

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

Fremdprojektanalyse

1 INTROSPEKTION

Zusammenfassung

Dieses Dokument soll Ihnen bei der Erstellung eines Plugins für Protégé behilflich sein. Es wurde der Versuch unternommen, den Inhalt so zu formen, dass er den Kollegen hoffentlich weiter hilft. UML-Diagramme sind zwar schöner Wandschmuck für einsame Wintertage, persönlich hilft mir Quellcode aber am schnellsten auf die Sprünge. Damit habe ich wahrscheinlich bewusst gegen Vorlagen von Balzert o.a. verstoßen. Falls das Dokument nicht Ihren Anforderungen genügt, bitte ich um Nachsicht. Bitte beachten Sie auch, dass Sie zum besseren Verständnis mit den Grundlagen der *javax.swing*¹ API vertraut sein sollten.

1 Introspektion

1.1 Allgemeines

Bei Protégé² handelt es sich um einen freien, in der Programmiersprache Java entwickelten Ontologieeditor³. Protégé ist durch ein strukturiertes Pluginkonzept beliebig erweiterbar und bietet eine fundierte Grundlage für Eigenentwicklungen, die eine Wissensbasis⁴ einbeziehen.

Das *Protégé OWL Plugin*⁵ ist eine Erweiterung für Protégé, und bietet die Möglichkeit mit OWL- und RDF-Dateien⁶ zu operieren.

Protégé wie auch das *Protégé OWL Plugin* stellen eine frei verfügbare Java-API bereit.

1.2 Operatives

Es wird ein J2SE Development Kit (JDK) in der Version 1.4.x benötigt. Das JDK ist für viele der gängigen Betriebssysteme wie IRIX und AIX erhältlich.

Desweiteren werden die Quellen von Protégé und des Protégé OWL Plugin benötigt.

Im folgenden Verlauf wird auf die Version 3.0 von Protégé und Version 1.3 des Protégé OWL Plugins bezug genommen.

¹*javax.swing* API: <http://java.sun.com/j2se/1.4.2/docs/api/javaw/swing/package-summary.html>

²Website des Protégé-Projektes: <http://protege.stanford.edu/>

³Funktionalitätsvergleich von Ontologieeditoren: http://www.xml.com/2002/11/06/Ontology_Editor_Survey.html

⁴Eine Wissensbasis, im weiteren auch mit dem englischen Begriff *knowledge base* referenziert, ist der Bereich eines Systemes, welcher Fachwissen in einer beliebigen Repräsentationsform enthält.

⁵Website des *Protégé OWL Plugin*: <http://protege.stanford.edu/plugins/owl/index.html>

⁶Einführung in die *Web Ontology Language*: <http://www.xml.com/pub/a/2003/08/20/deviant.html>. Spezifikation der OWL: <http://www.w3.org/TR/owl-features/>. Spezifikation des *Resource Description Framework* (RDF): <http://www.w3.org/TR/2002/WD-rdf-concepts-20021108/>

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

2 *PRODUKTÜBERSICHT*

2 Produktübersicht

2.1 Protégé

Protégé bietet verschiedene Einsatzmöglichkeiten - so ist es möglich, Protégé als Einzelplatzapplikation, als Applet oder als Server für den kooperativen Zugriff zu betreiben. Im weiteren werden wir uns auf den Einsatz als Einzelplatzapplikation begrenzen.

Pluginvariationen

Protégé bietet vielfältige Möglichkeiten, Erweiterungen transparent einzubinden. Die relevanten Plugintypen werden im folgenden aufgelistet.

Import-Plugin Importieren von Fremdformaten in die Wissensbasis von Protégé

Tab-Plugin Erweiterung um ein neues Tab mit Eigenfunktion

2.2 Protégé OWL Plugin

Das Protégé OWL Plugin ersetzt und erweitert die Benutzeroberfläche in der Hinsicht, dass sie auf die Anwendung mit OWL-Ontologien zugeschnitten wird.

Pluginreferenz

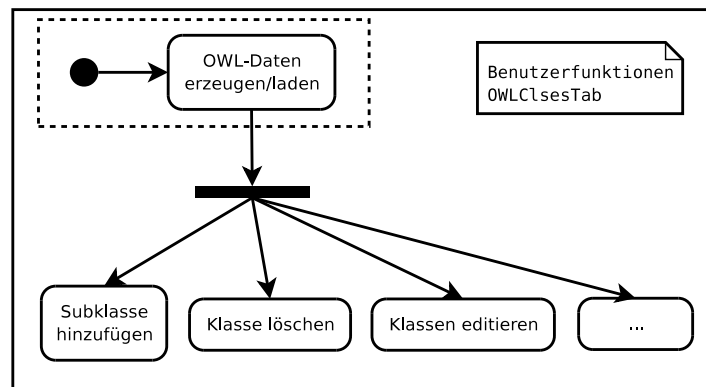


Abbildung 1: Benutzerfunktionen des *OWLClasesTab*

Exemplarisch werden wir uns auf das *OWLClasesTab* aus dem Protégé OWL Plugin festlegen. Es ersetzt das *ClasesTab* welches Protégé bereit stellt.

Das *OWLClasesTab* ermöglicht dem Benutzer, über den Klassen einer Ontologie zu operieren. Hierunter fallen Operationen wie das anzeigen, hinzufügen und durchsuchen von Klassen.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3 *DESIGNSCHWERPUNKTE***3 Designschwerpunkte**

Die im folgenden abgehandelten Schwerpunkte sollen Ihnen ein möglichst umfassendes Bild von Protégé in Konjugation mit dem Protégé OWL Plugin vermitteln.

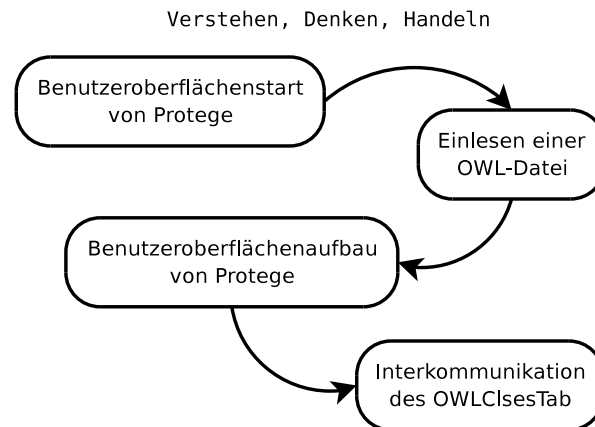


Abbildung 2: Designschwerpunkte

3.1 Benutzeroberflächenstart von Protégé

Im folgenden wird der Startablauf der Benutzeroberfläche von Protégé beschrieben, wobei nur die essenziellen Schritte näher erläutert werden. Protégé bietet verschiedene Möglichkeiten die Benutzeroberfläche zu initialisieren, wir aber werden auf den Applikationsmodus fokussieren.

Der Quellcode für den Startablauf als Applikation befindet sich in der Datei *Application.java* im Hauptverzeichnis der Quellen von Protégé.

Zuerst wird ein *ThreadGroup*⁷ erzeugt, und die Funktion *uncaughtException()* überladen, um alle unabgefangenen Exceptions aufzeichnen zu können.

```

// edu.stanford.smi.protege
// Application.java
ThreadGroup group = new ThreadGroup(...) {
    public void uncaughtException(...) {
        Log.getLogger().log(
            Level.SEVERE, "Uncaught Exception", throwable
        );
    }
};
  
```

⁷*Thread Groups* erlauben es, einen Thread welcher wiederum viele Threads startet effizient zu kontrollieren. Thread Groups sollten aber außer für den obigen Anwendungsfall nicht verwendet werden. Eine Beschreibung der Thread Groups finden Sie unter: http://www.geocities.com/herong_yang/java/thread_group.html

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.1 Benutzeroberflächenstart von Protégé

3 DESIGNSCHWERPUNKTE

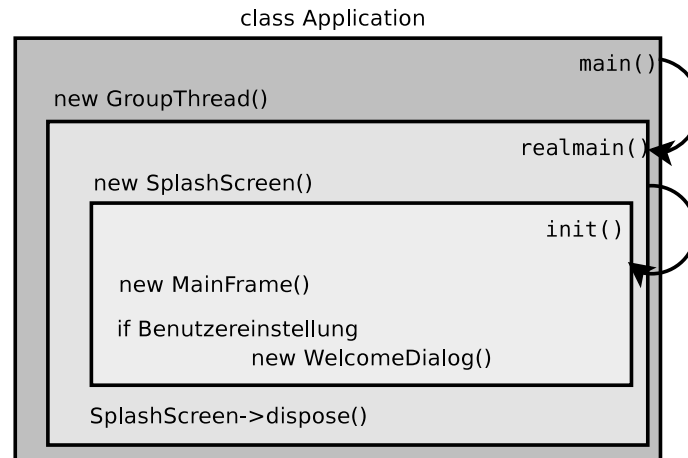


Abbildung 3: Programmablauf beim Benutzeroberflächenstart von Protégé

Bei dieser Gelegenheit soll nachdrücklich auf die vorzügliche *Java Logging API*⁸ hingewiesen werden. Das *java.util.logging* Paket erlaubt es, auch Tracing-Ausgaben zu protokollieren - und stellt somit auch im Rahmen der zu entwickelnden Software ein Framework für das Debugging bereit. Desweiteren stellt auch Protégé eine erweiterte Klasse *edu.stanford.smi.protege.util.Log* für das Logging bereit, diese bietet sich also besonders an.

Level	Bedeutung
SEVERE	Applikationsfehler
WARNING	Problem benachrichtigung
INFO	Nachricht
CONFIG	Konfigurationsinformationen
FINE, FINER, FINEST	Tracing in drei Detailstufen

Abbildung 4: Java Logging API Levels

Im nächsten Schritt wird ein neuer Thread-Daemon erzeugt, mit welchem die eigentliche Applikationslogik verknüpft wird. Er wird der soeben erzeugten ThreadGroup *group* untergeordnet.

```

Thread thread = new Thread(group, "Safe Main Thread") {
    public void run() {
        // realmain() erzeugt den SplashScreen und ruft init().
        realmain(...);
    }
}; thread.setDaemon(true); thread.start();

```

⁸Eine Einführung in die *Java Logging API* finden Sie unter: <http://www.onjava.com/pub/a/onjava/2002/06/19/log.html>

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.2 Einlesen einer OWL-Datei**3 DESIGNSCHWERPUNKTE**

Ein Thread-Daemon muss nicht explizit beendet werden. Wenn er der letzte überlebende Thread ist, beendet sich der Java-Interpreter von alleine.

Nun wird das *MainFrame* erzeugt, es handelt sich dabei um ein erweitertes *JFrame*.

```
// init()
// Construct the application's main frame.
_mainFrame = ComponentFactory.createMainFrame();
...
showMainFrame(); // _mainFrame.setVisible(true)
```

Wichtig und zu beachten ist hierbei, dass der Aufruf über die *util.ComponentFactory* erfolgt. Diese Klasse bietet eine Art Wrapper für die meisten Swing-Elemente, um diese dem Protégé eigenen *look and feel* anzupassen. Die Verwendung der *ComponentFactory* ist nicht zwingend, sollte aber nach Möglichkeit forciert werden.

Wenn der Benutzer in seinen Einstellungen den *WelcomeDialog* nicht deaktiviert hat, wird dieser als letzter Schritt angezeigt. Der *WelcomeDialog* erlaubt dem Benutzer zum Beispiel das Laden⁹ einer Ontologie.

```
// init()
// Check to see if the user wants to see the welcome dialog.
boolean b = ApplicationProperties.getWelcomeDialogShow();

if (b == true) { // Show the welcome dialog.
    _welcome = new WelcomeDialog(
        _mainFrame, Text.getProgramName(), true
    );
    _welcome.setLocationRelativeTo(_mainFrame);
    _welcome.setVisible(true);
}
```

Über die Klasse *util.ApplicationProperties* lassen sich die Einstellungen von Protégé abfragen. Mittels der Klasse *resource.Text* lässt sich z.B. die Protégé Build-Nummer abfragen. Für die nicht benötigten modalen Dialoge¹⁰ oder neuen *JFrames* ist der Aufruf *setLocationRelativeTo(_mainFrame)* interessant, hiermit lässt sich ein neues *JFrame* relativ zum *MainFrame* positionieren.

3.2 Einlesen einer OWL-Datei

Nachdem Protégé gestartet ist, wurden auch die Plugins bereits initialisiert. Protégé besitzt ein OWL-Plugin *edu.stanford.smi.protege.owl*, es handelt sich dabei um ein Project-Plugin - diesem ist es erlaubt den Projekttyp und die Benutzeroberfläche zu verändern. Für das Einlesen von OWL-Dateien ist die *storage.JenaKnowledgeBaseFactory* Klasse - ein Storage-Factory Plugin - verantwortlich, welche die Klasse *KnowledgeBaseFactory* erweitert. Als OWL-API findet das Jena-Framework¹¹ Verwendung.

⁹Das Laden einer OWL-Ontologie geschieht über den Button *Build* im *WelcomeDialog*. Ich habe das erst nach einer halben Stunde herausgefunden, und auch nur weil ich mir den Quellcode angesehen habe. Falls sie beim ersten Anlauf mit der Benutzeroberfläche nicht zurecht kommen, verzweifeln Sie nicht, es liegt nicht an Ihnen.

¹⁰Warum modale Dialoge nicht verwendet werden sollen: <http://www.plethora.net/~seebs/ops/ibm/cranky12.html>

¹¹Das Jena-Framework steht unter BSD-Lizenz und ist ein Projekt der HP Labs, siehe: <http://www.hpl.hp.com/semweb/jena.htm>

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.3 Benutzeroberflächenaufbau von Protégé3 DESIGNSCHWERPUNKTE

Der Benutzer initialisiert das Einlesen einer OWL-Datei zum Beispiel über den *Welcome-Dialog*.

```
// JenaKnowledgeBaseFactory.java
public void loadKnowledgeBase(KnowledgeBase kb, ...) {
    if (kb instanceof JenaOWLKnowledgeBase) {
        JenaOWLKnowledgeBase okb = (JenaOWLKnowledgeBase) kb;
        URI absoluteURI = getFileURI(..., okb.getProject());
        // com.hp.hpl.jena.ontology.OntDocumentManager
        OntDocumentManager.getInstance().reset(true);
        okb.load(absoluteURI, ...);
    }
}
```

Die Funktion *loadKnowledgeBase()* wird überladen, sie erhält unter anderem auch das *KnowledgeBase*¹² Objekt übergeben. Als erstes wird überprüft, ob es sich bei der *KnowledgeBase* um eine Instanz vom Objekttyp *JenaOWLKnowledgeBase* handelt - kann dies bejaht werden, so wird das *KnowledgeBase* Objekt nach *JenaOWLKnowledgeBase* gecastet. Im nächsten Schritt wird die absolute *URI*¹³ des Projektes erzeugt, wobei *getFileURI()* eine Funktion der *JenaKnowledgeBaseFactory* Klasse selbst ist. Bei *ontology.OntDocumentManager* handelt es sich um eine Klasse aus dem *com.hp.hpl.jena* Namespace. *OntDocumentManager* dient der Verwaltung von Ontologiedokumenten. Der Aufruf *reset(true)* setzt ihn in den Ursprungszustand zurück. Letztendlich wird das OWL-Dokument mittels *okb.load()* eingelesen.

```
// JenaOWLKnowledgeBase.java
public void load(URI uri, ...) {
    try {
        // jena.loader.JenaLoader.loadFile()
        JenaLoader.loadFile(uri, ...);
    } catch (Throwable t) {
        t.printStackTrace();
    }
}
```

jena.loader.JenaLoader.load() schreibt sodann die *JenaOWLKnowledgeBase* in die *KnowledgeBase* von Protégé.

3.3 Benutzeroberflächenaufbau von Protégé

Die Benutzeroberfläche von Protégé setzt sich im wesentlichen aus drei für uns relevanten Hauptkomponenten zusammen.

- *ProjectMenuBar* repräsentiert ein erweitertes *JMenu*, welches z.B. auch mittels eines Projekt-Plugins um etwaige weitere pluginspezifische Elemente erweitert werden kann.
- *ProjectToolBar* ist ein erweitertes *JToolBar*, es kann ebenfalls nach belieben angepasst werden.

¹²Eine *KnowledgeBase* ist ein Container für Frames. Frames sind Konstrukte zur Wissensrepräsentation, wie z.B. Fakten und Regeln.

¹³URLs und URNs werden auch unter dem Begriff URI zusammengefasst.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.4 Das OWLClsesTab

3 DESIGNSCHWERPUNKTE

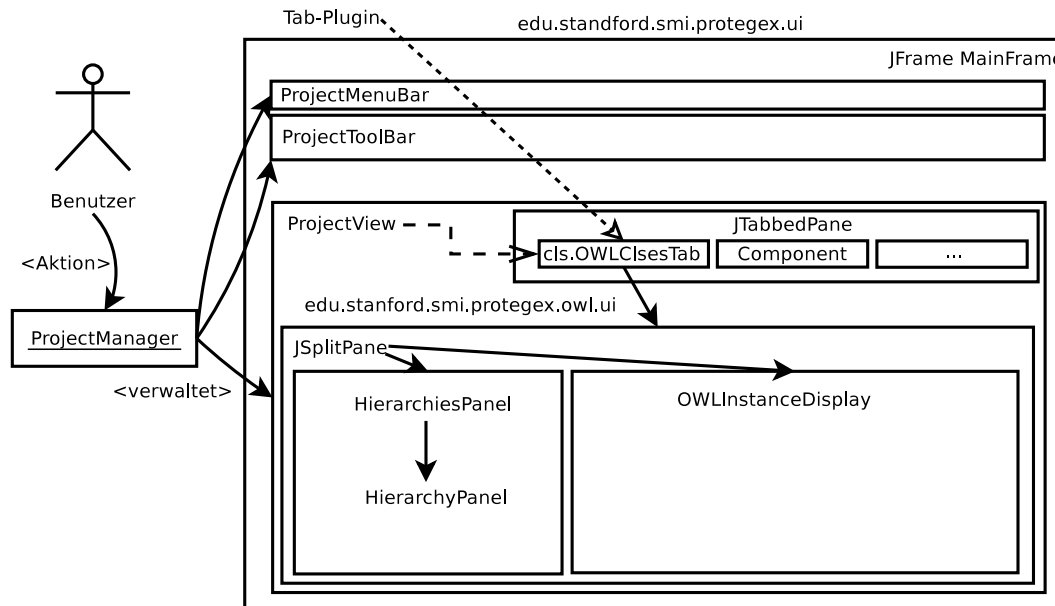


Abbildung 5: Benutzeroberflächenaufbau von Protégé

- *ProjectView* ist von besonderer Bedeutung, denn es nimmt die Tab-Plugins in einer *JTabbedPane* auf.

Alle drei Komponenten werden mit dem *JFrame MainFrame* verankert. Das *JTabbedPane* - verankert auf dem *ProjectView* - nimmt in unserem Beispiel auch das *OWLClsesTab* auf.

Grundsätzlich verwaltet werden diese Komponenten von dem *ProjectManager*. Er bietet beispielsweise Schnittstellen um neue Projekte zu erzeugen, zu laden und zu speichern. Nach dem Öffnen eines neuen Projektes kümmert er sich auch darum, dass die beschriebenen Komponenten ihren angestammten Platz finden.

3.4 Das OWLClsesTab

Beim *OWLClsesTab* handelt es sich um ein Tab-Plugin welches die abstrakte Klasse *AbstractTabWidget* erweitert.

3.4.1 Project

Zum besseren Verständnis sei hier die Klasse *Project* kurz umrissen. Bei der Klasse *Project* handelt es sich um eine wichtige und übergreifend verwendete Klasse. Sie umfasst beispielsweise die *KnowledgeBase* und die *URI* des gerade geladenen Projektes.

Ein neues *Project* wird mit *Project.createNewProject(null, errors)* erzeugt, wobei es sich bei *errors* um eine *ArrayList* handelt, die zur Fehlerprüfung verwendet wird. Um ein Protégé-Projekt (.prj-, .pins- und .pons-Dateien) einzulesen, genügt der Aufruf *p.loadProjectFromURI(projectURI, errors)*. Die aus dem eingelesenen Projekt erzeugte *KnowledgeBase* kann jederzeit mittels *p.getKnowledgeBase()* erreicht werden.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.4 Das OWLClasesTab**3 DESIGNSCHWERPUNKTE****3.4.2 JenaOWLKnowledgeBase**

Um eine Ontologie direkt in eine *JenaOWLKnowledgeBase* zu laden, wäre beispielsweise folgende Forgehensweise denkbar:

```
JenaOWLKnowledgeBase okb =
    (JenaOWLKnowledgeBase) p.getKnowledgeBase();
okb.load(new URI("http://indigo2.lan.dac/wine.rdf"));
```

3.4.3 Aufbau des OWLClasesTab

Das *OWLClasesTab* erweitert wie bereits erwähnt *AbstractTabWidget*, welches wiederum das *AbstractWidget* erweitert.

- *OWLClasesTab* -> *AbstractTabWidget* -> *AbstractWidget* -> *JPanel*

Das *AbstractWidget* bietet unter anderem auch die Funktionen *get-* und *setProject()* an. Somit besitzen wir jederzeit Zugriff auf das gerade geladene Projekt.

Es soll nun verfolgt werden, wie das *OWLClasesTab* aufgebaut wird.

```
// edu.stanford.smi.protege.owl.ui.cls
// OWLClasesTab.java
// initialize() wird automatisch beim Laden des Plugins
// aufgerufen.
public void initialize() {
    setLabel("OWLClasses"); // Label setzen
    // Erzeugen des JSplitPane.
    sp = ComponentFactory.createMainSplitPane();
    // JSplitPane dem OwlClasesTab hinzufügen.
    add(sp);
}
```

Hiermit wurden dem *OWLClasesTab* alle Komponenten hinzugefügt. Um aber den Zugriff auf das Projekt zu erschließen, müssen wir uns die Funktion *createMainSplitPane()* genauer ansehen.

```
// edu.stanford.smi.protege.owl.ui.cls
// OWLClasesTab.java
private JSplitPane createMainSplitPane() {
    // ComponentFactory erzeugt JSplitPane.
    sp = ComponentFactory.createLeftRightSplitPane(...);
    // OWLInstanceDisplay auf JComponent verankern.
    // Das Übergabeobjekt vom Typ Project.
    JComponent instanceDisplay =
        new OWLInstanceDisplay(getProject());
    hierarchiesPanel = new HierarchiesPanel(this);
    hierarchiesPanel.addHierarchyPanel(
        new HierarchyPanel(...))
};
// HierachiesPanel links verankern.
```

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

3.4 Das OWLClusesTab

3 DESIGNSCHWERPUNKTE

```

    sp.setLeftComponent(hierarchiesPanel);
    // OWLInstanceDisplay rechts verankern.
    sp.setRightComponent(instanceDisplay);
    return sp; // JSplitPane zurückgeben
}

```

Nun lässt sich feststellen, dass das *OWLInstanceDisplay* und *HierarchyPanel* die eigentlich zuständigen Klassen sind.

Nun wird die Funktionsweise des *OWLInstanceDisplay* weiter verfolgt.

```

// edu.stanford.smi.protege.owl.ui.instancedisplay
public class OWLInstanceDisplay extends InstanceDisplay {
    ...
    public OWLInstanceDisplay(Project p) {
        super(p);
    }
}

```

OWLInstanceDisplay erweitert *InstanceDisplay* aus dem Namensraum von Protégé. *InstanceDisplay* bietet bereits alle benötigten Funktionen an, *OWLInstanceDisplay* schneidert es auf den spezifischen Anwendungsfall zurecht. Durch den Aufruf von *super(p)* wird *InstanceDisplay* das *Project p* direkt übergeben.

```

// edu.stanford.smi.protege.ui
public class InstanceDisplay {
    private Project _project;
    private Instance _currentInstance;

    // Konstruktor
    public InstanceDisplay(Project p) {
        _project = p;
    }

    // Setzt die gerade ausgewählte Instanz.
    public void setInstance(Instance instance) {
        _currentInstance = instance;
    }
}

```

Nun besitzt *InstanceDisplay* ebenfalls eine Referenz auf das Objekt von *Project*, und es können über das *OWLInstanceDisplay* Funktionen wie *setInstance()* aufgerufen werden. Ein solcher Aufruf erfolgt zum Beispiel mittels eines *ActionListeners* der auf die *JTree*-Einträge der im *HierarchyPanel* enthaltenen Klasseneinträge der gerade geladenen Ontologie gesetzt ist.

3.4.4 Klassenzugriff

Der Zugriff auf die Klassen im *OWLClusesTab* funktioniert mittels eines *SelectionListener*, welcher in diesem Beispiel auf das *AssertedClusesPanel* gesetzt wird. Das *AssertedClusesPanel* erweitert das *ClusesPanel*.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess

4 RESUME

```
// edu.stanford.smi.protegex.owl.ui.cls
// OWLClsesPanel.java
ClsesPanel acp = new AssertedClsesPanel(...);
acp.addSelectionListener(new SelectionListener() {
    public void selectionChanged(SelectionEvent e) {
        // Aufruf dieser Routine, wenn der Benutzer
        // eine Auswahl getroffen hat.
        transmitSelection();
    }
})
// transmitSelection()
protected void transmitSelection() {
    // Rückgabe einer Collection, da der Benutzer
    // mehrere JTree-Einträge auswählen kann.
    Collection selection = clsesPanel.getSelection();
    // Ersten Eintrag holen.
    selectedInstance = (Instance)
        CollectionUtilities.getFirstItem(selection);
    // Die ausgewählte Instanz setzen.
    // Die Ansicht wird aktualisiert und der Benutzer
    // sieht die Eigenschaften der ausgewählten Klasse.
    instanceDisplay.setInstance(selectedInstance);
}
```

4 Resume

- Eine *KnowledgeBase* kann nach *OWLKnowledgeBase* gecasted werden.

```
OWLKnowledgeBase okb = (OWLKnowledgeBase) kb;
```

- Die *OWLKnowledgeBase* kapselt die Daten der Wissensbasis.
- Instanzen heißen auch *Individuals*.
- Jedes *Tab* bietet spezielle Sichten auf die Daten.
- Das *OWLClsesTab* fokussiert auf die Klassensicht.
- Die *ComponentFactory* bietet einen Wrapper für Swing-Komponenten.