

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Entwurfsbeschreibung V1.5

Version	Autor	QS	Datum	Status	Kommentar
1.0	M. Czygan		19.05.2005	Referenzdokument	<i>initiale Version</i>
1.1	M. Czygan		30.05.2005		<i>erste Iteration</i>
1.2	M. Czygan		04.06.2005		<i>zweite Iteration, Anhang A</i>
1.3	M. Czygan		10.06.2005		<i>dritte Iteration, Anhang B</i>
1.4	M. Czygan		20.06.2005		<i>vierte Iteration, Anhang c</i>
1.5	M. Czygan, A. Kiess		27.06.2005		<i>fünftes Release</i>

1. Allgemeines**(a) Ziel des Produktes**

Das Produkt beinhaltet zwei Erweiterungen (Plugins) für den Ontologie-Editor Protégé in der Version 3.1 beta¹.

- Das *Import-Plugin* bietet den Import von LSGM-Dateien in eine OWL-Wissensbasis an.
- Das *InstanceXL-Plugin* ermöglicht die Anzeige der Instanzen (OWL-Individuals oder Protégé-Instanzen) einer Klasse in einer tabellarischen Übersicht. Die Tabelle kann editiert, durchsucht und mit Hilfe weiterer Optionen den Bedürfnissen des Anwenders angepasst werden. Durch eine tabellarische Sicht der Instanzen ist es möglich, schnell und unkompliziert einen Überblick über alle Instanzen einer Klasse zu erhalten.

(b) Systemvoraussetzungen

Um die Plugins nutzen zu können, wird die Rahmenapplikation Protégé in der Version 3.1 benötigt. Protégé selbst benötigt eine JVM² in der Version 1.4 oder höher. Eine JVM ist für zahlreiche Plattformen verfügbar.

Um im InstanceXL-Plugin die Export-Funktion nutzen zu können, werden die JFreeReport Libraries benötigt, die in den jeweiligen Releases des Plugins enthalten sind. Um diese Libraries unter Protégé nutzen zu können, wird das modifizierte Protégé Start-Up Skript benötigt, welches ebenfalls in allen Release-Bündeln enthalten ist - oder eine Installation der JFreeReport-Bibliotheken in das *jre/lib/ext*-Verzeichnis der Java-Installation. Weitere Informationen dazu entnehmen Sie bitte der Datei LIESMICH des jeweiligen Releases.

(c) Plugin-Konzept von Protégé

Protégé als Rahmenapplikation setzt das Konzept der Erweiterbarkeit durch Plugins explizit um. Entwickler können durch Unterstützung der Protégé-API schnell mit dem Bau eigener Erweiterungen beginnen. Anwender können ohne Umwege die Funktionalität der zur Verfügung stehenden Plugins nutzen³.

(d) Präsentation des Plugins auf unserer Projekt-Webseite

Der aktuelle Stand unserer Plugins kann auf unserer Webseite eingesehen werden. Die Präsentation erfolgt als Applet⁴ oder als Java-Web-Start⁵ Applikation. Unser Plugin finden Sie auf unserer Webseite unter:

<http://dac.icore.at/p1/prototype>

¹Build 191. Es handelt sich um eine beta-Version von Protégé; die Lauffähigkeit des Plugins für andere Builds wurde nicht explizit getestet.

²Java Virtual Machine

³Um ein Plugin in Protégé zu aktivieren, gehen Sie auf den Menüpunkt *Project | Configure*. In dem neu geöffneten Fenster sind alle zur Verfügung stehenden Plugins aufgelistet. Aktivieren Sie das Plugin, in dem Sie die CheckBox des jeweiligen Plugins anklicken.

⁴Informationen zu Applets: <http://java.sun.com/applets/>

⁵Informationen zur Java-Web-Start Technologie: <http://java.sun.com/products/javawebstart/>

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(e) Aktueller Stand dieses Dokumentes

Dieses Dokument gibt einen Überblick über den Systementwurf der Plugins zum Zeitpunkt der aktuellen Iteration. Die wöchentlich veröffentlichten Releases des Produktes enthalten jeweils eine aktualisierte Version der Entwurfsbeschreibung.

Anmerkung (V1.2): Version 1.2 enthält einen Anhang (A), in dem aktuelle Designfragen dargestellt bzw. problematisiert werden.

Anmerkung (V1.3): Version 1.3 enthält einen Anhang (B), in dem die aktuellen Umstrukturierungen beschrieben werden.

Anmerkung (V1.4): Version 1.4 enthält einen Anhang (C), in dem die aktuellen Umstrukturierungen beschrieben werden.

Anmerkung (V1.5): Version 1.5 enthält die Beschreibung des Produktes zum Abschluss der Implementierungsphase. Diese Version dient auch als Leitfaden für die Abnahme. *Anhang (C) aus Version 1.4 entfällt.*

2. Produktübersicht**(a) Erscheinungsbild**

Das *InstanceXL-Plugin* ist ein *TabPlugin*⁶ und gliedert sich nach der Aktivierung in die Reihe der Tabs ein, die unterhalb der Iconbar angezeigt werden.

Das äußerliche Bild des Plugins gliedert sich in drei Komponenten:

- *ClsesPanel*

Das *ClsesPanel* dient der Navigation durch die Ontologie, wobei die Klassen in Form eines Baumes dargestellt werden. Klassen können innerhalb dieses Panels erstellt, gelöscht oder selektiert werden. Wie von anderen Protégé-Plugins bekannt, wird das *ClsesPanel* auf der linken Seite der Oberfläche angezeigt.

- *TablePanel*

Das *TablePanel* enthält die Tabelle, in der die Instanzen der jeweils ausgewählten Klasse aufgelistet werden. Jede Zeile der Tabelle entspricht einer Instanz, jede Spalte einer Eigenschaft, die die jeweilige Klasse definiert⁷. In den Zellen der Tabelle stehen also die Ausprägungen der Eigenschaften einer einzelnen Instanz. Die Ausprägungen können dabei Literale, Klassen oder Instanzen sein, wobei der jeweilige Typ der Ausprägung durch entsprechende Kennzeichnung⁸ in der Tabellen-Zelle ersichtlich ist.

Die **Tabelle** unterstützt folgende Funktionen (Diese Funktionen sind auch über ein **Kontextmenu**⁹ in der Tabelle verfügbar):

- **Editieren:** Instanzen können editiert werden. Dazu erscheinen die aus dem *InstancesTab* bekannten *Widgets* des *InstanceDisplay*, in dem Eigenschaften der jeweiligen Instanz bearbeitet werden können.
- **Sortierung:** Die Tabelle kann spaltenweise sortiert werden.
- **Rücksetzen der Ansicht:** Die gerade angezeigte Tabelle wird auf die Standardeinstellungen zurückgesetzt. Die Standardeinstellungen beinhalten eine Sicht aller Instanzen (Zeilen) und derjenigen Spalten, in denen Eigenschaften definiert sind.

⁶TabPlugins sind eine Art von Plugins, die Protégé unterstützt (weitere sind Import/Export-Plugins oder Backends). Plugins, deren Schwerpunkt auf der Interaktion mit dem Anwender liegen, werden gewöhnlich als TabPlugins implementiert.

⁷Voraussetzung für die Darstellung ist, dass in der jeweiligen Wissensbasis Instanzen der entsprechenden Klasse definiert sind.

⁸Kennzeichnungen erfolgen über Icons. Das Paket *resource* der Protégé-API stellt entsprechende Icons zur Verfügung.

⁹Gehen Sie dazu mit der Maus über die Tabelle und drücken Sie den rechten Mausknopf.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

- **Link:** Instanzen oder Klassen bieten zusätzlich die Möglichkeit an, direkt in die entsprechende Klasse zu wechseln. Links werden in der Tabelle gesondert dargestellt (*blau Schriftfarbe, unterstrichen*).

Die **Tabellenköpfe** unterstützen folgende Funktionen (diese Funktionen sind über einen Rechtsklick auf den Tabellenkopf verfügbar):

- **Spaltenfilter:** Im Kontextmenu des Tabellenkopfes erscheinen alle verfügbaren Eigenschaften. Diese können mit einem weiteren Klick auf den jeweiligen Spalten/Eigenschaftsname ein- bzw. ausgeblendet werden. Außerdem können mit einem Klick alle Eigenschaften bzw. alle Eigenschaften, in denen definierte Werte vorhanden sind, ausgewählt werden.

- **ToolBar**

Die Toolbar besteht aus einer Reihe von Icons, über die weitere Funktionen verfügbar sind:

- **Back in History, Forward in History:** Die Back/Forward-Buttons stellen eine Klassen-History-Funktion zur Verfügung. Nach dem Start des Plugins mit einem Projekt werden die besuchten (im ClsBrowser ausgewählten) Klassen gespeichert. Man kann vor und zurück springen, wie es aus anderen Applikationen, wie z.B. Web-Browsern, bekannt ist.
- **Filter Table-View:** Über den Filter-Button kann man Spalten der Tabelle ein- bzw. ausblenden. Außerdem können mit einem Klick alle Eigenschaften bzw. alle Eigenschaften, in denen definierte Werte vorhanden sind, ausgewählt werden. Man öffnet das Filter Table-View Panel mit einem Klick auf den entsprechenden Button. Geschlossen wird das Panel durch einen weiteren Klick auf diesen Button.
- **Search Table-View:** Mit einem Klick auf den Search-Button öffnet sich ein Such-Panel, in dem Suchanfrage gestellt werden können. Die Suchanfrage berücksichtigt die aktuell im TablePanel dargestellten Instanzen (und Eigenschaften). Man kann einen Suchstring bzw. einen regulären Ausdruck als Suchausdruck eingeben. Man kann außerdem verschiedene Suchausdrücke mit Hilfe elementarer Operatoren (*UND, ODER*) verknüpfen. Mit einem Klick auf den Search-Button im Search-Panel wird die Suchanfrage abgeschickt. Falls die Suche Treffer in der aktuell dargestellten Tabelle liefert, werden diese angezeigt. Falls keine Treffer gefunden werden, wird der Anwender davon in Kenntnis gesetzt und gebeten, einen weniger einschränkenden Suchausdruck einzugeben. Mit erneuten einem Klick auf den Search Table-View Button in der Toolbar schließt sich das Such-Panel wieder; die Treffer der Suchanfrage stellen weiterhin den aktuellen Inhalt der Tabelle dar.
- **Generate Report:** Durch Auswahl des Report-Button wird ein Report der aktuellen Tabellenansicht erzeugt, welcher auch ausgedruckt werden kann. Um zu garantieren, dass die Spalten auf den Ausdruck passen, wird die Tabelle in eine Liste transformiert. Es besteht nun die Möglichkeit den Preview zu schliessen oder den Report auszudrucken. Weiterhin können über *Page-Setup* weitere, den Ausdruck betreffende, Einstellungen vorgenommen werden.
- **Export Table-View:** Durch Auswahl des Export Table-View Button kann ein Export der aktuellen Tabellenansicht durchgeführt werden. Für alle Formate ausser CSV wird die Tabellenansicht in eine Liste transformiert. Hierbei wird kein Preview angezeigt. Als Formate für den Export stehen zur Verfügung:

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

- * **ASCII-Textdatei**
 - * **CSV** (*Comma Separated Value File*)
 - * **HTML** (*Hypertext Markup Language*)
 - * **PDF** (*Portable Document Format*)
 - * **RTF** (*Rich Text Format*)
 - * **XLS** (*Microsoft Excel Compatible Stylesheet*)
- **Help:** Durch Auswahl des Help-Button kann die programmeigene Hilfe-funktion gestartet werden. Hier werden dem Anwender die Funktionen des Programmes kurz und bündig erklärt.

3. Grundsätzliche Designentscheidungen

(a) Trennung von Model und View

Die Trennung von Daten und Darstellung stellt ein einfaches und effizientes Designprinzip dar. Es ermöglicht eine von den Daten unabhängige Art der Darstellung. Weiterhin können mehrere Nutzer (Clients) die Daten anfragen (Server) und für eigene Zwecke verwenden.

Grundsätzlich arbeitet das *InstanceXL-Plugin* mit den Daten des aktuellen Projektes, insbesondere mit der Wissensbasis (*KnowledgeBase*). Diese Daten werden dem Anwender in erster Hinsicht als Tabelle zur Verfügung gestellt. Die für die Darstellung verwendete und im Swing-Toolkit bereitgestellte Klasse *JTable*¹⁰ selbst unterstützt die Trennung von Model und View, in dem die darzustellenden Daten in einem Modell (*TableModel*) separiert werden. Das Modell hält die Verantwortlichkeit für die Daten, während das *JTable* die Ausgabe und deren Gestaltung übernimmt.

(b) Modularisierung

Um die Klassen übersichtlich zu halten, werden diese soweit wie möglich refaktoriert. Beispielsweise soll das Modell der Tabelle, welches die Daten der Tabelle enthält, so wenig wie möglich Code enthalten, der nicht mit dem im Interface *TableModel* definierten Funktionen verwandt ist. Speziell werden Methoden zum Zugriff auf die Daten der Wissensbasis in die Klasse *DataWrapper* ausgelagert, die die Auswahl der geforderten Datensätze realisiert.

Ein weiteres Beispiel für Modularisierung stellt die Aufteilung der UI-Klassen *ClsesPanel*, *TablePanel* und *Toolbar* dar. Um die Verzahnung dieser Klassen minimal zu halten, werden die Informationen, die die Klassen untereinander austauschen über ein weiteres Objekt der Klasse *State* geleitet. Die Klasse *State* enthält aktuell relevante Daten, hauptsächlich die gerade selektierte bzw. angezeigte Klasse, sowie Referenzen auf instanziierte UI-Objekte, Util-Objekte und Model-Objekte.

(c) Sortierung der Tabelle über Decorator-Entwurfsmuster

Das Sortieren der Spalten der Tabelle erfolgt über den Decorator¹¹ *TableSorter*. Diese Klasse, die von der Klasse *AbstractTableModel* abgeleitet ist, übernimmt die Sortierfunktion oberhalb des *TableModels*. Dadurch wird wiederum das Konzept der Trennung von Model und View eingesetzt, da die Sortierung der Daten lediglich eine Modifikation der Ausgabe, nicht jedoch eine Änderung des Modells erfordert.

Anmerkung (V1.1): Sowohl das Sortieren der einzelnen Spalten, als auch das Ein- bzw. Ausblenden einzelner Spalten wird über Decorator realisiert. Hierzu

¹⁰Die Java-API-Dokumentation stellt ein Tutorial zu Tabellen unter <http://java.sun.com/docs/books/tutorial/uiswing/components/table.html> zur Verfügung.

¹¹Weitere Informationen zu Design-Patterns können Sie unter anderem online unter <http://www.patternspot.com/put/8/JavaPatterns.htm> nachlesen.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Skizze (Abb. 1; *DataModel*, *ColumnChooser* und *TableSorter* sind jeweils von *AbstractTableModel* abgeleitet):

Als oberste Schicht schließt sich die *SearchLayer* an den *TableSorter* an. Somit kann auf den gefilterten (Spaltenfilter) und sortierten Daten gesucht werden. Natürlich ist eine Suche auf ungefilterten und unsortierten Daten möglich, jedoch entsprechen 'ungefiltert' bzw. 'unsortiert' ebenso bestimmten Zuständen des *ColumnChoosers* bzw. *TableSorters*, wie tatsächliche Filter- bzw. Suchanfragen. Dieses Schichtenmodell bietet klare Vorteile, unterliegt jedoch auch gewissen Einschränkungen:

Pros:

- Die Schichten (*ColumnChooser*, *TableSorter*, *SearchLayer*) sind unabhängige Klassen und eignen sich für die **Wiederverwendung** in anderen Programm, die mit *TableModels* arbeiten (jede der Schichtklasse ist eine Unterklasse von *AbstractTableModel*).
- Es existiert eine **klare Trennlinie** zwischen Model und View. Das Model in diesem Fall ist das *DataModel*. Dieses enthält die relevanten Daten, sonst nichts¹². Die Funktionen 'Sortieren' und 'Spaltenauswahl' verändern die Darstellung des Modell, nicht es selbst. Somit gehören die Schichten in den Bereich UI (View) bzw. Util.
- Die Schichten sind, obwohl wir leider keine genauen Profiling-Ergebnisse¹³ angeben können, **schnell und speicherschonend**. Die Schichtklassen besitzen jeweils eine Member-Variable, die eine Referenz auf ein *TableModel* ist. Somit werden die Daten nicht doppelt im Speicher gehalten, sondern *verlinkt*.

Cons:

- Die Schichtstruktur in unserem Programm ist (noch) **hardcoded**, d.h. die Schichten sind stets präsent, auch wenn sie nicht aktiv werden, und die Reihenfolge der Schichten kann nicht geändert werden¹⁴.

Trotz einiger Nachteile hat dieses Schichtenmodell den bisherigen Anforderungen an die Funktionalität standgehalten.

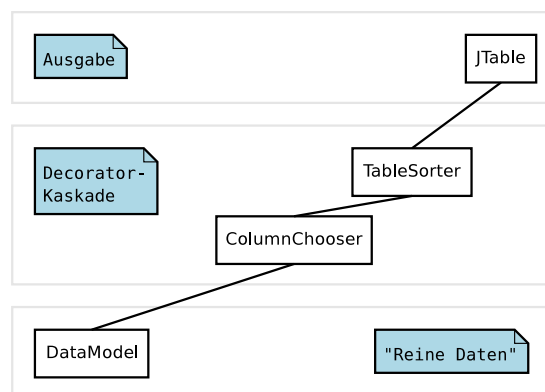


Abbildung 1: Decorator-Kaskade

¹²Fast nichts: Das *DataModel* besitzt wie die Schicht-Klasse die Methoden, die im *TableModel* Interface vorgeschrieben sind. Damit wäre eine direkte Kopplung des *DataModels* an eine Tabelle (*JTable*) möglich.

¹³Das Plugin wurde gelegentlich mit dem *MileStone 6 Profiler* für *Netbeans 4.1* profiliert. Ergebnisse unserer Geschwindigkeitsoptimierung finden Sie unter *Anhang C, Bottlenecks*.

¹⁴Eine möglicherweise nützliche Erweiterung wäre beispielsweise das dynamische Hinzufügen oder Entfernen von Schichten, z.B. dem *SearchLayer*.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(d) LSGM-Datenimport

Das Einlesen des LSGM-Datensatzes ist mittels des **Xerces DOM-Parsers** aus dem Apache Projekt realisiert. Die durch ihn gewonnenen Daten aus den einzelnen XML-Dateien werden einzeln bearbeitet, d.h. es wird kein einzelner grosser DOM-Baum erzeugt. Dadurch wird ein erheblich geringerer Verbrauch an wertvollem Arbeitsspeicher garantiert, und es kann früher mit dem Auswerten der Daten begonnen werden, da bereits nach dem Einlesen der ersten XML-Datei mit dem Umwandeln der Daten begonnen werden kann. Die Transformation der flachen XML-Datenstruktur in eine Ontologie wird mittels der Protégé OWL-API realisiert. Die einzelnen, in der XML-Datenstruktur vorhandenen Datenelemente werden jeweilig gesondert mit einer für sie eigenen Logik behandelt, und mittels dieser die Verknüpfungen innerhalb der entstehenden Ontologie ermöglicht. Der Import ist flexibel in der Hinsicht ausgelegt, dass er jederzeit auf andere ähnliche Datensätze erweitert und angewandt werden könnte.

(e) Activated Help

Auch das Hilfesystem wurde mit großer Sorgfalt implementiert. Ein ansprechendes, durchdachtes, integriertes und leicht bedienbares Hilfesystem steigert die Akzeptanz beim Benutzer, da er sich nicht alleine gelassen fühlt und jederzeit ohne großen Aufwand Nachforschungen zu bestimmten Funktionsbereichen der Applikation durchführen kann. Die Datenhaltung der Hilfestellungen erfolgt mittels einer zentralen, einfach gehaltenen XML-Datei, in der z.B. auch ein Entwickler seine Funktionen dokumentieren und erläutern kann. Diese Niederschrift steht sodann sofort für das integrierte Hilfesystem zur Verfügung, eine nicht zu vernachlässigende Zeitersparnis, welche wesentlich zur Senkung Auslieferungszeit beiträgt.

(f) Tabellenexport mittels JFreeReport

Der Tabellenexport wird mit Hilfe der JFreeReport-Libraries durchgeführt. **JFreeReport** bietet eine funktionale und mächtige API, mittels derer die Transformation einer Tabellenansicht in vielzählige Ausgabeformate umgesetzt werden kann. Die Tabellenansicht wird für den Export in eine Listenansicht gewandelt, um die korrekte Darstellung des Datensatzes in einem üblichen Ausgabeformat wie DIN-A4-Papier zu garantieren. Als Ausgabeformate stehen unter anderem das Portable Document Format (PDF) von Adobe und Hypertext Markup Language (HTML) zur Verfügung. Mittels dieser, weit verbreiteten und akzeptierten Standardformate, ist eine graphisch ansprechende Ausgabe und Austausch der Daten garantiert.

(g) Zentraler Controller

Die Controller-Klasse der Applikation in die Klasse *State* im Paket *Util*. Ein eigenes Package ist für diese einzelne Klasse nicht notwendig gewesen. Die Klasse *State* funktioniert recht archaisch: Alle Objekte, die während der Laufzeit erzeugt werden und für den Ablauf der Applikation relevant sind, registrieren sich im Controller. Benötigt eine Klasse den Zugriff auf eine andere (zum Beispiel benötigt die Klasse *XLExtendedSearchPanel* zur Anzeige der Anzahl der Suchergebnisse die Klasse *SearchLayer*), so erhält die Klasse, die die Anfrage stellt, die Objektreferenz des gewünschten Objektes aus dem *State* und führt darauf die entsprechende Methode aus. Für obiges Beispiel sieht die Codezeile in der Klasse *XLExtendedSearchPanel* wie folgt aus.

```
statusBar.setText(state.getSearchLayer().getSearchInfo());
```

Ein anderer, weniger verwendeter Aspekt des *State* sind tatsächliche Kontrollaufgaben. Die Methode, die für das Setzen der aktuell bearbeiteten Klasse

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(ausgewählt im ClassBrowser oder über einen Link) gehört zu den umfangreichsten in der Klasse State:

```
public void setActualCls(Cls cls) {
    actualCls = cls;
    if (tablePanel != null) { tablePanel.closeAnyPoppedUpPanel(); }
    dataModel.setCls(actualCls);
    classHistory.putNewClass(cls);
    // do we have a link?
    if (!clsPanel.getActualCls().equals(cls)) {
        clsPanel.setSelectedCls(cls);
    }
}
```

In der ersten Zeile wird hierbei die Member-Variable aktualisiert. Die nächste Zeile sorgt dafür, dass keine Search-, Edit- oder Filterpanel sichtbar sind, wenn die Tabellendarstellung der neuen Klasse aufgebaut wird. Anschließend wird im *DataModel* die aktuelle Klasse gesetzt (das Model wird dadurch auch aktualisiert). Danach wird die Klasse der Klassenhistory hinzugefügt. Abschließend wird festgestellt, ob die im ClsBrowser (*clsPanel*) selektierte Klasse der aktuell zu setzenden Klasse entspricht, woraus die Quelle des Requests bestimmen lässt.

Durch dieses absolut zentralisierte Konzept ist es möglich, Objekt unabhängig zu erzeugen und dennoch objektübergreifende Aktionen über wenig Zwischenstellen auszuführen.

4. Grundsätzliche und spezielle Entwurfs- und Strukturprinzipien

(a) Paketstruktur

i. Die Klassen des **InstanceXL**-Plugins sind in folgende Pakete aufgeteilt:

- **main** enthält die Hauptklasse des Plugin, *InstanceXLTab*
- **model**: enthält die Klassen *DataModel*, *DataWrapper*, die die Schnittstelle zwischen der Wissensbasis und den darüberliegenden UI-Klassen bereitstellt
- **ui** enthält die für die Ausgabe verantwortlichen Klassen: *XLAboutFrame*, *XLActivatedHelp*, *XLClusesPanel*, *XLEditPanel*, *XLExtendedSearchPanel*, *XLFilterPanel*, *XLFreeExport*, *XLFreeReport*, *XLModifiedInstanceClusesPanel*, *XLTablePanel*, *XLToolBar*
- **ui.activatedhelp** enthält Klassen für das Einlesen und darstellen der Hilfe: *HelpItem*, *HelpList*, *TopicSelector*
- **ui.animation** enthält Klassen für Animationen: *FlowingAnimation*, *FriskyAnimation*
- **ui.freereport** enthält Klasse für die Report-Generierung/Export: *BackButton*, *ExportDialog*, *FileTypeChooser*, *FreeExport*, *InstanceXLReport*
- **ui.interaction** enthält Komponenten für erweiterte grafische Ausgabe: *XLEmptyClassPane*, *XLLoadingReport*, *XLNoSearchResultsPane*, *XLWelcomePane*
- **ui.pane** enthält Komponenten für erweiterte grafische Ausgabe: *XLInteractionPane*, *XLResultPane*, *XLTextInteractionPane*
- **ui.renderer** enthält objektspezifische Renderer-Klassen, die eine Anpassung der Ausgabe im InstanceTable (z.B. durch geeignete Icons) ermöglichen: *InstanceRenderer*, *ClsRenderer*, *XLRenderUtil*
- **ui.resources.images** enthält Icon-Vorlagen

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

- **ui.resources.report** enthält XML-Report Definition
 - **ui.text** enthält Container-Klasse für graphische Ausgaben: *UltimativeText*
 - **ui.util** enthält UI-Hilfsklassen: *XL SwingUtil, XLWidgetUtil*
 - **util**: enthält die für die Synchronisation des aktuellen Zustandes benötigte Controller Klasse *State*, die Darstellungsmodifikatoren *TableSorter, ColumnChooser, SearchLayer* und *StringLayer*, sowie weitere Hilfsklassen: *ClassHistory, DOMUtil, XLRandomPhrase, XLUtil*
 - **util.resources** enthält die Datei *RandomPhrase*
- ii. Die Klassen des **ImportXL**-Moduls sind in folgende Pakete aufgeteilt:
- **main** enthält die Import-Hauptklasse *ImportXL*
 - **model** enthält die Modell-Klassen *FamilyModel, LSGMMModel*
 - **util** enthält weitere Hilfsklassen: *OnlyXML, QueryUtil, StringUtil*
 - **data** enthält die zu importierenden Daten

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(b) **Klassendiagramm**

Das Klassendiagramm stellt als statisches Modell die Hauptklassen und ihre Beziehungen untereinander dar (Abb. 2).

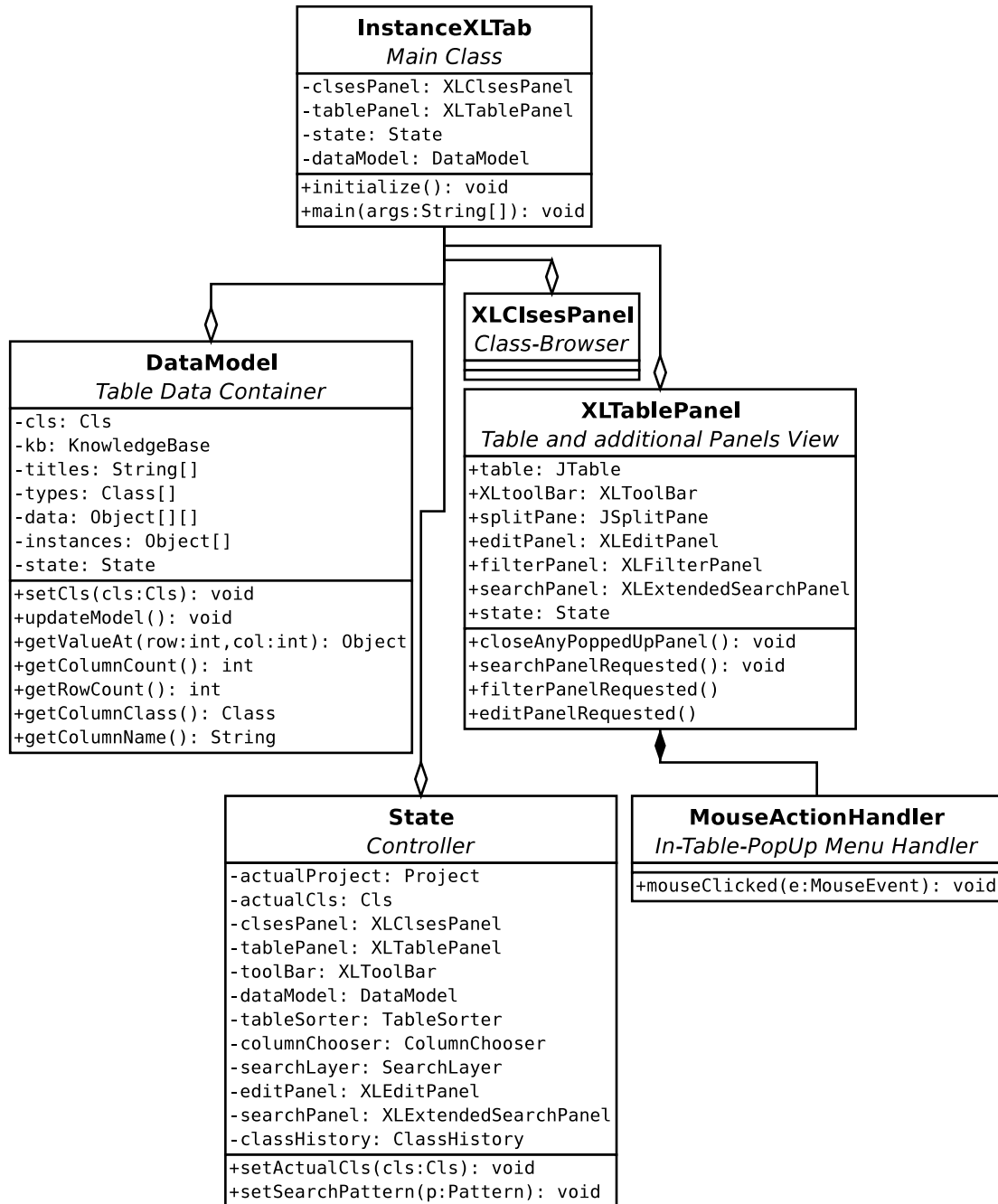


Abbildung 2: Klassendiagramm

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Folgendes Diagramm (Abb. 3) zeigt schematisch die Verteilung der Klassen des *InstanceXL Plugins*:

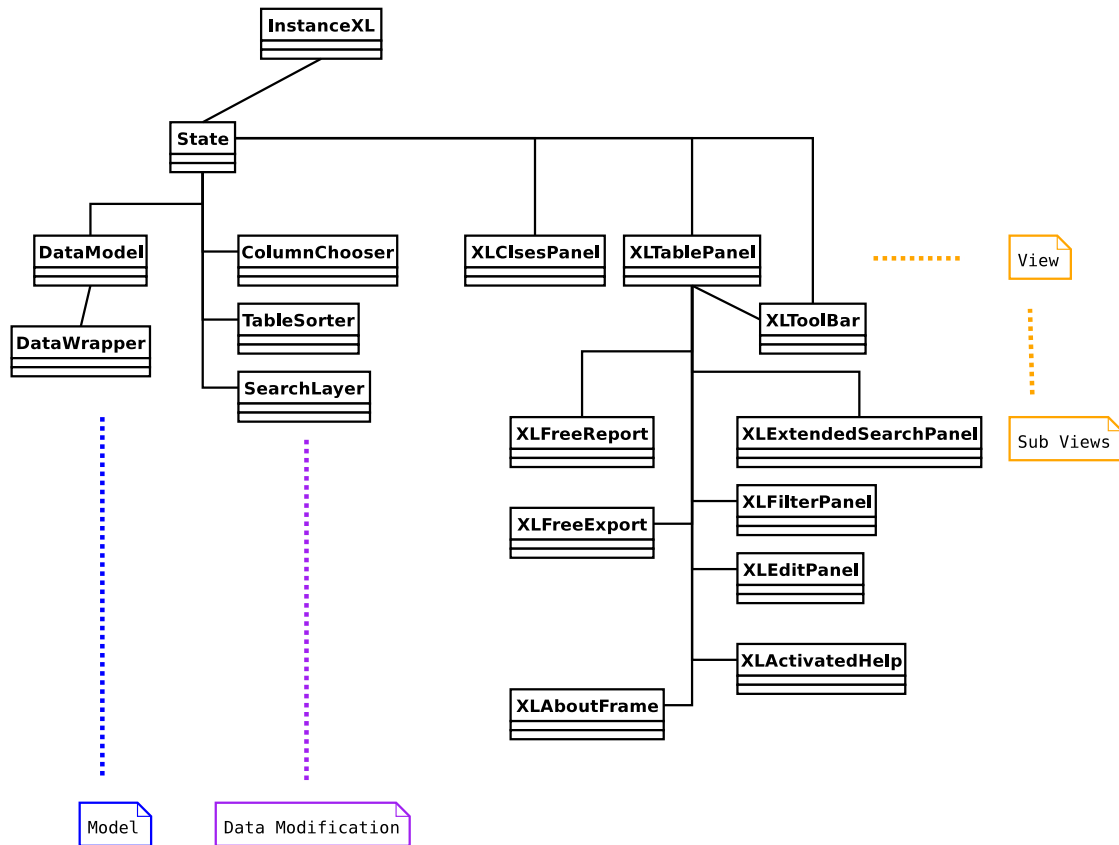


Abbildung 3: Skizze der Klassen

Charakterisierung der einzelnen Klassen:**• Main**

- *InstanceXLTab*: Dies ist die Hauptklasse des Plugins. Sie wird von *AbstractTabWidget* abgeleitet und stellt die Schnittstelle zur Protégé-Rahmenapplikation dar. Über sie ist es möglich auf das aktuelle Projekt und somit auf die aktuelle Wissensbasis zuzugreifen.

• Model

- *DataModel*: Diese Klasse dient als Modell für die Tabelle. Sie hält die aktuell relevanten Daten als Attribute und stellt diese bei Bedarf der Tabelle zur Verfügung.
- *DataWrapper*: Diese Klasse übernimmt die Verbindung von Tabellenmodell (Plugin) und Wissensbasis (Projekt). Dabei realisiert sie die Abstraktion und Transformation von der konkreten Klasse der Wissensbasis¹⁵, und die Vorformatierung der Daten entsprechend der Implementation des Tabellenmodells.

¹⁵Für Protégé-Projekte wird die Klasse *KnowledgeBase* genutzt; für OWL-Dateien die Klasse *JenaOWLKnowledgeBase*.

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

• Data-Modification

- *TableSorter*: Diese Klasse dient als Decorator für die Tabelle. Sie übernimmt die Sortierung und sorgt für die korrekte Weiterleitung von Events in der Tabelle an das Tabellenmodell.
- *ColumnChooser*: Diese Klasse dient als Decorator für die Tabelle. Sie übernimmt das Ein- bzw. Ausblenden von Tabellenspalten.
- *SearchLayer*: Diese Klasse dient zum Durchsuchen der Daten.

• View

- *XLClsesPanel*: Abgeleitet von `ClsesPanel`. Das `ClsesPanel` ist eine Klasse der Protégé-API¹⁶, die die Navigation durch die Wissensbasis ermöglicht.
- *XLTablePanel*: Das `TablePanel` dient als Container für die Instanzen-Tabelle.

• Subviews

- *XLExtendedSearchPanel*: Im `SearchPanel` können über verschiedene Textfelder Suchbegriffe oder reguläre Ausdrücke eingegeben werden. Diese können als Suchanfrage über die Tabelle abgeschickt werden. Das *XLExtendedSearchPanel* zeigt außerdem die Anzahl der gefundenen Treffer an.
- *XLFilterPanel*: Diese Klasse stellt ein Panel für die Spaltenauswahl zur Verfügung.
- *XLEditPanel*: Diese Klasse stellt ein Panel für das Editieren einer Property zu Verfügung. Die für die jeweilige Property verantwortliche Widgets stammen aus Protégé bzw. OWL. Zum Teil sind für bestimmte Properties keine Edit-Widget verfügbar.
- *XLAboutFrame*: Informationen zu den Autoren.
- *XLFreeReport*: Die Klasse unterstützt die den Export und den Export-Preview.
- *XLFreeExport*: Diese Klasse stellt ein graphisches Menu zur Export-Format Auswahl zur Verfügung.
- *XLActivatedHelp*: Hilfe und Erklärungen zu den Funktionen von `InstanceXL`.

• Controller

- *State*: Die Klasse `State` ist eine Controller-Klasse um die Informationen über den aktuellen Zustand des Plugin zu speichern und diese an alle interessierten Objekte zu verteilen.

Weitere Informationen zu den einzelnen Klasse finden Sie in der mit **javadoc** erstellten Quellcodedokumentation. Außerdem sind wichtige Abschnitte einiger Klassen *inline* dokumentiert.

¹⁶Vollständiger Name: `edu.stanford.smi.protege.ui.ClsesPanel`

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(c) **Aktivitätsdiagramm**

Folgendes Diagramm (Abb. 4) zeigt eine Grobübersicht über die Aktivitäten des InstanceXL-Plugins:

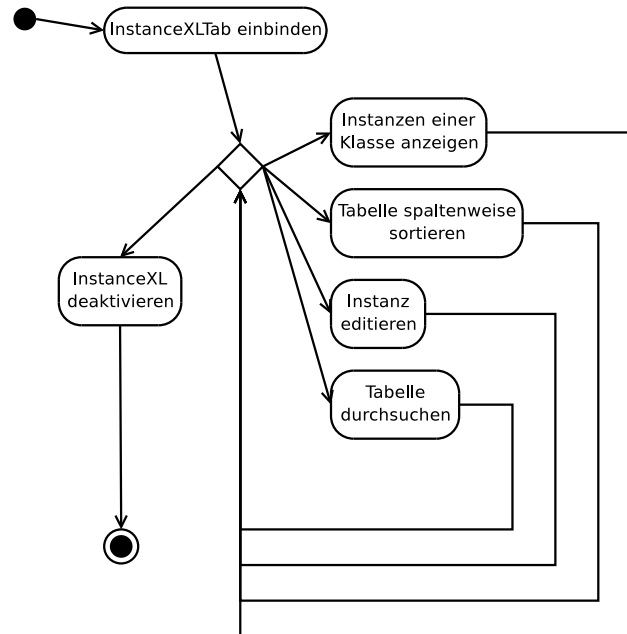


Abbildung 4: Aktivitäten, Übersicht

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Folgendes Diagramm (Abb. 5) zeigt eine Detailansicht der möglichen Aktivitäten während Arbeit mit dem InstanceXL-Plugin:

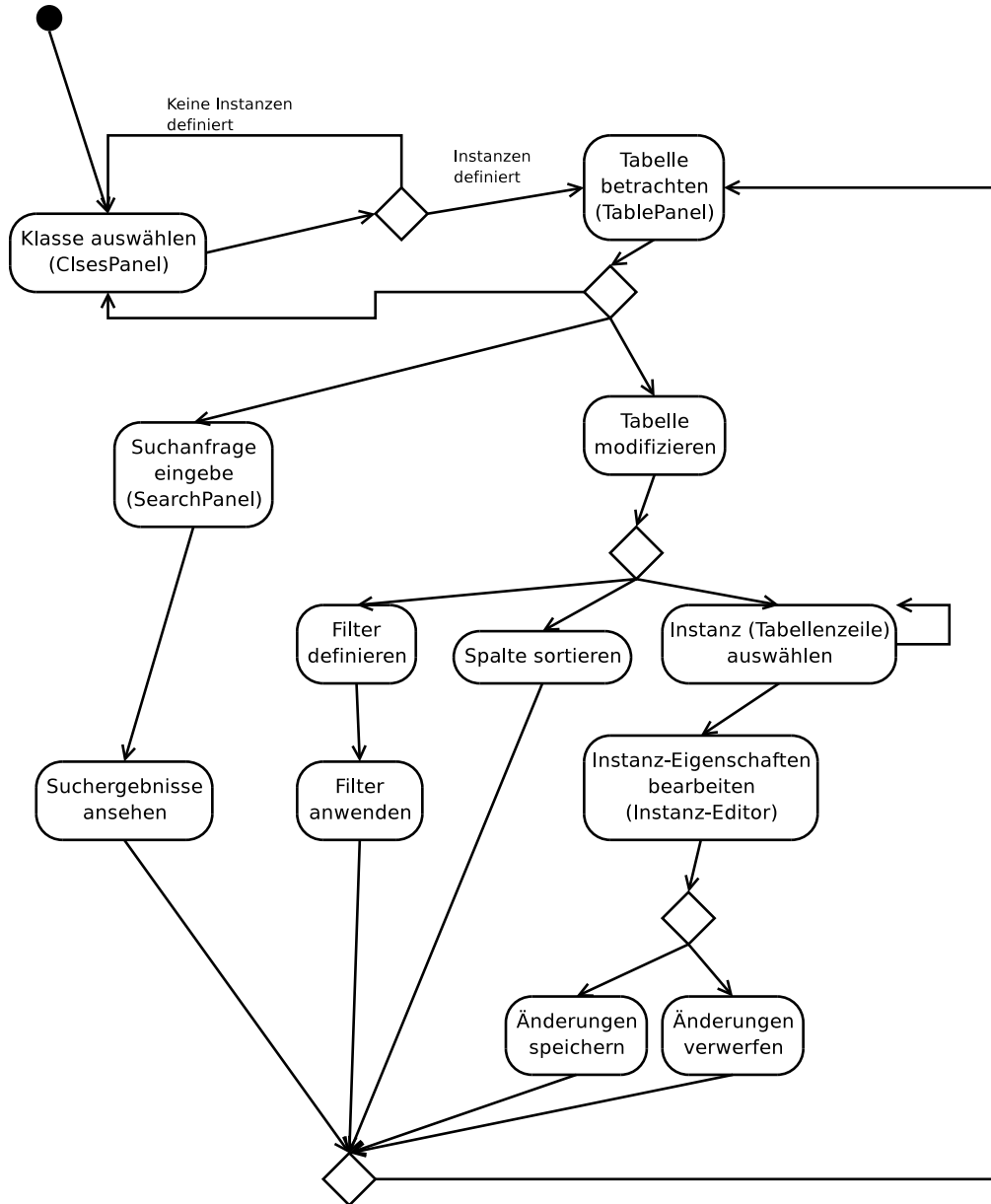


Abbildung 5: Aktivitäten, Details

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

(d) **Zustandsautomat**

Die Skizze des Zustandsautomates (Abb. 6) zeigt stark vereinfacht die Abfolge der Zustände, in denen sich das InstanceXL-Plugin befinden kann.

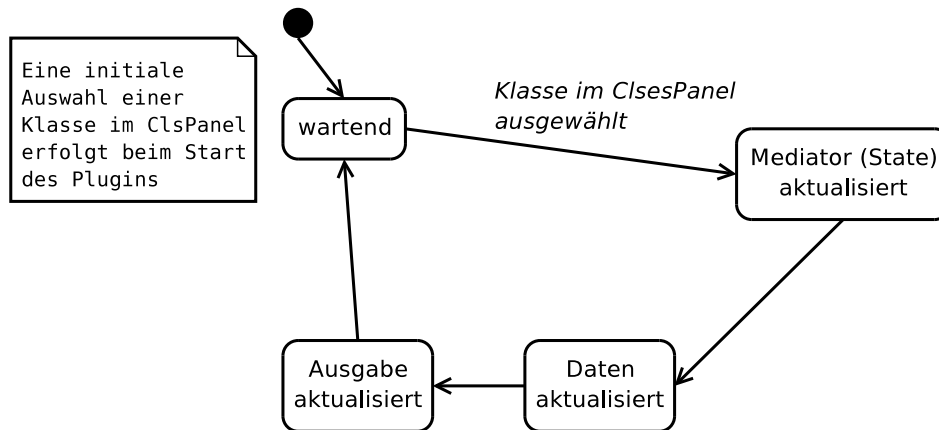


Abbildung 6: Zustandsautomat, Aktualisierung nach Auswahl einer Klasse

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Anhang A (seit Version 1.2):**MVC, Kommunikationsmodell****1. Klassen und Flüsse**

Dieser Abschnitt stellt und beantwortet die Frage: "Welche Klasse kommuniziert was über welche Kanäle?". Diese Frage ist in Bezug auf Erweiterung des Projektes wichtig, da gewisse Klassentypen (UI, Modell, Util-Klassen) in einer einheitlichen Form den Datenaustausch innerhalb des Programms realisieren sollen.

Die Frage dient somit gleichzeitig der Fixierung bewährter Implementierungen, sowie als Ausgangspunkt, um bessere und klarere Designideen umzusetzen.

(a) Woher kommen die Ereignisse?

Zunächst entscheidend ist die Auswahl einer Klasse. Die aktuelle ausgewählte Klasse wird sowohl vom Modell benötigt, als auch vom View bearbeitet. Das Plugin ist insgesamt klassenzentriert organisiert.

In der Übersicht sieht die Kommunikation wie folgt (Abb. 7) aus:

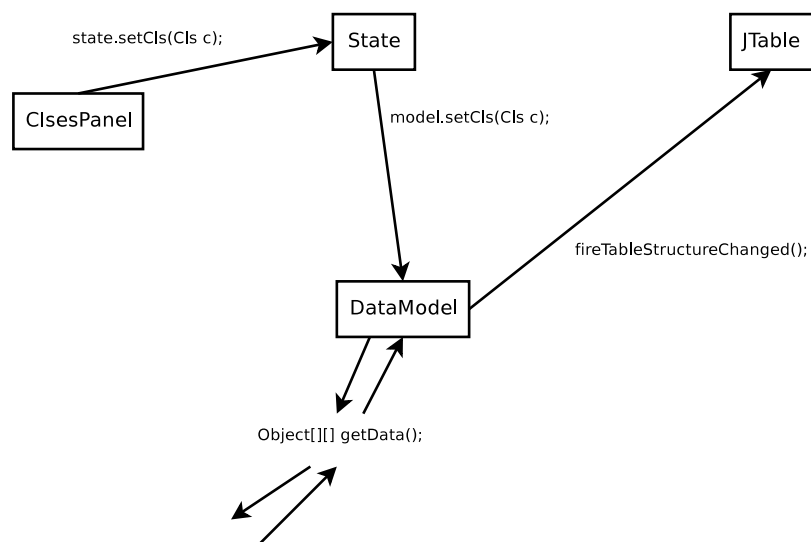


Abbildung 7: ClsesPanel löst Ereignis (Klasse ausgewählt) aus, Folgeereignisse

Abbildung 1 zeigt einen aktiven Eingriff in den Controller. Ein Beispiel für einen passiven (nur lesenden) Zugriff ist das Ereignis: "Exportiere Tabellendaten nach PDF" (siehe Abb. 8).

(b) Aufbau des Controller

Der Controller muß als vermittelnde Instanz die Ablauf-Fäden, Ereignisse und Folgeereignisse steuern. Solange es sich um eine überschaubare Anzahl von Klassen und Informationen handelt, die übertragen werden müssen, läßt sich der Controller auch sehr gut implementieren. Mit dem Hinzufügen weiterer Features zu dem Programm ergeben sich jedoch einige Probleme:

- Der Controller muß auf jedes Objekt, das er kontrolliert, eine Referenz halten (gleichzeitig sollte die Anzahl der Referenzen möglichst klein gehalten werden).

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

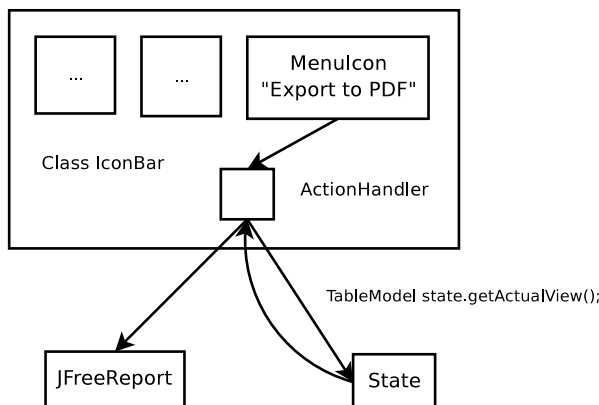


Abbildung 8: Nur-Lese-Zugriff auf den Controller (State)

- Die in InstanceXL verwendete Decorator-Kaskade stellt potentiell eine Gefahr für die saubere Implementation des Controlles dar, aus folgenden Gründen: Im besten Fall hält der Controller *eine* Referenz auf das Modell (Data-Model) und jeweils eine weitere für jede Top-Level-UI Klasse (TablePanel, ClsesPanel). Wird nun z.B. ein Filter in einer UI-Klasse definiert und aktiviert, so erreicht diese Information den Controller. Eine Änderung des Filter betrifft aber weder das Modell (die Daten selbst bleiben durch den Filter unverändert), noch direkt eine Top-Level-UI Klasse, wie z.B. das TablePanel. Der Filter betrifft ein Objekt *auf dem Weg zwischen* Modell und View.
- Es ist von entscheidender Bedeutung, die Darstellung vom Modell zu trennen, um andere Ausgabewege (z.B. über eine Webfrontend, Servlet) nahtlos zu unterstützen.

(c) **Übersicht/Nachrichtenaustausch**

Eine grobe Skizze des angedachten Nachrichtenaustausches zeigt Abb. 9.

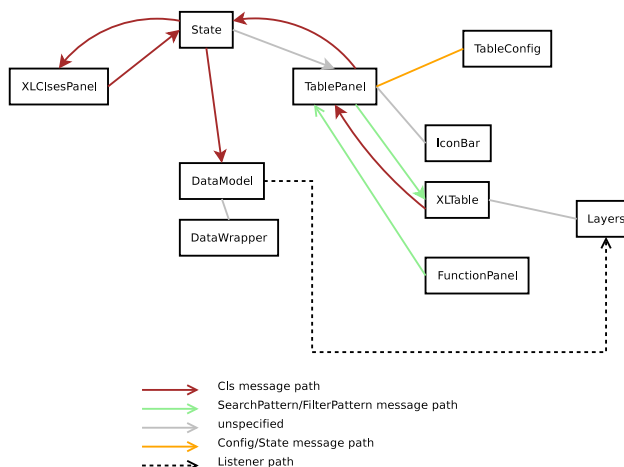


Abbildung 9: Nachrichtenaustausch

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Anhang B (seit Version 1.4):**1. Bemerkung**

Release **instanceXL 0-4-0** erweitert die Vorgängerversionen um eine erweiterte Suchfunktion, sowie um Hilfsfunktionen. Da die Implementierungsphase noch nicht abgeschlossen ist und aktuell eine rege Bearbeitung des Quellcodes stattfindet, enthält dieser Anhang lediglich einige *Ausblicke* (bzw. *Rückblicke*).

2. Planung der Darstellungspersistenz (immer noch Ausblick)

Die Darstellungspersistenz beinhaltet eine History-Funktion, wie sie aus Web-Browsern und anderen Applikationen bekannt sind. Diese sollen im Fall des Tab-Plugins folgende vom Benutzer veränderte Einstellungen memorisieren:

- Ausgewählte Klasse (im ClassBrowser)
- Ausgewählte Spalten einer Tabelle

Auf diese Weise soll dem Anwender ermöglicht werden, schnell zwischen verschiedenen bereits durchlaufenen Ansichten zu wechseln.

Für die Aufbewahrung der Daten der einzelnen Schritte bietet sich die Klasse Properties der Java-API an, die eine direkt Speicherung der gespeicherten Eigenschaften im XML-Format erlaubt.

3. Suchfunktion (Rückblick)

Die Suchfunktion bietet die Möglichkeit eine Tabelle nach Suchbegriffen zu durchsuchen. Dabei können Suchmuster mit Hilfe logischer Verknüpfungen (und, oder) formuliert werden.

Grober Ablauf der Suchfunktion:

- Eingabe der Suchbegriffe und der Verknüpfungen durch den Anwender
- Parsen der Eingaben
- Erstellung eines compilierbaren regulären Ausdrucks aus den tokens.
- Übergabe des Suchmusters an den Controller
- Weiterleiten des Suchmusters an die verantwortliche Darstellungsschicht (Search-Layer)
- Durchsuchen der aktuelle Tabellensicht nach dem Suchmuster
- Aktualisierung der Ausgabe

4. Hilfsfunktionen (Rückblick)

Die Hilfsfunktion dient der Erläuterung der Funktionen der Applikation. Das Hilfe-Konzept des Tab-Plugins sieht darüber hinaus eine ansprechende Begleitung des Anwenders durch das Programm vor. Dabei wurde und wird auch darauf geachtet, die entsprechenden Klassen soweit wiederverwendbar wie möglich zu gestalten.

Sensible Ereignisse:

- Start-Up (Welcome-Screen)
- Expliziter Aufruf der Hilfe bzw. der Programm-Dokumentation
- Auswahl einer Klasse im ClassBrowser, die keine Instanzen enthält
- Suchfunktion liefert keine Treffer

Softwaretechnik-Praktikum SS 2005

GR-1

Projektleiter: Adrian Kiess | Dokument erstellt von Adrian Kiess, Martin Czygan

Anhang C, Bottlenecks**1. Vorbemerkung**

Leider konnten wir die Anwendung nicht so intensiv profilieren, wie es in unserer Projektplanung angedacht gewesen ist.

2. Bottleneck #1 (Aufbau und Rendern der Tabelle)

Der Aufbau der Tabelle ist solange kein Problem, solange keine aufwendigen Renderer auf den Spalten definiert sind. Um jedoch Instanzen/Individuals oder Klassen-Objekte gesondert zu kennzeichnen, werden in den Tabellenzellen Icons benötigt, die sinnvoll nur über das Überschreiben der Methode `getTableCellComponent` realisiert werden können. Weiterhin sollten Properties¹⁷, die mehrere Slot-Values definieren, ebenfalls in der Tabelle dargestellt werden; und zwar in *einer* Tabellezelle. Dies ist nun eine tatsächliche Aufgabe für den Renderer. Folgende Codezeile kann - da sie bei jeder Veränderung der Tabelle aufgerufen wird - die CPU¹⁸ durchaus zu 30%-40% auslasten:

```
table.setRowHeight(row, p.getPreferredSize().height);
```

Mit dieser Anweisung wird die aktuelle Höhe der Tabellenzeile gesetzt, was notwendig ist, falls mehrere Instanzen in einer Property definiert sind und untereinander angezeigt werden müssen.

Folgende Modifikation löst das Problem:

```
final int height_wanted = p.getPreferredSize().height;
if (height_wanted != table.getRowHeight(row))
    table.setRowHeight(row, p.getPreferredSize().height);
```

¹⁷Die Begriffe Slot und Property werden hier synonym gebraucht.

¹⁸Getestet auf einer *Intel Centrino Plattform, Banias CPU 1.3Ghz*