Fachhochschule
Dortmund
University of Applied Sciences and Arts

Erstellung von grafischen Modellierungswerkzeugen mit Eclipse Sirius

Seminar Modellgetriebene Softwareentwicklung

Denis Ayana

FH Dortmund

03.06.2018

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Thema der Seminararbeit

Diese Seminararbeit handelt von Eclipse Sirius und dessen Einordnung in die modellgetriebene Softwareentwicklung, sowie die Konzepte, Realisierung und Verwendung.

1.2 Ziel der Seminararbeit

Ziel der Seminararbeit ist es ein Verständnis für Eclipse Sirus zu entwickeln. Und Eclipse Sirius in die modelgetrieben Softwareentwicklung einzuordnen, sowie zu verstehen wie modellgetriebene Softwareentwicklung mit Hilfe von Eclipse Sirius funktioniert.

1.3 Modelle in der Softwareentwicklung

In der Softwareentwicklung kommen Modelle in zwei verschiedenen Rollen zum Einsatz. Die erste Rolle von Modellen ist die Rolle von Modellen zur Dokumentation. Hierbei werden die Modelle eingesetzt, um Softwaresysteme zu dokumentieren. Da diese Modelle von einem Programmierer zu meist erst durch die Implementierung von Code zu einer lauffähigen Anwendung werden, haben diese nur eine unterstützende Funktion. Oft werden diese Modelle, von Programmierern, als Overhead empfunden. Ein weiterer Nachteil diese Modelle ist es, dass sie oft an die dynamischen Softwaresysteme angepasst werden müssen. Da die Verbindung zwischen diesen Modellen und einer Anwendung nur gedanklicher Natur ist, müssen die Veränderung manuell von Programmieren nachgehalten werden. [1, S. 10] Die Zweite Rolle von Modellen ist die Rolle von Modellen zur Entwicklung. Dabei sind die Modelle mit dem Code einer Anwendung gleichzusetzen. So stehen die Modelle und der Code in einer formalen Beziehung zueinander. Die Umsetzung der Modelle in Code

erfolgt automatisiert. Für diese automatisierte Umsetzung werden bestimmte Compiler, Transformatoren oder Interpreter benutzt. Durch die enge Verbindung zwischen Modellen und Code lässt sich die Qualität und Wartbarkeit von Softwaresystemen verbessern.[1, S. 11]

1.4 Modelgetriebene Softwareentwicklung

Bei der modelgetriebenen Softwareentwicklung geht es darum, durch eine domänenspezifische Abstraktion der Realität, formale Modelle zu erstellen. Dieser formalen Modelle werden dann mit Hilfe von Compilern, Transformatoren oder Interpretern zu lauffähigem Code automatisiert umgesetzt. Diese automatisierte Umsetzung kann durch zwei mögliche Wege geschehen. Zum einen kann durch einen Generator aus den Modellen Code generiert werden, welcher dann in den nachfolgen Build-Prozess einfließt. Zum anderen können die Modelle durch einen entsprechenden Interpreter direkt zu einem lauffähigen Code interpretiert werden.

Oft werden die Modelle selbst durch Metamodelle beschrieben. Metamodelle werden wiederrum durch Metametamodelle definiert. Das heißt, dass ein Modell dem festgelegten Alphabet und der im Metamodell definierten Ordnung entsprechen muss. Metamodelle beschreiben durch eine abstrakte Syntax und eine statische Semantik die formalen Modelle. So müssen die formalen Modelle sich strikt an die benannten Konstrukte aus dem Metamodel halten. Die konkrete Syntax ist hierbei irrelevant. Auch die Semantik ist im Metamodell vorgegeben. Das Metamodell beschreibt somit, nach welchen Regeln das Modell als gültig bzw. als valide anzusehen ist. Das Metametamodell beschreibt auf einer noch abstrakteren Ebene das Metamodell. Aber unter denselben Gegebenheiten, wie das Metamodell das Modell beschreibt. Somit kann gesagt werden, dass ein Metametamodel eine Instanz eines Metamodells ist.

Die Metaisierung von Modellen könnte theoretisch so weiter gesetzt werden. Jedoch macht dies wenig Sinn, da das Metametamodell das Fundament der Metamodelarchitektur darstellt. Es beschreibt sich somit selbst.

Zusammenfassen lässt sich die Metaisierung in einer 4-Ebenen-Metamodelarchitektur beschreiben. An oberster Stelle das Metametamodel, welches die Grundlagen für die Metamodellarchitektur bildet und die Sprache des Metamodells beschreibt. Darunter liegt das Metamodell, welches eine Instanz des Metametamodells ist und die Sprache des formalen Modells beschreibt. An zweit unterster Stelle liegt das eigentliche formale Modell, welches eine domänenspezifische Abstraktion der Realität ist. Zuletzt, an unterster Stelle, das Benutzer-Objekt, welches keine Abstraktion, sondern einer genaue Abbildung der Realität ist.

1.5 Ziele modellgetriebener Softwareentwicklung

Es gibt eine Hand voll Gründe warum bei der Entwicklung von Software modelgetriebene Verfahren zum Einsatz kommen sollten:

- Automation:

  Durch die Automation kann mit Hilfe von modelgetriebener Softwareentwicklung eine Verbesserung der Entwicklungsgeschwindigkeit erreicht werden. Aus formalen Modellen kann durch einen oder mehrere aufeinander folgende Transformationsschritte letztendlich lauffähiger Code erzeugt werden. [1, S. 22]

- Wiederverwendbarkeit:

  Einmal definierte Architekturen, Modellierungssprachen und Transformationen können im Sinne einer Software-Produktionsstraße zur Herstellung diverser Softwaresysteme verwendet werden. Dies führt zu einem höheren Grad der Wiederverwendung und macht Expertenwissen in Softwareform in der Breite verfügbar. [1, S. 22]

- Handhabbarkeit:

  Ein weiteres wesentliches Potenzial ist die bessere Handhabbarkeit von Komplexität durch Abstraktion. Mit den Modellierungssprachen soll „Programmierung“ oder Konfiguration auf einer abstrakteren Ebene möglich werden. Die Modelle müssen dazu in einer möglichst problemorientierten Modellierungssprache ausgedrückt werden. [1, S. 23]

1.6 Domäne und domänenspezifische Sprache

1.6.1 Domäne

In der Softwareentwicklung beschreibt der Begriff der Domäne ein bestimmtes abzugenzerndes Gebiet. Eine Domäne kann sowohl fachlicher, als auch technischer Natur sein. In der modellgetrieben Softwareentwicklung werden diese Domänen meist durch ein entsprechendes Metamodel beschrieben. Das Metamodel definiert die Konzepte der Domäne. Fachliche Domänen ist zum Beispiel ein Webshop. Die technische Domäne ist dann zum Beispiel die Architektur des Softwaresystems.[2]

1.6.2 Domänenspezifische Sprache

Als domänenspezifische Sprache versteht sich eine formale Sprache oder auch Modellierungssprache der Softwareentwicklung. Diese Sprache beschreibt die Konstrukte der entsprechenden Domäne. Dabei ist sie einfacher und prägnanter als herkömmliche Program-

miersprachen. Um dies sicher zustellen nutzt die domänenspezifische Sprache das Vokabular der entsprechenden Domäne und eine Notation die Sachverhalte aus der Domäne in einer geeigneten Form darstellt. Die Notation der Sachverhalte einer Domäne erfolgt dabei in grafischer oder auch in textueller Darstellungsart.[2]

Kapitel 2

Eclipse Sirius

Auf der Eclipse Con 2013 wurde Eclipse Sirius erstmals, von Obeo, vorgestellt. Derzeit ist Eclipse Sirius ein Opensource Projekt der Eclipse Foundation. Eclipse Sirus ist ein Framework zur Entwickelung von domänenspezifischen Modellierungswerkzeugen. Dazu nutzt Eclipse Sirius das Eclipse Modeling Framwork(EMF) für das Erstellen und bearbeiten von Modellen, sowie das Grafical Modeling Framework(GMF) für die grafische Darstellung dieser Modelle. Durch die Verwendung von Eclipse Sirius ist es den Benutzern möglich verschiedene Modelle grafisch darzustellen. Grundlage dieser Modelle ist eine domänenspezifische Sprache(DSL). Diese domänenspezifische Sprache ist in Form eines Metamodels definiert.

2.1 Konzept

Wie schon erwähnt ist die Grundlage für das Arbeiten mit Eclipse Sirius ein Metamodell. Durch die Erstellung des Metamodells ist es möglich Programmcode zu generieren. Dieser generierte Programmcode wird dann für die Ausführung einer neuen Entwicklungsumgebung genutzt. In der neuen Entwicklungsumgebung ist es den Benutzern möglich mit konkrete Modell zu arbeiten. Den Benutzern stehen dazu eine Menge von Eclipse-Editoren zur Verfügung um EMF Modelle erstellen, bearbeiten und visualisieren zu können. Mit Hilfe von Eclipse Sirius können die Benutzer nicht nur Diagramme, sondern auch Tabellen, Matrizen oder Bäume zu Visualisierung der Modelle erstellen.

2.1.1 Diagramme

Die mit Eclipse Sirius erstellten Diagramme zeigen grafisch die Elemente der Modelle. Welche Elemente dargestellt werden und in welcher Art und Weise dies geschieht wird zuvor von den Benutzern angegeben. Die angegebenen Regeln definieren auch die Beziehungen zwischen den einzelnen Elementen der Modelle. Eine Beziehung kann sowohl

als Kante zwischen zwei Elementen oder als ein Element, welches sich in einem weiteren Element befindet, dargestellt werden. Wenn ein Diagramm mit einem EMF Modell synchronisiert ist, wird es automatisch angepasst, sobald eine Änderung an den Darstellungsregeln vorgenommen wird. Damit die Benutzer ein Modell direkt aus dem Diagramm heraus bearbeiten können, muss der Ersteller der Entwicklungsumgebung eine bestimmte Menge von Bearbeitungswerkzeugen bereitstellen. Diese Bearbeitungswerkzeuge dienen dann dazu, dass die Benutzer neue Objekte oder Beziehungen erstellen können. Des Weiteren kann auch das Verhalten bei Eingaben durch die Benutzer oder klassische Aktionen, wie das Festhalten oder Loslassen von Objekten, definiert werden. Um den Benutzern zu unterstützen, die potenzielle Komplexität ihrer Modelle zu meistern, bietet Eclipse Sirius mehrere Mechanismen, um sich auf relevante Elemente zu fokussieren.[3]

- Bedingtes Aussehen:

  Die grafische Darstellung kann abhängig von den Eigenschaften eines Objekts geändert werden. Zum Beispiel kann ein Objekt proportional zu dem Wert eines Attributs sein Größe oder Farbe in dem Diagramm verändern.

- Schichten und Filter:

  Durch das Benutzen von Schichten oder Filtern kann eine bestimmte Menge von Objekten aus einem Diagramm, abhängig von einer angegebene Bedingung, angezeigt oder versteckt werden.

- Validierung und Schnell-Reparatur:

  Das Modell kann durch die Angabe von spezifizierten Regel validiert werden. Die Probleme, welche bei der Validierung auftauchen, werden in einer eigenen Übersicht aufgelistet. Das dazugehörige Objekt des Diagramms wird dabei farblich markiert. Für einige Probleme wird eine Schnell-Reparatur vorgeschlagen, um das Modell zu korrigieren.

2.1.2 Tabellen und Matrizen

Modelle könne auch als Tabelle und Matrix dargestellt werden. Die erste Spalte der Tabelle enthält dabei das Objekt. In den darauf folgen Spalten werden die zum Objekt gehörenden Eigenschaften aufgelistet. In den Zellen stehen die entsprechenden Werte, welche für die Eigenschaften der Objekte angegeben werden. Die Benutzer können die Werte in den Zellen bearbeiten und neue Werte eintragen. Die Art der grafischen Darstellung kann auch, wie bei den Diagrammen schon beschrieben, von dem Ersteller der Entwicklungsumgebung definiert und geändert werden. Eine Matrix enthält zwei Listen von Objekten. Die erste Liste wird in der ersten Spalte, die andere Liste in der ersten Zeile abgebildet. Der Inhalt der Zellen beschreibt dabei das Zusammen spiel der beiden außenliegenden Objekte, einer jeden Liste.[3]

2.1.3 Bäume

Die dritte und letzte Art der grafischen Darstellung sind Bäume. Dabei werden die Objekte eines Modells in hierarchischer Anordnung abgebildet. Wie bei den Diagrammen, Tabellen und Matrizen kann die Art der grafischen Darstellung von dem Ersteller der Entwicklungsumgebung definiert und geändert werden.[3]

2.1.4 Viewpoints

Ein Viewpoint, zu Deutsch Aussichtspunkt, kapselt eine bestimmte Teil der Präsentation eines Modells. Um es den Benutzern einfacher zu machen sich auf einen bestimmten Teil einer Domäne zu konzentrieren, werden Viewpoints benutzt. Innerhalb des Viewpoint sind die Art und Weise der grafischen Dartstellung, sowie das Verhalten bei Benutzereingaben definiert und gespeichert. Ein Modell kann somit durch verschiedene Viewpoints visualisiert werden. Jeder dieser Viewpoints kann sich dabei auf einen von Ersteller definierten, abgegrenzten Bereich des Modells fokussieren.[3]

Kapitel 3

Realisierung und Verwendung

Im nachfolgenden Teil der Seminararbeit wird die Verwendung von Eclipse Sirius und die Realisierung einer eigenen Entwicklungsumgebung, zum Erstellen von grafischen Modellen, beschrieben. Dazu wird an Hand eines konkreten Beispiels jeder Schritt einzeln erklärt. Am Anfang wird ein Metamodel erstellt, welches die Domäne abbildet. Aufbauend auf dem Metamodel wird in einer eigenen Entwicklungsumgebung ein konkretes Model erstellt. Zum Schluss wird noch die Erstellung eigener Modellierungswerkzeuge, zum Hinzufügen von Objekten und deren Beziehungen zueinander, erklärt.

3.1 Erstellung des Metamodells

Zu nächst wir ein neues Modeling Project angelegt, welches unter dem Namen „fh.de.mdsd.example.hochschule“ abgespeichert wird. Innerhalb dieses Projekts wird das Metamodell erstellt. Dieses Metamodell wird die Grundlage der später erstellten grafischen Darstellungen und Modellierungswerkzeuge sein.

Das gewählte Beispiel beschreibt die Domäne einer Hochschule. Neben der Hochschule gibt es Personen, die entweder ein Dozent oder ein Student sind. Des Weiteren gibt es Module. In der nachfolgen Abbildung sind die einzelnen Klassen im Metamodel aufgeführt.

Eine Hochschule besteht aus Personen und Modulen. Zu jedem Modul gibt es einen oder mehrere Dozenten und Null bis mehrere Studenten. Ein Dozent kann ein beliebig viele Module haben. Jedoch kann ein Student null bis mehrere Module haben. Sowohl Student als auch Dozent haben eine Hochschule, zu der sie gehören. Die Hochschule besitzt als Attribute einen „name“ und eine „addresse“. Beide Attribute sind vom Datentype EString. Die abstrakte Oberklasse Person hat ein „name“ Attribut, welches auch vom Datentyp EString ist. Der Student besitz darüber hinaus noch eine „matrikelnr“ Attribut vom Datentyp EInt. Der Dozent hat ein „titel“ Attribut vom Datentyp EString. Das Modul hat ein „bezeichnung“ Attribut vom Datentyp EString.

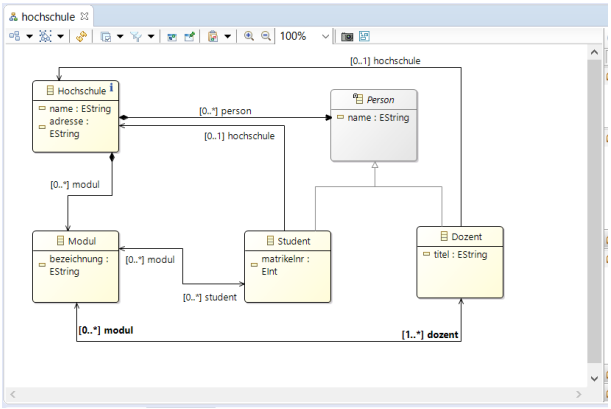

Abbildung 3.1: Metamodell der Domäne mit Attributen und Beziehungen

3.2 Workflow

Sobald die Erstellung des Metamodells abgeschlossen ist kann daraus Programmcode generiert werden. Durch einen Rechtsklick in die Oberfläche des Metamodells öffnet sich ein Kontextmenü. In dem Kontextmenü muss der Punkt „Generate -> All“ ausgewählt werden. Durch diese Aktion werden mehrere JAVA-Klassen und zwei weitere Projekte generiert.

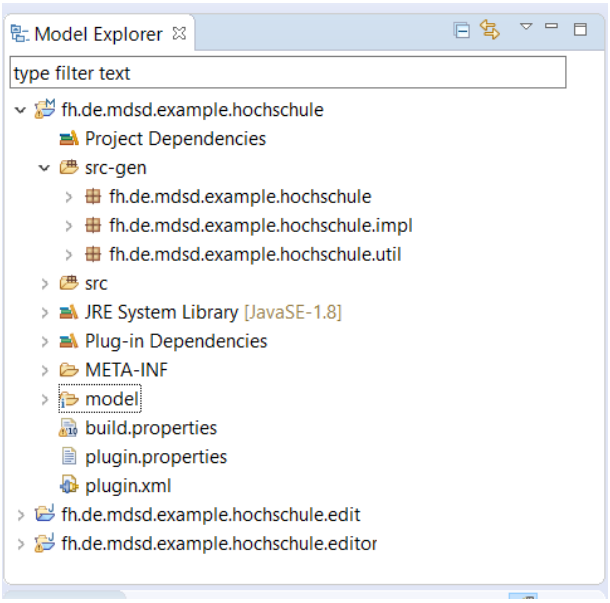

Abbildung 3.2: Generierter Code und generierte Projekte

Wurde die Einstellung einer neuen Run-Configuration vorgenommen, kann durch das Klicken auf den Run-Button eine neue Entwicklungsumgebung gestartet werden. In dieser Entwicklungsumgebung ist es nun möglich Modelle zu erzeugen, zu bearbeiten und neue Modellierungswerkzeuge zu erstellen. Bevor mit der Bearbeitung von Modellen und die Erstellung von Modellierungswerkzeugen begonnen werden kann, muss erst ein konkretes Abbild der Domäne erstellt werden. Dazu wird innerhalb der neu gestarteten Entwick-

lungsumgebung ein ebenfalls ein neues Modeling Projekt erstellt, welches Fachhochschule_Dortmung genannt wird. Ist dieses Projekt erstellt muss durch ein Rechtsklick auf das Projekt ein neues Hochschul Modell erstellt werden. Diese ist nach dem Rechtsklick unter New->Other->Example EMF Model Creation Wizards->Hochschule Model zu finden. Diese Modell enthält später die konkreten Objekte, welche in der Domäne der Hochschule vorhanden sein werden. Dort werden dann die Studenten, Dozenten und Module gespeichert, welche von den Benutzern erstellt werden. Nach dem Auswählen des Hochschul Modells muss angegeben werden, welches Objekt der Domäne zunächst erstellt werden soll. Hier wird die Hochschule ausgewählt. Und der Erstellungsprozess kann mit dem Klick auf Finish beendet werden.

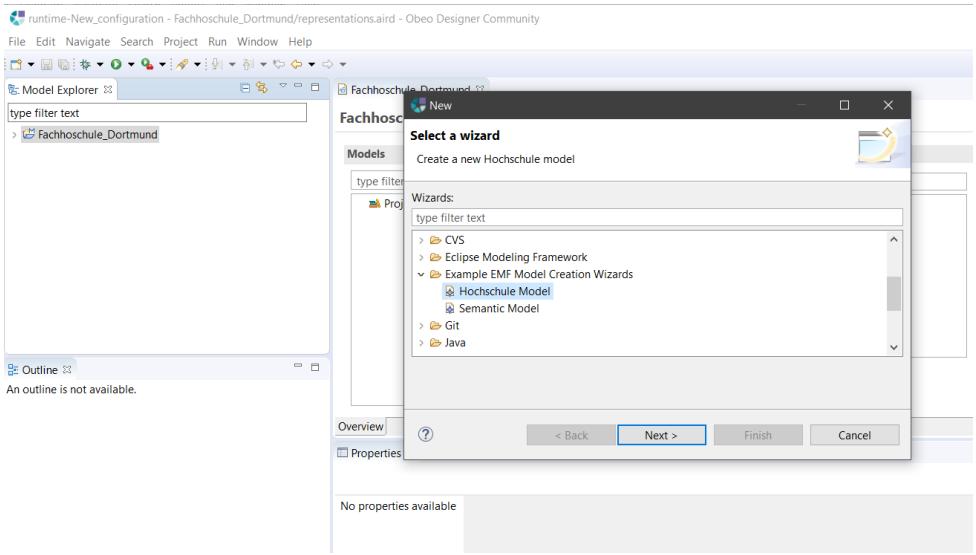

Abbildung 3.3: Auswahl des Metamodelles für die Erstellung eines neuen Projektes

3.2.1 Erstellung der Objekte

Nun kann mit der Erstellung der einzelnen Objekte innerhalb der Hochschule begonnen werden. Zu nächst wird ein Student erstellt. Dies kann durch den Rechtsklick auf das erstellte Hochschulobjekt und dann über New Child->Student erreicht werden. Das Erstellen von Dozenten und Modulen läuft äquivalent zur Erstellung von Studenten ab, nur das hier jeweils Dozent bzw. Modul gewählt werden muss. Nach der Erstellung von einem Studenten, zwei Dozenten und zwei Modulen, kann mit der Bearbeitung der Attribute begonnen werden. Der Student erhält als Wert für das Attribut Name „Denis“ und als Wert für das Attribut Adresse „Lünen“. Für das Attribut Matrikelnr. wird der Wert „7095563“ und für das Attribut Hochschule wird die „Hochschule Fachhochschule Dortmund“ eingestellt. Die Module werden erst nach der Bearbeitung, der Attribute der Module, dem Studenten zugeordnet. Der erste Dozent erhält als Wert für das Attribut Name den Wert „Kamsties“ und für das Attribut Titel den Wert „Prof. Dr.“. Auch hier wird, wie bei dem Studenten, die „Hochschule Fachhochschule Dortmund“ eingestellt. Da

die Module noch keinen Bezeichner besitzen wird die Bearbeitung dieser Attribute bei den Dozenten nachgeholt. Der zweite Dozent erhält als Wert für das Attribut Name den Wert „Vollmer“, für das Attribut Titel den Wert „Prof. Dr.“ und für die Hochschule wird auch hier „Hochschule Fachhochschule Dortmund“ eingestellt. Das erste Modul erhält für das Attribut Bezeichnung deren Wert „Modellgetrieben Softwareentwicklung“. Das zweite Modul bekommt für das Attribut Bezeichnung den Wert „Mobile App Engineering“. Das Attribut Dozent wird für das erste Modul auf den Dozenten Kamsties eingestellt. Für das zweite Modul wird das Attribut Dozent auf den Dozenten Vollmer eingestellt. Zum Schluss werden jeweils das Attribut Hochschule für beide Module auf „Hochschule Fachhochschule Dortmund“ festgelegt und die Module für das Attribut Modul des Studenten eingestellt.

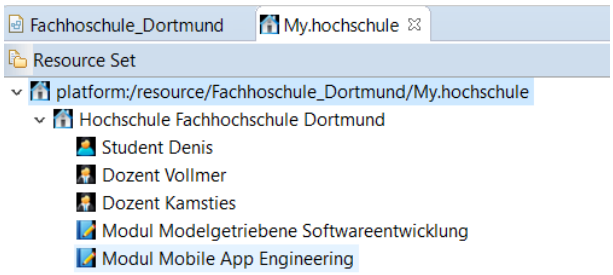

Abbildung 3.4: Objekte des Diagrammes

3.2.2 Grafische Darstellung der Objekte

Für die Erstellung der grafischen Darstellung der Objekte wird ein Viewpoint Specification Project angelegt. Dieses Projekt enthält dann alle Informationen über die grafische Darstellung der Objekte innerhalb des Diagramms und der Erstellungswerkzeuge für neue Objekte. Für den Namen des Viewpoint Specification Projects wählen wir „hochschule.design“. Durch das Klicken auf Finish wird die Erstellung eines Viewpoints beendet. Daraufhin öffnet sich in der Entwicklungsumgebung eine neue Oberfläche. Dort ist ein Viewpoint für die Hochschule vorhanden. Dieser wird von uns nun in „Hochschule“ umbenannt.

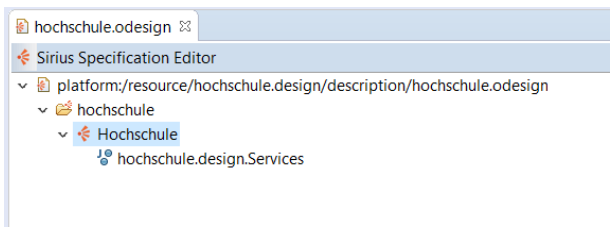

Abbildung 3.5: Viewpoint mit der Id Hochschule

Bevor wir mit der Erstellung der grafischen Darstellung der Elemente des Diagramms fortfahren können, werden erst die Dependencies, zu Deutsch Abhängigkeiten, für das Modell eingestellt. Hierbei wird in der MANIFEST.MF Datei das zuvor erstellte Metamodel-Projekt eingebunden. Über den Button Add öffnet sich das Suchfenster, wo wir durch die Eingabe des Wortes „hochschule“ nach dem Metamodel-Projekt suchen. Das gefundene Projekt wird dann durch das Klicken des OK-Buttons zu den Anhängigkeiten hinzugefügt.

Nach dem Hinzufügen der Abhängigkeit beginnen wir die Erstellung des Diagramms. In der Oberfläche, wo der Viewpoint Hochschule definiert wurde, fügen wir durch einen Rechtsklick auf den Viewpoint Hochschule ein neues Diagramm hinzu. Als Metamodell wählen wir das von uns definierte Hochschul Metamodel. Dieses Diagramm erhält als ID „Hochschule Diagramm“. Als wert für das Attribut Domain Class wird „hochschule::Hoschule“ eingetragen. Hier bei ist auf die korrekte Groß- und Kleinschreibung zu achten, da es sonst zu Komplikationen kommen kann. Des Weiteren ist darauf zu achten das bei Initialization ein Hacken gesetzt ist, da das Diagramm sonst nicht automatisch angezeigt wird.

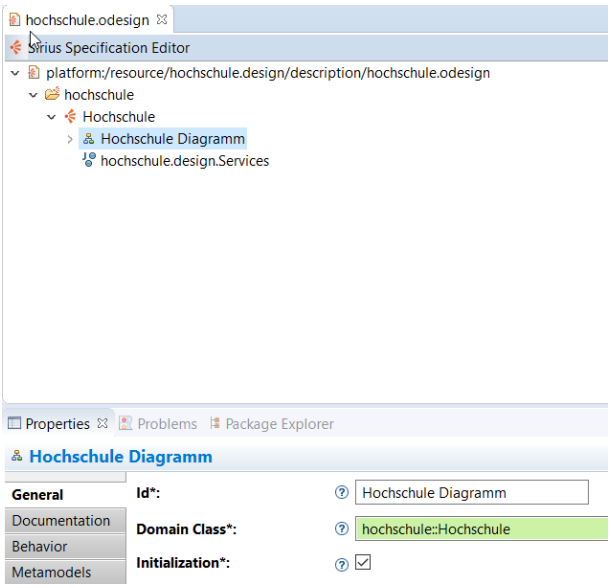

Abbildung 3.6: Diagramm des Viewpoints mit Id und Domain Class

3.2.2.1 Objekte im Diagramm

Nun können wir uns der Erstellung der grafischen Darstellung der Studenten, Dozenten, Module und der Beziehungen zwischen den Objekten zu wenden. Unterhalb des erstellten Hochschul Diagramms befindet sich bereits ein Default Layer. Innerhalb dieses Layers werden die Definitionen für die Nodes, zu Deutsch Knoten, und der Edges, zu Deutsch Kanten, erstellt. Beginnen wir mit der Erstellung eines Studenten Nodes. Dazu fügen wir dem Default Layer, über Rechtsklick-> New Diagramm Element->Node, ein neuen Node hinzu. Der Student-Node erhält als Id den Wert „StudentNode“, die Domain Class

definieren wir als „hochschule::Student“ und die für die Semantic Candidates Expression wird der Wert als „feature::person“ definiert. Damit der Student in dem Diagramm auch angezeigt wird noch ein Square dem Node hinzugefügt. Dies erfolgt über einen rechtsklick auf den StudentNode->New Style->Square. Grundsätzlich ist für die Label Expression, also dem anzuzeigenden Namen des Objekts, „feature::name“ angegeben. Bei Bedarf kann dieser Wert auch angepasst werden. Als Color für das Square wird „green“ angegeben. Für die Höhe wird der Wert auf 5 und für die Breite auf 10 eingestellt. Ähnlich wie bei dem Student-Node verfahren wir bei der Erstellung des Dozenten-Node und dem Modul-Node. Der Unterschied ist jedoch, dass beim Dozenten-Node die Domain Class „hochschule::Dozent“ und beim Modul-Node die Domain Class „hochschule::Modul“ eingestellt wird. Auch diesen beiden Nodes fügen wir eine Square als Style hinzu. Der Dozent-Node erhält die Farbe Orange für das Square und der Modul-Node die Farbe Rot. Bei dem Square des Modul-Nodes wird für die Label Expression der Wert „feature:bezeichnung“ eingetragen, damit der Name des Moduls angezeigt werden kann.

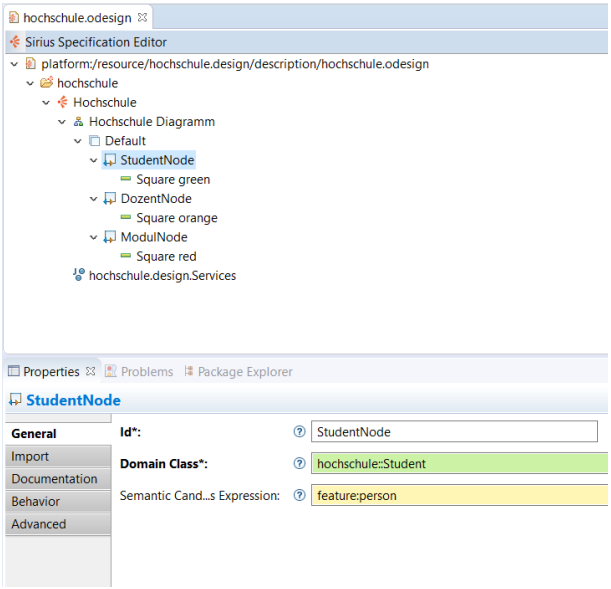

Abbildung 3.7: Erstellte Nodes mit Sqaures

3.2.2.2 Beziehungen im Diagramm

Als nächstes folgt die Erstellung der grafischen Dartstellung der Beziehungen zwischen den Objekten. Die Edges oder auch zu deutsch Kanten werden ähnlich wie die Nodes erstellt. Auch sie werden über Rechtsklick auf den Default Layer, dann New Diagramm Element->Realtion Based Edge, hinzugefügt. Die erste Kante bekommet die Id „Modul-ToStudentEdge“. Sie stellt die Beziehung zwischen einem Modul und einem Studenten grafisch dar. Für das Source Mapping wird der Modul-Node und für das Target Mapping der Student-Node eingestellt. Das Source Mapping bestimmt, von welchem Objekt aus die Kante beginnt. Das Target Mapping bestimmt hingegen das Ziel der Kante. Zu Letzt geben wir für die Target Finder Expression den Wert „feature:student“ an. Damit die

Kante in dem Diagramm grafisch dargestellt werden kann, wird ihr ein Style hinzugefügt. Dies geschieht ähnlich, wie das Hinzufügen eines Squares bei einem Node. Für den Style wird ein Edge Style verwendet. Die Einzige Konfigurationsänderung ist, dass die Farbe der Kante auf „red“ eingestellt wird. Dieser Ablauf wird für folgende Kanten mit den entsprechenden Werten wiederholt:

- Kantenname: „StudentToModulEdge“; Source Mapping: StudentNode; Target Mapping: ModulNode; Target Finder Expression: „feature:modul“; Edge Style Color: green
- Kantenname: „DozentToModulEdge“; Source Mapping: DozentNode; Target Mapping: ModulNode; Target Finder Expression: „feature:modul“; Edge Style Color: gray
- Kantenname: „ModulToDozentEdge“; Source Mapping: ModulNode; Target Mapping: DozentNode; Target Finder Expression: „feature:dozent“; Edge Style Color: orange

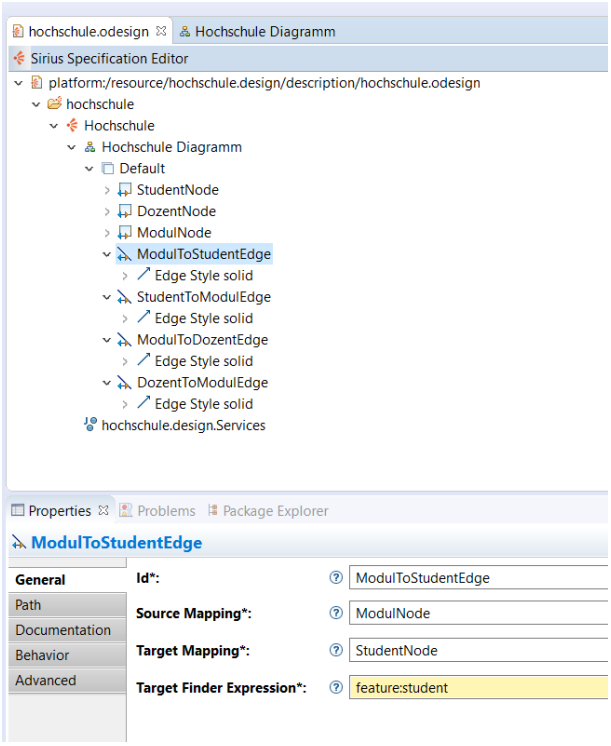

Abbildung 3.8: Erstellte Beziehungen zwischen den Objekten

Um das Diagramm zu testen wird dem zuvor erstellten Modeling Project eine Viewpoint Selection hinzugefügt. Über einen Rechtsklick auf das Projekt Fachhochschule_Dortmund wird dann über Viewpoint Selection der Viewpoint „hochschule“ ausgewählt und mit OK bestätigt. Damit dem Hochschul Modell ein Diagramm hinzugefügt wird, muss durch ein Rechtsklick auf Hochschule Fachhochschule Dortmund->New Representation->new Hochschul Diagramm das entsprechende Diagramm aus dem Viewpoint ausgewählt werden.

Darauf hin wird automatisch ein Diagramm erstellt und in der Oberfläche der Entwicklungsumgebung angezeigt.

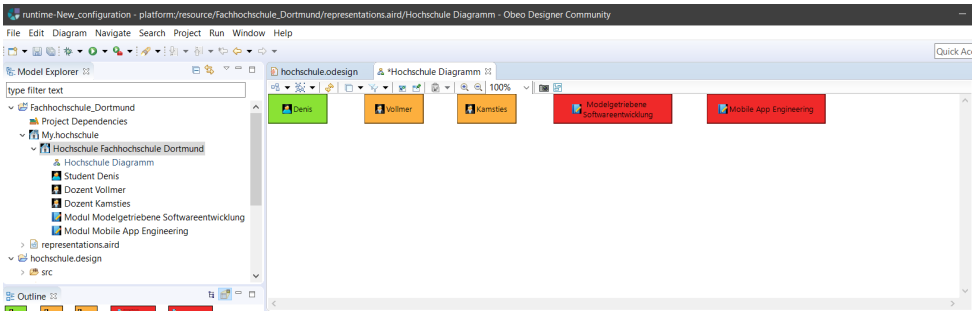

Abbildung 3.9: Vollständiges Diagramm mit Objekten und Beziehungen

3.2.3 Entwicklung grafischer Modellierungswerkzeuge

Da es beider Erstellung eines solchen Modellierungswerkzeuges auch sinnvoll ist, dass aus Diagramm heraus neue Objekte hinzugefügt werden können, werden zusätzliche Werkzeuge entwickelt. Diese Werkzeuge könne dann von den Benutzern eingesetzt werden, um z.B. einen neuen Studenten oder ein neues Modul, so wie einen Dozenten zu erstellen. Des Weiteren wird ein Werkzeug entwickelt, welches den Benutzern die Möglichkeit bietet, zwischen den Objekten Beziehungen zu erstellen.

3.2.3.1 Objekterstellung

Für die Entwicklung der genannten Werkzeuge wird zunächst eine Sektion für die Werkzeuge erstellt. Diese befindet sich, wie auch die Nodes und Edges, unterhalb des Default Layers. Diese Section wird über einen Rechtsklick auf den Default Layer->New Tool->Section angelegt. Innerhalb dieser Sektion wird, über Rechtsklick->New Element Creation->Node Creation, eine neues Werkzeug zum Anlegen von Nodes erstellt. Dieses Werkzeug bekommt die Id „createStudent“ und als Node Mapping wird StudentNode eingetragen. Unterhalb der Node Creation befindet sich eine Begin Action. Dieser Action wird, über Rechtsklick->New Operation->Change Context, ein Change Context hinzugefügt. Für die Browse Expression wird der Wert „var:container“ angeben. Diese Expression sorgt dafür das wir mit dem im Context definierten Container, welcher hier die Hochschulinstanz ist, arbeiten können. Innerhalb dieses Change Context wird, über Rechtsklick->New Operation-Create Instance, eine neue Create Instance erzeugt. Diese Create Instance wird für den Reference Name mit dem Wert „person“ und für den Type Name mit dem Wert „hochschule::Student“ definiert. Wiederum die Create Instance bekommt, über Rechtsklick->New Operation->Set, eine Set Operation hinzugefügt. Die Set Operation erhält für den Feature Name den Wert „name“ und für die Value Expression wir der Wert „aql:‚student‘+container.person->filter(hochschule::Student)->size()“ eingetragen. Der Wert der Value Expression sorgt dafür, dass bei dem Hinzufügen eines weiteren

Studenten der angezeigte Name die Zeichenkette „student“ plus die Anzahl aller vorhandenen Studenten im Diagramm angehängt bekommt. So erhält der nächste hinzugefügte Student den Namen „student2“, bei einem weitern „student3“ und bei noch einem weitern „student4“.

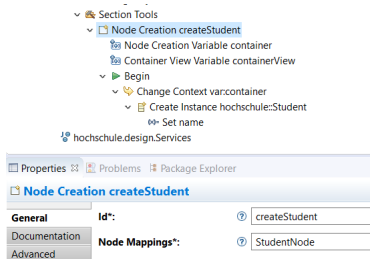

Abbildung 3.10: Node Creation „createStudent“ mit Id und Node Mapping

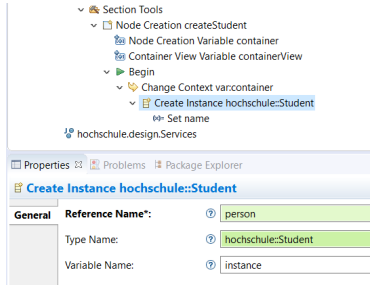

Abbildung 3.11: Create Instance mit Reference Name und Type Name

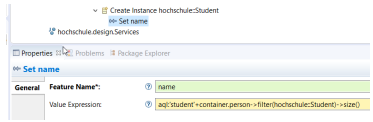

Abbildung 3.12: Set Operation mit Feature Name und Value Expression

Für die Erstellung weiterer Nodes sind im Folgenden die Werte, für die Erstellung des Werkzeuges, aufgelistet.

- Node Creation:
  1. Id: „createDozent“ Node Mapping: DozentNode
  2. Change Context: Browse Expression: „var:container“
  3. Create Instace Reference Name: „person“ Type Name: „hochschule::Dozent“
  4. Set Operation: Feature Name: „name“ Value Expression: „aql:‘dozent‘+container.person->filter(hochschule::Dozent)->size()“
- Node Creation:
  1. Id: „createModul“ Node Mapping: ModulNode
  2. Change Context: Browse Expression: „var:container“
  3. Create Instace: Reference Name: „modul“ Type Name: „hochschule::Modul“
  4. Set Operation: Feature Name: „bezeichnung“ Value Expression: „aql:‘modul‘+container.modul->filter(hochschule::Modul)->size()“

3.2.3.2 Beziehungserstellung

Da es zwischen den Objekten auch Beziehungen gibt, muss es auch eine Möglichkeit geben diese durch ein Modellierungswerkzeug zu erstellen. Die Erstellung dieser Werkzeuge läuft ähnlich, wie die Erstellung der Node Creation, ab. Zu nächst wird der Section ein Edge Creation Tool, über Rechtsklick auf Section->New Element Creation-> Edge Creation, hinzugefügt. Diese Edge Creation bekommt als Id den Wert „setStudentToModul“ und für das Edge Mapping wird die Egde StudentToModul ausgewählt. Auch hier wird in der Begin Action ein Change Context erstellt. Der Change Context erhält für die Browse Expression den Wert „var:source“ und nicht „var:target“, wie bei der Erstellung der Node Creation. Dem Change Context wiederum wird direkt eine Set Operation hinzugefügt. Die Set Operation hat für den Feature Name den Wert „modul“ und für die Value Expression den Wert „var:target“. Äquivalent zu der Erstellung des „createStudentToModul“ verläuft die Erstellung des „createDozentToModul“ Werkzeuges ab. Hierbei ist zu beachten das an Stelle des StudentToModulEdge die DozentToModulEdge verwendet wird. Die Restlichen Werte werden, wie bei der Erstellung der „setStudentToModul“ Edge Creation, verwendet.

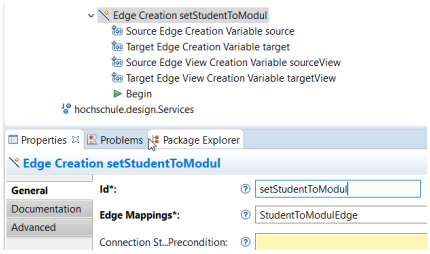

Abbildung 3.13: Edge cration „setStudentToModul“ mit Id und Edge Mapping

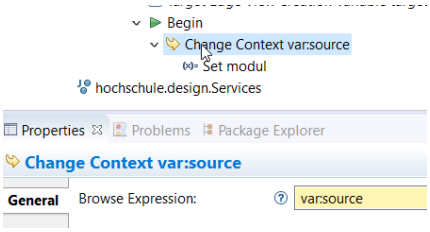

Abbildung 3.14: Change Context mit Browse Expression

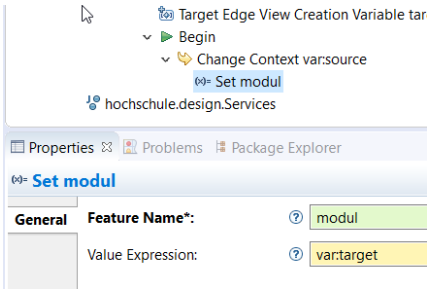

Abbildung 3.15: Set Operation mit Feature Name und Value Expression

3.2.3.3 Anzeige der Modellierungswerkzeuge

Nun da die Erstellung der Modellierungswerkzeuge abgeschlossen ist, können die Benutzer direkt Objekte und Beziehungen erstellen. Diese Modellierungswerkzeuge werden standardmäßig rechts neben dem Diagramm angezeigt.

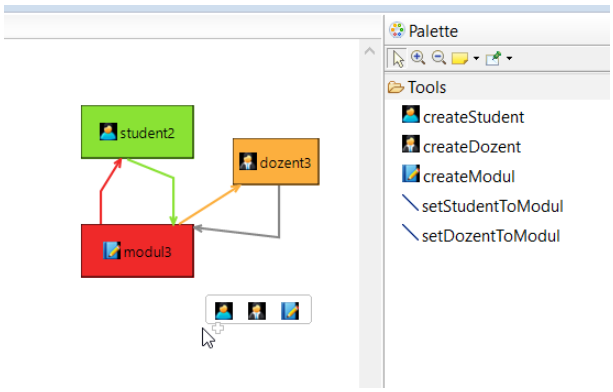

Abbildung 3.16: Erstellte Werkzeuge im Editor

Kapitel 4

Tooling und Installation

Für die Benutzung von Eclipse Sirius braucht man nicht viel Software und noch weniger Konfigurationsaufwand. Die Firma Obeo, welche für die Entwicklung von Eclipse Sirius verantwortlich ist, bietet auf Ihrer Website eine vorkonfigurierte Entwicklungsumgebung, den Obeo Designer, an. Der Obeo Designer basiert auf der herkömmlichen Entwicklungsumgebung Eclipse. Allerdings enthält der Obeo Designer schon Eclipse Sirius und die benötigten Frameworks, welche für ein reibungsloses Arbeiten benötigt werden. Deshalb wird in diesem Abschnitt nicht erklärt, wie man Eclipse Sirius in ein herkömmliches Eclipse einbindet. Es ist an dieser Stelle das Einfachste, um Zeit und Aufwand zu sparen, den Obeo Designer herunterzuladen und zu benutzen.

4.1 Tipps und Tricks

In den folgenden Abschnitten wird kurz erklärt, wie bei der Verwendung von Eclipse Sirius aufgetretene Fehler behoben werden können. Da, wie im oberen Abschnitt beschrieben, keine eigene Installation bzw. Einbindung von Eclipse Sirius stattgefunden hat, ist dieses Kapitel relativ kurzgehalten. ### Probleme mit der MANIFEST.MF Bei der Einbindung von einigen Abhängigkeiten kam das Problem auf, das einige Pakete nicht aufgelöst werden konnten. Dieses Problem ließ sich relativ einfach beheben. Dazu wird unter Window->Preferences->Plug-in Development->Target Platform der Hacken von der aktiven „running Platform“ entfernt und das Ganze mit Apply and Close gespeichert. Danach wird der Hacken auf dem gleichen Weg wieder gesetzt und das Ganze gespeichert. Es ist darauf zu achten, dass eine Running Plattform gewählt wird, welche nicht mit einem Roten „X“ versehen ist. Daraufhin hat das Auflösen der Pakete ohne Probleme funktioniert. ### Bilder der Elemente im Diagramm ändern Das Bild für das Hochschulelment wird unter /name.des.projekts.editor/icons/full/obj16/HocchschuleModelFile.gif gespeichert. Die Größe des Bildes beträt 16x16 Pixel. Die Weiteren Bilder befinden sich unter /name.des.projekts.edit/icons/full/obj16. Diese sind jweis als Dozent.gif, Student.gif,

Hochschule.gif und Modul.gif gespeichert. Auch diese Bilder haben eine Größe von 16x16 Pixel. Diese Bilder können einfach durch eigene Bilder ersetzt werden. es ist jedoch darauf zu achten das die neuen Bilder den gleichen entsprechenden Dateinamen haben.

Kapitel 5

Zusammenfassung

Eclipse Sirius ist ein mächtiges Framework um Modellierungswerkzeuge zu erstellen. Es bietet nicht nur die Möglichkeit eine Domäne grafisch durch die Unterstützung von Diagrammen, Tabellen, Matrizen oder Bäumen darzustellen. Sondern auch einen eigenen konkreten Blick auf diese definiert Domäne zu werfen, sie besser zu verstehen und in ihr zu arbeiten. Die Erstellung eigener Node oder Edge Creations hilft dem Benutzer die Modellierung einer Domäne in kürzester Zeit vorzunehmen. Obwohl das Beispiel in dieser Seminararbeit relativ simple gehalten wurde, kann mit Eclipse Sirius eine sehr spezielle und komplexe Domäne, auf eine einfache und leicht verständliche Art und Weise grafisch dargestellt werden. Das Schwierigste bei der Erstellung einer solchen Entwicklungsumgebung ist sehr wahrscheinlich die Erstellung des Metamodells. Da dies die Grundlage der konkreten Modelle ist. Das zusammen bauen der Entwicklungswerkzeuge ist zum Teil nur ein Zusammenbasteln und Konfigurieren von vorgefertigten Elementen. Was ein großer Vorteil für den Benutzer ist. So kann dieser sich voll und ganz auf die Spezifikation der Domäne fokussieren.

Literatur

[1] T. Stahl, M. Völter, und J. Bettin, *Modellgetriebene softwareentwicklung: Techniken, engineering, management*, 1 // 1. Aufl. Heidelberg: dPunkt; dpunkt-Verl., 2005 [Online]. Verfügbar unter: http://voelter.de/data/books/mdsd_de1a.pdf. [Zugegriffen]

[2] L. Corneliussen und M. Völter, „Kurze rede, langer sinn“, *DotNetPro*, Nr. 4, 2009 [Online]. Verfügbar unter: http://voelter.de/data/articles/KurzeRedeLangerSinn.pdf. [Zugegriffen]

[3] C. Brun und S. Bonnet, „What is sirius¿‘ 2013 [Online]. Verfügbar unter: http://www.eclipse.org/community/eclipse_newsletter/2013/november/article1.php. [Zugegriffen]