2. RECHERCHEBERICHT

Um von vornherein über einige Begrifflichkeiten im Klaren zu sein, werden zunächst einige wichtige Begriffe klarer definiert und erläutert. Dies dient unter anderem dazu, eine einheitliche Basis zur Kommunikation untereinander zu schaffen und somit den Arbeitsfluss in seiner Effizienz zu erhöhen.

2.1 Begriffsklärung

2.1.1 GEF

Das Graphical Editor Framework GEF stellt ein Framework dar, über das graphische Editoren für Eclipse programmiert werden können. Das GEF ist nur für den grafischen Editor zuständig, die konkrete Funktionalität eines Tools muss an anderer Stelle programmiert sein (z.B. über EMF).

2.1.2 EMF

Hinter dem Eclipse Modeling Framework (EMF) verbirgt sich ein Framework, das zur Entwicklung von Tools für die Eclipse-IDE genutzt wird. EMF enthält zusätzlich eine Bibliothek zum Erstellen und Modifizieren von XML-Schemas (XSD) und für Service Data Objects (SDO).

2.1.3 Framework

Ein Framework ist eine Rahmenstruktur die mit dem Software-Paket und der Softwarekomponente in Beziehung steht. Es wird im Rahmen der objektorientierten Softwareentwicklung für komponentenbasierte Entwicklungsansätze verwendet. Das Framework gibt in der Regel eine Anwendungsarchitektur vor.

2.1.4 DSL

Eine Domain Specific Language ist eine Modellierungssprache, die im Gegensatz zu der sehr allgemeinen Unified Modeling Language (UML) auf ein spezielles Einsatzgebiet/Geschäftsfeld abgestimmt ist.

2.1.5 GMF

Das Graphical Modeling Framework (GMF) ist ein Framework, welches sich zum Ziel ge setzt hat funktionsfähige graphische, Eclipse-basierte Editoren für selbst definierte EMF basierte Metamodelle zu generieren.

2.1.6 Metamodell

Modelle, die beschreiben, wie Modelle gebaut werden nennt man Meta-Modelle. Ein Metamodell enthält Methoden, die Syntax und Semantik der Modelle definieren. Diese Methoden selbst sind wieder reale Methoden, deswegen liegt es nahe, diese Methoden auch durch Modelle zu beschreiben.

2.1.7 ECore

Ein Meta-MetaModell zum definieren von Meta-Modellen.

2.1.8 Eclass

Eine EClass ist eine Instanz einer Klasse aus dem Paket org.eclipse.emf.ecore EClass. Eine EClass ist damit das Gegenstück zu java.lang.Class.Mehr zu EClass siehe auch im Abschnitt 2.3 zu Ecore.

2.1.9 Service Data Objects (SDO)

SDO ist eine Spezifikation für ein herstellerunabhängiges Framework zum einheitlichen Datenzugriff, das eine gute Konnektivität zu XML bereitstellt. Es ermöglicht damit einen uniformen Datenzugriff auf heterogene Datenquellen.

2.1.10 IDE

Eine integrierte Entwicklunsumgebung(kurz IDE für integrated development environment) ist ein Anwendungsprogramm zur Entwicklung von Software.

Nachdem die wichtigsten Begriffe geklärt wurden und somit allen zugänglich sind, steht eine grundlegende Recherche zur Diskussion, die im Folgenden durch die Untersuchung verschiedener bereits existierender Programme näher charakterisiert wurde. Die zu untersuchenden Tools waren zum einen MetaEdit+, die DSL Tools (integriert in Visual Studio 2005) und zum anderen stand das Eclipse GMF (und GEF) im Mittelpunkt des Interesses und erfuhr besondere Aufmerksamkeit, da auf dieses Framework das Projekt aufgesetzt werden soll. Im nachfolgenden daher nun der zusammengefasste Recherchebericht.

2.2 Recherchebericht

Im Mittelpunkt der Betrachtung steht eine zentrale Analyse des Konzeptes zur Erstellung eines graphischen Editors, der auf das GMF für Eclipse aufsetzt.
Die Grundlage dieser Recherche bildet damit die Untersuchung bestehender Tools, sowie deren Herangehensweise an die Anforderungen des Benutzers – ebenso wichtig ist eine eventuelle Evaluierung bestehender Features zur Verwendung im eigenen Projekt, sodass der Benutzer den höchstmöglichen Profit bereits bestehender Anwendungskonzepte genießt.
Erste grundlegende Schritte konnten mit **MetaEdit+** von Metacase gemacht werden. Der Editor überwältigt den Benutzer zunächst stark mit Funktionalität. Der Einstieg ist recht schwer zu finden, da eine riesige Masse an Aktionen zur Verfügung steht, was die sehr guten Eindrücke während der äußerst einfachen Installation des Programms doch sehr trübt. Obgleich man sich zunächst erschlagen fühlt, fällt nach kurzer Einarbeitung die Übersichtlichkeit in der Gestaltung des Benutzerinterfaces auf, das die Unmengen an Funktionen klar in Bereiche, Browser und Assistenten unterteilt. Der damit verbundene Komfort bei der Erstellung der Metamodelle erleichtert den Umgang mit der Materie – eine fehlende Drag'n'Drop-

Funktionalität zum schnellen Umgestalten des Modells erschwert den Arbeitsfluss jedoch wieder. Im Großen und Ganzen kann man durchaus sagen, dass Metacase sich hier an die erfahrenen Benutzer wendet – umfangreiche Kenntnisse und Erfahrungen im Umgang mit Metamodellen erleichtern die Arbeit erheblich. Zwar sind innerhalb des Supports einige Tutorials gegeben, allerdings fordern sie dem Benutzer einiges an Einarbeitungszeit ab – MetaEdit+ ist somit für den Einsatz unter Profis geeignet, die bereits einiges an Erfahrung mitbringen – dann, und nur dann bietet es Komfort und Qualität sowohl in der Benutzung als auch im Ergebnis. Das Programm glänzt in den Händen des erfahrenen Benutzers vor allem durch die umfangreiche Funktionalität im Umgang mit Metamodellen, sowohl funktionell als auch hervorragend visualisiert.

Um ein klareres Bild des aktuellen Marktstatus zu bekommen, wurden im nächsten Schritt die **DSL-Tools von Microsoft** unter die Lupe genommen. Es handelt sich hierbei weniger um eine Standalone Aplication, als vielmehr um ein Plugin für das – gerade im professionellen Bereich stark genutzte Visual Studio 2005.

Mit Hilfe dieser Werkzeuge ist es möglich, in Visual Studio eigene domänenspezifische Sprachen zu erstellen. Diese Werkzeuge erlauben die Generierung jeder Art von Artefakten. Dazu zählen Klassendiagramme, Style Sheets, XML und HTML Dokumente oder auch jegliche andere Files. Es ist möglich, Frameworks und Architekturen abzubilden. Artefakte werden als Symbole auf der grafischen Oberfläche dargestellt. Im Mittelpunkt der DSL Tools steht die Oberfläche. Diese besticht durch eine weiche Zeichnung der Komponenten. So werden beispielsweise Klassen durch abgerundete Rechtecke dargestellt. Farbige Unterschiede tragen zur Übersichtlichkeit und Anschauung bei. Dem Anwender ist es möglich Komponenten einer Sprache über eine Toolbar auszuwählen und direkt auf die Oberfläche zeichnen zu lassen. Parallel können die Tools Komponenten aus Quellcode einlesen und gezeichnete Komponenten in Quellcode generieren lassen. Die dargestellten Komponenten sind statisch und so nur sehr schwer zu editieren.

Die lange Einarbeitungszeit und Komplexität erschwert den Umgang für kleine Projekte.

Als zentraler Gegenstand der Betrachtung muss das **Eclipse GMF** (+EMF & GEF) angesehen werden, da auf dieses das anstehende Softwareprojekt aufgesetzt werden wird. Das GMF setzt zunächst auf die IDE Eclipse auf. Das Framework selbst bietet dadurch eine Vielzahl von Klassen an, die zur Entwicklung graphischer Editoren für die Eclipse IDE verwendet werden können. Positiv hervorzuheben ist der Communitysupport, der über Tutorials und Newsgroups Erstanwendern sehr unter die Arme greift. Die Übersichtlichkeit ist nach mehreren Einschätzungen eher gehobenes Mittelmaß, die Dialoge sind übersichtlich gestaltet, das zu Beginn erscheinende Dashboard ebenfalls übersichtlich gegliedert – allerdings nimmt die Zahl der Fenster mit fortlaufender Zeit exponentiell zu. Die Bedienbarkeit bremst sich somit durch steigende Komplexität selbst. Gerade unerfahrene Benutzer kapitulieren hier sehr schnell. Der stark modulare Aufbau wird die Verwendung in kleineren Projekten mit geringerem Übersichtlichkeitsbedarf zwar erleichtern, spätestens bei umfangreicheren Projekten in Kombination mit weniger erfahreneren Anwendern wird das Ganze schnell zur Last. Die ebenfalls fehlende Intuitivität der Benutzung erschwert die Sache zusätzlich – dazu kommt die Tatsache, dass die massive Verwendung von Baumstrukturen zur Visualisierung und Verdeutlichung allein wohl nicht ausreicht um Projekte übersichtlich zu halten. Zusammenfassend kann wohl gesagt werden, dass die Frameworks für Eclipse vor allem für kleinere Projekte und Anwendergruppen bestechen – keine Kosten in der Anschaffung geben einen zusätzlichen Grund für die Verwendung im kleineren Maßstab. Die Verwendung in größeren Projekten setzt leider ein mit Eclipse und

den Frameworks erfahrenes Entwicklerteam voraus und selbst dann wird es schwierig den Überblick zu behalten. Lobenswert allerdings ist die nahtlose Integration und Interaktion mit der Eclipse DIE sowie anderen Plugins und Erweiterungen, wie die nahtlos Verwendung des CVS-Systems, das anderen Produkten fehlt.
Betrachtet man diese Produkte nun aus den Augen des angehenden Entwicklerteams unter Berücksichtigung der gestellten Aufgabe so wird schnell deutlich, welche Anforderungen an das eigene Projekt gestellt werden müssen.
So sollte unbedingt Wert auf die Intuitivität der Benutzung gelegt werden, die allen gängigen Produkten bisher fehlt. Der Benutzer sollte das Programm nicht bedienen müssen, sondern das Programm sollte seine Funktionalität so darlegen, dass es den Benutzer „bedient“ (umgekehrtes Abhängigkeitsverhältnis) – dies wird durch übersichtliche Gestaltung von Toolbars, die Verwendung von Assistenten und die offensichtliche Darlegung von Funktionen erreicht – Funktionalität darf niemals vor dem Benutzer versteckt werden. Ein weiteres wichtiges Kriterium ist die einfache Benutzbarkeit, die neben der Intuitivität eine zentrale Rolle spielt – gerade in Programmen zur graphischen Veranschaulichung von Sachverhalten sind Konzepte wie Drag'n'Drop unerlässlich um den Arbeitsfluss zu vereinfachen und zu beschleunigen. Das endgültige Produkt muss sich so stark in den Arbeitsfluss des Benutzers einfügen, dass dieser das Vorhandensein der Funktionen als gegeben ansieht. Die Verwendung des Pluginkonzeptes der Eclipse IDE ist damit greifend für alle Produktüberlegung, um die nahtlose Integration in eine Programmierumgebung zu gewährleisten.
Ebenfalls zu gewährleisten ist die Vereinheitlichung der Präsentation des Modells – die Verwendung einer (evtl. durch den Benutzer erweiterbaren) Standardkomponentenpalette die zur Modellierung verwendet wird.
Die grundlegenden Anwendungsfälle des Produktes schließen vor allem die Modellierung eines Metamodells durch den Benutzer ein, sowie dessen Präsentation und letztliche Umsetzung im Team, als Softwaremodell und als letztendlicher Objekt- bzw. Programmcode. Da die letzten Anwendungsfälle bereits durch die Eclipse IDE abgedeckt sind, werden die ersten beiden Anwendungsfälle im Mittelpunkt der Entwicklung stehen – die Modellierung selbst und die Präsentation selbiger.

Nachfolgend soll nun eine nähere Betrachtung der Rahmenapplikation folgen.

2.3 Nähere Betrachtung der Rahmenapplikation

Eclipse ist ein Open-Source-Framework das sich im Bereich der Softwareentwicklung immer mehr durchsetzt. Die bekannteste Verwendung ist die Nutzung als Entwicklungsumgebung (IDE) für Java.
Eclipse kann aber durch verschiedenste PlugIns für andere Sprachen und Entwicklungsaufgaben Verwendung finden und Entwicklungswerkzeuge für verschiedenste Inhalte integrieren. Weiterhin ermöglicht es die Entwicklung von Rich-Client-Applikationen auf Basis der Eclipse Rich Client Platform (RCP). Die Plattform basiert selbst auf Java Technologie und ist unter www.eclipse.org frei beziehbar.
Mit Eclipse ist es möglich Diagramme jeglicher Art einfach zu bearbeiten, dabei ist es flexibel genug um

einfache Veränderungen an Textdateien zu ermöglichen. Weiter ermöglicht es die automatische Quellcodegenerierung und die Weiterverarbeitung aller erzeugten Dateien. Durch seine offene PlugIn-Struktur ist es sehr einfach Eclipse zu erweitern. Die Eclipse Plattform ist

„an IDE for anything, but nothing in particular."

Im Folgenden beziehen wir uns auf nebenstehende Abbildung, die den konzeptionellen Aufbau von Eclipse bildhaft darstellt.

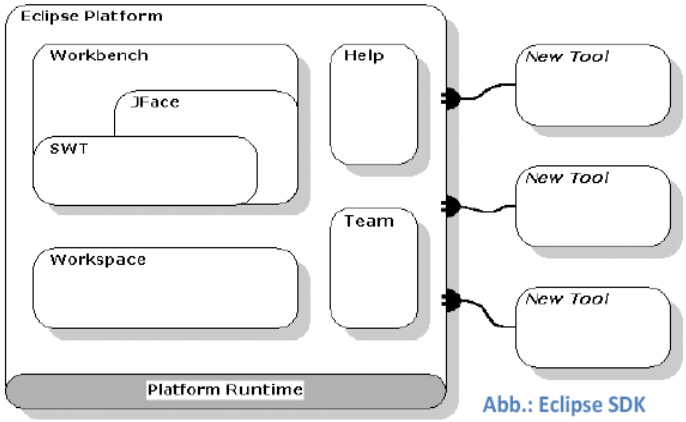


Abb.: Eclipse SDK

Eclipse arbeitet in einem Workspace, darin sind alle benötigten Dateien gespeichert. Es kann eine oder auch mehrere Instanzen von Eclipse gestartet werden, diese werden Workbench genannt. Alle Fenster innerhalb dieser Workbench sind mit der grafischen Bibliothek SWT(Standart Widget Toolkit) und dem darauf aufbauenden JFace erstellt. Daher müssen diese Bibliotheken auch beim erstellen eigener Oberflächen verwendet werden. Um grafische Modelle erstellen zu können, wird meist das EMF zusammen mit GEF benutzt. Zur Erleichterung stehen PDE oder auch JDT (Java Development Tooling) zur Verfügung. Eclipse selbst ist dabei in fünf Hauptbestandteile gegliedert.

Plattform Runtime

An sich besteht die Eclipse Plattform nur aus einem kleinen Kern – der so genannten Plattform Runtime. Diese stellt Mechanismen zum laden und speichern von PlugIns bereit und legt Regeln fest, wie diese kommunizieren und sich gegenseitig erweitern können. PlugIns selbst greifen über diesen Kern auf eine Fülle von Informationen zu, die sie zur Ausführung benötigen. Um ein schnelles Laden garantieren zu können werden nicht alle PlugIns automatisch gestartet. Allerdings werden ihre Metabeschreibungen eingelesen, um die Schnittstellen bekannt zu geben. Erst wenn die Funktionen des PlugIns benötigt werden, wird deren Code ausgeführt. Dieses Vorgehen vermeidet einen Overhead beim Laden und Ausführen der Umgebung.

Alle weiteren Teile von Eclipse sind als PlugIns konzipiert und bieten definierte Schnittstellen zur Erweiterung durch andere. Dennoch gehören sie direkt zur Plattform.

Workspace

Die Form zur Datenablage im Dateisystem des darunterliegenden Betriebssystems wird als Workspace bezeichnet. Dieses besteht aus einen oder mehreren Projekten, welche ein Verzeichnis im Dateisystem besitzen, und in einem standardisierten Workspace Root-Verzeichnis existiert. So sind alle im Projekt enthaltenen Dateien und Dokumente über das Workspace für Projekte zugänglich.

Workbench

Im oberen Abschnitt wurde eine Instanz von Eclipse als Workbench bezeichnet. Genauer stellt ein Workbench die Oberfläche von Eclipse dar. In diese werden die Oberflächenerweiterung der PlugIns hineingesetzt. Das Workbench liefert Schnittstellen zur Erweiterung dieser Oberfläche. Die Schnittstellen sind in zwei Teile gegliedert, welche das Erstellen von Oberflächenelementen ermöglichen. Die wichtigsten Erweiterungspunkte stellen dabei die Viewer und Editoren dar. Editoren ermöglichen das Öffnen, Ändern und Speichern von Daten. Viewer zeigen Informationen über die Objekte an, die momentan bearbeitet werden. Sie dienen unter anderem als Unterstützung für Editoren, um eine weitere Sicht auf die Daten zu haben.

Team

Um neue Team Repository Provider in die IDE zu integrieren bietet Eclipse eine einheitliche Schnittstelle. Damit sich Anwender nicht neu in jede Team Repository Software einarbeiten müssen gibt es so genannte Platzhalter für bestimmte Aktionen, Ansichten usw., die nur noch implementiert werden müssen. In der Eclipse Standardversion gibt es schon eine Unterstützung von CVS. Dabei kann auf ein CVS sowohl per psserver oder ssh Protokoll zugegriffen werden.

Help

Um zu jedem PlugIn eine strukturierte Hilfe anbieten zu können besitzt Eclipse ein integriertes Hilfekonzept. Diese wird dabei in zwei Bereiche geteilt. Zum ersten eine XML Struktur, die eine Navigation in der Hilfe selbst ermöglicht, zum zweiten die Hilfedateien in HTML Format. Dieses Konzept hat den Vorteil, dass sich schon bestehende Hilfen leichter in Eclipse integriert werden können. Die Navigation in der Hilfe ist durch die Baumansicht auch für den Anwender leicht verständlich und verwendbar. Um in der Oberfläche schnelle kontextabhängige Hilfe anbieten zu können, kann für jedes Element eine Kontexthilfe angeboten werden. Diese werden ebenfalls in einer XML Datei definiert. Da jedes PlugIn seine Hilfe in eine bestehende integrieren kann, ist sie leicht wart bar und erweiterbar.

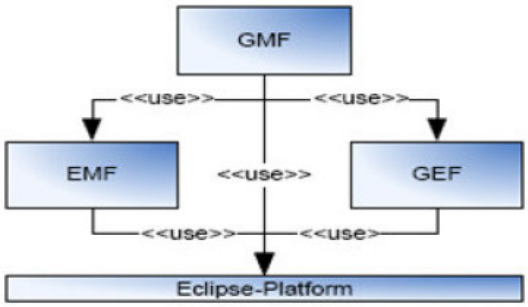
Im Weiteren gehen wir kurz auf das **PlugIn-Konzept** ein.

Ein PlugIn ist eine Einheit, die eine bestimmte Funktionalität der Umgebung hinzufügt. Es kann dabei gänzlich neue Funktionalität hinzufügen oder bereits Vorhandene erweitern.
Seit der ersten Version verfolgt Eclipse das Konzept einer reinen PlugIn Architektur. Das bedeutet, dass nicht nur bestimmte Funktionalitäten über PlugIns erweitert werden können, sondern dass praktisch jeder Teil von Eclipse ein PlugIn selbst ist. Dieser komponentenbasierte Ansatz bietet größtmögliche Flexibilität für den Ausbau der Plattform und die Erweiterung um einzelne Funktionen.
Seit Version 3 arbeitet Eclipse zur PlugIn Verwaltung auf der Basis von OSGi. Die Open Services Gateway Initiative (kurz OSGi) ist ein Industriekonsortium, das eine Softwareplattform für die Verwaltung von Diensten auf Komponentenbasis standardisiert. Dabei übernimmt die Eclipse Laufzeitumgebung die Dienste der OSGi-Server, genauer das PlugIn org.eclipse.osgi.
Eclipse wird als Software Development Kit (SDK) mit zwei PlugIns ausgeliefert. Dem Java Development Tooling (JDT), das Eclipse zur Java IDE erweitert und dem PlugIn Developer Environment (PDE), mit dem

man neue PlugIns entwickeln kann. Die Wurzel der PlugIn Architektur bildet das von der Laufzeitumgebung gelieferte PlugIn org.eclipse.core.runtime, die als halbes PlugIn auf keine Extension Points aufbaut, sondern sie nur bereitstellt. Dabei ist zwischen org.eclipse.osgi und org.eclipse.core.runtime zu unterscheiden, da PlugIns auf beide aufbauen können.
Ein PlugIn wird im PlugInverzeichnis unter einem eindeutigen Namen, der auch als Referenzierung dient, installiert. Jedes PlugIn besitzt dabei einen Erben der PlugInklasse von Eclipse (org.eclipse.core.runtime.Plugin). Diese PlugInklasse ist für die Konfiguration und Steuerung des PlugIns verantwortlich. Zusätzlich zur PlugInklasse gibt es die Datei plugin.xml, welche Metainformationen enthält, die Eclipse zum Starten des PlugIns benötigt. Durch diese Manifestdatei ist es Eclipse möglich das PlugIn zu laden und seine Informationen in der PlugInregistry verfügbar zu machen. PlugIns selbst werden über Extension Points verschachtelt. Dabei baut eine Menge von Erweiterungen (extensions) auf einen oder mehrere Extension Points auf. Diese Erweiterungen befinden sich auf derselben oder untergeordneten Ebene der PlugIn-Hierarchie, da durchaus auch
mehrere PlugIns einem Extension Point angeschlossen werden. PlugIns lassen sich über die OSGi Registry auch wieder entladen, so lassen sich Anwendungen auf eine flexible Weise mit PlugIn Funktionalität ausstatten.

In die Eclipse IDE wird das GMF integriert, mit dem schlussendlich gearbeitet wird.

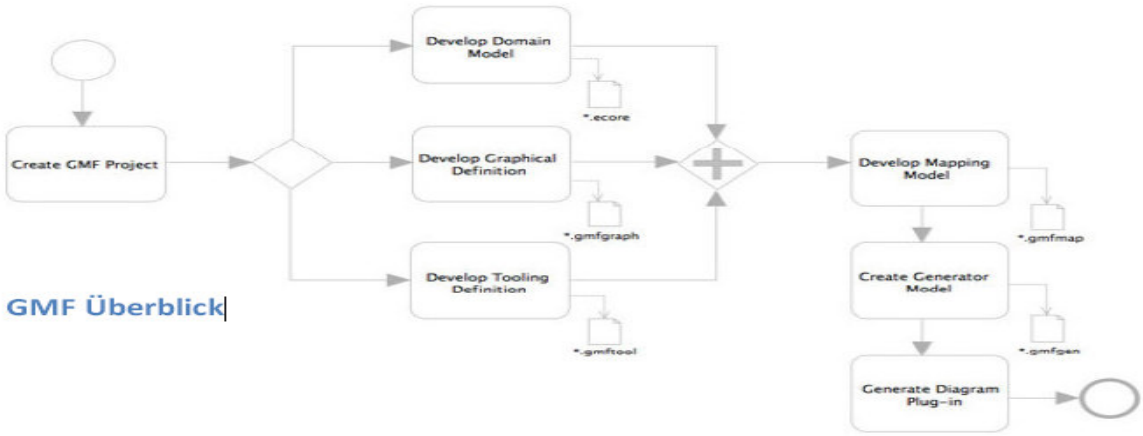
Eclipse GMF ist ein frei verfügbares Open-Source Werkzeug zur Metamodellierung. GMF kombiniert dabei EMF und GEF durch seinen modellgetriebenen Entwicklungsansatz. Dafür stellt GMF einen Baum basierten EMF Editor und Importfunktionalitäten für externe Modellierungswerkzeuge bereit. Als Ergebnis werden grafische Editoren als Metamodell generiert. Beispiele dafür wären unteranderen UML Diagramme, Editoren zur Abbildung oder auch Flussdiagramme. Typischerweise wird GMF aber benutzt, um eine zugeschnittene Modellierungssprache zu entwerfen.



Beziehungen zwischen den Frameworks

GMF kann in zwei Teile unterteilt werden. Einem Tooling Teil und einen Runtime Teil. Beim Tooling handelt es sich um Editoren, die zur Beschreibung eines Editors selbst genutzt werden, sowie einen Generator, welcher aus der Editor Definition einen lauffähigen Editor erstellt. Die Runtime stellt Funktionalitäten wie Oberklassen für Figuren, Layout Klassen und Layout Algorithmen zur Verfügung. Im Wesentlichen benutzt GMF zwei weitere Eclipse basierte Frameworks. Das EMF und das GEF. Das EMF stellt ein Meta-Metamodell namens Ecore zur Verfügung, mit dem man Metamodelle selbst erstellen kann. GEF hingegen ist ein Framework aus denen grafische Editoren erstellt werden. Dabei folgt es dem MVC Entwurfsmuster.

Diese Editoren können genutzt werden, um einen Graphen grafisch darzustellen und zu editieren oder erweitern. Die Aufgabe von GMF ist nun ein mit EMF Mitteln beschriebenes Metamodell auf einen grafischen Editor abzubilden. Dabei nutzt GMF drei Teile. Eines zur Beschreibung der Figuren, ein zweites für die Palette und ein drittes um beides zu verbinden.



Auf Grundlage des Models werden grafische Editoren definiert. Dieser bildet einen Teilbereich der Syntax ab und ermöglicht dessen Bearbeitung. In Eclipse erfolgt die Definition eines grafischen Editors durch folgende Modelle: **gmfgraph**, **gmftool** und **gmfmap**.

Das ***gmfgraph*** Modell beschreibt die grafischen Formen für den Editor. Dafür stehen *Elipse*, *Rectangle*, *Rounded Rectangle*, *Polygon*, *Line*, *Polyline* und verschiedene visuelle und topologische Gestaltungsmittel zur Verfügung. Man unterteilt dabei diese Elemente (*Figure*) und die zu zeichnenden Entitäten, die später im gmfmap referenziert werden. Jede Standartform kann durch Attribute wie Vorder- und Hintergrundfarbe oder Größe angepasst werden. Layout Klassen (Bsp.: *BorderLayout*) übernehmen die Anordnung. Zeichnende Entitäten werden in *Node*, *Connection*, *Compartment* und *Labels* unterschieden. *Nodes* sind Knoten die über Connections verbunden werden. *Label* wiederrum sind Beschriftungen im Diagramm. *Compartments* definieren Knoten, die innerhalb von anderen Knoten liegen können. Beschriftungen werden als Labels bezeichnet.

Im ***gmftool*** Modell wird eine Werkzeugleiste für den Editor beschrieben. Dabei können selbst definierte Symbole(*Image*) zur Hervorhebung verwendet werden. Jedes Tool besteht neben einer Tool Beschriftung und einen *Tooltip* aus der Angabe zweier Icons. Diese werden für unterschiedliche Auflösungen verwendet und erscheinen in der *Toolbar*. Elemente der Toolbar werden in einer *ToolGroup* gruppiert. *ToolGroupLinks* verbinden diese *ToolGroupElements,* welche in *ToolContainer* angeordnet werden. Elemente werden in einer *ToolRegistry* angemeldet. Zur Übersichtlichkeit stehen auch Menüs zur Verfügung – *Menu* und *PopUpMenu* – wie auch im Java eigenen awt oder swing. Weiter können Farbe in einer *Palette* angegeben werden, dies übernimmt die Klasse *PaletteImpl*, eine Implementierung von Palette.

Um Elemente aus dem gmfgraph und gmftool miteinander zu verbinden wird das Modell ***gmfmap*** benutzt. Gleichzeitig werden diese Elemente auch mit denen des Metamodells verbunden. Damit werden Zuordnungen von Elementen der Werkzeugliste zu grafischen Modellierungselementen definiert und Beziehungen zu Instanzen des Modells festgelegt. Außerdem werden Vorbelegungen von Modellelementen und deren Attributen auf Basis von konkreten Inhalten des Modells definiert. Im Mapping unterscheidet man zum einen die Knoten direkt, die auf der Zeichenfläche erscheinen sollen (*Top Node Reference*). Und zum anderen die, die als *Link* realisiert werden sollen. *Nodes* verknüpfen dabei intern einen Knoten aus der grafischen Repräsentation, ein Erzeugungs-Tool und eine Entität (Eclass) aus dem Modell. *Links* verknüpfen ebenfalls eine grafische Darstellung, ein Erzeugungs-Tools (bei Assoziations-Klassen) und eine Entität. Neben *Compartments* werden auch die zu setzendenden Attribute der Entität spezifiziert. *Nodes* und *Links* geben als *Containment* an, als Attribut welcher Modell-Entität sie erzeugt werden.

Ecore ist ein Meta-Metamodell, das von EMF zur Verfügung gestellt wird. Mit so einem Meta-Metamodell kann man Metamodelle selbst erstellen, aus deren Elemente wieder die Modelle selbst entstehen. Ecore ist der von der Eclipse Community unterstützte Standard. Die Metasprache Ecore wurde gewählt, da sie die Umsetzung von Mechanismen zur Abstraktion direkt unterstützt. Im Mittelpunkt steht hier *EClass*. *EClasses* werden von *org.eclipse.emf.ecore.EObject* abgeleitet und müssen als Instanz von *EClass* vorliegen. Diese Klasse beschreibt wie *EClasses* aufgebaut sind, enthält wiederum *EAttribute* und *EMethods*. Eine Instanz von *EAttribute* stellt ein modelliertes Attribut dar, dessen Datentyp in *EDataType* spezifiziert wurde. Eine *EReference* stellt dabei Verbindungen und *Assozationen* zwischen den *EClasses* dar. Um Änderungen am Modell mitzuteilen werden *Observer* über die Schnittstellen *Notifier* und *Adapter* realisiert. Bei Äderungen schicken die *Notifier* Objekte an angemeldete *Adapter*. Als Besonderheit kennen die *Adapter* hier das von ihnen beobachtete *Target*. Zum Schluss noch eine Klassenübersicht über die besprochenen *EClasses*.