Erstellt durch: Tobias, Peter, Liang

Entwurfsbeschreibung Fremdprojekt

1. Allgemeines

Beschrieben wird der (interne) Aufbau eines PlugIns für die Integrierte Entwicklungsumgebung (IDE) Eclipse. Mit diesem ist es möglich, eCore-Diagramme zu erstellen. Bezogen wird sich dabei auf den Quellcode des Releases 1.0 Maintenance.

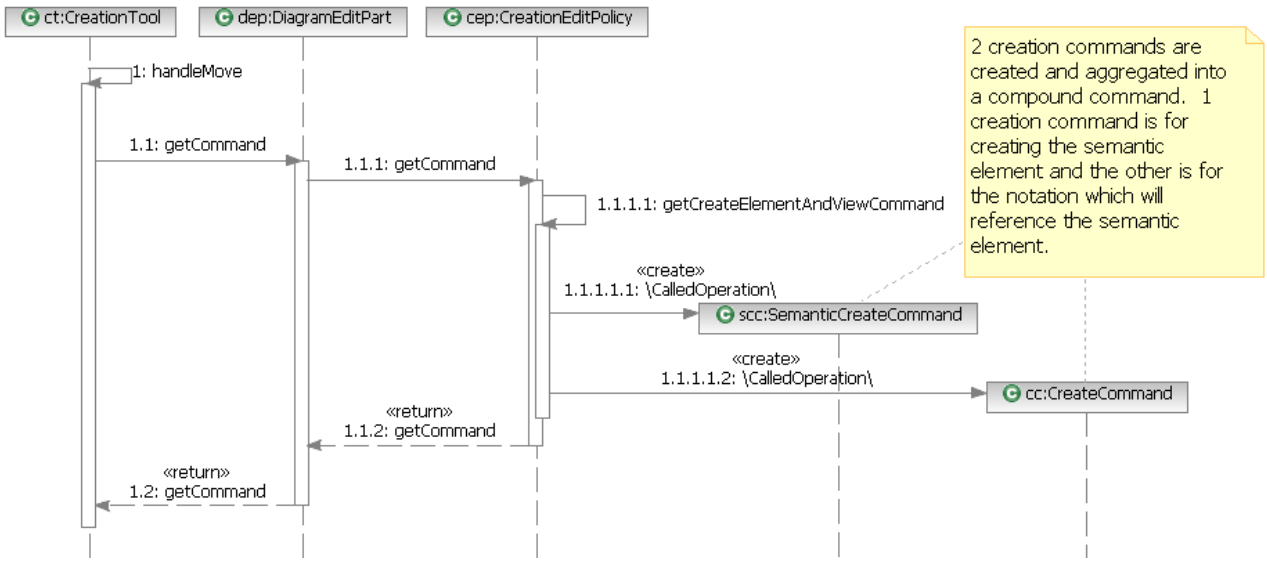
2. Produktübersicht

Der Editor stellt eine Zeichenoberfläche bereit, auf die, durch Auswahl der entsprechenden Tools aus der Toolbar Objekte gezeichnet werden können. Diese repräsentieren dann die EClasses, EPackages, EAnnotations, EDatatyps und EEnums. Zusätzlich besteht die Möglichkeit, die Objekt weiter zu präzisieren, zum Beispiel bei der EClass durch EAttributes, EAnnotations sowie EOperations. Aus dem Diagramm wird nach dem Speichern die zugehörige eCore-Datei erzeugt.

3. Struktur und Entwurfsprinzipien des Gesamtsystems

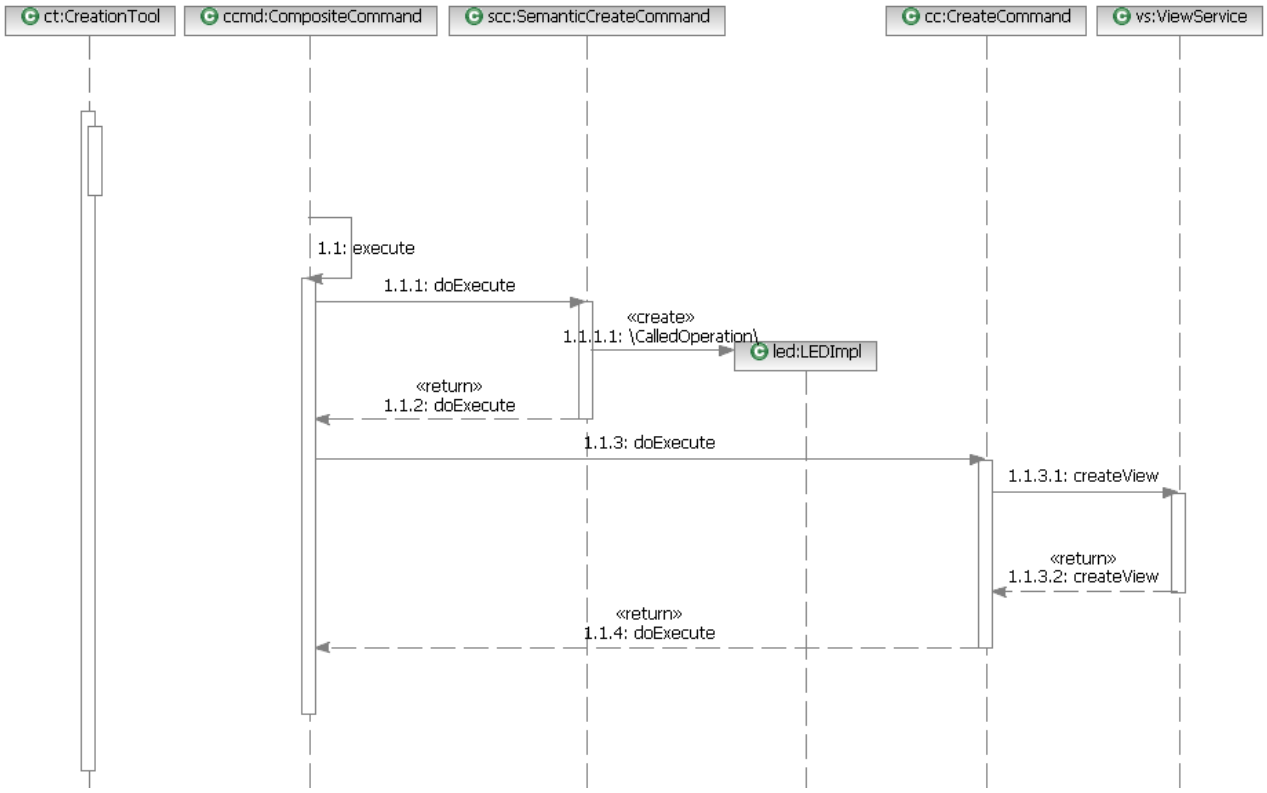
3.1. Beschreibung der internen Abläufe bei der Erstellung einer EClass

Um die Erzeugung einer EClass in GMF zu verstehen, sollte man zunächst wissen, das GMF selbst „nur“ eine Brücke baut zwischen EMF und GEF. GMF überschreibt ein Teil der Methoden in GEF, die für die Erzeugung von „Objekten“, in unserem Fall EClasses verantwortlich sind. Unförmlich gesagt, folgt die Erzeugung dem Prinzip eines Requests (Anfrage), der eine Factory bei einer Mausbewegung nach dem zu erzeugendem Element fragt und bei einem Mausklick das erzeugte Element zum Zielcontainer hinzufügt. Anmerkend sei hinzugefügt, das GEF nicht zwischen dem semantischen Modell und dem notatiellen Element bei der Erzeugung unterscheidet. Die nachfolgenden Abbildungen zeigen, ohne dabei nicht auf unser Beispiel übertragbar zu sein, die Erzeugung einer LED.

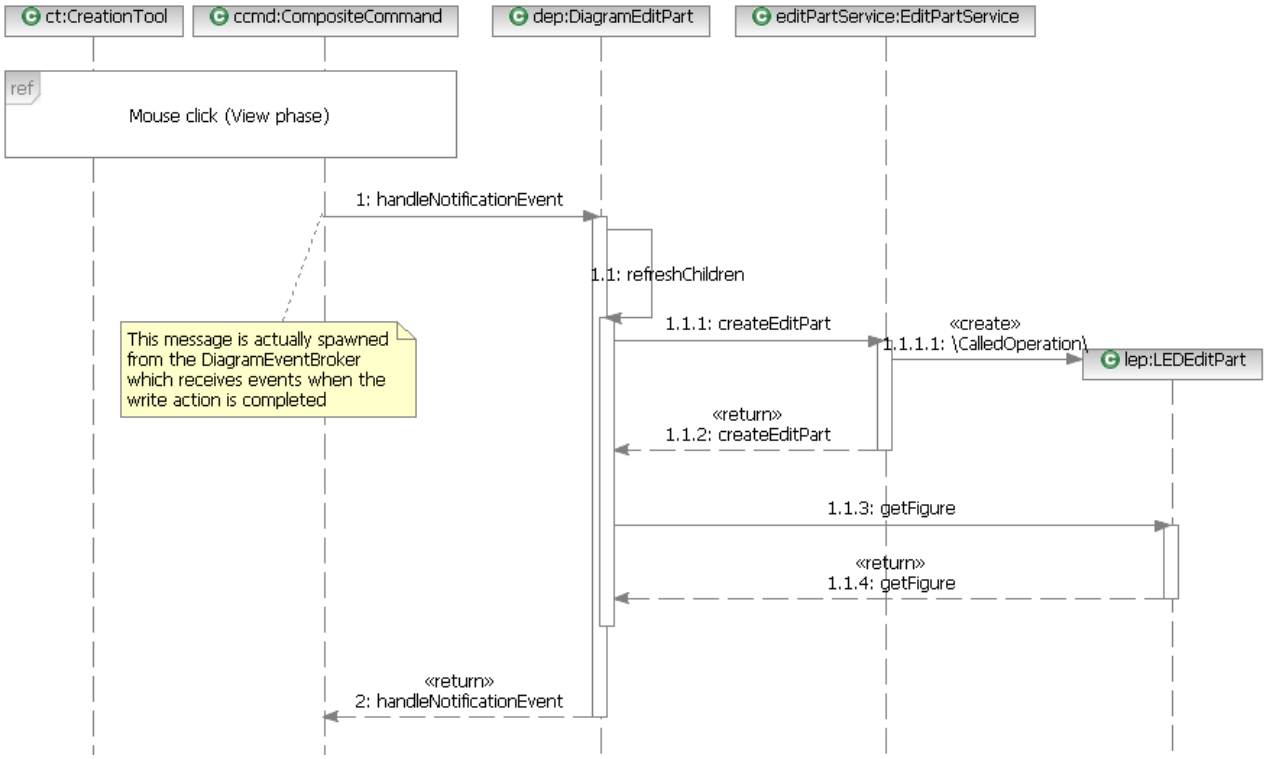


Die oben stehende Abbildung zeigt in einem UML Interaktionsdiagramm den Ablauf nach der Auswahl eines CreationTools. Sobald die Maus die Zeichenfläche berührt, wird die Ereigniskette ausgelöst. Der Request wird bei GMF durch die CreationEditPolicy behandelt, welche ein Semantic Creation Command

und ein Notational Creation Command zusammenfasst und zurück liefert. Dabei ist zu bemerken, dass diese Commands kein Element erzeugen, bevor eine Ausführung durch die aufrufende Factory initiiert wird. Dies ist wichtig um Undo/Redo Operationen zu behandeln, die durch die Command Infrastruktur behandelt werden. Ergänzungen zum Diagramm: die CreationTool Klasse des Diagramms entspricht der EClassCreationTool Klasse, welche in der GMFtool-Datei definiert und direkt durch GMF generiert wurde. Der DiagramEditPart entspricht dem EClassEditPart, die CreationEditPolicy entspricht der EClassItemSemanticEditPolicy.

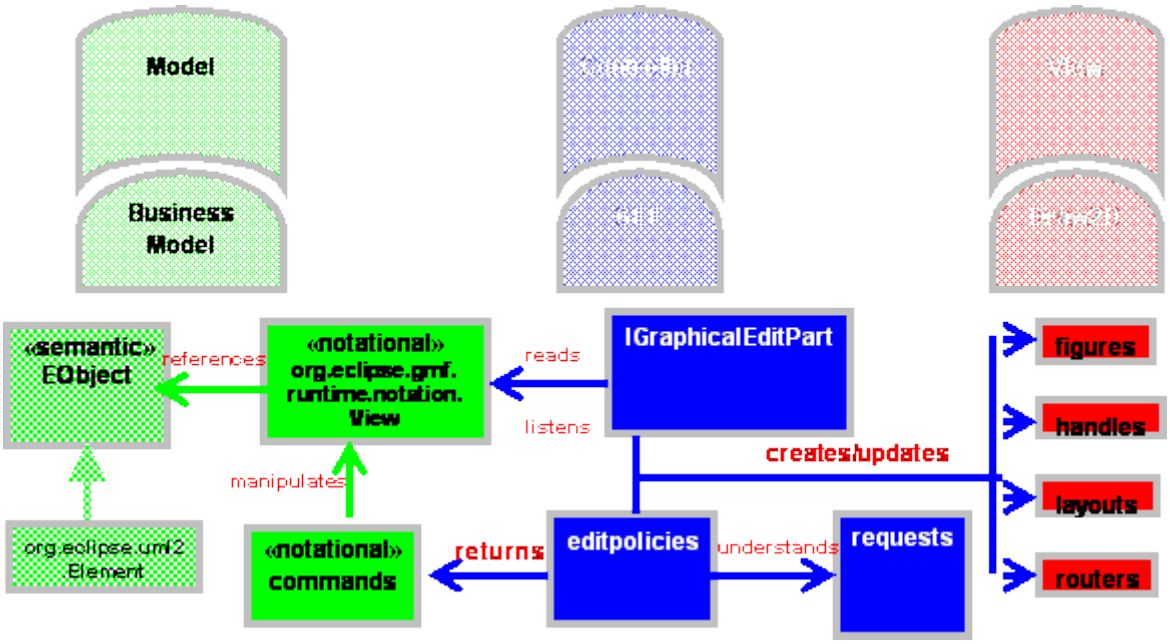


Das zweite Diagramm (s.o.) und das dritte Diagramm (s.u.), stellt den Ablauf nach dem Klick mit dem CreationTool auf der Diagrammzeichenfläche dar. Das obere stellt die Erzeugung des semantischen und des notatiellen Elements, das untere die Erzeugung des Editparts und der Figur dar, welche zu dem erzeugten Element gehören.
Beschreibung zum oberen Diagramm (erste Phase):
Da die Commands durch die Maus erzeugt wurden, führt das zusammengesetzte Kommando jedes Unterkommando seriell aus. Dabei wird das semantische Element als erstes erzeugt und dem Notational Creation Command übergeben, welches den ViewService aufruft, um zu dem gegebenen semantischen Element ein notatielles Element zu erzeugen. Nicht dargestellt in dem oberen Diagramm wird die Aktion, in welcher der ViewService nach einem zum Element passenden Provider unter allen registrierten Providern sucht. Dieser Provider gibt die ViewFactory zurück, welche das zugehörige notatielle View Element erstellt und initialisiert. Ergänzungen zu Diagramm: die CreationTool Klasse des Diagramms entspricht wieder der EClassCreationTool Klasse und der ViewService entspricht der EClassViewFactory.



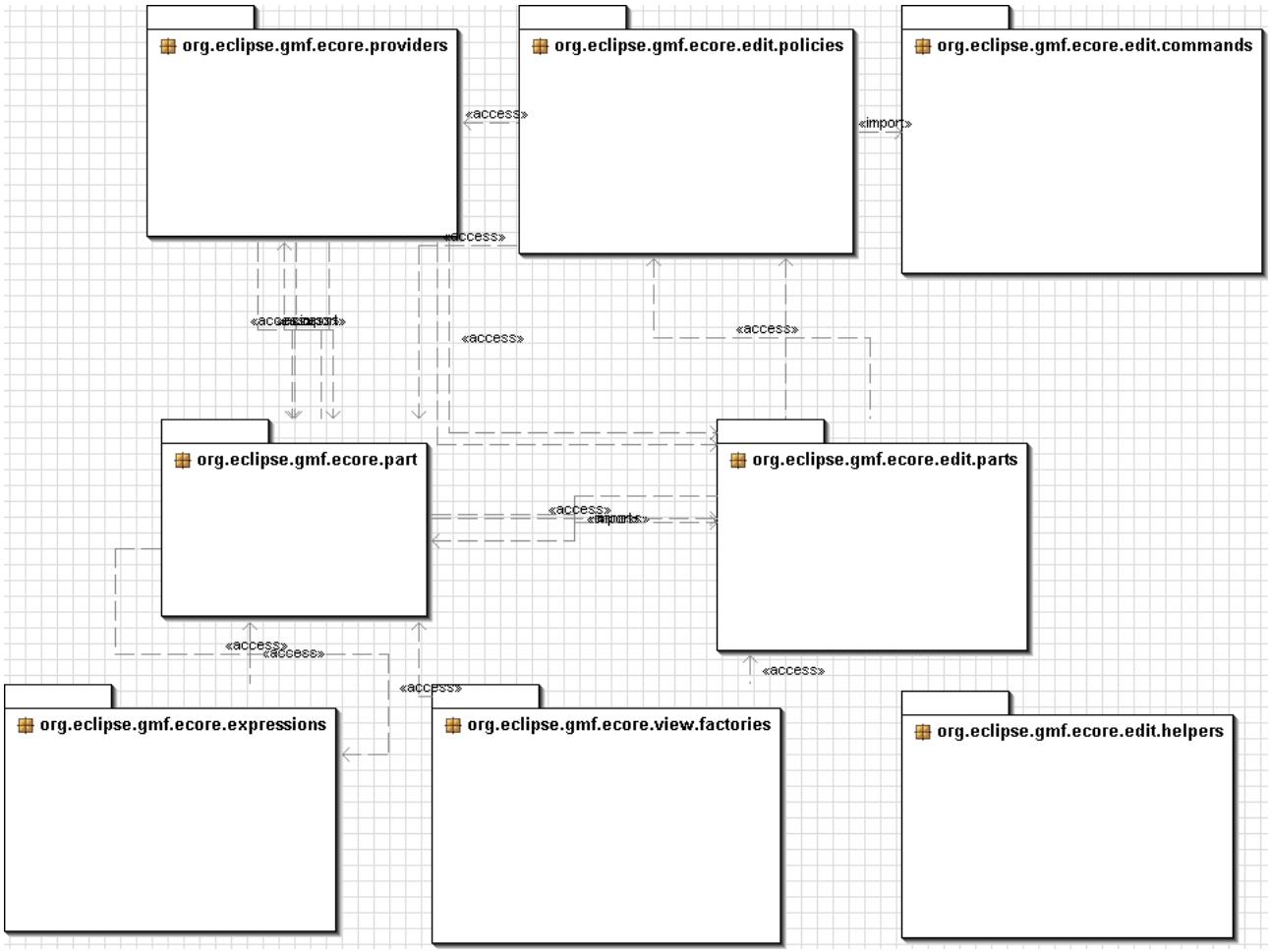
Beschreibung der zweiten Phase, der Erzeugung der EditParts und der graphischen Darstellungen: In dieser Phase wartet der EditPart des Containers, in der das semantische Element erzeugt wurde, auf die Benachrichtigung, das Element hinzuzufügen. Die Benachrichtigung wird ausgelöst, sobald der Schreibvorgang, der die Kommandoausführung umschließt, abgeschlossen ist. Alles innerhalb diese Schreibvorgangs wird registriert, sodass anschließend alles wieder rückgängig gemacht oder wiederholt werden kann. In dem oben abgebildeten Diagramm empfängt und sendet der DiagrammEditPart refreshCildren. Wenn die EditPartFactory benötigt wird, gibt GMF den EditPartService zurück, welcher dieses GEF Interface implementiert. Ähnlich dem ViewService, wird der EditPartService den zugehörigen Provider unter allen registrierten Providern finden, welcher wiederum die EditPart Klasse zurück gibt, die der EditPartService zu erzeugen hat. Ergänzungen zum Diagramm: die CreationTool Klasse des Diagramms entspricht wieder der EClassCreationTool Klasse, der DiagramEditPart entspricht dem EClassEditPart und der EditPartService entspricht der EditPartFactory Klasse.

3.2. Umsetzung der Modell-View-Controller-Architektur



Der Aufbau der MVC-Architektur in GMF ist wie in dem oben stehenden Bild dargestellt. Die unterste Schicht, hier durch „EObject“, „....view“, „commands“, „IGraphicalEditPart“, „editpolicies“, „requests“, sowie „figure“, „handles“, „layouts“ und „routers“ dargestellt, ist im Vergleich zu GEF deutlich komplexer. Dies ergründet sich aus der Tatsache, das GMF als eine Platform für verschiedene Domain Modell Editoren entwickelt wurde. Somit muss eine robuste, aber zugleich erweiterbare Lösung gefunden werden, die es Clients (andere GMF PlugIns) erlaubt, auf bereits bestehenden GMF PlugIns aufzubauen. Des weiteren sollten die Editoren selbst erweiterbar sein, durch die PlugIn-Architektur von Eclipse. Die Erweiterbarkeit von GMF erlaubt das Laden von Komponenten oder auch PlugIns zur Laufzeit unter Benutzung des Eclipse ExtensionPoints Mechanismus und der GMF Service Provider Infrastruktur.
Neben den Vorteilen der Architektur der Platform, existiert eine große Sammlung von Features, die Plugins zusätzlich in ihrem Editor nutzen oder direkt mit wenig Mehraufwand selber implementieren können. Ein besonderes Augenmerk wurde bei der Entwicklung auf die Usability der Editoren und der Services der Diagramm-Unterstützung geworfen, die als PopUps oder Verbindungsenden an den Objekten der Modelle erscheinen.
Der grüne Teil stellt das Modell (MVC), der blaue den Controller (MVC), der rote den View (MVC) dar.
Der Programmierer kann das Verhalten des Editors im wesentlichen durch die „editpolicies“ beeinflussen. Durch sie kann er bestehende Regeln/Funktionen der EditParts überschreiben, ohne den bestehenden, durch GMF erzeugten Code, manipulieren zu müssen. Er fängt die „Requests“ (die in der reinen MVC-Architektur dem Auslösen eines „ActionListeners“ entsprechen) in einer „EditPolicy“ durch ein „Command getCommand(Request req)“ ab und kann sich durch das holen des „org.eclipse.gmf.runtime.notation.View“'s, der auf das semantische Objekt referenziert, direkten Einfluss auf das Modell nehmen. Der Editor nutzt die ExtensionPontis von GMF in org.eclipse.gmf.runtime.diagram.core und org.eclipse.gmf.runtime.diagram.core.ui.

3.3. Strukturierung der Packages



Namespace: org.eclipse.gmf.ecore

edit.commands: Anhand der Oberklasse, von der die einzige, diesem Paket enthaltene Klasse erbt, gehe ich davon aus, das diese Paket die Undo/Redo Funktionalität auf EMF-Ebene implementiert.

edit.helpers: In diesem Paket sind die EditHelpers untergebracht, diese implementieren den DefaultEditCommand Algorithmus. Hier werden auch die Requests empfangen und die zugehörigen Commands zurückgegeben.

edit.parts: In diesem Paket werden die EditParts definiert, diese implementieren konkrete Funktionen zu Tools, installieren/initialisieren die DefaultEditPolicies, die Methoden zur Änderung des Labels, der Größe, sowie die Rückgabe der graphischen Darstellung.

edit.policies: In diesem Paket werden die Editpolicies eingefügt, diese ergänzen/überschreiben die bestehende Funktionalität der zugehörigen EditParts und werden auch von diesen aufgerufen.

expressions: In diesem Paket scheint eine Art Syntaxprüfung für das eCore Diagramm implementiert zu werden.

part: In diesem Package werden der Aufbau der Wizards, die Zusammensetzung der Toolbars und ähnliches definiert und die zugehörigen EditParts zu diesen hinzugefügt.

providers: In diesem Paket werden die Providers definiert, also Klassen, die für die Rückgabe der Klassen, welche eine konkrete Funktionalität implementieren, verantwortlich sind

view.factory: In diesem Paket werden die ViewFactorys definiert. In diesen wird das Aussehen und die Darstellung der einzelnen Objekte (EClasses, EPackages, ...) festgelegt.

3.4. Anbindung an die Eclipse IDE

Eclipse lässt sich über so genannte PlugIns erweitern. Diese PlugIns müssen direkt oder indirekt von org.eclipse.core.runtime.PlugIn abgeleitet sein. PlugIns stellen Erweiterungen zu einem bestehenden Programm bereit, ohne dabei in dessen Quelltext eingreifen zu müssen. Eclipse verwaltet diese PlugIns in speziellen Klassen und lädt diese PlugIns, sowie konfiguriert diese. Der eCore Editor implementiert die PlugIn-Schnittstelle in der Datei EcoreDiagramEditorPlugin im part-Package.
Der Editor nutzt eine Reihe von Extentionpoints, die die Anbindung an die Eclipse IDE darstellen. Dabei werden unter anderem Extetionpoints anderer PlugIns genutzt, zum Beispiel das PlugIn GMF, welches von Eclipse selbst, EMF, GEF, sowie EMF OCL, EMF Validation, EMF Query und EMF Transaction abhängt. Sämtliche genutzte, sowie zur Verfügung gestellte ExtensionPoints werden in der Plugin.xml definiert.

Für eine vollständige Liste aller Abhängigkeiten, verweise ich auf die Plugin.xml