Lösung – Aufgabenblatt 2

Verantwortlicher für Recherche: *Tobias Rosenkranz*

Inhalt:

1. Glossar

Eclipse ist ein Open-Source-Framework zur Entwicklung von Software nahezu aller Art. Die bekannteste Verwendung ist die Nutzung als Entwicklungsumgebung (IDE) für die Programmiersprache Java. Eclipse ist nicht auf Java festgelegt und wird aufgrund seiner offenen plugin-basierten Struktur mittlerweile für sehr unterschiedliche Entwicklungsaufgaben eingesetzt.

Plugin (von engl. to plug in - einstöpseln, anschließen) oder Ergänzungs- oder Zusatzmodul ist eine gängige Bezeichnung für ein kompatibles Softwareprogramm, das in ein anderes Softwareprodukt "eingeklinkt" wird und dieses um zusätzliche Features erweitert.

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.

Das Eclipse **Graphical Modeling Framework/GMF** stellt einen generativen Bestandteil und eine Laufzeitinfrastruktur zur Entwicklung graphischer Editoren basierend auf EMF und GEF zur Verfügung.

Das **Graphical Editing Framework/GEF** erlaubt es Entwicklern ein existierendes Modell einer Applikation zu nutzen um daraus in relativ kurzer Zeit ein umfangreichen graphischen Editor zu erzeugen. GEF setzt eine MVC Architektur ein, die eine einfache Manipulation des Modells durch das GUI ermöglicht.

Modell-View-Controller/MVC-Architektur ist eine Architektur, die das Modell, in diesem Fall die Daten und die zugehörigen Algorithmen (Modell) , von der Repräsentation der Daten (View) und der Manipulation der Daten (Controller) trennt.

Framework (engl. Rahmenwerk, Fachwerk) ist ein Begriff aus der Softwaretechnik und wird insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungs-Ansätzen verwendet. Es ist eine Sammlung von aufeinander abgestimmten Klassenbibliotheken, die in ihrer Gesamtheit ein wiederverwendbares und erweiterbares Gerüst für die Entwicklung von Software bilden zu denen ggf. auch passende Builder/Compiler gehören können.

Ecore ist ein Metamodell des EMF und basiert auf der MOF (Meta Object Facilites), stark Vereinfacht einer Art Klassendiagramm, das direkt von UML importiert werden kann und ergänzt diese um weitere Fähigkeiten. Es ist dabei selbst seiner eigenen Sprache beschrieben, unförmlich: es ist sein eigenes Metamodell.

MOF (Meta-Object-Facility) ist eine Spezifikation der Object Management Group und stellt eine abstrakte Sprache und zur Spezifizierung´, Konstruktion und Verwaltung von technologieunabhängigen Metamodellen dar.

Metamodellierungssprache ist eine abstrakte Sprache, die in ihrer Abstraktionsebene noch höher steht als eine Modellierungssprache. Sie dient dazu, Modellierungssprachen zu entwickeln.

2. Eclipse – Konzeptioneller Aufbau und Pluginkonzept

Eclipse existiert in den neueren Versionen (seit 3.0) selber „nur noch“ als Kern, der die Plugins einbindet, welche die eigentliche Funktionalität ausmachen. Weiterhin können die Plugins wiederum über Schnittstellen erweitert werden und können sich untereinander verwenden.

3. Beschreibung der Darstellung der abstrakten und konkreten Syntax in den einzelnen Tools

3.1. GMF

In **GMF** wird die abstrakte, sowie die konkrete Syntax nur in einer baumartigen Struktur dargestellt, sämtliche Atome und Verbindungen zwischen diesen werden auch nur in dieser Struktur definiert. Eine optische Aufwertung durch Farben, unterschiedliche Symbole oder Darstellung von Abhängigkeiten in Form von Zusammenfassungen zusammenhängender Symbole findet nicht statt (nur durch die Verzweigungen im Baum).

3.2. DSL-Tools

Die Microsoft **DSL-Tools** unterstützen die Erstellung von Metamodellierungssprachen in folgender Weise: Es existiert ein Wizard zur Erstellung des Projekts, der bereits vorgefertigte Lösungsansätze vorhält. Auf einer Zeichenfläche kann man mit Hilfe einer Vielzahl von Icons und Symbolen, die individuell anpassbar sind und durch farbige Darstellung eine optische Trennung zwischen den Entitäten und den Darstellung der Beziehungen zwischen diesen ein Modell durch einfaches Drag'n Drop realisieren. Man kann Entitäten und Beziehungen durch einfaches Drag'n Drop erstellen, modifizieren und löschen. Den Beziehungen können Namen zugewiesen, sowie Multiplizitäten hinzugefügt werden. Der Codeparser, der das Modell der Metamodellierungssprache in ein Modelleditor übersetzt, kann durch Designer Definitionen beeinflusst werden.

3.3. MetaEdit+

MetaEdit+ von MetaCase ist ein weiteres Tool zur Erzeugung von Metamodellierungssprachen. Es besitzt eine Menge von Werkzeugen, die zur Erzeugung von Metamodellierungssprachen hilfreich sind, insbesondere ist die Möglichkeit, die Daten des Metamodells als Diagramm, Matrizen oder Tabelle darzustellen hierbei zu erwähnen. Das Metamodell von MetaEdit+ besteht im Wesentlichen aus 6 Grundkonzepten:

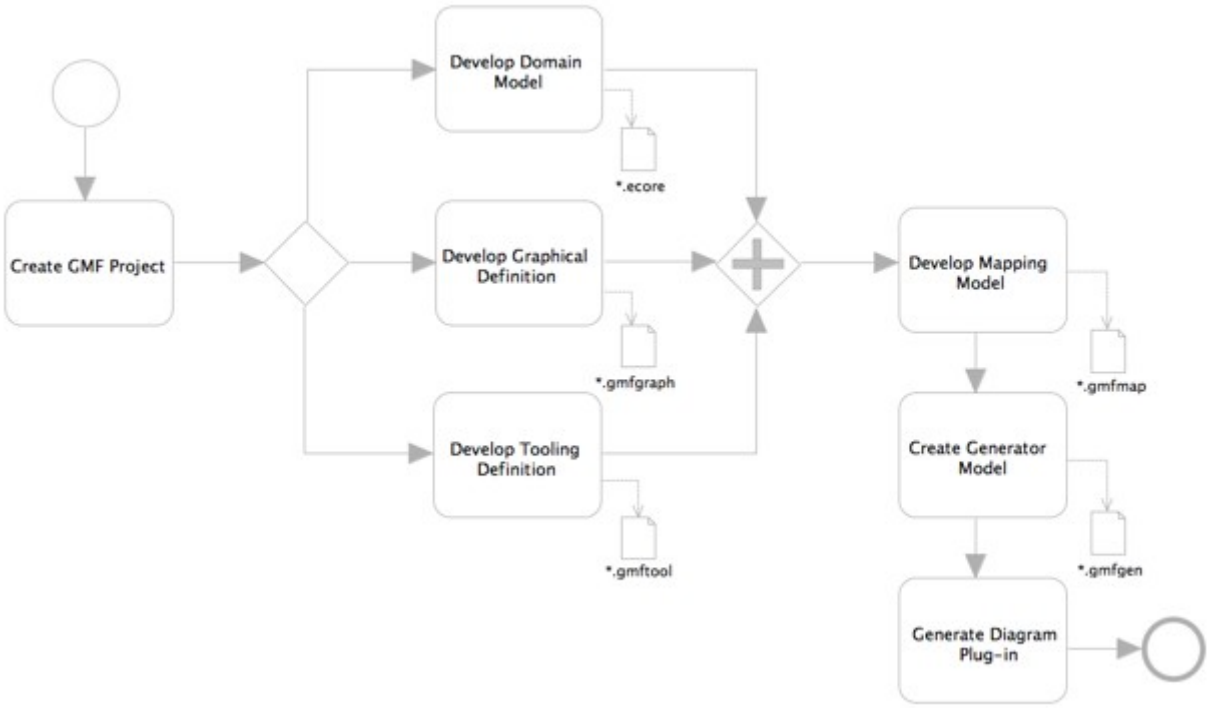
i. „Properties“ zur Definition von Attributen und deren Typ (z.B. Attribut name als String).
ii. „Objects“ zur Definition von Objekten, die über Properties verfügen können (z.B. Klasse mit dem Attribut name).
iii. „Relations“ zur Definition von Beziehungen zwischen Objekten (z.B. Vererbung in Klassendiagrammen).
iv. „Roles“ zur Definition der Rollen von Objekten in einer Beziehung (z.B. Super- und Subklasse bei einer Vererbung).
v. „Ports“ zur Definition von Schnittstellen der Objekte zur Anbindung bestimmter Beziehungen (bekannt aus Kompositionsstrukturdiagrammen).
vi. „Graphs“ zur Definition des Aufbaus eines gültigen Modelles, insbesondere werden hier „Constraints“ für Beziehungen (erlaubte Partner, Kardinalität etc.) festgelegt.

Die Definition der Sprachelemente wird im wesentlichen dialogbasiert vorgenommen, erst die Objekte samt ihrer zu speichernden Daten, dann werden die Regeln festgelegt,wie die Objekte zueinander in Relation stehen dürfen.

4. GMF (Graphical Modeling Framework)

GMF ist ein Eclipse Projekt, das eine Brücke zwischen dem Meta-Modelling-Framework EMF (Eclipse Modeling Framework) und dem GEF (Graphical Editing Framework) schlagen soll. Letzteres erlaubt die Erstellung von graphischen Editoren für ein bestimmtes Anwendungsmodell. GMF ermöglicht eine einfachere Erstellung von graphischen Editoren und bietet, durch die Integration von EMF, zusätzliche Funktionen im Bezug auf das Domain Modell.

Das GMF besteht aus zwei großen Teilen, die GMF-Runtime, die die für die Editoren notwendige Funktionalität bereitstellt, sowie der Tooling-Komponente, die ein Framework für die automatische Generierung eines graphischen Editor-Plugins aus textuellen Definitionen heraus anbietet.



Überblick zu GMF

4.1. Ecore

Ecore ist das auf der MOF (Meta Object Facilites. Stark vereinfacht definiert MOF eine Art Klassendiagramm [das direkt von UML importiert wird] und ergänzt dieses um weitere Fähigkeiten.) basierende Metamodell der EMF und ist dabei auch in seiner eigenen Sprache, d.h. durch sein eigenes Metamodell in Ecore beschrieben. Ecore-basierte Klassen sind Java-Klassen und damit von *java.lang.Object* abgeleitet. Zusätzlich sind hier aber alle Modellklassen auch von *org.eclipse.emf.ecore.EObject* abgeleitet. Über *eClass()* haben diese Klassen Zugriff auf die Ecore-Klasse *org.eclipse.emf.ecore.E-Class*, das Gegenstück zu *java.lang.Class*, mit Zugriff auf die Metainformationen, die Ecore bereitstellen. Alle mit Ecore zusammenhängenden Klassen beginnen mit einem großen „E“, alle Funktionen mit einem kleinen „e“. Die ecore-Datei enthält das Metamodell.

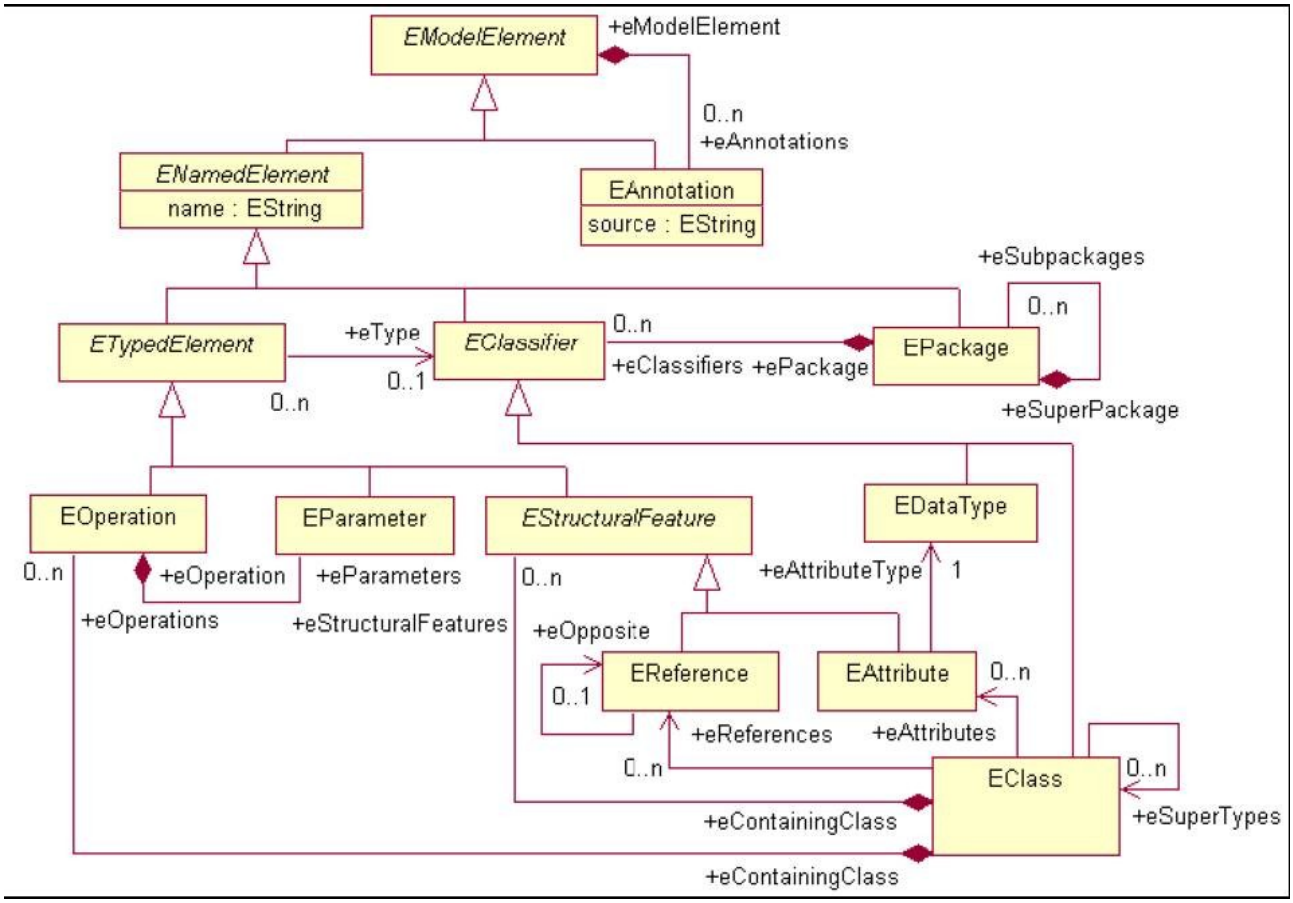
4.1.1. Die vier wichtigsten Klassen von Ecore

i. Durch **EClass** werden Klassen selbst modelliert. Diese sind durch einen Namen eindeutig beschrieben, können Attribute und Referenzen beinhalten und können von beliebig vielen *super classes* abgeleitet sein.

ii. **EAttribute** beschreibt ein Attribut einer Klasse, das durch seinen Namen und seinen Typ definiert ist.

iii. **EDataType** beschreibt einen möglichen Datentyp für ein Attribut.

iv. **EReference** wird verwendet, um Beziehungen zwischen Klassen zu modellieren. Eine Referenz ist dabei wiederum über ihren Namen sowie deren Typ definiert. Der Typ der Referenz beinhaltet immer den Namen (oder den Identifier) der Klasse, die sich am anderen Ende der Klasse befindet. Durch das Attribut *containment* kann weiteres auch die gleich lautende, semantisch strengere Form der Beziehung definiert werden.



Der Kern des eCore-Modells

4.2. GMFgraph - Graphical Definition - *.gmfgraph

In dieser Datei wird die graphische Repräsentation für Elemente definiert (plattform-neutral). Hier werden intern in die rein graphische Repräsentation (Figures) auf der Zeichenfläche und die zu zeichnenden Entitäten (Nodes, Links, Compartments) unterteilt, die später im Mapping (gmfmap) referenziert werden. Nodes sind dabei Knoten-Elemente, die über Connections verbunden werden können. Label sind Beschriftungen im Diagramm und Compartments definieren schließlich Knoten, die innerhalb von anderen Knoten liegen können.
Für die graphische Repräsentation (Figure) werden allgemeine Formen wie Rechtecke, abgerundete Rechtecke, Linien und Labels, verwendet. Zusätzlich können selbst-definierte Formen benutzt werden, um die graphische Erscheinungsweise den eigenen Bedürfnissen anzupassen.

4.2.1. Wichtige Klassen von gmfgraph

Alignment	- Anordnung der Elemente
ColorConstants	- Liste der gebräuchlichen Farben
Direction	- enthält Richtungsangaben für die Positionierung von Objekten
FontStyle	- Definition der Schriftstile (bold, italic, normal)
LineKind	- Gestaltungsarten von Verbindungslinien

4.2.2. Einige Interfaces von gmfgraph

Canvas	- Graphische Zeichenoberfläche
Color	- setzen von Farben
Connection	- setzen von Verbindungen zwischen Objekten (Figure)
Label	- implementieren von Schriftflächen
Ellipse	- zeichnen von Ellipsen

4.3. GMFtool - Tooling Definition - *.gmftool

Hier werden die Informationen über Elemente (Tools) der Palette des Editors definiert. Jedes Tool besteht neben einer Tool-Beschriftung und einem Tooltip aus der Angabe zweier Icons, die dann als Symbole in der Toolbar erscheinen.

4.3.1.Wichtige Klassen von gmftool

ActionKind	- modifizieren / kreieren von (eigenen) Aktionen
AppearanceStyle	- Erscheinungsbild des Toolbareintrages
StandardToolKind	- standarisierten Schaltflächen z.B. Zoom

4.3.2. Einige Interfaces von gmftool

Menu	- abstrakte Oberklasse für Werkzeugleiste, auf der alle Werkzeuge angeordnet werden
Image	- ablegen von Icons für die Werkzeuge
AbstractTool	- erstellen von eigenen Werkzeugen

4.4. GMFmap - Mapping Model - *.gmfmap

Dieses Mapping spezifiziert die Beziehungen / Verknüpfungen zwischen Domain-Elementen (*.ecore), Graphical-Elementen (*.gmfgraph) und Tooling-Elementen (*.gmftool). Hier können auch die Regeln zur Prüfung der Integrität (Audits) definiert werden. Es wird auf oberster Ebene zwischen jenen Elementen unterschieden, die als Knoten direkt auf der Zeichenfläche erscheinen sollen (Top Node Reference) und jenen Elementen, die als Link (also Verbindung
zwischen Knoten) realisiert werden sollen. Links verknüpfen zwei graphische Repräsentation (Nodes) miteinander. Im Mapping werden auch Compartments spezifiziert, das sind die Elemente, die Container-Funktionalität übernehmen und deswegen weitere Nodes in sich aufnehmen können.

4.4.1.Wichtige Klassen von gmfmappings
Language - Sprache in der das Mapping erfolgen soll
Severity - Charakterisierung von Systemmeldungen

4.4.2 Ein Interface von gmftool
Mapping - Mappen von Links, Nodes, Diagram …