HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Object-Graph-Visualization

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autoren:	Simon Gwerder und Adrian Rieser
Betreuer:	Thomas Letsch
Experte:	Prof. Dr. Martin Zimmermann
Gegenleser:	Ivan Bütler

Abstract

Für Einsteiger in die objektorientierte Softwareentwicklung ist der Zusammenhang von Klassen und den dazugehörigen Instanzen oft schwierig zu verstehen. Das Ziel der Bachelorarbeit «Object-Graph-Visualization» ist es, eine Applikation zu entwickeln, um Personen das Verhalten von OO-Klassen und den dazugehörigen OO-Instanzen besser erklären zu können. Diagramme, wie das UML Klassen- und Objektdiagramm, sowie der Objektgraph sind in der objektorientierten Entwicklung weit verbreitet, werden aber nicht kombiniert dargestellt. Die Idee ist, die Diagramme im dreidimensionalen Raum zu platzieren. Dazu ist das Klassendiagramm zweidimensional in der xy-Ebene und das Objektdiagramm oder der Objektgraph auf der z-Achse darüberliegend anzuordnen. Die Diplom- und Vorgängerarbeit «3D-Class-Object-Visualization» (3DCOV) konnte diese geforderte Funktionalität erfüllen. Jedoch war die Applikation durch die gewählten Bibliotheken und Frameworks stark an das Betriebssystem gebunden.

Für das GUI und das 3D-Rendering wird JavaFX eingesetzt. Dieses Framework stellt Plattform-unabhängigkeit sicher und erlaubt so, möglichst viele Benutzer zu erreichen. Mittels JavaFX Scene Builder werden die Menüstrukturen und Oberflächen der Klassen- und Objekteboxen modelliert. Um die Architektur ausbaufähig zu halten, wurde auf eine starke Layertrennung geachtet und viel Wert auf das MVC Pattern gelegt.

Das neue Visualisierungstool «Object Graph Visualizer» (OGV) ermöglicht die Gestaltung eigener 3D-Diagramme. Diese können im XML-Format persistiert werden. Ausserdem wurde eine XMI-Schnitt-stelle realisiert. Klassendiagramme, die in anderen Applikationen, wie zum Beispiel dem Enterprise Architect entworfen wurden, können importiert werden. Es ist zudem möglich, einfach zwischen Objektdiagramm- und Objektgraph-Modus umzuschalten. Der OGV soll künftig in den Grundlagen-fächern der Informatik zum Einsatz kommen und das Zusammenspiel von Klassen und deren Objekten auf eine Art visualisieren, wie es bisher nicht möglich war.

Management Summary

Ausgangslage

Das Ziel der Bachelorarbeit «Object-Graph-Visualization» ist es, eine Applikation zu entwickeln, um Personen das Verhalten von OO-Klassen und den dazugehörigen OO-Instanzen besser erklären zu können.

Für Einsteiger in die objektorientierte Softwareentwicklung ist der Zusammenhang von OO-Klassen und den dazugehörigen OO-Instanzen oft schwierig zu verstehen. Die UML-Diagramme wie Klassen- und Objektdiagramme sind weit verbreitet und haben sich in der objektorientierten Entwicklung als Standard etabliert. Eine Verbindung zwischen Klassen- und Objektdiagramm ist jedoch nicht ersichtlich. Zusätzlich zu den beiden UML Diagrammen gibt es noch einen weiteren Diagrammtyp, welcher für diese Bachelorarbeit besonders interessant ist: Der Objektgraph. Dieser zeigt – ähnlich wie ein Objektdiagramm – zu einem bestimmten Zeitpunkt die instanziierten Objekte des Programms, ist aber bei den dazugehörigen Referenzen detailreicher. Eine Kombination von Klassen- und Objekt-diagramm, beziehungsweise Klassendiagramm und Objektgraph soll als dreidimensionales Diagramm dargestellt werden. Dazu ist das Klassendiagramm zweidimensional auf der X- und Y-Achse zu platzieren und das Objektdiagramm oder der Objektgraph auf der Z-Achse darüberliegend.

Die Diplom- und Vorgängerarbeit «3D-Class-Object Visualization (3DCOV)» aus dem Jahr 2007 konnte diese geforderte Funktionalität erfüllen. Jedoch war die Applikation durch die gewählten Bibliotheken und Frameworks, wie Java3D, stark an das Betriebssystem gebunden und konnte nur mit einigen Aufwand auf unterschiedlichen Systemen ausgeführt werden. Zudem ist die nun achtjährige Applikation bezüglich eingesetzten Technologien und der Gestaltung der Benutzeroberfläche mittels Swing nicht mehr aktuell.

Vorgehen / Technologien

Als Vorgehensmodell zur Softwareentwicklung wurde Rational Unified Process (RUP) gewählt.

Für das GUI und das 3D Rendering wird JavaFX eingesetzt. Dieses Framework stellt Plattformunab-hängigkeit sicher und erlaubt so möglichst viele Benutzer zu erreichen. Mittels JavaFX Scene Builder werden die Menüstrukturen und Oberfächen der Klassen und Objekten im Object Graph Visualizer modelliert. Um die Architektur ausbaufähig zu halten, wurde auf eine starke Layertrennung geachtet und viel Wert auf das MVC Pattern gelegt.

Ergebnis

Das neue Visualisierungstool «Object Graph Visualizer» (OGV) lässt sich nicht nur durch die Toolbar im Header bedienen, sondern gleich in der Ansicht direkt über Kontextmenüs. Die selbst erstellten Diagramme können im XML-Format persistiert werden. Die Persistenz-Schnittstelle wurde ausserdem verwendet, einen XMI-Import zu realisieren. Klassendiagramme, die von anderen Applikationen entworfen wurden, wie zum Beispiel dem Enterprise Architect, können in den OGV eingelesen werden. Es ist zudem möglich, einfach zwischen Objektdiagramm- und Objektgraph-Modus umzuschalten.

Der OGV soll künftig in den Grundlagenfächern der Informatik zum Einsatz kommen und das Zusammenspiel von Klassen und deren Objekten auf eine Art visualisieren, wie es bisher nicht möglich war. Zusätzlich kann es in der weiteren Ausbildung, bei privaten und Industrieprojekten unterstützend eingesetzt werden, um komplexe Strukturen abzubilden oder zu erklären.

Aufgabenstellung

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL INFORMATIK

Aufgabenstellung Bachelorarbeit FS 2015

Object-Graph-Visualization

Studenten

- Simon Gwerder
- Adrian Rieser

Einführung

Im Bereich der objektorientierten Software-Entwicklung soll zu Ausbildungszwecken ein Tool gebaut werden, welches Objekt-Graphen im 3-dimensionalen Raum visuell und interaktiv darstellen kann.

Aufgabenstellung

Erstellung eines entsprechenden Tools, welches obige Fähigkeit besitzt und im Weiteren:

Als Ausgangslage soll jeweils ein UML-Klassendiagramm dienen, welches in der X/Y-Ebene dargestellt wird.
Dieses kann entweder interaktiv im Tool erstellt oder als XMI importiert werden.

Von diesem Klassendiagramm sollen sodann interaktiv Objekte erstellt werden können.
Dabei sollen 2 Varianten unterstützt werden:

1. UML-Objektdiagramm
   Die Objekte werden wie in einem UML-Objektdiagramm dargestellt, wobei die Platzierung im 3-dimensionalen Raum jeweils automatisch in der Vertikalen (Z-Ebene) über der entsprechenden Klasse erfolgt.

2. Java-Objektdiagramm
   Zusätzlich werden auch Java-Referenzen und Arrays von Referenzen dargestellt.

Die Instantiierung der Objekte soll interaktiv erfolgen können.
Ebenso das Setzen von Attributen und Beziehungen (Assoziationen) zu anderen Objekten.

Thomas Letsch 16.02.2015 1/3

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Aufgabenstellung Bachelorarbeit **FS 2015**

Der Beobachtungspunkt im Raum soll zur Laufzeit beliebig gewählt werden können.

Im Weiteren sollen optional folgende Erweiterungen untersucht werden:

1. Bau einer Programmier-Schnittstelle (API), über welche das Tool remote gesteuert werden kann (z.B. Instantiierung eines Objektes, etc.)

2. Darstellung des Objekt-Graphen einer (anderen, beliebigen) Java-Applikation

Unter Berücksichtigung von aktuellen Software-Engineering-Methoden soll ein geeigneter Entwicklungsprozess definiert und darauf basierend das Tool entsprechend gebaut werden.

Technologien

- Java, JavaFX
- Enterprise Architect

Generelles

- Die Vorgaben der Abteilung Informatik [1] sind einzuhalten, insbesondere die Anleitung zur Dokumentation [2].
- Die "Generelle Richtlinien für Studien- und Bachelorarbeiten" [3] sind einzuhalten.
- Mit dem CASE-Tool Enterprise Architect ist ein UML-Modell zu führen, welches synchron mit den Programm-Sourcen und der Projekt-Dokumentation ist.
- Ein Java-Entwickler muss mit der Projekt-Dokumentation in die Lage versetzt werden, die Applikation in Betrieb zu nehmen und weiter entwickeln zu können.

Termine

- Montag, 16.02.15 Beginn der Bachelorarbeit
- Freitag, 12.06.15 12:00 Uhr Abgabe der Bachelorarbeit

Betreuung

- Betreuer
  Thomas Letsch
  tletsch@hsr.ch
  055 - 22 24 567 (HSR Büro 5.204); 055 - 214 43 50 (Geschäft)
- Besprechungen
  Wöchentliche Besprechung jeweils Freitag 17:10 Uhr

HSR HOCHSCHULE FÜR TECHNIK RAPPERSWIL
INFORMATIK

Aufgabenstellung Bachelorarbeit **FS 2015**

Referenzen

[1] www.hsr.ch>HSR-intern>Bachelor-Studiengänge>Informatik>Allgemeine Infos Bachelor- und Studienarbeiten
https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html
[2] DokuAnleitungBA_SA_140210.pdf
[3] "Generelle Richtlinien für Studien- und Bachelorarbeiten"
(v1.6 / 26.01.2015, Thomas Letsch)

Rapperswil, 16.Februar 2015

Thomas Letsch

Inhaltsverzeichnis

Teil I – Technischer Bericht

1. Einführung

1.1. Vision

Die UML-Diagramme, wie Klassen- und Objektdiagramme, sind weit verbreitet und haben sich in der objektorientierten Softwareentwicklung als Standard etabliert. Sie können verwendet werden, um domänenspezifische Probleme zu erklären oder auch zur Darstellung von OO-Designpattern. Neben den UML-Diagrammen existieren noch weitere Diagrammtypen, für diese Bachelorarbeit vor allem interessant ist der Objektgraph. Dieser zeigt, ähnlich wie ein Objektdiagramm, zu einem bestimmten Zeitpunkt des Programms die instanziierten Objekte und zusätzlich die dazugehörigen Erreichbarkeiten.

Die Kombination von Klassen- und Objektdiagramm oder Objektgraph kann man als dreidimensionales Diagramm darstellen. Dazu ist das zweidimensionale Klassendiagramm auf der X- und Y-Achse zu platzieren und das Objektdiagramm bzw. der Objektgraph auf der Z-Achse darüberliegend. Dieser innovative Diagrammtyp soll mit Hilfe einer Applikation erstellt werden können.

Die Diplom- und Vorgängerarbeit «3D-Class-Object-Visualization» aus dem Jahr 2007 konnte zwar diese geforderten Aufgaben mit dem Klassen- und Objektdiagramm sinngemäss erfüllen, jedoch war die Applikation durch die gewählten Bibliotheken und Frameworks stark an das Betriebssystem gebunden und konnte nur mit einigen Fixes auf unterschiedlichen Systemen ausgeführt werden. Zudem ist die nun achtjährige Applikation bezüglich eingesetzten Technologien und der Gestaltung der Benutzeroberfläche nicht mehr aktuell.

Dies soll nun mit der Bachelorarbeit «Object-Graph-Visualization» geändert werden. Statt Swing und Java 3D soll neu JavaFX verwendet werden. Die restlichen funktionalen Anforderungen können für diese Bachelorarbeit übernommen und um den Objektgraphen erweitert werden. Ebenfalls übernommen werden kann die Anforderung zur enthalten XMI-Importfunktion. Mithilfe dieser können die zum Beispiel im Enterprise Architekt entworfenen Klassendiagramme eingelesen werden.

Die resultierende Applikation dieser Arbeit soll künftig in den Grundlagenfächern der Informatik zum Einsatz kommen und das Zusammenspiel von Klassen und deren Objekten auf eine Art visualisieren, wie es bisher nicht möglich war. Zusätzlich kann es in der weiteren Ausbildung, bei privaten und Industrieprojekten unterstützend eingesetzt werden um komplexe Strukturen abzubilden oder zu erklären.

Abbildung 1: Das Logo des Object Graph Visualizer

1.2. Aufgabenstellung und Rahmenbedingungen

Diese Arbeit wird im Rahmen einer Bachelorarbeit (BA) an der Hochschule für Technik Rapperswil (HSR) durchgeführt. Die Aufgabenstellung ist vorgegeben und kann auf Seite 4 im Detail eingesehen werden. In Folge der wöchentlichen Projektbesprechungen mit dem Betreuer wurden die in der Aufgabenstellung erwähnten Begriffe definiert:

Die erste Variante entspricht nicht nur dem UML-Objektdiagramm, sondern auch dem UML-Klassendiagramm. Die zweite Variante «Java-Objektdiagramm» wurde als «Objektgraph» erkannt und so benannt.

Lieferobjekte

Es ist eine eigenständige Applikation zu entwickeln, welche die gewünschten Anforderungen erfüllt. Die Applikation läuft unter dem Arbeitstitel «Object Graph Visualizer» (OGV).

Rahmenbedingungen

- Programmiersprache: Java
- Framework: JavaFX
- Mit dem CASE-Tool Enterprise Architect ist ein UML-Modell zu führen, welches synchron mit den Programm-Sourcen und der Projekt-Dokumentation ist.
- Die Studenten entscheiden sich nach Rücksprache mit dem Betreuer für eine SW-Entwicklungsmethodik. Die Meilensteine werden mit dem Betreuer vereinbart.
- User Interface, Source Code, Code-Kommentare, Versionsverwaltung und Installationsanleitungen sind in Englisch. Alles andere ist deutsch.

1.3. Aufbau der Arbeit

Die Dokumentation dieser Arbeit ist in vier Teile gegliedert:

Teil	Beschreibung
Teil I – Technischer Bericht	Der erste Teil der Dokumentation gibt eine Einführung in das Projekt, zeigt die Aufgabenstellung und Bedingungen, den Stand der Technik, die Umsetzung und Resultate der Arbeit.
Teil II – SW-Projektdokumentation	Dieser Teil beinhaltet Software-Engineering-spezifische Dokumentationen. Dazu gehören Analyse mit Anforderungsspezifikation, Domain, Packages und Klassen, Implementation und Testing.
Teil III – Projektplanung	Hier werden das Management und das Monitoring des Projekts aufgeführt.
Teil IV – Anhang	Der Anhang umfasst: Glossar, Quellenverzeichnis, Eigenständigkeitserklärung, Lizenzvereinbarung, eine persönliche Reflektion, Protokolle, das Poster und ein Instruction Manual (in Englisch).

2. Stand der Technik

2.1. Bestehende Lösungsansätze und Begriffe

2.1.1. Klassendiagramm

Das Klassendiagramm in der Unified Modeling Language (UML) erklärt die Struktur eines Systems mithilfe der Klassen, deren Attribute und Relationen untereinander. Es bildet die Basis der objektorientierten Entwicklung und kann auch in der Datenmodellierung eingesetzt werden. Klassendiagramme werden in fast allen Informatikmodulen an der HSR eingesetzt und sind ein wichtiges Lerninstrument für Einsteiger in der objektorientierten Programmierung. Im Object Graph Visualizer dient das Klassendiagramm als Grundstein für alle Modelle. Klassendiagramme sind jedoch statisch und sagen wenig über das Laufzeitverhalten eines Systems aus.

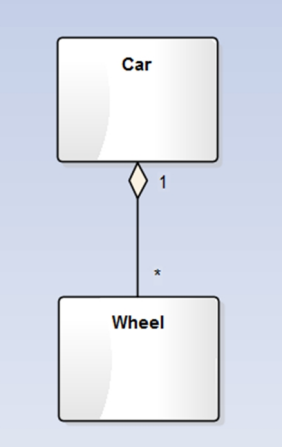


Abbildung 2: Beispiel eines Klassendiagramms

2.1.2. Objektdiagramm

Das Objektdiagramm ist ebenfalls in der Unified Modeling Language geschrieben. Auch dieser Diagrammtyp zeigt die Struktur eines Systems, allerdings zu einem spezifischen Zeitpunkt der Laufzeit. Von den im Klassendiagramm beinhalteten Klassen sind im Objektdiagramm die Objekte und dessen Relationen sichtbar. Die Attribute dieser Objekte können nun mit den momentanen Werten dargestellt werden. Eine direkte Verbindung mit einem Klassendiagramm ist aber nicht ersichtlich.

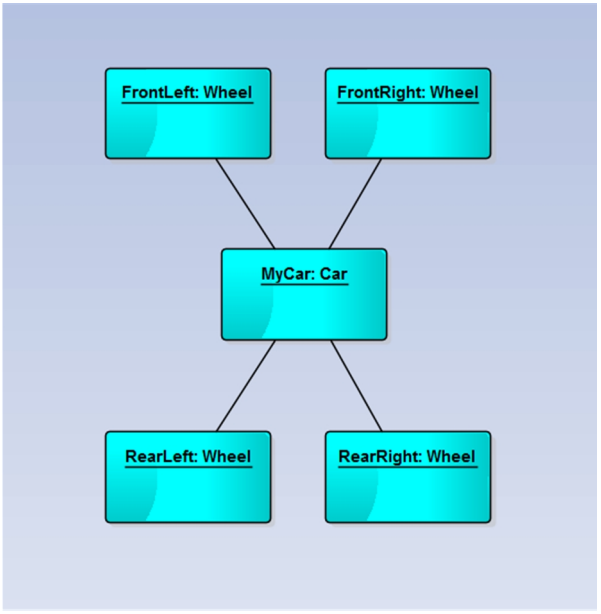


Abbildung 3: Beispiel eines Objektdiagramms

2.1.3. Objektgraph

Ebenfalls wie das Objektdiagramm zeigt der Objektgraph das System zu einem spezifischen Zeitpunkt der Laufzeit. Anstatt die Relationen des Objektdiagramms werden im Objektgraph Referenzen dargestellt. Deshalb kann er auch als Erreichbarkeitsgraph verstanden werden. Mit diesem Diagrammtyp können die tatsächlich erreichten Objekte über die Referenzen aufgezeigt werden.

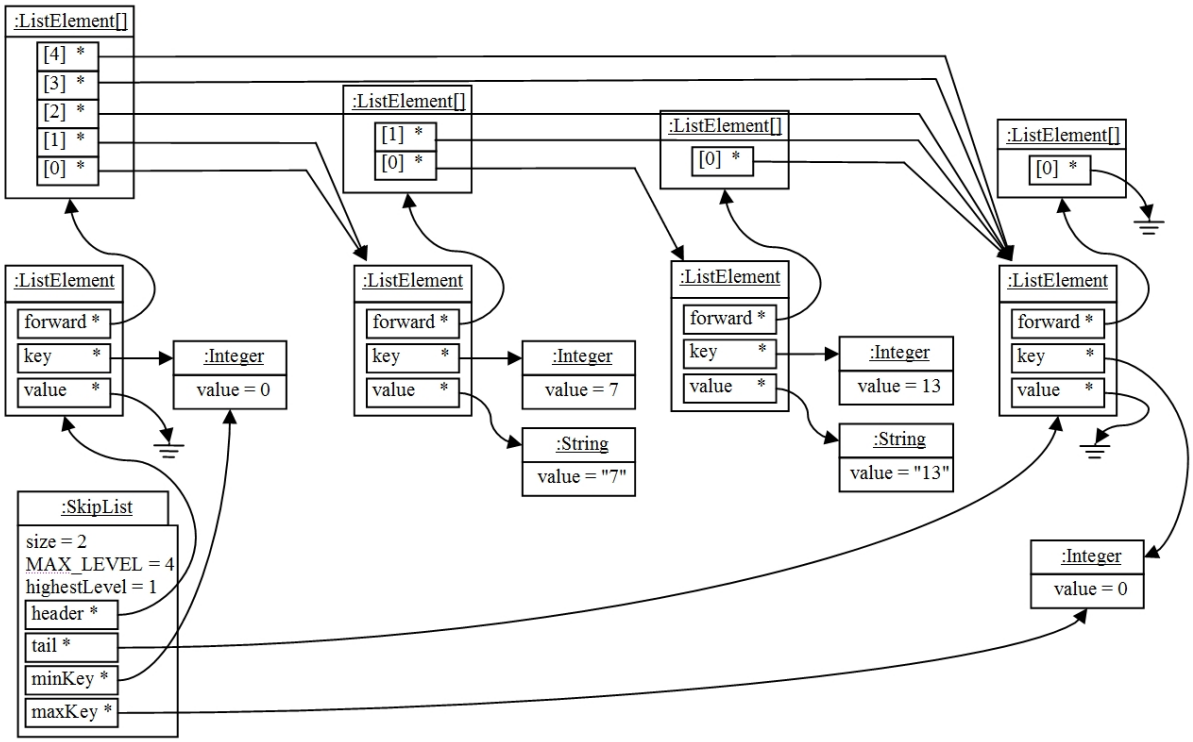


Abbildung 4: Beispiel eines Objektgraphen [13]

2.1.4.3D-Class-Object-Visalization

In der Vorgängerarbeit 3D-Class-Object-Visualizer (3DCOV) wurde die gleichnamige Applikation umgesetzt. Diese wurde als Diplomarbeit im Jahr 2007 von Dario Vonäsch und Daniel Hartmann verfasst und ebenfalls von Thomas Letsch betreut.

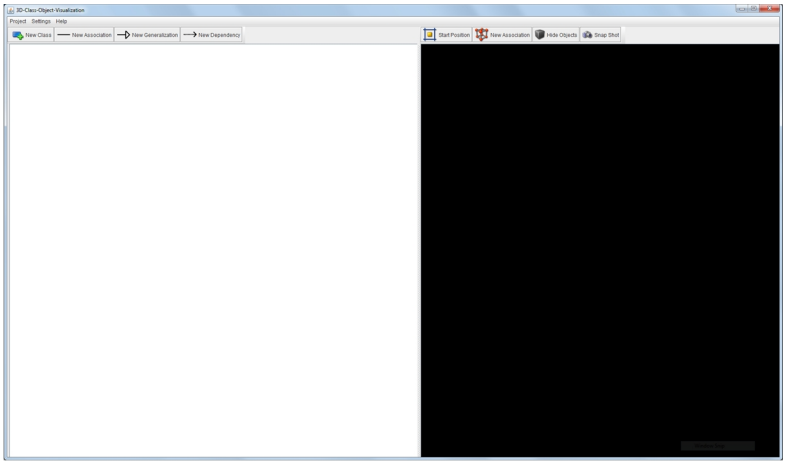


Abbildung 5: Screenshot von 3DCOV mit vertikaler Aufteilung

In 3D-Class-Object-Visalization wurde zur Anzeige Swing verwendet. Die Benutzeroberfläche ist vertikal in zwei Spalten eingeteilt. In der linken Spalte wird das 2D-Klassendiagramm erstellt. Dazu wurde die JGraph-Bibliothek eingesetzt. In der rechten Spalte wird dann das 3-dimensionale Objektdiagramm kreiert. Hierbei wird Java3D verwendet, welches aber durch die Funktionsaufrufe der 3D-Engine der Grafikkarte eine starke Abhängigkeit von der Hardware mit sich zieht [11]. Eine Objektgraph-Ansicht gibt es nicht in 3DCOV. Für das Persistieren wurde XStream und für das Logging TraceLib eingesetzt.

Die Sourcen des 3DCOVs sind zwar für das Projekt Object-Graph-Visualization verfügbar, da jedoch der Object-Graph-Visualizer mit JavaFX grundlegend neu entwickelt werden muss, kann nicht viel des Codes wiederverwendet werden. Das Parsing des XMI-Importers war das Einzige, welches vom 3DCOV in den OGV übernommen werden konnte.

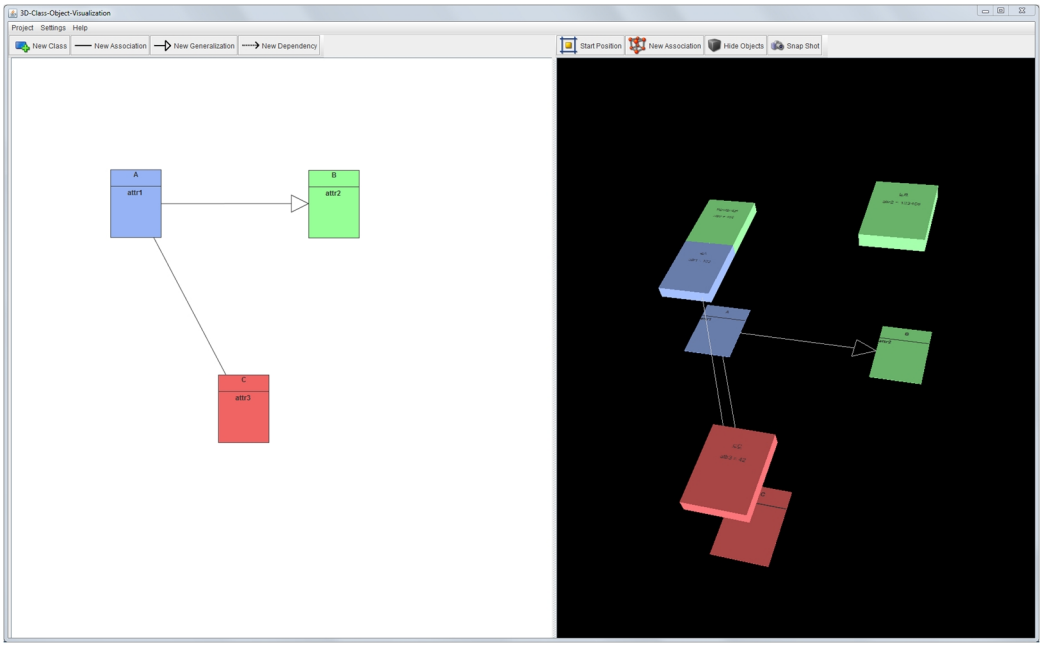


Abbildung 6: Screenshot von 3DCOV mit Klassen, Objekten und Relationen

2.1.5.Astah

In vielen Grundlagenmodulen an der HSR wird Astah eingesetzt. Diese Applikation ist ein Editor der Firma Change Vision und unterstützt viele Diagrammtypen, unter anderem Klassendiagramme, Use-Case-Diagramme, State-Diagramme und Sequenzdiagramme. Die frei verfügbare Astah Community-Version ist auf vielen privaten Laptops der Studenten zu finden. Auch Einsteiger sind in kurzer Zeit sehr produktiv, da sich der Funktionsumfang auf das Diagrammbearbeiten beschränkt. Astah überzeugt wegen seiner simplen Bedienung und wurde deswegen als Vorbild für den Object Graph Visualizer gewählt, bezüglich der Bedienelemente (Toolbar mit Comboboxen und Kontext-menüs).

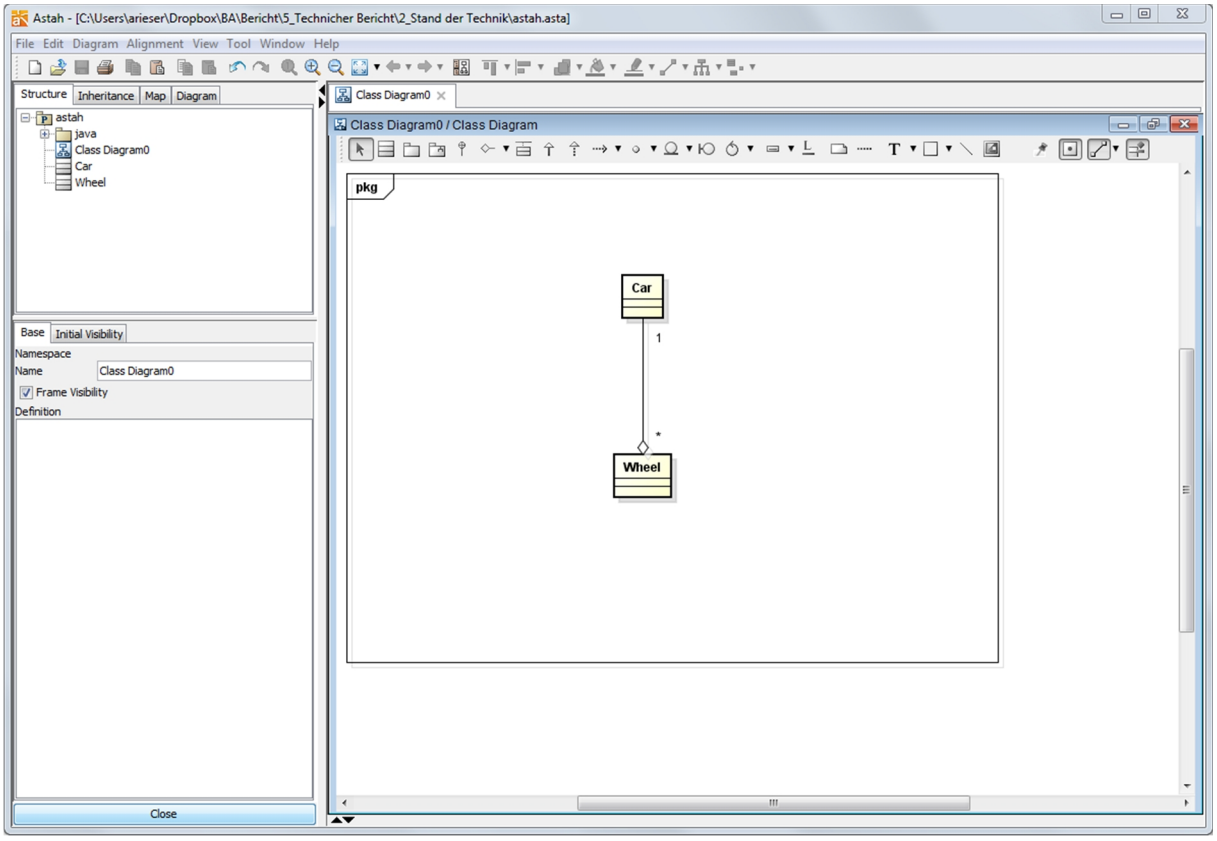


Abbildung 7: Screenshot von Astah

2.1.6. Enterprise Architect

Der Enterprise Architect (EA) ist ein Tool mit sehr grossem Funktionsumfang von der Firma Sparx Systems. Es wurde an der HSR nur im Modul Software-Engineering 3 (SE3) eingesetzt. Neben diversen UML Diagrammen unterstützt der EA auch das Erzeugen und Reverse-Engineering von Sourcecode und XMI-Export. Alle anderen Funktionen des EA sind nicht relevant für dieses Bachelorarbeitsprojekt. Für simple Klassen- und Objektdiagramme ist der Enterprise Architect von der Bedienung her zu stark hemmend, da er dafür nicht ausgelegt ist.

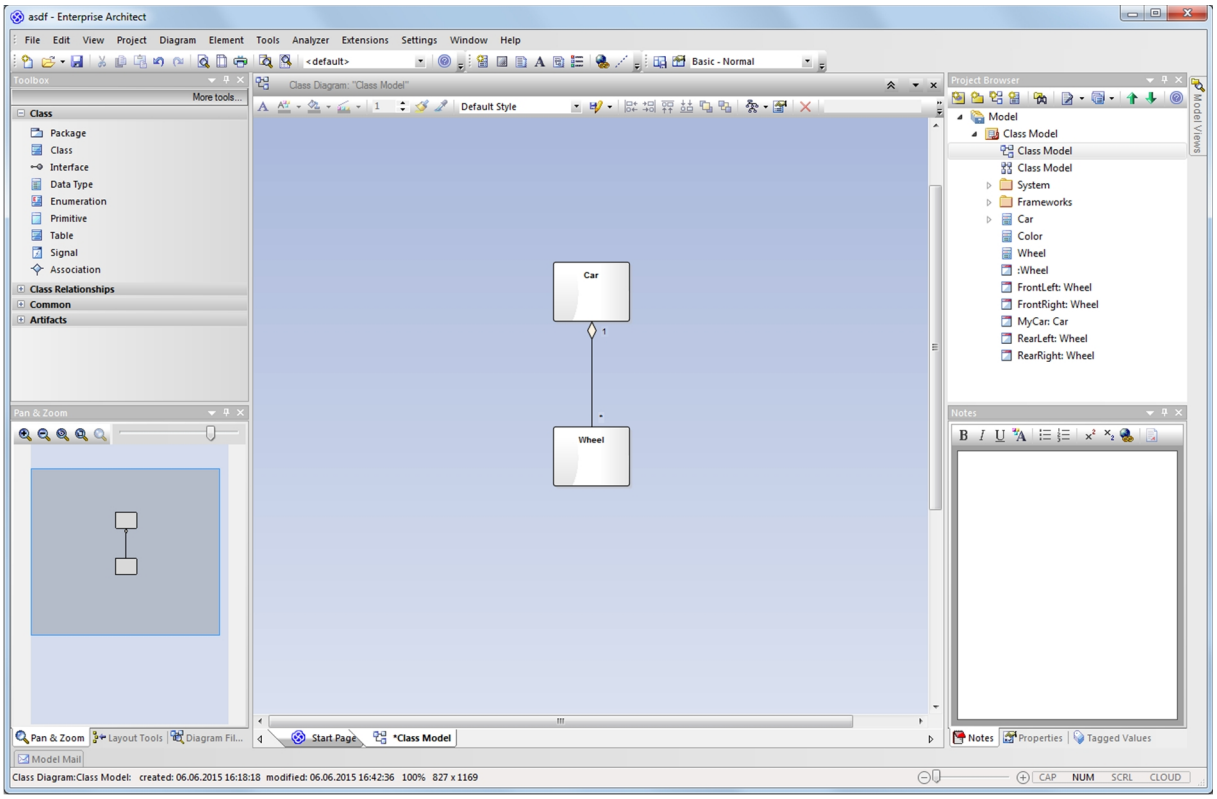


Abbildung 8: Screenshot des Enterprise Architect

3. Umsetzung

Als Vorgehensmodell zur Softwareentwicklung wurde Rational Unified Process (RUP) gewählt. Dies wurde in den Software Engineering Modulen an der HSR unterrichtet und ist somit beiden Projektmitarbeitern bekannt. In der ersten Phase, der Inception, wurde die Aufgabenstellung ausführlich besprochen und eine Vision verfasst. Alle Anpassungen und Verfeinerungen der Aufgabenstellung sind in den Besprechungsprotokollen ersichtlich.

Gleichzeitig wurden mehrere JavaFX-Tutorials durchgearbeitet, da keiner der Projektmitarbeiter praktische Erfahrung mit diesem Framework besass. Die beiden wichtigsten waren das «Getting Started with JavaFX» von Oracle [3] und das «JavaFX 8 Tutorial» von code.makery [2]. So konnte schon zu Beginn viel über die Funktionsweise von JavaFX in Erfahrung gebracht und das Risiko für unerwartete Probleme mit der neuen Technologie minimiert werden. Das Tutorial von Oracle war hilfreich, um eine Übersicht der verschiedenen UI-Komponenten zu erhalten. Im Tutorial von code.makery wurde dann der frei verfügbare JavaFX Scene Builder vorgestellt, welcher ab sofort immer eingesetzt wurde um die .fxml-Dateien zu erstellen oder zu editieren. Ein WYSIWYG-Editor mit sichtbaren Vorschlägen für alle Property-Felder zu allen Komponenten ist sehr hilfreich. Die schrittweise erarbeitete Applikation des Tutorials wurde dann auch als Vorlage bezüglich der Strukturierung für das Projekt Object-Graph-Visualization verwendet.

Nach der Domainanalyse, in der Elaboration-Phase, wurden die dort modellierten Klassen in Javacode implementiert. Die MVC-Architektur wurde gleichzeitig eingeführt, indem bei der Applikation aus dem code.makery-Tutorial die entsprechenden Model-, View- und Controller-Packages erstellt wurden und die bestehenden Klassen entweder neu eingeteilt oder aufgeteilt in die Packages integriert wurden. Danach wurde in drei grossen Versionen zuerst die Anforderungsspezifikationen verfasst, gefolgt von der entsprechenden Software «Object Graph Visualizer» (OGV) mit den neuen Funktionen. Version 1.0 wurde am Ende der Elaboration-Phase veröffentlicht und unterstützt die grundlegenden Funktionen für das Darstellen von Klassen und Objektdiagrammen. Während der Constrution-Phase folgte dann Version 2.0 mit der Persistierung und Version 3.0 mit dem Object Graph Mode. Neben der Implementierung der funktionalen Anforderungen wurde auch die komplexe Controller-Struktur stetig erweitert. Die finale Version 3.1 des Object Graph Visualizers erhielt Bugfixes und wurde terminlich zusammen mit dem Codefreeze auf den 29. Mai 2015 angesetzt.

4. Ergebnis

4.1. Zielerreichung

Das Ziel des Projekts wurde erreicht. Der Object Graph Visualizer wurde wie in den Anforderungen definiert umgesetzt und sieht wie folgt aus:

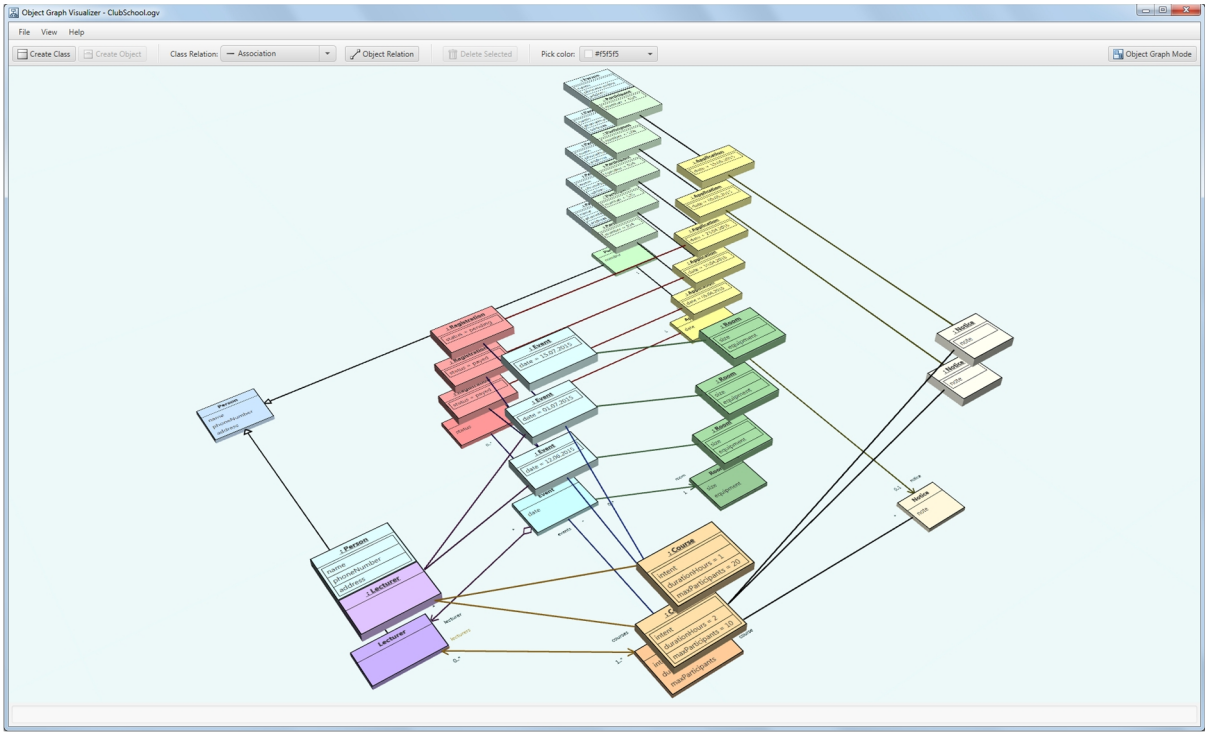


Abbildung 9: Screenshot OGV mit Beispielmodell

Alle drei Diagrammtypen (Klassen-, Objektdiagramm und Objektgraph) sind implementiert. Alle funktionalen und nicht-funktionalen Anforderungen werden abgedeckt. Das Design des GUIs ist ansprechend und modern. Der Benutzer kann schnell Klassen mit entsprechenden Attributen erstellen und diese untereinander mit verschiedener Relationstypen verbinden. Für erstelle Klassen können die jeweiligen Objekte vertikal über den Klassen erzeugt werden, welche wiederum mit Objekt-Relationen verbunden werden können. Die Software wurde unter verschiedenen Betriebssystemen getestet und läuft auf allen stabil. Der Betreuer beziehungsweise Auftraggeber ist sehr zufrieden mit dem Resultat. Einem Einsatz im Unterricht der Grundlagen-Module an der HSR spricht nichts mehr entgegen.

4.2. Bewertung

Im direkten Vergleich zum 3D-Class-Object-Visualization (3DCOV) lässt sich der OGV ohne Probleme installieren. Er startet schnell und das Fenster kann auch in der Grösse angepasst werden, sodass das Layout angepasst wird, damit alle Buttons erreichbar bleiben. Der Arbeitsfluss wird nicht gestört, da auf Dialogfenster verzichtet wurde. Ebenfalls sind im Gegensatz zum 3DCOV keine Artefakte in der 3D-Ansicht zu finden. Wie gewünscht, ist der OGV komplett über die Toolbar oder durch die Kontextmenüs bedienbar. Durch die Vereinigung der beiden Spaltenansichten des 3DCOV in eine einzige 3D-Anzeige ist mehr Raum für das eigentliche Modell vorhanden. Mehrere Konsistenzprüfungen stellen sicher, dass eingegebene Namen der verschiedenen Elemente entweder als nicht empfohlen markiert oder abgelehnt werden.

4.3. Ausblick

Auch wenn der Object Graph Visualizer als fertige und sehr gelungene Applikation angesehen werden kann, für Erweiterungen gibt es auch viele Vorschläge. Die beiden in der Aufgabenstellung erwähnten optionalen Features, das API für ein remotes Erstellen von Elementen sowie die Möglichkeit, den kompletten Aufbau einer anderen laufenden Java-Applikation zu zeigen.

4.3.1. API für remotes Erstellen

Es soll eine Programmierschnittstelle realisiert werden, die alle Use Cases abdecken würde. Diese erlaubt es, das Modell von externen Quellen aufzubauen oder zu manipulieren. Ein Software-Entwickler könnte bei Änderungen an einem Objekt gleichzeitig einen entsprechenden Befehl an den OGV senden. Der vollständige Programmaufbau könnte zur Laufzeit dynamisch visualisiert werden, was für den Gesamtüberblick und für das Debugging sehr hilfreich sein könnte.

4.3.2. Live-Zustand abbilden

Dabei soll eine zweite Java-Applikation seine verwendeten Klassen, Objekte und dessen Verbindungen automatisch per API dem OGV-Tool melden. Der Software-Entwickler müsste seinen Code für eine Anbindung an OGV nicht mehr anpassen.

Teil II – SW-Projektdokumentation

5. Anforderungsspezifikation

5.1. Allgemeine Beschreibung

5.1.1. Benutzercharakteristik

Definiert wurden zwei Benutzercharaktertypen für den Object Graph Visualizer.

Lehrpersonen

Diese Benutzer sind Experten im objektorientierten Design und kennen sich mit anderen Diagrammeditoren sehr gut aus. Sie möchten möglichst effizient Diagramme erstellen und den anderen Benutzern zeigen. Möglicherweise verfügen die Lehrpersonen bereits über in anderen Editoren erstellte Diagramme, die sie im OGV betrachten wollen. Sie möchten Personen, welche der zweiten Benutzercharakteristik angehören, zeigen, wie die verschiedenen Collection-Implementationen von Java aufgebaut sind.

OO-Anfänger

Diese Benutzer sollen mit dem OGV das Verständnis der objektorientierten Entwicklung mit Klassen und den dazugehörigen Objekten erlernen. Die OO-Anfänger werden von Lehrpersonen unterstützt und geschult.

5.3. Funktionale Anforderungen

5.3.1. Aktoren

Akteur	Akteur-Typ	Ziele
Benutzer	Primary	- Klassen mit Attributen erstellen - Objekte von erstellten Klassen instanziieren - Klassen und Objekte mit den entsprechenden Relationen verknüpfen - Den Object Graph Mode betrachten

5.3.2. Use Case Diagramm

Release Version 1.0: Prototyp

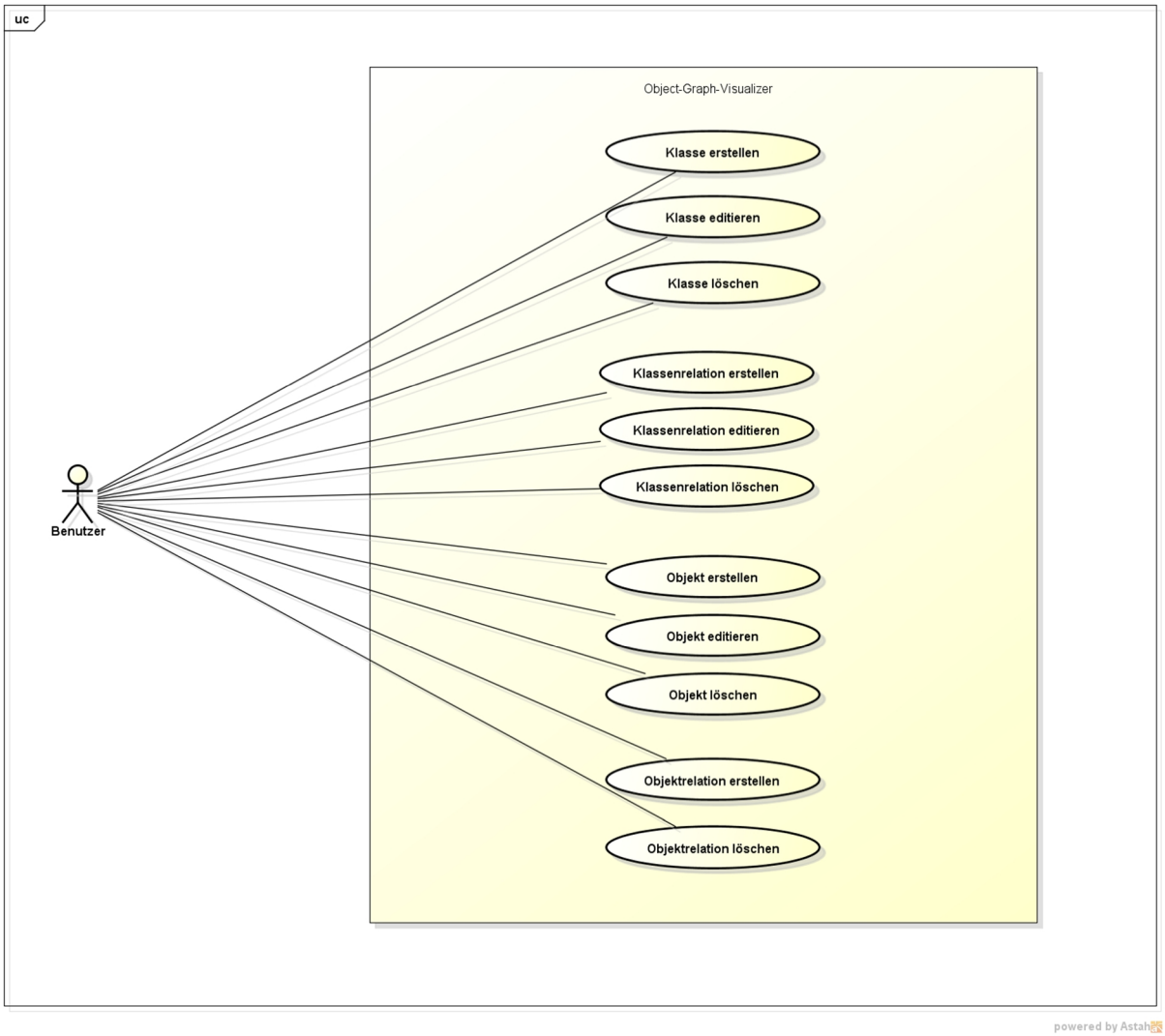


Abbildung 10: Use-Case-Diagramm Prototyp

Release Version 2.0: Persistency

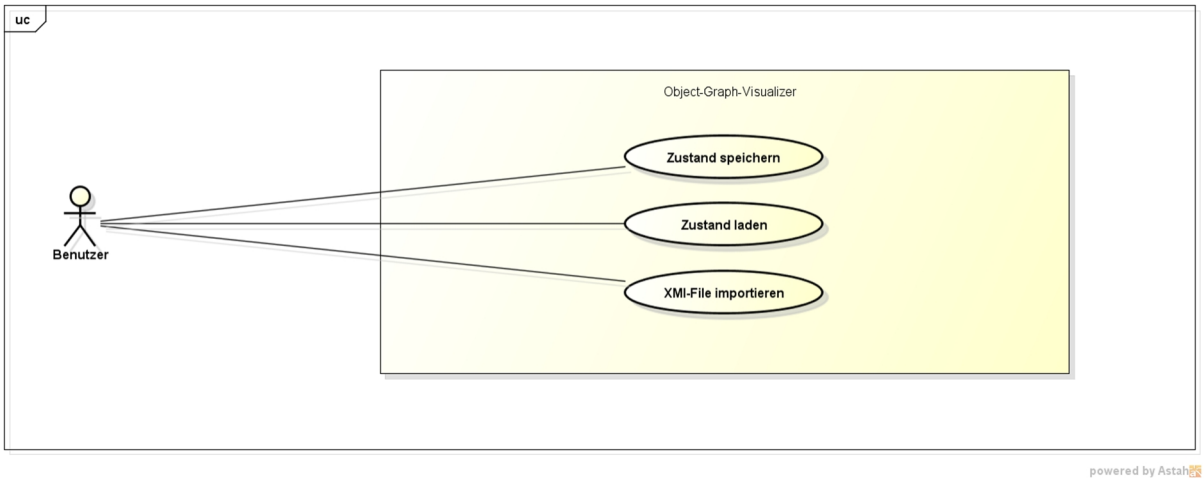


Abbildung 11: Use-Case-Diagramm Persistency

Release Version 3.0: Object Graph

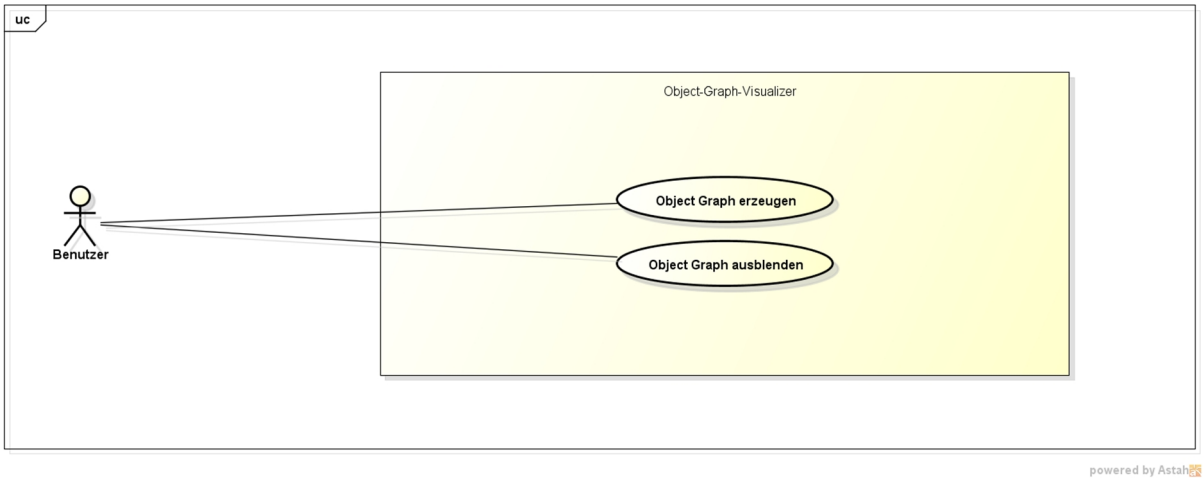


Abbildung 12: Use-Case-Diagramm Object Graph

5.4. Use Cases

Nachfolgend werden die Use Cases im «fully dressed»-Format beschrieben.

UC 01: Klasse erstellen

Use Case ID	**01**
Use Case Name	**Klasse erstellen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Klasse erstellen. Ausserdem soll es eine Möglichkeit geben, beim erstmaligen Erstellen einer Klasse das Erzeugen von Attributen zu beschleunigen (Quick-Creation).
Stakeholder and Interests	Benutzer: Eine Klasse erstellen.
Preconditions	OGV ist gestartet.
Postconditions	Der Benutzer hat eine Klasse mit gültigem Namen und optional auch mit gültigen Attributen erstellt.
Main Success Scenario	1. Der Benutzer platziert eine neue Klasse. 2. Der Benutzer gibt den Namen der neu zu erstellenden Klasse ein. 3. Falls die neu zu erstellende Klasse Attribute besitzt, kann der Benutzer nun diese mit Namen erfassen.

UC 02: Klasse editieren

Use Case ID	**02**
Use Case Name	**Klasse editieren**
Overview	Der Benutzer kann via Doppelklick auf ein Label (Name oder Attribut) oder Kontextmenü eine Klasse editieren.
Stakeholder and Interests	Benutzer: Eine Klasse nachträglich editieren.
Preconditions	Die zu editierende Klasse ist vorhanden.
Postconditions	Die editierte Klasse hat die gewünschten Änderungen übernommen.
Main Success Scenario	1. Der Benutzer wählt die zu editierende Klasse. 2. a) Der Benutzer kann den Klassennamen ändern. 2. b) Der Benutzer kann die Attributnamen ändern.

UC 03: Klasse löschen

Use Case ID	**03**
Use Case Name	**Klasse löschen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Klasse löschen.
Stakeholder and Interests	Benutzer: Eine Klasse löschen.
Preconditions	Die zu löschende Klasse ist vorhanden.
Postconditions	Der Benutzer hat die Klasse gelöscht. Klassenrelationen mit dieser Klasse sind ebenfalls entfernt.
Main Success Scenario	1. Der Benutzer wählt die zu löschende Klasse. 2. Der Benutzer löscht die gewählte Klasse. 3. Klassenrelationen, welche mit dieser Klasse in Verbindung stehen, werden ebenfalls mitgelöscht.

UC 04: Klassenrelation erstellen

Use Case ID	**04**
Use Case Name	**Klassenrelation erstellen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Klassenrelation erstellen.
Stakeholder and Interests	Benutzer: Eine Klassenrelation erstellen.
Preconditions	Für eine reflexive Klassenrelation: Mindestens eine Klasse vorhanden. Für nicht reflexive Klassenrelationen: Mindestens zwei Klassen vorhanden.
Postconditions	Der Benutzer hat die Klassenrelation und optional mit Rollennamen und Multiplizitäten erstellt.
Main Success Scenario	1. a) 1. Der Benutzer wählt den gewünschten Relationstyp in der Toolbar. 1. a) 2. Der Benutzer wählt eine Klasse als Startpunkt. 1. a) 3. Der Benutzer wählt eine Klasse als Endpunkt. 1. b) 1. Der Benutzer wählt eine Klasse als Startpunkt. 1. b) 2. Der Benutzer wählt den gewünschten Relationstyp im Kontextmenü. 1. b) 3. Der Benutzer wählt eine Klasse als Endpunkt. 2. Falls die Relation gültig ist, wird das System sie anzeigen. 3. Falls die neu erstellte Klassenrelation Rollennamen oder Multiplizitäten besitzt, kann der Benutzer nun diese mit Inhalt erfassen.

UC 05: Klassenrelation editieren

Use Case ID	**05**
Use Case Name	**Klassenrelation editieren**
Overview	Der Benutzer kann via Doppelklick auf ein Label (Rollenname oder Muliplizität) oder Kontextmenü eine Klasse editieren.
Stakeholder and Interests	Benutzer: Eine Klassenrelation nachträglich editieren.
Preconditions	Die zu editierende Klassenrelation ist vorhanden.
Postconditions	Die editierte Klassenrelation hat die gewünschten Änderungen übernommen.
Main Success Scenario	1. Der Benutzer markiert eine Klassenrelation. 2. a) Der Benutzer kann den Rollennamen ändern. 2. b) Der Benutzer kann die Multiplizitäten ändern.

UC 06: Klassenrelation löschen

Use Case ID	**06**
Use Case Name	**Klassenrelation löschen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Klassenrelation löschen.
Stakeholder and Interests	Benutzer: Eine Klassenrelation löschen.
Preconditions	Die zu löschende Klassenrelation ist vorhanden.
Postconditions	Der Benutzer hat die Klassenrelation gelöscht. Objektrelationen mit dieser Klassenrelation sind ebenfalls entfernt.
Main Success Scenario	1. Der Benutzer wählt die zu löschende Klassenrelation. 2. Der Benutzer löscht die gewählte Klassenrelation. 3. Objektrelationen welche mit dieser Klassenrelation in Verbindung stehen, werden ebenfalls mitgelöscht.

UC 07: Objekt erstellen

Use Case ID	**07**
Use Case Name	**Objekt erstellen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü ein Objekt erstellen. Ausserdem soll es eine Möglichkeit geben, beim erstmaligen Erstellen eines Objekts das füllen von Attributwerten zu beschleunigen (Quick-Creation).
Stakeholder and Interests	Benutzer: Ein Objekt erstellen.
Preconditions	Die zu instanziierende Klasse muss vorhanden sein.
Postconditions	Der Benutzer hat ein Objekt, optional mit gültigen Attributwerten, erstellt.
Main Success Scenario	1. Der Benutzer wählt die zu instanziierende Klasse. 2. Der Benutzer instanziiert ein Objekt der gewählten Klasse. 3. Falls das neu erstellte Objekt Attribute besitzt, kann der Benutzer nun diese mit Werten erfassen.

UC 08: Objekt editieren

Use Case ID	**08**
Use Case Name	**Objekt editieren**
Overview	Der Benutzer kann via Doppelklick auf ein Label (Name oder Attributwerte) oder Kontextmenü ein Objekt editieren.
Stakeholder and Interests	Benutzer: Ein Objekt nachträglich editieren.
Preconditions	Das zu editierende Objekt ist vorhanden.
Postconditions	Das editierte Objekt hat die gewünschten Änderungen übernommen.
Main Success Scenario	1. Der Benutzer wählt das zu editierende Objekt. 2. a) Der Benutzer kann den Objektnamen ändern. 2. b) Der Benutzer kann die Attributwerte ändern.

UC 09: Objekt löschen

Use Case ID	**09**
Use Case Name	**Objekt löschen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü ein Objekt löschen.
Stakeholder and Interests	Benutzer: Ein Objekt löschen.
Preconditions	Das zu löschende Objekt ist vorhanden.
Postconditions	Der Benutzer hat das Objekt gelöscht. Objektrelationen mit diesem Objekt sind ebenfalls entfernt.
Main Success Scenario	1. Der Benutzer wählt das zu löschende Objekt. 2. Der Benutzer löscht das gewählte Objekt. 3. Objektrelationen welche mit diesem Objekt in Verbindung stehen, werden ebenfalls mitgelöscht.

UC 10: Objektrelation erstellen

Use Case ID	**10**
Use Case Name	**Objektrelation erstellen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Objektrelation erstellen.
Stakeholder and Interests	Benutzer: Eine Objektrelation erstellen.
Preconditions	Für eine reflexive Objektrelation: Mindestens ein Objekt vorhanden. Für nicht reflexive Objektrelationen: Mindestens zwei Objekte vorhanden.
Postconditions	Der Benutzer hat eine Klasse mit gültigem Namen und optional auch mit gültigen Attributnamen erstellt.
Main Success Scenario	1. a) 1. Der Benutzer wählt den gewünschten Relationstyp in der Toolbar. 1. a) 2. Der Benutzer wählt ein Objekt als Startpunkt. 1. a) 3. Der Benutzer wählt ein Objekt als Endpunkt. 1. b) 1. Der Benutzer wählt ein Objekt als Startpunkt. 1. b) 2. Der Benutzer wählt den gewünschten Relationstyp im Kontextmenü. 1. b) 3. Der Benutzer wählt ein Objekt als Endpunkt. 2. Falls die Relation gültig ist, wird das System sie anzeigen.

UC 11: Objektrelation löschen

Use Case ID	**11**
Use Case Name	**Objektrelation löschen**
Overview	Der Benutzer kann via Toolbar oder Kontextmenü eine Objektrelation löschen.
Stakeholder and Interests	Benutzer: Eine Objektrelation löschen.
Preconditions	Die zu löschende Objektrelation ist vorhanden.
Postconditions	Der Benutzer hat die Objektrelation gelöscht.
Main Success Scenario	1. Der Benutzer wählt die zu löschende Objektrelation. 2. Der Benutzer löscht die gewählte Objektrelation.

UC 12: Zustand speichern

Use Case ID	**12**
Use Case Name	**Zustand speichern**
Overview	Der Benutzer kann via Menüleiste das aktuelle Modell mit Klassen, Objekten und Relationen persistent auf dem Dateisystem abspeichern.
Stakeholder and Interests	Benutzer: Das erstellte Modell soll gespeichert werden können, um Zwischenstände zu speichern, die Arbeit zu pausieren oder um erstellte Modelle mit anderen Geräten auszutauschen.
Preconditions	OGV ist gestartet.
Postconditions	Das System speichert den aktuellen Zustand in ein ogv-File persistent auf das Dateisystem ab.
Main Success Scenario	1. Der Benutzer wählt in der Menüleiste den Menüpunkt zum Speichern. 2. Ein Dateiauswahldialog für die Pfadangabe der zu speichernden Datei erscheint. 3. Das System speichert an die Pfadangabe die ogv-Datei.

UC 13: Zustand laden

Use Case ID	**13**
Use Case Name	**Zustand laden**
Overview	Der Benutzer kann via Menüleiste ein Modell mit Klassen, Objekten und Relationen, welches auf dem Dateisystem abgespeichert ist, wieder einlesen und darstellen.
Stakeholder and Interests	Benutzer: Ein bereits erstelltes Modell soll geladen werden können, um Zwischenstände wiederherzustellen, die pausierte Arbeit fortzusetzen oder um Modelle, welche auf anderen Geräten erstellt wurden, einzulesen.
Preconditions	Eine korrekt abgespeicherte ogv-Datei auf dem Dateisystem ist vorhanden.
Postconditions	Das angegebene ogv-File wurde geladen und wird angezeigt.
Main Success Scenario	1. Der Benutzer wählt in der Menüleiste den Menüpunkt zum Laden. 2. Ein Dateiauswahldialog für die Pfadangabe der zu ladenden Datei erscheint. 3. Das System lädt die angegebene ogv-Datei und zeigt diese an.

UC 14: XMI-File importieren

Use Case ID	**14**
Use Case Name	**XMI-File importieren**
Overview	Der Benutzer kann via Menüleiste ein Modell mit Klassen und Relationen welches auf dem Dateisystem abgespeichert ist einlesen und darstellen.
Stakeholder and Interests	Benutzer: Ein in einem anderen Editor erstelltes Modell soll geladen werden können, um die Erstellungsaufgabe nicht wiederholen zu müssen.
Preconditions	Ein korrekt erzeugtes XMI-Dokument ist auf dem Dateisystem vorhanden.
Postconditions	Das angegebene XMI-Dokument wurde geladen und wird angezeigt.
Main Success Scenario	1. Der Benutzer wählt in der Menüleiste den Menüpunkt zum XMI-Import. 2. Ein Dateiauswahldialog für die Pfadangabe der zu importierenden Datei erscheint. 3. Das System lädt das angegebene XMI-Dokument, parst diese und zeigt sie anschliessend an.

UC 15: Object Graph erzeugen

Use Case ID	**15**
Use Case Name	**Object Graph erzeugen**
Overview	Der Benutzer kann via Menüleiste den Objektgraph-Modus einschalten.
Stakeholder and Interests	Benutzer: Generieren und anzeigen lassen des Objektgraphen.
Preconditions	Der Objektgraph ist nicht eingeschaltet. Sinnvollerweise sollten mindestens zwei Objekte mit Objektrelationen vorhanden sein.
Postconditions	Der Objektgraph wird angezeigt. Das UML Objektdiagramm wird nicht mehr angezeigt.
Main Success Scenario	1. Der Benutzer schaltet in der Toolbar der Objektgraph-Modus ein. 2. Das System ergänzt anhand der Rollennamen der Klassenrelationen die Attribute der Klassen und Objekte. 3. Das System erstellt für Relationen mit unbekannter Multiplizität oder Multiplizität höher als 1 ein Array-Objekt und verbindet die Objektrelationen mit diesem. 4. Das System erstellt die nötigen Referenzpfeile.

UC 16: Object Graph ausblenden

Use Case ID	**16**
Use Case Name	**Object Graph ausblenden**
Overview	Der Benutzer kann via Menüleiste den Objektgraph-Modus ausschalten.
Stakeholder and Interests	Benutzer: Abschalten des generierten Objektgraphen und wieder-herstellen der vorherigen Darstellung.
Preconditions	Der Objektgraph ist eingeschaltet.
Postconditions	Der Objektgraph wird nicht mehr angezeigt. Das UML Objektdiagramm wird angezeigt.
Main Success Scenario	1. Der Benutzer schaltet in der Toolbar der Objektgraph-Modus aus. 2. Das System entfernt die generierten Referenzpfeile. 3. Das System entfernt alle Array-Objekte. 4. Das System entfernt die Attribute der Klassen und Objekte welche anhand der Rollennamen der Klassenrelationen generiert wurden.

5.5. Nichtfunktionale Anforderungen

NFR 01	**Keine Dialogfenster**
	Der exzessive Einsatz von Dialogfenstern des 3DCOV ist als Negativpunkt aufgefallen. Der Benutzer wird im Arbeitsfluss gestört, da er sich bei geöffneten Fenstern neu orientieren muss. Im OGV soll während der Entwicklung eines Modells auf Dialogfenster verzichtet werden.
NFR 02	**Multiple Bedienmöglichkeiten**
	Neben der Bedienung über die Toolbar soll der OGV auch mit Kontextmenüs und Tastaturkürzeln benutzt werden können.
NFR 03	**Unterstützung wichtiger Betriebssysteme**
	Um möglichst viele Benutzer zu erreichen, soll neben Windows 7 von den Übungsrechnern der HSR auch Windows 8.1, OS X 10.10 und Ubuntu 15.04 unterstützt werden.
NFR 04	**Erlernbarkeit**
	Der Umgang mit dem OGV Tool soll so simpel und selbsterklärend wie möglich gestaltet werden. Neben einer ausführlichen Benutzerdokumentation soll die Einarbeitungszeit dadurch verkürzt werden, dass bestimmte Prozessabläufe (wie zum Beispiel Speichern und Laden) ähnlich oder gleich wie bei bekannten anderen Applikationen sind.
NFR 05	**Speicherbedarf**
	Der Speicherbedarf soll 100 MB nicht überschreiten. Davon ausgenommen sind die gespeicherten Modelle.
NFR 06	**Performance: Anzahl der Klassen und Objekte**
	Es sollen mindestens 200 Klassen und Objekte gleichzeitig dargestellt werden können.
NFR 07	**Performance: Reaktion und Wartezeit**
	Auch wenn die Anzahl der Klassen und Objekte steigt, soll das Drehen und Verschieben der Kamera ohne Verzögerung erscheinen.

5.6. Schnittstellen

5.6.1. XMI-Import

XML Metadata Interchange (XMI) ist ein von der Object Management Group (OMG) standardisiertes Austauschformat für UML-Modelle. Um in anderen Applikationen erstellte Klassendiagramme in den OGV einlesen zu können, soll eine XMI-Schnittstelle für den Import realisiert werden. XMI-Inhalte werden in XML-Dateien gespeichert und können so geparst werden. Im OGV soll eine XMI-Datei mittels File-Dialog ausgewählt werden können. Anschliessen wird es geparst und anschliessend werden die entsprechenden Komponenten im Hauptfenster dargestellt.

```
<?xml version="1.0" encoding="windows-1252"?>
<XMI xmi.version="1.1" xmlns:UML="omg.org/UML1.3" timestamp="2015-05-25
21:54:04">
    <XMI.header>
        <XMI.documentation>
            <XMI.exporter>Enterprise Architect</XMI.exporter>
            <XMI.exporterVersion>2.5</XMI.exporterVersion>
        </XMI.documentation>
    </XMI.header>
    <XMI.content>
        <UML:Model name="EA Model" … >
            …
            <UML:Class name="Astronaut" … visibility="public" … >
                …
                <UML:Attribute name="mName" … visibility="private" … >
```

Code Snippet 1: Beispielinhalt eines XMI-Dokuments

6. Analyse

6.1. Domain Model

Die erste Version des Domain Models war stark beeinflusst vom Metamodell zum UML-Klassendiagramm. Der Grundgedanke war, dass Teile der dort verwendeten Strukturen für das Domain Model übernommen werden können.

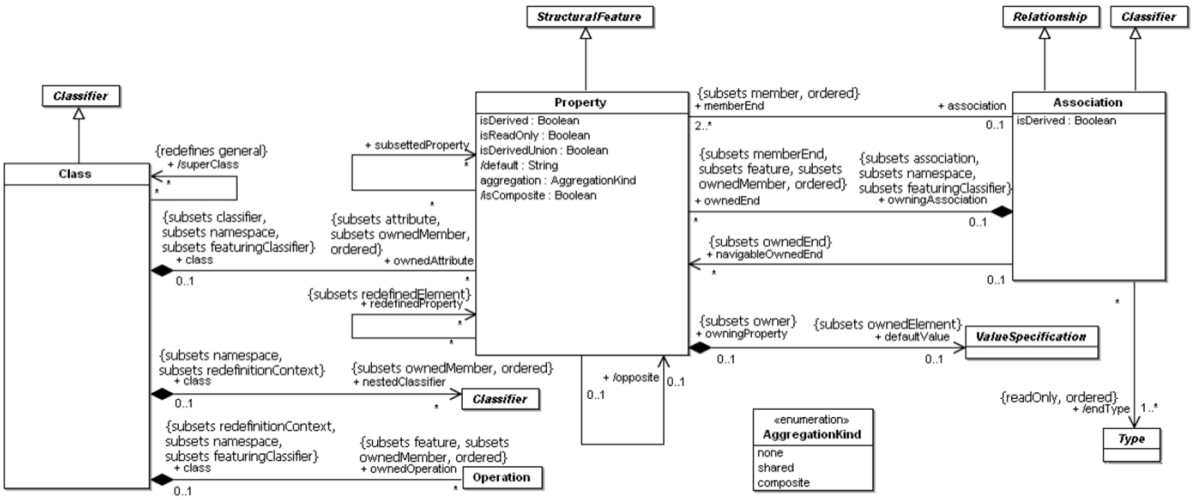


Abbildung 13: Klassendiagramm-Metamodell von OMG [14]

Leider liefert die Object Management Group (OMG) nur entweder das komplette Metamodell des UML Klassendiagramms oder nur einzelne modellierte Komponenten davon (z.B. Class oder Multiplicity-Element). Das komplette Modell ist zu generell gehalten und die einzelnen Komponenten zu feingranular modelliert für den Einsatz im OGV. Selbst das vereinfachte Metamodel aus dem HSR Modul Software Engineering 3 (SE3) war noch zu detailreich. In der dann folgenden Internetrecherche wurde von der University of Texas at Dallas (UT Dallas) ein vereinfachtes Metamodell des Klassendiagramms gefunden [12], welches für die anfängliche Verwendung beim OGV geeignet war.

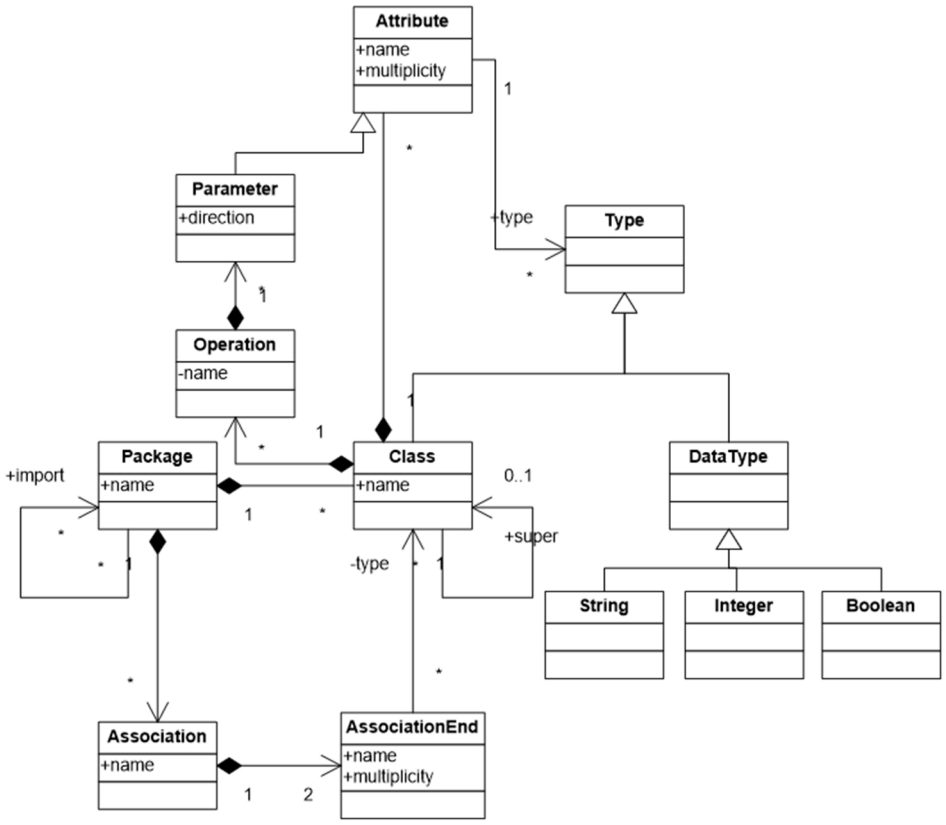


Abbildung 14: Klassendiagramm-Metamodell von UT Dallas [12]

In mehreren Überarbeitungsschritten wurde dann immer mehr Abschied vom Metamodell genommen, da einerseits nicht ein komplettes Metamodell gebraucht wurde und andererseits der Fokus vom Object Graph Visualizer auf der Anzeige von Klassenmodellen liegt. Dafür bedarf es schlicht nicht dem Kohäsionsgrad eines Metamodells, was wiederum die Anzahl von Objekten zur Laufzeit reduziert und sich somit positiv auf die Performance auswirkt.

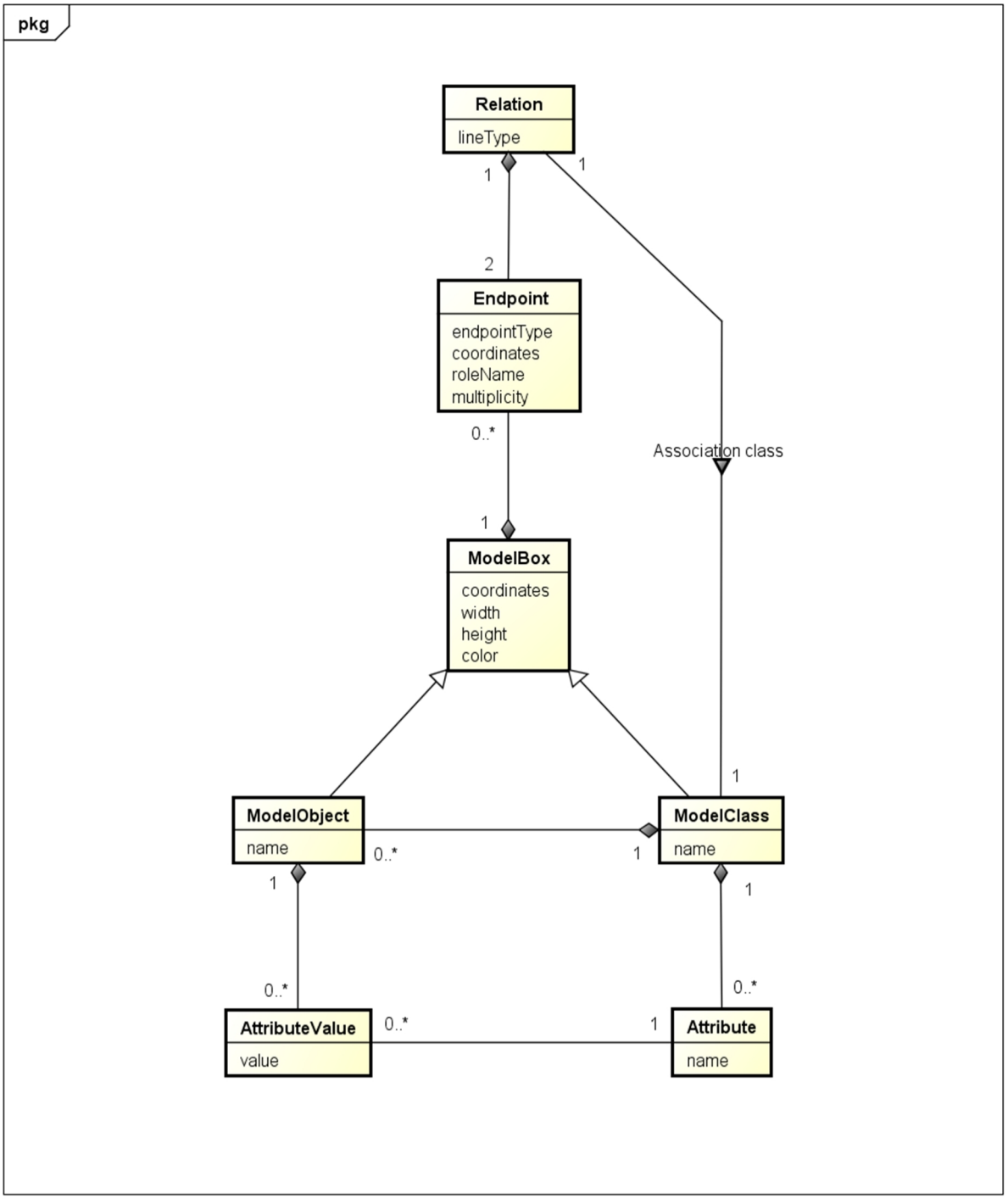


Abbildung 15: Klassendiagramm am Ende der Analyse

6.2. Klassenkatalog

6.2.1. Attribute

Die Klasse Attribute repräsentiert ein Attribut bzw. ein Feld einer Klasse. Eine ModelClass kann über mehrere Attributes verfügen. Diese Attributes besitzen einen pro ModelClass eindeutigen Namen.

6.2.2. AttributeValue

Die Klasse AttributValue repräsentiert ein Wert eines Attributs, der einem ModelObject zugewiesen werden kann. Der Wert kann auch null sein.

6.2.3. Endpoint

Die Klasse Endpoint repräsentiert ein Endpunkt einer Relation zwischen zwei ModelClasses oder zwei ModelObjects. Entweder ist es ein Start-Endpoint oder ein End-Endpoint. Zwei Endpoints sind mit einer Relation verbunden. Zusätzlich dazu beinhaltet ein Endpoint Informationen über seinen Typ (die Pfeilart).

6.2.4. ModelBox

Die Klasse ModelBox repräsentiert die Superklasse von ModelClass und ModelObject. Sie ist abstrakt und übernimmt auch Werte, die in der Ansicht relevant sind: Neben den Mittelpunktkoordinaten und Angaben zu der Box-Dimension besitzt die ModelBox eine Farbeigenschaft. Da ModelClasses sowie auch ModelObjects untereinander mit Relations verbunden werden können, enthält die ModelBox alle Endpoints, welche mit der jeweiligen Box verbunden sind.

6.2.5. ModelClass

Die Klasse ModelClass repräsentiert eine Klasse im Sinne der Objektorientierung. Sie ist abgeleitet von der ModelBox. Eine ModelClass hat systemweit einen eindeutigen Namen. Neben besitzt die Model-Class auch Informationen über ihre instanziierten ModelObjects.

6.2.6. ModelObject

Die Klasse ModelObject repräsentiert ein Objekt im Sinne der Objektorientierung. Sie ist ebenfalls abgeleitet von der ModelBox. Ein ModelObject hat für jedes der Attribute seiner ModelClass eine AttributeValue.

6.2.7. Relation

Die Klasse Relation repräsentiert eine Verbindung zwischen zwei Endpoints bzw. zwei ModelBoxen. Sie besitzt je einen Start-Endpoint und einen End-Endpoint. Neben den Endpoints kennt die Relation noch ihren Linientyp (durchzogene Line, gestrichelte Linie).

Das untenstehende Diagramm zeigt Zusammenspiel und Funktionsweise der gesamten Relation.

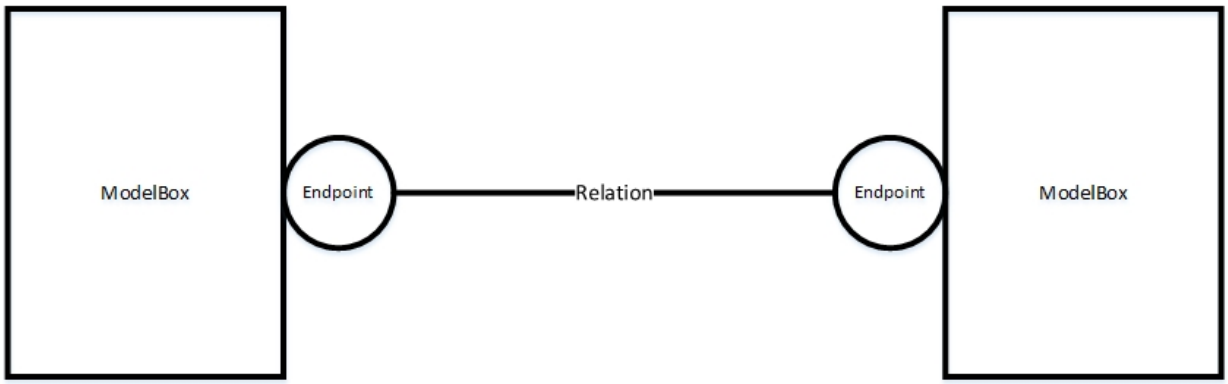


Abbildung 16: Übersicht von einer Relation

7. Design

7.1. Architektur

7.1.1. Ziele und Einschränkungen

Die wichtigsten Ansprüche an die Architektur für den ObjectGraphVisualizer sind:

Austauschbare Layers

Die Packages werden in logische Einheiten aufgeteilt und erfüllen ihre zentralen Aufgaben.

Beispielsweise sollen Klassen, die Benutzereingaben behandeln, komplett von den Klassen im Resource Layer getrennt werden.

Einfache Erweiterbarkeit

Im Hinblick auf eine spätere Weiterentwicklung (siehe Kapitel 10 auf Seite 75) ist das OGV System so aufgebaut, dass es einfach erweitert und angepasst werden kann.

Plattformunabhängigkeit

Durch die Verwendung von Java als Programmiersprache ist schon der wichtigste Grundstein für die Plattformunabhängigkeit gelegt: Java-Programme werden in der Java Virtual Machine (JVM) ausgeführt, welche für viele Betriebssysteme erhältlich ist.

Dieser Punkt stellt jedoch eine Einschränkung für die externen Libraries und Frameworks dar. Nur solche Bibliotheken kommen in Frage, die native Aufrufe kapseln.

7.1.2. Projektstruktur

Es wird eine einzelnes Projekt erstellt. Die Projektstruktur ist massgeblich durch den Einsatz von JavaFX vorgegeben. In der folgenden Tabelle ist eine Beschreibung der wichtigsten Elemente und Verzeichnisse des Projekts ersichtlich.

Verzeichnis	Beschreibung
src	Der gesamte Quellcode wird in diesem Verzeichnis abgelegt.
resources	Das Resources-Verzeichnis beinhaltet alle Assets wie Menü-Icons, FXML Templates, OBJ-Modelle und CSS.
test	Test-Klassen befinden sich alle unter diesem Verzeichnis.
build	In dieses Verzeichnis werden die Releases deployed. Ant Build Files befinden sich ebenfalls hier.
ea	Verzeichnis für das Enterprise Architect Reverse Engineering.
examples	Einige Beispiel *.ogv- und *.xml (XMI)-Dateien.
lib	Alle verwendeten Bibliotheken sind in diesem Verzeichnis.
logs	Die Log-Files werden hier zentral abgelegt.

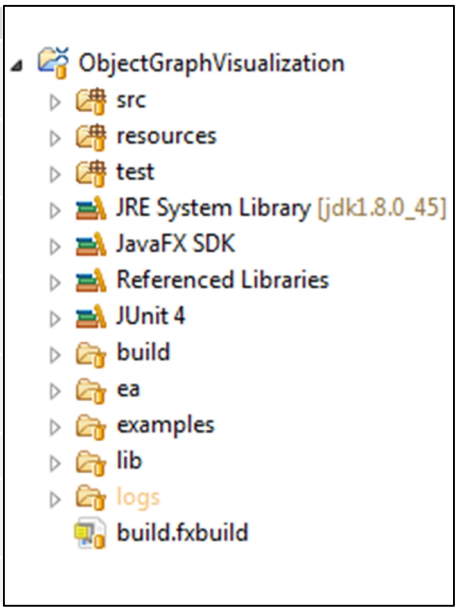


Abbildung 17: Projektstruktur

7.1.3. Architekturübersicht

Dieses Kapitel gibt einen Überblick über die Architektur. Untenstehende Abbildung illustriert den Aufbau des Systems und die Package-Abhängigkeiten.

Da der Object Graph Visualizer eine 1-Tier-Applikation ist, wurde viel Wert auf eine Umsetzung des Model-View-Controller-Patterns (MVC-Patterns) gelegt. Der Kontrollfluss erfolgt von oben nach unten. Auf das Util-Package (oben rechts) wird von allen anderen Packages zugegriffen, diese Dependencies sind aus Gründen der Übersicht nicht eingezeichnet.

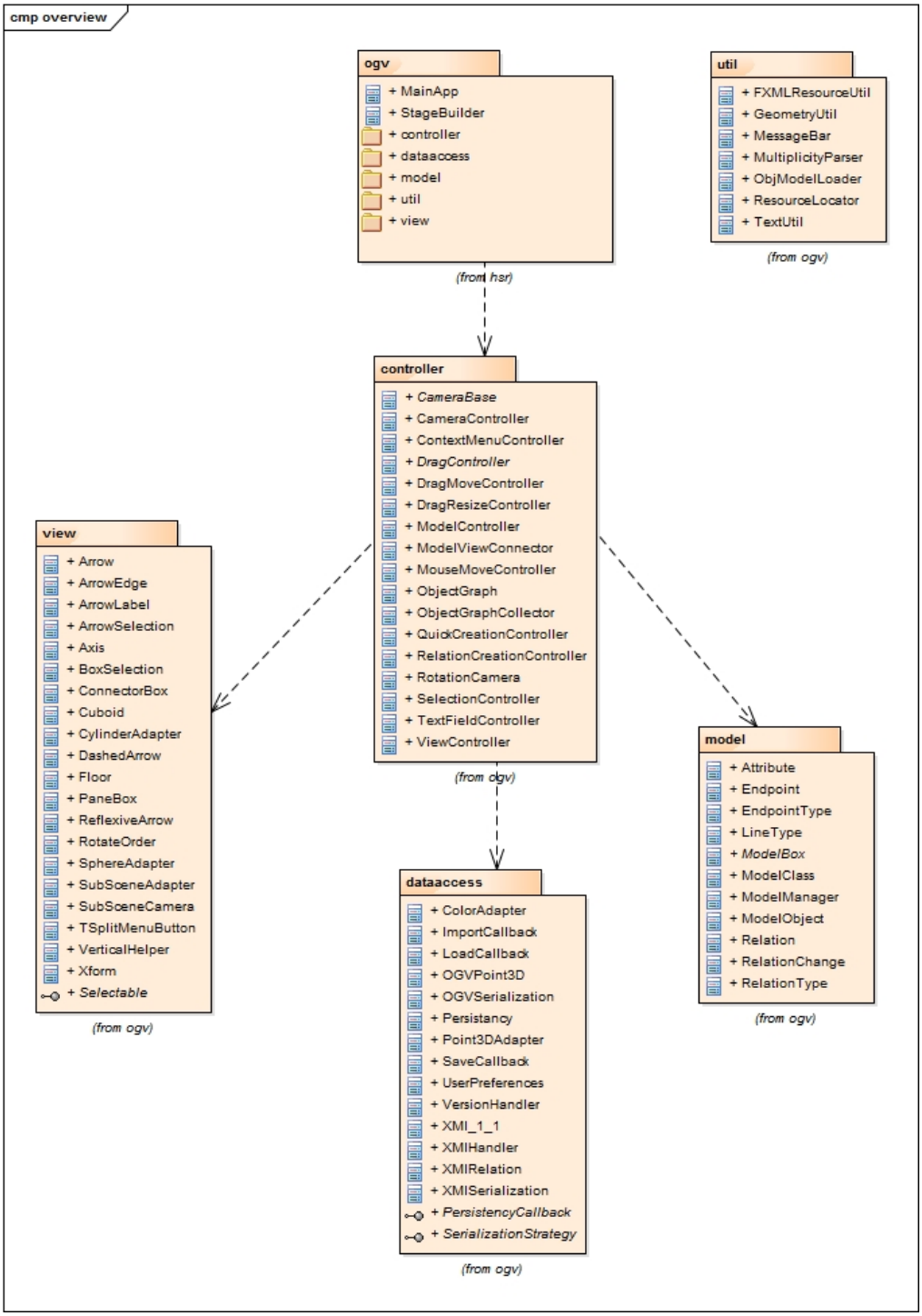


Abbildung 18: Architekturübersicht

7.2. Packages

Hier werden die Packages anhand von Grafiken im Detail beschrieben. Die Grafiken sind mit dem Eclipse-Plugin STAN generiert und zeigen alle Beziehungen zwischen den Klassen. Das heisst nicht nur Referenzierung, sondern auch Aufruf, Implementierung, Typabfrage, Parameter, Import, Erweiterung etc. Die Klassen innerhalb eines Packages sind grösstenteils so modelliert, dass keine zyklischen Beziehungen auftreten.

7.2.1. OGV-Root

Im Oberverzeichnis des Projekts befinden neben den Unterpackages auch zwei Klassen, mit welchen die Applikation initialisiert wird.

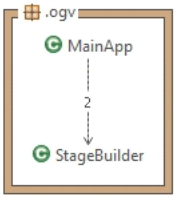


Abbildung 19: OGV-Root Package

7.2.2. Controller

Dieses Package übernimmt die Aufgabe, das Programm zentral zu steuern: Benutzeraktionen werden entgegengenommen, in der entsprechenden Controller-Klasse weiterverarbeitet (z. B. bei ungültigen Texteingaben verworfen) und allenfalls wird ein Änderungsbefehl an das View-, Model- oder DataAccess-Package gegeben. Allfällige Rückmeldungen erfolgen über das Observer- oder Callback-Pattern. So ist die Steuerung ein «Beobachter» des Modells, um bei Änderung der Daten direkt die Präsentation im View-Package zu manipulieren.

Es wird zwischen Application-Controllern und Input-Controllern unterschieden:

- Application-Controller Klassen: Programmsteuerung und Verbindung zwischen View und Model *ModelController*, *ViewController*, *RelationCreationController*, *ObjectGraphCollector*, *ObjectGraph* und *ModelViewConnector*

- Input-Controller Klassen: Spezialisiert auf verschiedene Benutzereingaben *ContextMenuController*, *DragResizeController*, *TextFieldController*, *QuickCreationController*, *DragController*, *DragMoveController*, *DragResizeController*, *MouseMoveController*, *SelectionController*, *CameraBase*, *CameraController* und *RotationCamera*

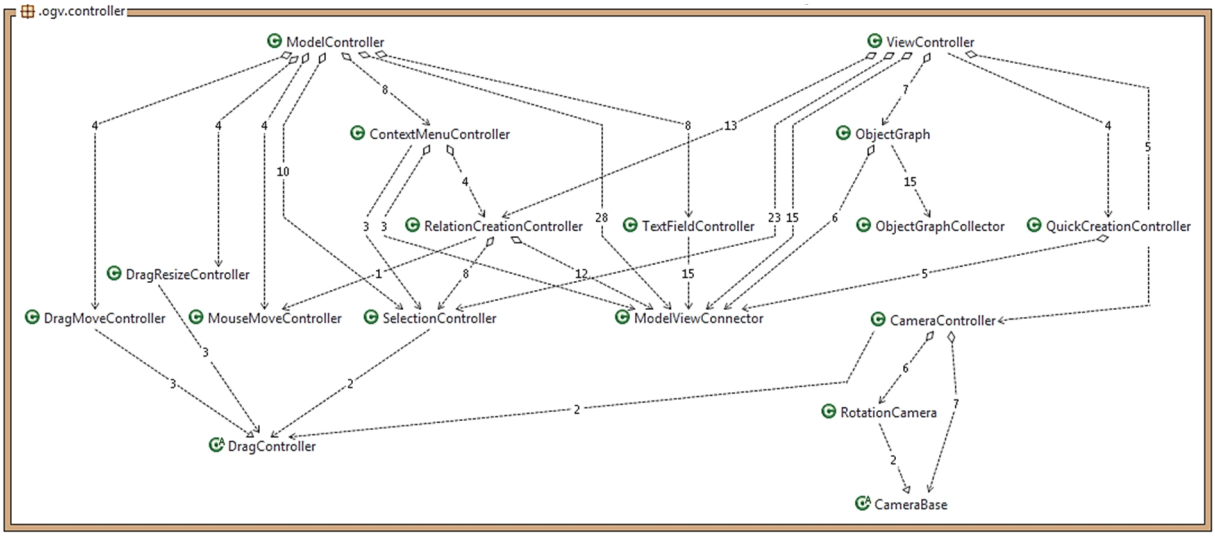


Abbildung 20: Klassenbeziehungen im Controller-Package

7.2.3. View

Das View-Package bildet die Präsentationsschicht, für die Darstellung der benötigten Daten aus dem Modell. Die Domainklassen haben eine entsprechende Repräsentation im View-Package. Änderungen werden hingegen nur vom Controller veranlasst.

Die Klassen im View-Package haben nur mit der Präsentation in der 3D-Szene zu tun. Eine Ausnahme bildet die TSplitMenuButton Klasse. (Die Menus und Toolbars, in welche auch die 3D-Szene eingebettet sind, werden nicht durch Klassen gelöst, sondern liegen vollständig als FXML-Template vor.)

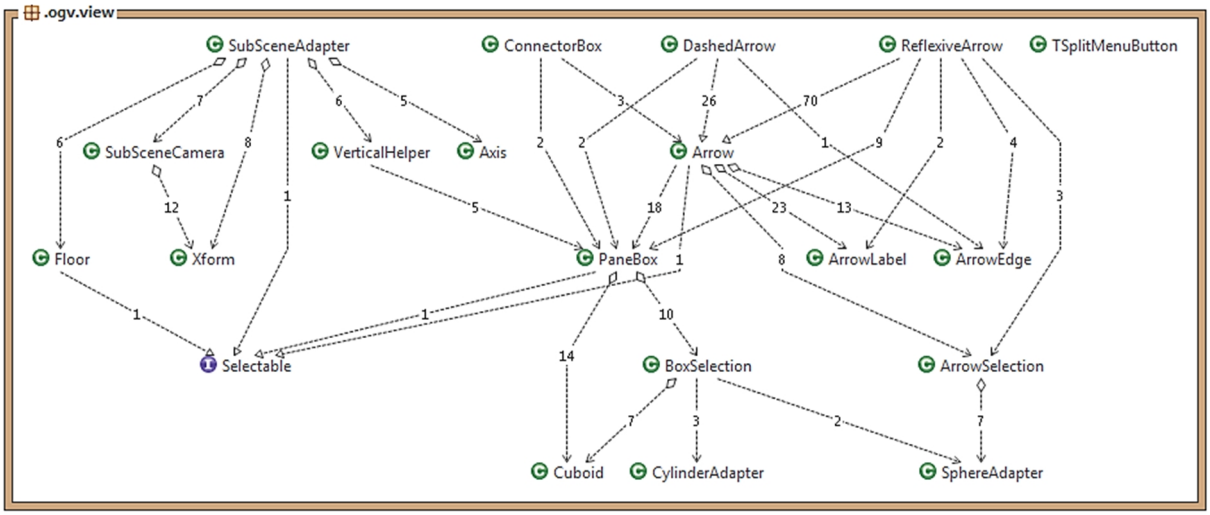


Abbildung 21: Klassenbeziehungen im View-Package

7.2.4. Model

In diesem Package befinden sich die Domainklassen mit den zu repräsentierenden Daten. Das Model ist unabhängig von View und Controller. Aufrufe erfolgen über die ModelManager-Fassadenklasse. Anders herum geschieht die Bekanntgabe von Änderungen an Werten im Modell per Observer an den ModelController.

Es ist das einzige Package, welches Klassen mit zyklischen Assoziationen aufweist. Einerseits zwischen den Klassen Relation-Endpoint und Endpoint-ModelBox, andererseits zwischen ModelClass und ModelObject.

Die ersten beiden Fälle lassen sich durch das Relationen-Konzept des Domainmodells erklären (siehe Kapitel 6.1).

Die gegenseitige Beziehung zwischen ModelClass und ModelObject ist aus Komfortgründen für die Implementation der Vererbung eingeführt worden.

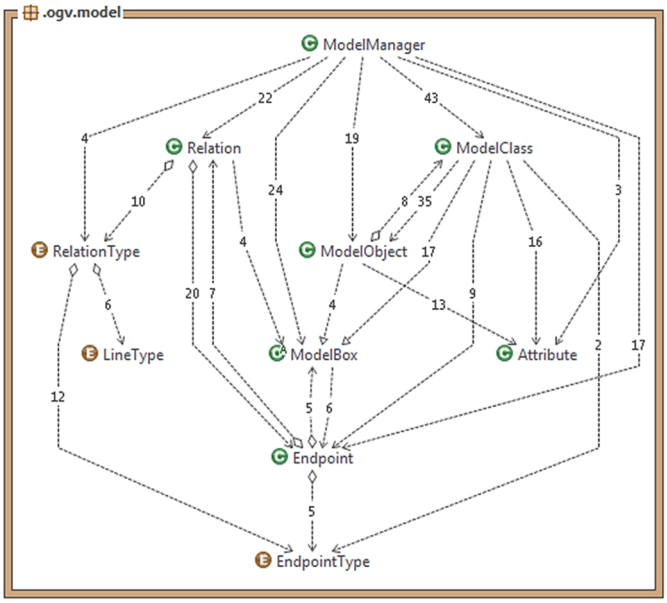


Abbildung 22: Klassenbeziehungen im Model-Package

7.2.5. DataAccess

Das DataAccess-Package kümmert sich um die Datenhaltung und ist als Teil des Resource Layers unterhalb der anderen Packages angesiedelt. Zugriff auf Funktionen des Packages erfolgt ebenfalls gekapselt nach dem Fassaden-Pattern. Die Funktionen sind Speichern, Laden und Import des Modells.

Ist ein Vorgang abgeschlossen, wird dies per Callback mitgeteilt.

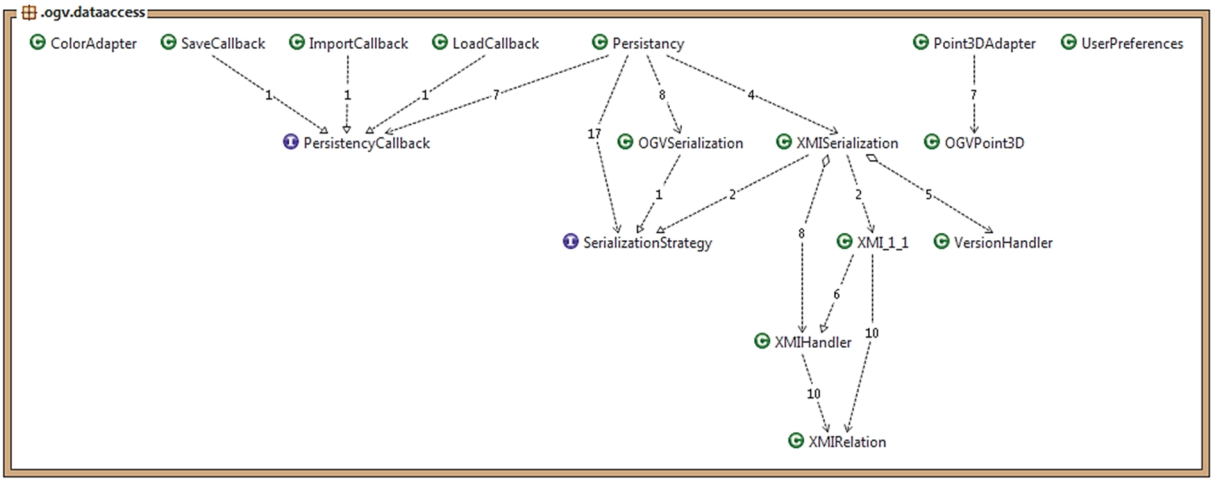


Abbildung 23: Klassenbeziehungen im DataAccess-Package

7.2.6. Util

Das Util-Package ist eine Ansammlung an Hilfsklassen, die von mehreren Klassen aus verschiedenen Packages gebraucht werden oder Berechnung vornehmen. Aus diesem Grund werden die Hilfsmethoden «static» aufgerufen.

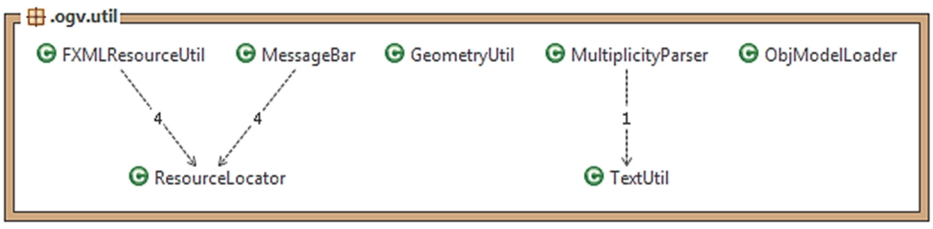


Abbildung 24: Klassenbeziehungen im Util-Package

7.3. OGV-Root-Klassendiagramm

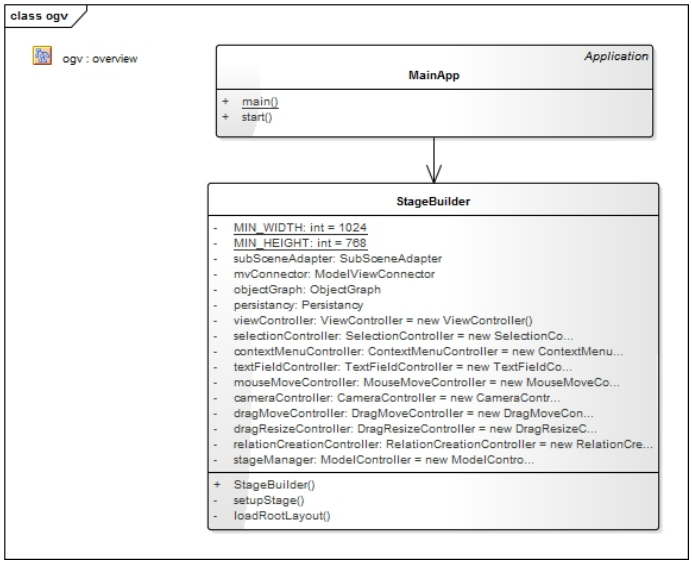


Abbildung 25: Klassendiagramm des OGV-Root Packages

7.4. OGV-Root-Klassenkatalog

7.4.1. MainApp

Die Klasse MainApp beinhaltet die main-Methode und ist somit der Einstiegspunkt des Object Graph Visualizers. Hier wird auch gleich die JavaFX Platform mit Scene Graph und GUI-Thread initialisiert.

7.4.2. StageBuilder

Der StageBuilder lädt zuerst die FXML-Datei für das GUI und erstellt dann die Stage. Gleich danach werden alle Controller-Klassen instanziiert und in der korrekten Reihenfolge initialisiert um die Abhängigkeiten richtig zu setzen.

7.5. Controller Klassendiagramm

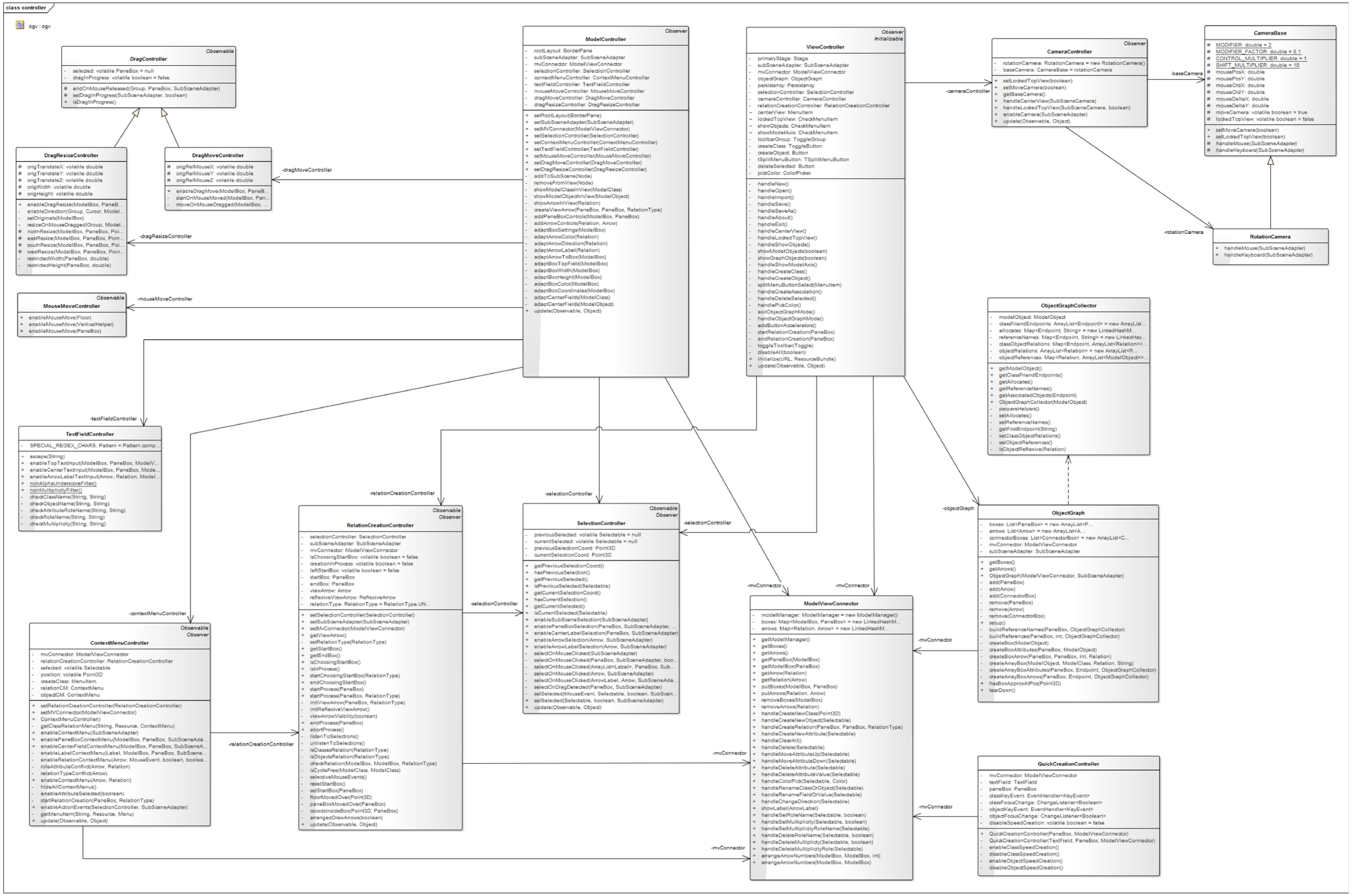


Abbildung 26: Klassendiagramm des Controller Packages

7.6. Controller Klassenkatalog

7.6.1. CameraBase

Eine Kamera ermöglicht es, den Beobachtungspunkt im Raum zur Laufzeit frei wählen zu können. Die CameraBase ist als eine abstrakte Klasse für alle Camera-Typen anzusehen. Sie legt das Grundverhalten für alle von ihr abgeleiteten Camera-Klassen fest.

7.6.2. RotationCamera

Die RotationCamera ist momentan die einzige Camera im OGV und enthält die Funktionalitäten, um die Ansicht zu rotieren.

7.6.3. CameraController

Der CameraController ist die Controller-Klasse der Camera. Er beinhaltet die CameraBase und wird vom ViewController angesprochen.

7.6.4. ModelController

Der ModelController überwacht das Model als Observer und meldet alle Änderungen der View. Seine Verbindung ins Model-Package erfolgt über den ModelViewConnector.

7.6.5. ViewController

Der ViewController regelt die Aktionen vom Menü in der Menüleiste.

7.6.6. ModelViewConnector

Der ModelViewConnector mappt einerseits die Model-Klassen auf die entsprechenden View-Klassen (z. B. Relation zu Arrow), andererseits ist er für Zugriffe in das Model-Package zuständig.

7.6.7. ContextMenuController

Der ContextMenuController stellt alle verschiedenen Kontextmenüs zur Verfügung. Er wird vom ModelController jedes Mal angesprochen, wenn ein neues Element in der Ansicht erstellt wurde, um dort die entsprechenden Kontextmenüs einzubauen.

7.6.8. DragController

Der DragController ist die abstrakte Superklasse von allen DragControllers. Er ist für das Grundverhalten der verschiedenen Bewegungen der Klassen und Objekte zuständig.

7.6.9. DragMoveController

Der DragMoveController ist für das korrekte Verschieben der Klassen in der Ebene und der Objekten auf der Achse über der Klasse zuständig.

7.6.10. DragResizeController

Der DragMoveController ist für das korrekte Resizing der Klassen in der Ebene zuständig.

7.6.11. MouseMoveController

Die Aufgabe des MouseMoveController ist es, die Position des Cursors im 3D-Raum korrekt zu erkennen und bei Mausbewegungen die registrierten Interessenten zu informieren.

7.6.12. QuickCreationController

Der QuickCreationController behandelt das Verhalten einer Klasse oder eines Objekts, wenn es zum ersten Mal erstellt wird und damit der Quick-Creation-Vorgang aktiv ist.

7.6.13. RelationCreationController

Der RelationCreationController ist für den komplexen Erstellungsprozess einer Relation verantwortlich.

7.6.14. SelectionController

Der SelectionController verarbeitet alle Selektionsevents von den Elementen und leitet Selektionsänderungen an seine Observer weiter.

7.6.15. TextFieldController

Der TextFieldController behandelt alle Events von den Textfeldern in Klassen, Objekten, Rollennamen und Multiplizitäten und übernimmt die Validierung der Texteingabe.

7.6.16. ObjectGraph

Die ObjectGraph-Klasse erstellt den eigentlichen Objektgraph. Er wird vom ObjectGraphCollector mit den benötigten Informationen beliefert.

7.6.17. ObjectGraphCollector

Der ObjectGraphCollector sammelt alle Informationen des aktuellen Models und übergibt diese dem ObjectGraph zur Erstellung der neuen Anzeige.

7.7. View Klassendiagramm

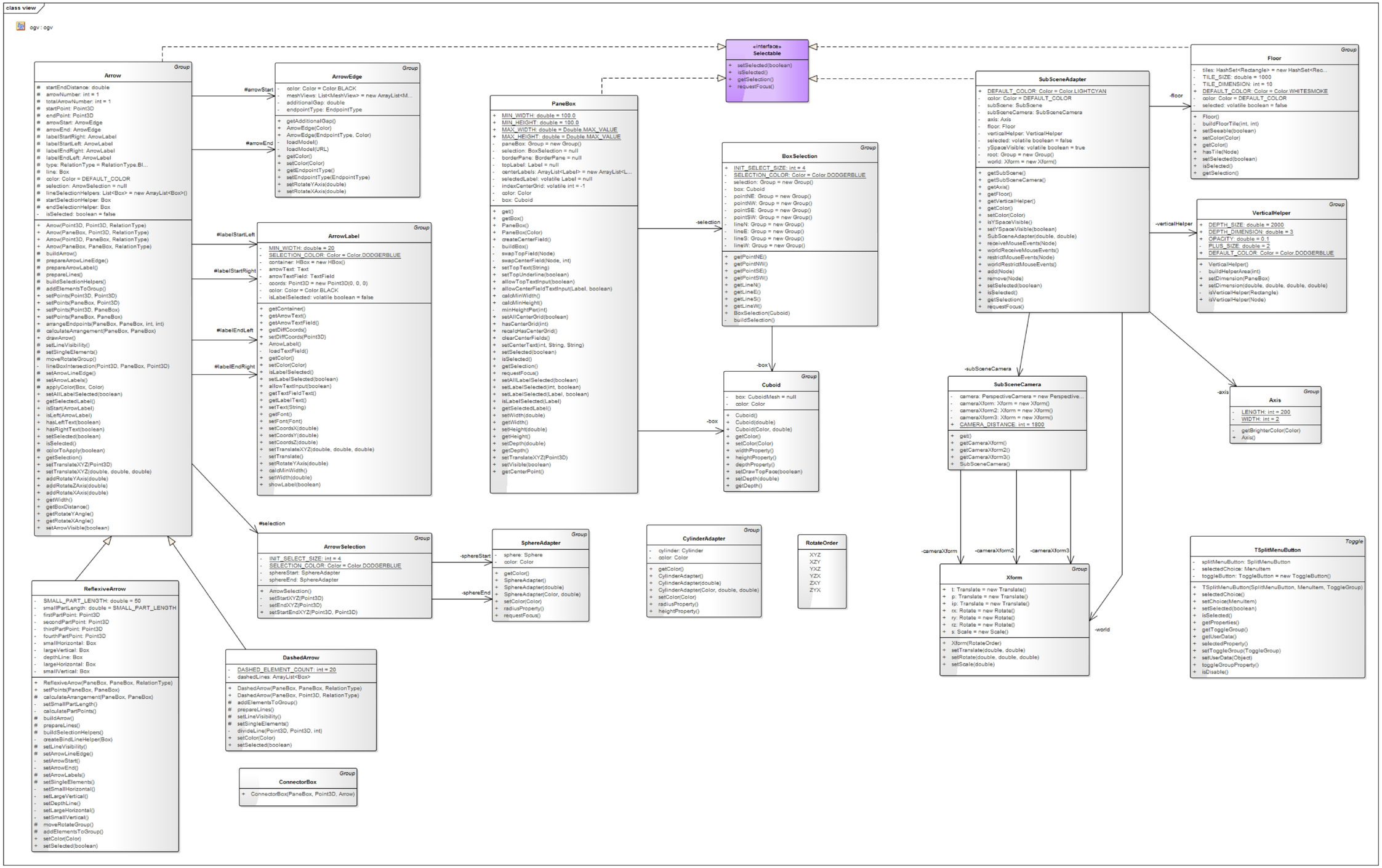


Abbildung 27: Klassendiagramm des View Packages

7.8. View Klassenkatalog

7.8.1. SubSceneAdapter

Die SubSceneAdapter adaptiert die JavaFX SubScene-Klasse. Diese stellt das Fenster für die 3D-Szene zur Verfügung. Die SubSceneAdapter-Klasse regelt ebenfalls ob die verschiedenen Elemente in der 3D-Szene MouseEvents erhalten oder nicht.

7.8.2. Xform

Wurde aus dem Oracle Sample-Tutorial [4] entnommen und erlaubt eigene Transformationen und Rotationen im 3D-Raum.

7.8.3. SubSceneCamera

Die SubSceneCamera adaptiert die Perspektivische Kamera für erleichterten Zugriff.

7.8.4. Floor

Der Floor ist die flach in der Ebene liegende Referenzfläche, welche das Platzieren und das Verschieben von Elementen erlaubt.

7.8.5. VerticalHelper

Der VerticalHelper ist der modellierte Referenzraum oberhalb einer Klasse, welcher das Platzieren der Objekte erlaubt.

7.8.6. Axis

Die Axis-Klasse zeigt als Orientierungshilfe im Mittelpunkt des Koordinatensystems die drei Achsen an.

7.8.7. Selectable

Das Selectable-Interface markiert bei der PaneBox-, Arrow- und Floor-Klasse, dass diese selektierbar sind.

7.8.8. CylinderAdapter

Diese Klasse adaptiert die JavaFX-Cylinder 3D-Primitive für die benötigten Methoden.

7.8.9. SphereAdapter

Der SphereAdapter adaptiert die JavaFX-Sphere 3D-Primitive für die benötigten Methoden.

7.8.10. Cuboid

Die Cuboid-Klasse adaptiert die CuboidMesh-Klasse aus der FXYZ-Library.

7.8.11. PaneBox

Eine PaneBox ist die modellierte Darstellung einer ModelClass oder eines ModelObjects.

7.8.12. BoxSelection

Die BoxSelection-Klasse ist die modellierte Selektion um eine PaneBox.

7.8.13. Arrow

Die Arrow-Klasse ist die graphische Umsetzung einer Relation. Der Arrow verfügt über zwei Arrow-Edges und vier ArrowLabels.

7.8.14. DashedArrow

Die DashedArrow-Klasse repräsentiert die gestrichelte Dependency.

7.8.15. ReflexiveArrow

Der ReflexiveArrow ist eine Sonderform des Arrows und erweitert diesen um die Funktionalität einer reflexiven Relation.

7.8.16. ArrowEdge

Die ArrowEdge-Klasse ist die graphische Umsetzung eines Endpoints.

7.8.17. ArrowLabel

Die ArrowLabels werden für die Angabe des Rollennamen und der Multiplizität verwendet.

7.8.18. ArrowSelection

Die ArrowSelection-Klasse ist die modellierte Selektion um ein Arrow.

7.8.19. ConnectorBox

Diese Klasse wird für die Darstellung der Referenzen im Objektgraph benötigt.

7.8.20. TSplitMenuButton

Eine eigene Implementation eines SplitMenuButtons mit zusätzlicher Toggle-Möglichkeit.

7.9. Model Klassendiagramm

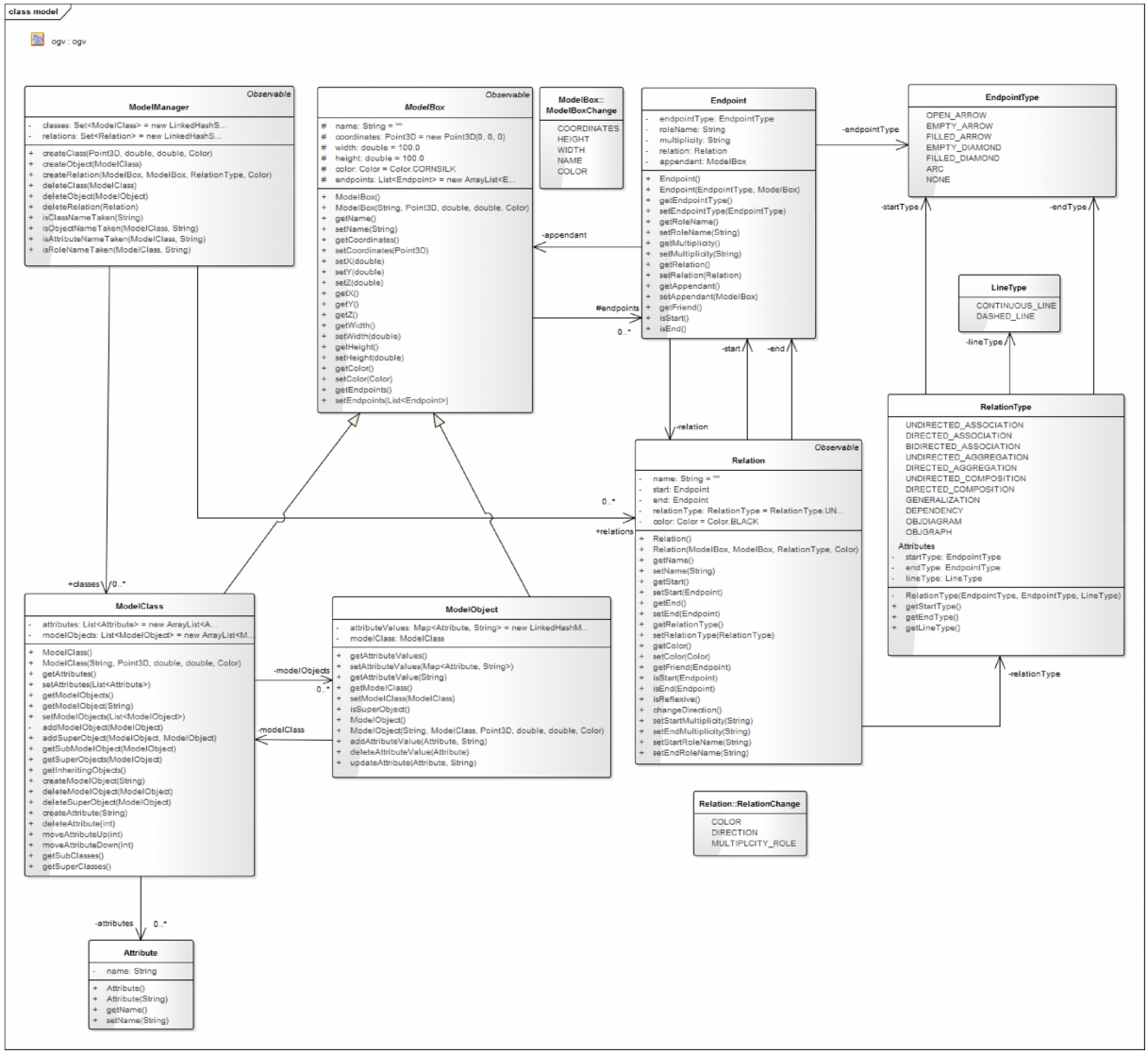


Abbildung 28: Klassendiagramm des Model Packages

7.10. Model Klassenkatalog

7.10.1. ModelManager

Der ModelManager enthält alle Informationen über das Model und regelt gemäss des Entwurfsmusters Fassade den Zugriff. Er unterhält die Liste von Klassen, welche wiederum die Listen von Endpoints und Listen von ModelObjects beinhalten. Für die einfachere Handhabung beinhaltet er ebenfalls eine Liste von Relationen.

7.10.2. ModelBox

Entspricht der Domainklasse ModelBox.

7.10.3. ModelClass

Entspricht der Domainklasse ModelClass.

7.10.4. ModelObject

Entspricht der Domainklasse ModelObject.

7.10.5. Attribute

Entspricht der Domainklasse Attribute.

7.10.6. Relation

Entspricht der Domainklasse Relation.

7.10.7. RelationType

Entspricht dem Typ der Relation (Assoziation, Generalisierung, etc.) und wurde als Enumeration realisiert.

7.10.8. LineType

Entspricht dem Typ des Relation-Body (gestrichelt oder durchgezogen) und wurde als Enumeration realisiert.

7.10.9. Endpoint

Entspricht der Domainklasse Endpoint.

7.10.10. EndpointType

Entspricht dem Typ des Endpoints und wurde als Enumeration realisiert.

7.11. DataAccess Klassendiagramm

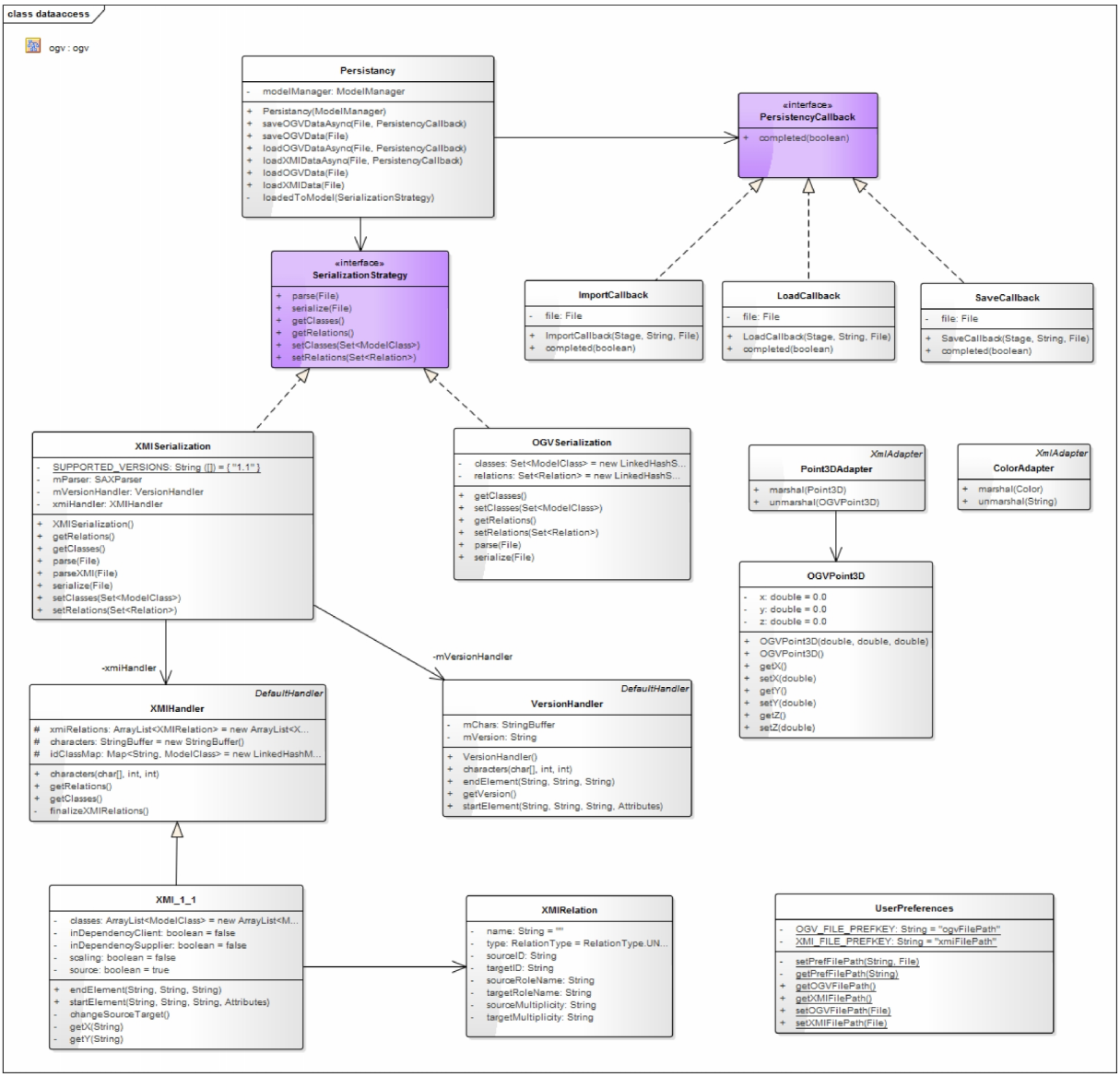


Abbildung 29: Klassendiagramm des DataAccess Packages

7.12. DataAccess Klassenkatalog

7.12.1. Persistancy

Die Persistancy-Klasse ist als Fassade für alle Persistenz-Funktionen zu verstehen.

7.12.2. PersistencyCallback

Das PersistencyCallback-Interface wird von allen Callbacks der Persistenz implementiert.

7.12.3. LoadCallback

Der LoadCallback wird nach dem Abschluss des asynchronen Ladens ausgeführt.

7.12.4. SaveCallback

Der SaveCallback wird nach dem Abschluss der asynchronen Speicherung ausgeführt.

7.12.5. ImportCallback

Der ImportCallback wird nach dem Abschluss des asynchronen Imports ausgeführt.

7.12.6. SerializationStrategy

Falls in Zukunft andere Speicherungsarten implementiert werden, können diese gegen das SerializationStrategy-Interface entwickelt werden. Momentan wird eine JAXB-Persistierung und den XMI-Import unterstützt.

7.12.7. OGVSerialization

Die Klasse OGVSerialization wird für das Speichern und Laden verwendet und wurde mit einer JAXB-Serialisierung umgesetzt.

7.12.8. XMISerialization

Die XMISerialization wurde von 3DCOV im Rahmen des XMI-Imports übernommen und angepasst. Sie implementiert die SerializationStrategy und repräsentiert einen XMI-Parser.

7.12.9. ColorAdapter

Die ColorAdapter-Klasse wird als XmlAdapter für die Color-Klasse von JavaFX im JAXB-Framework verwendet.

7.12.10. Point3DAdapter

Die Point3DAdapter-Klasse wird als XmlAdapter für die Point3D-Klasse von JavaFX im JAXB-Framework verwendet.

7.12.11. OGVPoint3D

Der OGVPoint3D wird im Point3DAdapter verwendet da die Klasse Point3D von JavaFX nicht serialisierbar ist.

7.12.12. VersionHandler

Der VersionHandler wurde von 3DCOV im Rahmen des XMI-Imports übernommen. Er liest die XMI-Version einer Datei.

7.12.13. XMIHandler

Der XMIHandler wurde von 3DCOV im Rahmen des XMI-Imports übernommen. Er ist für die Finalisierung des Imports zuständig.

7.12.14. XMI_1_1

Die XMI_1_1-Klasse wurde von 3DCOV im Rahmen des XMI-Imports übernommen und angepasst. Sie erweitert die XMIHandler-Klasse und liest ein XMI-File der Version 1.1 per SAX-Parser aus.

7.12.15. XMIRelation

Die XMIRelation-Klasse ist ein POJO um alle benötigten Werte einer Relation aus dem XMI-File zu kapseln.

7.12.16. UserPreferences

Die UserPreferences-Klasse wird verwendet um die benutzerspezifischen Einstellungen (z.B. Save-File-Pfad) zu speichern.

7.13. Util-Klassendiagramm

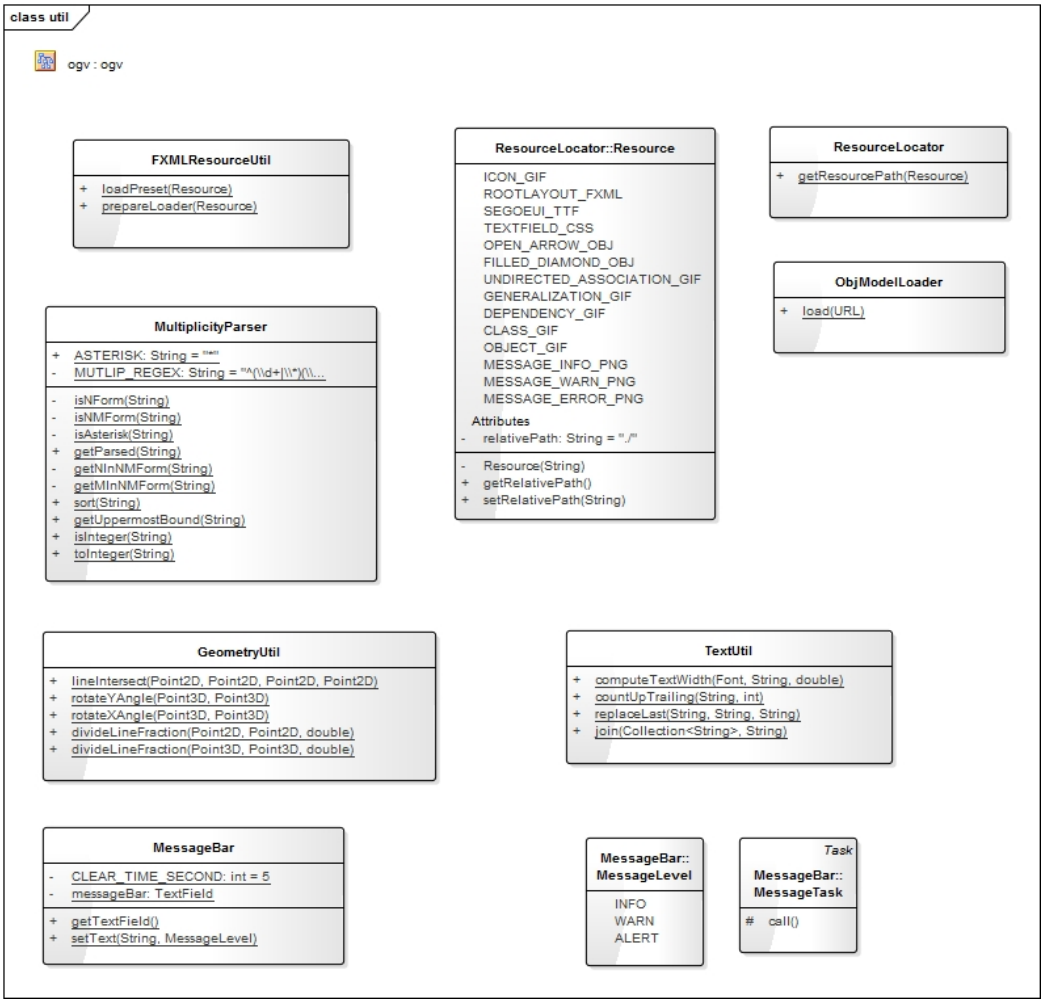


Abbildung 30: Klassendiagramm des Util Packages

7.14. Util Klassenkatalog

7.14.1. FXMLResourceUtil

Diese Klasse wird verwendet, um die einzelnen FXML Templates zu laden.

7.14.2. GeometryUtil

Die GeometryUtil-Klasse wird verwendet, um Berechnungen im 2D- und 3D-Raum, die an mehreren Stellen vorkommen, zusammenzufassen.

7.14.3. MessageBar

Mit der MessageBar können an den Benutzer gerichtete Informationen oder Fehlermeldungen ausgegeben werden. Sie wird in der Statusleiste unterhalb der Hauptanzeige / 3D-Szene eingefügt.

7.14.4. MultiplicityParser

Die MultiplicityParser-Klasse stellt Methoden für die Überprüfung und Anpassung einer Multiplizität einer Relation zur Verfügung.

7.14.5. ObjModelLoader

Diese Klasse wird verwendet, um die verschiedenen OBJ-Modell zu laden.

7.14.6. ResourceLocator

Der ResourceLocator wird gebraucht, um die Dateienpfade von Ressourcen wie FXML Templates, GIF Icons, JPG Icons und OBJ Modelle zu erhalten.

7.14.7. TextUtil

Die TextUtil-Klasse wird verwendet, um gewisse Hilfsmethoden, die mit Strings zusammenhängen, zu kapseln.

7.16. Externe Libraries

Das Projekt wird von 4 (bzw. 6) externen Libraries unterstützt, aufgelistet und beschrieben in folgender Tabelle:

Library	Version	Info
fxyzlib.jar	OGV-3.1	Beschreibung: JavaFX 3D Visualization and Component Library Verwendung: Cuboid 3D Primitive. Die Sourcen wurden angepasst und die Library gebildet, sodass Top Face des Cuboids nicht gerendert wird. Links: http://birdasaur.github.io/FXyz/ https://github.com/FXyz/FXyzLib
jfxtras-labs-8.0-r3.jar	8.0-r3	Beschreibung: JavaFX Unterstützung, viele Hilfsfunktionen Verwendung: Umwandlung von JavaFX Color nach Webcode oder CSS-Code und umgekehrt. Links: http://jfxtras.org http://central.maven.org/maven2/org/jfxtras/jfxtras-labs/
objimporterjfx.jar	0.8	Beschreibung: Erlaubt es 3D Modelle (u. a.) im Format *.obj als JavaFX Mesh zu importieren. Verwendung: Import von Modellen für die Pfeilenden. (z. B. Composition-Rhombus) Link: http://www.interactivemesh.org/models/jfx3dimporter.html
log4j-1.2.17.jar slf4j-api-1.7.10.jar slf4j-log4j12-1.7.10.jar	1.2.17 1.7.10 1.7.10	Beschreibung: Simple Logging Facade for Java (SLF4J) Verwendung: Als Logging-Framework Link: http://www.slf4j.org/

8. Implementation

Dieser Abschnitt beschreibt ausgesuchte Implementationen und behandelt interessante sowie herausfordernde Aspekte aus diesem Projekt.

8.1. Visualisierung

8.1.1. User Interface

Das Graphical User Interface (GUI) des Object Graph Visualizers konnte komplett mit der JavaFX-Technologie umgesetzt werden. Die Oberfläche besteht aus einer Menüleiste und einer Werkzeugleiste oben und einer Statusleiste unten. Dazwischen ist die 3D-Szene als sogenannte SubScene eingebettet. Die Bedienoberfläche (also Menü-, Werkzeug- und Statusleiste) wurde mit Hilfe des JavaFX Scene Builder zusammengestellt. Mit diesem Visual Layout Tool lassen sich alle grafischen Elemente und Kontrollkomponenten, wie zum Beispiel Panels, Menüpunkte, Labels und Buttons, in der Szene platzieren. Allen Elementen lässt sich ausserdem eine freiwählbare fx:id geben und im Falle von Kontrollkomponenten [5] auch Methodennamen für verschiedene Events. Untenstehende Abbildung zeigt einen Screenshot des Scene Builders. Der Button «Create Object» ist angewählt. In der rechten Spalte sieht man, dass der Button die fx:id «createObject» hat und bei einem On Action Event, d. h. bei einem Klick auf den Button, die Methode «handleCreateObject» aufgerufen wird:

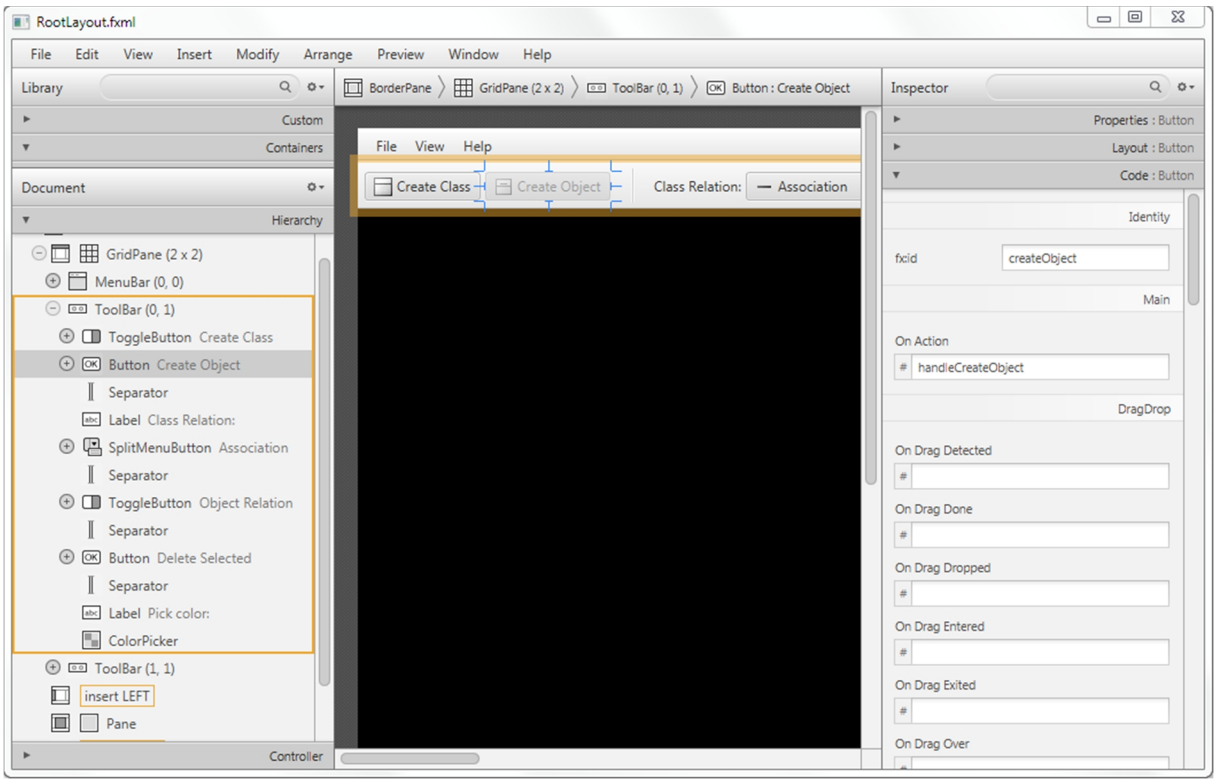


Abbildung 31: Screenshot des JavaFX Scene Builder. Gezeigt wird ein Ausschnitt der Bedienoberfläche (RootLayout.fxml).

Der Scene Builder generiert aus dem erstellten User Interface eine FXML Datei. FXML ist eine auf XML basierende Auszeichnungssprache. Sie ähnelt dem bekannteren *WPF* (Windows Presentation Foundation) von der .NET Plattform. Das FXML Template wird zur Laufzeit in die Applikation geladen, die Elemente werden dem bestehenden Scene Graph hinzugefügt und anschliessend gerendert.

```
<Button fx:id="createObject" disable="true" mnemonicParsing="false"
         onAction="#handleCreateObject" text="Create Object">
    <graphic>
        <ImageView pickOnBounds="true" preserveRatio="true">
            <image>
                <Image url="@../images/menu/instance.gif" />
            </image>
        </ImageView>
    </graphic>
</Button>
```

Code Snippet 2: Ausschnitt aus dem RootLayout.fxml. Zeigt die FXML Umsetzung des «Create Object» Buttons.

Im Quellcode kann auf die spezifizierten fx:id's und Methodendeklarationen mittels @FXML-Annotation referenziert werden (siehe Code Snippet 3). Diese Auslagerung von GUI-Komponenten und Layout in ein FXML mit anschliessender Dependecy Injection hat den Vorteil einer besseren Trennung zwischen der View- und den Input-Controllern.

```
@FXML
private Button createObject;

@FXML
private void handleCreateObject() {
    toggleToolbar(null);
    if (selectionController.hasCurrentSelection) {
        Selectable selected = selectionController.getCurrentSelected();
        PaneBox newPaneBox = mvConnector.handleCreateNewObject(selected);
        if (newPaneBox != null) {
            new QuickCreationController(newPaneBox, mvConnector);
            selectionController.setSelected(newPaneBox, true, subSceneAdapter);
        }
    }
}
```

Code Snippet 3: Java Code Snippet aus der ViewController Klasse. Gezeigt wird die Referenzierung mittels @FXML Annotation für das Attribute «createObject» und die Methode «handleCreateObject».

8.1.2. Klassen und Objekten

Die Klassen und Objekte sollen in der 3D-Szene durch Quader repräsentiert werden. Die Beschriftung wird auf die Oberseite des Quaders (dem sog. Top Face) gezeichnet. Ziel ist es, dass die Klasse oder das Objekt von oben betrachtet wie in der UML-Spezifikation aussieht.

Für Quader-3D-Primitiven gibt es in JavaFX die Klasse «Box». Die Idee war, diese Klasse zu verwenden und auf die Boxoberfläche ein gestaltetes Panel zu legen. Es hat sich aber herausgestellt, dass es bei dieser Technik beim Rendering des Top Face zu unschönen Überlagerungseffekten kommt.

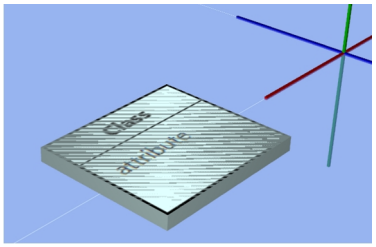


Abbildung 32: Überlagerungseffekt bei Verwendung der JavaFX Box Klasse.

Um diesen Effekt zu vermeiden, muss die Oberseite des Quaders zuerst entfernt werden. Dazu müsste jedoch anstelle der Box das ganze Quader-Trianglemesh selbst berechnet und gezeichnet werden. Glückerweise konnte auf die freie Library FXYZ (siehe Kapitel 7.16) zurückgegriffen werden. Die Source enthält die Klasse CuboidMesh, welche diesen Zweck erfüllt. Diese Klasse wurde angepasst und in der eigenen Cuboid-Klasse adaptiert.

Die Klasse PaneBox verbindet dann das Top Face Panel mit dem Cuboid über Propertybinding. Position und Dimension werden so immer gemeinsam verändert:

```
private void buildBox() {
    box = new Cuboid(INIT_DEPTH);
    box.setDrawTopFace(false);
    box.getTransforms().add(new Rotate(90, Rotate.X_AXIS));
    box.widthProperty().bind(borderPane.widthProperty());
    box.heightProperty().bind(borderPane.heightProperty());

    DoubleBinding halfWidth = borderPane.widthProperty().divide(2);
    DoubleBinding halfHeight = borderPane.heightProperty().divide(2);
    double halfDepth = INIT_DEPTH / 2; // depth doesn't change
    DoubleBinding transX = borderPane.translateXProperty().subtract(halfWidth);
    DoubleBinding transY = borderPane.translateZProperty().subtract(halfHeight);
    DoubleBinding transZ = borderPane.translateYProperty().subtract(halfDepth);

    box.translateXProperty().bind(transX);
    box.translateZProperty().bind(transY);
    box.translateYProperty().bind(transZ);
}
```

Code Snippet 4:Java Code Snippet aus der PaneBox Klasse. Zeigt die Erstellung der Cuboid-Box und das Propertybinding an das BorderPane (Top Face Panel).

8.1.3. Relationen

Eine gute Visualisierung der Relationen stellte sich als Herausforderung heraus. Es wurde anfangs versucht, die benötigten Linien, Pfeile und Rhomben mit 2D-Linien zu zeichnen. Jedoch sahen diese in der 3D-Szene nicht ansprechend aus. Die 2D-Linien wirken durch das Antialiasing verpixelt und können nicht von allen Seiten betrachtet werden. Wir sind zur Erkenntnis gekommen, dass die Relationen ebenfalls komplett in 3D gerendert werden müssen.

Linien

Für die Linien wird wiederum auf die JavaFX-Box-Primitive gesetzt. Eine kleine Box kann beliebig in die Länge gestreckt werden und sieht dann wie eine Linie aus. Diese Streckung geschieht aber immer vom Mittelpunkt ausgehend in beide Richtungen, was die korrekte Platzierung der Relation zwischen der Source- und Targetbox erschwert. Ausserdem muss berücksichtigt werden, dass zwischen den Boxen mehrere Relationen auftreten können. Mehrfachbeziehungen müssen mit einem Abstand in die Szene gesetzt werden.

Die Berechnung von Start- und Endkoordinaten einer Relation ist ein komplexer Vorgang, welcher bei jeder Verschiebung und Grössenänderung von Source- und Targetbox erneut durchgeführt werden muss:

1. Ausgehend von den Mittelpunktkoordinaten der Sourcebox und der Targetbox wird eine Referenzlinie gebildet. Sie verbindet die beiden Mittelpunkte.

2. Um beide Mittelpunkte wird jeweils ein Kreis gelegt, dessen Durchmesser der kleinsten Boxaussenseite entspricht.

3. Orthogonal auf die Referenzlinie wird eine Zentrale (Sekante durch den Mittelpunkt) gelegt.

4. Diese Zentrale wird anschliessend durch die Anzahl Verbindungen + 1 geteilt.

5. Durch diese Teilung entstehen Punkte, welche die neuen Referenzpunkte der Relationen darstellen.

6. Anhand der Referenzpunkte können die Schnittpunkte mit der Aussenkante der Box eruiert werden. Sie bilden die tatsächlichen Start- und Endpunkte der Relationen.

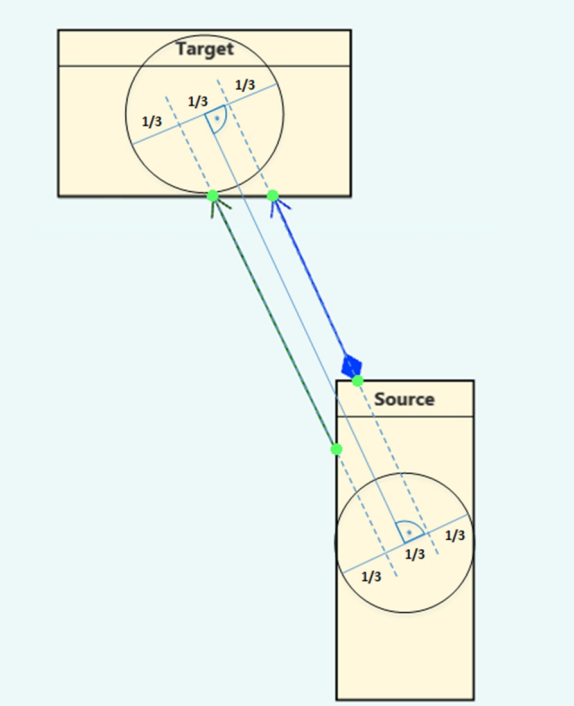


Abbildung 33: Die Abbildung veranschaulicht den Berechnungsvorgang der Start- und Endkoordinaten (in grün) für zwei Relationen zwischen einer Source- und einer Target-Klasse.

Pfeile und Rhomben

Abgesehen von ungerichteten Assoziationen, werden die Relationen in der UML mit unterschiedlichen Pfeilspitzen dargestellt. Eine Komposition oder eine Aggregation hat zusätzlich einen ausgefüllten bzw. nicht ausgefüllten Rhombus am Startpunkt. Für diese Körper gibt es keine passenden JavaFX-3D-Primitiven.

Dieses Problem wurde mit Hilfe der ObjImporter-Library gelöst. Die Bibliothek erlaubt OBJ-Modelle als JavaFX Meshs zu importieren. Die OBJ-Modelle für alle Pfeilspitzen und Rhomben wurden mit der 3D-Grafiksoftware «SketchUp» modelliert:

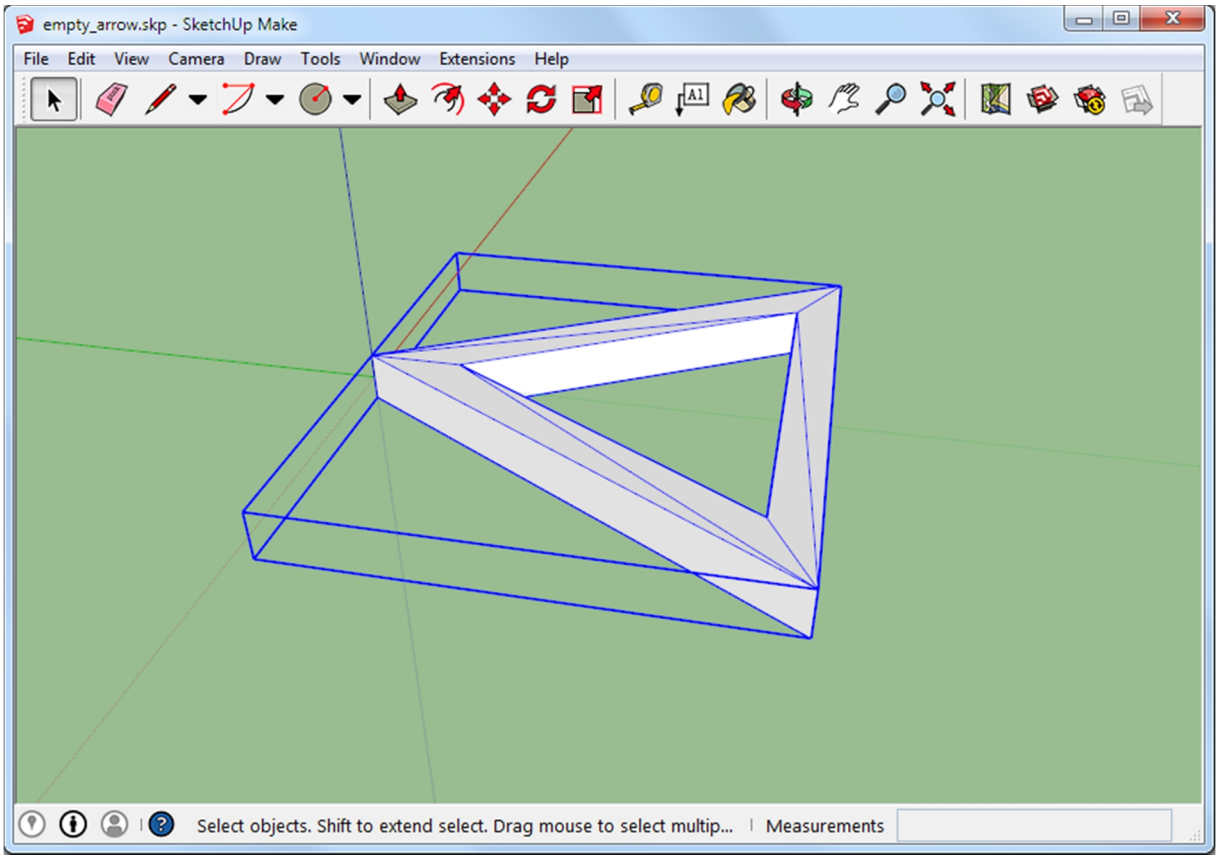


Abbildung 34: Screenshot von SketchUp. Gezeigt wird das Modell einer Pfeilspitze für eine Generalisation.

```
public static Node[] load(URL modelUrl) {
    Node[] mesh = {};
    if (modelUrl != null) {
        ObjModelImporter tdsImporter = new ObjModelImporter();
        try {
            tdsImporter.read(modelUrl);
        }
        catch (ImportException e) {
            Logger.debug(e.getMessage());
        }
        mesh = tdsImporter.getImport();
    }
    return mesh;
}
```

Code Snippet 5: Java Code Snippet aus der ObjModelLoader Klasse für den Import eines OBJ-Modells als JavaFX Mesh.

8.1.4. Selektion

Eine Box (Klasse oder Objekt) oder ein Label (Attribut, Multiplizität oder Rolle) ist selektierbar. Das Ziel ist es, die Selektion einheitlich für diese beiden Typen darzustellen. JavaFX bietet allerdings nur für die Labels eine Selektionsfunktion, für die 3D-Primitiven fehlt etwas entsprechendes.

Aus diesem Grund wurde für die selektierbaren Elemente ein Interface geschaffen:

```
public interface Selectable {
    public void setSelected(boolean selected);
    public boolean isSelected();
    public Group getSelection();
    public void requestFocus();
}
```

Code Snippet 6: Java Code Snippet des Selectable Interface.

Die Selektionslogik erfolgt im SelectionController: Bei einem Mausklick-Event wird auf dem Element setSelected(true) aufgerufen. Bei Labeln ist es die bestehende JavaFX Selektionsfunktion, bei den 3D-Primitiven die eigene Implementierung.

Für Klassen und Objekte wird um die Box ein blauer Rahmen eingeblendet. Der Rahmen besteht aus Zylindern für die Seiten und Kugeln für die Ecken. Bei Relationen wird die Verbindungslinie blau eingefärbt und die Start- und Endpunkte durch Kugeln hervorgehoben.

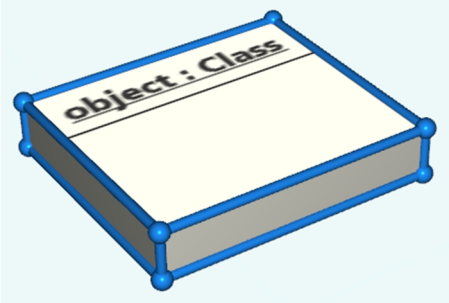


Abbildung 36: Ein selektiertes Objekt.

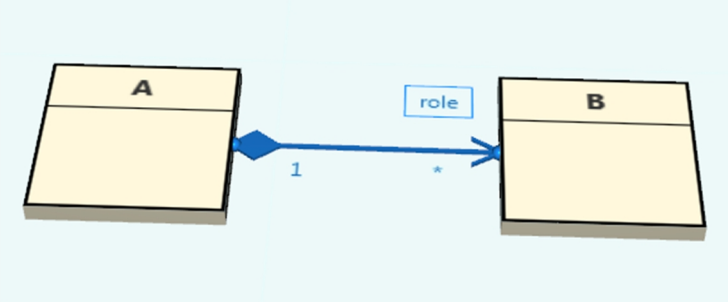


Abbildung 35: Die Abbildung zeigt eine selektierte Rolle. Die dazugehörige Relation wird ebenfalls blau gefärbt.

8.2. Persistenz

8.2.1. Speichern und Laden

Für das Speichern und Laden des Zustands wurde das SerializationStrategy-Interface eingeführt. Dieses Interface markiert alle nach dem Strategy-Pattern implementierten Klassen, welche für das Speichern und Laden verwendet werden können. Diese Klassen müssen die beiden Sets mit ModelClasses und Relations entweder mit Daten befüllen oder auslesen und abspeichern können.

```
public interface SerializationStrategy {
    public boolean parse(File file);
    public boolean serialize(File file);
    public Set<ModelClass> getClasses();
    public Set<Relation> getRelations();
    public void setClasses(Set<ModelClass> modelClasses);
    public void setRelations(Set<Relation> relations);
}
```

Code Snippet 7: Java Code Snippet des SerializationStrategy Interface.

Momentan gibt es zwei Klassen, welche dieses Interface umsetzen:

OGVSerialization

Als Standard-Serialization werden mittels JAXB die ModelClasses und Relations im XML-Format persistiert [8][9][10]. Die entsprechenden Annotations sind in der OGVSerialization zu finden.

```
@XmlElementWrapper(name = "classes")
@XmlElement(name = "class")
@Override
public Set<ModelClass> getClasses() {
    return this.classes;
}
```

Code Snippet 8: Java Code Snippet der JAXB-Annotations.

XMISerialization

Da der XMI-Import als Laden-Funktion angesehen werden kann, implementiert diese ebenfalls das SerializationStrategy-Interface. Der Mechanismus dieses XMI-Imports konnte als einzige Komponente aus dem 3DCOV übernommen werden. Die entsprechenden Klassen wurden für die Verwendung im OGV angepasst. Neu werden auch gerichtete Relationen korrekt geparst.

Einen XMI-Export wurde nicht implementiert, da je nach Zielprogramm (z. B. für Enterprise Architect) spezifische XMI-IDs generiert werden müssten.

9. Testing

9.1. Codetests

9.1.1. Eingesetzte Werkzeuge

Um eine stetige Korrektheit der einzelnen Komponenten sicherzustellen, wurden automatisierte Tests eingesetzt. Für das Unittesting wurde JUnit 4 eingesetzt. Um die Testabdeckung zu überwachen und zu visualisieren, wurde das Eclipse-Plugin EclEmma 2.3.2 verwendet.

9.1.2. Unittests

Parallel zum src-Verzeichnis wurde ein test-Verzeichnis erstellt. In diesem befinden sich alle JUnit-Testklassen zum Projekt. Die Testklassen sind nach der zu testenden Klasse benannt und haben einen «Test»-Postfix. Der Fokus wurde auf die Model- und Util-Klassen gelegt, das heisst Klassen mit wenig Bezug auf das GUI. Die restliche Funktionalität des Codes wird mit Systemtests abgedeckt.

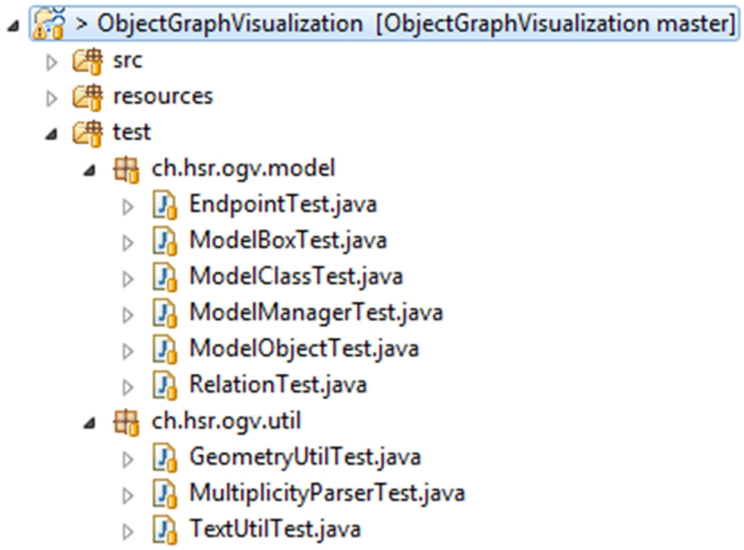


Abbildung 37: Testklassen im Package Explorer

Alle erstellten Testmethoden bestehen den Durchlauf erfolgreich.

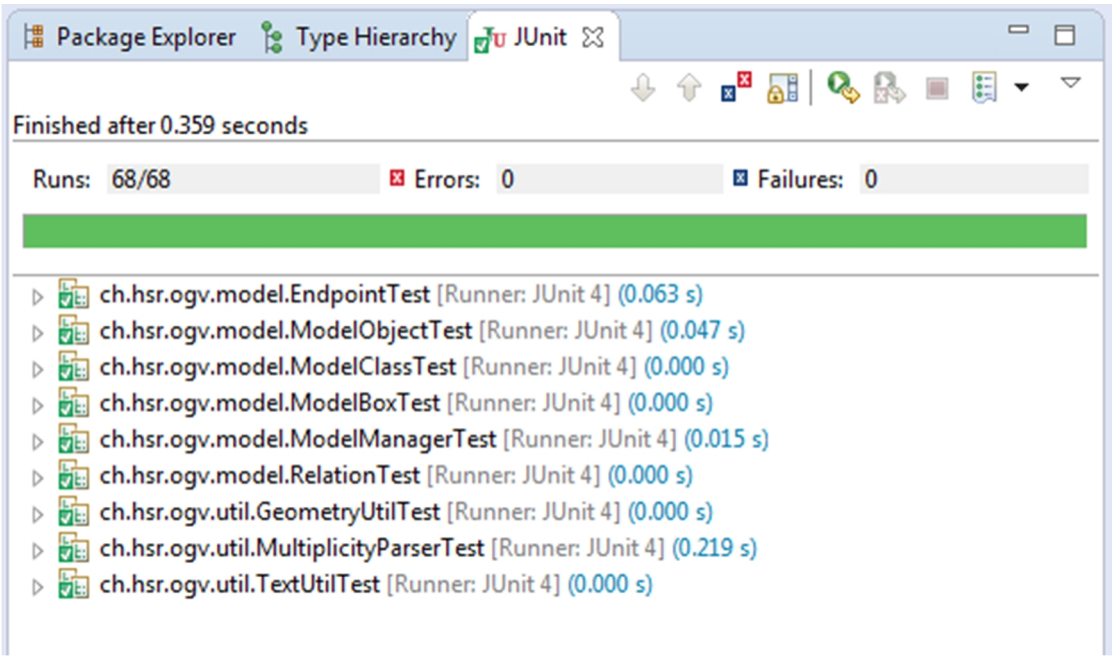
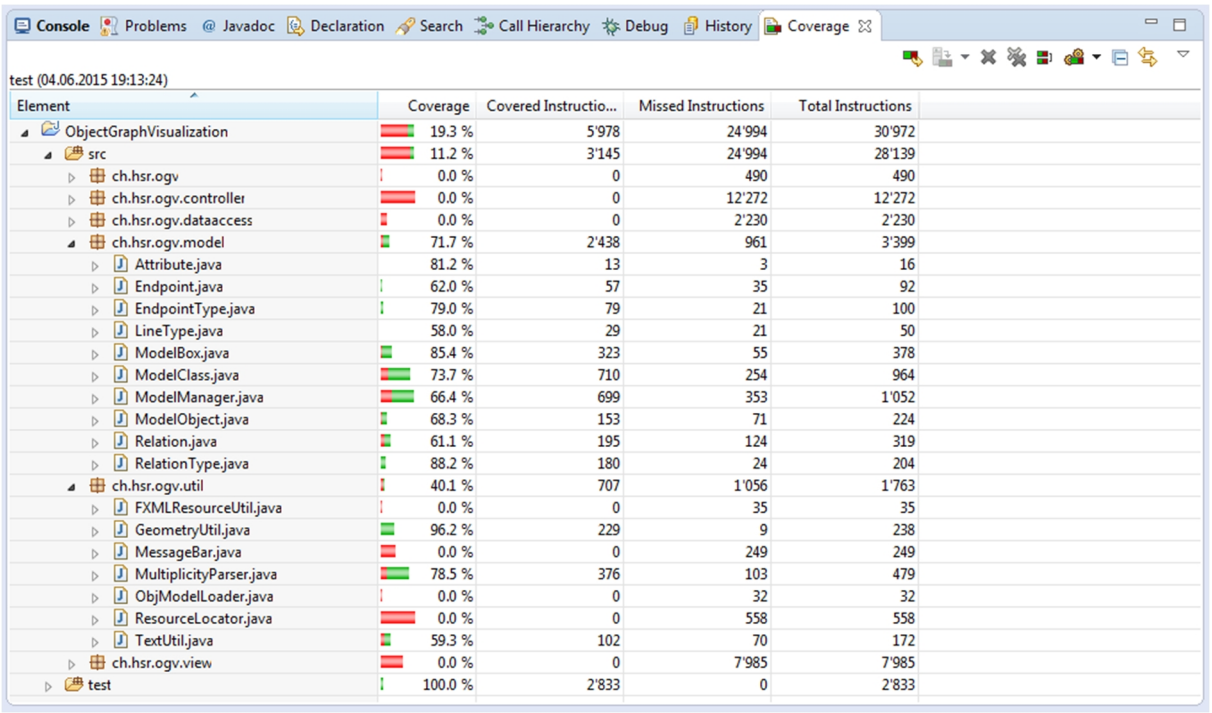


Abbildung 38: Testdurchführung mit JUnit

9.1.3. Testabdeckung

Bei der Testabdeckung wurde der Fokus ebenfalls auf die Model- und Util-Klassen gelegt. Es wurde je nach Funktionalität eine Testabdeckung von 60% bis 80% angestrebt.



Element	Coverage	Covered Instructio...	Missed Instructions	Total Instructions
ObjectGraphVisualization	19.3 %	5'978	24'994	30'972
src	11.2 %	3'145	24'994	28'139
ch.hsr.ogv	0.0 %	0	490	490
ch.hsr.ogv.controller	0.0 %	0	12'272	12'272
ch.hsr.ogv.dataaccess	0.0 %	0	2'230	2'230
ch.hsr.ogv.model	71.7 %	2'438	961	3'399
Attribute.java	81.2 %	13	3	16
Endpoint.java	62.0 %	57	35	92
EndpointType.java	79.0 %	79	21	100
LineType.java	58.0 %	29	21	50
ModelBox.java	85.4 %	323	55	378
ModelClass.java	73.7 %	710	254	964
ModelManager.java	66.4 %	699	353	1'052
ModelObject.java	68.3 %	153	71	224
Relation.java	61.1 %	195	124	319
RelationType.java	88.2 %	180	24	204
ch.hsr.ogv.util	40.1 %	707	1'056	1'763
FXMLResourceUtil.java	0.0 %	0	35	35
GeometryUtil.java	96.2 %	229	9	238
MessageBar.java	0.0 %	0	249	249
MultiplicityParser.java	78.5 %	376	103	479
ObjModelLoader.java	0.0 %	0	32	32
ResourceLocator.java	0.0 %	0	558	558
TextUtil.java	59.3 %	102	70	172
ch.hsr.ogv.view	0.0 %	0	7'985	7'985
test	100.0 %	2'833	0	2'833

Abbildung 39: Testabdeckung mit EclEmma

9.2. Systemtests

9.2.1. Systemtestspezifikation

Hier folgt eine tabellarische Beschreibung der einzelnen Tests der Use Cases.

ST Nr.	UC Nr.	Use Case	Beschreibung des Tests	Erwartetes Resultat
ST 01	UC 01	Klasse erstellen	Klasse via Toolbar erstellen	OK
ST 02	UC 01	Klasse erstellen	Klasse via Kontextmenu erstellen	OK
ST 03	UC 01	Klasse erstellen	Klasse mit kleingeschriebenen Klassennamen erstellen	Warnung
ST 04	UC 01	Klasse erstellen	Klasse ohne Klassennamen erstellen	Warnung
ST 05	UC 01	Klasse erstellen	Klasse mit «Quick-Creation» erstellen	OK
ST 06	UC 01	Klasse erstellen	Klasse mit «C» + Linksklick erstellen	OK
ST 07	UC 02	Klasse editieren	Klassennamen per Doppelklick editieren	OK
ST 08	UC 02	Klasse editieren	Klassennamen per Kontextmenü editieren	OK
ST 09	UC 02	Klasse editieren	Attributnamen per Doppelklick editieren	OK
ST 10	UC 02	Klasse editieren	Attributnamen per Kontextmenü editieren	OK
ST 11	UC 02	Klasse editieren	Attribut hinzufügen	OK
ST 12	UC 02	Klasse editieren	Attributposition nach oben schieben	OK
ST 13	UC 02	Klasse editieren	Attributposition nach unten schieben	OK
ST 14	UC 02	Klasse editieren	Attribut via Toolbar löschen	OK
ST 15	UC 02	Klasse editieren	Attribut via Kontextmenü löschen	OK
ST 16	UC 02	Klasse editieren	Attribut mit «Delete» löschen	OK
ST 17	UC 03	Klasse löschen	Klasse via Toolbar löschen	OK
ST 18	UC 03	Klasse löschen	Klasse via Kontextmenu löschen	OK
ST 19	UC 03	Klasse löschen	Klasse mit «Delete» löschen	OK
ST 20	UC 03	Klasse löschen	Klasse mit Objekten löschen. Objs. werden mitgel.	OK
ST 21	UC 04	Klassenrelation erstellen	Klassen via Toolbar mit Klassenrelationen verbinden	OK
ST 22	UC 04	Klassenrelation erstellen	Klassen via KM. mit Klassenrelationen verbinden	OK
ST 23	UC 04	Klassenrelation erstellen	Klassen mit zyklischen Gen. verbinden	Nicht möglich
ST 24	UC 04	Klassenrelation erstellen	Klasse reflexiv mit allem ausser Gen. verbinden	OK
ST 25	UC 04	Klassenrelation erstellen	Klasse reflexiv mit Generalization verbinden	Nicht möglich
ST 26	UC 04	Klassenrelation erstellen	Klasse mit einem Objekt verbinden	Nicht möglich
ST 27	UC 04	Klassenrelation erstellen	Während dem Verbinden mit «ESC» abbrechen	OK
ST 28	UC 05	Klassenrelation editieren	Change Direction erstes Mal	OK
ST 29	UC 05	Klassenrelation editieren	Change Direction ein weiteres Mal	OK
ST 30	UC 05	Klassenrelation editieren	Set Role erstes Mal	OK
ST 31	UC 05	Klassenrelation editieren	Set Role via Kontextmenü ein weiteres Mal	OK
ST 32	UC 05	Klassenrelation editieren	Set Role via Doppelklick ein weiteres Mal	OK
ST 33	UC 05	Klassenrelation editieren	Set Muliplicity erstes Mal	OK

ST 34	UC 05	Klassenrelation editieren	Set Muliplicity via Kontextmenü ein weiteres Mal	OK
ST 35	UC 05	Klassenrelation editieren	Set Muliplicity via Doppelklick ein weiteres Mal	OK
ST 36	UC 06	Klassenrelation löschen	Klassenrelation via Toolbar löschen	OK
ST 37	UC 06	Klassenrelation löschen	Klassenrelation via Kontextmenu löschen	OK
ST 38	UC 06	Klassenrelation löschen	Klassenrelation mit «Delete» löschen	OK
ST 39	UC 06	Klassenrelation löschen	Generalization löschen, Subobjs. werden mitgel.	OK
ST 40	UC 07	Objekt erstellen	Objekt via Toolbar erstellen	OK
ST 41	UC 07	Objekt erstellen	Objekt via Kontextmenu erstellen	OK
ST 42	UC 07	Objekt erstellen	Objekt mit «O» erstellen	OK
ST 43	UC 07	Objekt erstellen	Objekt mit «Quick-Creation» erstellen	OK
ST 44	UC 08	Objekt editieren	Objektnamen per Doppelklick editieren	OK
ST 45	UC 08	Objekt editieren	Objektnamen per Kontextmenü editieren	OK
ST 46	UC 08	Objekt editieren	Attributwert per Doppelklick editieren	OK
ST 47	UC 08	Objekt editieren	Attributwert per Kontextmenü editieren	OK
ST 48	UC 09	Objekt löschen	Objekt via Toolbar löschen	OK
ST 49	UC 09	Objekt löschen	Objekt via Kontextmenu löschen	OK
ST 50	UC 09	Objekt löschen	Objekt mit «Delete» löschen	OK
ST 51	UC 09	Objekt löschen	Objekt mit Subobjekt löschen. Subobj. wird mitgel.	OK
ST 52	UC 10	Objektrelation erstellen	Obj. via Toolbar mit Objektrel. verbinden	OK
ST 53	UC 10	Objektrelation erstellen	Obj. via KM. mit Objektrel. verbinden	OK
ST 54	UC 10	Objektrelation erstellen	Obj. mit Obj. verbinden ohne Klassenrel. besitzen	Nicht möglich
ST 55	UC 11	Objektrelation löschen	Objektrelation via Toolbar löschen	OK
ST 56	UC 11	Objektrelation löschen	Objektrelation via Kontextmenu löschen	OK
ST 57	UC 11	Objektrelation löschen	Objektrelation mit «Delete» löschen	OK
ST 58	UC 12	Zustand speichern	Zustand mit Klassen, Objs. und Relationen speichern	OK
ST 59	UC 12	Zustand speichern	Zustand ohne Elemente speichern	OK
ST 60	UC 13	Zustand laden	Zustand mit Klassen, Objs. und Relationen laden	OK
ST 61	UC 13	Zustand laden	Zustand ohne Elemente laden	OK
ST 62	UC 14	XMI-File importieren	XMI-File von Enterprise Architect importieren	OK
ST 63	UC 15	Object Graph erzeugen	Obj.Gr. von Obj. mit 1-Multiplizität erzeugen	OK
ST 64	UC 15	Object Graph erzeugen	Obj.Gr. von Obj. mit >1-Multiplizität erzeugen	OK
ST 65	UC 15	Object Graph erzeugen	Obj.Gr. von Obj. mit *-Multiplizität erzeugen	OK
ST 66	UC 15	Object Graph erzeugen	Object Graph ohne Objekte erzeugen	OK
ST 67	UC 16	Object Graph ausblender	Object Graph ausblenden	OK

9.2.2. Angaben zur Durchführung

Die Systemtests wurden anfangs Juni auf einem Übungsrechner der HSR durchgeführt.

9.2.3. Protokoll

Hier folgt das tabellarische Protokoll der einzelnen Tests der Systemtestspezifikation.

ST Nr.	Beschreibung des Tests	Erwartetes Resultat eingetroffen
ST 01	Klasse via Toolbar erstellen	Ja
ST 02	Klasse via Kontextmenu erstellen	Ja
ST 03	Klasse mit kleingeschriebenen Klassennamen erstellen	Ja
ST 04	Klasse ohne Klassennamen erstellen	Ja
ST 05	Klasse mit «Quick-Creation» erstellen	Ja
ST 06	Klasse mit «C» + Linksklick erstellen	Ja
ST 07	Klassennamen per Doppelklick editieren	Ja
ST 08	Klassennamen per Kontextmenü editieren	Ja
ST 09	Attributnamen per Doppelklick editieren	Ja
ST 10	Attributnamen per Kontextmenü editieren	Ja
ST 11	Attribut hinzufügen	Ja
ST 12	Attributposition nach oben schieben	Ja
ST 13	Attributposition nach unten schieben	Ja
ST 14	Attribut via Toolbar löschen	Ja
ST 15	Attribut via Kontextmenü löschen	Ja
ST 16	Attribut mit «Delete» löschen	Ja
ST 17	Klasse via Toolbar löschen	Ja
ST 18	Klasse via Kontextmenu löschen	Ja
ST 19	Klasse mit «Delete» löschen	Ja
ST 20	Klasse mit Objekten löschen. Objs. werden mitgel.	Ja
ST 21	Klassen via Toolbar mit Klassenrelationen verbinden	Ja
ST 22	Klassen via KM. mit Klassenrelationen verbinden	Ja
ST 23	Klassen mit zyklischen Gen. verbinden	Ja
ST 24	Klasse reflexiv mit allem ausser Gen. verbinden	Ja
ST 25	Klasse reflexiv mit Generalization verbinden	Ja
ST 26	Klasse mit einem Objekt verbinden	Ja
ST 27	Während dem Verbinden mit «ESC» abbrechen	Ja
ST 28	Change Direction erstes Mal	Ja
ST 29	Change Direction ein weiteres Mal	Ja
ST 30	Set Role erstes Mal	Ja
ST 31	Set Role via Kontextmenü ein weiteres Mal	Ja
ST 32	Set Role via Doppelklick ein weiteres Mal	Ja

ST 33	Set Muliplicity erstes Mal	Ja
ST 34	Set Muliplicity via Kontextmenü ein weiteres Mal	Ja
ST 35	Set Muliplicity via Doppelklick ein weiteres Mal	Ja
ST 36	Klassenrelation via Toolbar löschen	Ja
ST 37	Klassenrelation via Kontextmenu löschen	Ja
ST 38	Klassenrelation mit «Delete» löschen	Ja
ST 39	Generalization löschen, Subobjs. werden mitgel.	Ja
ST 40	Objekt via Toolbar erstellen	Ja
ST 41	Objekt via Kontextmenu erstellen	Ja
ST 42	Objekt mit «O» erstellen	Ja
ST 43	Objekt mit «Quick-Creation» erstellen	Ja
ST 44	Objektnamen per Doppelklick editieren	Ja
ST 45	Objektnamen per Kontextmenü editieren	Ja
ST 46	Attributwert per Doppelklick editieren	Ja
ST 47	Attributwert per Kontextmenü editieren	Ja
ST 48	Objekt via Toolbar löschen	Ja
ST 49	Objekt via Kontextmenu löschen	Ja
ST 50	Objekt mit «Delete» löschen	Ja
ST 51	Objekt mit Subobjekt löschen. Subobj. wird mitgel.	Ja
ST 52	Obj. via Toolbar mit Objektrel. verbinden	Ja
ST 53	Obj. via KM. mit Objektrel. verbinden	Ja
ST 54	Obj. mit Obj. verbinden ohne Klassenrel. besitzen	Ja
ST 55	Objektrelation via Toolbar löschen	Ja
ST 56	Objektrelation via Kontextmenu löschen	Ja
ST 57	Objektrelation mit «Delete» löschen	Ja
ST 58	Zustand mit Klassen, Objs. und Relationen speichern	Ja
ST 59	Zustand ohne Elemente speichern	Ja
ST 60	Zustand mit Klassen, Objs. und Relationen laden	Ja
ST 61	Zustand ohne Elemente laden	Ja
ST 62	XMI-File von Enterprise Architect importieren	Ja
ST 63	Obj.Gr. von Obj. mit 1-Multiplizität erzeugen	Ja
ST 64	Obj.Gr. von Obj. mit >1-Multiplizität erzeugen	Ja
ST 65	Obj.Gr. von Obj. mit *-Multiplizität erzeugen	Ja
ST 66	Object Graph ohne Objekte erzeugen	Ja
ST 67	Object Graph ausblenden	Ja

10. Weiterentwicklung

Dieses Kapitel beleuchtet die Möglichkeiten der Weiterentwicklung des Object Graph Visualizers, ist jedoch – im Gegensatz zu Kapitel 4.3 im Technischen Bericht – an Architekten und Software Entwickler gerichtet und weniger allgemein. Bei einigen Punkten werden Überlegungen angestellt, wie Erweiterungen angegangen und in die bestehende Architektur am besten eingebunden werden können.

Oft werden wünschenswerte Funktionalitäten beschrieben, die wegen tieferer Priorität und Zeitmangel nicht mehr umgesetzt werden konnten.

10.1. Known Issues

Im Folgenden werden drei bekannte Bugs aufgelistet. Eine Bugbeseitigung wäre mit höherem Aufwand verbunden bei relativ geringem Nutzen.

10.1.1. Vererbte Referenzen im Objekt Graph

Es ist möglich die Objekte zweier Klassen mit einer Objektrelation zu verbinden, auch wenn die Klassen nur indirekt über die Vererbung verknüpft sind. Diese Referenz wird jedoch im Object Graph Mode nicht angezeigt.

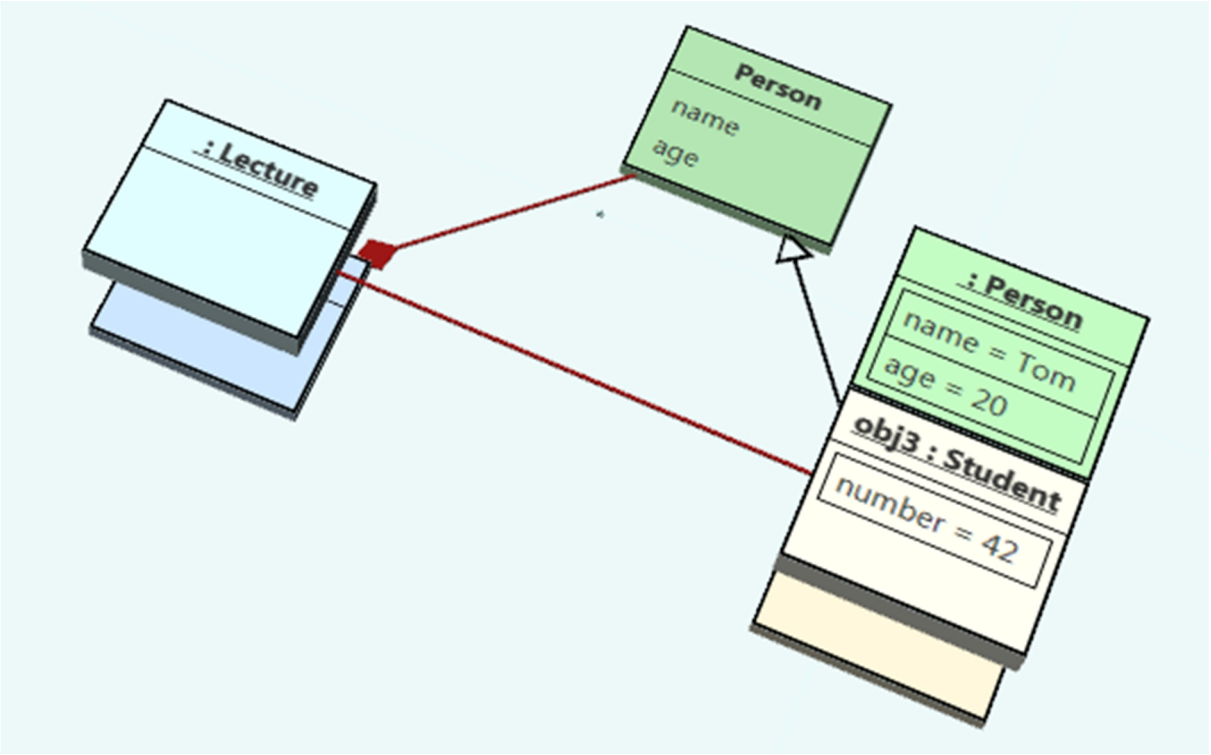


Abbildung 40: Screenshot einer Situation, die im Object Graph Modus nicht korrekt umgesetzt wird (rote Objektrelation)

10.1.2. Attributwerte von vererbten Objekten

Zurzeit werden die «Teilobjekte» einer Superklasse – die bei den Objekten der Subklassen angehängt werden – bei der Erstellung einer Vererbungsrelation neu berechnet. Dies hat den unerwünschten Effekt, dass dessen Attributwerte beim Laden eines .ogv-Files nicht berücksichtigt werden.

10.1.3. Mausklick Registrierung

Um zum Beispiel eine Klasse zu selektieren oder das Kontextmenu zu öffnen, muss ein kurzer Links- bzw. Rechtsklick registriert werden können. Das Problem tritt dann auf, wenn sich die Maus vor dem Loslassen der Maustaste auch nur um wenige Pixel bewegt hat: Der MouseEvent wird nicht mehr als «MOUSE_CLICKED» registriert, sondern als Mausbewegung «DRAG_DETECTED». Dieser Event für Mausbewegung ist jedoch unter anderem schon für eine Verschiebung von Klassen vorgesehen. Die Selektion bzw. das Öffnen des Kontextmenus findet folglich nicht statt.

Dieser Bug müsste in der SelectionController Klasse per if-Abfrage behoben werden. Die JavaFX MouseEvent Klasse kennt eine Methode «isStillSincePress()» welche genau für diesen Fall vorgesehen ist.

10.2. Performance

10.2.1. 3D-Elemente

Das 3D-Rendering ist ein sehr ressourcenintensiver Vorgang, welcher die Applikation jedoch mit der bestehenden Umsetzung in den meisten Fällen problemlos bewältigen kann. Vor allem die Kameraführung läuft auch bei einer hohen Zahl an Elementen in der Szene noch flüssig.

Bei einer Bewegung oder Grössenänderung von Klassen mit vielen Objekten kann es hingegen zu spürbaren Performanceeinbussen kommen. Da dieser Fall selten vorkommen sollte, wurde die genaue Ursache nicht ermittelt. Es wird auch ein Zusammenhang mit der Hardwareauslastung vermutet, da vor allem die Grafikkarte schnell an ihre Grenzen kommt. Um diese zu entlasten, wäre es sinnvoll, das Verschieben von Klassen bzw. das Resizing nur anzudeuten und die tatsächlichen Positionsberechnungen erst dann einzuleiten, wenn der Benutzer die Bewegung abgeschlossen hat.

10.2.2. Lade- und Importvorgang

Konkrete Verbesserungsmöglichkeiten gibt es beim Laden und Importieren. Bei grösseren Diagrammen kann es mehrere Sekunden dauern, bis alle Elemente geladen sind und in der 3D-Szene erscheinen. In dieser Zeit ist das GUI blockiert und der Benutzer sieht nur eine Meldung, dass der Ladevorgang gestartet wurde. Eine Fortschrittsanzeige wäre hier wünschenswert.

Aufwändiger wäre es, den Ladevorgang zu beschleunigen: Viel Zeit geht dadurch verloren, dass bei jeder Box – das heisst bei allen Klassen und Objekten – die FXML-Datei, welche das Top Face definiert, erneut von der Harddisk gelesen wird. Harddiskzugriffe sind zeitlich sehr teuer. Diese Zugriffe könnten eingespart werden, wenn man das Top Face Panel direkt in Code umsetzen würde. Dasselbe gilt für die Relationen: Die Modell-Dateien werden ebenfalls für Anfang und Ende der Relation jeweils erneut in ein Mesh-View-Objekt importiert. Besser wäre es, den Modellimport nur einmal pro Relationsendtyp beim Starten der Applikation durchzuführen und falls benötigt, das Mesh zu kopieren.

10.3. Weitere Funktionen

Hier folgt eine Auflistung von Features, die für den Benutzer nützlich wären oder in der 3DCOV-Vorgängerarbeit vorhanden sind:

- «Undo»-Funktion für Benutzeraktionen
- Raster, Linien, «Snap to Grid» und andere Platzierungshilfen für Klassen
- Überlagerung von Klassen verhindern
- Configfile für alle zur Zeit konstanten Werte
- Weitere Kameras. Zum Beispiel eine «First Person»-Kamera
- Assoziationsklassen
- UML-Assoziationsbedingungen (subsets und {or}-constraints)
- Assoziationsnamen mit Leserichtung
- Freie Platzierung von Multiplizitäten und Rollen
- Freie Platzierung von Relationsendpunkten am Rand der jeweiligen Klasse
- Einfügen von verschiebbaren Knotenpunkten bei Relationen
- Unterstützung von XMI 1.2 sowie von Enterprise-Architect-Objektdiagrammen beim Import
- XMI-Exportfunktion
- Editieren im Object Graph Mode

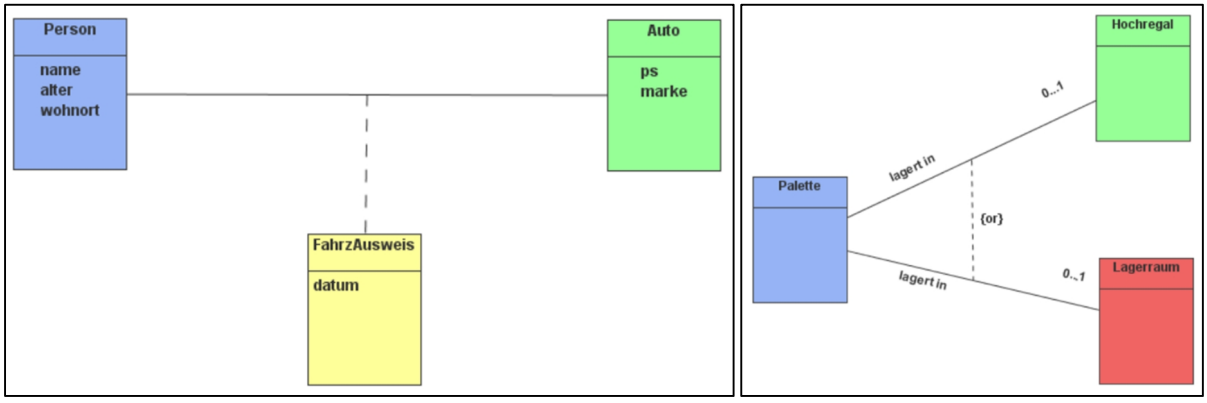


Abbildung 41: Beispiele aus dem 3DCOV: Eine Assoziationsklasse (links) und ein {or}-constraint (rechts). [11]

10.4. Optionale Erweiterungen

Die Aufgabenstellung erwähnt grosse Erweiterungen, die in der Umsetzung optional sind. Diese zusätzliche Funktionalitäten für den OGV sind vor allem deshalb interessant, weil sie eine weitere Benutzergruppe ansprechen: Software-Architekten und -Entwickler.

10.4.1. API für remotes Erstellen

Um die Modularität nicht zu verletzen, müsste das API in einem eigenen Service-Layer vorgelagert werden. Der Zugriff auf das Model kann mit Hilfe der ModelViewConnector Klasse erfolgen. Der Zugriff könnte über ein Socket mit einem eigenen definierten Protokoll erfolgen. So könnte von der aufzurufenden Applikation unabhängig eine OGV-Instanz laufen und das gewünschte Modell darstellen.

10.4.2. Live-Zustand abbilden

Dieser Punkt ist als Nachfolgeprojekt zum API gedacht. Eine Umsetzung könnte auf Reflection und den Java-Compiler zurückgreifen. Eine einfachere Möglichkeit bietet aspektorientierte Programmierung, zum Beispiel mittels AspectJ. Neben Applikationen, die in der JVM laufen, lassen sich natürlich auch Programme in anderen Sprachen abbilden. Vor allem Sprachen für die Verwendung mit dem .NET-Framework und interpretierte Programmiersprachen, wie etwa Python, kommen wegen einfacherer Introspektion in Frage.

Teil IV – Anhang

11. Quellenverzeichnis

11.1. JavaFX

[1] JavaFX 8 API, http://docs.oracle.com/javase/8/javafx/api/toc.htm, letzter Zugriff am 4.6.2015
[2] JavaFX 8 Tutorial, http://code.makery.ch/library/javafx-8-tutorial/, letzter Zugriff am 4.6.2015
[3] Getting Started with JavaFX, http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm, letzter Zugriff am 4.6.2015
[4] Building a 3D Sample Application, https://docs.oracle.com/javafx/8/3d_graphics/sampleapp.htm, letzter Zugriff am 4.6.2015
[5] Using JavaFX UI Controls, http://docs.oracle.com/javafx/2/ui_controls/jfxpub-ui_controls.htm, letzter Zugriff am 4.6.2015
[6] JavaFX Ensemble, http://download.oracle.com/otndocs/products/javafx/2/samples/Ensemble/index.html, letzter Zugriff am 4.6.2015
[7] JDK™ 8u40 Early Access Releases, https://jdk8.java.net/download.html, letzter Zugriff am 4.6.2015

11.2. JAXB

[8] JAXB unmarshal example, http://examples.javacodegeeks.com/core-java/xml/bind/jaxb-unmarshal-example/, letzter Zugriff am 4.6.2015
[9] Is there a way to configure rendering depth in JAXB?, http://stackoverflow.com/questions/2313962/is-there-a-way-to-configure-rendering-depth-in-jaxb, letzter Zugriff am 4.6.2015
[10] Using an XmlAdapter, http://www.eclipse.org/eclipselink/documentation/2.6/moxy/advanced_concepts006.htm, letzter Zugriff am 4.6.2015

11.3. 3D-Object-Class-Visualization

[11] Dario Vonäsch, "3D-Object-Class-Visualization Technischer Bericht", 2007

11.4. UML

[12] UML Metamodel http://www.utdallas.edu/~chung/OOAD/M05_Metamodel.ppt, letzter Zugriff am 4.6.2015
[13] Thomas Letsch, "Presentation Slides Object-Graph-Visualization", 16.12.2014
[14] OMG, "OMG Unified Modeling Language (OMG UML), Superstructure", Mai 2010

12. Glossar

3DCOV	3D-Class-Object-Visualization, Vorgängerprojekt- und Applikationsnamen
EE	Enterprise Architect, Softwaremodelierungswerkzeug von SparxSystems
FXML	XML-basierte Beschreibungssprache für JavaFX-UIs
JAXB	Java Architecture for XML Binding, API zur Datenbindung
MVC	Model View Controller, Entwurfsmuster für die Trennung von Datenmodell, Präsentation und Programmsteuerung
OGV	Object Graph Visualizer, Applikationsname dieser Bachelorarbeit
OMG	Object Management Group, Konsortium welches Standards für die OO-Programmierung entwickelt
SE	Abkürzung für «Software Engineering»
SW	Abkürzung für «Software»
UML	Unified Modeling Language, grafische Modellierungssprache von SW-Systemen
WYSIWYG	Abkürzung für «What You See Is What You Get»
XMI	XML Metadata Interchange, Austauschformat zwischen Software - Entwicklungs-werkzeugen
XML	Extensible Markup Language, Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten