Entwurfsbeschreibung OntoTable

1 Allgemeines

Das Plugin *OntoTable* für Protégé 3.1 ensteht im Rahmen des Softwaretechnikpraktikums 2005 an der Universität Leipzig.
Die Firma SoftConsult möchte die Möglichkeit haben, Ihre im XML-Format vorliegenden Daten in die OWL-Knowledgebase zu importieren. Das Plugin *OntoTable* soll datenbankähnliche Funktionen wie Suchen und Filtern bereitstellen. Der Schwerpunkt liegt auf der leicht verständlichen und übersichtlichen Darstellung dieser Daten in Tabellenform.

1.1 Installation

Um das Plugin *OntoTable* zu installieren, muss ein Ordner „OntoTable“ im Plugins-Ordner der Protege Installation erstellt werden, in die dann die *OntoTable*-jar-Datei hineinkopiert wird.
Nach dem Start von Protege 3.1 kann das Tab über die Checkbox „OntoTable“ im Menü *Project* → *configure* aktiviert werden. Nach erfolgreicher Installation sollte das *OntoTable*-Tab erscheinen.

2 Produktübersicht

2.1 Graphische Oberfläche

Die Oberfläche von OntoTable entspricht dem Look&Feel von Protégé 3.1 bzw OWL 2.0. Der Benutzer sieht beim Öffnen des Tabs drei Frames:

2.1.1 Class Browser

Links oben im Tab zeigt der aus Protégé und OWL bekannte Class Browser die Klassenhierarchie und die Anzahl der zu jeder Klasse gehörenden Individuals.

2.1.2 Table Tree

Links unten im Tab werden die bereits erstellten Tabellen in einer Baumstruktur dargestellt, wobei jede Tabelle zu der Klasse deren Individuals sie darstellt als Kind zugeordnet wird. Dementsprechend werden auch die Suchergebnisse, der Suche über einer Tabelle, im Tree als Kind der entsprechenden Tabelle dargestellt.

2.1.3 Table View

Die komplette rechte Seite des Tabs bietet den Rahmen für die Darstellung der aktuell selektierten Tabelle, also der Individuals einer Klasse. Außerdem sind hier alle Grundfunktionen des Plugins verfügbar.

2.2 Grundfunktionen

Das Plugin bietet dem Nutzer eine Reihe von Funktionen zur strukturierten Darstellung von Individuals einer Klasse.

2.2.1 Sortieren

Durch einen Klick auf die Kopfzeile der Tabelle wird die angeklickte Spalte lexikalisch auf- oder absteigend sortiert. Die Sortierreihenfolge (auf- bzw absteigend) kann durch wiederholten Klick auf die Kopfzeile geändert werden. Bei Properties mit mehreren Werten wird stets der kleinste Wert zur Sortierung herangezogen.

2.2.2 Stichwortsuche

Durch einen Klick auf den Suchbutton im Table View öffnet sich ein Dialog in welchem das gesuchte Stichwort abgefragt wird. Die Tabelle mit den Individuals in denen das Wort auftaucht wird im Table View angezeigt. Außerdem erscheint diese Tabelle als Eintrag im Table Tree.

2.2.3 Tabelle drucken

Durch Betätigung des Tabelle-drucken-Buttons öffnet sich ein Druckdialog mit Druckvorschau in dem alle nötigen Einstellungen gemacht werden und das Drucken gestartet wird. Die Tabelle wird auf dem Drucker ausgegeben.

2.2.4 Spalten ein-/ausblenden

Durch Betätigung des Spalten-ein-/ausblenden-Buttons öffnet sich ein Dialog in dem durch Checkboxen die Spalten, welche angezeigt werden sollen vom Nutzer ausgewählt werden können. Die Tabelle wird entsprechend der Einstellungen im Table View angezeigt.

2.2.5 Tabelle exportieren

Durch Betätigung des Exportieren-Buttons kann in einem Export-Dialog die Datei, in die geschrieben werden soll ausgewählt werden. Die aktuelle Tabelle wird dann als .txt Datei im CSV-Format abgespeichert.

2.2.6 Daten importieren

Durch Klick auf den Importieren-Button öffnet sich ein Dialog. Der Benutzer muss eine OWL-Datei und einen Ordner mit XML-Dateien auswählen.

Die OWL Datei wird zu Beginn des Imports geöffnet und bildet den Behälter für eine Anzahl von Individuals. Es ist wichtig, dass diese OWL-Datei einige Klassen definiert (Author, Book, Article ..., Klassen des LIT-Datensatzes). Wenn die OWL-Datei diese Klassen nicht definiert, wird auch nicht importiert!!

Die benötigte OWL-Datei wurde von Prof. Gräbe erstellt und als Schablone vorgegeben. Für die speziellen Zwecke des LIT-Imports wurde sie noch etwas manipuliert. Die Datei LIT.owl im Verzeichnis LIT→XML-Data des mithochgeladenen LIT-Ordners wird zum Import empfohlen.

Der auszuwählende XML-Ordner muss selbst, oder in seinen Unterordnern, eine oder mehrere XML-Dateien enthalten. Wenn der Weiter-Knopf gedrückt wird, werden aus diesem Ordner alle Dateien ausgewählt, die folgenden Bedingungen genügen:

1. Die Dateieindung ist „xml“
2. Das Wurzelelement der Datei heisst „LIT“

Diese Dateien werden in eine Liste geschrieben, die während des Imports abgearbeitet wird. Den Vorgang können Sie mittels einer Progress-Bar verfolgen, in der der Installationsverlauf in Prozent angegeben wird, und der Dateiname der aktuell importierten Datei angezeigt wird. Dieses Progress-Fenster erscheint, wenn der Weiter-Button gedrückt wurde.

Wenn das Projekt importiert ist, fragt Protégé, ob Sie das vorher geöffnete Projekt speichern wollen, da nun das neue, importierte Projekt geöffnet wird. Nach dieser Abfrage wird das neue, importierte Projekt geöffnet.

2.2.7 Filtern

Durch Klick auf den Filter-Button öffnet sich ein Filter-Dialog, in welchem nach Belieben verschiedene Filterkriterien mit 'und' oder 'oder' Verknüpfung hinzugefügt, sowie entfernt werden können. Die gefilterte Tabelle erscheint im Table View.

2.2.8 Autofiltern

In der Kopfzeile jeder Spalte befindet sich ein Button, bei dessen Betätigung sich ein Drop-Down-Menü öffnet. In diesem Menü befinden sich alle verschiedenen Einträge der entsprechenden Spalte. Durch Anwahl eines Eintrags werden alle Zeilen, die nicht diesen Eintrag in dieser Spalte haben, ausgeblendet.

3 Grundsätzliche Struktur- und Entwurfsprinzipien für das Gesamtsystem

Um ein TabPlugin für Protégé 3.1 zu realisieren, muss von der Klasse AbstractTabWidget geerbt werden. So erhält man Zugriff auf die Projekte, sowie auf die KnowledgeBase. Auf dieser Basis kann dann innerhalb des Tabs ein eigenes Userinterface und eigene Funktionalitäten zur Darstellung und Manipulation der in Protégé vorliegenden Daten entwickelt werden.

3.1 View

3.1.1 Class Browser

Der Class Browser (*OntoClsPanel*) ist ein aus Protégé bekanntes Frame, welches die Klassen hierarchisch und mit der Anzahl der zu einer Klasse existierenden Individuals, in Klammern hinter dem Klassennamen, darstellt. Einzig die Action Listener werden überschrieben, um die gewünschte Reaktion, nämlich die Darstellung einer Tabelle mit den Individuals, zu ermöglichen.

3.1.2 Table Tree

Der Table Tree (*OntoTableTreePanel*) erbt von einem JPanel. Die Hauptkomponente ist ein JTree in dem die vom Benutzer erstellten Tabellen aufgelistet werden. Die Information über die Tabellen stammt vom *OntoTableModelManager*, welcher ein Aggregat des *OntoTableTreePanel*s darstellt.

3.1.3 Table View

Im Table View (*OntoTablePanel*), welches ebenfalls vom JPanel erbt, befinden sich die Funktionsbuttons, über die die verschiedenen Dialoge (Aggregate des *OntoTablePanels*) zur Ausführung der Funktionen des Plugins erreicht werden können. Hauptsächlich dient der Table View der Darstellung der aktuellen Tabelle (*OntoTable*).

3.1.4 Tabellendarstellung

Die Tabellendarstellung (*OntoTable*) ist eine Spezialisierung eines JTables und dient der Visualisierung der Tabellenmodelle. Sie ist ein Aggregat des *OntoTablePanel*s. In dieser Klasse wird zu Beginn festgestellt wie breit und wie hoch die Zellen der Tabelle sein müssen, hierzu werden die Inhalte Spaltenweise überprüft.

3.2 Modelseite der Tabelle

3.2.1 *OntoTableModelManager*

Der *OntoTableModelManager* stellt den Controller der Tabellenmodelle dar. Er verwaltet den Zugriff auf die aktuell angezeigte Tabelle und strukturiert die Hierarchie der Tabellenmodelle untereinander. Ausserdem hält er permanent ein *OntoTableModel* für die temporäre Darstellung einer Tabelle (Vorschaufunktion).

3.2.2 *OntoDataModel*

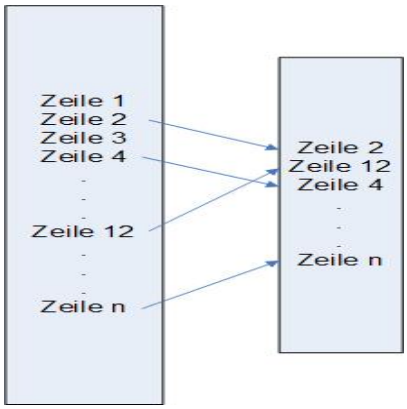

Abbildung 1: Abbildung vom Datenmodell in die darzustellende Tabelle mittels Zuordnungsarrays

Das *OntoDataModel* schreibt die Daten, welche durch die Hilfsklasse *OntoOWLTool* aus der KnowledgeBase organisiert werden, in Individual- und Property- Arraylisten . Das *OntoDataModel* kann bereits als Tabelle dargestellt werden, die anderen Tabellenmodelle werden ausschliesslich für die weitere Aufbereitung, respektive Umsetzung der Funktionen, benötigt.

3.2.3 *OntoFilterModel*

Das *OntoFilterModel* bekommt die Ergebnisse der Filter als Boolean-Arraylisten übergeben und erstellt daraus eine Zuordnung der Zeilen aus dem *OntoDataModel* auf ein

internes Array, sodass bei jeder Anfrage nur valide Zeilen weitergegeben werden, invalide hingegen nicht weitergegeben werden.

3.2.4 *OntoSortModel*

Das *OntoSortModel* erstellt mit Hilfe von Arraylisten eine Abbildung der Zeilenpositionen aus dem *OntoFilterModel* auf die neuen, sortierten Zeilenpositionen.

3.2.5 *OntoTableModel*

Im *OntoTableModel* wird das Spalten ein- und ausblenden realisiert. Dies erfolgt analog zum *OntoFilterModel*, wobei hier die sichtbaren Spalten weitergereicht werden, die nicht sichtbaren nicht.

3.3 Import / Export

3.3.1 LIB Import

Die *LIBImport*-Klasse implementiert die ImportPlugin-Schnittstelle von Protégé. Sie organisiert den Import der XML-Dateien. Voraussetzung für einen Import ist, dass eine OWL-Datei existiert, die die spezielle Struktur eines LIB-Datensatzes besitzt, und dass eine Menge von XML-Dateien in einem Ordner und seinen Unterordnern liegen, deren Root-Element den Namen LIB trägt und die einen BibItem-Typ (zB. Article, Proceeding. . .) besitzen.

Der Import öffnet die OWL-Datei als Protege-Projekt. Jede XML-Datei aus dem gewählten Ordner repräsentiert mehrere Individuals, zB: Article, oder Book, oder Proceeding und Author, die in das geöffnete Projekt hinzugefügt werden. Wenn jede XML-Datei bearbeitet ist, wird das so entstandene Projekt an Protégé zurückgegeben.

3.3.2 CVS-Export

Inhalte der Zellen einer Tabelle werden in das CSV-Format überführt und mit Hilfe eines FileWriters in eine im Dialog vom Benutzer gewählte Datei geschrieben. Als Trennzeichen werden Kommata verwendet.

3.4 Filtern

Das *OntoFilterModel* ruft die in ihm enthaltenen *Filter* auf, welche wiederum die in dem jeweiligen Filter enthaltenen *FilterRule*s aufruft und diese prüfen nach deren Kriterien, Operatoren und Werten, ob ein Individual gefiltert wird oder nicht. Da OWL verschiedenste Datentypen zur Verfügung stellt, welche auf String, Integer, Float, Boolean, OneOf (Selektion aus einer Vorgegebenen Auswahl) oder Object Properties reduziert werden können, implementieren dementsprechend 6 FilterRules das *FilterRule* Interface. Dadurch wird zum einen die Erweiterbarkeit auf eventuell neue Datentypen gewährleistet, zum anderen ein relativ einheitliches Filtern, auch wenn sich die einzelnen Datentypen teilweise gravierend unterscheiden.

Im *FilterDialog* wird der Datentyp des gewählten Properties ermittelt und dementsprechend die zum Datentyp passenden Operatoren bereitgestellt und eine passende Werteingabe gefordert.

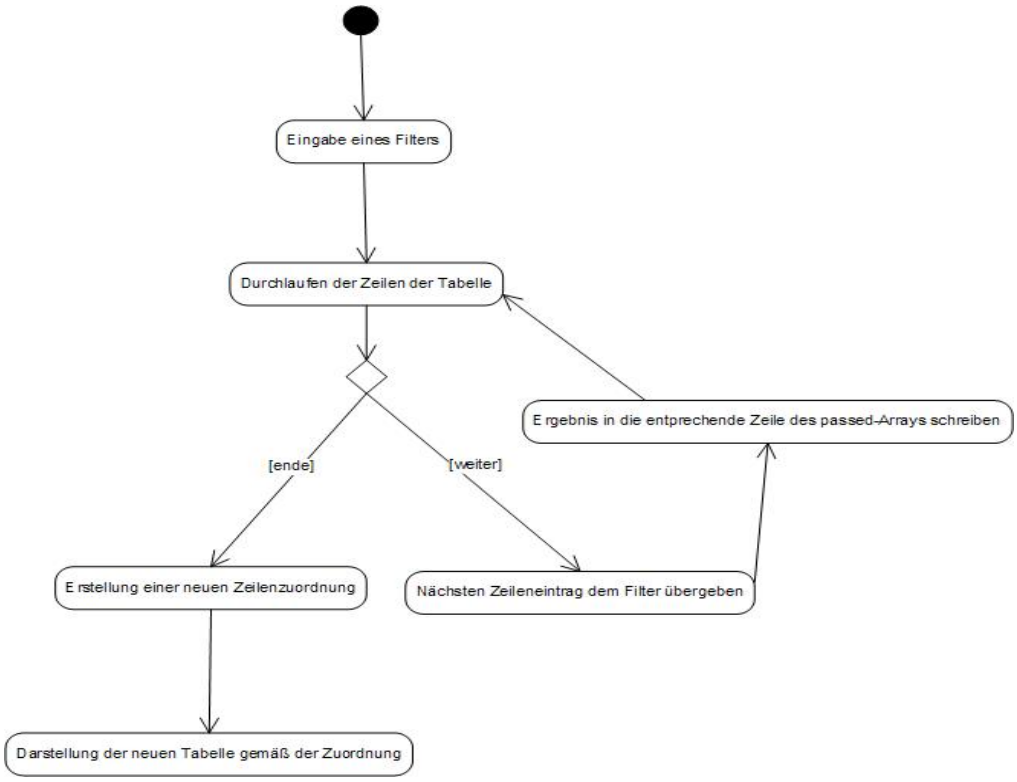

Abbildung 2: Aktionsdiagramm eines Filtervorgangs

3.4.1 Autofilter

Im *OntoTableModel* wird in der Methode makeHeader() eine Autofilter-ArrayListe angelegt, in welche das Icon und die entsprechenden Werte der Spalte geschrieben werden.

Im *OntoTable* wird mit Hilfe der privaten inneren Klassen MyHeaderRenderer und MyComboBoxRenderer eine in der ersten Spalte der Tabelle lokalisierte ComboBox erstellt. Wird diese ComboBox geöffnet und ein Tabelleneintrag ausgewählt, so wird mit diesem eine AutofilterRule erstellt. Criteria der FilterRule ist die entsprechende Spalte, der Operator ist in einer AutfilterRule immer == und als Value wird der entsprechende Tabelleneintrag übernommen. Mit dieser FilterRule wird wie oben beschrieben gefiltert.

3.4.2 Suchen

Das Suchen stellt eine Sonderform des Filterns dar, bei der innerhalb der Tabelle nach einem bestimmten String gesucht wird, also nach einem Individual, welches in einer der Spalten diesen Eintrag enthält gefiltert. Aus diesem Grund wird das Suchen über das Filtersystem realisiert, mit Hilfe der *SearchFilterRule*. Diese ist nicht auf einen bestimmten Datentypen festgelegt, sondern vergleicht nur Strings (alle Datentypen werden mittels toString() vereinfacht). Allerdings gewährleistet dieses System eine gute Erweiterbarkeit auf beliebige Datentypen bei der Suche.

3.5 Drucken

Soll eine Tabelle gedruckt werden und der Druck-Button wurde betätigt, erscheint ein Druckdialog, welcher eine Druckvorschau zeigt. Das Drucken wird durch die Klasse *Onto-*

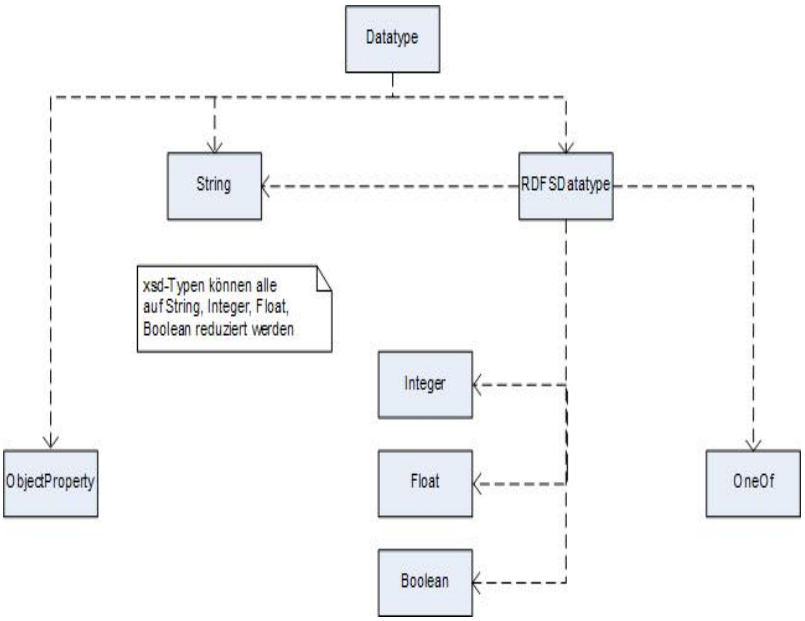

Abbildung 3: Zusammenhang der Datentypen

TablePrintable realisiert. Hier wird die Tabelle, falls sie für eine Seite zu groß ist entsprechend dem gewünschten Papierformat aufgeteilt und die Spalten und Zeilengröße wird festgelegt. Außerdem wird, falls ein Autofilter angezeigt wird, die erste Zeile nicht mitgedruckt und alle Seiten werden mit Seitennummern versehen.

4 Grundsätzliche Struktur- und Entwurfsprinzipien der einzelnen Pakete

4.1 ui

Das Package ui enthält alle Klassen zur Gestaltung der Oberfläche, sowie die Hauptklasse *OntoTab*, welche das Zentrum des Plugins darstellt und als im View gekapselter Controller verstanden werden kann.

4.2 ui.dialog

Die in ui.dialog enthaltenen Klassen sind ebenfalls GUI-Klassen, welche durch Betätigung von Buttons aufgerufen werden. Sie fordern Benutzereingaben um Aufgaben wie Spalten ein- und ausblenden zu realisieren.

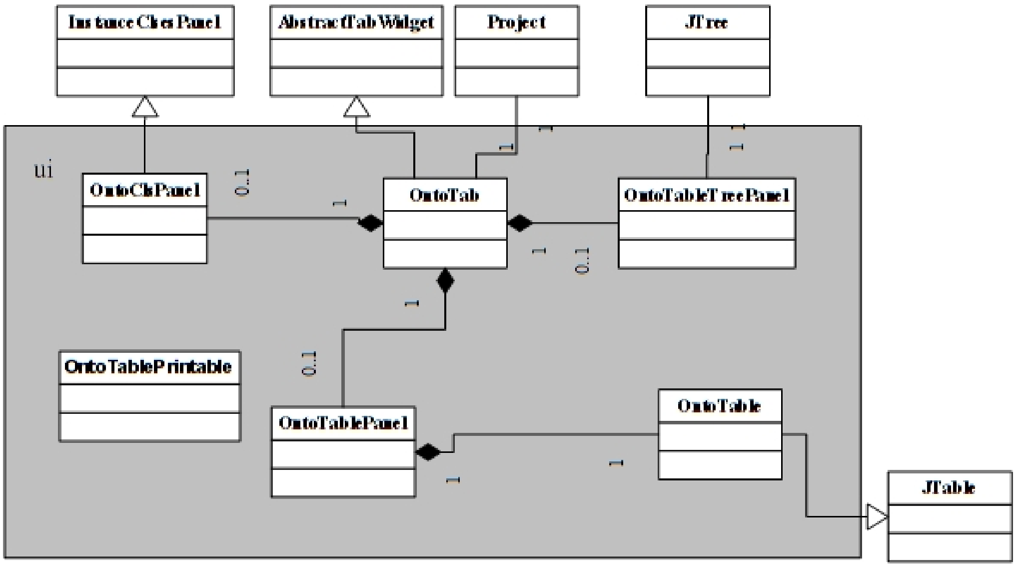

Abbildung 4: UML Diagramm des ui - Packages

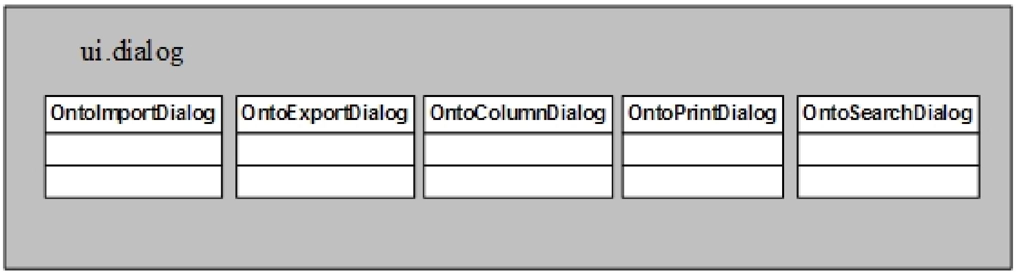

Abbildung 5: UML Diagramm des dialog.ui - Packages

4.3 ui.dialog.filter

Um die komplexe Funktion Filtern zu realisieren, werden innerhalb des Filterdialoges mehrere Klassen benötigt, was ein eigenes Package für die Filterdialoge nötig macht. Für die verschiedenen Datentypen werden hier die einzelnen Klassen zusammengefasst.

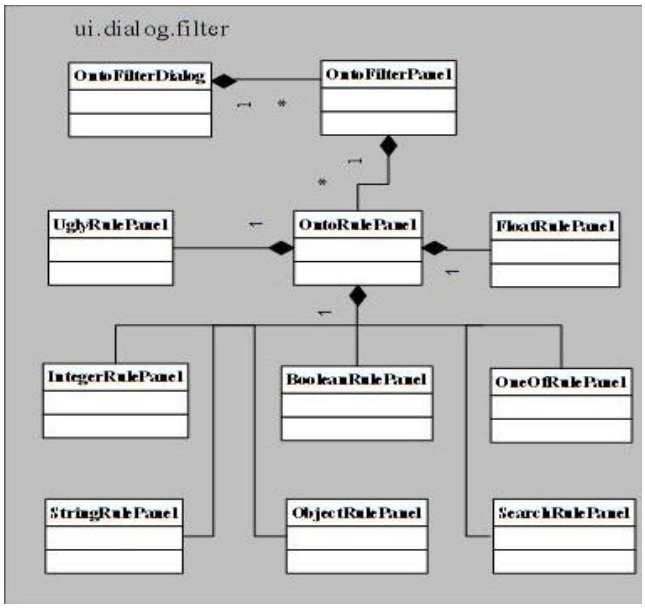

Abbildung 6: UML-Diagramm des ui.dialog.filter Packeges

4.4 model

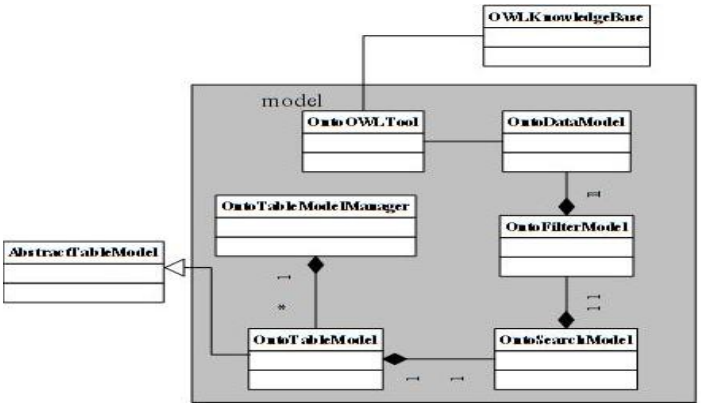

Abbildung 7: UML-Diagramm des model Packeges

Das Package model beinhaltet alle Klassen, die die Datenhaltung und die beschriebenen Funktionen der Tabelle realisieren (Details siehe Kapitel 3).

4.5 model.filter

Das Package model.filter enthält die Regeln, um Filter von verschiedenen Datentypen zu ermöglichen.

4.6 importExport

Im Package importExport und importExport.dialog werden die Klassen gebündelt, die die Logik und die GUI für die Import- und Exportfunktionen zur Verfügung stellen.

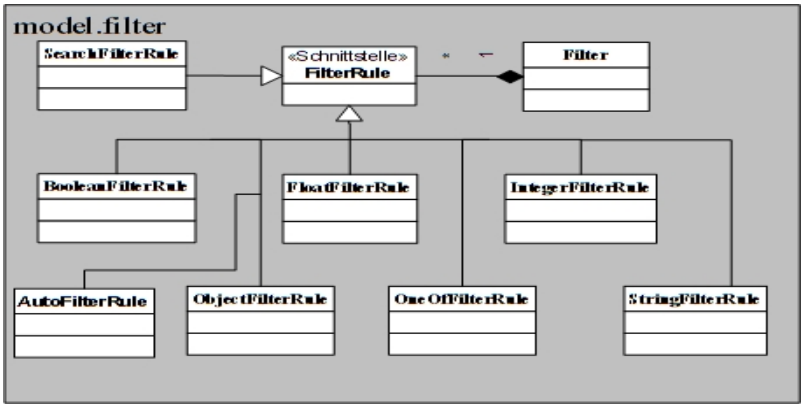

Abbildung 8: UML-Diagramm des model.filter

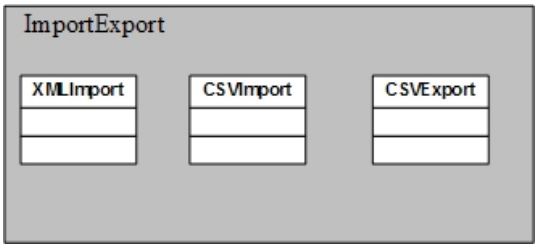

Abbildung 9: UML-Diagramm des ImportExport - Packeges