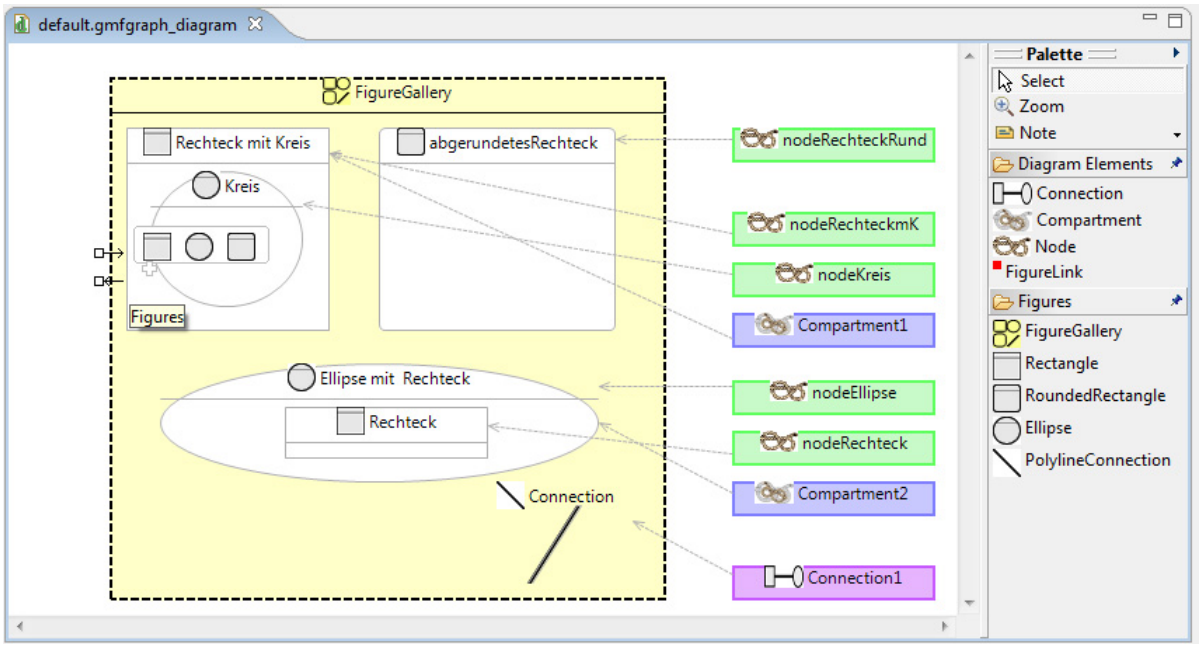
Gruppe: HK-07-4
Gruppenleiter: Stanley Hillner 25.06.2007

# Entwurfsbeschreibung

## 1. Allgemeines

Mit dem zu entwickelnden GMF-Editor ist es möglich, die für ein graphisches Editor-Plugin für Eclipse benötigten Dateien (*.gmfgraph, *.gmftool & *.gmfmap) zu einem gegebenen DomainModel (*.ecore) auf graphischem Weg zu erstellen, also graphisch zu modellieren. Dies vereinfacht die Erstellung dieser Dateien enorm, da die bisherige Erstellung in einer Baumstruktur stattfand, was erstens sehr kompliziert und zweitens unübersichtlich ist.
Der Editor ist ein Eclipse-Plugin, welches mittels GMF erzeugt wird. Dafür müssen also ebenfalls das Domain-Model (*.ecore) und die zugehörigen GMF-Modelle (*.gmfgraph, *.gmftool & *.gmfmap) entwickelt werden.

## 2. Produktübersicht



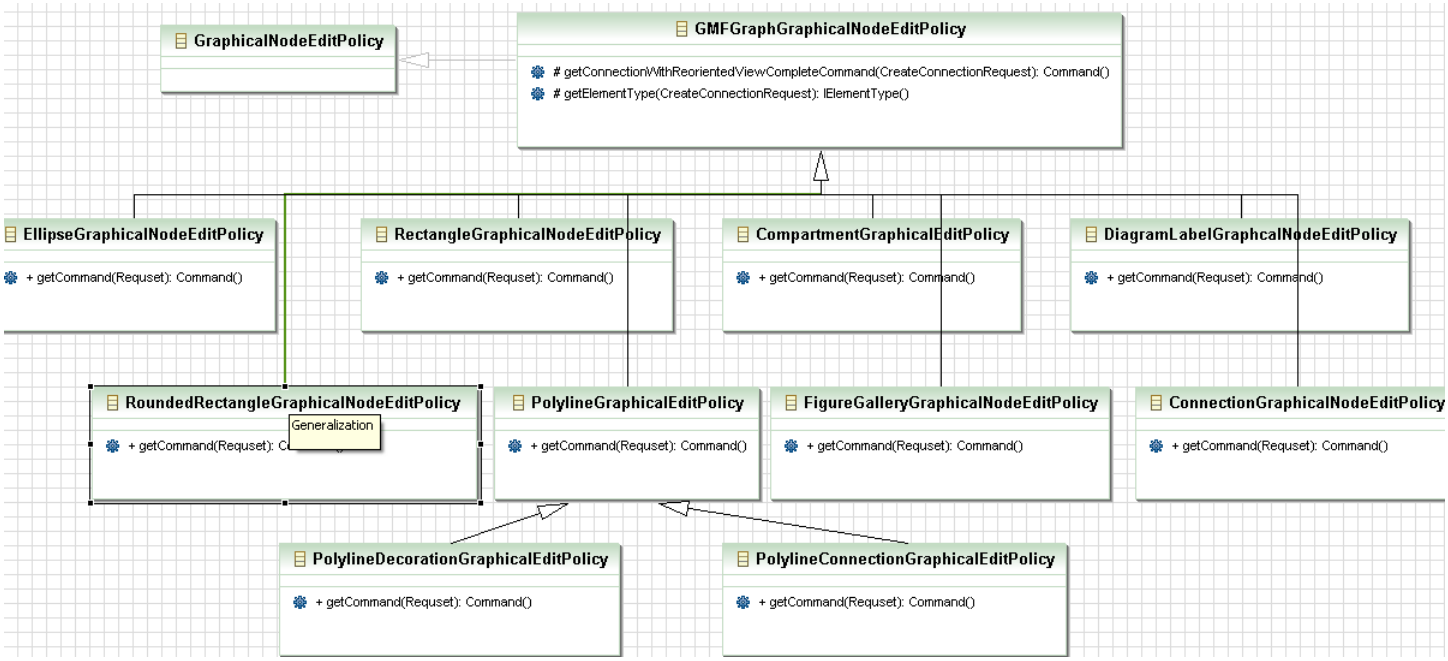

Dieser Editor bietet eine in Eclipse integrierte Benutzeroberfläche, welche aus einer Zeichenfläche und einer Werkzeugpalette besteht (siehe Screenshot oben). Die Werkzeugpalette beinhaltet die Zeichenwerkzeuge für die Elemente, die in einem GMF-GRAPH-Modell eingefügt werden können. Diese sind durch die Toolgroup Figures gruppiert. Man kann z.B. eine FigureGallery durch Klicken und Ziehen auf die Zeichenfläche zeichnen, welche im Bild gelb dargestellt ist. In dieser FigureGallery ist es möglich, in gleicher Weise Elemente, wie Rechtecke, Ellipsen, abgerundete Rechtecke und PolylineConnections zu zeichnen. Diese werden dann intern in der *.gmfgraph als Kinder der FigureGallery instanziiert. Außerhalb der FigureGallery ist es dem Nutzer möglich, Nodes, Compartments oder

Connections als Rechtecke zu zeichnen, welche er dann mittels FigureLink mit dem entsprechenden Element in der FigureGallery verbinden muss. Zu den Nodes und Connections wird in der *.gmftool automatisch nach der Vergabe eines Namens ein entsprechendes CreationTool erzeugt. Nun muss der Nutzer noch durch Rechtsklick zu einer Node oder Connection die entsprechende EClass seines Ecore-Modells aus einem Kontextmenü auswählen. Dann wird der User aufgefordert, für die Node die Referenz anzugeben, die in der *.gmfmap als ContainmentFeature dienen soll. Selbiges wird für das targetFeature der Connection gefordert. Anschließend wird in der *.gmfmap der gewünschte Mapping-Eintrag erstellt. Dazu muss der Nutzer vor Beginn der Arbeiten an seinem Editor sein Ecore-Modell spezifizieren, wozu er durch ein Popup-Menü aufgefordert wird. Klickt der Nutzer hier auf Abbrechen, so wird der Editor nicht gestartet und es werden keine Dateien vom Editor erzeugt. Wenn der User eine Ecore-Datei wählt, die sich nicht im aktuellen Workspace befindet, dann bekommt er eine Meldung, bei welcher er die Möglichkeit hat, die Ecore direkt in seinen Workspace zu kopieren. Nach der Auswahl der Ecore wird der Benutzer aufgefordert, eine EClass zu wählen, die dem Mapping als DomainMetaElement dienen soll.
Das Erstellen eines Compartments läuft wie folgt ab:
Der User zeichnet eine Figur in der FigureGallery und die Figur, die das Compartment sein soll, in diese Figur hinein. Anschließend zeichnet der User eine Node und verbindet diese mittels FigureLink mit dem Element, das das Compartment enthalten soll. Dann muss der Nutzer ein Compartment auf die Zeichenfläche zeichnen und dieses mit demselben Element verbinden. Danach muss der Nutzer zu dieser Node die EClass und das ContainmentFeature wählen. Nachdem der Nutzer dies erledigt hat, muss er eine Node zu dem Element zeichen, welches das Compartment ist und diese verbinden. Danach wählt der Nutzer zu der Node des Compartments die EClass und das passende ContainmentFeature. Nun wird der passende Map-eintrag erstellt. Löscht der Nutzer eine Node oder Connection, so wird auch der Map-Eintrag und der Eintrag in der gmftool gelöscht.

# 3. Grundsätzliche Struktur und Entwurfsprinzipien

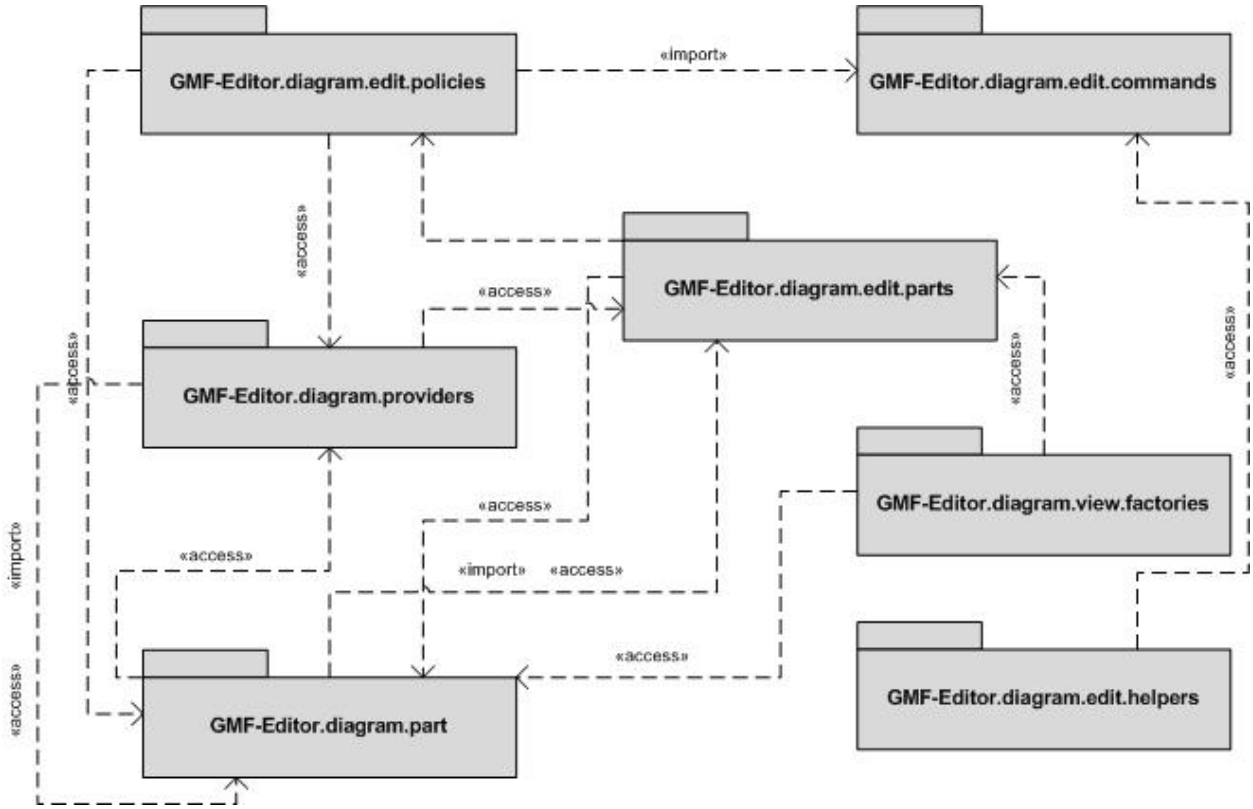
### *1. Aufbau des Statischen Modells*

Das zu entwickelnde Plugin wird von GMF generiert. Aus diesem Grund wird bei nachstehender Beschreibung der statischen Struktur des Editors mittels Klassendiagramm das Augenmerk auf die Klassen gelegt, welche von uns geändert werden, damit der Editor die geforderte Funktionalität erhält.




Zur Erstellung des korrekten Editor-Codes wurden im Wesentlichen die Methoden „getCommand“ der jeweiligen GraphicalEditPolicy-Klassen und die Methoden „activate“ der EditPart-Klassen überschrieben. In diesen Methoden werden alle wichtigen Änderungen der grafischen Elemente auf die zu erzeugenden Dateien übertragen. Des Weiteren wurden noch die Methoden „getLabelText“ in der Klassen NodeNameEditPart und ConnectionNameEditPart, die Methoden „getDestroyElementCommand“ der Klassen NodeItemSemanticEditPolicy und ConnectionItemSemanticEditPolicy, sowie die Methode „createNewDiagramFile“ der Klasse GmfgraphDiagramEditorUtil um einige Anweisungen ergänzt, um die Einträge der *.gmftool zu realisieren. Um die Mapping-Einträge zu realisieren, wurden die Klassen des Packages custom erstellt, welche die Einträge per xml speichern und zuvor die Ecore auslesen. Funktionalitäten sind in den Klassen mittels Kommentaren erklärt. Außerdem dient dieses Package dazu, die Mappin-Einträge zu erstellen und zu löschen.

**Strukturierung der Packages**




Das oben stehende Diagramm zeigt die Paketstruktur des Plugins mit deren Abhängigkeiten. 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 untergebracht.

Diese werden von edit.parts aufgerufen. Eine edit.policy kann von mehreren edit.parts verwendet werden.

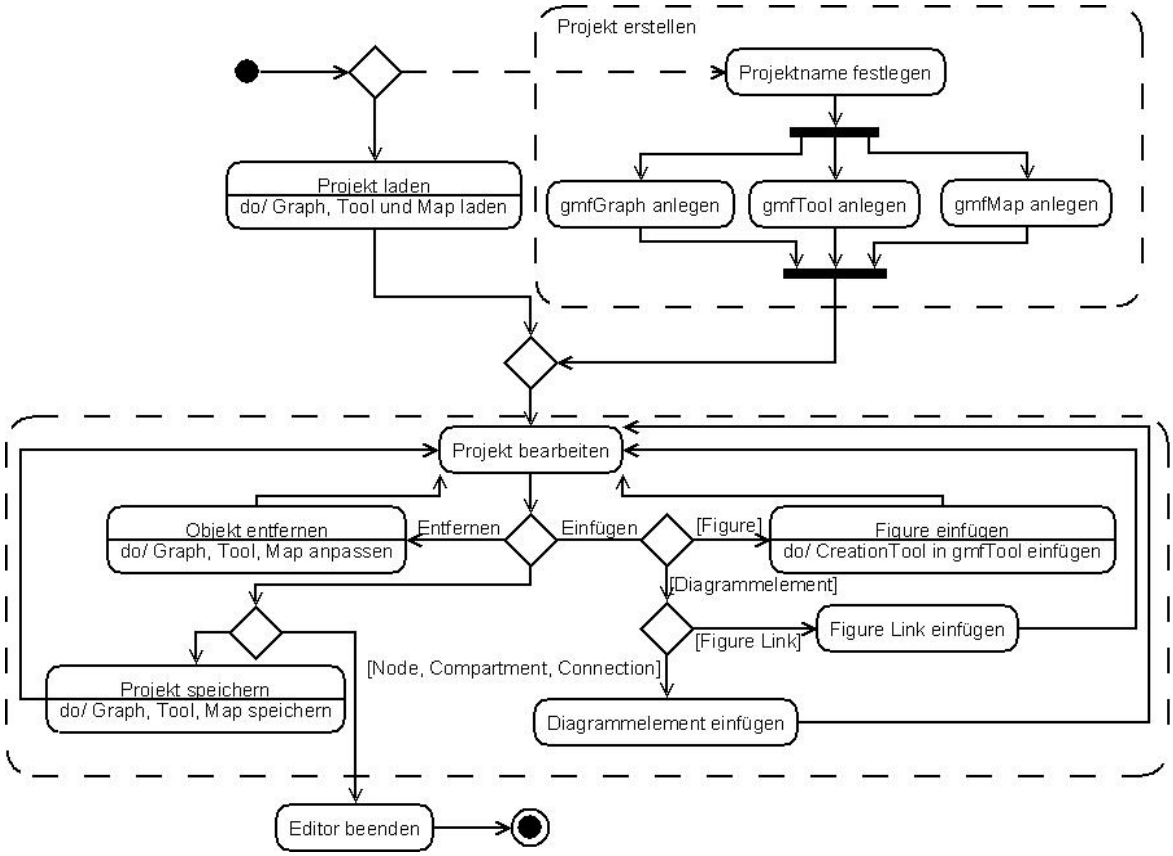
**providers**
Dieses Paket enthält Klassen, welche die Klassen zurückgeben, die eine konkrete Funktionalität bereitstellen.

**view.factories**
Factories sind für die Erstellung der einzelnen grafischen Elemente zuständig, die auf der Zeichenfläche gezeichnet werden. Sie erstellen außerdem die Editparts für jedes Element des Editors.

## *2. dynamisches Modell*

**Aktivitätsdiagramm**

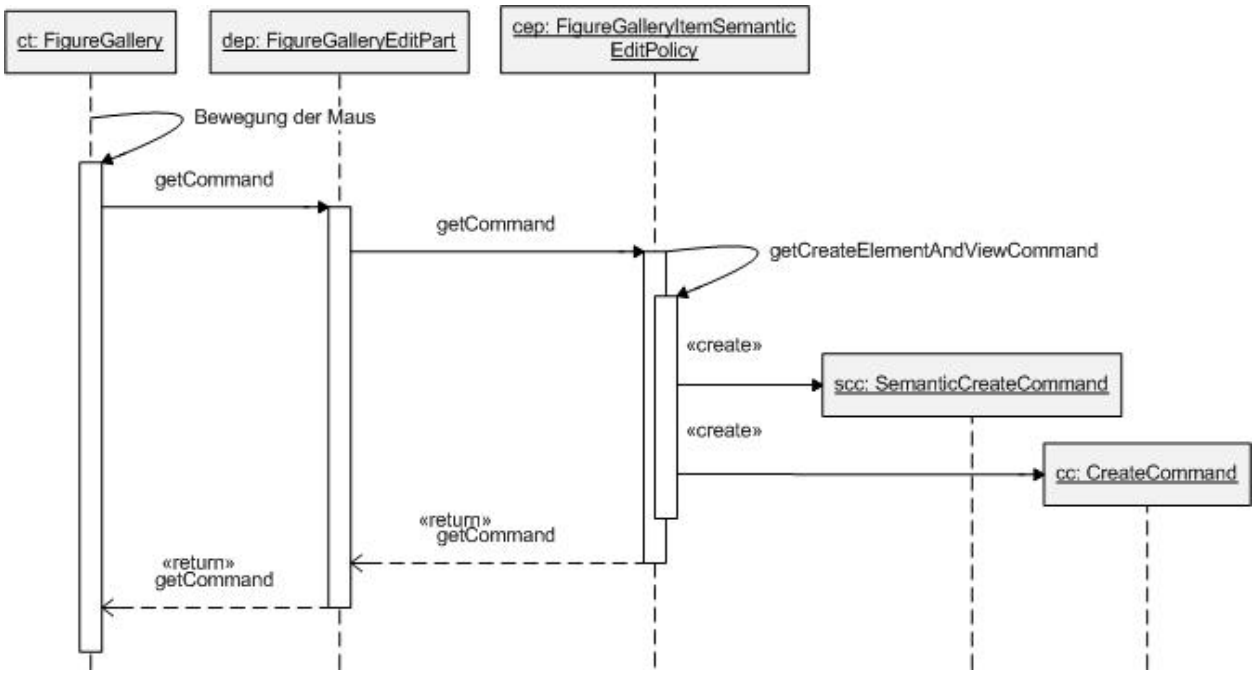
Das Aktivitätsdiagramm beschreibt die Abläufe im Editor bei der Bearbeitung des Projekts. Der User hat die Auswahl, entweder ein neues Projekt zu erstellen, also die *.gmfgraph und die dazugehörigen Dateien *.gmftool und *.gmfmap zu modellieren, oder ein bereits bestehendes Projekt zu laden, indem er die *.gmfgraph in den Editor lädt und so ein neues Diagramm erzeugt.
Während der Modellierung des Diagramms hat er die Möglichkeit, Objekte, wie etwa eine FigureGallery, Rechtecke, Ellipsen oder ähnliches der Zeichenfläche hinzuzufügen. Er kann aber auch Elemente wie Nodes, Compartments und Connections zur Zeichenfläche hinzufügen und diese mittels FigureLink mit einem der Objekte in der FigureGallery verbinden. Intern werden diese Objekte dann entsprechend instanziiert und die Änderungen werden beim Speichern übernommen. Es ist ebenfalls möglich, Elemente wieder zu löschen.



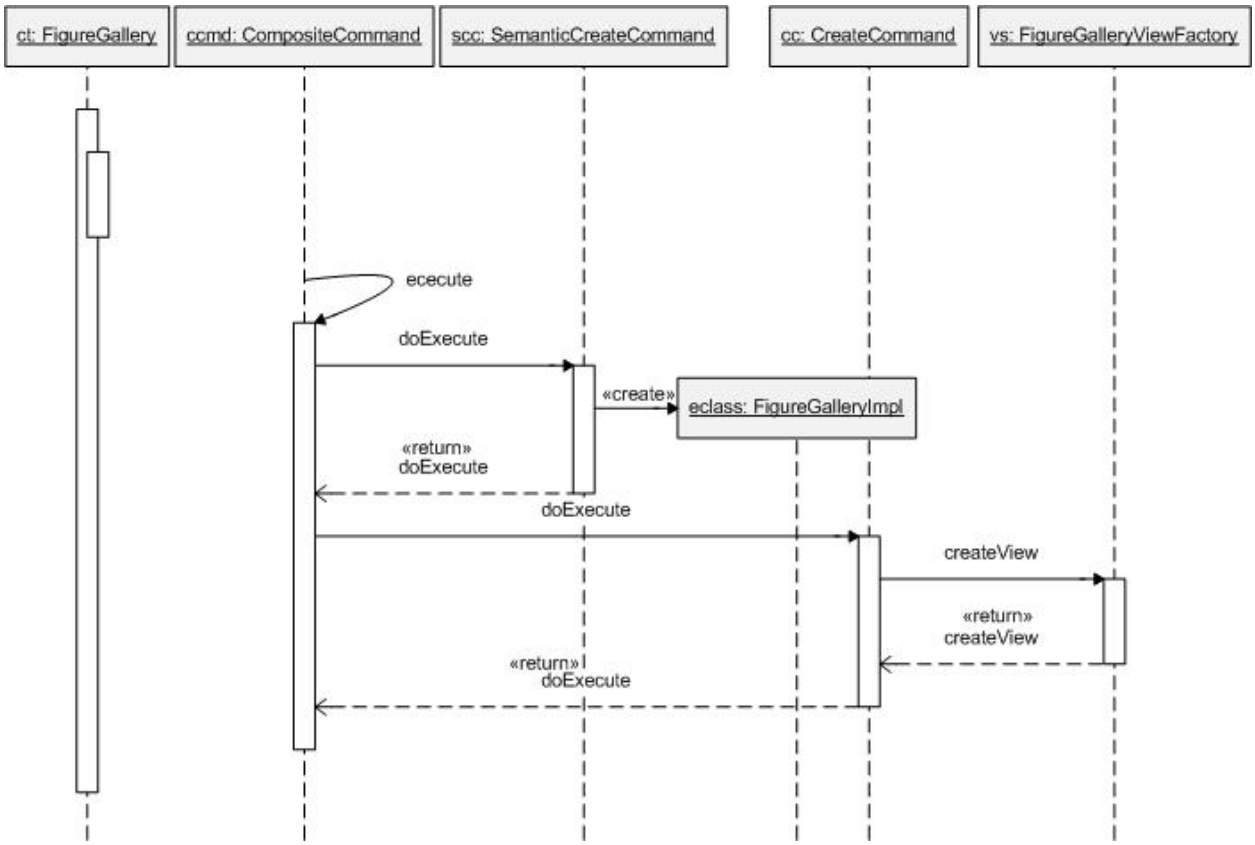

**Sequenzdiagramme**

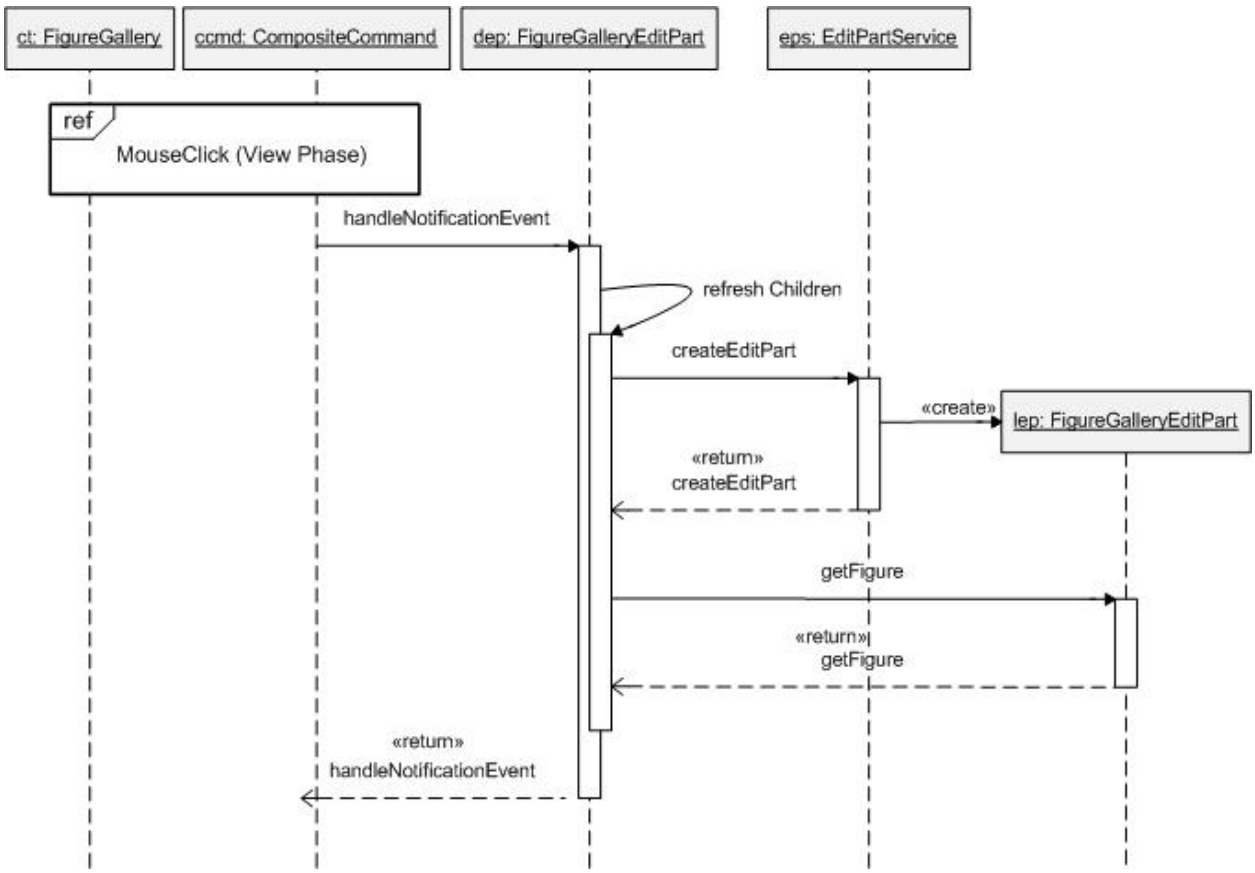
Nachfolgend werden die internen Abläufe am Beispiel der Erstellung einer FigureGallery mittels Sequenzdiagramme verdeutlicht.

Für die Erstellung einer FigureGallery wird das entsprechende CreationTool gewählt und beim Bewegen der Maus über die Zeichenfläche ein Request an die Klasse FigureGalleryEditPart gesendet, welche mit diesem Request nach dem Command zur Erstellung der FigureGallery gefragt wird. Diese Klasse leitet den Request als Anfrage nach dem Command an die FigureGallerySemanticItemEditPolicy weiter, welche dann mit Hilfe der Instanzen von SemanticCreateCommand und CreateCommand aus der GMF-runtime den Command zum Erstellen einer FigureGallery erzeugt. Dieser Command wird dann an die aufrufende Klasse (FigureGalleryEditPolicy) 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 FigureGalleryViewFactory 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 FigureGalleryViewFactory) 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 FigureGalleryEditPart 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.


ct: FigureGallery
ccmd: CompositeCommand
scc: SemanticCreateCommand
cc: CreateCommand
vs: FigureGalleryViewFactory
ececute
doExecute
«create»
eclass: FigureGalleryImpl
«return»
doExecute
doExecute
createView
«return»
createView
«return»
doExecute


ct: FigureGallery
ccmd: CompositeCommand
dep: FigureGalleryEditPart
eps: EditPartService
ref
MouseClick (View Phase)
handleNotificationEvent
refresh Children
createEditPart
«create»
lep: FigureGalleryEditPart
«return»
createEditPart
getFigure
«return»
getFigure
«return»
handleNotificationEvent