Benutzerhandbuch

Benutzerhandbuch

Henshin TGG Editor

Ein Editor zur Entwicklung von Tripel Graph Grammatiken

Version 0.9

Daniel Binanzer Waka Nagasawa Frank Röske
Sebastian Schasse

1. Mai 2013

Technische Universität Berlin

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Der TGG Editor wurde an der Technischen Universität Berlin im Rahmen des Projekts *Visuelle Sprachen* im Wintersemester 2011/2012 von den Studenten Daniel Binanzer, Waka Nagasawa, Frank Röske und Sebastian Schasse als Eclipse Plugin in Java entwickelt und baut auf die Plugins EMF, GEF und MuvitorKit auf. Der TGG Editor ist eine Spezialisierung des ebenfalls in einem vorhergehenden Projekt *Visuelle Sprachen* entwickelten Henshin Editors[1] . Mit dem TGG Editor lassen sich grafische Tripel Graph Grammatiken für beliebige Sprachen erzeugen und testen. Die entwickelten TGGs werden im TGG Editor als Instanzen des Henshin EMF Modells mit zusätzlichen Layoutinformationen gespeichert. Die verwendeten Quell- und Zielsprachen müssen als EMF Modell definiert sein und importiert werden. TGGs ermöglichen die formal korrekte Transformation von einer Quell- in eine Zielsprache. Beispiele für Quell- und Zielsprachen sind grafische Sprachen wie UML Klassendiagramme oder Relationale Datenbankmodelle, aber auch textbasierte Sprachen wie COBOL und C++. Der hier vorgestellte TGG Editor unterstützt allerdings nur visuelle Sprachen. Die Entwicklung der Transformationsregeln in einer TGG sind sehr einfach. Nachdem diese Regeln für die Quell- und Zielsprache erstellt wurden, können mit einem Klick Graphen von einer Quellsprache in die Zielsprache übersetzt werden.

1.2 Formale Spezifikation

Dieses Kapital umreißt kurz die theoretischen Hintergründe der Graph- und Modelltransformation und der *Tripelgraphgrammatik* (TGG). Für ein tieferes Verständnis empfiehlt sich eine tiefere Recherche in entsprechender Literatur[2][3][4].

[1] **henshinwebsite**.
[2] **ehrig2006**.
[3] **ehrig2009**.
[4] **schuerr1995**.

1.2.1 Graphen

Die folgende Definition bezieht sich nur auf gerichtete Graphen, da wir in dem hier vorgestellten Graphtransformation nur mit gerichteten Graphen arbeiten.

Ein *gerichteter Graph* ist ist ein Quadrupel (V, E, src, tar) wobei V eine Menge von Knoten, E eine Menge von Kanten, src die source-Funktion $src : E \to V$, die einer Kante einen Startknoten zuordnet, und tar die target-Funktion $tar : E \to V$, die einer Kante einen Endknoten zuordnet, ist.

1.2.2 Typgraph

Ein *Typgraph* ist ein Graph $TG = (V_{TG}, E_{TG}, src_{TG}, tar_{TG}$, bei dem die Mengen V_{TG} und E_{TG} die Typalphabete für Knoten und Kanten darstellen.

Im TGG Editor sind die *Meta-Modelle* für Source-, Corresbondence- und Targetgraphen (siehe Tripelgraphgrammatik **??**) sogenannte Typgraphen. Sie ermögliche erst die Erzeugung von *getypten Graphen*

1.2.3 getypter Graph

Ein *getypter Graph* ist ein Graph, in dem jeder Knoten und jede Kante einen Typen hat.

Im TGG Editor sind alle Graphen und Regeln getypte Graphen und müssen verträglich mit dem *Triple-Typgraph* sein. Dieser Triple-Typgraph wird über die drei Typgraphen, also den Meta-Modellen, für Source, Correspondence und Target erzeugt.

1.2.4 Graphmorphismus

Seien G_1 und G_2 zwei Graphen mit $G_1 = (V_1, E_1, src_1, tar_1)$ und $G_2 = (V_2, E_2, src_2, tar_2)$. Dann ist $\varphi : G_1 \to G_2$ ein *Graphmorphismus* mit $\varphi = (\varphi_V, \varphi_E)$, wenn die Funktionen $\varphi_V : V_1 \to V_2$ und $\varphi_E : E_1 \to E_2$ sowohl ziel- $(\varphi_V \circ src_1 = src_2 \circ \varphi_E)$ als auch quellerhaltend $(\varphi_V \circ tar_1 = tar_2 \circ \varphi_E)$ sind.

1.2.5 Regel

Diese Definition bezieht sich auf getypte Regeln Graphen und Graphmorphismen, da nur diese im TGG Editor verwendet werden.

Eine Regel r besteht aus den drei Graphen L (auch *left-hand side*, *Linke-Hand-Seite* oder *LHS* genannt), K (auch *Klebegraph* genannt) und R (auch *right-hand side*, *Rechte-Hand-Seite* oder *RHS* genannt) und zwei injektiven Graphmorphismen l und r.
Im TGG Editor wird der Klebegraph K nur impliziert, da er in unserer Anwendung nur den Schnitt von L und R darstellt.

1.2.6 Graphtransformation

Eine Graphtransformation ist eine auf Regeln basierte lokale Modifikation eines Graphen.

Um eine Graphtransformation mit einer Regel r auszuführen, wird zuerst ein Match m von L im Hostgraph G gesucht. Dabei werden alle Knoten und Kanten aus L auf typgleiche und im gleichen Gefüge zueinander stehende Knoten und Kanten des Graphen G abgebildet. Im Transformationschritt werden die Elemente aus $L \setminus K$ gelöscht, die Elemente aus K erhalten, und die aus $R \setminus K$ hinzugefügt. Anschließend werden Kanten auf denen die Funktionen *src* oder *tar* leere Werte zurückliefern gelöscht.[5]

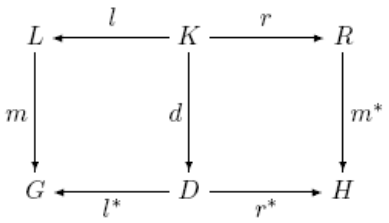


Abbildung 1.1: DPO Graphtransformation

1.2.6.1 EMF Besonderheiten

Eclipse Modeling Framework (*EMF*) bietet die Möglichkeit, Domain-Modelle zu modellieren. EMF unterscheidet das *Meta-Modell* und das aktuelle Modell. Das Meta-Modell stellt die Struktur des Modells dar. Ein Modell ist ein Instanz vom Meta-Modell.
EMF hat zwei Meta-Modelle. Eins ist das *Ecoremodell*, das Informationen der Klassen enthält, das Andere ist das Genmodell, welches zusätzliche Informationen zur Code-Generierung enthält, z.B. Infomationen über Dateien und Pfade.

EMF bietet das Framework als Plugin an, damit Informationenen des Modells gespeichert werden können. Wenn die EMF Meta-Modelle in TGG-Editor

[5]Kanten, die keinen Start- oder Zielknoten haben werden auch „hängende Kanten“ oder „dangling edges“ genannt und sind nach Definition des Graphens nicht zulässig.

importiert werden, können Instanz-Modelle basierend auf den Meta-Modellen erstellt werden. Alle verwendbare Kanten und Knoten für Modelle im TGG-Editor sind durch das importierte EMF Modell definiert.

1.2.7 Negative Anwendungsbedingung (NAC)

Die Menge *negativen Anwendungsbedingungen* $N = \{N_1, ..., N_n\}$ über den LHS-Graphen L ist eine Menge partieller Erweiterungen N_i von L. Die Menge $N_i \cap L$ wird im TGG Editor über das sogenannte Mapping definiert (siehe 3.4.1.2). Die NAC wird dazu verwendet, die Ausführbarkeit einer Graphtransformation auf das Nichtvorhandensein bestimmter Muster im Graphen zu beschränken.
Die negativen Anwendungsbedingungen erweitern die Regeldefinition, sodass ein Match nur noch ausführbar wird, wenn keine der vorhandenen NACs gematcht werden kann. Es gibt auch komplexere Bedingungen, bei denen einzelne NACs über logische Operationen „AND“ und „OR“ miteinander verknüpft werden können. Die Implementation des TGG Editors unterstützt nur die Verundung aller NACs. Das heißt sobald eine NAC einer Regel matcht ist diese Regel nicht mehr anwendbar. Formal bedeutet das, dass ein Match $m : L \to G$ N erfüllt (keine NAC matcht), wenn für $\forall N_i \in N$ gilt, es exisitert kein injektiver Graphmorphismus $q_i : N_i \to G$ verträglich mit m.

1.2.8 Tripelgraphgrammatik

Die *Tripelgraphgrammatik (TGG)* ist ein Formalismus für regelbasierte Transformation von zwei Graphen oder Modellen mit unterschiedlichen Typisierungen. Modelle werden als Paare von Sourcegraphen und Targetgraphen definiert, die über Correspondencegraphen verbunden werden. Eine Tripelgraphgramatik $TGG = (TG, S, TR)$ besteht aus einem Tripeltypgraph TG, einem Tripelstartgraph $S = \phi$ und einer Menge von Tripleregeln TR.

1.2.8.1 Tripelgraphen

Ein *Tripelgraph* $g = (G_S \xleftarrow{s_G} G_C \xrightarrow{t_G} G_T)$ besteht aus den drei Graphen G_S (Sourcegraph), G_C (Correspondencegraph), G_t (Targetgraph) und den zwei Graphmorphismen $s_G : G_C \to G_S$ und $t_G : G_C \to G_T$.
Ein *Tripelgraphmorphismus* $m = (ms, mc, mt) : G \to H$ zwischen zwei Tripelgraphen G und H besteht aus drei Graphmorphismen $m_S : G_S \to H_S$, $m_C : G_C \to H_C$ und $m_T : G_T \to H_T$, so dass $m_S \circ s_G = s_H \circ m_C$ und $m_T \circ t_G = t_H \circ m_C$ gilt.

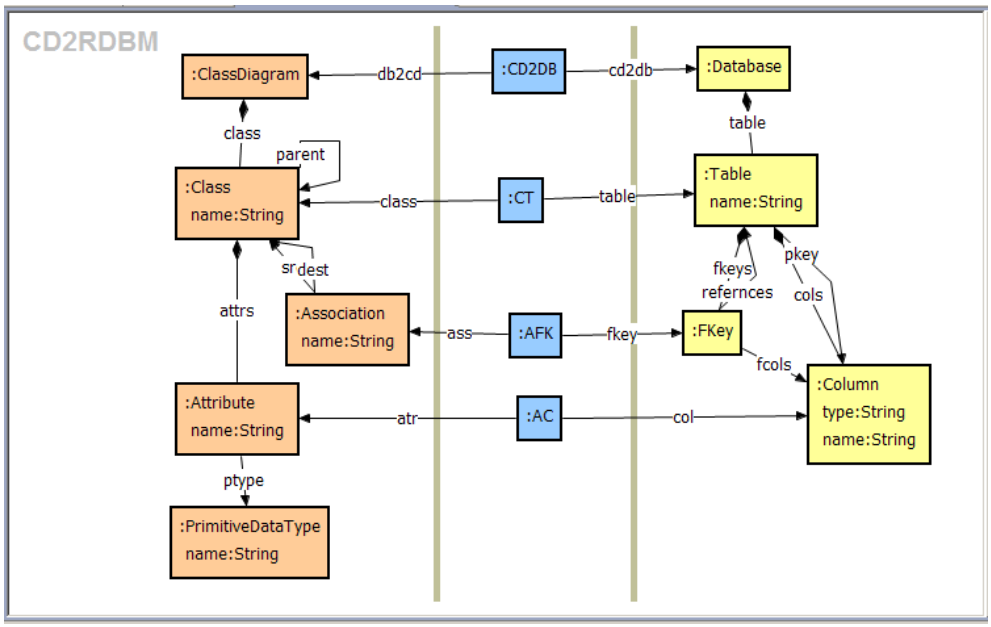


Abbildung 1.2: Beispiel eines Tripelgraphen für CD2RDBM

Abbildung 1.2 zeigt den Tripeltypgraphen für eine Beispielmodelltransformation von Klassendiagrammen zu Datenbankmodellen.

In der Abbildung steht der Sourcetypgraph TG_S links, der Correspondencetypgraph TG_C in der Mitte und der Targettypgraph TG_T rechts. TG_S definiert die Struktur von Klassendiagrammen und TG_T definiert die Struktur von relationalen Datenbanken. So entspricht eine Klasse aus dem Klassendiagramm einer Tabelle, ein Attribut einer Spalte und eine Assoziation einem Fremdschlüssel.

1.2.8.2 Tripelregel

Tripelregeln bauen Source-, Target-, und Correspondencegraphen synchron auf. Eine Tripelregel tr ist ein injektiver Tripelgraphmorphismus $tr = (tr_S, tr_C, tr_T) : L \to R$ (siehe linken Teil in Abbildung 1.3).

$$\begin{array}{ccccc} L = (L^S & \xleftarrow{s_L} & L^C & \xrightarrow{t_L} & L^T) \\ {\scriptstyle tr}\downarrow & {\scriptstyle tr^S}\downarrow & {\scriptstyle tr^C}\downarrow & {\scriptstyle tr^T}\downarrow & \\ R = (R^S & \xleftarrow[s_R]{} & R^C & \xrightarrow[t_R]{} & R^T) \end{array} \qquad \begin{array}{ccc} L & \xrightarrow{tr} & R \\ {\scriptstyle m}\downarrow & (PO) & \downarrow \\ G & \longrightarrow & H \end{array}$$

Abbildung 1.3: Tripelregel und Tripeltransformationsschritt

Gegeben sei ein Tripelgraphmorphismus $m : L \to G$. Dann ist ein *Schritt* einer *Tripelgraphtransformation* (TGT) $G \overset{tr,m}{\Longrightarrow} H$ (rechts in Abbildung 1.3) von G nach H durch den Pushout von Tripelgraphen mit Co-Match $n : R \to H$ und Transformation inclusion $t : G \hookrightarrow H$. In Abbildung 1.4 ist eine Tripelregel für eine Beispielmodelltransformation CD2RDBM (ClassDiagram to DataBase Model) in verkürzter[6] Notation abgebildet. LHS und RHS der Regel sind in einem Tripelgraph dargestellt. Elemente, die durch die Regel erstellt werden, sind mit einem grünen „<++>“ markiert. Die Regel „Class2Table“ erstellt eine Klasse mit dem Attribut name, dessen Wert „cn“ ist, zusammen mit der entsprechenden Tabelle im relationalen Datenbankmodell.

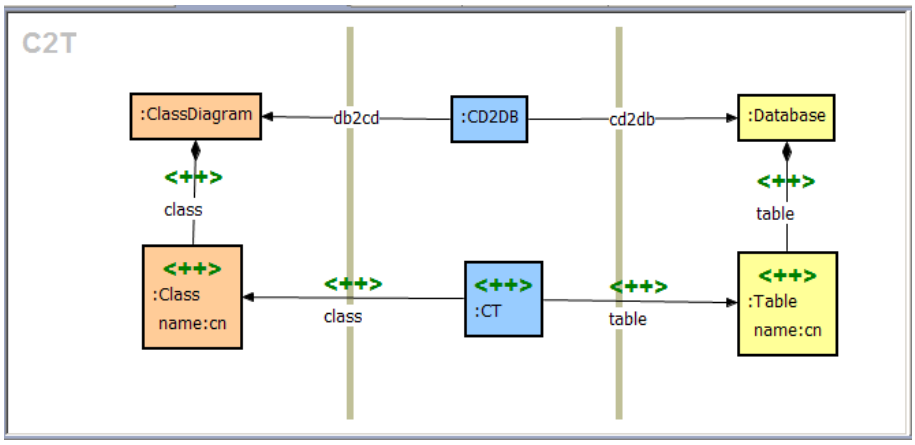


Abbildung 1.4: Beispiel Tripelregel: Class2Table

1.2.8.3 Operationale Regeln

Aus jeder Tripelregel tr werden eine Sourceregel tr_S für die Konstruktion von einem Modell der Sourcesprache und eine Forwardregel tr_F für die Forward-Transformationsseqenz abgeleitet, wie in Abbildung 1.5 und Abbildung 1.6 gezeigt.

In **??** wird die Forwardregel, die aus der Regel Attribute2ForeignKey abgeleitet wurde, als Beispiel gezeigt. Im Sourcegraphen sind die Elemente, die noch zu übersetzen sind, mit dem grünen Zeichen „<tr>“ markiert.

[6]die Verkürzung ist Abbildung von LHS und RHS in einem Grpahen durch <++>-Markierung

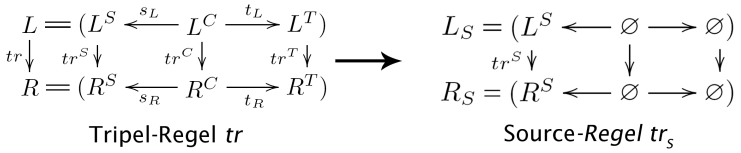


Abbildung 1.5: Ableitung der Sourceregel

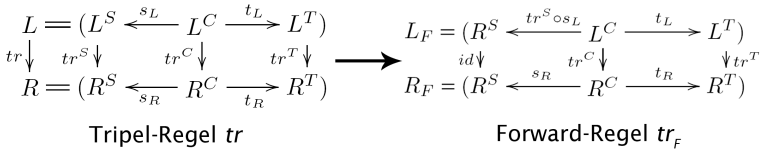


Abbildung 1.6: Ableitung der Forwardregel tr_F

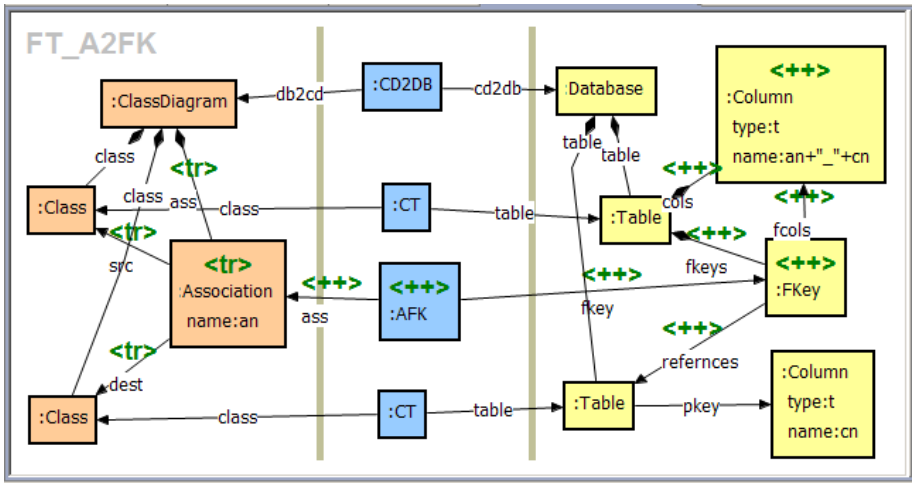


Abbildung 1.7: Beispiel Forwardregel: FT_Attribute2ForeignKey

1.3 Ein paar Worte zur Implementation

1.3.1 Entwicklungsumgebung und Plugins

Der Henshin TGG Editor ist als Plugin für Eclipse entwickelt worden und benötigt die Plugins EMF und GEF. Die verwendeten Technologien werden im Folgenden näher beschrieben.

1.3.1.1 Eclipse

Eclipse[7] geschrieben und ist deswegen auch für die drei großen Betriebssysteme Windows, Mac und Linux erhältlich. Eclipse ist kostenlos und Quell-offen. In erster Linie wird Eclipse als Entwicklungsumgebung für Java verwendet.

Wegen der Vielzahl an Plugins kann Eclipse aber auch für etliche andere Anwendungsfälle verwendet werden. So ist auch der TGG Editor ein Plugin von Eclipse.

1.3.1.2 EMF

Das Eclipse Modeling Framework[8] grafisch ähnlich zu UML Klassendiagrammen zu modellieren. Aus einem so entstandenen EMF Modell mit der Abkürzung .ecore kann automatisiert Java Code generiert werden. Der Code kann dann weiter verwendet werden, um Instanzen des Modells zu erstellen und zu manipulieren. EMF kann auch einen grundlegenden grafischen Tree-Editor als Eclipse Plugin generieren. Der TGG Editor verwendet das Henshin Modell, das mit EMF entwickelt wurde, und erweitert dieses mit eigenen Klassen (siehe hierzu auch Abschnitt 1.3.2).

1.3.1.3 GEF

Das Graphical Editing Framework[9] in der Model-View-Controller Architektur entwickelt wurde. Mit GEF lassen sich sehr gut grafische Editoren als Plugins für Eclipse Entwickeln. GEF bietet ein Framework, damit der Entwickler grundlegende Funktionalitäten eines grafischen Editors nicht von Grund auf implementieren muss.

So ist es in GEF relativ einfach grafische Objekte mit Verbindungen auf dem Canvas zu zeichnen. Hier setzt GEF auf die Klassen des Packages Draw2d[10] GEF bietet weiter eine Palette mit Tools mit denen die grafischen Objekte manipuliert werden können. Es ist außerdem durch GEF sehr einfach die Undo/Redo für die einzelnen Operationen zu implementieren. In unserem TGG Editor sind genau die Views für Graphen und Regeln mit den Tools in der Palette die GEF Editoren.

[7] **eclipsewebsite**.
[8] **emfwebsite**.
[9] **gefwebsite**.
[10] **draw2dwebsite**.

1.3.1.4 MuvitorKit

MuvitorKit leitet sich ab von Multi-View-Editor-Plugin ist ein Paket, das von Toni Modica an der TU Berlin entwickelt wurde. MuvitorKit baut auf GEF auf und lässt es zu mehrere GEF Views in einer einzigen Page anzuzeigen und zu editieren. Wir verwenden diese Funktionalität im TGG Editor im Rule View. Denn hier werden zusätzlich NACs angezeigt und können editiert werden, falls die Regel NACs besitzt. Außerdem verwenden wir im TGG Editor MuvitorKit für den Tree Editor.

1.3.2 Das Henshin Metamodell und die Erweiterungen des TGG Editors

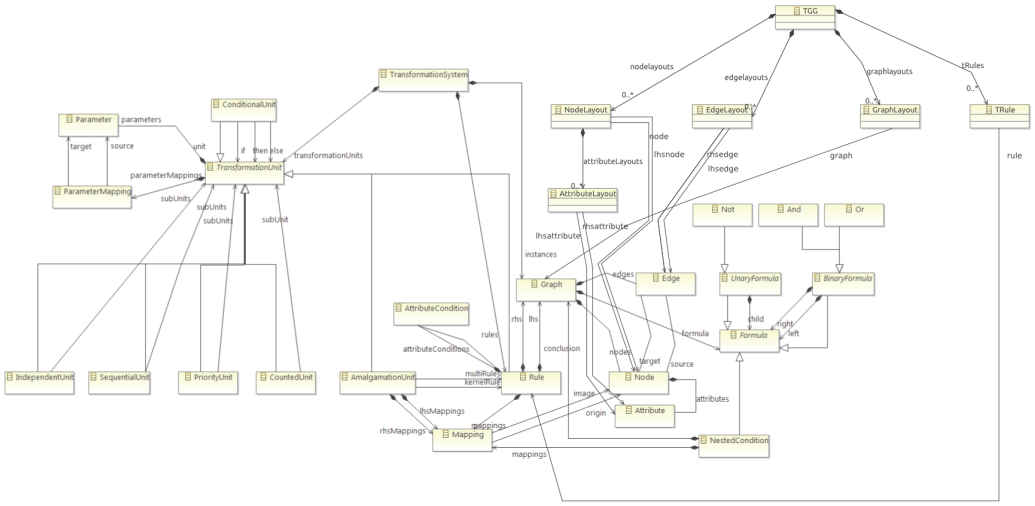


Abbildung 1.8: Das TGG Henshin Metamodell.

Der TGG Editor baut auf dem Henshin Modell auf. Das in EMF modellierte Henshin Modell wurde entwickelt für die allgemeineren Graphtransformationen. Für den TGG Editor wurden in einem eigenen EMF Modell alle nötigen Erweiterungen für Tripel Graph Grammatiken und Layoutinformationen gemacht. Die Klassen aus dem TGG Modell referenzieren die Klassen aus dem Henshin Modell nur. So bleibt das Henshin Modell unmodifiziert und macht Instanzen des Henshin Modells kompatibel zu anderen Editoren, die Instanzen des Henshin Modells verarbeiten können. In Abbildung 1.8 sind beide Modelle in einem Diagramm vereint. Das Diagramm gibt eine globale Sicht auf das

gesamte Metamodell, das vom TGG Editor verwendet wird[11].

1.3.2.1 Die Erweiterungen für den TGG Editor

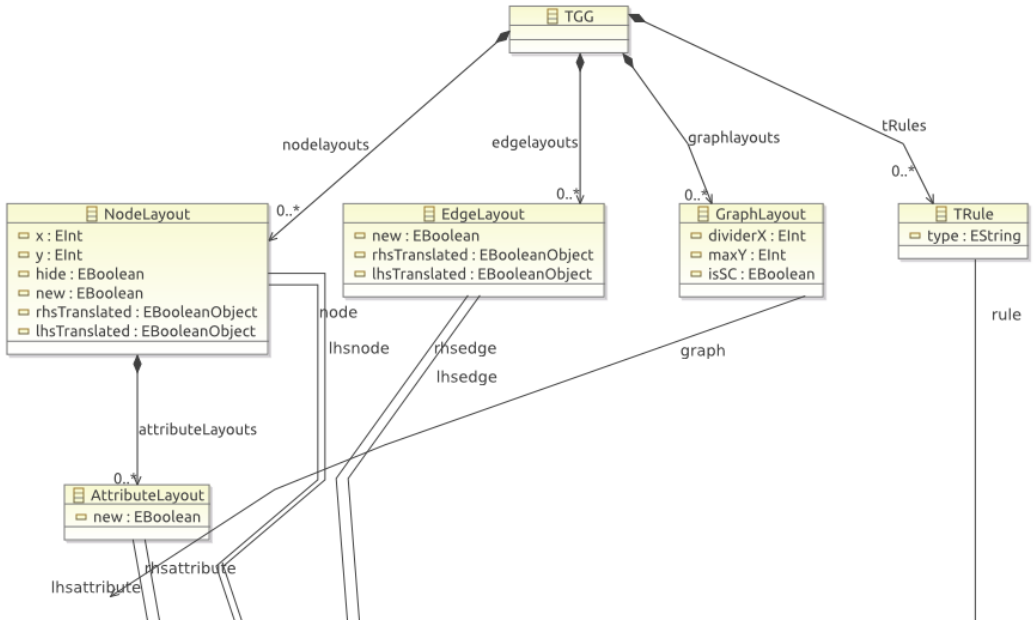


Abbildung 1.9: Das TGG Metamodell mit Attributen als Ausschnitt aus dem globalen Diagramm.

In diesem Abschnitt möchten wir auf das eigenständige EMF Modell des TGG Editors etwas genauer eingehen. Die Klassen *NodeLayout*, *EdgeLayout*, *GraphLayout*, *TRule* und *AttributeLayout* haben je eine Referenz zu den entsprechenden Klassen im Henshin Modell. Falls Instanzen einer Rule eine solche Layoutklasse haben gibt es je eine Referenz zur entsprechenden Instanz im LHS und RHS Graphen. So hat beispielsweise ein EdgeLayout für „eine“[12] Edge in einer Rule die Referenzen *lhsedge* und *rhsedge*.

[11]Für diejenigen, die sich mit dem Henshin Modell näher auskennen ist noch hinzuzufügen, dass der TGG Editor TransformationUnits nicht benutzt. Es werden ausschließlich Graphen und Rules manipuliert. Außerdem können für Rules lediglich NACs (NotApplicationCondition) und keine allgemeinen ACs erstellt werden, da dies bei Tripel Graph Grammatiken nicht unbedingt nötig ist.

[12]Eigentlich sind es zwei Instanzen einer Edge, nämlich eine Edge im LHS Graphen und eine Edge im RHS Graphen. Im Rule View des TGG Editors wird die Edge allerdings nur als eine einzige Edge angezeigt. Siehe hierzu auch Abschnitt **??**

1.3.2.1.1 NodeLayout Das *NodeLayout* enthält nicht nur Layoutinformationen für einen Node sondern es hat auch noch weitere zusätzliche Attribute, die für Tripelgraphen benötigt sind. Die Attribute x und y geben die Position eines Nodes auf dem Canvas an. Für Knoten in einer Rule gibt es das Attribut *new*. Es zeigt an, ob ein Knoten mit einer Rule neu erstellt wird. Die Attribute *rhsTranslated* und *lhsTranslated* sind lediglich für Nodes aus dem Sourceteil in einer FTRule bzw. TRule. Sie zeigen an, ob ein LHS bzw. RHS Node in einer FTRule als *to be translated* markiert werden soll.

1.3.2.1.2 AttributeLayout Das *AttributeLayout* hat ebenfalls das Attribute *new*. Dieses Attribut ist äquivalent zu dem gleichnamigen Attribut aus dem NodeLayout.

1.3.2.1.3 EdgeLayout Die Attribute *new*, *rhsTranslated* und *lhsTranslated* im *EdgeLayout* entsprechen im wesentlich den gleichnamigen Attributen aus dem NodeLayout. Sie gelten nur eben für Edges.

1.3.2.1.4 GraphLayout Die Attribute *dividerX* und *maxY* sind Layoutinformationen für die Divider in den Views, die die einzelnen Teilgraphen Source, Correspondence und Target unterteilen.

1.3.2.1.5 TRule Wenn eine Instanz Rule von *TRule* referenziert wird, dann ist die Rule keine normale Regel sonder eine FTRule. Das Attribute *type* gibt den Typen einer Rule an, in diesem Fall nur FTRule, da andere Spezialregeln noch nicht implementiert sind.

2 Installation

2.1 Installationsanleitung

Umm den Henshin TGG Editor nutzen können ist vorerst die Installation von Eclipse und einigen Plugins notwendig, auf denen der Henshin TGG Editor basiert.

Um den Henshin TGG Editor möglichst fehlerfrei nutzen zu können wird empfohlen von der Downloadseite des Eclipseprojekts[1] die Ëclipse IDE for Java Developersïn der Version 3.7[2] herunterzuladen und zu installieren.

Nach der Installation installiert man über den Eclipse Marketplace folgende Plugins herunter: Ecore Tools SDK (Incrubation), EMF - Eclipse Modeling Framework SDK, Graphical Editing Framework GEF SDK, Graphical Modeling Framework (GMF) Runtime SDK. Sind diese Plugins installiert sind, ist es notwendig das Muvitor-Plugin der TU Berlin, die Plugins EMF-Henshin-Interpreter, EMF-Henshin-Matching, EMF-Henshin-Model und zuletzt den TGG-Editor selbst zu installieren. Das geht ganz einfach mit dem ausgelieferten TGG-Editor.zip Archiv. Dieses wird komplett in den Pluginordner der Eclipseinstallation entpackt. Nach einem Neustart stehen alle Plugins zur Verfügung.

2.2 Erster Schritt

Bevor man mit dem Editieren einer Tripelgraphgrammatik beginnen kann ist es notwendig die Metamodelle (siehe **??**) für Source-, Correspondence- und Targetteilgraph der Tripelgraphgrammatik zu erstellen und zu importieren. Die Metamodelle werden als Ecoremodell mit dem EMF Ecore Baum-Editor oder dem graphischen Ecore-Editor von GMF in Eclipse erstellt. Sind die Metamodelle fertig kann man wie im Beispielworkflow (siehe Kapitel **??**) seine eigene Tripelgraphgrammatik erzeugen.

[1] http://www.eclipse.org/downloads/

[2] Version 3.7 entspricht der Eclipse Indigo Version

3 Funktionen

In diesem Kapitel werden alle Funktionalitäten des TGG Editors erklärt. Die Basis Funktionen, sind Funktionen, die so gut wie jeder gewöhnliche Editor unterstützt. Danach folgen die speziellen Funktionen des TGG Editors.

3.1 Basis Funktionen

3.1.1 Copy/Paste

Copy und Paste werden vom TGG Editor leider nicht unterstützt.

3.1.2 Undo/Redo

Während des Arbeitens mit dem Editor kann man jederzeit die einzelnen ausgeführten Schritte mit *Undo* rückgängig machen und mit *Redo* wiederholen.

1. Wähle per Rechtsklick das Kontextmenü und dort *Undo* oder *Redo*.
2. Wähle in der Menüleiste `Edit`→`Undo` und `Edit`→`Redo`.
3. Drücke den Button für *Undo* und für *Redo*.
4. Oder per Shortcut: `Strg+z` für *Undo* und `Strg+y` für *Redo*.

3.1.3 Property View

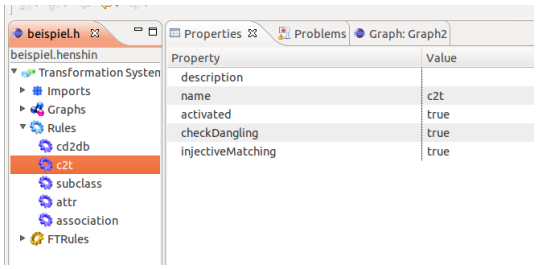


Abbildung 3.1: Property View

Namen und Typen können im *Property View* editiert werden. Zunächst muss man den *Property View* wie folgt öffnen: Man wählt im Hauptmenü `Window`→`Show View`→`Other` oder benutzt den Shortcut `Shift+Alt+q q`. Es erscheint ein Fenster. Dort wählt man im Ordner `General`

den Eintrag `Properties`. Es erscheint ein neuer View wie in Abbildung 3.1 dargestellt.

3.1.4 Direct Edit

Man hat die Möglichkeit die Namen von Objekten im oben beschriebenen Property View zu verändern oder einfach mit dem *Direct Edit* Feature. Dazu muss man das gewünscht Objekt markieren und dann noch einmal auf den Namenstext klicken. Dann erscheint der Text wie in Abbildung ... zu sehen ist. Jetzt kann man einen Namen der Wahl eingeben und auf der Tastatur mit `Enter` abschließen oder das Objekt wieder deselektieren.

3.1.5 Delete

Sobald ein Objekt ausgewählt ist kann man mit der Taste `Entfernen` das Objekt löschen. Alternativ kann man aber auch mit einem Rechtsklick auf das Objekt klicken und aus dem Kontextmenü den Eintrag *Delete* ausführen.

3.2 Tree View

Der *Tree View* ist eine baumartige Ansicht des gesamten Projektes, des *Transformation System*. Man findet hier die Ordner *Imports*, *Graphs*, *Rules* und *FTRules*.

3.2.1 Imports

Im Ordner Imports erscheinen die importierten EMF Modelle für Source, Correspondence und Target. Diese Modelle sollen die einzelnen Teile einer TGG beschreiben. Sie sind die Metamodelle. Das Metamodell für Source beschreibt demnach die Quellsprache. Das Metamodell für Target beschreibt entsprechend die Zielsprache. Wie wir wissen benötigt man außerdem noch einen Correspndence Teil, der die beiden anderen beiden Teile verbindet.

3.2.1.1 Ein EMF Modell importieren

Um ein solches Metamodell zu importieren muss es zunächst in den Workspace importiert werden. Gehe dann wie folgt vor:

1. Mit einem Rechtsklick auf den Ordner Imports kann die entsprechende Action, entweder $ImportSource, Correspondece$ oder $Target$ aufgerufen werden.

2. Wähle über *Workspace...* die entsprechende ECORE-Datei für Source, Correspondence oder Target.

Sobald ein Metamodell importiert ist, kann man in im Graph- und Rule View Instanzen des Metamodells erstellen.

3.2.2 Graphs

Im Ordner Graphs werden die Graphen, die das Transformation System enthält, angezeigt. Durch einen Doppleklick auf einen Graphen öffnet sich der Graph View. Klappt man im Tree View den Graphen auf so erscheinen alle Komponenten, die ein Graph enthält, Nodes und Edges. Nodes werden in folgendem Schema angezeigt: `Name:Typ`. Der `Name` ist der zugewiesene Node Name. `Typ` ist einer der Typen aus einem der Metamodelle (entweder Source, Correspondence oder Target). Klappt man die auch die Nodes auf so erscheinen die Edges eines Nodes. Die werden in folgendem Schema angezeigt `EdgeTyp: fromTyp->toTyp`. Der `EdgeTyp` ist einer der Typen einer Edge aus einem der Metamodelle. Da die Edges im Henshin Modell gerichtet sind, gibt es einen Typen, `fromType`, für den Source-Node und einen Typen,`toType`, für den Target-Node.

3.2.2.1 Einen neuen Graphen erstellen

Um einen neuen Graph zu erstellen gehe wie folgt vor:

1. Über einen Rechtsklick auf den Ordner *Graphs* lässt sich die Aktion *Create Graph* ausführen.

2. Wähle einen Namen für den Graphen.

3.2.3 Rules

Im Ordner Rules werden die Rules, die das Transformation System enthält, angezeigt. Ein weiteres aufklappen einer Rule funktioniert nicht. Die einzelnen Komponenten einer Rule werden demnach nicht angezeigt. Durch anklicken mit einem Doppelklick einer Rule öffnet sich die Rule View.

3.2.3.1 Eine neue Rule erstellen

Um eine neue Rule zu erstellen gehe wie folgt vor:

1. Über einen Rechtsklick auf den Ordner *Graphs* lösst sich die Aktion *Create Rule* ausführen.

2. Wähle einen Namen für die neue Rule.

Über Rechtsklick im Tree View kann man noch weitere Aktionen ausführen. Diese Aktionen stehen aber in näherem Zusammenhang mit den Views, die im folgenden Abschnitten beschrieben werden. Sie sind meistens auch zusätzlich in der Toolbar der View zu erreichen. Aus diesem Grund werden auch die Aktionen in den entsprechenden Abschnitten für die View beschrieben.

3.3 Graph View

Im *Graph View* werden Tripelgraphen einer TGG dargestellt und können editiert werden. Die Zeichenfläche des Graph Views ist dreigeteilt. Der linke Teil für Source, der mittlere für Correspondence, der rechte für Target. Zusätzlich sind die Nodes der einzelnen Teile rot, blau und gelb eingefärbt.

3.3.1 Palette

Die Palette bietet einige Tools, um den Graphen zu editieren.

3.3.1.1 Select

Das *Select Tool* ist das standardmäßig ausgewählte Tool. Mit dem Select Tool kann man einzelne Objekte mit einem Klick auswählen. Man kann aber auch mehrere Objekte auswählen, indem man einen Kasten über die Objekte zieht. Alternativ kann man aber auch alle auszuwählenden Objekte anklicken während man `Shift` gedrückt hält. Das primär ausgewählte Objekt wird rot markiert, die sekundär gewählten Objekte werden grün markiert. Dies ist hilfreich, wenn man Mappings für eine NAC editiert.

3.3.1.2 Node

Das Tool mit dem man *Nodes* zeichnen kann wird nicht direkt mit dem Namen Node angezeigt. Vielmehr existieren für je Source, Correspondence und Target Teil je eine Gruppe von Node Tools. Es wird nämlich nicht allgemein ein Node

kreiert, sondern über die Auswahl des entsprechenden Tools wird ein Node *mit einem bestimmten Typen* kreiert. Die einzelnen Tools haben den Namen der jeweiligen Typen. Hat man wie in unserem Beispiel[1] den Typ *Class* im importierten EMF Modell für den Source Teil, so heißt das Node Tool auch Class. Wenn man versucht einen Node, der eigentlich in den Source Teil gehört, in den Target Bereich zeichnen möchte, so ist dies nicht möglich. Der Mauszeiger verwandelt sich dann in ein Kreuz.

3.3.2 Edge

Mit dem *Edge Tool* kann man Edges zwischen Nodes zeichnen. Man muss hierzu zuerst auf den Quell-Node klicken, danach auf den Ziel-Node. Falls mehr als ein Typ von Edges zwischen den Nodes existieren kann, öffnet sich ein Dialog zur Auswahl des Typen. Falls die Edge nicht gezeichnet werden kann, wird dies am Mauszeiger mit einem Verbotssymbol angezeigt.

3.3.2.1 Attribute

Wenn das *Attribute Tool* ausgewählt ist, können Attribute erstellt werden. Dazu muss man auf einen Node klicken, zu welchem man das Attribut erstellen möchte. Daraufhin erscheint ein Dialog, indem man den Namen und den Wert des Attributes angeben kann.

3.3.3 Toolbar

In der *Toolbar* befinden sich Aktionen, die auf den gesamten Tripelgraphen ausgeführt werden können.

3.3.3.1 Validate Graph

Mit der Aktion Validate Graph kann man den Graphen auf seine EMF Konsistenz überprüfen. Falls die Überprüfung erfolgreich war, wird dies in einem Dialog angezeigt. Falls sie nicht erfolgreich war, werden die Fehler im Dialog angezeigt. Diese Aktion ist auch über das Kontextmenü im Tree View verfügbar, wenn man auf einen Graphen klickt.

[1] Siehe Abschnitt **??**

3.3.3.2 Execute FT Rules

Mit dieser Aktion werden alle Forward Translation Rules, die es im gesamten Transformation System gibt, ausgeführt. Nach der Ausführung wird überprüft, ob die Modelltransformation *Source Consistent* war oder nicht. Die Nodes und Edges, die nicht übersetzt wurden, werden in der Meldung angezeigt. Außerdem werden die Regeln in ihrer Ausführungsreihenfolge angezeigt. Bisher kann diese Ausführungsreihenfolge vom Benutzer nicht verändert werden. Außerdem wird sie deterministisch gewählt. Das heißt, dass bei mehrfacher Ausführung der Aktion immer wieder die selbe Reihenfolge von FT Rules gewählt wird. Die Aktion ist ebenfalls über das Kontextmenü im Tree View verfügbar, wenn man auf eine Rule klickt.

3.4 Rule View

Der Rule View ist genauso aufgebaut wie der Graph View, ist aber um einige Funktionen wie der NAC-Ansicht, einigen Palette Tools und Aktionen in der Toolbar erweitert worden. Er dient zur graphischen Erstellung von Regeln, welche später auf einen Graphen angewendet, oder zur Generierung von FT-Regeln verwendet werden können. Wir haben für den Rule View eine kompakte Art der Darstellung gewählt. Eine Rule einer TGG hat eine Graphen als linke Seite und einen Graphen als rechte Seite. Dies wird im Rule View allerdings dem Benutzer nur indirekt angezeigt. Man sieht erst einmal nur die rechte Seite. Das macht den Rule View viel übersichtlicher. Wenn man ein Element hinzufügt, wird automatisch ein Element für den Graphen der linken Seite und eines für den der rechten Seite erstellt und falls nötig wird noch ein Mapping zwischen den beiden Elementen erstellt. In unserem Rule View ist allerdings nur möglich Regeln zu erstellen, die Elemente hinzufügen (siehe Abschnitt 3.4.1.1). Hier kümmert sich der Editor automatisch um die rechte und linke Seite und die Mappings.

3.4.1 Palette

Die Palette im Rule View wurde im Vergleich zum Graph View um die Tools ++ und Mapping erweitert. Die Palette-Tools Select (3.3.1.1), Node (3.3.1.2), Edge (3.3.2), Attribute (3.3.2.1) verhalten sich wie in den verherigen Kapiteln beschrieben.

3.4.1.1 ++

Mit dem ++ Tool ist es möglich Nodes und Edges als neu zu markieren. Dies sind dann die Elemente, die bei einer Regelanwendung in einem Graphen neu erstellt werden. Folglich kann dieses Tool auch nicht auf Elemente in der NAC angewendet werden.
Zu beachten ist, dass falsches Markieren zu invaliden Regeln führen kann.

3.4.1.2 Mapping

Mit dem Mapping Tool ist es möglich ein Mapping zwischen einem Node in einem NAC Graphen und einem Node im Regelgraphen zu erstellen. Dabei können Mapping nur zwischen Nodes aus einer NAC und einer Regel erstellt werden, vom gleichen Typen sind und nicht als neu gekennzeichnet sind. Beim Erstellen eines Mapping ist zu beachten, dass als erstes der Node im Regelgraphen gewählt werden muss (dieser ist dann als einziger Node im Rule View markiert) und anschließend der gewünschte Node im NAC Graphen. Ist eine Auswahl eines bestimmten Nodes nicht möglich wird dies mit einem Verbotssymbol am Mauszeiger kenntlich gemacht.

3.4.2 Toolbar

Die Toolbar des Rule Views enthält Aktionen zur Generierung einer FT-Regel, zum Validieren und Ausführen der aktuellen Regel

3.4.2.1 Generate FT Rule

Die Aktion *Generate FT Rule* löscht, wenn vorhanden, die zu der aktuellen Regel gehörende FT-Regel und generiert aus der aktuellen Regel eine FT-Regel. Die generierte Regel lässt sich dann in Tree Editor unter *FT Rules* finden. Möchte man für alle erstellten Rules gleichzeitig die FT Rules erstellen, nutzt man mit Rechtsklick im Kontextmenü des Regelordners im Treeview die Aktion *"Generate All FT Rules"*.

3.4.2.2 Validate Rule

Mit der Aktion *Validate Rule* wird die Regel auf ihre EMF Konsistenz überprüft. Falls die Überprüfung erfolgreich war, wird dies in einem Dialog angezeigt. Falls sie nicht erfolgreich war, werden die Fehler im Dialog angezeigt. Diese Aktion ist auch über das Kontextmenü im Tree View verfügbar, wenn man mit einem Rechtsklick auf eine Rule klickt.

3.4.2.3 Execute Rule

Die Aktion *Execute Rule* für die Rule auf einem gewünschten Graphen aus. Sind mehrere Graphen im System registriert wird nach dem Starten der Aktion ein Dialog zur Auswahl des Graphens, auf dem die Rule ausgeführt werden soll, zur Verfügung gestellt. Vor der Ausführung der Regelanwendung wir die Regel allerdings noch auf Validität geprüft. Schlägt diese fehl, wird die Aktion abgebrochen. Dies wird mit einer entsprechenden Meldung angezeigt. Kann die Rule trotz erfolgreicher Validitätsprüfung nicht aus geführt werden, wird dies als Fehlermeldung angezeigt.

3.4.3 NACs

Im Tree View ist es möglich für eine Regel eine oder mehrere *NACs* (Negative Anwendungsbedingung) zu erstellen. Diese werden allerdings erst nach nochmaligen öffnen der Rule und auch nur eine NAC gleichzeitig angezeigt. Möchte man zwischen mehreren NACs wechseln, muss man die betroffene Rule im Tree View aufklappen und mit Doppelklick auf die gewünschte NAC klicken.
blablablabla

3.5 FT Rule View

Der *FT Rule View* hat den selben Aufbau wie der Rule View. Der FT Rule View dient zum Kontrollieren der automatisch generierten *FT Rules* und einer nachträglichen Erweiterung oder Änderung der NACs.

3.5.1 Palette

Im FT Rule View sind alle in der Palette verfügbaren Tools nur im Zusammenhang mit der Erstellung und Bearbeitung von NACs und deren Mappings auf die FT Rule möglich. Ein nachträgliches Editieren der FT Rule ist nicht vorgesehen.
Ansonsten verhalten sich die Palette-Tools Select (3.3.1.1), Node (3.3.1.2), Edge (3.3.2), Attribute (3.3.2.1) und Mapping (3.4.1.2) wie in der vorherigen Kapiteln.

3.5.2 Toolbar

Die Toolbar der FT Rule View besitzt im Gegensatz zum Graph View und Rule View keine Aktionen.

4 Beispielablauf

Im Folgenden wird ein Beispielablauf gezeigt, wie man den TGG-Editor benutzen kann. An simplen Beispielen werden die wichtigsten Funktionen gezeigt. Eine Übersicht aller Funktionen ist im Kapitel (3) zu finden.

4.1 Projekt anlegen

Als erstes muss ein Projekt samt Henshin-Datei angelegt werden.

1. Gehe zu *File* → *New* → *Project...* → *General* → *Project*.
2. Wähle einen beliebigen Projektnamen (z.B. "Editor") und erstelle das Projekt.

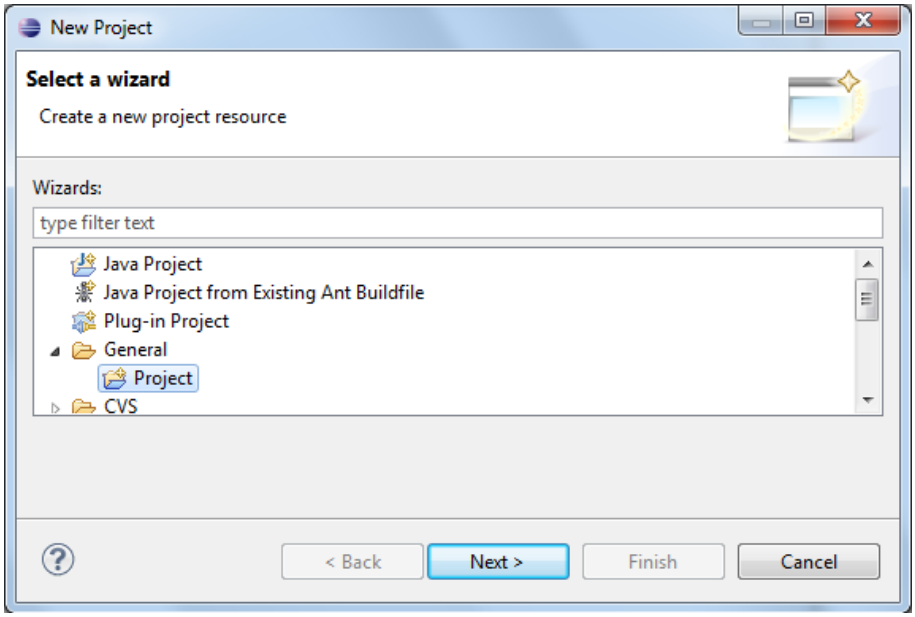


Abbildung 4.1: Erstellen eines Standard-Projektes

Füge nun dem neu erstellen Projekt ein TGG hinzu.

3. Gehe zu → *File* → *New* → *Other...* → *Other* → *TGG File Creation Wizard*

4. Wähle einen beliebigen Namen oder verwende den vorgegebenen.
   ACHTUNG! Die Dateiendung muss .henshin bleiben.
   Erstelle dann die TGG.

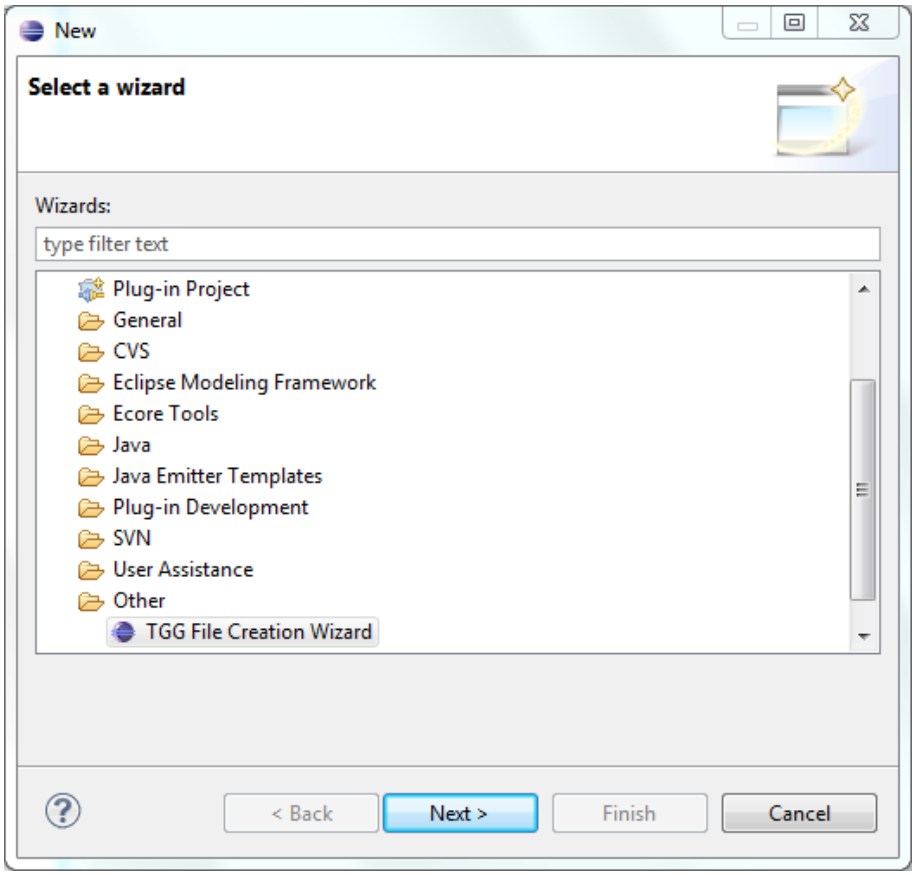


Abbildung 4.2: Erstellen einer TGG-Datei

Mit einem Doppelklick auf das Transformationssystem (im default-Fall heißt es "Transformation System") werden vier Ordner angezeigt, deren Funktionen im Folgenden beschrieben werden.

4.2 EMF Modelle importieren

EMF Modelle legen den Graphen zugrundeliegende Strukturen fest. Um einen kompletten Graphen zu erstellen, müssen drei Modelle importiert werden, je

eine für Elemente des Source-, des Target- und des Correspondence- Bereichs.

Es wird das Importieren eines Modells beschrieben. Der Import der anderen beiden Modell erfolgt analog.

4.2.1 EMF Projekt importieren

Damit der Editor die Referenzen zu dem EMF Modell speichert, muss zuvor noch ein Projekt importiert werden, dass entsprechende Modelle für alle drei Teilgraphen enthält.

1. Gehe zu *File* → *Import* → *Existing Projects into Workspace*
2. Wähle das EMF Projekt als zip-Archiv oder das Quellverzeichnis des Projekt und importiere das Projekt.

4.2.2 Import eines EMF Modells

1. Wähle den Ordner *Imports* und per Rechtsklicks auf den Ordner die Aktion *Import Source*
2. Wähle über *Workspace...* und das importierte EMF-Projekt, die entsprechende ECORE-Datei für die Source-Komponenten des TGG aus.

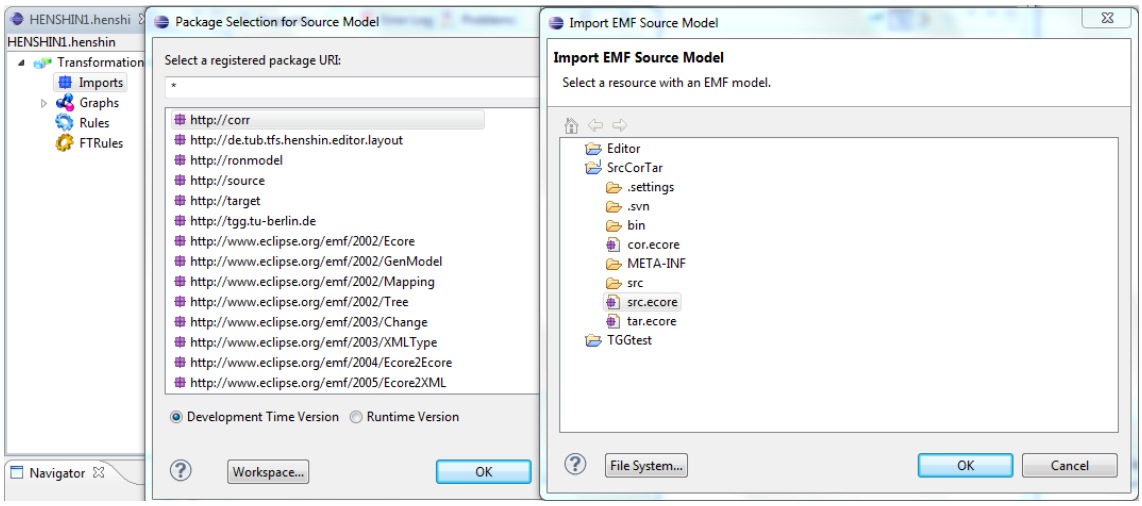


Abbildung 4.3: Importieren eines EMF Modells

Dies muss für Target- und Correspondence-Komponenten wiederholt werden.
Falls aus Versehen eine falsche Datei importiert wurde, kann der Vorgang mit der richtigen Datei wiederholt werden. Danach muss allerdings das Projekt geschlossen und neu geöffnet werden.

4.3 Graphen

Nun werden wir einen Graphen erstellen, an dem wir die ersten Editieroperationen zeigen werden.

4.3.1 Einen Graphen erstellen

1. Öffne über einen Rechtsklick auf den Ordner *Graphs* das Menü und führe die Aktion *Create Graph* aus.

2. Benenne den Graphen in "GraphExample".

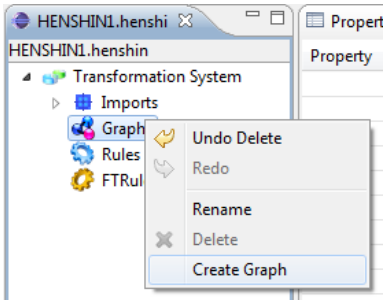


Abbildung 4.4: Erstellen eines Graphen

4.3.2 Graphen editieren

1. Ein Doppelklick auf den Ordner *Graphs* zeigt alle erstellten Graphen an.

2. Per Doppelklick auf einen Graphen wird die Ansicht des Graphen geöffnet und man erhält im rechten Rand eine Palette mit diversen Editieroperationen und einigen Aktionen in der Menüleiste des Graphen.

4.3.2.1 Knoten erstellen

Welche Typen von Knoten erstellt werden dürfen, ist in den EMF Modellen definiert.

1. Wähle in der Palette das Tool zum Erstellen eines Knoten vom Typ *ClassDiagram* aus.

2. Gehe mit der Maus in den Source-Bereich des Graphen und Erstelle einen Knoten mit einem Linksklick.

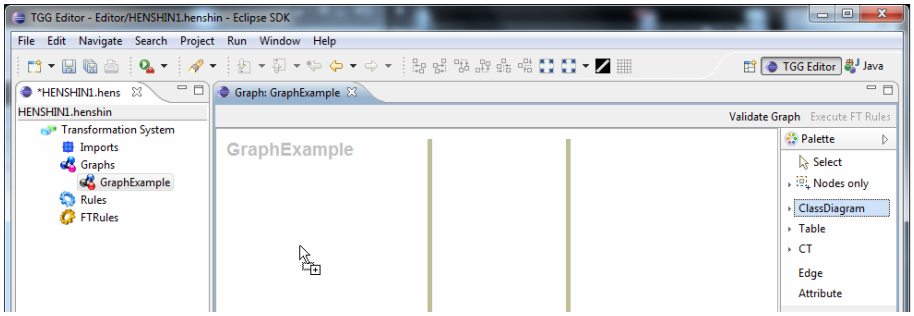


Abbildung 4.5: Erstellen eines Knoten

3. Klicke auf einen selektierten Knoten, um diesen umzubenennen (z.B. ”CD”).

4. erstelle zwei weitere Knoten, einen vom Typen *Database* im Targetbereich und einen vom Typen *CD2DB* im Correspondence-Bereich und benenne den *Database*-Knoten um (z.B. ”DB”).

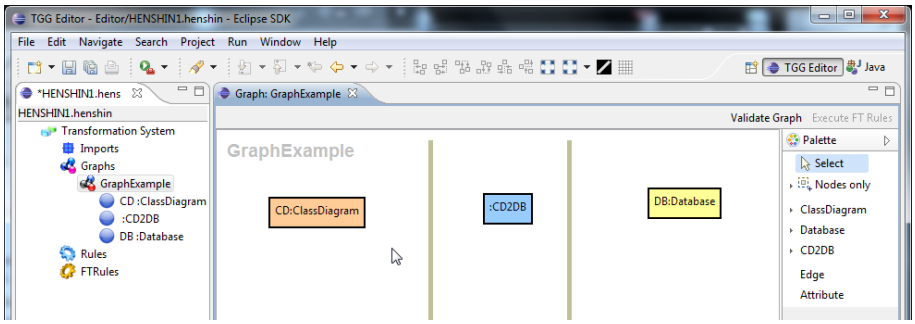


Abbildung 4.6: Ein Graph mit allen Root-Knoten

4.3.2.2 Kante erstellen

Kanten haben einen Quell- und einen Ziel-Knoten. Kanten werden immer vom Quell- zum Ziel-Knoten gezeichnet.
Es kann vorkommen, dass ein Knotentyp sowohl Ziel- als auch Quellknoten einer Kante sein darf.
Welche Typen von Kanten es gibt, wurde in dem EMF Modellen definiert.

1. Wähle in der Palette das Tool zum Erstellen einer Kante (Edge) aus.
2. Klicke den Knoten *:CD2DB* und danach den Knoten *DB:Database*.

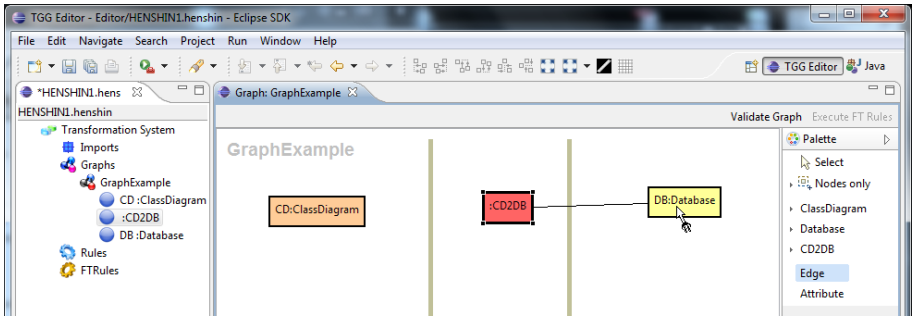


Abbildung 4.7: Eine Kante zwischen zwei Knoten ziehen

3. Erstelle eine zweite Kante und ziehe sie vom Knoten *CD2DB* zum Knoten *CD:ClassDiagram*.

4.3.2.3 Attribute erstellen

Nicht alle Knoten besitzen Attribute, so z.B. ClassDiagram- und Database-Knoten.
Da an den bisherigen Knoten keine Attribute erstellt werden können, wird diese Operation später noch gezeigt.
Welche Knoten Attribute besitzen und wieviele, ist ebenfalls in dem EMF Modellen definiert.

4.4 Regeln

4.4.1 Eine Regel erstellen

1. Über ein Rechtsklick auf den Ordner *Rules* lässt sich die Aktion *Create Rule* ausführen

2. Wähle einen beliebigen Namen für eine Regel (z.B. "RuleExample", der noch nicht vergeben ist.

4.4.2 Eine Regel editieren

Die grundlegenden Editieroperationen wurden schon im Kapitel 4.3.2 Graphen editieren erklärt. Nun werden spezielle Operationen für Regeln erklärt, für die allerdings noch einige Elemente in der Regel erstellt werden müssen, wie es schon für den Graphen gezeigt wurde.

- Erstelle dazu eine neue Regel, öffne sie und erstelle alle Elemente aus den Beispielen für den Graphen auch in der Regel.
- Erstelle drei weitere Knoten der Typen *Class*, *CT* und *Table*.
- Ziehe Kanten zwischen den eben erstellten Knoten ($CT \longrightarrow Class$ und $CT \longrightarrow Table$).
- Erzeuge noch zwei Kanten vom Knoten *CD:ClassDiagram* zum Knoten *:Class* und vom Knoten *DB:Database* zum Knoten *:Table*.

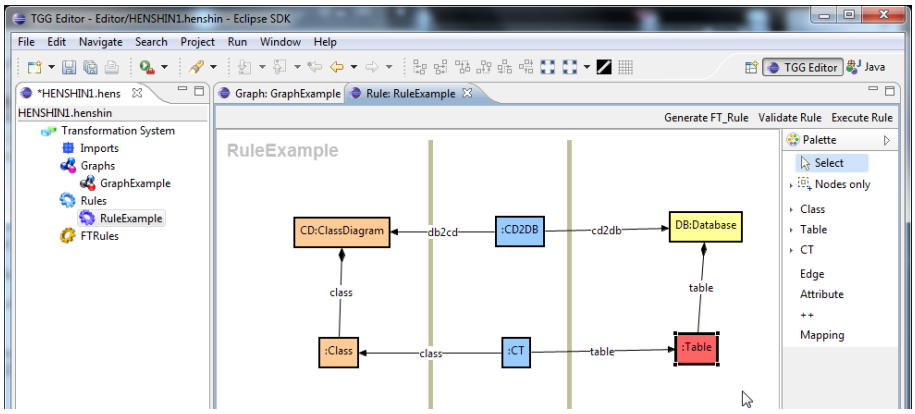


Abbildung 4.8: Ein Regel ohne als neu markierte Elemente

4.4.2.1 Element als *new* <++> markieren

1. Wähle in der Palette das Tool *++* um ein Element als neu zu markieren.
2. Klick auf die Knoten *:Class*, *:CT* und *:Table*.

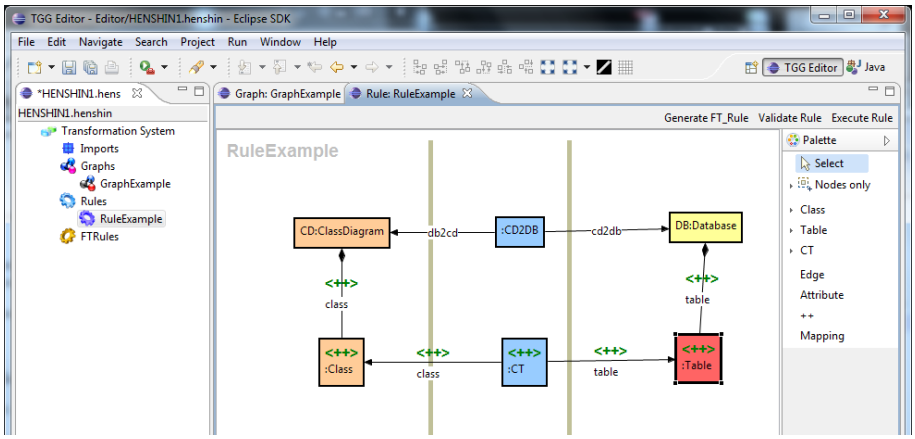


Abbildung 4.9: Eine Regel, die Elemente neu erzeugt

Hinweise:

- Markiert man einen Knoten mit <++>, werden alle anhängenden Kanten ebenfalls markiert.
- Demarkiert man einen Knoten werden alle Kanten ebenfalls demarkiert, es sei denn, der andere Knoten der Kante ist noch mit <++> markiert.
- Kanten können einzeln de-/markiert werden.

4.4.2.2 Parameter erstellen

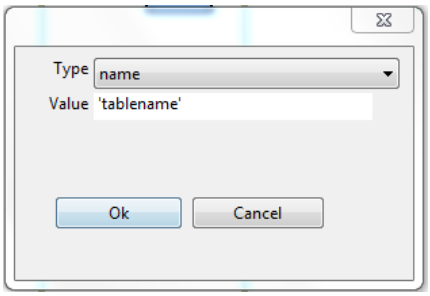
1. Wähle in der Baumansicht die Regel ”RuleExample” aus.
2. Aktiviere per Linksklick lässt sich die Aktion *Create Parameter*.
3. Gib dem Parameter einen Namen (z.B. ”cn”).

4.4.2.3 Attribute erstellen

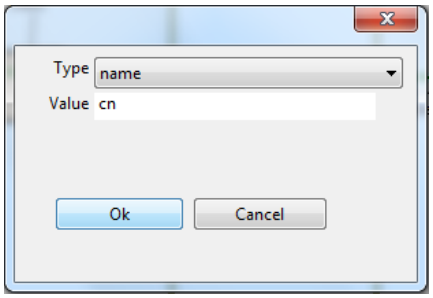
1. Wähle das Toole *Attribute* in der Palette und klicke auf den Knoten *:Table*.

2. Suche das Attribute *name* aus und gib ihm den Wert 'tablename'.

3. Füge dem Knoten *:Class* ein Attribute hinzu und nenne es cn (ohne die ").



(a) Den Wert eines Attributs sofort setzen



(b) Den Wert eines Attributs über einen Parameter setzen

Abbildung 4.10: Attribute erstellen

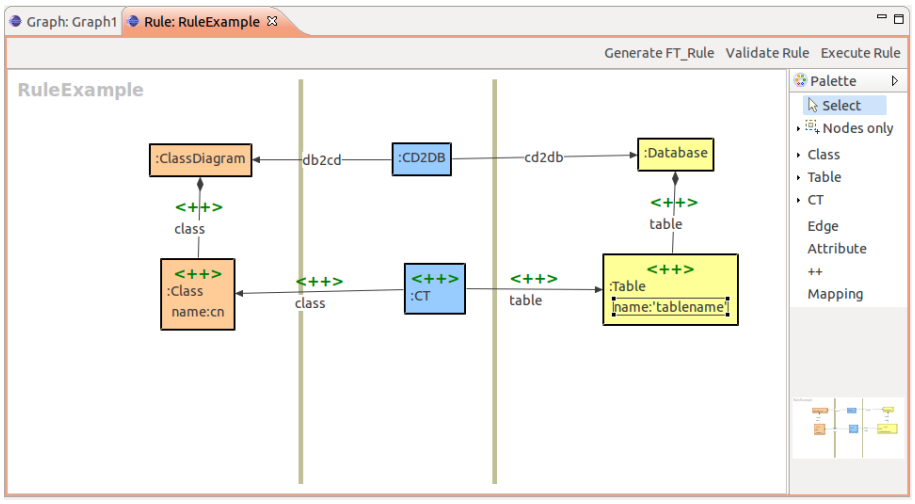


Abbildung 4.11: Eine gültige Regel

4.4.2.4 NAC erstellen

1. Öffne das Menü der Regel "RuleExample" und führe die Aktion *Create NAC* aus und benenne die NAC 'NAC1'.

2. Schließe die Regel "RuleExample" und öffne sie wieder. Jetzt erscheint die NAC in der Editor-View.

4.4.2.5 NAC editieren

Das Editieren einer NAC läuft nach dem gleichen Schema wie das Editieren eines Graphen ab. Siehe dazu 4.3.2 Graphen editieren.
Erstelle in der NAC einen Knoten vom Typ *Class*.

4.4.2.6 NAC-Mapping erstellen

1. Erstelle einen weiteren Knoten vom Typ *Class* in der Regel und benenne ihn "A".

2. Wähle in der Palette das Tool *Mapping* aus.

3. Markiere den *:Class*-Knoten in der Regel als Quell-Knoten des Mappings.

4. Markiere den Knoten aus der NAC als Ziel-Knoten.

Wie in Abbildung 4.12 "Ansicht einer Regel mit einem NAC-Mapping" abgebildet, werden NAC-Mappings durch eine simple Nummerierung angezeigt. Die Nummerierung geschieht automatisch nach dem Index des Knotens in der Liste aller Knoten im Graphen.

4.4.2.7 NAC-Mapping löschen

1. Selektiere den NAC-Knoten führe über das Menü die Aktion *Delete NAC Mappings* aus.

4.4.2.8 Knoten löschen

1. Selektiere den Knoten *A:Class* und führe über das Menü die Aktion *Delete* aus.

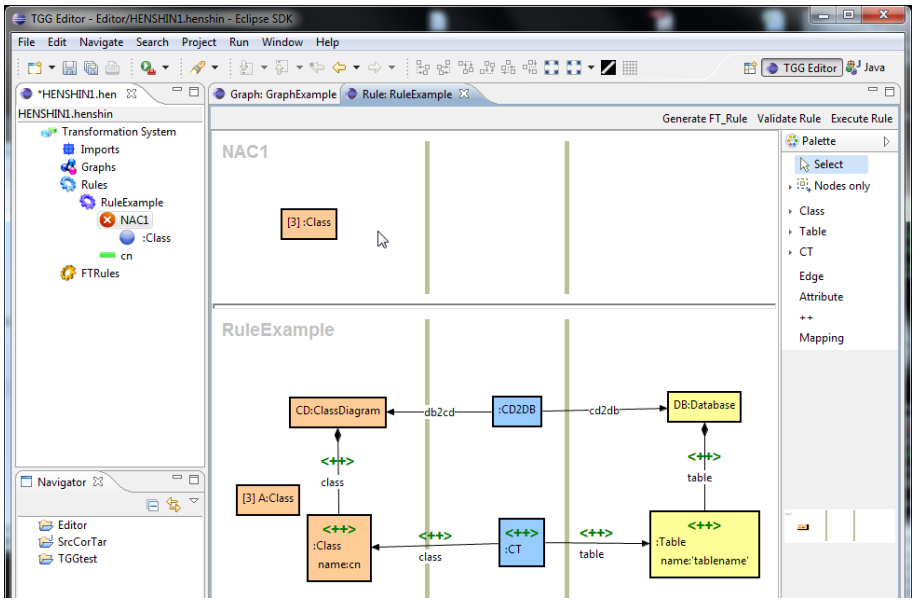


Abbildung 4.12: Ansicht einer Regel mit einem NAC-Mapping

4.4.2.9 NAC löschen

1. Selektiere die NAC der Regel in der Baumansicht und führe über das Menü die Aktion *Delete* aus.

2. Schließe und öffne die Regel neu, damit die Ansicht aktualisiert wird.

Sie sollte nun wieder so aussehen, wie in Abbildung 4.11 Eine gültige Regel gezeigt.

4.5 Forward-Translation-Regeln

4.5.1 FT_Regel generieren

1. Öffne die Regel "RuleExample".

2. Klicke in der Menüleiste der Regel den Button *Generate FT_Rule*.

4.5.2 FT_Regel editieren

Es wird nur gestattet, die NACs einer FT_Regel zu editieren (siehe NAC editieren) sowie die NAC-Mappings zwischen der FT_Regel und deren NACs.

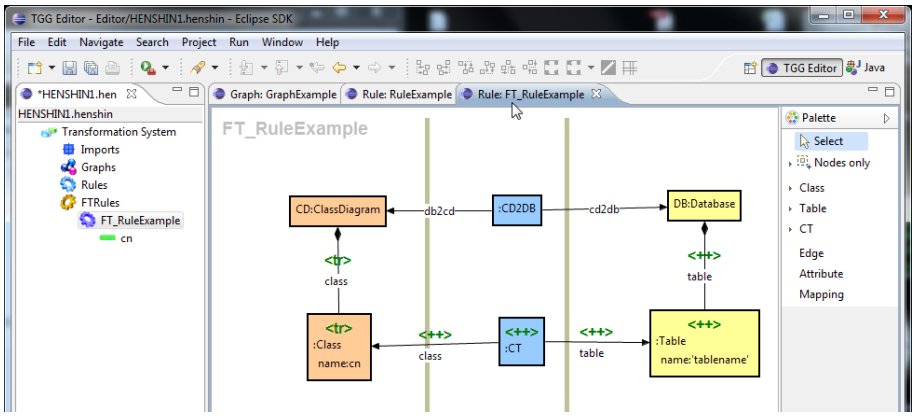


Abbildung 4.13: Ansicht einer FT_Regel

Um Änderungen an einer FT_Regel vorzunehmen, muss die entsprechende Regel geändert werden und daraus dann eine neue FT_Regel generiert werden (vorhandene FT_Regeln werden in diesem Fall überschrieben).

4.6 Regeln Ausführen

4.6.1 Graph validieren

Um eine Regel ausführen zu können, muss der Graph auf dem die Regel ausgeführt werden soll, valide sein.

1. Öffne den Graphen "GraphExample".
2. Klicke den Button *Validate Graph*.
3. Es erscheint eine Erfolgsmeldung, wenn alles korrekt war. Editiere den Graphen ggf. entsprechend der Hinweise, falls Fehler aufgetreten sind.

4.6.2 Regel validieren

Eine Regel muss ebenfalls valide sein.

1. Öffne den Graphen "RuleExample".
2. Klicke den Button *Validate Rule*.
3. Es erscheint eine Erfolgsmeldung, wenn alles korrekt war. Editiere die Regel ggf. entsprechend der Hinweise, falls Fehler aufgetreten sind.

4.6.3 Regel ausführen

1. Öffne die Regel ”RuleExample” und klicke den Button *Execute Rule*.

2. Setze den Wert des Parameters auf ’classname’.

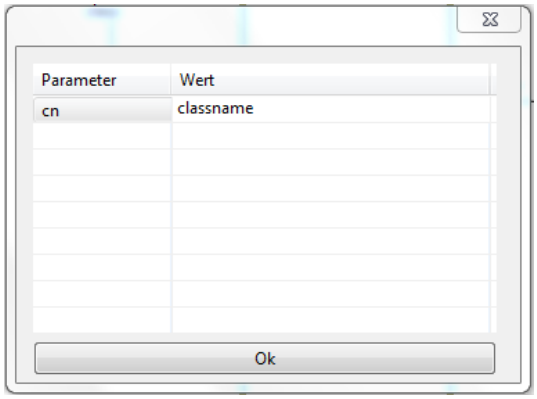


Abbildung 4.14: Das Setzen eines Parameterwertes beim Ausführen einer Regel

3. Öffne anschließend den Graphen ”GraphExample”.

Alle Elemente, die in der Regel mit <++> markiert wurden, sind in dem Graphen neu erstellt worden.

4.6.4 FT_Regel ausführen

1. Erstelle eine neue Regel, die Knoten von Typ *ClassDiagram*, *CD2DB* und *Database* sowie Kanten vom Typ *db2cd* und *cd2db* neu erstellt und generiere dazu eine FT_Regel.

2. Löschen in dem Graphen ”GraphExample” alle Elemente die nicht zum Source-Bereich des Graphen gehören.

3. Öffne das Menü des Graphen in der Baumansicht und führe die Aktion *Execute FT_Rules* aus

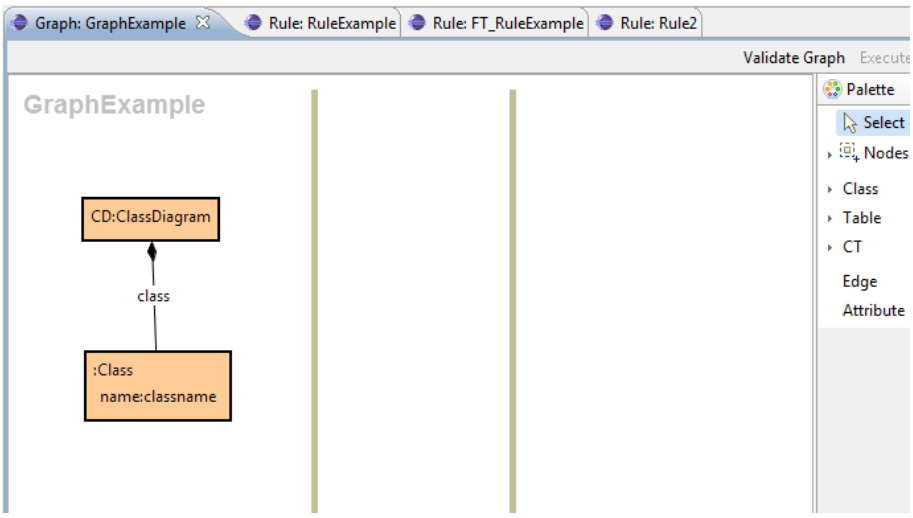


Abbildung 4.15: Aussehen eines Graphen für die Ausführung der FT_Regeln

5 Anmerkungen und Fehler

5.1 Direct Edit

Bei der Nutzung der Direct Edit Funktionalitäten vergrößert sich der Eingabebereich bei jedem Tastendruck. Dies scheint ein Bug in GEF zu sein und tritt auch nur unter Ubuntu auf. Bei längeren Eingaben kann dieser Fehler auch zum Absturz von Eclipse führen.

5.2 Rule View und FT Rule View

Bei der Anzeige von Regeln oder FT Regeln, die NACs enthalten, wird nur die NAC aber nicht die (FT) Regel beim öffnen nicht angezeigt. Dieser Fehler tritt nur unter Mac OSX auf und kann durch eine Größenänderung des Hauptfensters für den aktuellen View umgangen werden.