Untersuchung eines Fremdprojektes

13.05.07

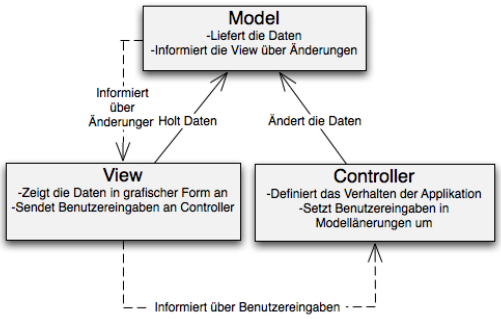
Inhaltsverzeichnis

1 Allgemeines

Wir beschäftigten uns während der Softwarestudie eingehender mit GMF 1.02. Ziel war, die Abhängigkeiten und Mechanismen hinter der Oberfläche zu verstehen, um sie im Projekt selbst verbessern zu können. Dafür dient uns das Fremdprojekt org.eclipse.gmf.ecore.editor, dessen Struktur wir Softwarestudie untersuchen. Die Schwerpunkte sind: Umsetzung der MVC-Architektur Strukturierung der packages Anbindung an die eclipse-IDE die internen Abläufe bei der Erstellung einer EClass

2 Umsetzung der Model-View-Controller-Architektur

Der Nutzer legt mit Hilfe der grafischen Benutzeroberfläche EClass, EPackage, EAnotation, EAtribute, Association usw. an und manipuliert deren Parameter. Diese Elemente, samt ihrer Parameter und ihrer Beziehungen untereinander, müssen in einem internen Datenmodell gespeichert werden. Der aktuelle Zustand die- ses Modells, kann dann auf der grafischen Benutzeroberfläche visualisiert werden. Zur Umsetzung dieser Aufgaben eignet sich die sogenannte Model-View-Controller-Architektur (MVC-Architektur) sehr gut. Dabei werden die anfallenden Aufgaben auf die drei Komponenten Model, View und Cont- roller verteilt. Durch die Trennung in diese drei Komponenten, werden die Wiederverwendbarkeit einzelner Programmteile ermöglicht sowie spätere Änderungen und Erweiterungen vereinfacht. Die folgende Abbildung zeigt das Zusammenspiel der Komponenten:



Modell Das Model wird die persistenten Daten der Anwendung enthalten. Es kennt weder die View noch den Controller. Es können sich jedoch beliebig viele Views beim Model registrieren, um über Änderungen informiert zu werden. Die von Nutzer angelegten EClasses, EPackage, EAtribute werden im Model gespeichert. Alle Änderungen, die über die graphische Benutzeroberfläche an den Parametern dieser Elemente vorgenommen werden, haben eine Änderung innerhalb des Datenmodells zur Folge.

View Die View ist eine visuelle Darstellung des Datenmodells. Für ein Model können beliebig viele Views existieren, die das Model auf unterschiedlichste Weise darstellen. Alle Interaktionen, die zwischen Anwender und Entwicklungswerkzeug stattfinden, werden über die Benutzeroberfläche ablaufen.

Controller Die Eingaben, die der Nutzer in einer View vornimmt, werden an den Controller weitergeleitet, der diese Eingaben verarbeitet, das Datenmodell entsprechend manipuliert und darüber entscheidet, wel- che View angezeigt wird.

3 Strukturierung der Packages

3.1 org.eclipse.gmf.ecore.edit.parts

CONTROLLER-Paket : startet den Ablauf, vermittelt zwischen Model und View
Enthält die ECoreEditPartFactory mit der Methode createEditPart(EditPart, Object), die die editPart-Klassen für die einzelnen EClasses anlegt:Jedes Node bekommt eine EditPart-Klasse, die von ShapeNodeEditPart erbt und folgende wichtige Eigenschaften hat:

- Attribut public static final int VISUAL ID
- Überschreiben der Methode createDefaultEditPolicies, mit der die edit.policies installiert werden.
- Enthalten der Klasse Figure aus org.eclipse.draw2d

Jedes Attribut eines Nodes bekommt eine EditPart-Klasse, mit der es angesprochen, erzeugt und angezeigt werden kann. Typisch: setLabelText, createNode, setLabel, addVisualChild, refreshLabel, createFigure, removeVisualChild-Methoden in der Classe EClassESuperTypesEditPart.java werden die Beziehugen zwischen den klassen festgelegt, und die Decoration der Verbindung.

3.2 org.eclipse.gmf.ecore.edit.policies

vor allem Controllercharakter, reagiert auf Benutzerereignisse, enthält aber auch View-Regeln, und schickt Refresh-Requests an das eCore-Model
Diese Package dient zur Festlegung der Regeln, bzw. das Vorgehen bei bestimmten Interaktionen des Benuzers Alle Nodes haben eine GraphicalNodeEditPolicy, die von EcoreGraphicalNodeEditPolicy erben und keine zusätzlichen Eigenschaften haben. Die ist fürs Node zuständig.Alle Nodes haben mindestens eine CanonicalEditPolicy: Eine die von CanonicalEditPolicy erbt und wenn sie Referenzen haben die von CanonicalConnectionEditPolicy erbt. Hier wird eine Liste der Kinder des semantischen Objekts ausgegeben.Auch Connections haben so eine CanonicalEditPolicy, mit der semantische Verbindungen erstellt werden und zB. source und target ausgegeben werden: protected EObject getSourceElement(EObject relationship). Die Methode refreshSemantic() dient zur Aktualisierung des Views.Alle Elemente haben eine ItemSemanticEditPolicy, die von ECoreBaseItemSemanticEditPolicy erbt. Wichtige Methoden:getSemanticElement, getSemanticCommand, getMoveCommand, getCreateCommand, getDuplicateCommand, getDestroyCommand → Verbindung zum ECore.

3.3 org.eclipse.gmf.ecore.part

CONTROLLER-Paket - startet den Ablauf, vermittelt zwischen Model und View Enthält Hauptverbindungen zur IDE: Wizards zum Erzeugen und Öffnen der Diagramme, VID-Registry und Toolverwaltung. EcorePaletteFactory.java - In dieser Klasse werden sämtliche Eigenschaften der Tool- Palette bestimmt, wie zum Beispiel: Elemente die zur Palette gehören, sämtliche Beschreibungen und Icons. In dem Ecore-Beispiel werden 3 PaletteContainer erzeugt namens: 'createNodes1Group', 'createChildNodes2Group', 'createLinks3Group'. Diese sind logische Gruppierungen der entsprechenden Tools.

3.4 org.eclipse.gmf.ecore.providers

Das Providers-Paket verbindet die Schichten von ECore und zu den Edit.Parts → Model ↔ Controller
ECoreViewProvider importiert alle view.factories und alle edit.parts → verbindung stellt die Verbindung her. getDiagramViewClass gibt das Diagram-Root-Element zurück. getNodeViewClass und getEdgeViewClass bekommen (IAdaptable semanticAdapter, View container- View, String semanticHint) übergeben und geben die ViewFactory.class des entsprechenden Nodes oder Links zurück. ECoreElementTypes enthält eine Liste aller

existierenden EClasses und gibt diese auf Anfrage zurück: public static ENamedElement getElement(IAdaptable hint) EcoreStructuralFeatureParser enthält Methoden zum Überprüfen der Gültigkeit der Werte oder Kommandos einer Klasse. EcorePropertyProvider stellt Eigenschaften der Elements zur Verfügung. EcoreEditPartProvider verwaltet den EditPart und wird mit einem bestimmten View initialisiert.

3.5 org.eclipse.gmf.ecore.factories

CONTROLLER ↔ VIEW - Parser Das Package view.factories enthält Klassen für die im Diagramm angezeigten Elemente, die die Informationen aus den EditPart-Klassen ihrer Attribute holen und sie dann ans View weiterschicken. Dabei bekommt jede Klasse mindestens eine ViewFactory, je nach Anzahl der Attribute, meistens ist das die BasicNodeViewFactory oder AbstractShapeView-Factory.

3.6 org.eclipse.gmf.ecore.edit.commands

Unwichtig - es gibt nur eine Klasse ECoreReorientConnetionViewCommand, die dafür zuständig ist, eine Connection umzudrehen.

3.7 org.eclipse.gmf.ecore.edit.helpers

unwichtige Hilfsklassen

4 Anbindung an die Eclipse-IDE

Das Einbinden des Plugins erfolgt über die 'plugin.xml', in der folgender extension point definiert ist:

```
<extension point="org.eclipse.ui.editors">
<editor
    id="org.eclipse.gmf.ecore.part.EcoreDiagramEditorID"
      name="Ecore Diagram Editor"
      icon="icons/full/obj16/EcoreModelFile.gif"
      extensions="ecore_diagram"
      default="true"
      class="org.eclipse.gmf.ecore.part.EcoreDiagramEditor"
       (Pluginklasse wird hier festgelegt)
      matchingStrategy="org.eclipse.gmf.ecore.part.EcoreMatchingStrategy"
      contributorClass="org.eclipse.gmf.ecore.part.EcoreDiagramActionBarContributor">
     </editor>
</extension>
```

Das Plugin ist mit einem OSGi-Manifest versehen, das auf die folgende Datei verweist. → BundleActivator = org.eclipse.gmf.ecore.part.EcoreDiagramEditorPlugin Das ECoreDiagramEditorPlugin erbt von der Klasse AbstractUIPlugin, die wiederum von Plugin erbt. überschrieben werden die folgenden Methoden:

- start - verweist auf die super.start des AbstractUIPlugin und ¸bergibt einen BundleContext dahin, um sich als Bundle im OSGi-Framework zu registrieren.
- stop - verweist vor allem auf die super.stop und ¸bergibt den BundleContext, um sich daraus zu entfernen.

Das Starten eines neuen Diagramms erfolgt entweder über das Kontextmenü oder mit dem New→Examples→ECore→Diagram→Wizard.
KONTEXTMENÜ:
Der Erweiterunspunkt wird in der plugin.xml festgelegt, dort sieht der Verweis aus wie folgt:

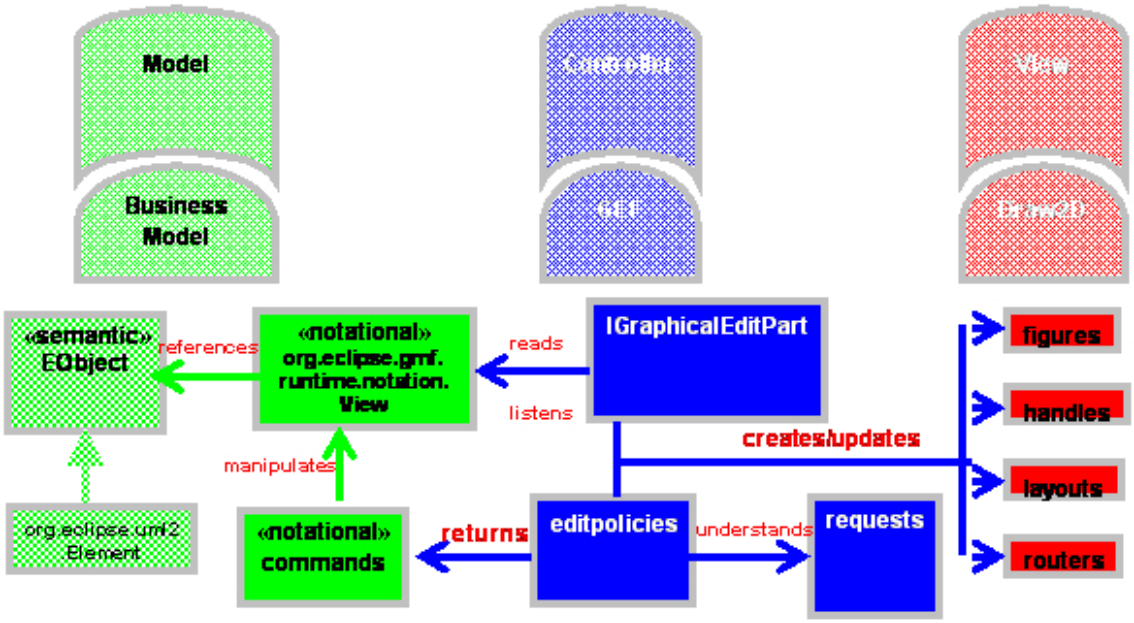
```
<objectContribution
      id="org.eclipse.gmf.ecore.editor.ui.objectContribution.IFile1"
      nameFilter="*.ecore"
      objectClass="org.eclipse.core.resources.IFile">
   <action
         label="Initialize ecore_diagram diagram file"
         class="org.eclipse.gmf.ecore.part.EcoreInitDiagramFileAction"
         menubarPath="additions"
         enablesFor="1"
         id="org.eclipse.gmf.ecore.part.EcoreInitDiagramFileActionID">
   </action>
</objectContribution>
```

Auf dieser Klasse wird die Methode run ausgeföhrt:
WIZARD: Der wird in der plugin.xml über den Einsprungspunkt 'org.eclipse.ui.newWizards' angesprochen, ist also über File→New.. zu erreichen. Im Einsprungspunkt steht der Verweis auf die Klasse class='org.eclipse.gmf.ecore.part.EcoreCreationWizard'. Diese erbt von EditorCreationWizard und hängt über addPages eine EcoreCreationWizardPage an den Wizard an, die die Methode createAndOpenDiagram auf EcoreDiagramEditorUtil ausführt:

```
public static final IFile createAndOpenDiagram(DiagramFileCreator diagramFileCreator,
IPath containerPath, String fileName, InputStream initialContents, String kind,
IWorkbenchWindow window, IProgressMonitor progressMonitor,
  boolean openEditor, boolean saveDiagram) {
IFile diagramFile = EcoreDiagramEditorUtil.createNewDiagramFile(diagramFileCreator,
 containerPath, fileName, initialContents, kind, window.getShell(), progressMonitor);
if (diagramFile != null && openEditor) {
IDEEditorUtil.openDiagram(diagramFile, window, saveDiagram, progressMonitor);
}
return diagramFile;
}
```

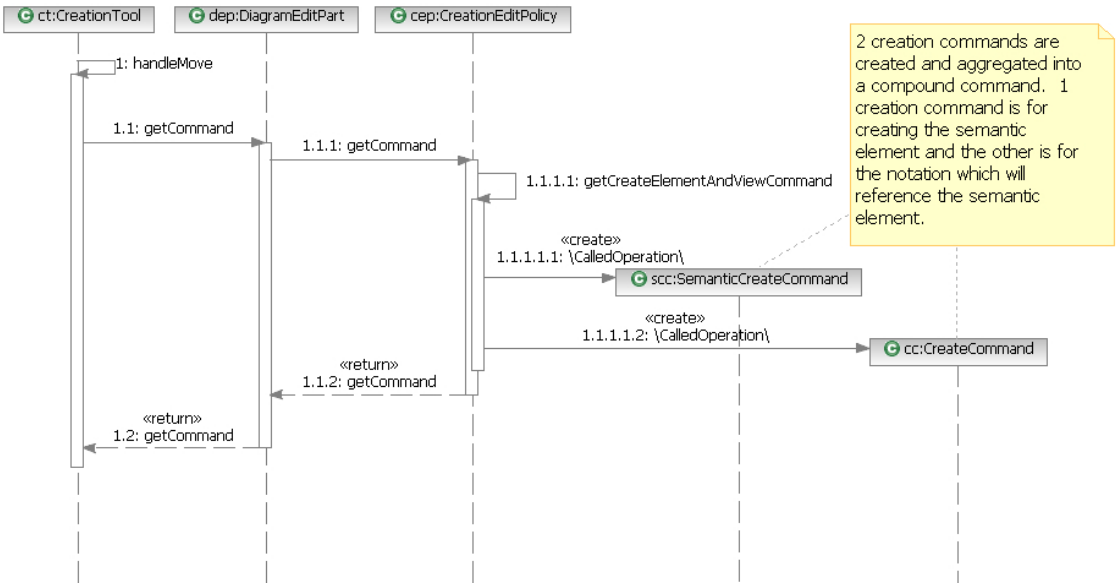
Von hier an ist das Diagramm geöffnet und kann editiert werden.

5 Interne Abläufe des Editors bei der Modellierung einer EClass

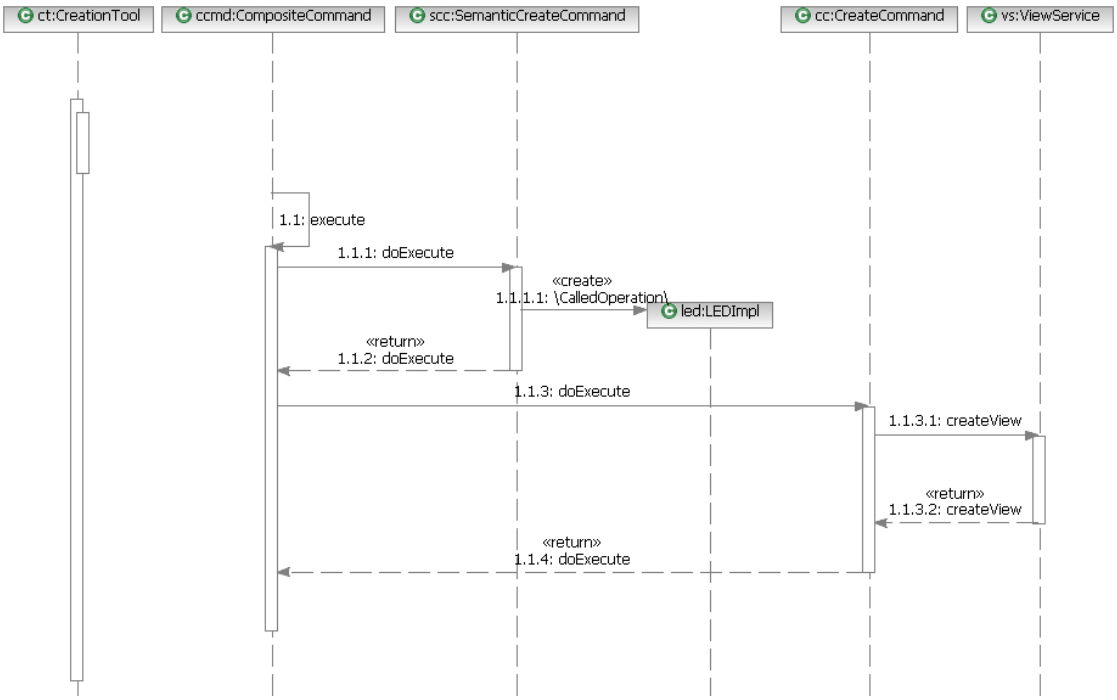


Den GMF-Editoren liegt die MVC-Architektur zugrunde, deswegen sind sie logischerweise in drei Komponenten zerlegt, wobei jede aus mehreren Teilen besteht. So gehören „EObject“,
„org.eclipse.gmf.runtime.notation.View“ und „commands“ zum von der Benutzerschnittstelle unabhängigen Model. „IGraphicalEditPart“, „editpolicies“ und „requests“ dienen zur Interaktion zwischen Model und View, und „figures“, „handles“, „layouts“ und „routers“ zur Präsentation der Informationen durch eine View. Die Unterteilung der MVC-Architektur in mehrere Unterpunkte macht es deutlicher, dass es sich um eine komplexe und umfangreiche Struktur handelt, die dem Entwickler eine Vielzahl an Möglichkeiten bietet. Die Controller (sogenannte EditParts) steuern das Zusammenwirken des Modells (z.B. der von EMF generierten Klassen) mit seinen Sichten (Views) (z.B. das Panel des grafischen Editors). So führt beispielweise das Absetzen des Kommandos Delete zunächst zu einer Änderung des Models (das Entfernen eines Objekts), woraufhin das Model eine Nachricht an den entsprechenden „EditPart“ sendet, der darauf hin einen Update der Darstellung des Modells im Editorfenster initialisiert. „GraphicalEditParts“ enthalten Methoden, um die entsprechenden Bilder im Editor zu erzeugen oder zu löschen und sie bei Modelländerungen zu aktualisieren. Dabei können diese Methoden durch “editpolicies überschrieben werden und somit die Funktionsweise des Editors bestimmen. Durch die Methode „getCommand“ aus „EditPolicy“ werden Abfragen („requests“) abgefangen und somit wird mit Hilfe des „org.eclipse.gmf.runtime.notation.View“ indirekt auf das

Modell zugegriffen. Modelle in GMF basieren auf dem Ecore-Metamodell. Das oberste Element eines Modells ist immer ein EPackage, das wiederum in mehrere Subpackages unterteilt sein kann. Klassen werden durch EClasses, primitive Datentypen durch EDataTypes repräsentiert. Beide sind unter dem abstrakten Typ EClassifier zusammengefasst. Attribute und Referenzen einer Klasse werden durch EAttribute bzw. EReference dargestellt, die beide wiederum von dem abstrakten Typ EStucturalFeature erben. Methoden einer Klasse und deren Parameter werden im Ecore-Metamodell durch EOperation bzw EParameter repräsentiert. Alle Klassen stellen Teile eines Modells bzw dessen Gesamtheit (EPackage) dar - daneben gibt es zu jedem EPackage noch jeweils eine eigene EFactory, die alle (nicht abstrakten) EClasses eines Modells erzeugen kann. GMF stellt eine Überbrückung zwischen EMF und GEF dar. Es werden Methoden in GEF überschrieben und somit EClasses (die für einzelne Objekte stehen) erzeugt. Grob formuliert handelt es sich bei der Erzeugung von EClasses durch den ecore diagram-editor um die Reaktion der EFactory, die bei einer Mausbewegung nach dem zu erzeugenden Objekt fragt und durch das Betätigen auch dieses Objekt erzeugt, d.h. es in einen bestimmten Container hinzufügt. Die folgenden Abbildungen beziehen sich nicht auf unseres Beispiel, veranschaulichen aber deutlich Erzeugung der EClasses, genau genommen in diesem Fall die Erzeugung der LED.



In der Abbildung 2 wird eine Ereigniskette dargestellt, die ausgelöst wird, wenn ein CreationTool per Mausklick ausgewählt wird. CreationEditPolicy behandelt den „request“, indem es ein Semantic Creation Command und ein Notational Creation Command zusammenfasst und zurück liefert.



Die Abbildung 3 und 4 zeigen den internen Ablauf nach dem per Mausklick ausgewählten CreationTool. In der Abbildung 3 wird das semantische und das Markierung- (Notational) Objekt erzeugt, in Abbildung 4 die dazugehörigen EditParts und die Figur. Abbildung 3. Zuerst wird das semantische Objekt erzeugt und dem „Notational Creation Command“ übertragen. Danach folgt die Erzeugung des „Notational“-Objekts, indem es ein „ViewService“ aufgerufen wird. Der „ViewService“ sucht nach einem Provider, der zum Objekt passt, und welcher die „ViewFactory “ zurückgibt, die notwendig ist, um das zum Objekt gehörende „Notational“-View Objekt zu erstellen und zu initialisieren. Alle „commands“ werden zeitlich nacheinander ausgeführt.

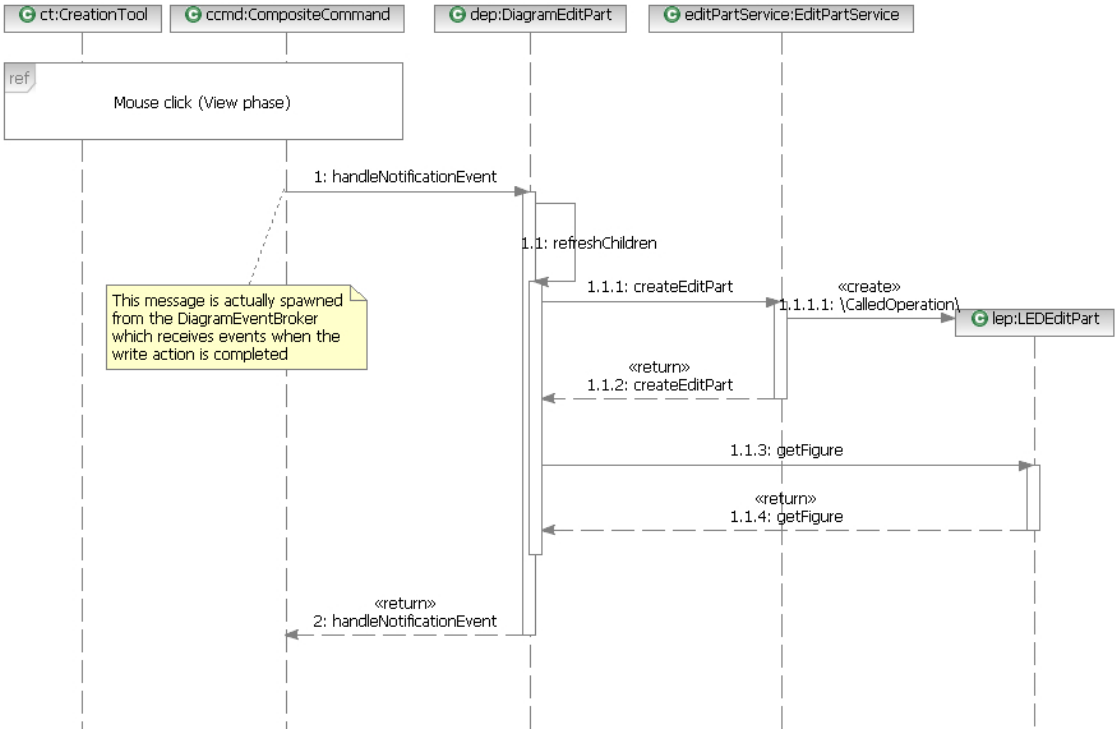


Abbildung 4. In der Abbildung 4 werden EditParts erzeugt. Der EditPart(„refreshChildren“) braucht eine Benachrichtigung, dass er das Objekt in den Container hinzufügen kann, die er auch bekommt, sobald der Schreibvorgang mit allen Unter-„commands“ abgeschlossen wird. Dabei wird alles registriert, um es wieder rückgängig zu machen bzw. zu kopieren zu können. Der „EditPartService“ sucht nach dem passenden Provider, welcher die EClass zurückgibt, die der „EditPartService“ erzeugt hat.