Recherchebericht

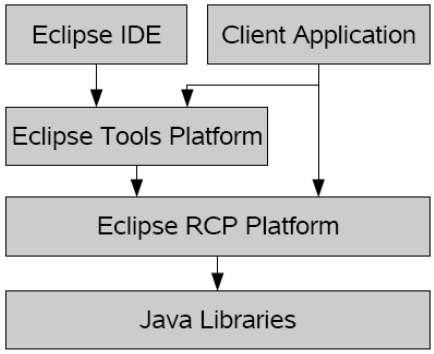
1.Allgemein

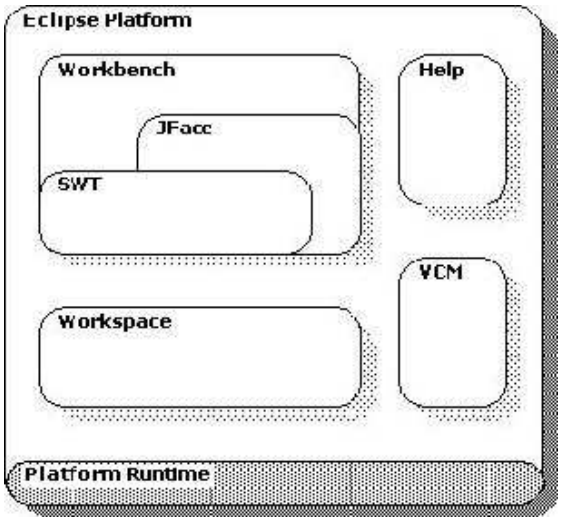
1.1.Konzeptioneller Aufbau von Eclipse, GMF und Plugins

Eclipse:

Eclipse stellt eine Framework für integrierte Entwicklungsumgebungen (IDE) bereit. Das „Eclipse Project“ stellt eine Plattform dar, die für sich genommen noch keine Entwicklungs-Funktionalität implementiert.
Erst zusätzliche Werkzeuge ermöglichen eine Software-Entwicklung mit der Eclipse-Plattform. Diese Tools werden in Form von Plug-ins in die Plattform integriert. Einige Tools (Java, C/C++, Plug-in-Entwicklung) können bei eclipse.org heruntergeladen werden

Aufbau:





Platform Runtime:
Grundlegende Funktionalität, steuert den Kontrollfluß

Workspace: Verwaltung der Ressourcen.
Über ihn wird auf die verwalteten Ressourcen zugegriffen Die Ressourcen werden im Dateisystem abgelegt, um auch Nicht-Eclipse-Software den Zugriff zu ermöglichen

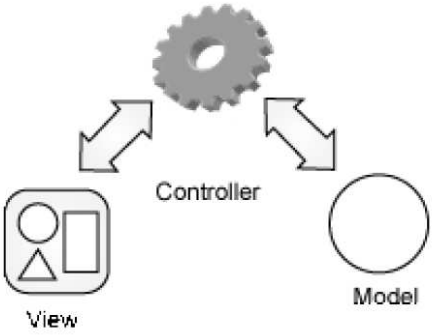
VCM: Versions-und konfigurationsmanagement

Help: HTML-Hilfe-System

Workbench: Eigentliches Benutzerinterface (mit dazugehörigen Hilfskomponenten)

1.1.2. GMF:

Model-View-Controller
Trennung von abstrakter und konkreter Syntax sowohl in Editordesign als auch im generierten Editor.
Hier steht der Controller als „Brücke“ zwischen View und Modell .

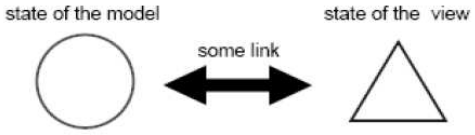


-Modell:
Datentragende Objekte (abstrakte/ konkrete Syntax)

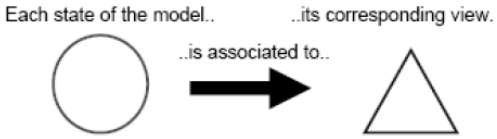
-Views:
Ansichten auf das Modell , zb: Graphischer Editor und Baumansicht des Modells

-Controller:
Verbindung zwischen Modell und View mit der Änderungen kontrolliert werden können.

Typische Model-View-Architektur:

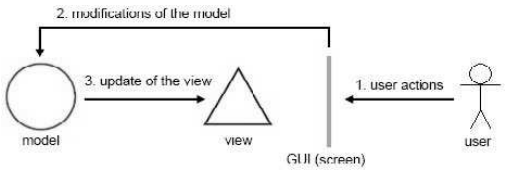


Modell und View sind ***unabhängig*** voneinander. Wenn das Modell geändert wird, muss der View aktualisiert werden, wärend umgekehrt Änderungen im View auch eine Änderung im Modell nötig machen kann.



View ist nur ***abhängig*** vom Modell und ist damit für jedes Modell eindeutig bestimmt
(=> Modell in konkreter Syntax)
Änderungen werden nur am Modell vorgenommen
Änderungen am Modell bewirken eine Aktualisierung der Ansichten

Einheitliches Schema für Editieroperationen:



Aktionen des Anwenders lösen Änderungen im Modell aus
Änderungen im Modell lösen Aktualisierung der Ansichten aus (dafür ist aber nicht das Modell zuständig!)

1.1.3. das Pluginkonzept:

Ein Plugin wird als Komponente begriffen, die einen bestimmten Dienst zur Verfügung stellt. Plugins müssen beim Start von eclipse zur Verfügung stehen. Jedes Plugin ist als Exemplar einer Plugin -Klasse vorhanden, die für die Konfiguration und die Steuerung des Plugins sorgt. Die Klasse org.eclipse.core.runtime.Plugin, von der alle Plugin-Klassen abgeleitet sein müssen, legt diese Funktionalität fest. Plugin-Management verwendet zwei Methoden dieser Klasse (startup und shutdown) , um den Lebenszyklus des Plugin zu steuern.Zusätzlich zu dem ausführbaren Code enthält ein Plugin Metainformationen, mit denen eclipse das Plugin verwaltet. Dazu trägt die Verwaltung die relevanten Daten in eine Plugin-Registry ein. Die Informationen liegen als Manifest genannte XML-Datei vor.

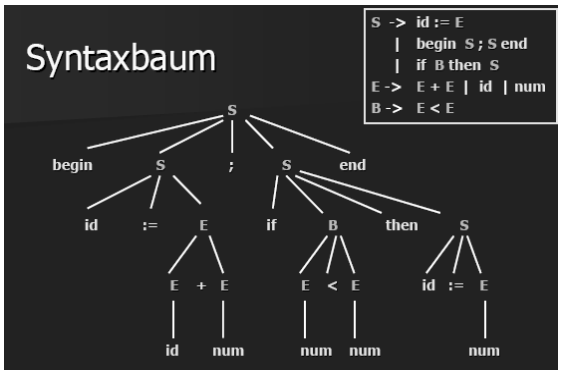
Plugin-Beschreibung, in der die Minimalinformation enthalten ist:

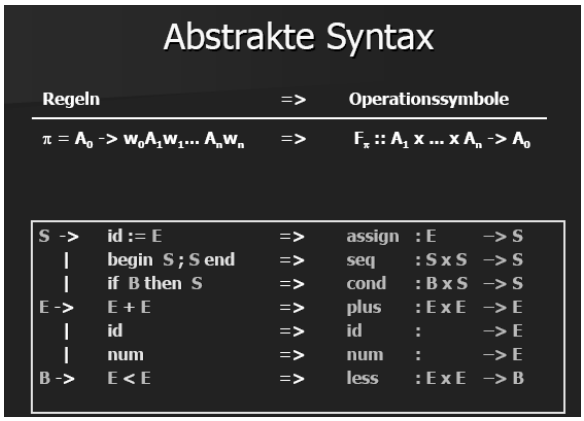
```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
  name="Plugin Name"
 id="de.plugin.name"
 version="1.0"
 provider-name="MyProvider.de">
 <runtime>
    <library name="codelocation.jar">
 <export name="*"/>
 </library>
 </runtime>
</plugin>
```

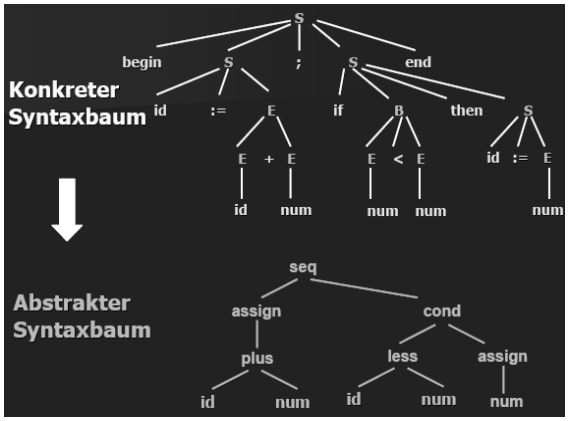
1.2.Syntax

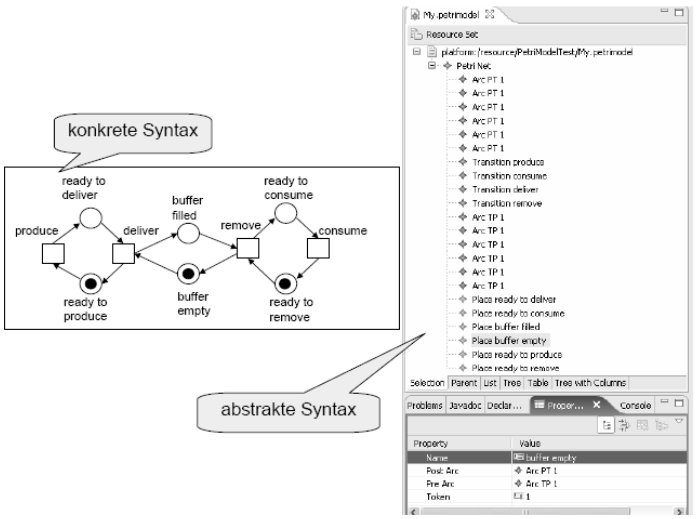
Solange man sich nur als Anwender mit einer Sprache beschäftigt kommt man nur mit der konkreten Syntax in Kontakt. Das ist die Syntax, die im jeweiligen Sprachreport definiert wird. Im Compilerbau gibt es darüber hinaus den Begriff der abstrakten Syntax. Darunter versteht man die interne Darstellung von Strukturen, die in der konkreten Syntax definiert wurden. Je nach Implementierung wird die abstrakte Syntax von einem Interpreter ausgeführt oder von einem Compiler weiterverarbeitet. Die abstrakte Syntax definiert also Elemente die miteinander in Verbindung gesetzt werden. Während die konkrete Syntax das Layout beschreibt.

Im allgemeinen kann man die konkrete Syntax als Syntaxbaum, gemäß der die Sprache definierenden kontextfreien Grammatik oder auch als benutzerfreundliche Mixfixnotation unter Verwendung natürlicher Sprache für Terminalsymbole sehen. Wärend die abstrakte Syntax als darstellungsunabhängige algebraische Struktur, welche im Programmwort auftretende Konstrukte und Schachtelbeziehungen identifiziert, anzusehen ist. Die abstrakte Syntax kann bzw. wird für weitere Phasen der Compilierung, wie semantische Analyse und Codegenerierung benutzt.









Beispiel Petrinetz

Diagrammdefinition

- abstrakte Syntax: interne Struktur des Diagramms
- konkrete Syntax: Layoutinformationen zum Diagramm

Konkrete Syntax:

- sprachspezifisch: Layoutinformationen in allen Diagrammen gleich
- modellspezifisch: individuelle Layoutinformationen für ein Modell

Layoutinformationen:

- geometrische Formen, Farben, Größen, Positionen, Textfont, Textgröße,⋯
- graphische Constraints: über, unter, innerhalb, ⋯

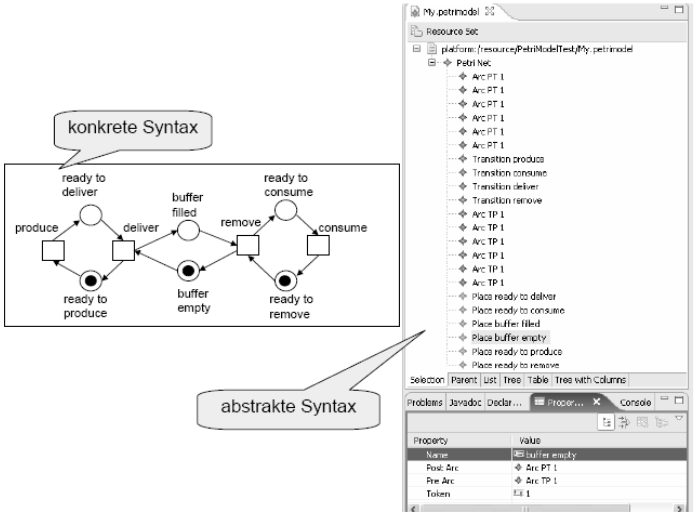
2. Genauere Beschreibung der Aplikationen

2.1.Meta-Edit

MetaEdit+

Begriffsübersicht:
Method Workbench - zur Definition von Modellierungsmethoden/-sprache bestehend aus Konzepten, Regeln, Notationen und Codegeneratoren
Object Repository - Eine Datenbank zur Speicherung von, mit dem Method Workbench erstellten, Methodendefinitionen sowie zusätzlicher Informationen
Bindings - Regeln zur Beschreibung von Beziehungen zwischen den Objekten
Constraints - Regeln zur Einschränkung von Beziehungen zwischen den Objekten
MetaEngine - eine Schnittstelle zwischen den Modulen und dem Object Repository

Syntax:
Beschreibung der abstrakten und konkrete Syntax , und die Verbindung zwischen beiden.
Die abstrakte Syntax wird definiert durch Elemente die miteinander in Verbindung gesetzt werden, während die konkrete Syntax das Layout beschreibt.



Beispiel Petrinetz

Diagrammdefinition
- abstrakte Syntax: interne Struktur des Diagramms
- konkrete Syntax: Layoutinformationen zum Diagramm

Konkrete Syntax:
- sprachspezifisch: Layoutinformationen in allen Diagrammen gleich
- modellspezifisch: individuelle Layoutinformationen für ein Modell

Layoutinformationen:
- geometrische Formen, Farben, Größen, Positionen, Textfont, Textgröße,⋯
- graphische Constraints: über, unter, innerhalb, ⋯

Der Editor von Metacase besteht aus zwei Teilen, dem Method Workbench und MetaEdit+ als CASE-Tool. Mit dem Method Workbench definiert ein Domänenexperte die Modellierungsmethoden/-sprache bestehend aus Konzepten, Regeln, Notationen und Codegeneratoren. Aus den erzeugten Methodendefinitionen wird dann automatisch ein CASE-Tool erstellt und MetaEdit+ stellt passende Werkzeuge bereit, wie z.B. Diagrammeditor, Generatoren und Browser.

Architektur
Method Workbench (Experte)
Metamodellierungssprache und Tool-Suite
Definition von Methoden Konzepte, Regeln, Notationen und Design von Generatoren
Methodendefinition wird als Metamodell im MetaEdit+-Repository gespeichert.

MetaEdit+ (Entwickler)
Ausgehend von der Methodendefinition automatische Generierung von CASE-Tool-Funktionalitäten :
Diagramm-Editoren,
Browser,
Generatoren,
Unterstützung mehrerer Benutzer (Multi-User)

Im Object Repository, einer Datenbank, werden die vom Experten mit dem Method Workbench erstellten Methodendefinitionen sowie zusätzliche Informationen getrennt nach Projekten gespeichert. MetaEdit+ wendet die Methoden automatisch an, indem es diese aus dem Repository abfragt. Wichtigster Bestandteil von MetaEdit+ ist hierbei die MetaEngine, die eine Schnittstelle zwischen den Modulen und dem Object Repository darstellt, welches wiederum die Konsistenz zwischen den einzelnen Tools gewährleistet. Die Module lassen sich in vier Gruppen einteilen: Modellbearbeitungswerkzeuge (Diagrammeditor, Matrixeditor, Tabelleneditor), Such- und Abfragewerkzeuge (Browser, Report und Codegeneratoren) , Werkzeuge für Methodenmanagement (Method Tool, API & XML connectivity). Hinzu kommen noch Tools zum Methodenmanagement in Gestalt des Method Workbench (z.B. Object Tool, Graph Tool, Symboleditor).

Method Workbench
Die domänenspezifische Modellierung mit MetaEdit+ erfolgt mittels des Method Workbench und umfasst die folgenden Schritte:

1 Definition von Methodenkonzepten mit Hilfe der vorgegebenen Formulare (GOPPRR)
In Abhängigkeit von der Anwendungsdomäne können die Konzepte aus folgenden Bereiche stammen:
- gewünschte Ausgaben, die generiert werden sollen,
- Produktkomponenten, die benutzt werden,
- Produktarchitektur,
- Produktlinien-Charakteristika usw.

ME-Formulare: Object Tool, Property Tool, Relationship Tool, Role Tool, Port Tool, Graph Tool

2 Definition von Regeln
Regeln darüber, wie Konzepte identifiziert und miteinander verbunden werden. Es gibt zwei Möglichkeiten: Bindings und Constraint. Bindings beschreiben die Beziehungen zwischen den Objekten und Constraints die Einschränkungen dieser Beziehungen. Regeln können zu einem späteren Zeitpunkt noch verändert werden, da MetaEdit+ ein Modell-Updates durchführt

3 Zeichnen von Symbolen
Mit Hilfe des Symboleditors lassen sich Objekte grafisch darstellen. Die so erzeugte grafische Repräsentation wird später im Diagramm- und Matrixeditor verwendet. Unterschieden wird zwischen statischen und dynamischen Elementen. Dynamische Elemente werden nur unter

bestimmten Voraussetzungen angezeigt, statische hingegen immer. Häufig verwendet Symbole oder auch Teile von Symbolen können in der Symbolbibliothek gespeichert werden, um sie später wiederzuverwenden.

4 Erstellung von Generatoren

Mit dem Report Browser lassen sich Reports erstellen, die die Modellkonsistenz prüfen, Modellverknüpfungen analysieren, Wörterbücher (Data Dictionaries) und Dokumentation produzieren, Code und Konfigurationsinformationen erzeugen oder Modelle an andere Programme exportieren wie Simulatoren oder Programme zur Versionierung. Mit einer eigene Skriptsprache ist es möglich Designs in unterschiedlichen Formaten zu zeichnen/drucken, generierte Ausgaben in verschiedenen Dateien (und verschiedenen Formaten) zu speichern, externe Programme aufzurufen und der Zugriff auf die gesamte Leistungsstärke vorhandener Betriebssystem-Befehle.

MetaEdit+ als CASE-Tool

Für den Einsatz als CASE-Tool bietet MetaEdit+ eine Reihe an Werkzeugen (Diagram Editor, Matrix Editor, Table Editor, Browsers, Report and Code Generation, API & XML connectivity), die über das Object Repository auf die erstellten Methodenkonzepte zugreifen können. Das Repository sorgt für konsistente Daten, so dass eine Veränderungen von Methoden möglich ist, während die Modelle gleichzeitig aktualisiert werden. Neben Unterstützung des Zugriffs mehrerer Benutzer findet eine Sperrung(Lock) auf feingranularer Ebene statt; es werden einzelne Objekte gesperrt statt des gesamten Graphen.

2.2. GMF-tools

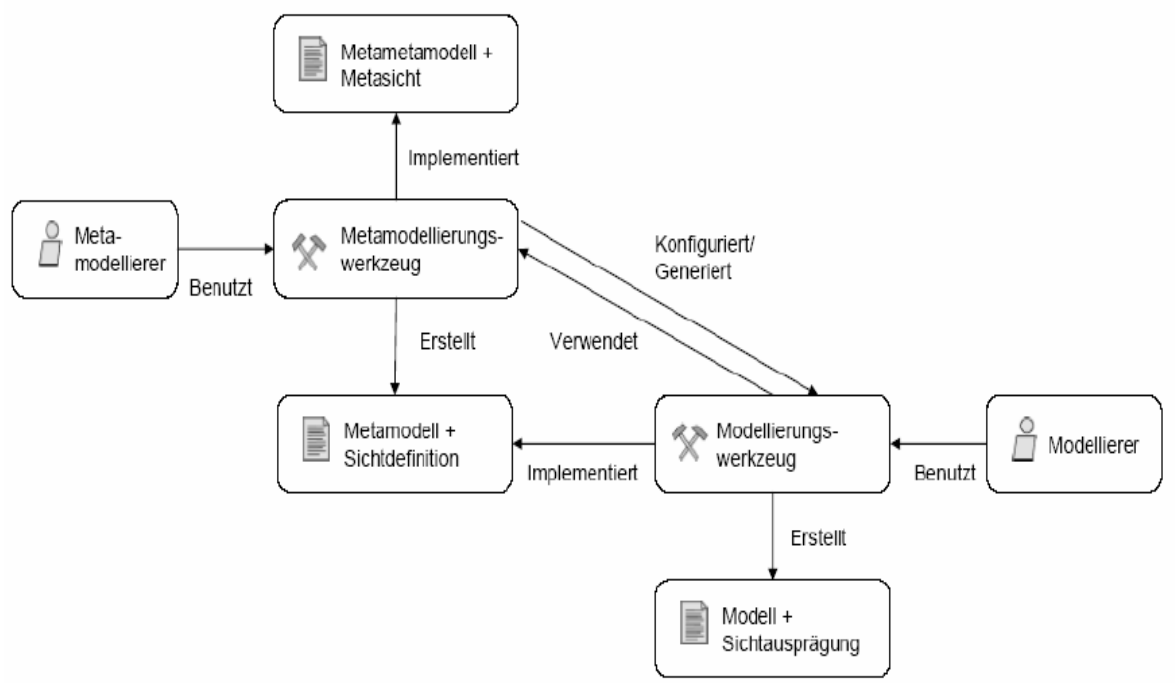
Graphical Modeling Framework , kurz GMF versteht sich als Brücke zwischen EMF und GEF und setzt genau bei der aufwendigen Editorerstellung an. Es erzeugt mit wenigen Handgriffen einen kompletten, lauffähigen grafischen Editor, aufbauend auf einem EMF-Modell, ohne dass auch nur eine Zeile Java-Code geschrieben werden muss. GMF lässt sich unterteilen in eine Runtime und ein Paket von Entwicklungswerkzeugen. Das Werkzeugpaket besteht dabei im Wesentlichen aus Editoren, mit denen die Eigenschaften des späteren grafischen Editors angepasst werden können, und aus Generatoren, welche den Features der Runtime Leben einhauchen. Die Runtime selber stellt viele Funktionen zur Verfügung, die aus kommerziellen grafischen Editoren bekannt sind und somit nicht von neuem entwickelt werden müssen. GMF dient somit also dem erleichterten Arbeiten mit EMF und GEF.

-Graphical Modeling Framework (GMF) ist ein Werkzeug zur generativen Erstellung von metamodellbasierenden Editoren.
-Der Editor wird als Plug In generiert.
-Mapping verbindet eigenständige teile

Leistungmerkmale von GMF:

-Elemente zeichnen
-Elemente verbinden
-Eigenschaften ändern (Possition ,Größe, Attribute)
-Elemente/Verbindungen löschen
-Baumstrucktur ahnliche Ansicht
-Relationen zwischen den Elementen darstellen
-Öffnen / speichern von Diagrammen

UseCase:



EMF:

-Das Eclipse Modeling Framework (EMF) ist ein Open-Source Java-Framework zur automatisierten Erzeugung von Quelltext anhand von strukturierten Modellen, basierend auf offenen Standards. Es ist ein Projekt der Eclipse Open Source Community.

-EMF kann aus einem Modell Java-Code erzeugen, Instanzen dieses Modells erstellen, abfragen, manipulieren, serialisieren (eingebaut als XMI oder anderes XML, mit Plugin auch in einer relationalen DB), validieren und auf Änderungen überwachen (für MVC). Darüber hinaus wird JUnit-Code erzeugt, der den generierten Code testet.

-Das Modell selbst kann aus einer XSD (wie etwa bei JAXB), aus annotierten Java-Interfaces oder aus UML-Diagrammen (Rose, Magic Draw und Omondo) generiert werden, oder auch von Hand (mit einem "Baumeditor") erstellt werden.

-Der aus dem Modell generierte Code umfasst den eigentlichen Modell-Code (wie ihn etwa JAXB erzeugt), Code für Wizards, Editoren, bis hin zum Code für die eigentliche RCP-Anwendung.

-Das Modell selbst, die Generierung daraus sowie der generierte Code können angepasst werden, implementierte Funktionalität und neu generierter Code werden dabei gemerged

Für weitergehende Ansprüche bietet EMF etwa die Möglichkeit, Modelle dynamisch zur Laufzeit zu generieren (etwa wenn erst dann das Modell bekannt ist).

GEF:

(arbeitet nach dem Model-View-Controller (siehe Konzepte.))

Das Graphical Editing Framework (GEF) dient der Erstellung von grafischen Ansichten für Modelle. Die angebotene Funktionalität erstreckt sich dabei einerseits über Unterstützung zur Verwaltung und Manipulation der EMF-Modellobjekte, als auch über zahlreiche grafische Hilfsmittel zur Erstellung eines Editors. Eine ausführliche Abhandlung von GEF liegt nicht im Rahmen dieser Arbeit, jedoch ist ein

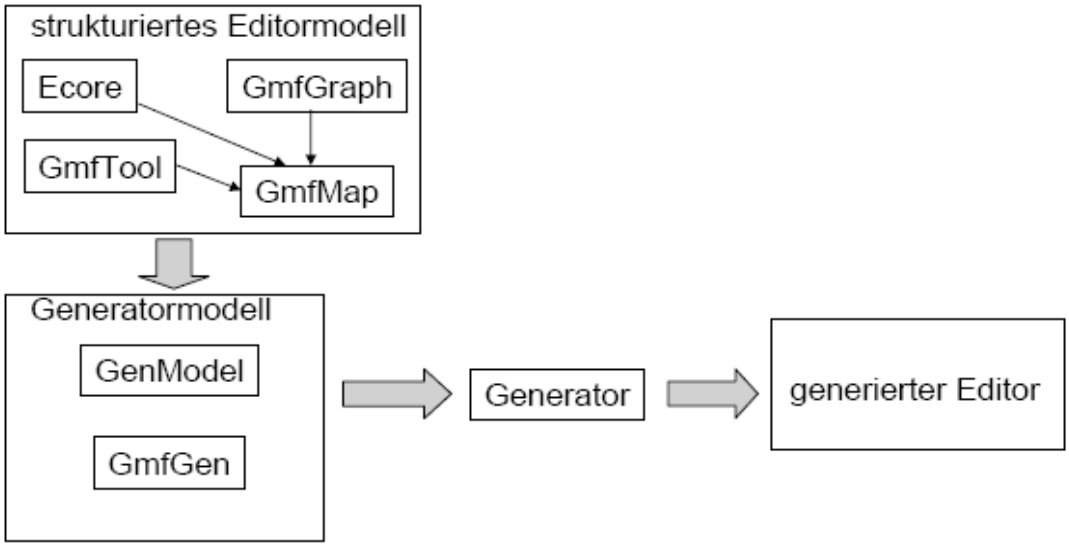
grundlegendes Verständnis der Arbeitsweise von GEF für das danach vorgestellte Graphical Modeling Framework (siehe 4.4), das wiederum auf GEF basiert, sehr von Vorteil, insbesondere, wenn man dort in den generierten Code eingreifen möchte, um die grafische Notation über die durch den Generator gebotenen Möglichkeiten hinaus zu verfeinern.

Draw2D

GEF basiert auf einem Framework namens Draw2D, das mit GEF als eigenständiges Plug-In

org.eclipse.draw2d ausgeliefert wird, und eine Bibliothek zum Erstellen von 2D-Grafiken darstellt. Draw2D ist ein so genanntes lightweight graphical system und unterscheidet sich damit von einem konventionellen Fenstersystem in einigen wichtigen Punkten:

eCore-Modell
Struktureller Aufbau



GMF-Graf: Figuren definiern

-eclass
- -Rectangle: erzeugt ein Rechteck
- -rounded Rectangle : erzeugt ein abgerundetes Rechteck
- -Ellipse : erzeugt eine Ellipse
- -Polygon : erzeugt ein Polygon
- -Lable : erzeugt ein Label

GMF-Tool: erzeugen der Werkzeuge für Palette

-eclass
- -Zoomen: vergrößern der Ansicht
- -Ausdrucken: ausdrucken des Diagrammes
- -Export : exportiern des Diagrammes in eine Datei

GMF-Map: Hier wird die Verbindung der bisher erstellten eigenständigen Teildefinitionen spezifiziert: Metamodell, grafische Definition und Tooling Definition. D.h., die eigentliche Bauanleitung für die Generierung des Editors wird genau jetzt erstellt.

-eclass
- -Canvas Mapping
- -Node Mapping
- -Link Mapping- Verbindung zwischen den nodes (source und target)

2.3. DSL-Tools

Die domain specific language tools sind als Bestandteil in Microsofts Visual Studio integriert. Sie sind für die Erstellung von Modellen gedacht, die die Experten auf dem jeweiligen Gebiet in ihrer domänenbezogenen Sprache entwerfen. Dabei werden sie durch eine größtmögliche Vereinfachung und Automation von den Werkzeugen unterstützt.
Das soll die Zusammenarbeit von Experten auf verschiedenen Gebieten erleichtern, die an einem gemeinsamen Projekt arbeiten. Häufig sind das Ingenieure, Ärzte, Buchhalter und eben Softwareentwickler.
Die abstrakte Modellierungssprache ist im datenbankartigen domain model framework gespeichert. Dort steht, wie Artefakte im Modell arbeiten und miteinander in Kontakt treten können. Das wären im Fall einer Arztpraxis ein Patient, ein Symptom, eine Krankheit, aber auch ein Anruf, ein Termin oder ein Hausbesuch, der im Kalender steht und bei besonderer Wichtigkeit mit roter Farbe hervorgehoben wird. Hier werden datenbankbasierte Operationen wie die Rückgängig-Option angeboten.
Konkret wird diese Sprache verwendet, wenn man daraus ein Modell für einen Arztbesuch erschafft. Für das Zusammensetzen dieses Modells wird das designer surface framework angeboten. Dort kann der Anwender die Daten des Metamodells benutzen, um ein Modell zu bauen. Weiterhin werden Funktionen wie das autolayout der Modellierungselemente angeboten.
Das validation framework, ein weiterer Teil der DSL-Tools, dient dazu, die verwendeten Artefakte im Modell auf ihre Einschränkungen zu überprüfen, die im domain model framework gespeichert sind. Ebenfalls von dort aus geht die template engine, die die Datenbank als Hilfe benutzt, um Vorlagen neuer Artefakte zu erzeugen, die sich an bereits vorhandenen orientieren.

3. unsere Modellierungsartefakte:

Einige Artefakte fielen uns bei der Betrachtung der Programme auf, die wir weiterverwenden wollen:

- Rechteck: Platzierung oder Gruppierung von Inhalten
- Kreis: als Start- oder Endpunkt oder zur Markierung
- Raute: Entscheidungs- oder Abfragepunkt
- Kommentar: zur Verständlichkeit des Modells
- Pfeil: Verwandschaft, Zugriff oder andere Aktivität
- export: als Bild oder Modell
- import: als Modell
- zoomen: für die Übersichtlichkeit
- Farbe: in die Elemente, für die Anschaulichkeit