Gruppe: HK-07-4
Gruppenleiter: Stanley Hillner 14.05.2007

Entwurfsbeschreibung

1. Allgemeines

Mit dem Ecore-Editor ist es möglich, Ecore Dateien, also Ecore-Modelle graphisch zu modellieren. Dies vereinfacht die Erstellung eines Ecore-Modells, da man die EClasses übersichtlich als Klassen in der bekannten, an UML angelehnten Form visualisiert hat.

2. Produktübersicht

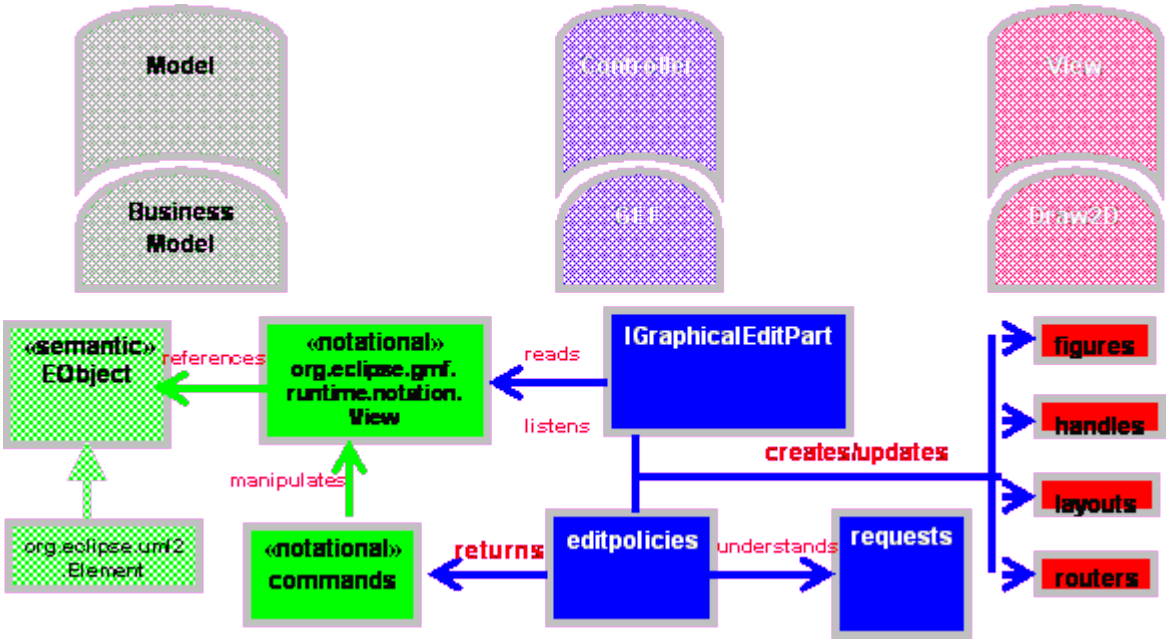
Dieser Editor bietet eine in Eclipse integrierte Benutzeroberfläche, welche aus einer Zeichenfläche und einer Werkzeugpalette besteht. Die Werkzeugpalette enthält Zeichenelemente für EClasses, EPackages, EAnnotations, EDataTypes und ENums. Des Weiteren gibt es noch Eine Palette für Compartments, wie EAttribute, EOperations, EAnnotationDetails und ENumLiterals. Die dritte Gruppe von Tools sind die Verbindungen zwischen den Elementen, wie Assoziation, Aggregation, Generalisation und EAnnotationReferenz.
Somit ist es möglich, ein Ecore-Modell graphisch zu modellieren, da alle in Ecore verwendbaren Elemente auch in diesem Editor „regelgerecht“ eingesetzt werden können.

3. Grundsätzliche Struktur und Entwurfsprinzipien

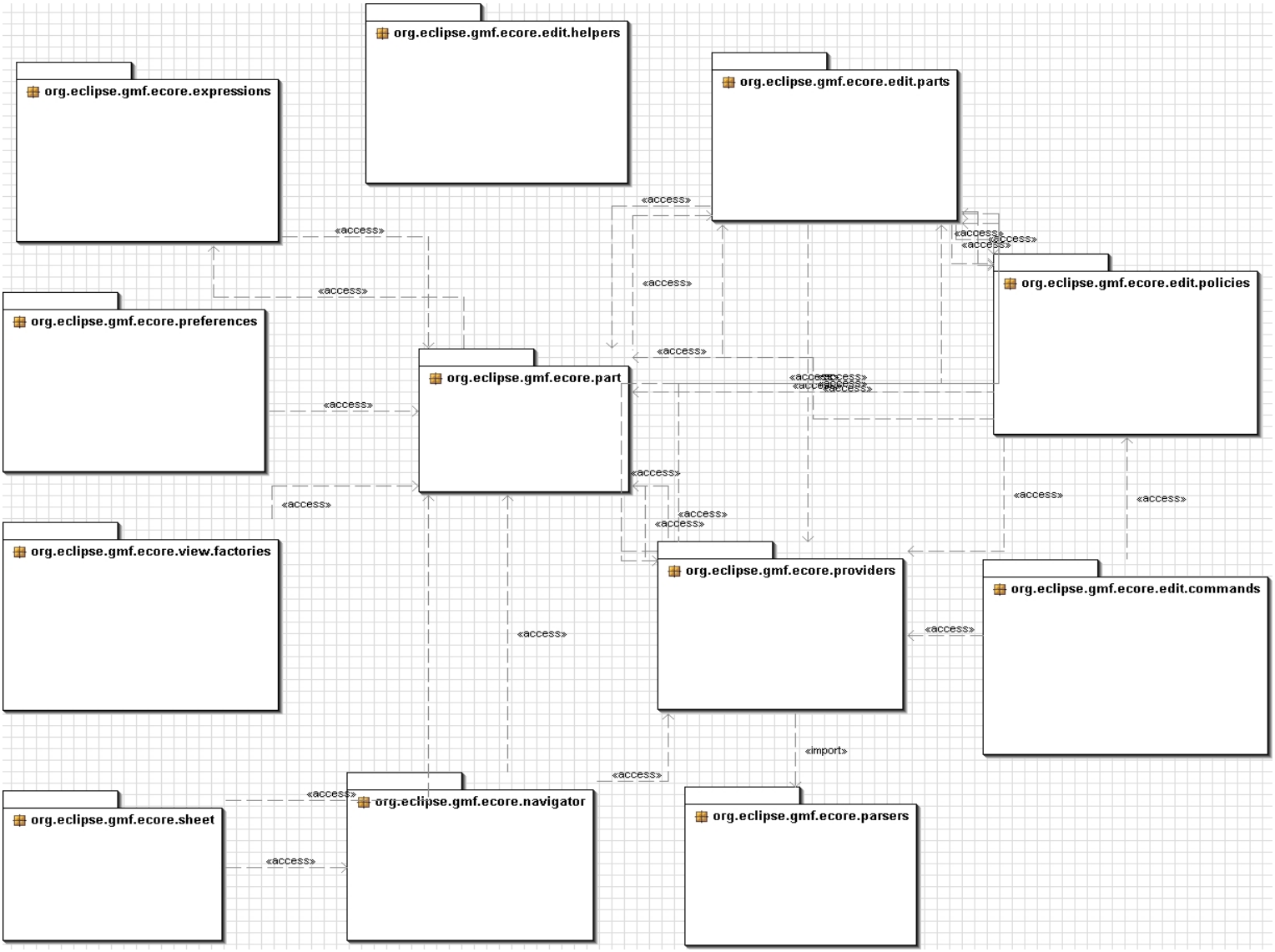
Umsetzung der Model-View-Controller-Architektur

Die MVC-Architektur in GMF ist wie in unten stehendem Bild aufgebaut. Die untere Schicht, welche hier durch EObject, IGraphicalEditPart, View, ... dargestellt wird, ist im Vergleich zu der MVC-Architektur in GEF um einiges komplexer. Die MVC-Architektur ist in GMF deshalb so komplex, weil alle erzeugbaren Editoren durch diese Architektur verwaltet werden müssen und es soll dem Benutzter auch noch ausreichend Spielraum zur eigenen Einflussnahme, also für eigene Implementierungen bleiben. Grün stellt hier das Modell der MVC-Architektur dar, blau den Controller und rot den View.

Das Verhalten des Editors kann vom Programmierer vorrangig in den Policies geändert werden. Der Programmierer kann so bestehende Regeln und Funktionen der EditParts überschreiben, ohne den bestehenden Code, welcher durch GMF erzeugt wurde, zu manipulieren. Dazu muss der Anwender die gesendeten Requests, die ähnlich wie ActionListener fungieren, in der entsprechenden EditPolicy abfangen. Dann kann der Programmierer durch das ermitteln des zum View gehörigen semantischen Elements die entsprechenden Einstellungen an diesem vornehmen.


Model
Business Model
Controller
GEF
View
Draw2D
«semantic» EObject
references
«notational» org.eclipse.gmf. runtime.notation. View
reads
IGraphicalEditPart
listens
figures
handles
layouts
routers
creates/updates
manipulates
org.eclipse.uml2 Element
«notational» commands
returns
editpolicies
understands
requests

Strukturierung der Packages



Das oben stehende Diagramm zeigt die Paketstruktur des Plugins. Die Wichtigsten Pakete und ihre Funktionen werden im nachfolgenden etwas näher erklärt.

edit.commands
Im Paket edit.commands sind Klassen, die eine Sammlung von commands beinhalten um Model-Elemente zu editieren.

edit.helpers
In diesem Paket werden Requests empfangen und in commands umgewandelt, damit die requests entsprechend verarbeitet werden können. Dafür wird auf die edit.commands

zugegriffen um die den requests entsprechenden commands zu finden.

edit.parts
Dieses Paket beinhaltet Klassen für alle im Diagrammmodell definierten Diagrammelemente. Die Klassen enthalten den Code für:
-die definierte Figur
-das Erzeugen des entsprechenden Diagrammelements
-Labels: Erzeugen, Refresh, direktes Editieren, Löschen, Listner und Notifications verwalten

edit.policies
In diesem Paket sind ausgelagerte Operation zum Behandeln von Requests. Diese werden von edit.parts aufgerufen. Eine edit.policy kann von mehreren edit.parts verwendet werden.

expressions
Paket stellt API und Implementierung von Klassen für eine einheitliche XML-Sprache, die in den extension points verwendet werden. Diese Sprache ist nicht an einen speziellen extension point gebunden und kann in allen extension points verwendet werden.

providers
Interfaces die die services implementieren und dadurch bieten sie die Möglichkeit die vorhandenen services zu erweitern.

view.factories
Dieses Paket beinhaltet die Beschreibung des Metamodells und enthält createView, welche vom View aufgerufen wird um das Metamodell grafisch umzusetzen.

Anbindung an die Eclipse-IDE

In Eclipse ist ein Plugin eine Komponente die einen bestimmten Dienst im Kontextmenü der Arbeitsumgebung in Eclipse zur Verfügung stellt. Mit Komponente sei hier ein Objekt gemeint, dass während der Systemstartpahse in ein System eingebunden werden kann. Die Eclipse Laufzeitumgebung stellt eine Infrastruktur bereit, welche die gleichzeitige Aktivierung und Steuerung mehrerer Plugins im nahtlosen Zusammenspiel für die Entwicklungsaktivitäten ermöglicht. Innerhalb einer laufenden Eclpise-Instanz, ist ein Plugin in einer Instanz irgendeiner Plugin-Laufzeitumgebung-Klasse, oder kurz Plugin-Klasse enthalten. Die Plugin-Klasse stellt die Konfigurations- und Verwaltungsunterstützung für die Pluginumgebung bereit. Eine Plugin-Klasse muss für Eclipse auf dem org.eclipse.core.runtime.Plugin aufbauen, welches eine abstrakte Klasse mit allgemeinen Einrichtungen zur Verwaltung der Plugins ist.
In dem zu analysierenden Fremdprojekt wird im Package „org.eclipse.gmf.ecore.part“ eine Instanz der Klasse Plugin „EcoreDiagrammEditorPlugin“ implementiert, welche von der abstrakten Superklasse für UIPlugins „org.eclipse.ui.plugin.AbstractUIPlugin“ abgeleitet wird. Diese ist wieder von der Superklasse für alle Plugins „org.elcipse.core.runtime.Plugin“

abgeleitet.
Jedes Plugin wird in einer XML-Definionsdatei „plugin.xml“ definiert. Die Datei teilt der Eclipse-Laufzeitumgebung mit, was getan werden muss um das Plugin zu aktivieren. Der Plug-in Manifest-Editor bietet eine graphische Darstellung und Manipulation der Inhalte der „plugin.xml“. Die „plugin.xml“ enthält für die Einbindung der Funktionalitäten des Editors in die Eclipse IDE sogenannte extensions, welche mittels extension points an eclipse gebunden werden.
Beispielweise wird die Funktionalität des Erstellens einer neuen Ecore-Datei mittels des graphischen Editors über die unten aufgeführten extension points in die Eclipse IDE eingebunden.

```
<extension point="org.eclipse.ui.editors">
      <?gmfgen generated="true"?>
      <editor
         id="org.eclipse.gmf.ecore.part.EcoreDiagramEditorID"
         name="%editorName"
         icon="icons/full/obj16/EcoreModelFile.gif"
         extensions="ecore_diagram"
         default="true"
         class="org.eclipse.gmf.ecore.part.EcoreDiagramEditor"
         matchingStrategy="org.eclipse.gmf.ecore.part.EcoreMatchingStrategy"
         contributorClass="org.eclipse.gmf.ecore.part.EcoreDiagramActionBarCo
                           ntributor">
      </editor>
   </extension>

<extension point="org.eclipse.ui.newWizards">
   <?gmfgen generated="true"?>
   <wizard
         name="%newWizardName"
         icon="icons/full/obj16/EcoreModelFile.gif"
         category="org.eclipse.ui.Examples/org.eclipse.gmf.examples"
         class="org.eclipse.gmf.ecore.part.EcoreCreationWizard"
         id="org.eclipse.gmf.ecore.part.EcoreCreationWizardID">
         <description>%newWizardDesc</description>
      </wizard>
   </extension>
```

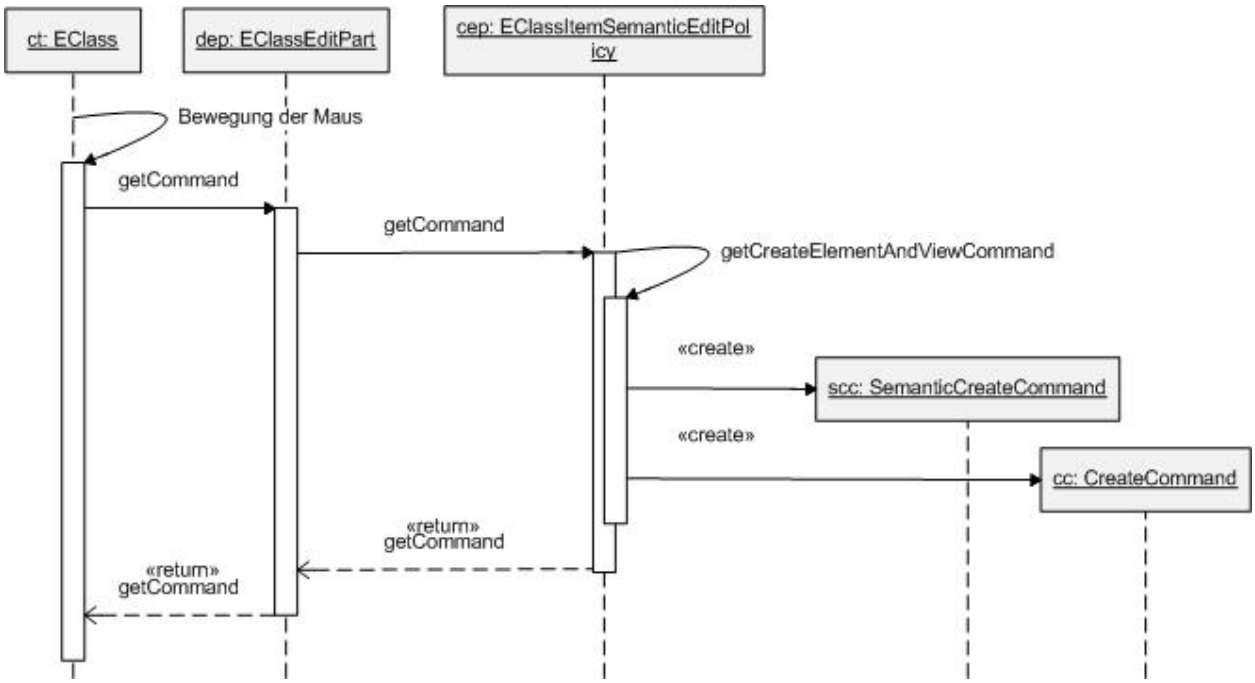
Andere Funktionalitäten, wie zum Beispiel Popup-Menüs oder auch die Erweiterung des Kontextmenüs können ähnlich hinzugefügt werden. Zu jeder Funktionalität gibt es natürlich spezifische extension points, welche sich von den obigen entsprechend unterscheiden.

Interne Abläufe bei der Modellierung einer EClass

Die Erzeugung einer EClass wird durch die unten stehenden Sequenzdiagramme verdeutlicht. Dabei werden von GMF einige Methoden von GEF überschrieben, um somit die Erzeugung der

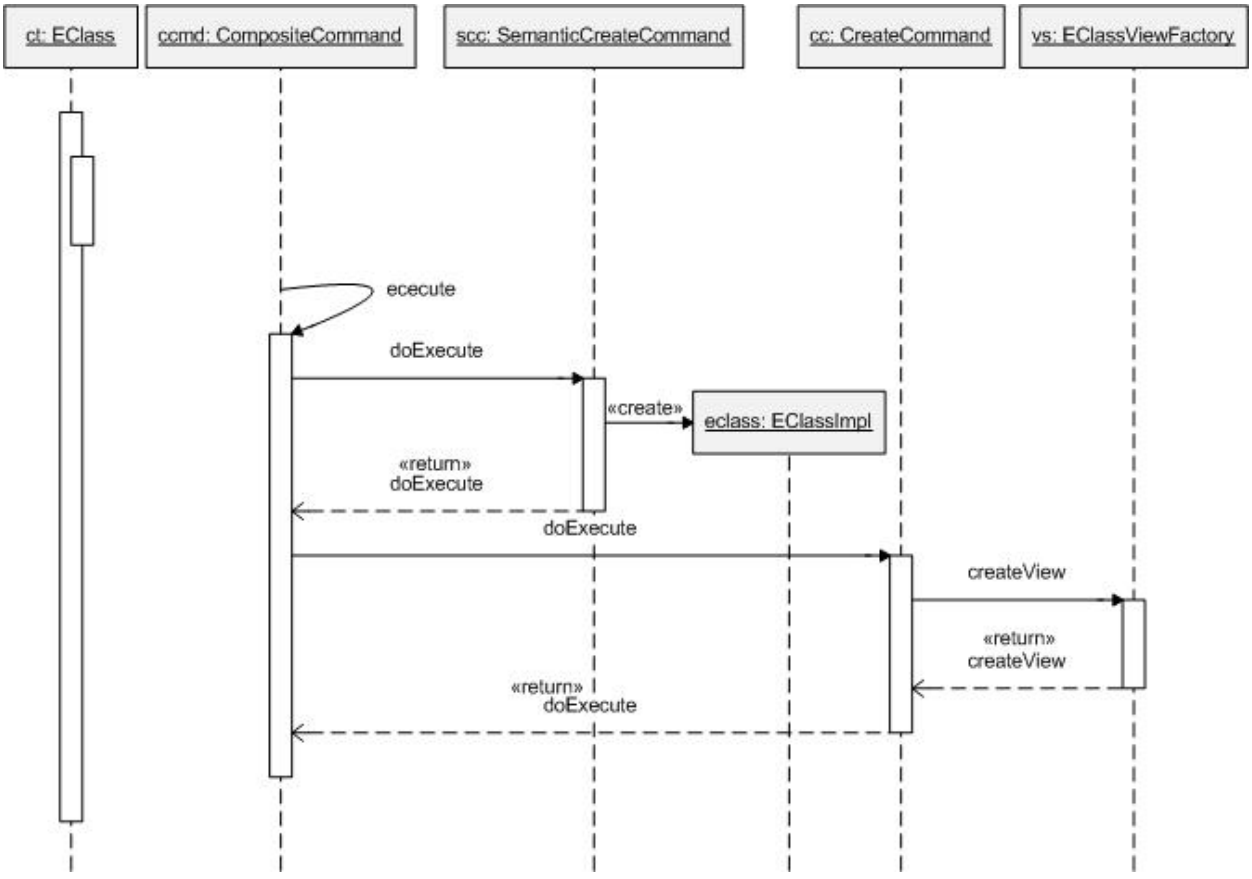
Objekte (hier die EClass) zu realisieren.

Für die Erstellung einer EClass wird das entsprechende CreationTool gewählt und beim Bewegen der Maus über die Zeichenfläche ein Request an die Klasse EClassEditPart gesendet, welche mit diesem Request nach dem Cammand zur Erstellung der Eclass gefragt wird. Diese Klasse leitet den Request als Anfrage nach dem Command an die EClassSemanticItemEditPolicy weiter, welche dann mit Hilfe der Instanzen von SemanticCreateCommand und CreateCommand aus der GMF-runtime den Command zum Erstellen einer EClass erzeugt. Dieser Command wird dann an die Aufrufende Klasse (EClassEditPolicy) zurückgegeben und diese liefert den Command letztendlich an das CreationTool. Diese Commands erzeugen kein Element, bevor die Ausführung durch die aufrufende Factory initiiert wird. Dies ist wichtig um Undo/Redo Operationen zu behandeln, die durch die Command Infrastruktur behandelt werden.



Das unten stehende Diagramm zeigt den Ablauf der Erstellung bei Klick auf die Zeichenfläche. Wenn der Command“Stack“ erzeugt wurde, wird jeder TeilCommand seriell abgearbeitet. Bei der Ausführung wird zuerst das semantische Element erzeugt und das Resultat daraus wird der CreateCommand Klasse übergeben übergeben, welche dann die Methode createView der EClassViewFactory aufruft, die dann das Notation Element (View Element) zu dem gegebenem semantischen Element erzeugt. Die Methode createView wird eigentlich in der GMF-runtime-Klasse ViewService aufgerufen, welche dann aber über den entsprechenden Provider, den diese Klasse aus allen registrierten Providern heraussucht, die entsprechende ViewFactory (hier EClassViewFactory) zurückliefert. Die ViewFactory erstellt und initialisiert das Notation

Element, welches angefordert wurde. Das Zweite Sequenzdiagramm zeigt die zweite Phase der Erstellung des Elements, also die Erstellung des EditParts und der Figur. Hier hört die EditPart des Containers, zu der das semantische Element hinzugefügt wurde, das notification event ab, um das Element hinzufügen. Die Notification entsteht erst, wenn die Schreibaktion, die die Kommandoausführung umhüllt, abgeschlossen ist. Dabei wird alles, was in der Schreibaktion passiert, in einer Transaktion registriert, damit die Aktionen rückgängig gemacht und wiederhergestellt werden können. Die EClassEditPart ruft dann die Methode refreshChildren auf. Wenn dann die EditPartFactory benötigt wird, liefert GMF die EditPartService zurück, welche dieses GEF Interface implementiert. Dieser EditPartService findet ebenso, wie der ViewService den zugehörigen Provider aus allen registrierten Providern. Dieser Provider liefert den EditPart zurück, den der EditPartService erzeugt.



Gruppe: HK-07-4

Gruppenleiter: Stanley Hillner 14.05.2007