Inhaltsverzeichnis

1 **Begriffe** **2-3**

2 **Grundsätzliche Konzepte** **3-9**
- 2.1 Plug-In-Konzept 3
- 2.2 Meta-Modellierung 3
- 2.3 Test-Konzept 3
- 2.4 MVC-Konzept 3-4
- 2.5 MetaEdit+ 4
- 2.6 DSL-Tools 5-6
- 2.7 Eclipse-Architektur & Konzepte 6-7
- 2.8 GMF 7-9

3 **Beschreibung der Rahmenapplikation** **10**
- 3.1 Leistungsmerkmale 10
- 3.2 Anwendungsfälle 10-11
- 3.3 Äußerlich sichtbare Aspekte der inneren Logik 11

1 Begriffe

Begriff	Beschreibung
DSL Tools	*(Domain Specific Language Tools)* Mit den DSL Tools ist es möglich, graphische Sprachen zu entwerfen und daraus Quelltext oder eine Textausgabe zu generieren.
Ecore	Ecore ist das auf MOF basierende Metamodell der EMF und ist auch in seiner eigenen Sprache, d.h. Durch sein eigenes Metamodell in Ecore geschrieben.
EMF	*(Eclipse Modeling Framework)* *Das 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.*
Framework	Ein Framework ist eine Grundstruktur, ein Rahmenwerk. In der Softwareentwicklung bestimmt das Framework die Softwarearchitektur der Anwendung und stellt für die Entwicklung Bausteine für die Designgrundstruktur zur Verfügung.
GEF	*(Graphical Editing Framework)* Das GEF ist ein Framework zur Entwicklung von graphischen Anwendungen in der Eclipse Plattform.
GMF	*(Graphical Modeling Framework)* GMF ist ein Framework, mit welchem man voll funktionsfähige, graphische, Eclipse-basierte Editoren für selbst zu definierende EMF-basierte Metamodelle zu generieren.
MetaEdit+	MetaEdit+ dient zur Definition von Meta-Modellen und generiert einen zugehörigen Editor.
Metamodell	Modelle, die beschreiben, wie Modelle gebaut werden. In der Systemanalyse werden Modelle realer Systeme erstellt. Wie diese Modelle zu bauen (Syntax) und zu interpretieren (Semantik) sind, wird durch Methoden beschrieben. Methoden sind selbst wieder reale Systeme. Es liegt also nahe, auch die Methoden wieder durch Modelle zu beschreiben.

MOF	*(Meta Object Facility)* MOF wurde von der Object Management Group (OMG) eingeführt und beschreibt eine spezielle Metadaten-Architektur.
Plug-In	Ein Plug-In ist ein Computerprogramm, welches in ein anderes Software-Produkt integriert wird. Es ergänzt dabei die Software. Anders als ein Add-on stellt es jedoch eine eigenständige Software dar.

2 Grundsätzliche Konzepte

2.1 **Plug-In-Konzept:**

Hierbei handelt es sich um ein gängiges Konzept zur Erweiterung von Softwareprodukten durch neue Funktionalitäten. Dazu muss die Rahmenapplikation, welche erweitert werden soll, eine (im besten Fall dokumentierte) Schnittstelle bieten, die es ermöglicht, Software zu entwickeln, die bestimmte Dienste der Rahmenapplikation nutzt oder neue Dienste über die definierte Schnittstelle bereitstellt.

2.2 **Meta-Modellierung:**

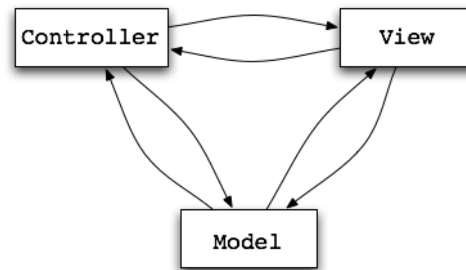
Um die Vorteile der modellgetriebenen Softwareentwicklung für Systeme mit unterschiedlichen Kriterien und Anforderungen nutzen zu können, bieten übliche Modellierungssprachen oftmals nur unzureichende Unterstützung. Eine Lösung zur Überwindung dieser Lücke ist der Entwurf geeigneter Modellierungssprachen mittels Metamodellierung. Es existieren Metamodellierungswerkzeuge, die ausreichend Unterstützung bieten, um neue Modellierungssprachen zu entwerfen und entsprechende Werkzeuge zu generieren.

2.3 **Test-Konzept:**

Das systematische Testen von Software, das die Abgrenzungskriterien zur Behebung von Mängeln der Software definiert, ist ein essenzieller Bestandteil des Entwicklungsprozesses.

2.4 **MVC-Konzept:**

Beim MVC-Konzept handelt es sich um ein Entwurfsmuster zur Trennung bestimmter Programmeigenschaften. Die Idee des Musters ist die Trennung des Programms in die drei Einheiten: Datenmodellen (Model), Datendarstellung (View) und Programmsteuerung (Controller). Ziel des Modells ist ein flexibles Programmdesign, um u.a. Änderungen oder Erweiterungen einfach zu realisieren und die Wiederverwendbarkeit der einzelnen Komponenten zu ermöglichen. Außerdem sorgt das Modell bei großen Anwendungen für mehr Übersicht und Ordnung durch Reduzierung der Komplexität.



2.5 **MetaEdit+:**

MetaEdit ist, wie bereits in der Begriffsbeschreibung gesagt, ein Tool zum Definieren von Meta-Modellen der Generierung der zugehörigen Editoren. Das Interface von MetaEdit+ besteht aus vier Teilen: dem Canvas (rechts), in welchem die Diagramme erstellt werden, einer baumartigen Liste (links oben), in der alle im Diagramm verwendeten Objekte nach Typ geordnet sind und einem Fenster, in dem Informationen über das gerade ausgewählte Objekt wiedergegeben werden, dazu später mehr. Die Toolbar (oben) ermöglicht es, Objekte schnell und einfach dem Diagramm hinzu zufügen.

In MetaEdit+ ist es möglich neben der standardisierten UML-Notation auch selbst definierte Symbole zu verwenden, was den Umgang mit den Diagrammen intuitiver gestaltet. Auch die Objekte können hinsichtlich ihrer Parameter/Spezifikationen selbst definiert werden, wodurch das Diagramm flexibler an das Projekt angepasst werden kann. So kommen zur Generierung des Quellcodes meistens modifizierte Aktivitätsdiagramme zum Einsatz, wo auch die Möglichkeit zum Tragen kommt, Symbole selbst zu definieren, da die standardisierte UML-Notation erstens zu unflexibel ist und zweitens nicht genug Symbole bietet, um auf die Bedürfnisse des Projekts zugeschnitten zu werden. Die Objekte können grob unterteilt werden in Eingabefenster, mathema-tische Berechnungen, zu erfüllende Bedingungen und Ausgabefenster. Ein Doppelklick auf ein Objekt öffnet ein Menü, in welchem man dessen Parameter festlegen kann. Hiervon sollen im Folgenden einige aufgelistet werden:

- **Eingabefenster**
  Name, Art der Eingabe (Text, Zahl, Code, Datum, Zeit, Abfrage mit Auswahlmöglichkeiten)
- **mathematische Berechnungen**
  Name, Definition der mathematischen Funktion in Quellcode
- **Bedingungen**
  Name der zu untersuchenden Variable, zu untersuchende Bedingung in Quellcode
- **Ausgaben**
  Ausgabetext, Art der Ausgabe (Bestätigung, Fehler, Information)

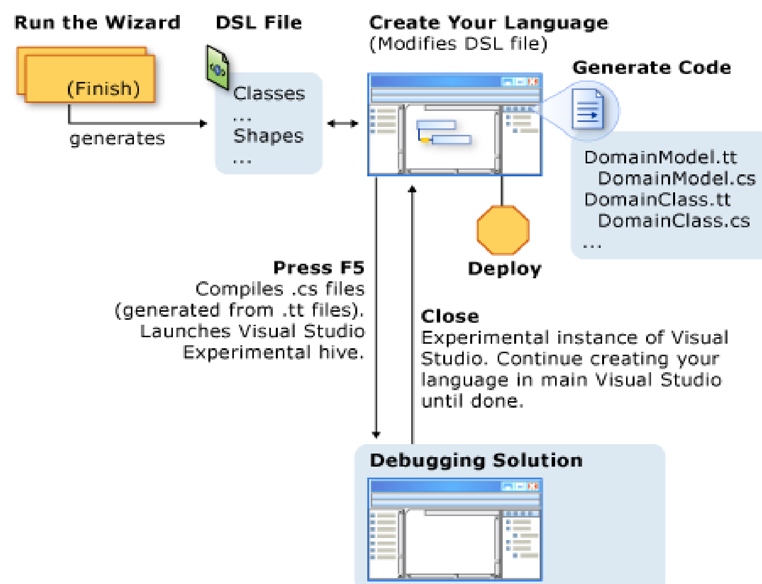
Diese Objekte werden durch Pfeile verbunden, welche die Methodenaufrufe im Quellcode darstellen, sowie die Zustandsübergänge im Diagramm repräsentieren.

2.6 **Domain-Specific Language Tools von Microsoft (DSL Tools):**

Mit den DSL-Tools ist es möglich graphische Sprachen zu entwerfen und daraus Quelltext oder eine Textausgabe zu generieren.

DSL-Tools besteht aus folgenden Bestandteilen:

- **Ein Projekt-Wizard**, der vorgefertigte Lösungen beinhaltet. Aus diesen kann man über einen Designer und einen Text-Output-Generator ein Metamodell erstellen.
- **Einen graphischen Designer**, mit dem man das Metamodell definierten und editieren kann.
- **Designer-definitions** in XML. Der Quellcode für die Implementierung der Designer wird aus den Definitionen in dieser Datei generiert. Dadurch ist es möglich einen Designer in Visual Studio vollständig zu definieren, ohne eine einzige Zeile Quellcode zu schreiben.
- **Code-Generatoren**, die aus der Definition des Metamodells und den Designer-definitionen einen Quellcode generieren. Dabei werden Metamodell und Designerinformationen überprüft und auftretende Fehler und Warnungen werde ausgegeben.
- **Einen Definitionsrahmen** für Textausgabe. Diese Generatoren generieren aus den Daten des Metamodells Textausgaben. Parameter werden dabei durch Ergebnisse ersetzt, deren Berechnung ein in die Umgebung integriertes Visual C#-Skript übernimmt.



Für die Funktionalität der DSL-Tools wird Visual Studio 2005 Professional vorausgesetzt. Nach dem Start von Visual Studio 2005 Professional kann man den Domain Specific Language Designer nutzen. Damit können Projekte für die Definition von Metamodellen erstellt werden. Der Projekt-Wizard stellt dabei schon verschiedene Modelltypen zur Verfügung, die man dann den Anforderungen entsprechend erweitern und modifizieren kann. Zur Verfügung stehen dabei:

- Aktivitätsdiagramm
- Use-Case-Diagramm
- Klassendiagramm
- Minimal Language

Das Metamodell wird graphisch in einer an UML-Diagramme angelehnten Darstellung definiert. Dabei ist der Bildschirm u.a. in folgende Bereiche unterteilt:

- **Designer**
  Hier wird das Metamodell graphisch definiert.
- **DSL-Toolbox**
  In dieser Bibliothek befinden sich alle Elemente, die für die Definition des Metamodells verfügbar sind. Sie können per Mausklick ausgewählt und per Drag and Drop eingefügt werden. Solche Elemente sind z.B. Klassen, Beziehungen zwischen den Klassen, Ports.
- **Eigenschaften**
  Hier werden alle Eigenschaften eines Elementes, das man ausgewählt hat detailliert aufgelistet und können geändert werden.
- **DSL Details**
  In diesem Fenster kann das Verhalten einzelner Elemente definiert werden.

Nachdem man sich für eine Modellvorlage aus dem Wizard entschieden hat, werden in Visual Studio 2 Projekte angelegt:

- **DSL**
  Das DSL-Projekt definiert die Metaspezifische Sprache und die zugehörigen Editier- und ausführenden Tools.
- **DslPackage**
  Das DslPackage-Projekt bestimmt, wie die Sprachtools in Visual Studio intergriert sind.

Der Quellcode wird hauptsächlich aus den „language definition files“ generiert, in den Vorgaben DomainModel.tt. Die meisten Änderungen, welche die Sprache betreffen, werden in dieser Datei vorgenommen. Der gesamte Quellcode wird aus den „text templates“ (*.tt) generiert, welche die Sprachdefinition auslesen und den entsprechenden Quellcode ausgeben. Mit DSL-Tools definiert man also in einem ersten Designer das Metamodell der Sprache. Dann wird die Definition von Visual Studio nach Fehlern und Unstimmigkeiten durchsucht, bevor eine zweite Instanz geöffnet wird, in der man dann mit der definierten Sprache graphisch programmieren kann. Man hat dann die Möglichkeit Quellcode zu generieren oder Textausgaben machen zu lassen.

2.7 **Eclipse-Architektur & Konzepte:**

Das Ziel von Eclipse ist es, eine leicht erweiterbare Entwicklungsumgebung für jede erdenkliche Anwendung zu schaffen. Der Grundgedanke ist es, eine offene Syntax zu schaffen, die sämtliche Werkzeuge unter einem Dach integriert, was den Aufwand

für Portierung und Kommunikation bei Entwicklungen minimieren soll.
Eclipse folgt einer reinen Plug-In-Architektur. Dies bedeutet, dass Eclipse nicht nur durch Plug-Ins um bestimmte Funktionalitäten erweitert werden kann, sondern praktisch jeder Teil von Eclipse stellt ein Plug-In dar. Seit der Version 3 ist die interne Plu-In-Verwaltung auf eine OSGi-Plattform aufgebaut, was eine Grundlegende Architektur-Änderung darstellt und es ermöglicht, Plu-Ins in Eclipse zur Laufzeit zu laden und wieder zu entladen. Es ist unter anderem möglich, eigene Anwendungen auf die Eclipse-Plattform aufzusetzen und mit dieser laufen zu lassen. Dies bietet unter anderem den Vorteil, dass diese Anwendungen dann auch über die Möglichkeit verfügen, durch Plug-Ins erweitert zu werden.

2.8 **GMF:**

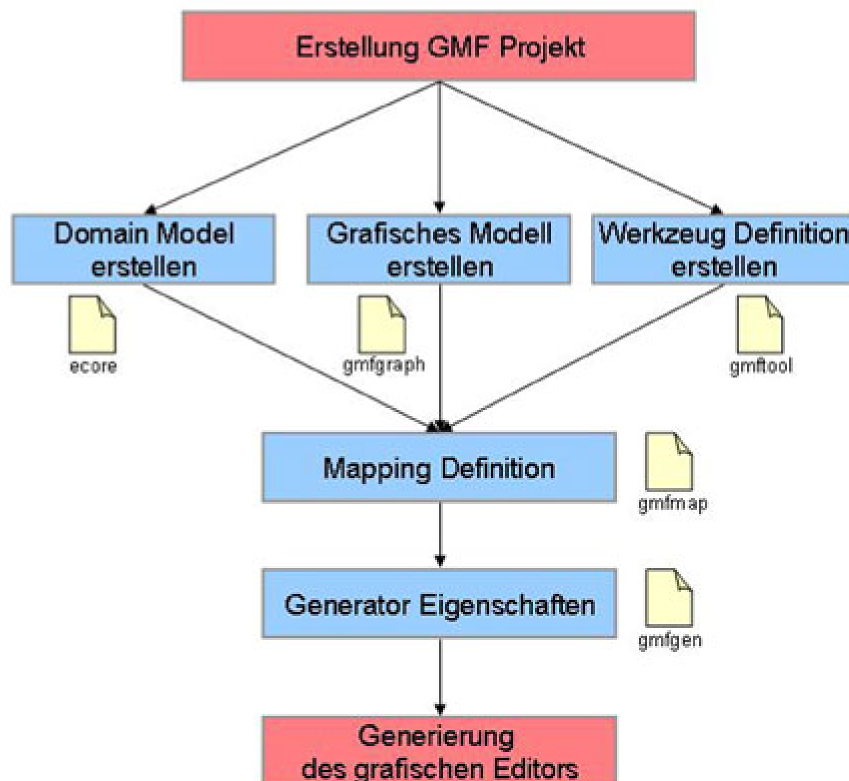
Das Graphical Modeling Framework stellt ein Framework zur automatisierten Erstellung von grafischen Diagrammen auf Basis von EMF und GEF dar.
GMF ist in der Lage, mit Hilfe von Transformationsanweisungen, Mappings sowie einem Daten-Modell einen graphischen Editor unter GEF zu erstellen.
Es besteht aus zwei großen Teilen:

(1) **GMF Runtime**, welche die notwendigen Funktionalitäten für die Editoren bereitstellt. Sie unterteilt sich dabei in zwei große Bereiche:
  - **Common Frameworks Layer**, das einige Basisfunktionen bereitstellt
  - **Diagram Layer**, welches auf das Common Framework Layer aufbaut und die mit Diagrammen in Verbindung stehende Funktionen bereitstellt, z.B.:
    - Shapes, welches grafische Objekte mit beliebiger Form im Diagramm sind
    - die Palette, was dem Bereich im Diagramm entspricht, in dem mögliche auswählbare Modellelemente dem Benutzer angezeigt werden
    - eine Menge von vorgefertigten Figuren(Kreis,Dreieck, etc.)
    - Layout-Tools für die automatische Anordnung von Diagrammelementen
    - Connections als Verbindungen zwischen Shapes
    - Mehrere Diagramm-Assistenten

(2) **Tooling-Komponente**, welche ein Framework bereitstellt mit dem automatisch ein grafisches Editor-Plugin auf Basis des Ecore-Models generiert wird, wobei versucht wird den Erzeuger soweit wie möglich von Programmiertätigkeiten zu befreien. Bei dieser Generierung trennt das GMF die verschiedenen voneinander unabhängigen Einzeldefinitonen und vereinigt sie dann durch das sogenannte Mapping. Diese einzelnen Komponenten sind:

- **Domain Generation Model**
  Das GMF gewinnt Informationen dazu aus Ecore-Dateien, in denen das Domain Model liegt. Der Kern des Ecore-Models besteht dabei aus 4 Entitäten. Durch *EClass* werden Klassen selbst modelliert. *EAttribute* beschreibt ein Attribut einer Klasse, das durch seinen Namen und Typ definiert ist. *EDataType* beschreibt einen möglichen Datentyp für ein Attribut. Und *EReference* wird verwendet, um Beziehungen zwischen Klassen zu modellieren.
- **Graph Definition Model (.gmfgraph)**
  In dieser Datei wird die graphische Repräsentation für Elemente der Zeichenfläche(*Canvas*) definiert. GMF unterteilt intern in die rein graphische Repräsentation auf der Zeichenfläche und die zu zeichnenden Entitäten, die später im Mapping Model referenziert werden. Als Unterelement der *Canvas* ist die *FigureGallery*, in der alle im Editor verfügbaren Figures definiert werden. Dabei gibt es einmal die die Möglichkeit Standardfigures aus *Shape*(z.B. Rechtecke, Ellipsen, Linien) zu wählen oder eigene *CustomFigures* zu definieren. Diese Figures können außerdem weitere Figures sowie Attribute, die das Aussehen weiter konkretisieren, enthalten.
  Die dann tatsächlich zu zeichnenden Entitäten unterteilt GMF in *Node, Connection, Compartment* und *Label*. Nodes sind dabei die Diagramm- bzw. Knoten-Elemente, die über Connections verbunden werden können. Labels sind Beschriftungen im Diagramm und Compartments definieren schließlich Knoten, die innerhalb von anderen Knoten liegen können. Jede zu zeichnende Entität stellt mindestens ein Tupel aus Figure und einem eigenen Namen dar. Ergänzend dazu sind dann weitere Attribute, abhängig von der Art der zu zeichnenden Entität, zu definieren.
- **Tooling Definition Model (.gmftool)**
  Hier wird im wesentlichen das Aussehen der Palette festgelegt. Es wird dabei genau eine Palette angelegt die dann im Editor sichtbar ist und dem Benutzer sämtliche im Mapping definierten Elemente anbietet. Die einzelnen Elemente können in Gruppen(*ToolGroup)* zusammengefasst werden. Die wesentlichen Einträge darin sind die *CreationTool*s, mit denen ein Element im Diagramm erzeugt werden kann. Jedes Tool besteht dabei neben Namen und Beschreibung aus zwei Icons(je nach Auflösung dann gewählt), die als Bilddatei in dem entsprechenden Pfad abgelegt sind und in der Toolbar des Editors dann erscheinen.
- **Mapping Definition Model (.gmfmap)**
  Über das Mapping werden Graph Definiton Model, Tooling Definiton Model und Domain Generation Model miteinander verknüpft. An oberster Stelle befindet sich ein *Mapping*. Darunter befindet sich die *TopNodeReference*, von der jedes auf dem Diagramm gezeichnete

Element einen Eintrag braucht. Jede *TopNodeReference* benötigt ein *NodeMapping* wodurch eine EClass des Domain Models mit einem Node des Graphical Models, einem Tool des Tooling Models und einem Kontextmenue des Tooling Models verknüpft wird. Neben der *TopNodeReference* gibt es noch ein *LinkMapping,* wo jede Verbindung von zwei Nodes definiert werden muss und genau ein *CanvasMapping*, wo das Diagramm beschrieben wird. Desweiteren besteht die Möglichkeit diese drei aufgezählten Mappings durch darunter liegende Knoten genauer konfiguriert werden können. So können z.B. unter NodeMappings Child- und Compartment-Einträge definiert werden oder unter NodeMappings und LinkMappings Labels angefügt werden.



3 Beschreibung der Rahmenapplikation

3.1 Leistungsmerkmale

Für eine Beschreibung der Leistungsmerkmale der Rahmenapplikation GMF, siehe Kapitel 2.8.

Da GMF auf Eclipse aufbaut (dies als Plug-In erweitert) ist ebenfalls die grundlegende Struktur von Eclipse, wie sie in Kapitel 2.7 beschrieben ist, von Bedeutung.

3.2 Anwendungsfälle

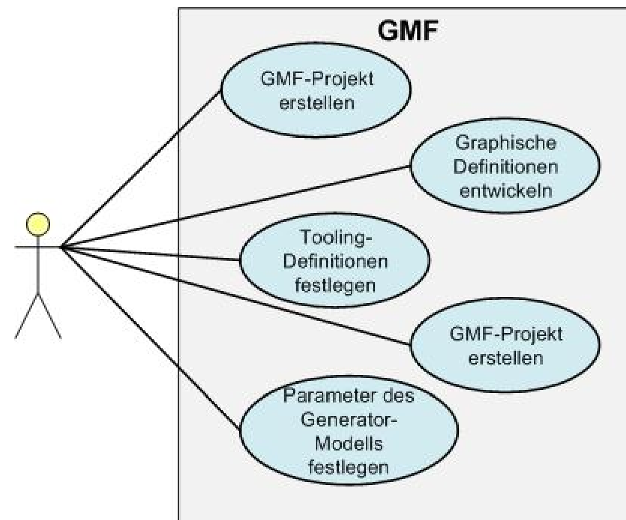
Name: GMF-Projekt erstellen
Akteur: User
Beschreibung: User erstellt ein neues GMF-Projekt zur Entwicklung des Plug-Ins.
Ergebnis: Ecore-Datei mit Domain Model.

Name: graphische Definitionen entwickeln
Akteur: User
Beschreibung:User definiert die graphische Representation der Elemente der Zeichenfläche.
Ergebnis: Datei (*.gmfgraph) mit Informationen über Elemente der Canvas, bzw. FigureGallery.

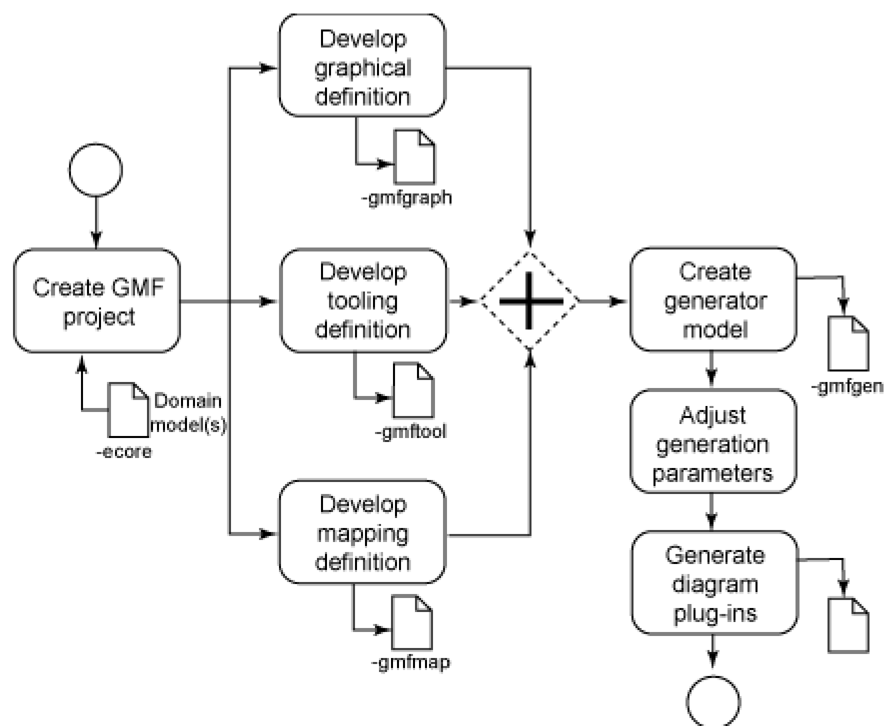
Name: Tooling-Definitionen festlegen
Akteur: User
Beschreibung: Festlegen der Palette, die im Editor angezeigt wird (Aussehen).
Ergebnis: Datei (*.gmftool) mit sämtlichen Elementen, die im Mapping festgelegt wurden.

Name: Mapping-Definitionen erstellen
Akteur: User
Beschreibung: Verknüpfen von Graph Definition Model, Tooling Definition Model und Domain Generation Model.
Ergebnis: Datei (*.gmfmap) mit Verknüpfungsinformationen zu den einzelnen Bestandteilen des Editors.

Name: Parameter des Generator-Modells einstellen
Akteur: User
Beschreibung: Einstellen der Parameter des von GMF generierten Generator-Modells.
Ergebnis: Generator-Modell wird mit Parametern ausgestattet



3.3 **Äußerlich sichtbare Aspekte der inneren Logik**



Dieses Diagramm stellt die inneren Abläufe bei der Erstellung eines Metamodells dar. Durch Definition, grafische Darstellung und die Tooling-Elemente wird das Interface festgelegt. Hierbei entstehen die Dateien gmfgraph und gmftool. Die Definition der gmfmap verknüpft die gmfgraph und gmftool miteinander und mit dem Ecore, wodurch festgelegt wird, welche Ecoreteile durch grafische Elemente dargestellt werden sollen und wie die entsprechenden Tooling-Elemente heißen, die die Modellierung der grafischen Elemente anbieten. Dabei entsteht das GMF-Generatormodel, welches nach einer eventuellen Nachbearbeitung ein fertiges GMF-Editor-Plug-In generiert.