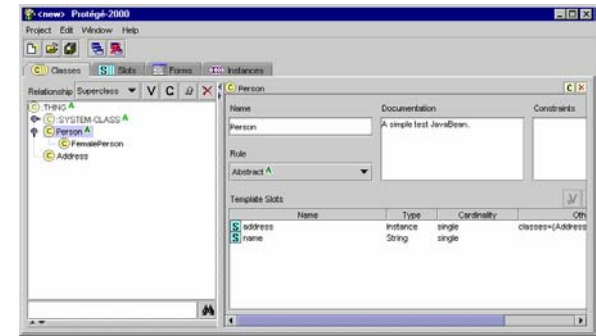
The dialog also shows a "Documentation" field with the text "A simple test JavaBean." and a "Constraints" field.

Tool Integration / Integration Architecture

JMI compliant Tools (NetBeans Modules)



Ontology Tools (Protégé Modules)



Protégé JMI
(MOF) Classes
(Data container)

Adapted
Protégé Model
Classes

Wrap data from

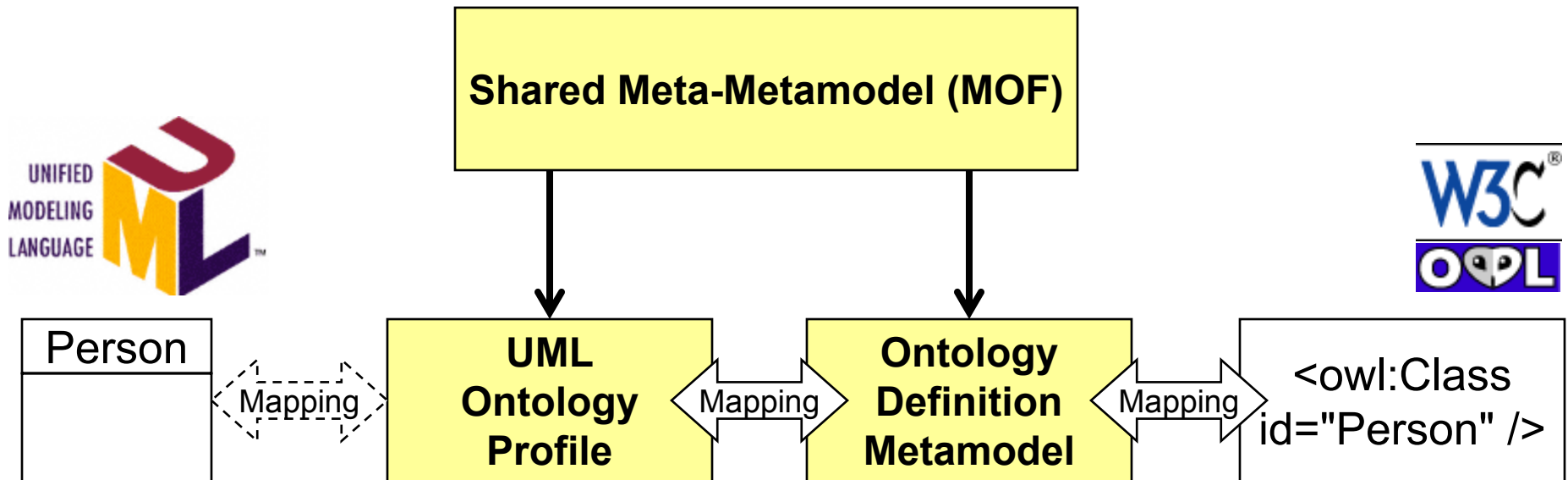
Code generated
automatically
from MOF model

Problems with the OMG Standards

- Tool support in its infancy
- Partly vapor-ware
- Current UML version ill-defined (Incompatible tools)
- Implementation of mappings is non-trivial
- OMG has lost impact in the age of Extreme Programming (XP plus W3C Standards like XML/OWL might be alternative)
- But they already have UML and powerful tools
- Interesting Specification will be available in 2004

OMG's Ontology Definition Metamodel

- Currently: Request for Proposal, due August 18
- Goals:
 - A standard MOF compliant metamodel for Ontology Definition (ODM)
 - UML profile to support reuse of UML notation for ontology definition
 - A mapping from the Ontology Definition Metamodel to the profile
 - A mapping from ODM into OWL DL



Protege's Role in the Model-Driven Architecture

- XMI Backend
 - Proof of concept
 - Integration into MOF-based tools
 - Potential for new development processes
- UML + Class Explorer Tree + Instance forms + Semantics + DL
- Integrated tool can be used to make mapping explicit / automatic
 - Structural knowledge → Software architecture
 - Process knowledge → Communication infrastructure
 - Medical knowledge → Terminology / Classes
 - Diagnostic knowledge → Inference engine
 - Action knowledge → Procedural code
- Reuse of mapping technology (e.g. to Enterprise Beans)

Summary

- Ontologies not only for “scientific” knowledge-based projects
- Knowledge modeling as part of modern software development
- Model-Driven-Architecture and related standards
 - can help to secure a place for Protégé in the Software Engineering community
 - may also evolve as a serious competition to Protege