Virtual Laboratories

Ivan Marsic

Department of Electrical and Computer Engineering
and Center for Advanced Information Processing (CAIP)
Rutgers — The State University of New Jersey
Piscataway, NJ 08854-8058 USA

`http://www.caip.rutgers.edu/~marsic`

Contents

- **Introduction and Requirements**
- **Java Beans** (½ hour)
  - Common software design patterns
  - Events, properties, persistence, and introspection
  - Levels of abstraction
- **Extensible Markup Language (XML)** (½ hour)
  - Concepts and examples
  - Extensible Stylesheet Language (XSL)
  - Bean Markup Language (BML)
- **UML Design for Virtual Laboratories on the Internet** (¾ hour)
  - Analysis
  - Design

Total duration: 2 hours (17:30 – 19:30)

Goals

- Web-based Virtual Laboratories
  - Distance education
  - Available any-time, any-place, any number of repetitions
  - Learn and practice in spite of making errors
- Development & Deployment Dynamics
  - Short-term goal: supplement for actual labs (preparation & rehearsal)
  - Long-term goal: virtual labs substitute actual labs
- Development Emphasis
  - Look-and-feel Fidelity vs. Learning Concepts

Related Work

Existing Interactive Virtual Labs:

- Non-Web-based or require plug-in
  - California State University's Center for Distributed Learning
  - The University of Melbourne's Science Media Teaching Unit
  - Edmark, Inc.
  - Olympus America Inc. and The Florida State University
- Java Applets
  - Hughes Medical Institute: Bio-Interactive
  - University of Colorado (Boulder): Physics 2000
  - TeleLearning Network of Centers of Excellence

Related Work (Cont'd)

Summary:

- Not Web-based or require plug-in
  - Not platform-independent Java Applets
- No generic software architecture
- Interaction by "clicking" rather than direct manipulation
- "Linear" interaction
  - All users have the "exact" same lab experience
  - Our goal: Users have different experience, but learn the same concepts

Part I

Java Beans

Java Beans: Main Features

- Span from a simple button to a full-fledged spreadsheet application
- Uniform attribute (property) access
- Persistence support
- Event model
- Self-description, introspection, and reflection
- Customization and combination at development time
- Development tool support
- Visual representation

Bean: A reusable software component that can be manipulated visually in a builder tool.

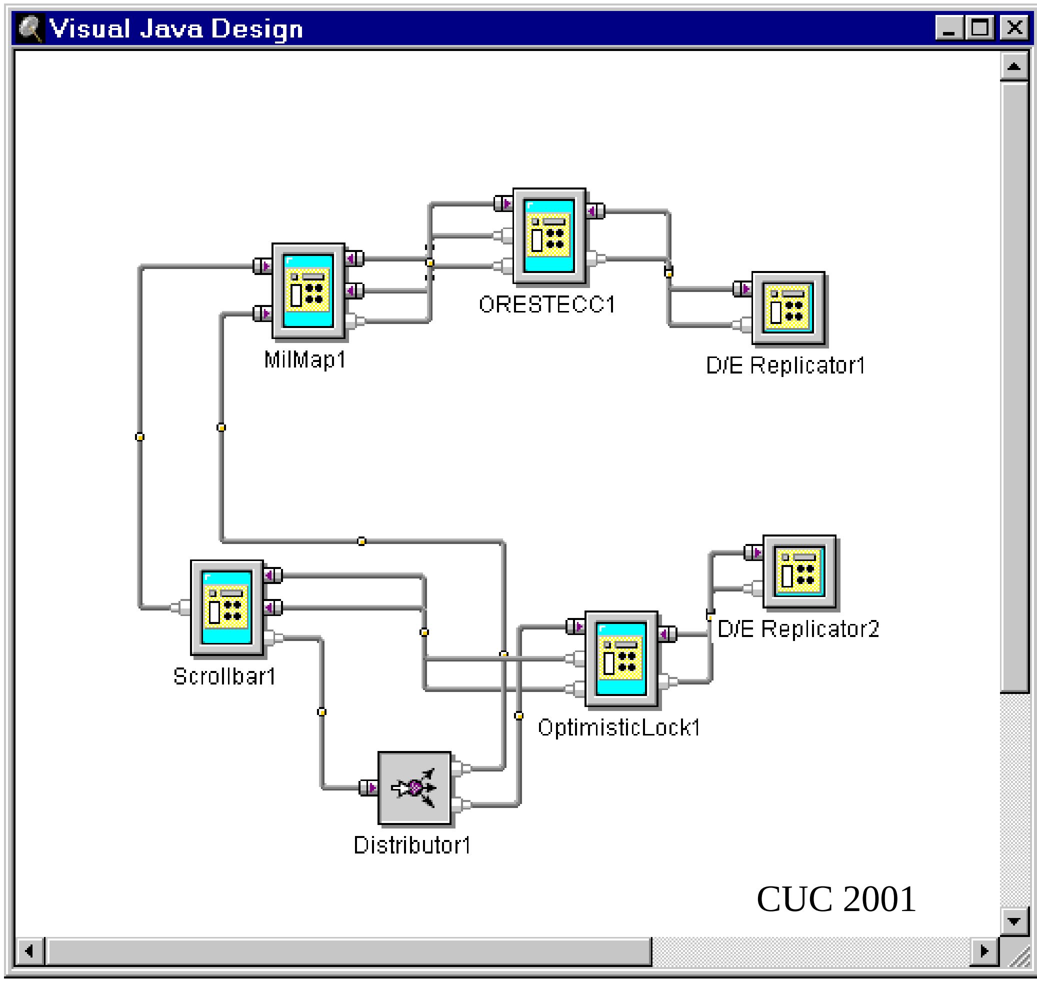
Java Beans: Paradigm

- All beans must implement certain **interfaces** and **classes**, so that a certain behavior can be expected, e.g., for persistence
- The classes and methods of all beans follow certain **design patterns** for naming and signatures, so that this can be **analyzed automatically** by an introspector using reflection
- Java beans may provide explicit **meta-information** for development support (classes that describe the bean)
- There are **standard** mechanisms for **connecting** beans together (by events)
- Beans may include a **graphical** presentation, e.g., a button
- There are **standard** ways for **customizing** beans, even graphically or using custom wizards

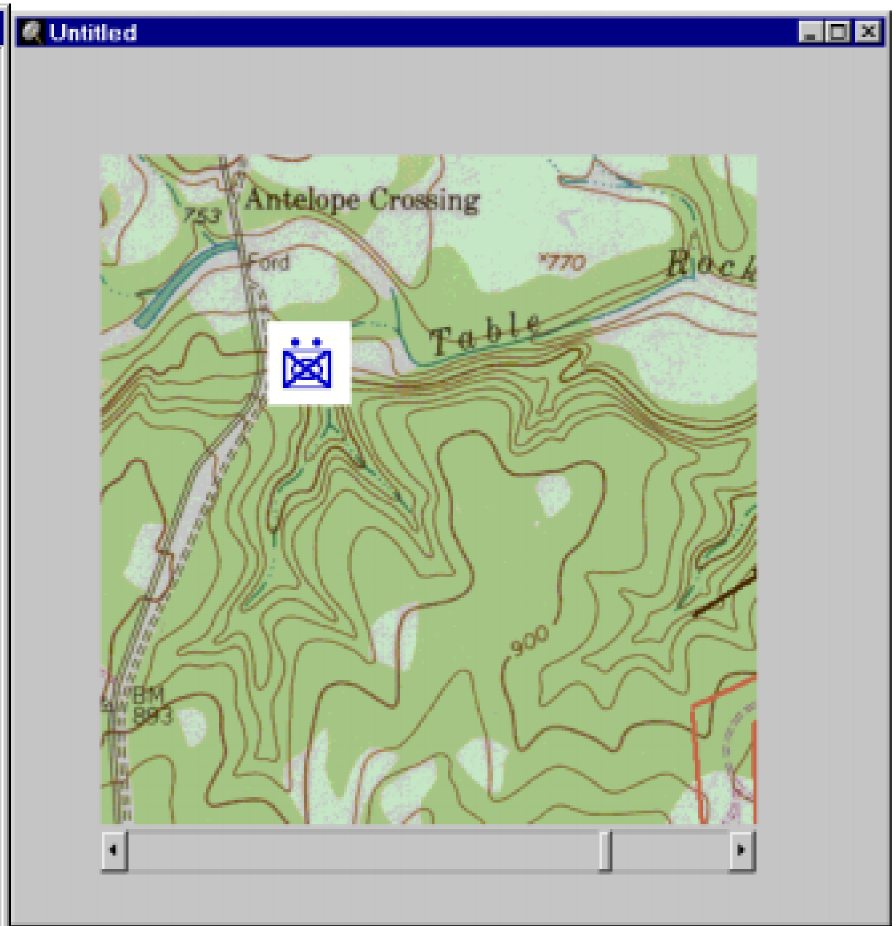
Thus all beans look in a way the same and so you may buy beans and **plug** all of them together **interactively** using a **GUI** beans development tool.

Example: Mapping Application

Development Environment



Application View



Parts of a Bean

A bean consists of:

- **Properties**: attribute values that may be stored persistently; represented by getter/setter methods
- **Events**: State changes that other components may listen for by registering listeners
- **Methods**: Arbitrary public methods

All these methods follow naming and typing patterns, eg.

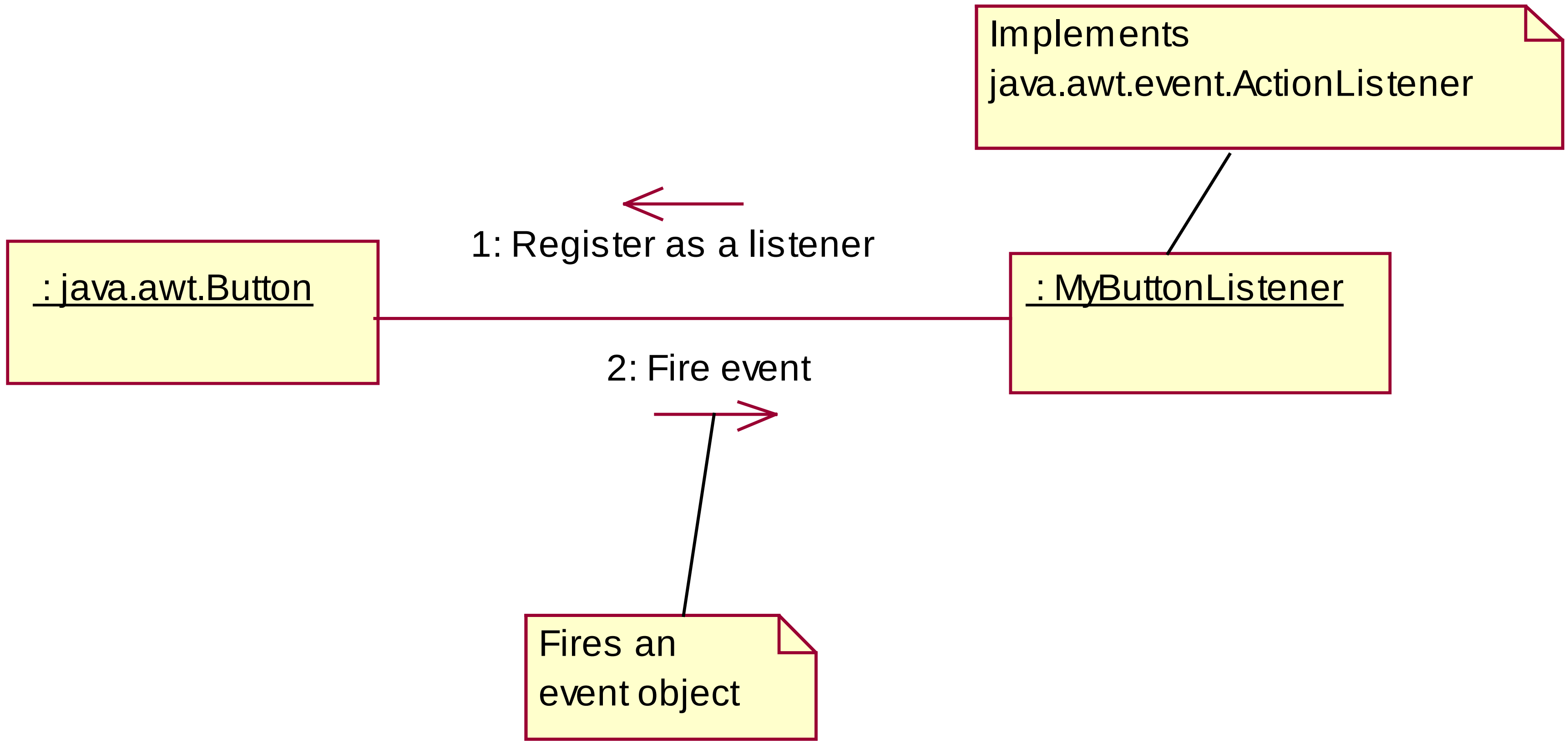
public *TYPE* get*PROPERTY*() {…}
public void set*PROPERTY*(*TYPE* x) {…}

All properties are accessed by methods named "get…" and "set…" with these signatures.

An introspector identifies methods (pairs) following this pattern.

Development tools then allow to customize the beans using this information.

Java Event Model



Events Handled by Beans

Events that an object may handle are expressed through methods that take a parameter whose type is a subclass of **`java.util.EventObject`** (*)

public void eventHandler(MyEvent e) {…}

General signature pattern for events of type *Type*:

public void *methodName*(*Type* e) {…}

All events have to inherit from **`java.util.EventObject`** (*)

Class EventObject:

The object on which the event initially occurred:

Object getSource()

Registering Event Listeners

The administration of event listeners has to be synchronized:

```
private java.util.Vector listeners = new java.util.Vector();
public synchronized void addMyListener(MyListener l) {listeners.addElement(l);}
public synchronized void removeMyListener(MyListener l) {listeners.removeElement(l);}
```

General signature pattern for event listener registration:

```
public synchronized void addTYPE(TYPE listener);
public synchronized void removeTYPE(TYPE listener);
```

All event listeners have to implement **`java.util.EventListener`** and follow the naming pattern *TYPE*Listener, where *TYPE* is the event type.

Dispatching Events to Listeners

When dispatching events to listeners, race conditions have to be considered:

```
private void fireAction() {
    Vector targets;
    synchronized (this) {            // could use Enumeration
        targets = (Vector) listeners.clone();
    }
    MyEvent evt = new MyEvent(this, …);
    for (int i = 0; i < targets.size(); i++) {
        MyListener target = (MyListener) targets.elementAt(i);
        target.eventHappened(evt);
    }
}
```

Bean Properties

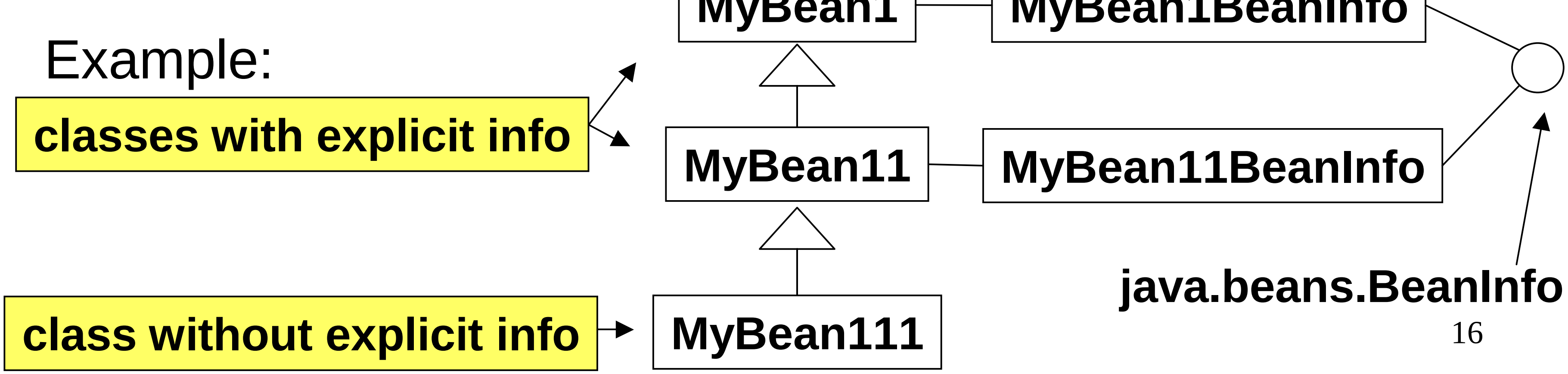
- **Simple**: single-valued, get and/or set methods
- **Indexed**: multi-valued (array) with indexed access
- **Bound**: inform other beans about changes of property values (change event)
- **Constrained**: ask other beans if changes of property values are OK (veto event)

Getting Design Information

Design tools working with and on beans need information about the bean to be able to display its properties, events, methods, and to customize it for an application (*interaction methods*). The **Introspector** class uses two mechanisms for providing this information:

- **Explicit** by **introspection**: A BeanInfo class supplements a Bean class with explicit information about the bean.
- **Implicit** by **reflection**: The reflection capabilities of Java are used to analyze the names and parameters of the Bean classes and methods.

Example:



Introspection and Reflection

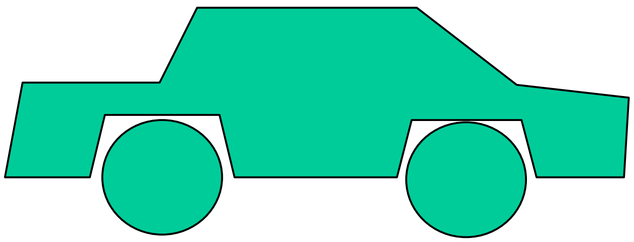
The introspector, the development tools and the Java environment introspect beans using reflection:

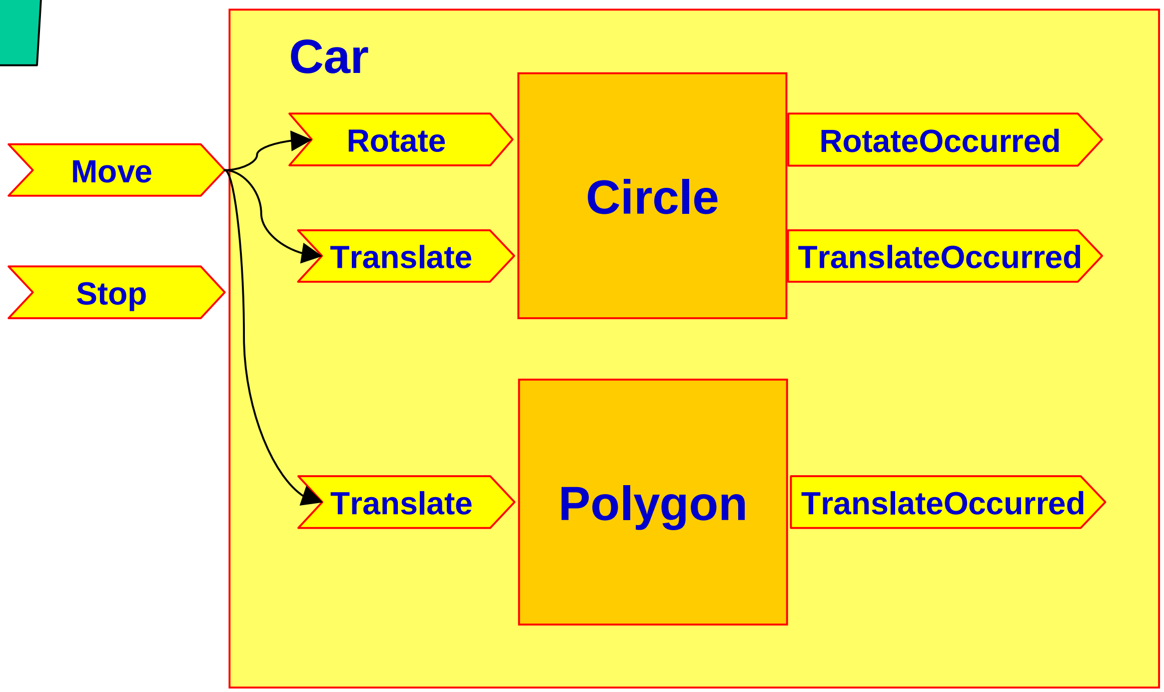
```
MyBean b = Beans.instantiate(null, "MyBean");
BeanInfo bi = Introspector.getBeanInfo(b.getClass());
```

- Signature patterns by method naming (get…, set…) and/or typing (subtypes of EventListener, …)
- Implementation relationships and subclasses (Serializable, EventObject, …)
- Related class names (*MyBean*, *MyBeanBeanInfo*, …)
- Explicit meta-information (given my MyBeanBeanInfo, …)
- Information in JAR-files (**`JavaBean: True`**, …)

Levels of Abstraction

Example of a composite bean:





Note: The composite bean may have different events than the constituent beans.

(Re-)Construction of Beans

Beans may be aggregated and may need context information.

Therefore: Do not (re-)construct a bean using the **new** constructor.

Instead, use **`Beans.instantiate()`**.

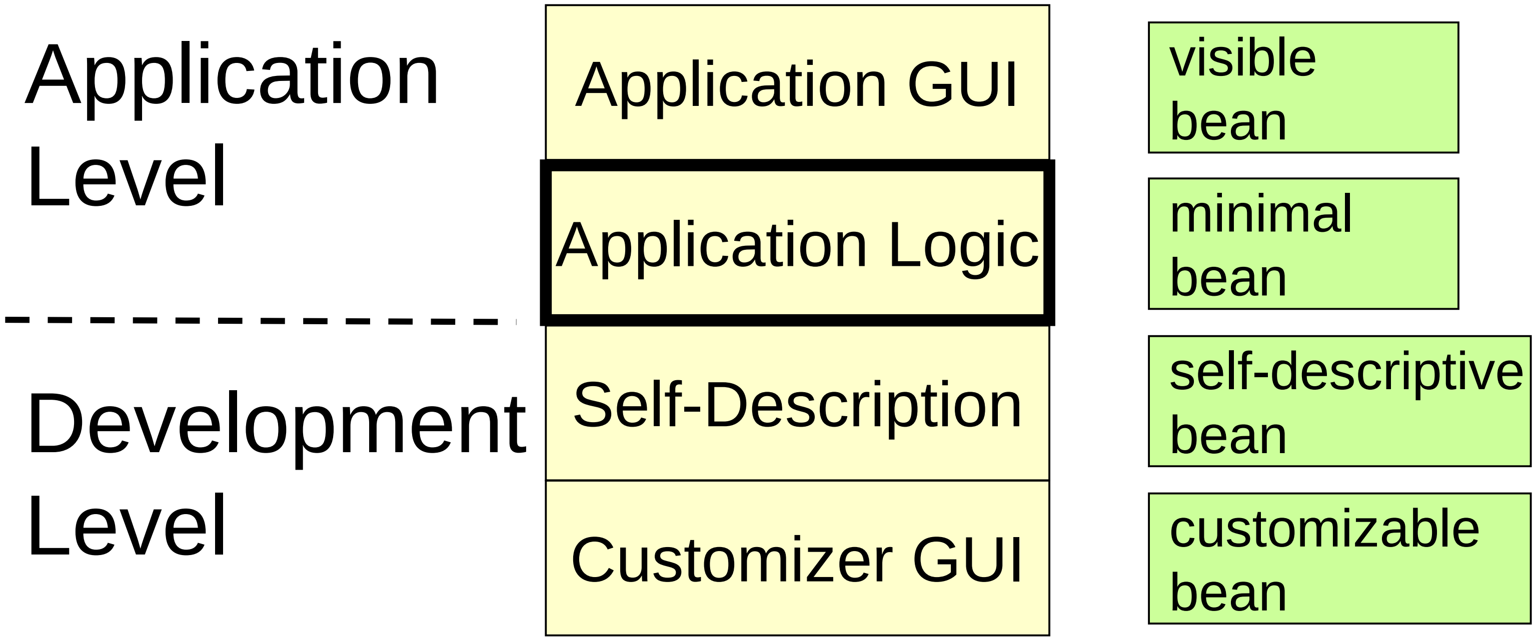
- Creation of a bean class MyBean in package MyPackage:

```
ClassLoader cl = this.getClass().getClassLoader();
MyBean c = (MyBean)Beans.instantiate(cl, “MyBeanPackage.MyBean”);
```

- Recreation of a bean from a serialized form in file MyBeanPackage.MyBean1234.ser:

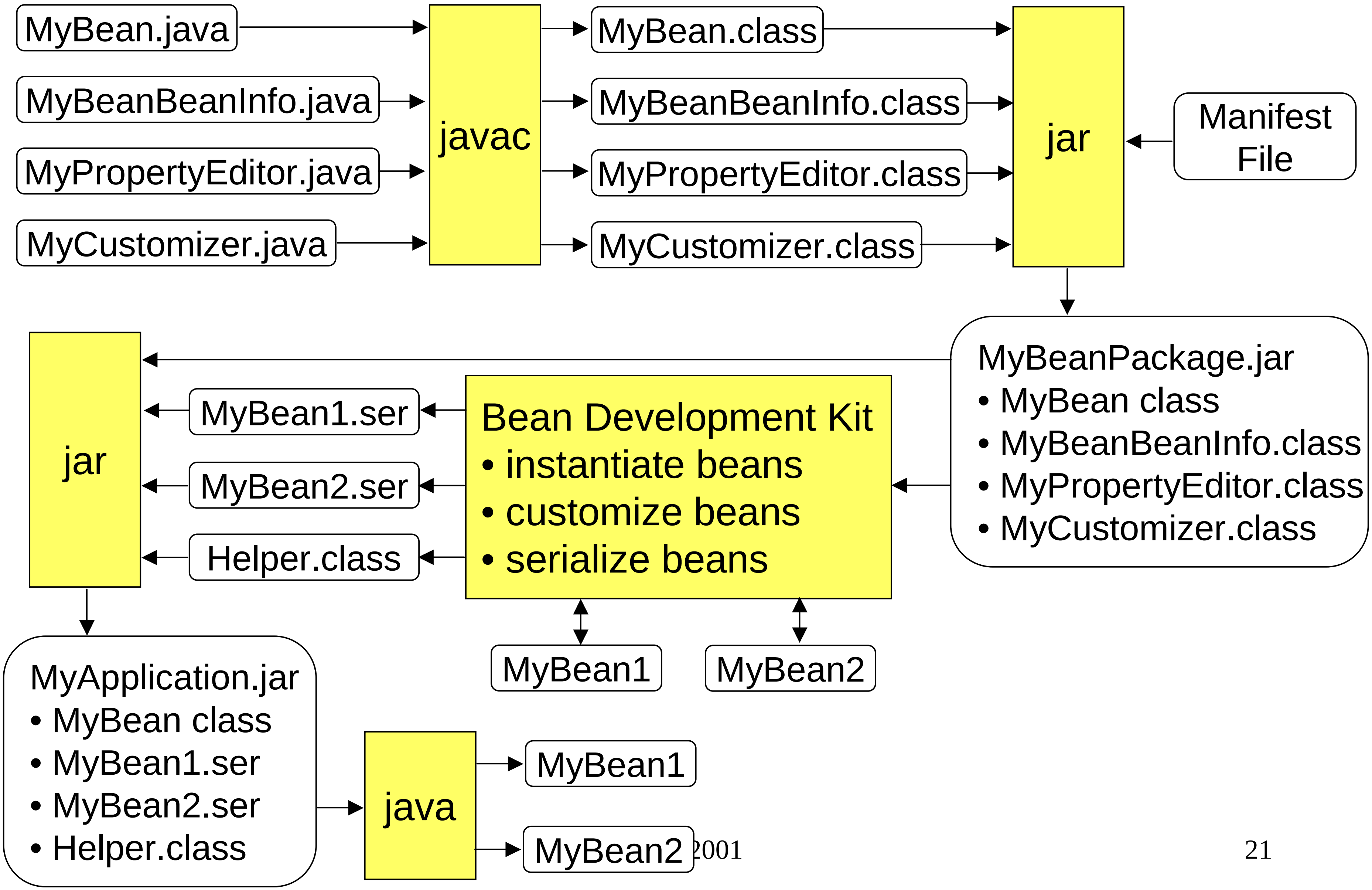
```
ClassLoader cl = this.getClass().getClassLoader();
MyBean c = (MyBean)Beans.instantiate(cl, “MyBeanPackage.MyBean1234”);
```

Application and Development Levels



Visible beans (with GUI) have to inherit from **`java.awt.Component`**

Development and Deployment Process



JavaBeans Across Platforms

- Unfortunately, JavaBeans are not supported on Appliances, e.g., J2ME
- However, JavaBeans Design Patterns still very useful for cross-platform development
  - Event Model
  - Properties
  - Introspection and Reflection ?

Part II

Extensible Markup Language
(XML)

XML Recap

- Method for putting structured data in a (text) file
  - HTML is an implementation of SGML
  - XML is a subset of SGML
- XML is text but not meant to be read
- XML is a family of technologies
  - XML: base specification
  - XSL/XSLT: transformational language
  - XLink: describes logical links between different elements
  - XPointer/XPath …

Understanding XML Documents

- Parsing:
  - tokenizer + lexical analyzer
- How XML parsers work:
  - XML is a ***Markup***
  - Delimiters “<” “>” “</” “/>” etc.
  - Tags are identified as nodes, which are objects conforming to interfaces recommended by W3
  - Attributes are used to define properties of nodes
- Validation:
  - Document Type Definition (DTD)
  - Type of XML docs
    - Well formed
    - Valid

XML Parsers

- Types:
  - DOM (Document Object Model)
    - Tree structure is maintained
    - Actual tree representation of XML doc in memory exists for manipulation
    - Supported by commercial browsers
  - SAX (Simple API for XML)
    - Event based
    - No complete representation exists at any time
    - Good for large XML documents / small terminals
- Libraries available from:
  - XML4J from AlphaWorks IBM(XML for Java) very popular
  - Xerces: joint effort, mainly Apache/AlphaWorks
  - JXML, Sun, Microsoft, Oracle etc.

SAX Parser: Java Example

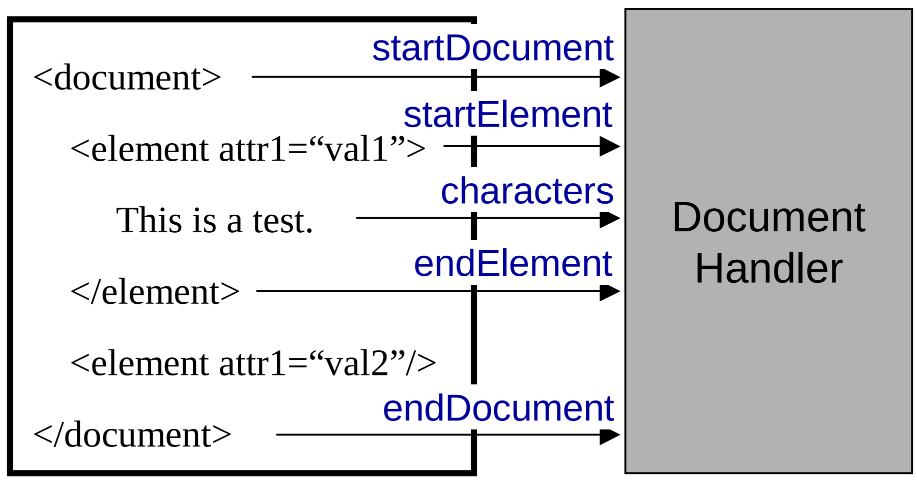
***Initiating** the parser*

```
. . .
Parser parser =
    ParserFactory.makeParser
    ("com.sun.xml.parser.Parser");

parser.setDocumentHandler(new
    DocumentHandlerImpl());

parser.parse (input);
. . .
```

DocumentHandler Interface

```
public void startDocument()throws
    SAXException{}
public void endDocument()throws
    SAXException{}
public void startElement(String name,
    AttributeList attrs) throws
    SAXException{}
public void endElement(String
    name)throws SAXException{}
public void characters(char buf [],
    int offset, int len)throws
    SAXException{}
```

Event triggering in SAX parser:



DTD: Document Type Definition

- Lets define your own markup language
- Specifies constraints on the valid tags and tag sequences that can be in the document
- Defined as an XML document itself
- Includes:
  - ***local subset***, defined in the current file
  - ***external subset***, which consists of the definitions contained in external “*.dtd”

DTD Example

External DTD

```
<?xml encoding="UTF-8"?>
<!DOCTYPE mydoc [<!ENTITY % first SYSTEM "first.dtd"><!ENTITY %
    second SYSTEM "second.dtd"><!ENTITY% third SYSTEM
    "third.dtd">%first;%second;%third;]>
<!ELEMENT APPLICATION (VIEW*,MODEL+)+>
<!ATTLIST APPLICATION NAME CDATA #IMPLIED>
<!ELEMENT MODEL (DATAMODEL*,DATA_OPTIONS*)*>
<!ELEMENT DATAMODEL EMPTY>
• • •
```

Element definition

Attribute listing

? ≡ can be skipped

* ≡ can be skipped or included one or more times

\+ ≡ must be included one or more times

DOM: Document Object Model

- Tree form representation of a Document, even if it is a non-document form database
- Converts an XML document into a collection of objects in your program. You can then manipulate the DOM.
  - Mechanism is also known as the "random access" protocol, because you can visit any part of the data at any time. You can then modify the data, remove it, or insert new data.
- Different node types
  - Root node
  - Text node
  - Element node (with attributes)
- Interfaces defined in IDL
  - Implemented differently on different platforms (http://www.w3.org/DOM/faq.html#what)

DOM Interface Example

- Hierarchy of **Node** objects that also implement other, more specialized interfaces
- **Node types**, and node types they may have as children, are as follows:

  Document -- Element (max 1), ProcessingInstruction, Comment, DocumentType

  DocumentFragment -- Element, ProcessingInstruction, . . .

  DocumentType -- no children

  Element -- Element, Text, Comment, ProcessingInstruction, . . .

  Attr -- Text, EntityReference

  ...
- Node Description in IDL:
  - **Interface Document**:

```
Node { readonly attribute DocumentType doctype;
    readonly attribute DOMImplementation implementation;
    readonly attribute Element documentElement;
    Element createElement(in DOMString tagName)
       raises(DOMException);
    DocumentFragment createDocumentFragment();
    Text createTextNode(in DOMString data);
    Comment createComment(in DOMString data);
    NodeList getElementsByTagName(in DOMString tagname);
};
```

XSL Concepts

- XML is not a fixed tag set (like HTML)
- XML by itself has no (application) semantics
- A generic XML processor has no idea what is “meant” by the XML
- XML markup does not (usually) include formatting information
- The information in an XML document may not be in the form in which it is desired to present it
- Therefore there must be something in addition to the XML document that provides information on how to present or otherwise process the XML

XSL

- XSL is an XML language
- Can be used for specifying the layout of an XML document
- Can be used as a transformational language to transform documents from one DTD to another
- XSLT

Advantages of Separating Content from Style

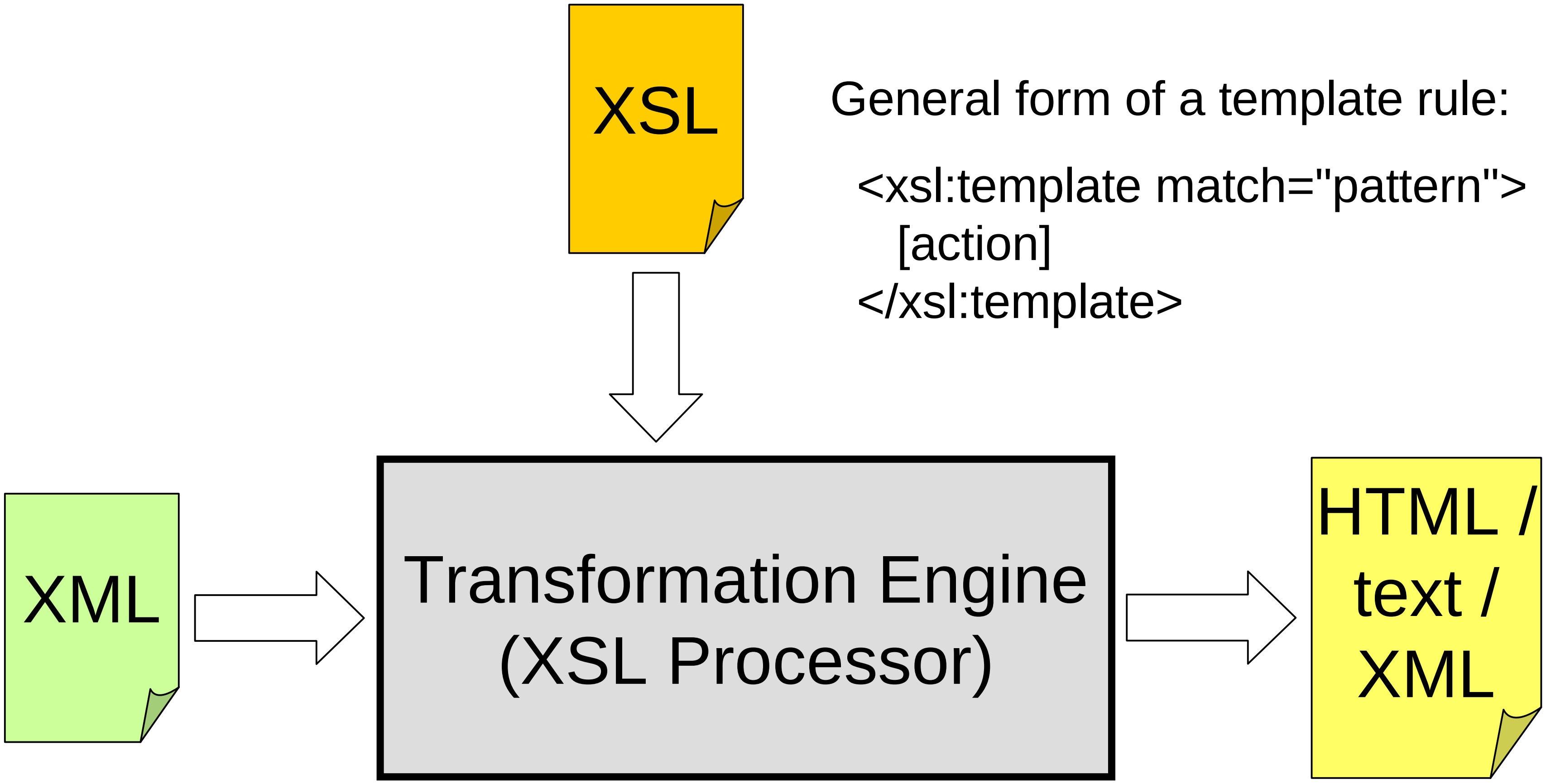
- Reuse of data; different styles
- Multiple output formats
- Reader's preferences
- Standardized styles
- Freedom for content authors

XSL Transformation

XSL is a transformation language:

General form of a template rule:

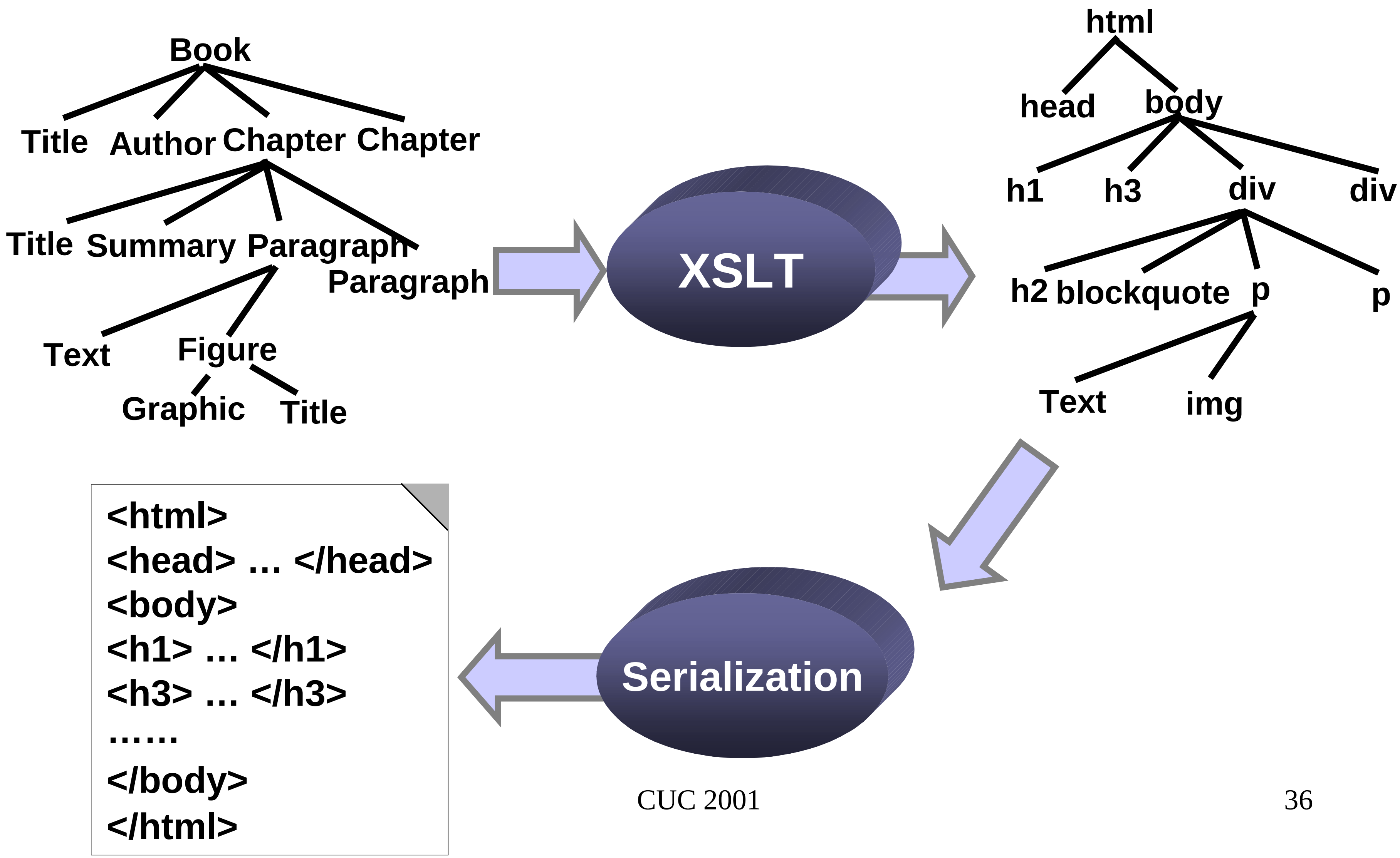
<xsl:template match="pattern">
  [action]
</xsl:template>



XML to Result Tree

XML Source Tree

XHTML Result Tree



XSL Example

- Original XML source

```
<?xml version='1.0'?>
<para>This is a <emphasis>test</emphasis>.</para>
```

- XSL stylesheet

```
<?xml version='1.0'?>
   <xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
   <xsl:template match="para">
            <p><xsl:apply-templates/></p>
   </xsl:template>

   <xsl:template match="emphasis">
       <i><xsl:apply-templates/></i>
   </xsl:template>

</xsl:stylesheet>
```

- Resultant XML source

```
<?xml version="1.0" encoding="utf-8"?>
<p>This is a <i>test</i>.</p>
```

XSL Processors

- Available XSL processors
  - Xalan
- Allows one to tie the XML input to XSL file and delivers output in the form of:
  - Resultant file
  - Resultant DOM tree
  - Set of SAX events as per the resultant tree

XLink

- Allows elements to be inserted into XML documents in order to create and describe links between resources
- Framework for creating both basic unidirectional links and more complex linking structures (bi-directional)
- Allows XML documents to:
  - Assert linking relationships among more than two resources
  - Associate metadata with a link (giving a "role" to the link)
  - Create link to databases that reside in a location separate from the linked resources
- Example:

```
<myElement

    xmlns:xlink="http://www.W3.Org/1999/xlink/namespace/
    ">
  ...
</myElement>
```

Browser Support

- IE 5's full support for XML
  - Namespaces recommendation
  - XML DOM
    - DOM Tree available for manipulation from within the browser
- Netscape support in Ver. 5 (Aurora)
  - Full information integration on desktop
  - Built-in parser
  - Available in the latest release (6.1)?

Part III

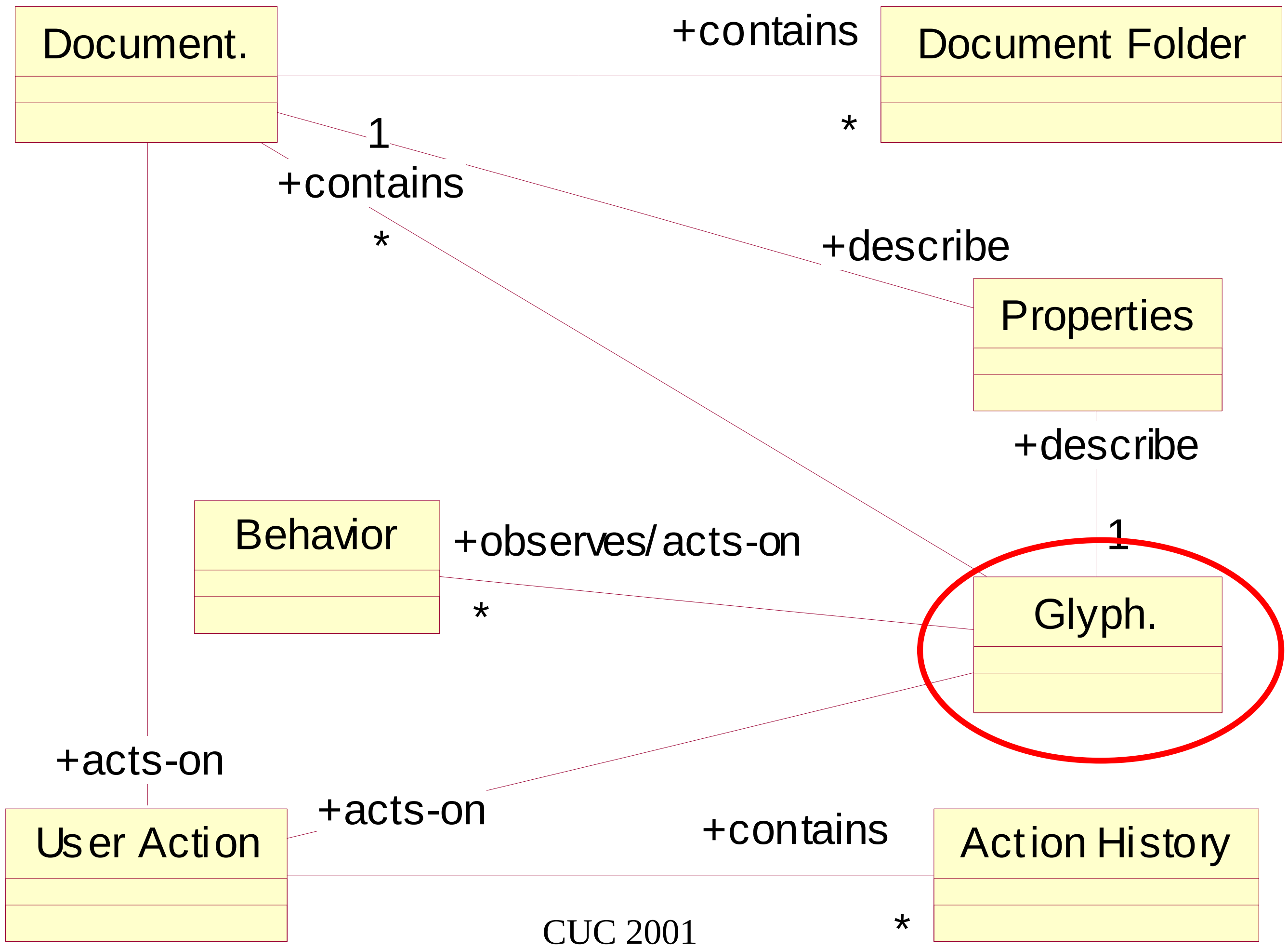
UML Design for
Virtual Laboratories on the Internet

Requirements

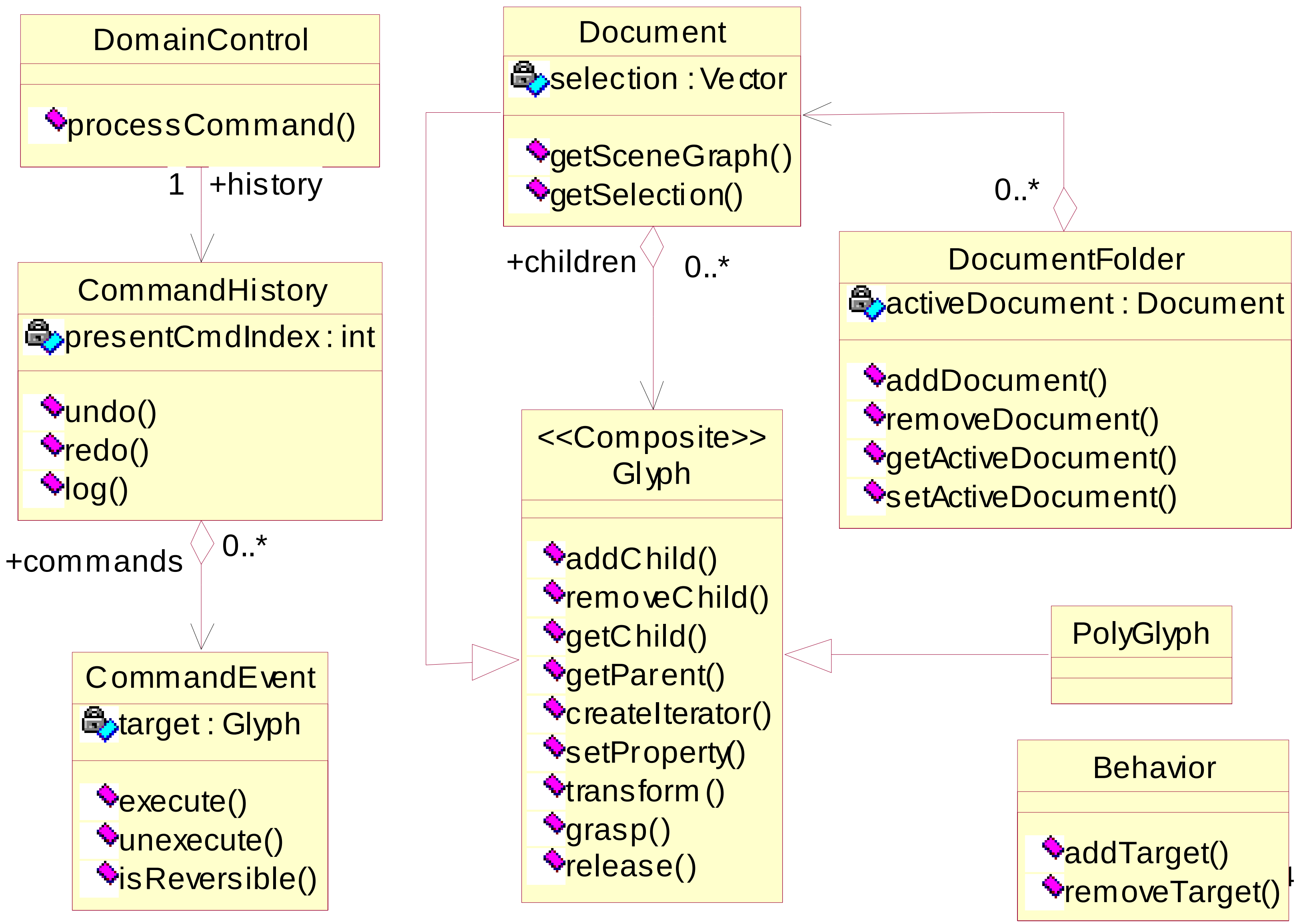
1. Lightweight scalable design for groupware applications
   - Rapid application development; Easy customization
   - From handhelds (or cell phones) to workstations
2. Generic means for application interoperability
   - Standardized transformations between platform- or task-specific representations
3. Automatic run-time adaptation to dynamic environment
   - Adaptation of shared data
     (available resources, task requirements, user preferences)
   - Account for communications breakouts and offline work

Generalized Editor

Conceptual design: Editing structured documents

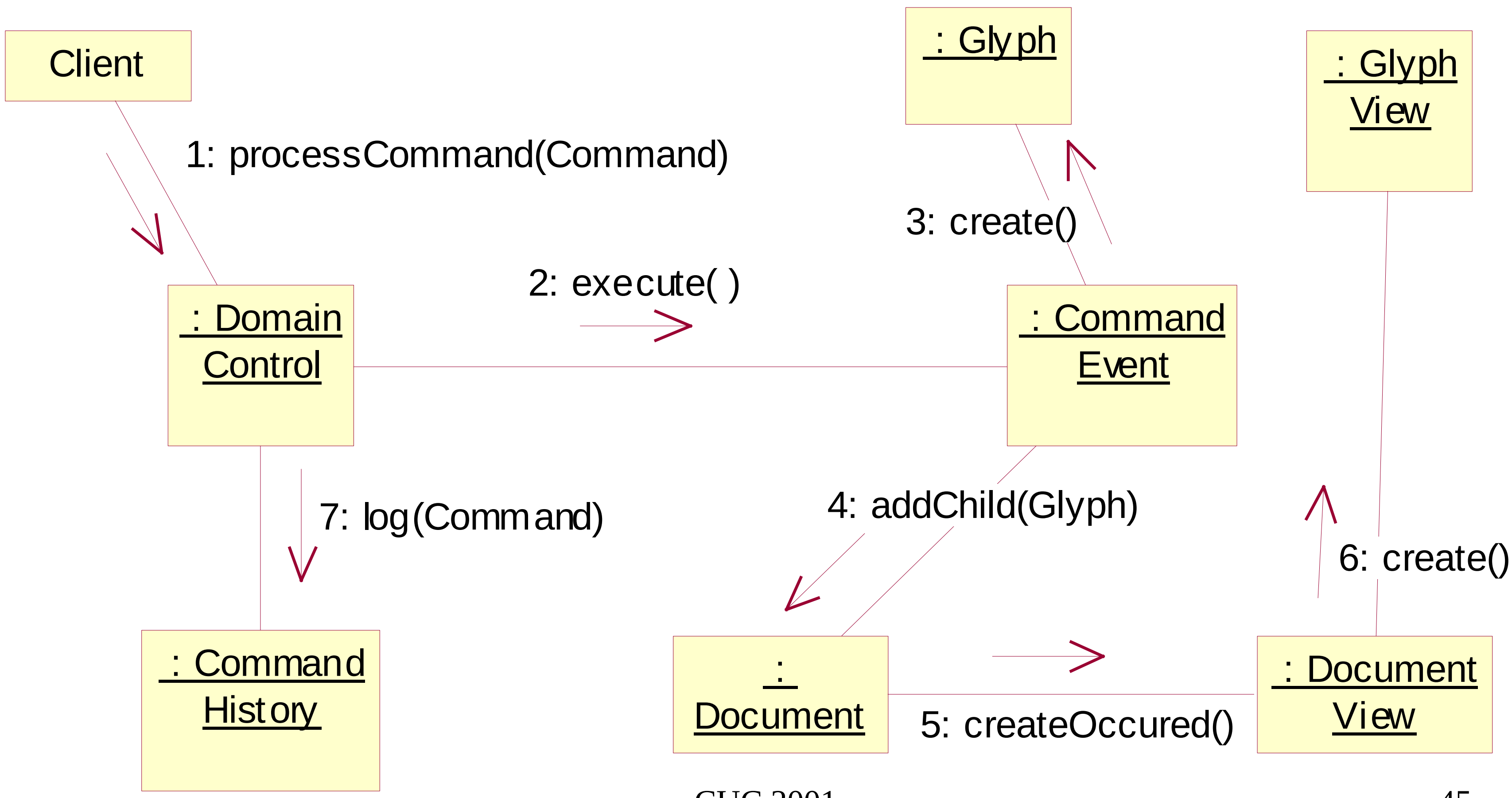


Domain Class Diagram of a Generalized Editor

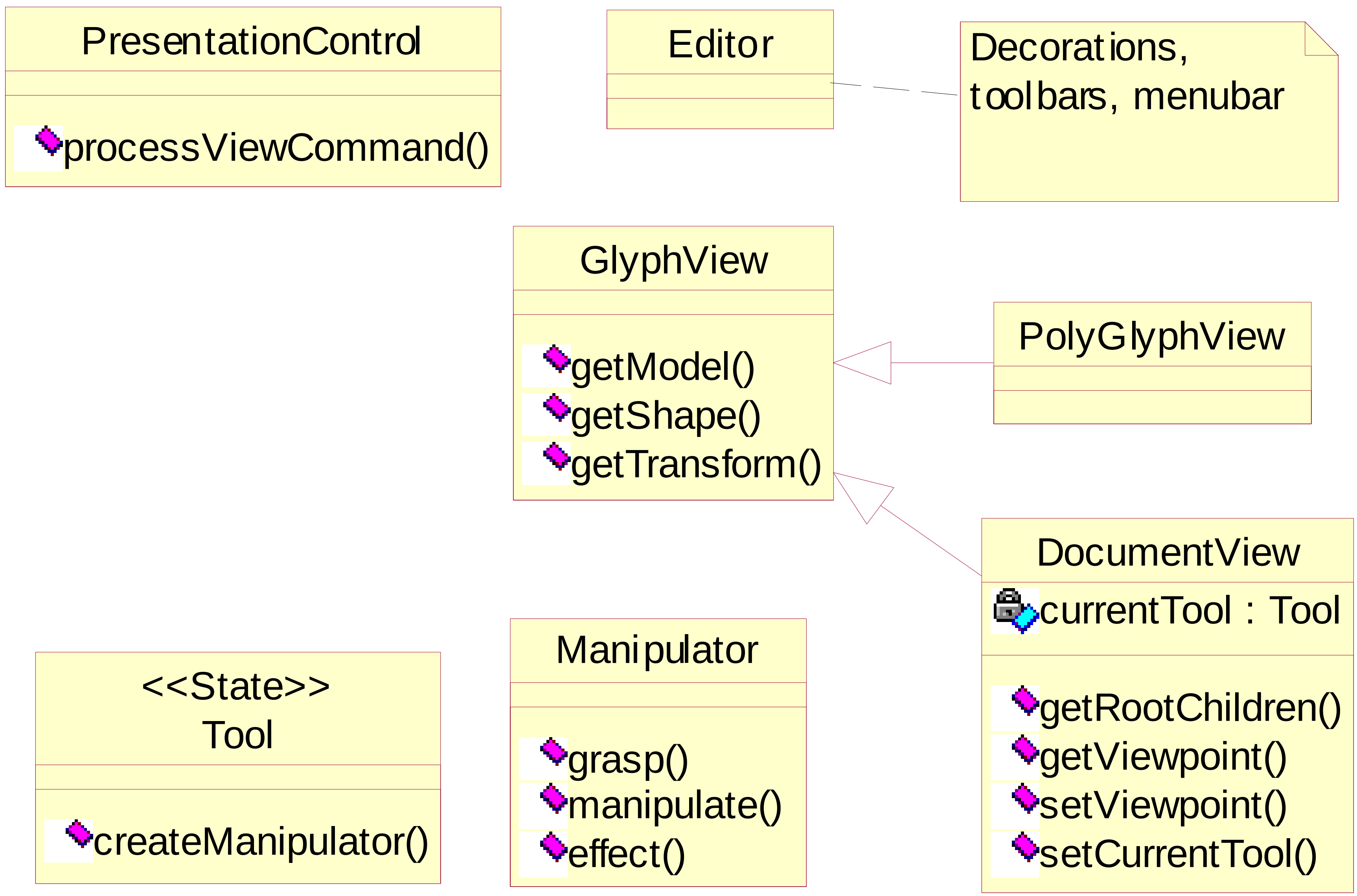


Application Logic

Collaboration diagram for creating a Glyph

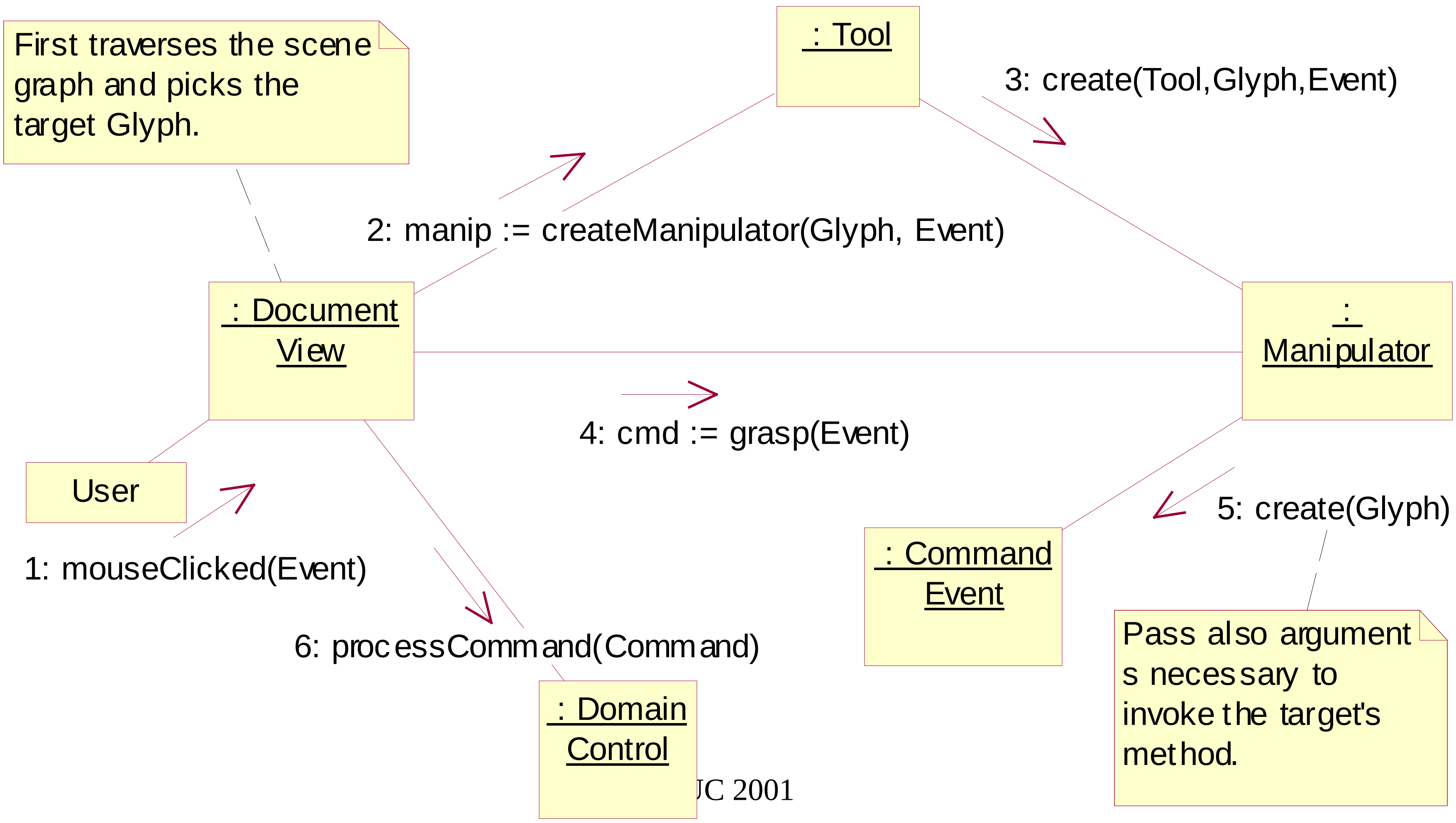


Presentation Class Diagram of a Generalized Editor



Presentation Logic

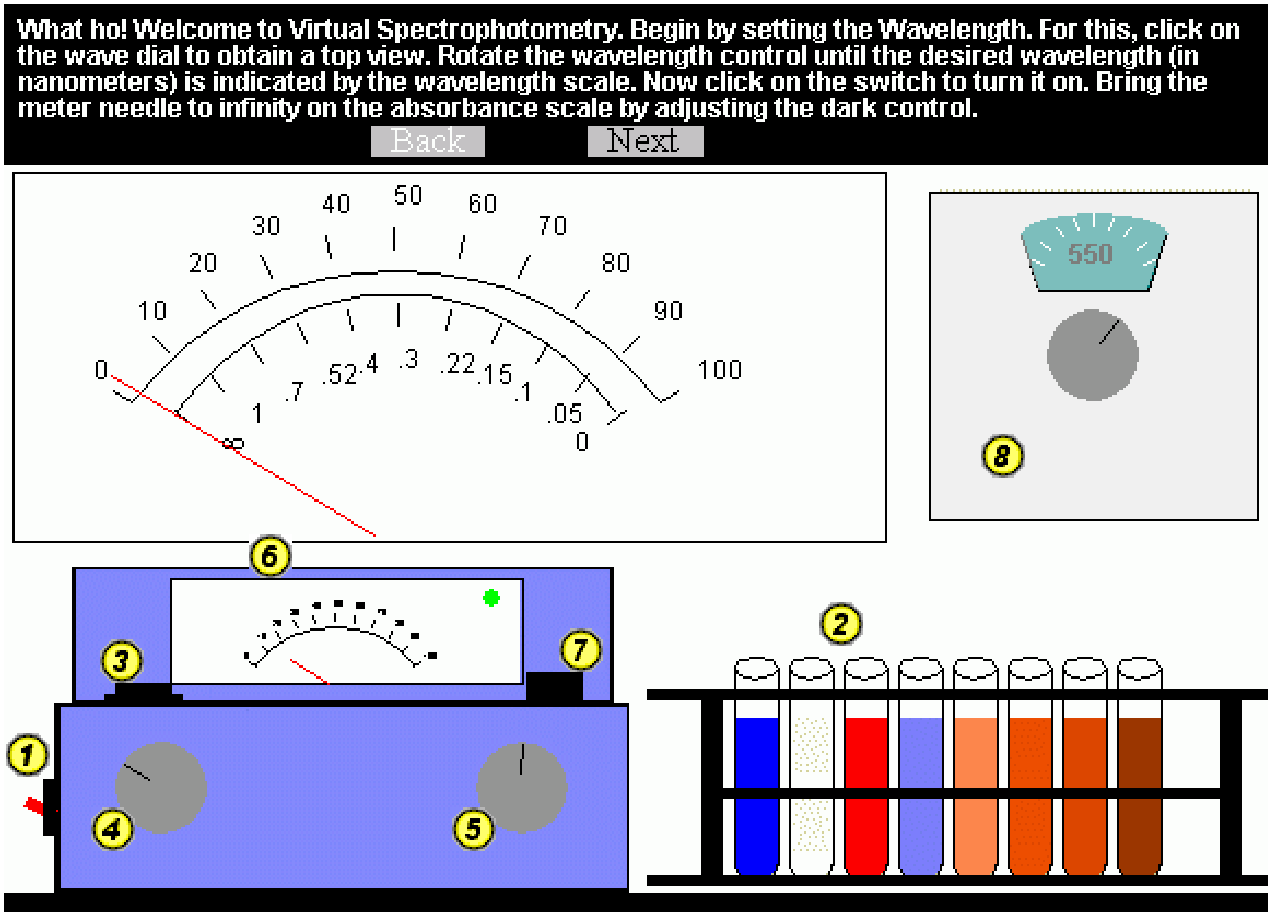
Collaboration diagram for direct manipulation



Technical Challenges

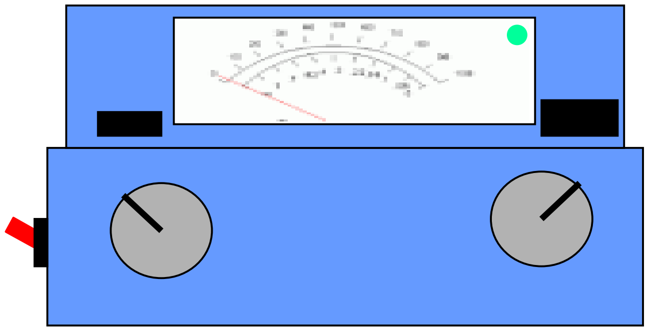
- Software Architecture
  - Web-based (Java Applets)
  - Extensible; Multipurpose; Rapid development
- Cost vs. Quality of Service
  - Visualization fidelity; Interaction agents
  - Responsiveness
- Framework for Application Development
  - Application specification language; End-user programming
  - Easy modifications
- Human Factors
  - Usability
  - Educational impact

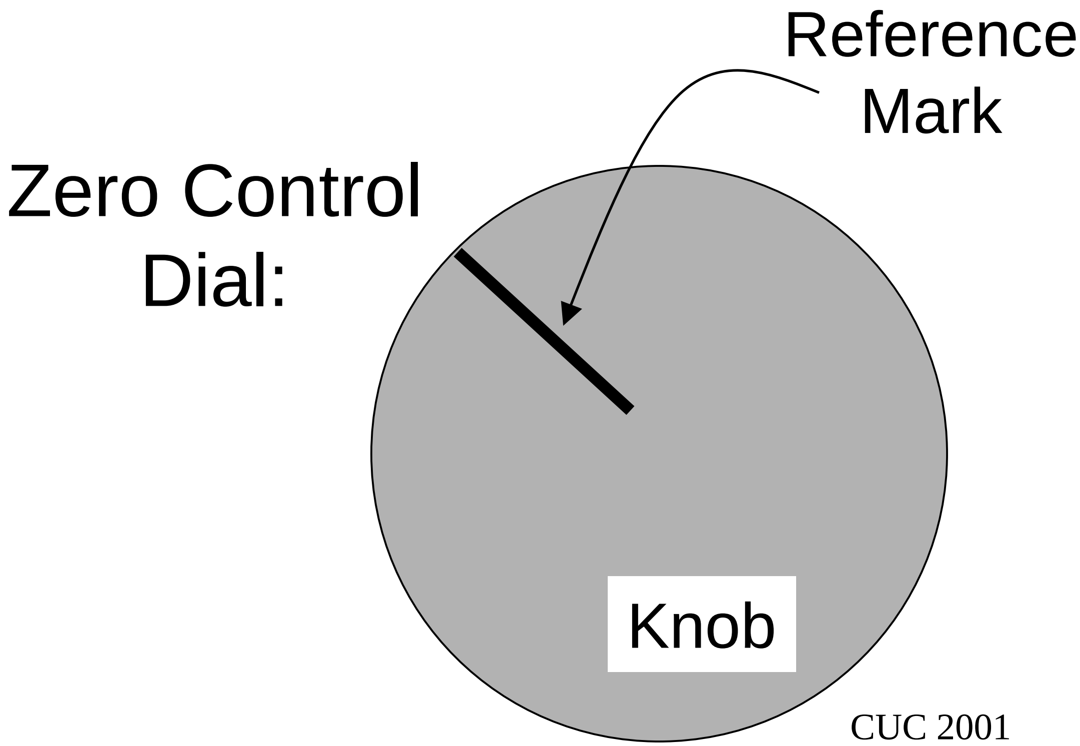
Spectrophotometer Lab

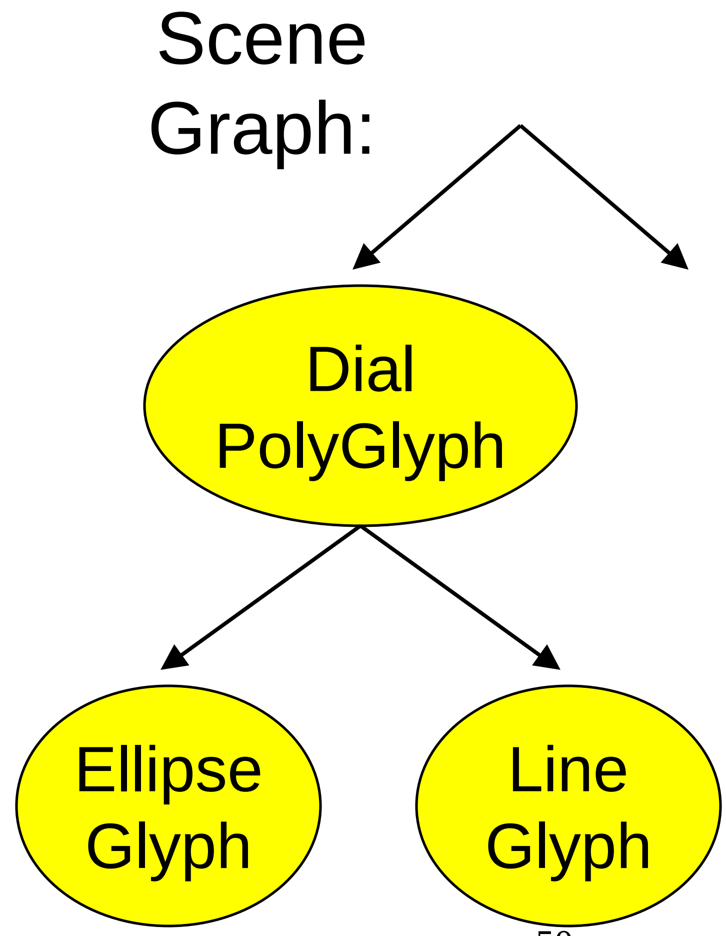


Measures the concentration of a substance in a solution and displays the % transmittance of light received by the photocell

Building Spectro Lab GUI



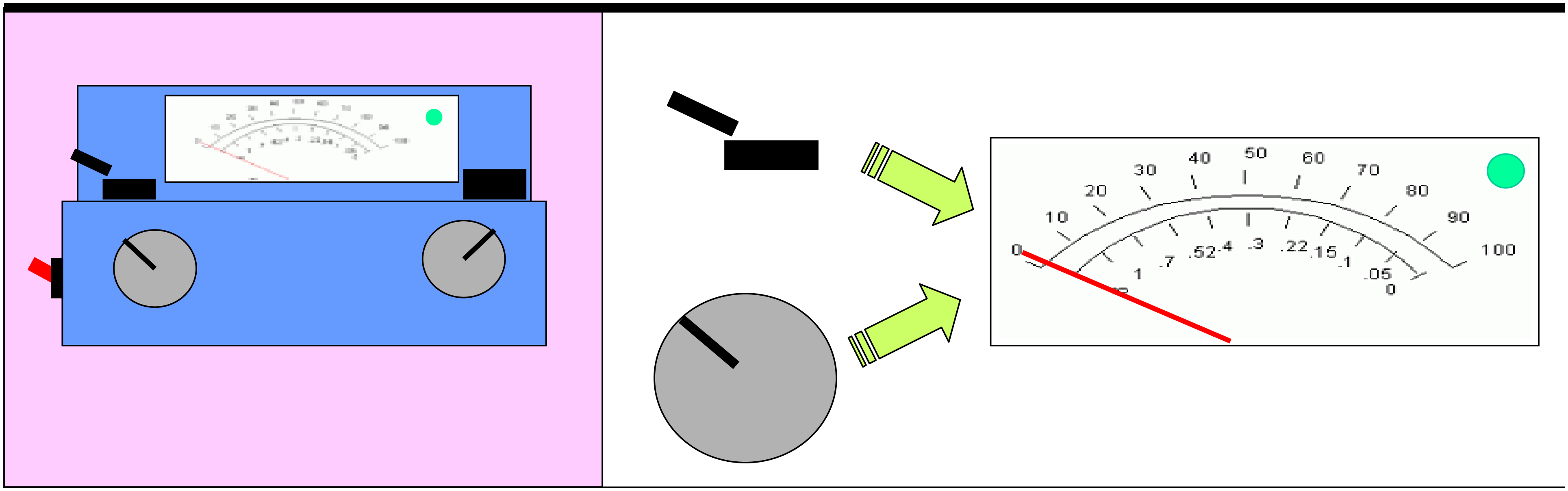


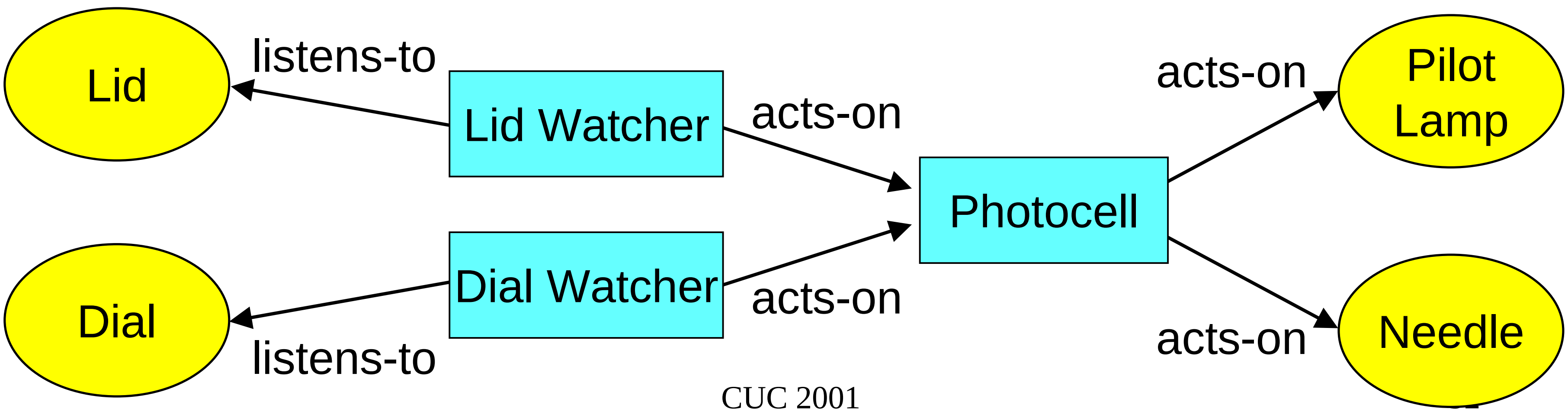


SpectroLab GUI Programming in XML

```
<POLYGLYPH  id="zeroDial“  type="flatscape.domain.PolyGlyph2D">
 <PROPERTY name="glyph.permittedUserTransform“
  type="java.lang.String“  value="rotate" />
 <PROPERTY name="glyph.dialType“ type="java.lang.String" value="zer
 <TRANSFORMATION type="flatscape.domain.Transform2D“
   value="79.0 495.0 1.0 1.0 0.0 0.0 6.5" />

 <GLYPH  id="zeroDialKnob"  type="flatscape.domain.EllipseFigure">
   <PROPERTY name="glyph.height"  type="java.lang.Double" value="42.0" /
   <PROPERTY name="glyph.width“   type="java.lang.Double" value="42.0" />
   <PROPERTY name="fill.color"  type="java.awt.Color“
    value="java.awt.Color[r=150,g=150,b=150]" />
   <TRANSFORMATION  type="flatscape.domain.Transform2D“
    value="0.0 0.0 1.0 1.0 0.0 " />
 </GLYPH>

 <GLYPH id="zeroDialReferenceMark"   type="flatscape.domain.LineFigure
   <PROPERTY name="glyph.length"   type="java.lang.Double" value="13.5" /
   <TRANSFORMATION   type="flatscape.domain.Transform2D“
    value="0.0 -13.0 1.0 1.0 -1.57 0.0 0.0"/>
 </GLYPH>
</POLYGLYPH>
```

Specifying Spectro Behavior





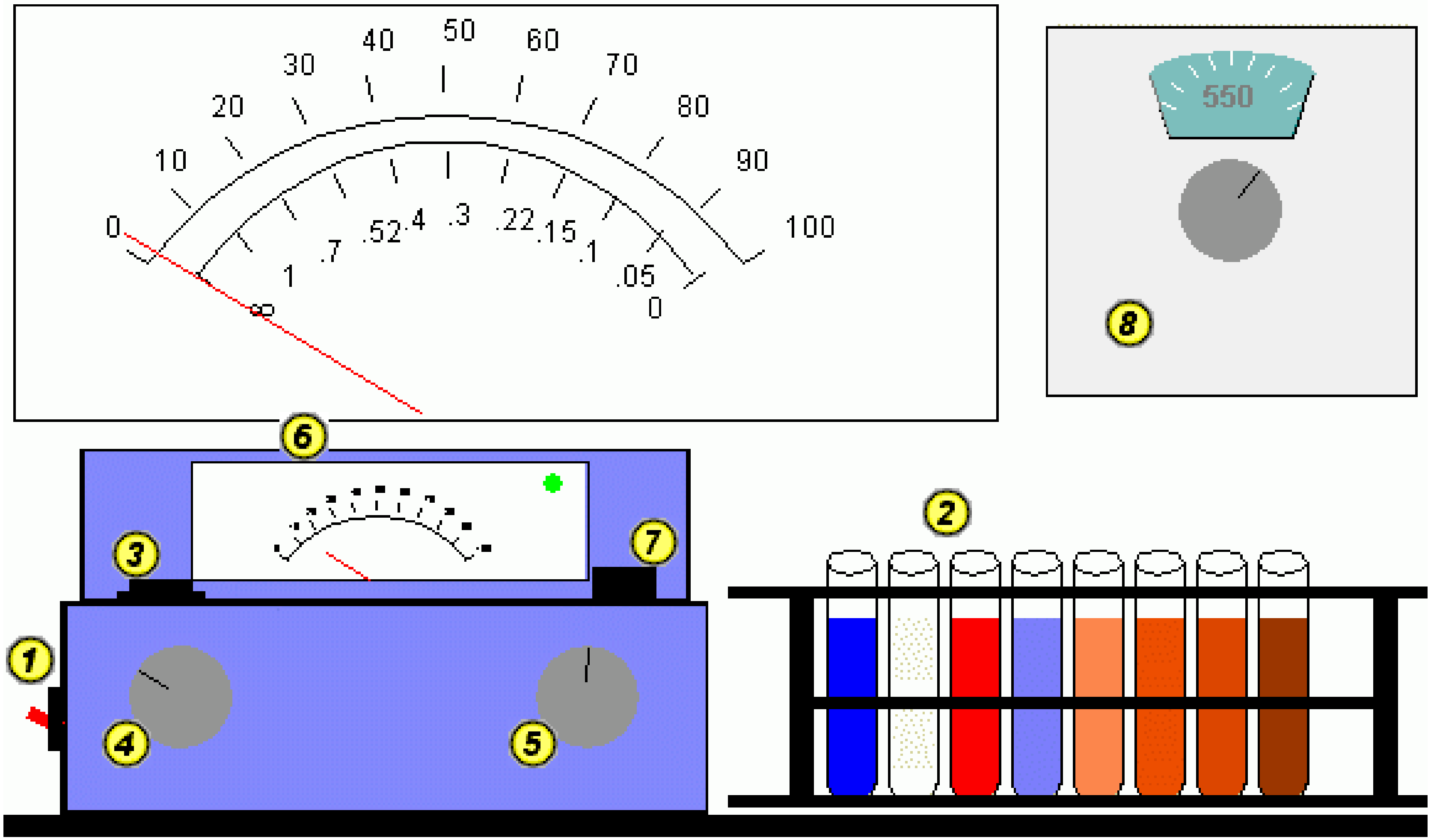
Spectro Behavior Programming in XML

```
<BEHAVIOR id="photocell" type="biology.spectro.domain.Photocell">
  <TARGET name="pilotLamp" ref="pilotLamp" />
  <TARGET name="needle" ref="needle" />
</BEHAVIOR>

<BEHAVIOR id="opening"   type="biology.spectro.domain.LidWatcher">
  <LISTENER type="manifold.domain.PropertyValueChangeListener"
    source="sampleHolder" />
  <TARGET name="lightMeasure" ref="photocell" />
</BEHAVIOR>

<BEHAVIOR id="turning"   type="biology.spectro.domain.DialWatcher">
  <LISTENER  type="manifold.domain.TransformListener" source="lightDial
  <LISTENER  type="manifold.domain.TransformListener" source="zeroDial"
  <TARGET name="lightMeasure"  ref="photocell" />
</BEHAVIOR>
```

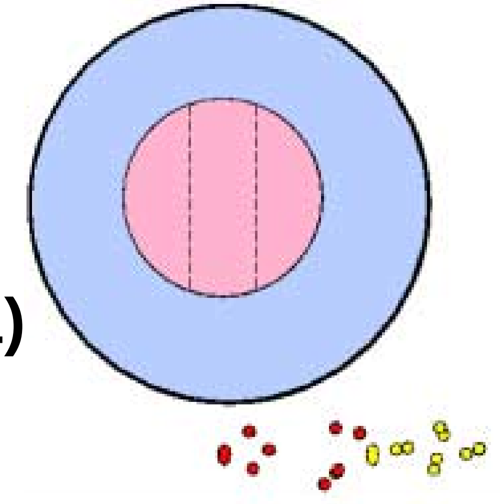
Spectrophotometer Lab (2)

What ho! Welcome to Virtual Spectrophotometry. Begin by setting the Wavelength. For this, click on the wave dial to obtain a top view. Rotate the wavelength control until the desired wavelength (in nanometers) is indicated by the wavelength scale. Now click on the switch to turn it on. Bring the meter needle to infinity on the absorbance scale by adjusting the dark control.

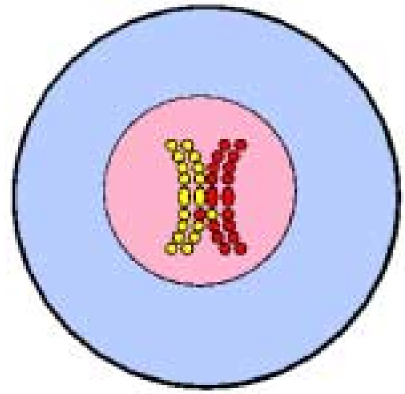
Back Next



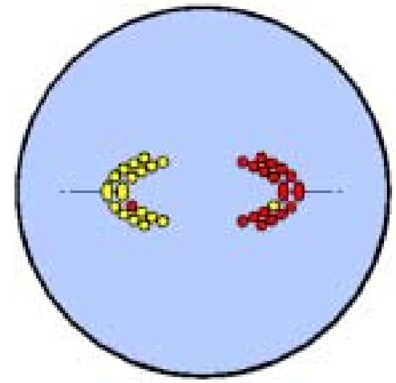
Meiosis Lab



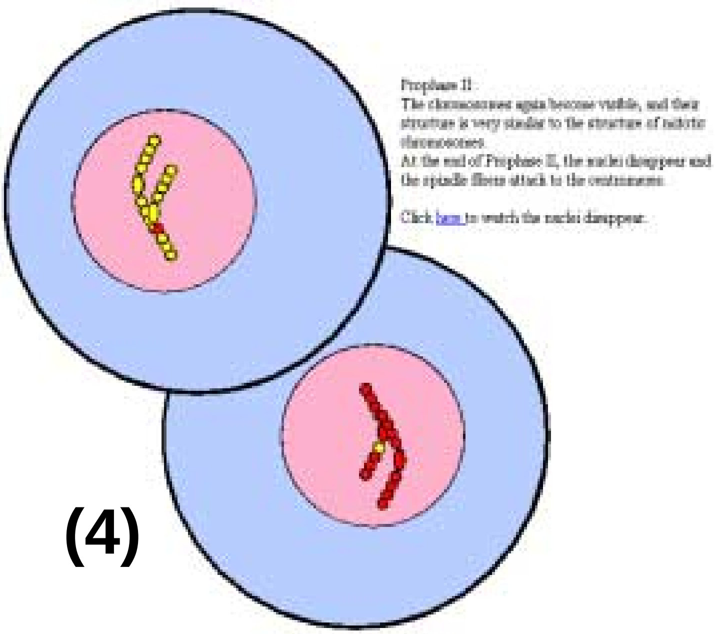
(1)



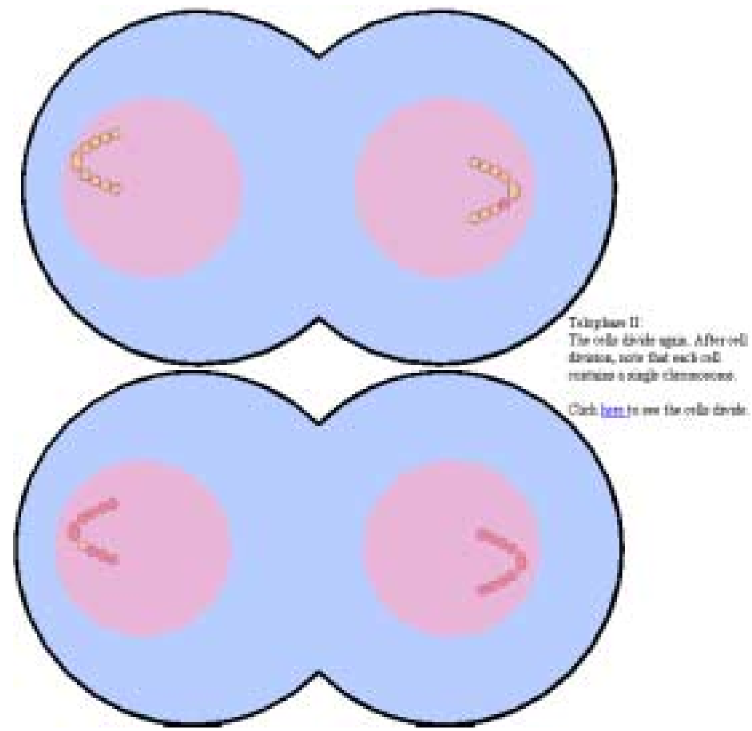
(2)



(3)



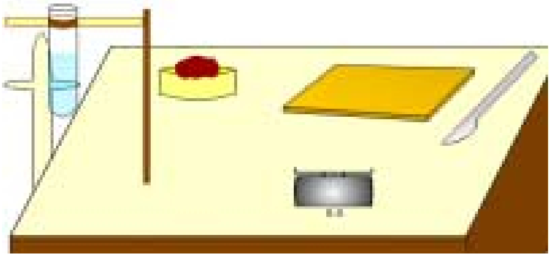
(4)



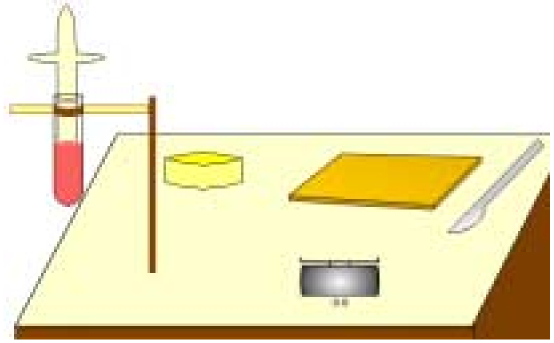
(5)

CUC 2001

Differential Centrifugation Lab

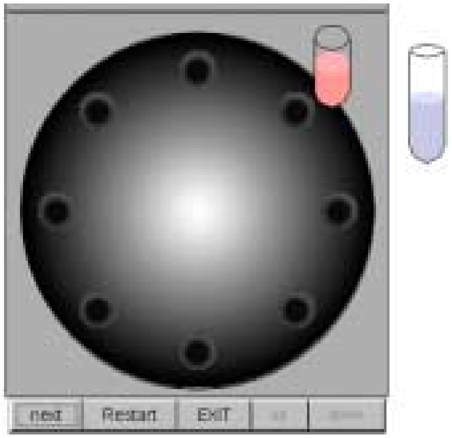


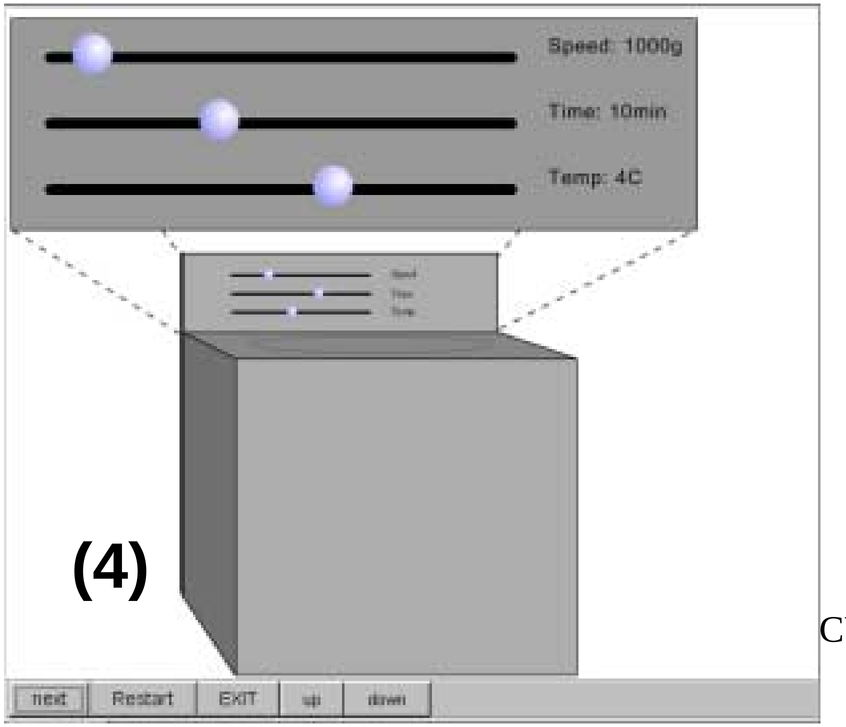
(1)



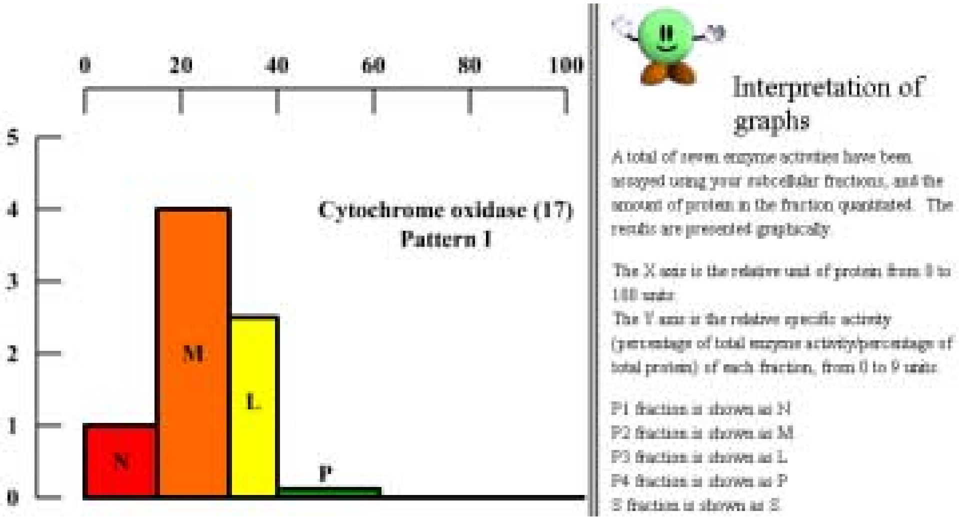
(2)

(3)



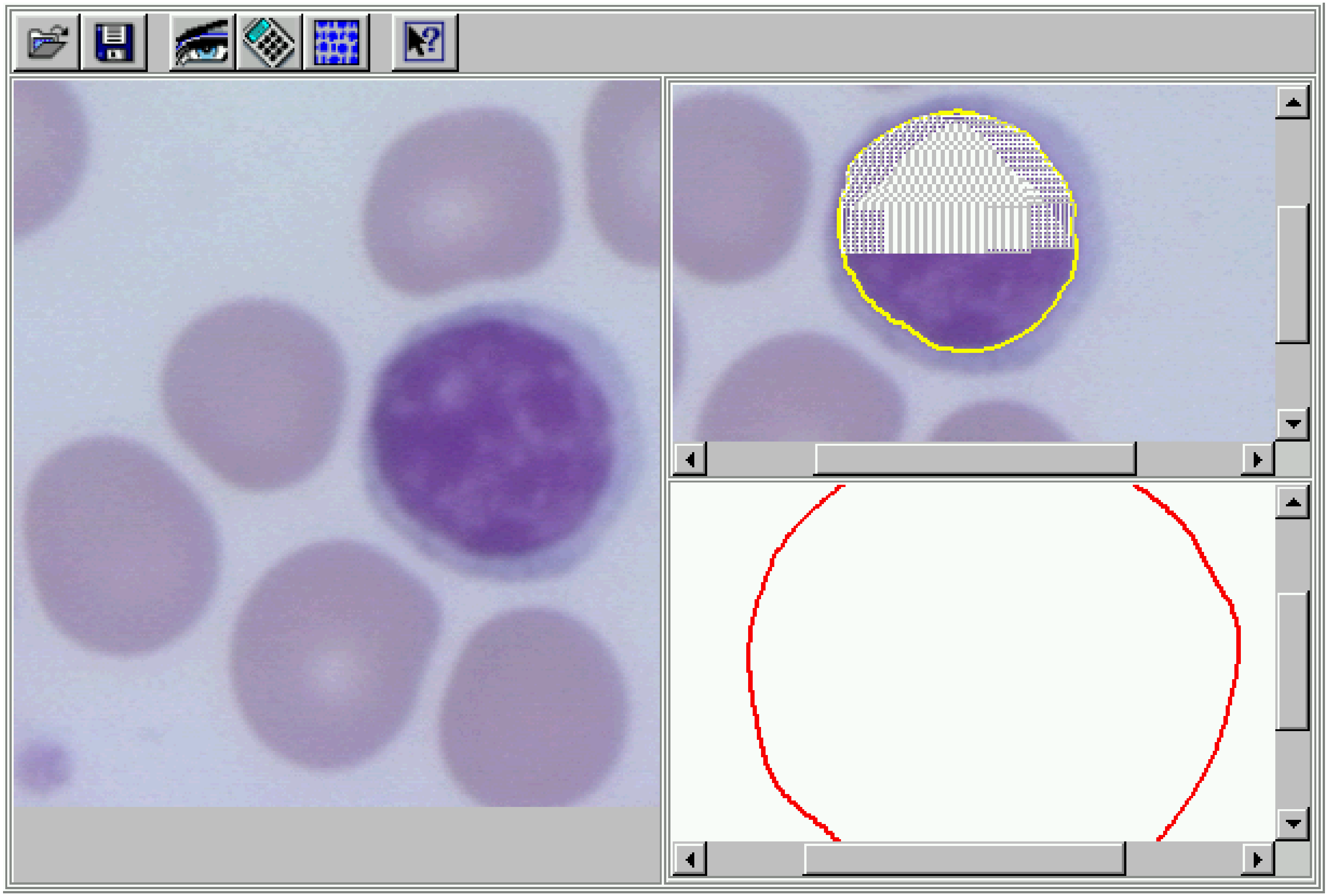


(4)

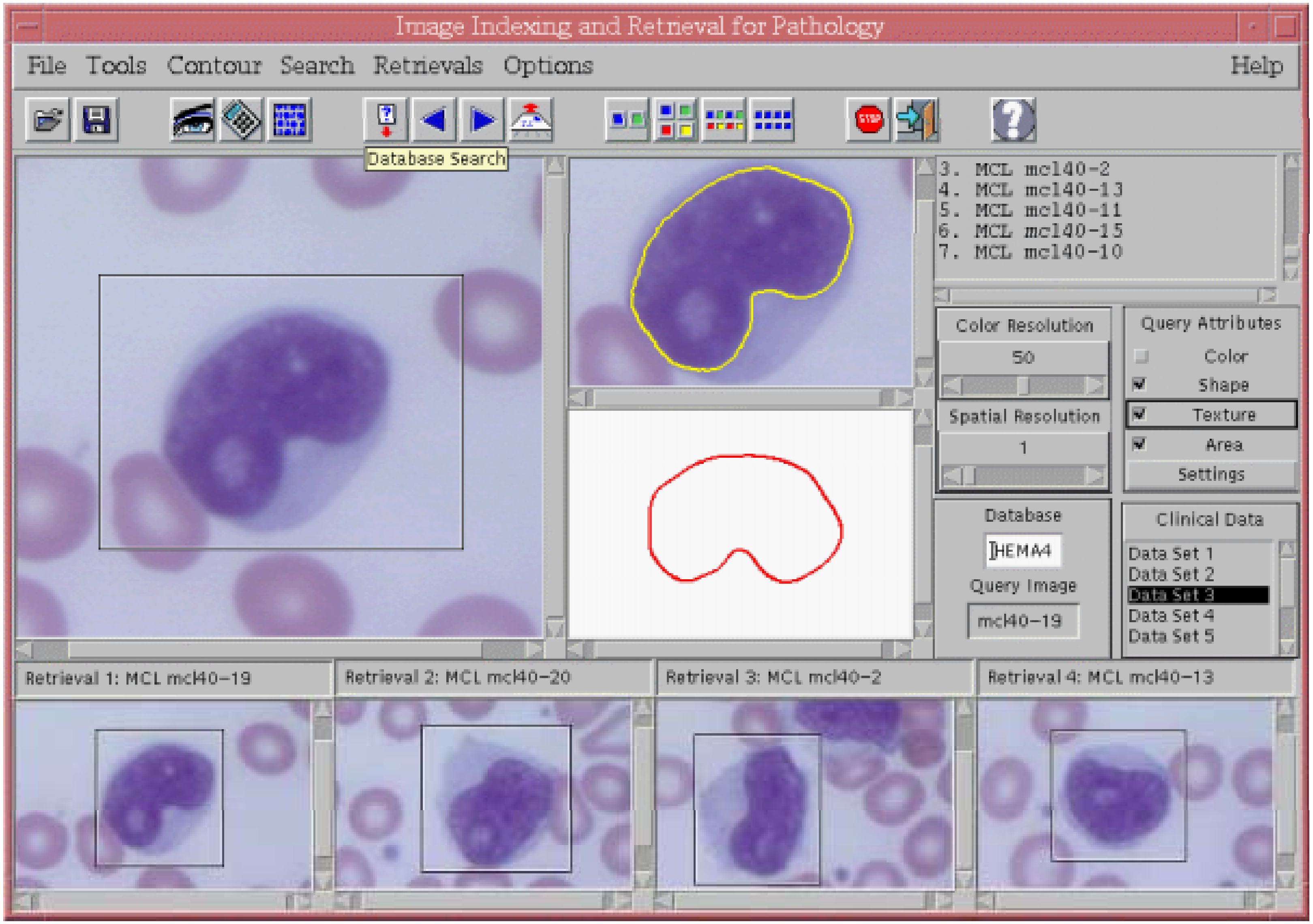


(5)

Virtual Microscope Lab



Medical Diagnosis Support Application



Classroom Evaluation

- Evaluation results encouraging
  - Observations show that interface design choices are mostly correct
  - Student surveys show that students found the labs useful
  - Student performance studies currently done by Rutgers Dept. Education
- Reduced need for teacher intervention
- Students liked interactive engagement
  - Increased student interest and control
  - Passive viewing and listening pre-recorded lectures not popular
- Exposed design problems & missing features
  - Need for user-centered design
  - Need for expert-system-based automatic help and guidance

Conclusions

- Software architecture for rapid development of virtual laboratories
  - Used to develop five example virtual biology labs
- Easy programming and modification
  - XML-based scripting language
- Labs evaluated in classroom
  - Evaluation results and student comments demonstrate the value of the labs
  - Currently a supplement to physical labs

Continuing Work

- Expert System Help
  - JESS: Java expert system shell
  - Servlets for performance
- Talking Faces
  - Increase user engagement
- Distributed Real-time Collaboration
- Behavior Programming
  - Use procedural scripting language?
- Classroom Evaluation
  - Joint work with Rutgers Department of Education

Try It!

Source code as well as publications and further information available at:

`http://www.caip.rutgers.edu/disciple/`

THE STATE UNIVERSITY OF NEW JERSEY
RUTGERS