Entwurf

ROSE (Road System Editor)
Edit Highway Roads Easily

Planung: Jannes Wagner
Entwurf: Yannik Sproll
Implementierung: Philipp Seidel
Qualitätssicherung: Cristian Gorun
Abnahme: Max Schweikart

Betreuung: Erik Burger, Jelle Kübler und Marvin Baumann

Institut für Verkehrswesen
Fakultät für Bauingenieur-, Geo- und Umweltwissenschaften
Karlsruher Institut für Technologie

Dezember 2021

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument enthält den Entwurf des Straßennetzeditors "ROSE", dessen Anforderungen im "ROSE"-Pflichtenheft festgehalten sind.

In diesem Dokument wird die Architektur von "ROSE" vorgestellt. Dabei wird auf die Strukturierung der Software eingegangen, sowie auf die Entwurfsentscheidungen die zu dieser Strukturierung geführt haben. Des Weiteren wird auf die Einarbeitung verwendeter Frameworks eingegangen.

Außerdem enthält dieses Dokument die entworfene Klassenstruktur. Um diese darzustellen finden sich hier die Deklarationen aller Klassen und öffentlichen Methoden, sowie die zugehörigen JavaDoc-Kommentare.

Die Assoziationen zwischen den Klassen, Interfaces und Enums werden in Form von UML-Klassendiagrammen dargestellt.

Die Paketstruktur wird in UML-Packetdiagrammen dargestellt.

Zur Beschreibung der Interaktion zwischen internen Komponenten eines Subsystems enthält dieses Dokument UML-Sequenzdiagramme. Komplexere Vorgänge werden zur Veranschaulichung anhand von Aktivitätsdiagrammen und Zustandsdiagrammen dargestellt.

2 Architektur

Dieser Abschnitt enthält eine Beschreibung der Architektur des ROSE-Entwurfs. Dabei wird auf die Verwendung von Model-View-Controller und einer Schichtenarchitektur eingegangen. Des Weiteren wird die Verwendung von Enwturfsmustern beschrieben.

2.1 Model-View-Controller

Die Struktur des Entwurfs von ROSE ist eine Umsetzung des Architekturmusters Model-View-Controller (kurz MVC). MVC wird für den Entwurf verwendet, da es eine klare Trennung der Darstellungslogik, der Ablaufsteuerung und der Domänenlogik bietet. In einem grafischen Editor wie ROSE ist die Darstellunglogik sehr komplex. Die Trennung der Darstellungslogik vom Rest des Programms (nach MVC) hilft die Komplexität der Darstellungslogik möglichst gering zu halten. Außerdem erlaubt MVC, dass das Model-Subsystem unabhängig vom Controller-Subsystem und dem View-Subsystem wiederverwendet werden kann. Abbildung 2.1 illustriert das Konzept der verwendeten MVC-Variante bzw. den Zusammenhang der drei MVC-Subsysteme.

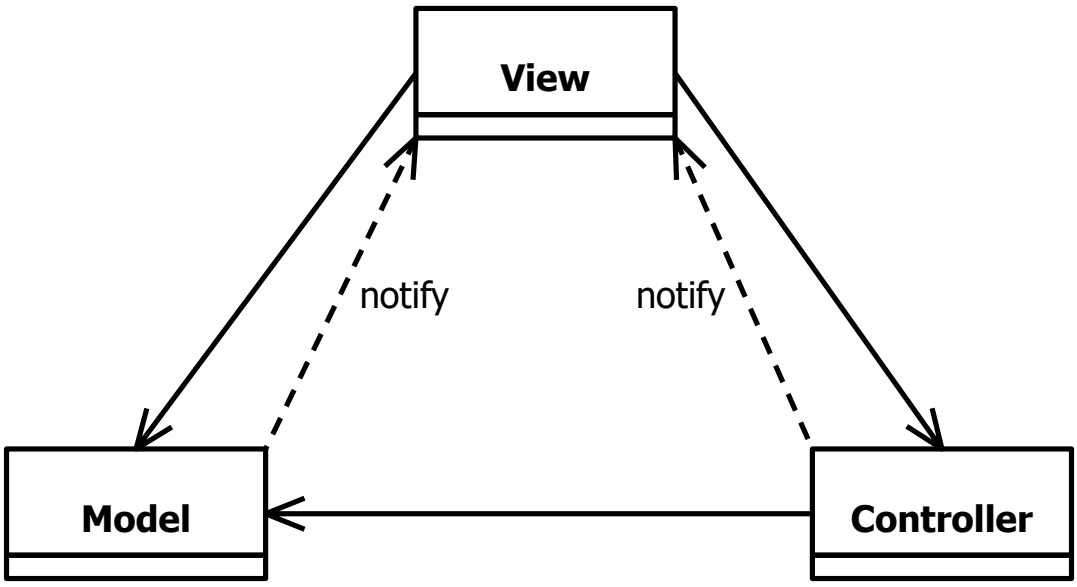

Abbildung 1: Exemplarisches Konzept für MVC im ROSE-Entwurf

Eine UI-Komponente aus dem View-Subsystem kennt sowohl die Controller, die sie verwendet, als auch das Model. Außerdem kennt jeder Controller das Model. Das Senden von Nachrichten der View an den Controller findet in Form von Methodenaufrufen statt.

Benachrichtigungen des Models oder des Controllers an die View werden mit Hilfe des Observer-Entwurfsmusters realisiert.Somit kommt das Observer-Entwurfsmuster als unterstützendes Architekturmuster zu MVC hinzu. An einigen Stellen wird eine Variante des Observer-Entwurfsmusters verwendet, welche eine anonyme Funktion anstatt des Observer-Ojektes beim Subject registriert. Außerdem wird an einigen Stellen ein Observer verwendet, der darauf spezialisiert ist Benachrichtigungnen über Änderungen an einem Set zu verbreiten. Das bringt den Vorteil, dass das Observer-Interface nicht zu groß wird, oder für jeden Event ein eigenes Observer-Interface existieren muss.

2.1.1 Model

Das Model-Subsystem enthält die Domänenlogik von ROSE, sowie das Datenmodell. Es befindet sich im Paket edu.kit.rose.model. Das Model unterscheidet die beiden großen Funktionalitätsbreiche Roadsystem und PlausibilitySystem, die jeweils beide in einem eigenen Unterpaket von edu.kit.rose.model sind.

In Roadsystem werden Segmente und deren Hierarchie verwaltet, sowie Attributwerte und Measurements gesetzt. Das Roadsystem wird intern durch einen Graph repräsentiert. Für diesen Graph wird das Framework JGraphT eingebunden. Die Verwendung eines Graphen, sowie die Verwendung von JGraphT wird von der Schnittstelle von Roadsystem nach außen hin verborgen.

Im PlausibilitySystem werden die Plausibilitätskriterien, sowie deren Validierung implementiert. Ein Plausibilitätskriterium tritt als Observer von Segmenten aus dem Roadsystem auf. Auf diese Weise werden Plausibilitätskriterien über Änderungen an Segmenten informiert und können sich selbst neu validieren.

Das Model bietet die Möglichkeit, dass andere Komponenten sich als Observer registrieren können, um über Änderungen von Werten im Model benachrichtigt zu werden. Außerdem bietet das Model anderen Programm-Komponenten eine vereinfachte Schnittstelle in Form der Fassaden-Klassen ApplicationDataSystem und Project an.

2.1.2 Controller

Die Controller in ROSE haben den Zweck Ablaufsteuerung von Operationen auf dem Model zu realisieren, Daten von der View an das Model weiterzuleiten und die Navigation zwischen den Fenstern zu steuern. Diese Navigation umfasst auch das Anzeigen von Dateisystem-Dialogen.

Um einen einzelnen Controller, und damit im Falle von ROSE eine Gottklasse, zu ver-

meiden, sind im Entwurf von ROSE mehrere Controller vorgesehen.

Die einzelnen Controller sind nach Funktionalität aufgeteilt, sodass jeder Controller einen Aspekt der Funktionalität der Anwendung abdeckt. So gibt es z.B. einen Controller, der die Verwaltung der Hierarchie übernimmt, und einen anderen Controller, der die Konfiguration der Plausibilitätskriterien übernimmt. Jeder dieser Controller wird in einem eigenen Unterpaket von edu.kit.rose.controller gehalten. Im Entwurf von ROSE sind Controller für die Verarbeitung von Nutzereingaben und Nutzerevents verantwortlich.

Außerdem sind sie für die Ablaufsteuerung von Operationen auf dem Model und für die Navigation zwischen den Fenstern von ROSE verantwortlich. Für diese Navigation definieren die Controller ein Navigator-Interface, jedoch keine eigene Klasse. Ein Klient eines Controllers, wie z.B. eine Klasse der View, kann eine eigene Klasse dieses Navigator-Interface implementieren lassen. Diese Realisierung der Navigation entspricht dem Dependency-Inversion-Principle. Auf diese Weise findet Navigation in den Controllern statt, ohne dass diese die UI-Komponenten kennen müssen.

Auch die Change-Funktionalität wird hier implementiert. Sie wir mit dem Command-Entwurfsmuster umgesetzt. Im Paket edu.kit.rose.controller.command wird das Interface ChangeCommand implementiert. Es gibt die Methoden zum Ausführen und Rückgängig machen eines ChangeCommands vor. Kann eine Aktion eines Controllers rückgängig gemacht werden, existiert im Unterpaket des Controllers ein Command-Objekt, das diese Aktion kapselt. Diese Command-Objekt implementiert ChangeCommand. Die erstellten Command-Objekte werden in der Komponente ChangeCommandBuffer im Paket edu.kit.rose.controller.command gespeichert. Die ChangeCommandBuffer-Komponente bietet die Methoden an die zum Rückgängigmachen und Wiederherstellen aufgerufen werden. Die ChangeCommandBuffer-Komponente koordiniert dann die Methodenaufrufe mit den Commands die sie enthält.

Außerdem bieten die Controller anderen Programm-Komponenten die Möglichkeit sich als Observer zu registrieren, und bei wichtigen Events, die während den Abläufen eintreten, benachrichtigt zu werden. Insbesondere das Senden von Benachrichtigungen der Controller an Komponenten aus dem View-Subsystem wird auf diese Weise realisiert.

Um eine korrekte Instanziierung und Initialisierung einer Menge von Controllern zu gewährleisten enthält das Paket edu.kit.rose.controller die nach dem Factory-Muster realisierte Klasse ControllerFactory. Das Paket edu.kit.rose.controller enthält die Klasse ControllerFactory. Sie stellt sicher, dass alle Controller korrekt erstellt werden.

2.1.3 View

Die View ist für die visuelle Darstellung aller Inhalte verantwortlich. Sie befindet sich im Paket edu.kit.rose.view.

Die View holt sich die dafür benötigten Daten direkt vom Model. Im Allgemeinen ist der Zugriff der View auf das Model in ROSE ausschließlich lesend. Die View registiert sich beim Model als Observer und lädt, bei Benachrichtigungen über Änderungen im Model, die aktualisierten Werte selbst.

Eine Komponente der View kennt immer genau die Controller, deren Funktionalität sie benötigt. Nutzereingaben und Nutzerevents werden an diese Controller weitergeleitet und von diesen verarbeitet. Kleinere Validierungen werden bereits in der View erledigt.

Alle Fenster in ROSE liegen im Paket edu.kit.rose.view.window. Die Panels die in diesen Fenstern eingebettet sind, liegen jeweils in einem eigenen Unterpaket von edu.kit.rose.view.panels.

Basisklassen für Fenster, gemeinsam genutzte UI-Komponenten befinden sich im Paket edu.kit.rose.view.commons. Auf diese Weise werden zirkuläre Abhängigkeiten zwischen dem Paket edu.kit.rose.view und seinen Unterpaketen vermieden.

Zur Realisierung der View wird in ROSE JavaFx verwendet. Tatsächlich beschränkt sich der Einsatz von JavaFx auf das Paket edu.kit.rose.view. So wird sichergestellt, dass Model und Controller keine Abhängigkeiten auf die JavaFx-Bibliothekt haben. Das soll die View austauschbarer machen.

JavaFx bringt ein eigenes Konzept der Verbindung von Controller und View mit. Dieses Konzept wird in ROSE nicht zur Realisierung von MVC verwendet. Das hat den Grund, dass nach dem Konzept von JavaFx ein Controller einzelne Steuerelemente der zugehörigen View-Komponente kennen muss.

Die Instanziierung und Initialiserung der Komponenten aus den Subsystemen Model, View und Controller, sowie die Konstruktion der Anwendung aus diesen Komponenten, über nimmt die Klassen edu.kit.rose.view.RoseApplication.

2.2 Schichtenarchitektur

Im Entwurf von ROSE werden die einzelnen MVC-Subsysteme den Schichten einer Schichtenarchitektur zugeordnet. Konkret bedeutet das, dass die Pakete edu.kit.rose.model, edu.kit.rose.controller und edu.kit.rose.view verschiedenen Schichten zugeordnet werden.

Diese Zuordnung hat den Zweck zyklische Abhängigkeiten der MVC-Komponenten zu verhindern und. Die in ROSE eingebaute Schichtenarchitektur soll eine transparente 4-Schichtenarchitektur sein. Eine transparente Schichtenarchitektur ist hier sinnvoll wegen der nicht notwendigen Datenwandlung in den Controllern. Diese hätte folgende Nachteile:

- Die Datenwandlung in den Controllern, sowie der Umweg über einen Controller beim Abruf von Daten des Models durch die View, kosten Performance und Speicherplatz.

- Die zusätzliche Datenwandlung einer intransparenten Schichtenarchitektur erhöht die Komplexität der Controllern.

Abbildung 2.2 zeigt die Zuordnung der Pakete zu den Schichten.

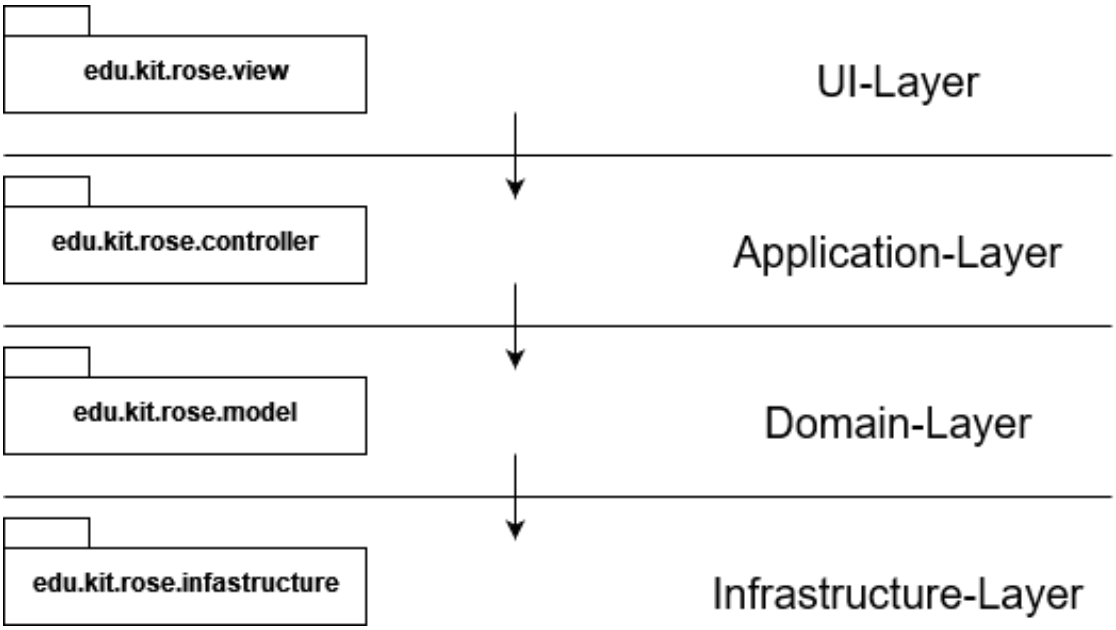

Abbildung 2: Zuordnung der MVC-Subsysteme zu ihren Schichten

Die Abhängigkeiten zwischen den MVC-Komponenten gehen ausschließlich in Richtung der Pfeile. Wegen der Transparenz der Schichtenarchitektur können Abhängigkeiten zwar von Komponenten auch unterliegende Schichten überspringen, die Pfeilrichtung muss dabei beachtet werden.

Das Paket edu.kit.rose.view befindet sich im UI-Layer. Hier befindet sich die Darstellungslogik von ROSE.

Das Paket edu.kit.rose.controller befindet sich im Application-Layer.

Das Paket edu.kit.rose.model enthält das Domänenmodell von ROSE. Wie bereits oben beschrieben liegt hier die Domänenlogik der Anwendungsdomäne Straßenverkehr.

Das Paket edu.kit.rose.infrastructure befindet sich im Infrastructure-Layer. Hier wird Anwendungslogik plaziert, die nicht zur Domänenlogik gehört, bzw. von dieser unabhängig ist. Der Platz dieser Anwendungslogik wird auch im MVC-Architekturmuster nicht spezifiziert. Ein Beispiel für solche Logik ist die Unterstützung für Mehrsprachigkeit in ROSE. Diese wird im Paket edu.kit.rose.infrastructure.language implementiert.

2.3 Paketaufteilung

In diesem Abschnitt werden die Unterpakete der Pakete für MVC-Subsysteme, sowie des Pakets edu.kit.rose.infrastructure anhand von UML-Paketdiagrammen dargestellt.

2.3.1 Model-Subsystem

Abbildung 2.3.1 zeigt die Paketaufteilung des Model-Subsystems.

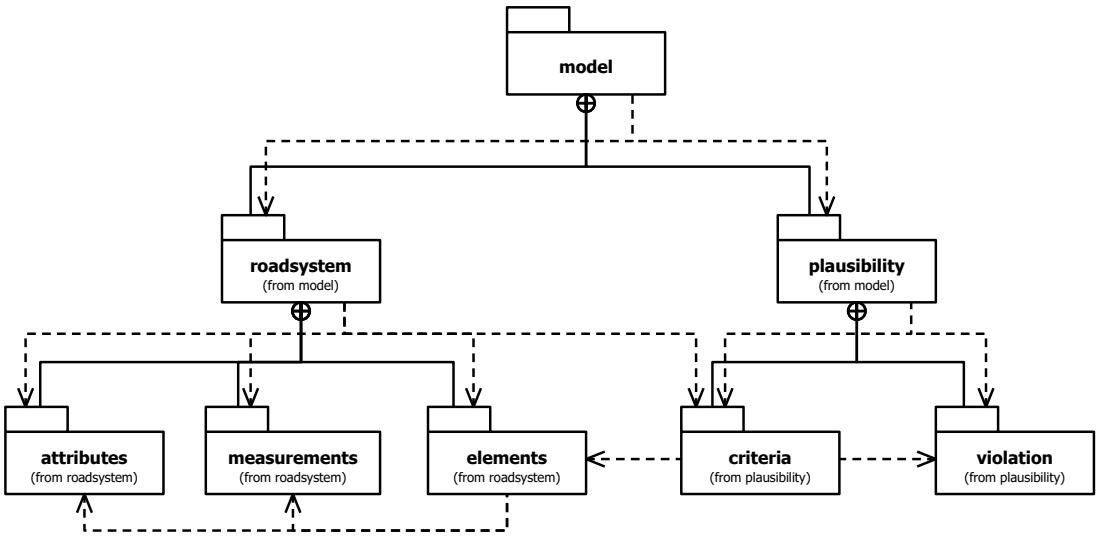

Abbildung 3: Paketstruktur des Pakets edu.kit.rose.model

2.3.2 Controller-Subsystem

Abbildung 2.3.3 zeigt die Paketaufteilung des Controller-Subsystems.

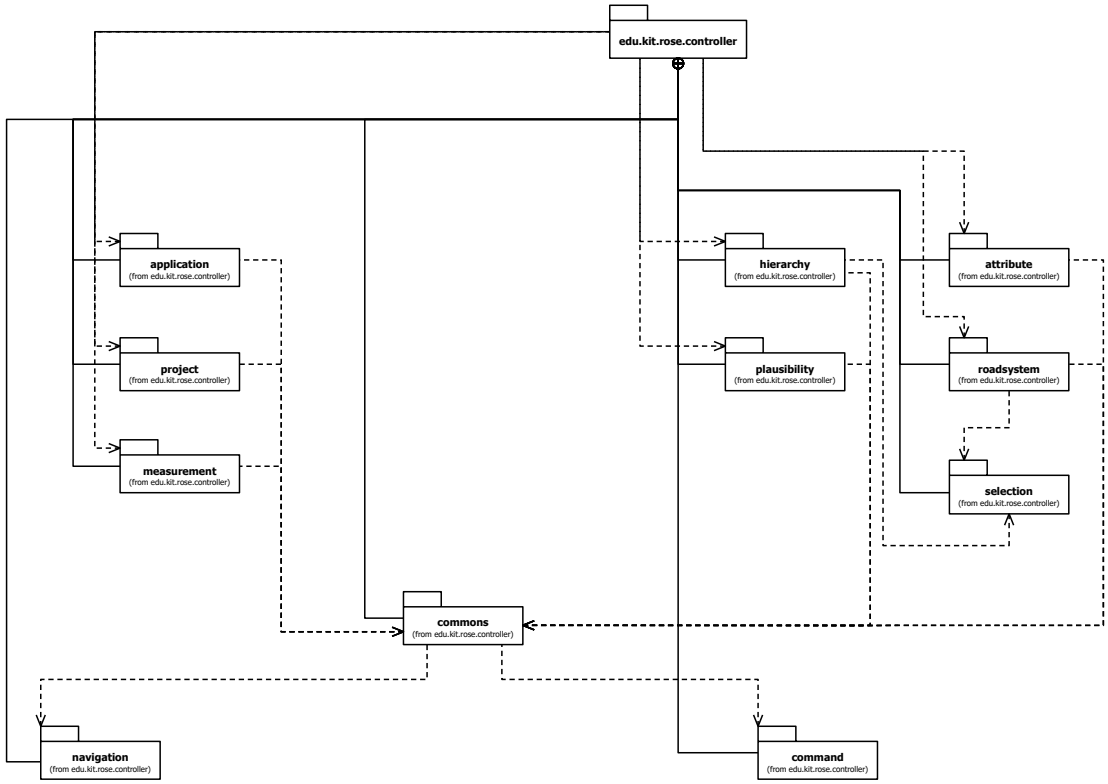

Abbildung 4: Paketstruktur des Pakets edu.kit.rose.controller

2.3.3 View-Subsystem

Abbildung 2.3.3 zeigt die Paketaufteilung des View-Subsystems.

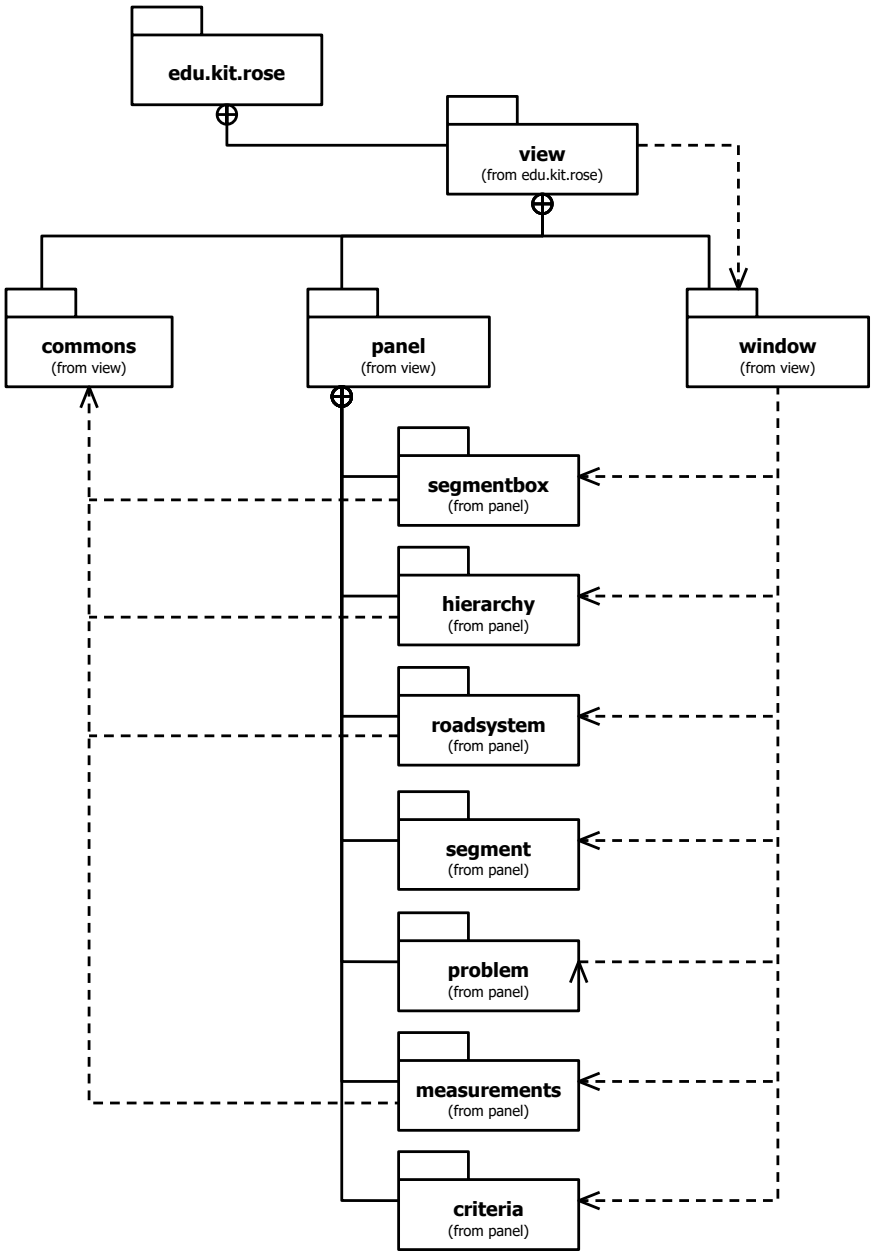

Abbildung 5: Paketstruktur des Pakets edu.kit.rose.view

2.3.4 Infrastructure

Abbildung 2.3.4 zeigt die Paketaufteilung des Pakets edu.kit.rose.infrastructure.

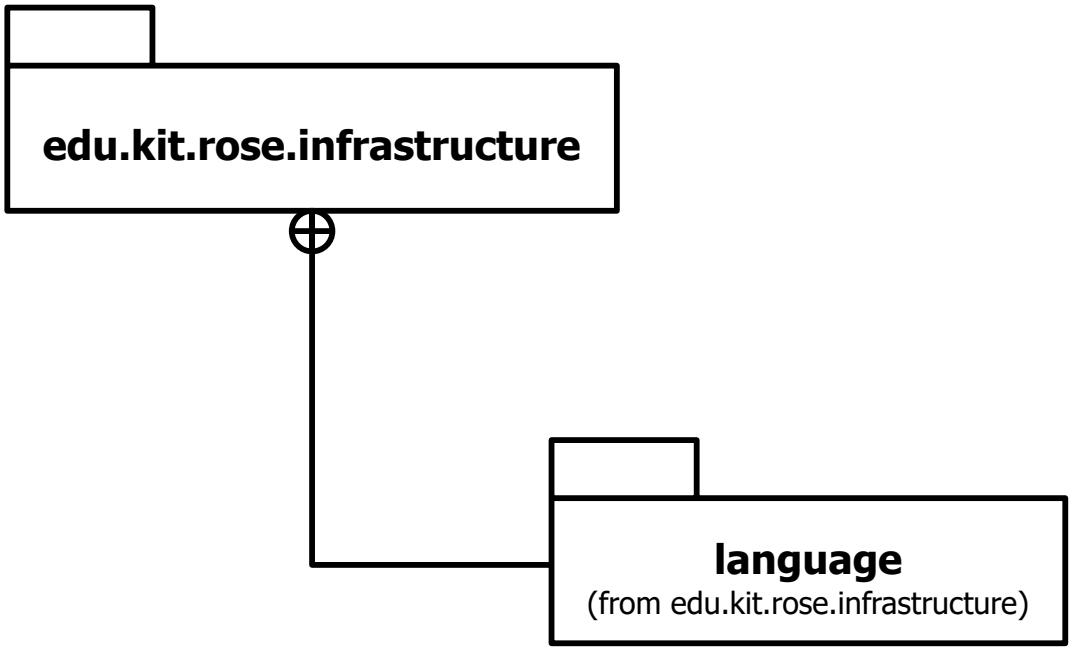

Abbildung 6: Paketstruktur des Pakets edu.kit.rose.infrastructure

3 Objektmodell

In diesem Kapitel befinden sich die Definitionen des gesamten Objektmodells. Für jedes Paket gibt es einen Abschnitt der ein detailliertes UML-Klassendiagramm, sowie die Definitionen aller Klassen, Interfaces und Enums des zum Abschnitt gehörenden Pakets enthält.

Das Objektmodell folgt an den meisten Stellen dem UML-Standard. An manchen Stellen wurden jedoch Methoden- und Konstruktorparameter verkürzt angegeben, um eine bessere Lesbarkeit der Diagramme zu gewährleisten. In jedem Fall sind die Parameter jedoch in der entsprechenden Klassenbeschreibung vollständig aufgeführt. Zudem ist in den Klassendiagrammen immer das Paket, das im jeweiligen Abschnitt beschrieben wird, rot hervorgehoben. Klassen, Interfaces und Enums außerhalb des rot hervorgehobenen Bereiches sind in der Regel verkürzt dargestellt und werden im Abschnitt zum zugehörigen Paket vollständig beschrieben.

3.1 Package edu.kit.rose.model

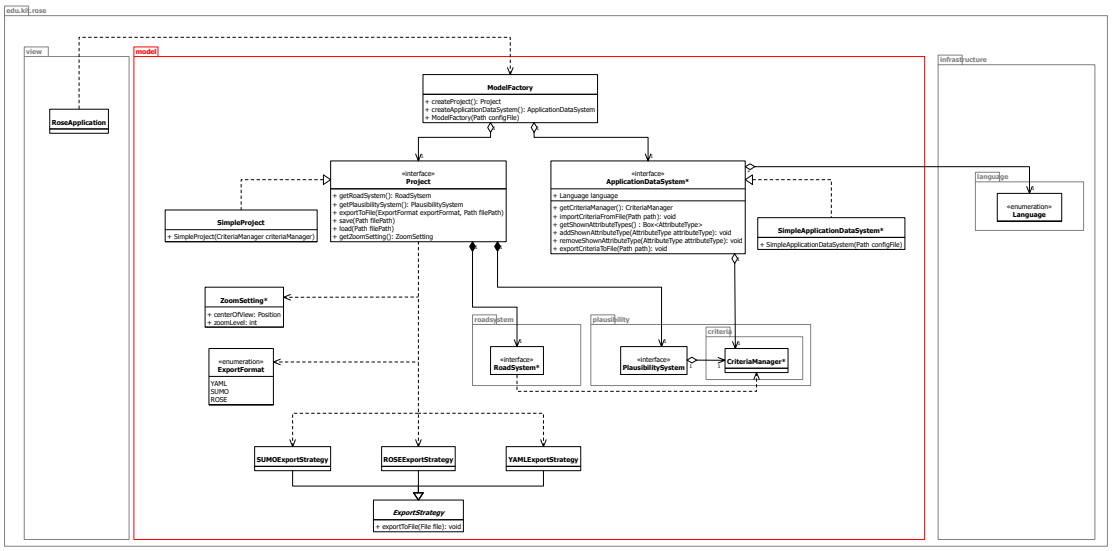

Abbildung 7: Class diagram for package edu.kit.rose.model

Contains the model logic of the Program. It provides a RoadSystem as well as a ApplicationDataSystem.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.1.1 Interface ApplicationDataSystem

ApplicationData is all Data that has to be saved independently of the project. It includes the current language setting as well as the CriteriaManager as these are to be configured on a project independent level. Is to be observed by the view. Observes a Criteriamanager in order to be informed of changes to the Criteria held within, as the ApplicationDataSystem needs to write these changes to the config file.

Declaration

- `public interface ApplicationDataSystem extends UnitObservable<ApplicationDataSystem >, SetObserver<PlausibilityCriterion, CriteriaManager>`

Methods

- `Language getLanguage()`
  Returns the currently selected Language.
- `void setLanguage(Language language)`
  Sets a Language. Requires the Language that is to be set.
- `CriteriaManager getCriteriaManager()`
  Returns the CriteriaManager that is held by this ApplicationDataSystem.
- `void importCriteriaFromFile(Path path)`
  Imports all PlausibilityCriteria from the File at the given Path. This adds the included Criteria to the currently active Criteria.
  Requires the path to the File that contains the Criteria.
- `void exportCriteriaToFile(Path path)`
  Exports all PlausibilityCriteria into a File at the given Path.
  Requires the path giving the location of where to save the new File.

3.1.2 Enum ExportFormat

An enum holding the different formats to which a Project can be exported.

Declaration

- `public enum ExportFormat`

Values

- `YAML`
- `SUMO`
- `ROSE`

Attributes

- none

Constructors

- none

Methods

- none

3.1.3 Class ExportStrategy

Describes a Strategy for exporting a Project. Implements the Strategy in the Strategy design Pattern with the Project as the Client.

Declaration

- `abstract class ExportStrategy`

Attributes

- none

Constructors

- none

Methods

- `void exportToFile(File file)`
  Exports the current Project to the given File.
  Requires the location to export the Project to.

3.1.4 Class ImportStrategy

Describes a Strategy for importing a Project.
Implements the Strategy in the Strategy design Pattern.

Declaration

- `abstract class ImportStrategy`

Attributes

- none

Constructors

- none

Methods

- `abstract void importFromFile (File file, Project project)`
  Imports the Project from the given File and replaces the given Project with it.
  Requires the location of the Project File to import.
  Requires the Project to replace with the one from the File.

3.1.5 Class ModelFactory

Constructs both a Project and an ApplicationDataSystem.

Declaration

- `public class ModelFactory`

Attributes

- none

Constructors

- `public ModelFactory(Path configFile)`
  Constructs both the ApplicationDataSystem and the Project held within.
  Requires the config file that is to be used to configure the ApplicationDataSystem.

Methods

- `public Project createProject()`
  Returns the constructed Project returns the constructed Project
- `public ApplicationDataSystem createApplicationDataSystem()`
  Returns the constructed ApplicationDataSystem

3.1.6 Interface Project

A Project holds all data specified in "Pflichtenheft: Projekt" as well as the current Zoom-Setting and the current TimeSliceSetting.

Declaration

- `public interface Project`

Methods

- `void exportToFile (ExportFormat exportFormat, Path filePath)`
  Formats the Project to fit a specified ExportFormat and exports it to a file at the given Path
  Requires the ExportFormat to save in.
  Requires the Path of where to store the export.
- `void save (Path filePath)`
  Saves the Project as a ROSE file. This saves everything the Project needs to be

reopened in the program. Including PlausibilityCriteria and Positions.
Requires the Path of where to store the file.

- `void load (Path filePath)`
  Loads a ROSE File. This rewrites the Project to hold the information specified in the provided file.
  Requires the Path of the File.
- `ZoomSetting getZoomSetting()`
  returns the ZoomSetting of a view that displays the RoadSystem.
- `RoadSystem getRoadSystem()`
  Returns the road system of this project.
- `PlausibilitySystem getPlausibilitySystem()`
  Returns the plausibility system of this project.

3.1.7 Class ROSEExportStrategy

Implements the export for the whole Project into the ROSE-Format.
This Format allows the Project to be reopened in the ROSE-Program.

Declaration

- `class ROSEExportstrategy extends ExportStrategy`

Attributes

- none

Constructors

- none

Methods

- none

3.1.8 Class ROSEImportstrategy

Implements the import for the whole Project from the ROSE-Format.This replaces the current Project with the one from the File.

Declaration

- `class ROSEImportstrategy extends ImportStrategy`

Attributes

- none

Constructors

- none

Methods

- none

3.1.9 Class SimpleApplicationDataSystem

A standard implementation for the ApplicationDataSystem.
Provided with a global config file it will write changes in the applicationData to the config file in real time.

Declaration

- `class SimpleApplicationDataSystem extends SimpleUnitObservable<ApplicationDataSystem > implements ApplicationDataSystem`

Attributes

- none

Constructors

- `public SimpleApplicationDataSystem(Path configFile)`
  Needs to be provided with a Path to a config File for global Settings.
  Requires the Path to a config File containing global Settings.

Methods

- none

3.1.10 Class SimpleProject

A standard implementation for Project.

Declaration

- `class SimpleProject implements Project`

Attributes

- none

Constructors

- `public SimpleProject (CriteriaManager criteriaManager)`
  Requires a CriteriaManager to instantiate a PlausibilitySystem.

Methods

- none

3.1.11 Class SUMOExportStrategy

Implements the export for the RoadSystem into the SUMO-Format.
This format allows the Project to be reopened by a Program supporting the SUMO-Format.

Declaration

- `class SUMOExportstrategy extends ExportStrategy`

Attributes

- none

Constructors

- none

Methods

- none

3.1.12 Class YAMLExportStrategy

Implements the export for the RoadSystem into the YAML-Format.
This format allows the Project to be reopened by a Program supporting the YAML-Format.

Declaration

- `class YAMLExportstrategy extends ExportStrategy`

Attributes

- none

Constructors

- none

Methods

- none

3.1.13 Class ZoomSetting

A ZoomSetting describes the Position and "level of Zoom" a possible View of the Road-System has.
It uses an int to describe the amount of Zoom a view has and a Position for the center of the current view. Is to be observed by the view.

Declaration

- `public class ZoomSetting extends SimpleUnitObservable<ZoomSetting>`

Attributes

- none

Constructors

- none

Methods

- `Position getCenterOfView()`
  Returns the Position of the center of the View.
- `void setCenterOfView(Position position)`
  Requires the new Position of the view.
- `int getZoomLevel()`
  Returns the level of Zoom the View has.
- `void setZoomLevel(int zoomLevel)`
  Requires the new level of zoom of the view.

3.2 Package edu.kit.rose.model.roadsystem

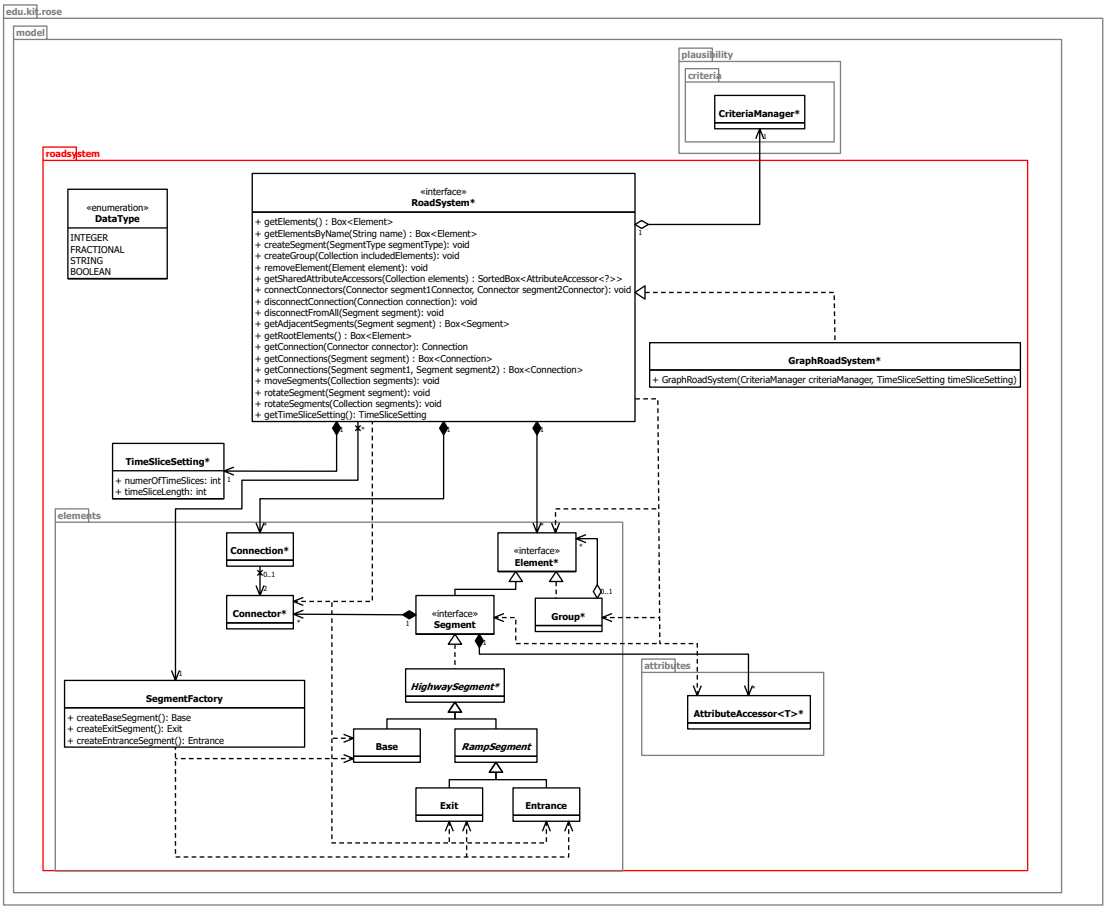

Abbildung 8: Class diagram for package edu.kit.rose.model.roadsystem

This package contains everything needed for a Roadsystem. This includes both Groups and Segments as well as their Attributes, Measurements and Connections.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.2.1 Enum DataType

An enum holding different Datatypes.
Used to determine how an attribute will be displayed in the view.

Declaration

- `public enum DataType`

Values

- `INTEGER`
- `FRACTIONAL`
- `STRING`
- `BOOLEAN`

Attributes

- none

Constructors

- none

Methods

- none

3.2.2 Class GraphRoadSystem

A standard implementation of a RoadSystem using a Graph for holding the Connections between Segments.

Declaration

- `class GraphRoadSystem extends SimpleDualSetObservable<Element, Connection , RoadSystem> implements RoadSystem`

Attributes

- none

Constructors

- `public GraphRoadSystem(CriteriaManager criteriaManager, TimeSliceSetting timeSliceSetting)`
  Constructor.

The CriteriaManager is needed for the subscription of PlausibilityCriterion to Segments. The TimeSliceSetting is for the configuration of Measurements.
Requires the applicationDataSystem to use for this GraphRoadSystem.
Requires the time slice setting to use for this GraphRoadSystem.

Methods

- `public void rotateSegment(Segment segment)`
  Rotates the given Segment on its current Position by 15 degrees. Uses the center of the segment as a center of rotation.
  Requires the Segment to rotate.
- `public void rotateSegments(Collection<Segment> segments)`
  Rotates the given Segments on their current Position by 15 degrees. Uses the average center of the segments as a center of rotation.
  Requires the Segments to rotate.

3.2.3 Interface RoadSystem

A RoadSystem models a set of Elements that can be Segments or Groups of Segments. It also manages Connections between these Segments and allows the creation of new Connections as well as their removal. It provides methods to create and remove Segments. In addition to this it allows for easy access to all information held within. Is to be observed by the view, providing information about both Segments and Connections being added or removed.

Declaration

- `public interface RoadSystem extends DualSetObservable<Element, Connection , RoadSystem>, UnitObserver<Connector>`

Methods

- `Box<Element> getElements()`
  Returns a Box containing all Elements of the RoadSystem.
- `Box<Element> getElementsByName(String name)`
  Returns a Box of all Elements with a name similar to the given String. What will count as similar is left open to the implementation.
  Requires the name of the Elements to return.
  Returns a Box of all Elements with the given name.
- `void createSegment(SegmentType segmentType)`
  Creates a Segment with the given SegmentType.
  Requires the SegmentType the Segment shall have.
- `void createGroup(Collection<Element> includedElements)`

Creates a Group containing the given Elements
Requires the Elements that shall be in the new Group

- `void removeElement(Element element)`
  Removes an Element from the RoadSystem. If the given Element does not exist nothing happens.
  Requires the Element to remove.
- `SortedBox<AttributeAccessor<?>> getSharedAttributeAccessors(Collection <Element> elements)`
  Returns a SortedBox of all shared AttributeAccessors of the given Elements. The AttributeAccessors will redirect method calls to the respective AttributeAccessors of all elements of the given collection.
  Requires the Collection of Elements to get the AttributeAccessors for.
  Returns a SortedBox containing all shared AttributeAccessors of the Elements.
- `void connectConnectors(Connector segment1Connector, Connector segment2Connector )`
  Creates a new Connection between the two given Connectors. Disconnects previous Connections including these Connectors.
  Requires both Connectors.
- `void disconnectConnection(Connection connection)`
  Disconnects a given Connection. This means the Connectors are no longer connected (same for their respective Segments).
  Requires the Connection to disconnect.
- `void disconnectFromAll(Segment segment)`
  Disconnect all Connections the given Segment has. This will disconnect it from all Segments it was previously connected to.
  Requires the Segment to disconnect.
- `Box<Segment> getAdjacentSegments(Segment segment)`
  Returns a Box of the Segments a given Segment is currently connected to.
  Requires the Segment of which the connected Segments are to be returned.
- `Box<Element> getRootElements()`
  Returns a Box of Elements that are the uppermost in the element hierarchy.
- `Connection getConnection(Connector connector)`
  Returns the Connection a given Connector is involved in.
  Requires the Connector to look for the Connection.
- `Box<Connection> getConnections(Segment segment)`
  Returns all Connections the given Segment is currently involved in. (This is th case if at least one of the Segments Connectors is part of a Connection).
  Requires the segments of which the Connections are to be returned.
  Returns a Box of all Connections of the Segment.
- `Box<Connection> getConnections(Segment segment1, Segment segment2)`
  Returns all Connections the two given Segments have together.
  Requires the two Segments
  Returns a Box of all Connections the two given Segments have.
- `void moveSegments(Collection<Segment> segments, Movement movement)`

Moves all the given Segments from their current Position to a new one. Uses the Movement and applies it to the old Position.
Does not disconnect the Segments.
Requires the Segments to move.
Requires the Movement describing how to move the Segments.

- `void rotateSegment(Segment segment)`
  Rotates the given Segment on its current Position.
  Requires the Segment to rotate.
- `void rotateSegments(Collection<Segment> segments)`
  Rotates the given Segments on their current Position.
  Requires the Segments to rotate.
- `TimeSliceSetting getTimeSliceSetting()`
  Provides the TimeSliceSetting this RoadSystem uses.
  Returns the used TimeSliceSetting.

3.2.4 Class TimeSliceSetting

A TimeSliceSetting describes the properties of the time slices that Measurements use. It describes how many time slices there are and how long they are (in Minutes). Is to be observed by Measurements of Segments.

Declaration

- `public class TimeSliceSetting extends SimpleUnitObservable<TimeSliceSetting>`

Attributes

- none

Constructors

- none

Methods

- `int getNumberOfTimeSliceSteps()`
  Returns the number of time slices.
- `void setNumberOfTimeSliceSteps(int numberOfTimeSliceSteps)`
  Requires the new number of time slices.
- `int getTimeSliceLength()`
  Returns the length of the time slices.
- `void setTimeSliceLength(int length)`
  Requires the new length of the time slices.

3.3 Package edu.kit.rose.model.roadsystem.attributes

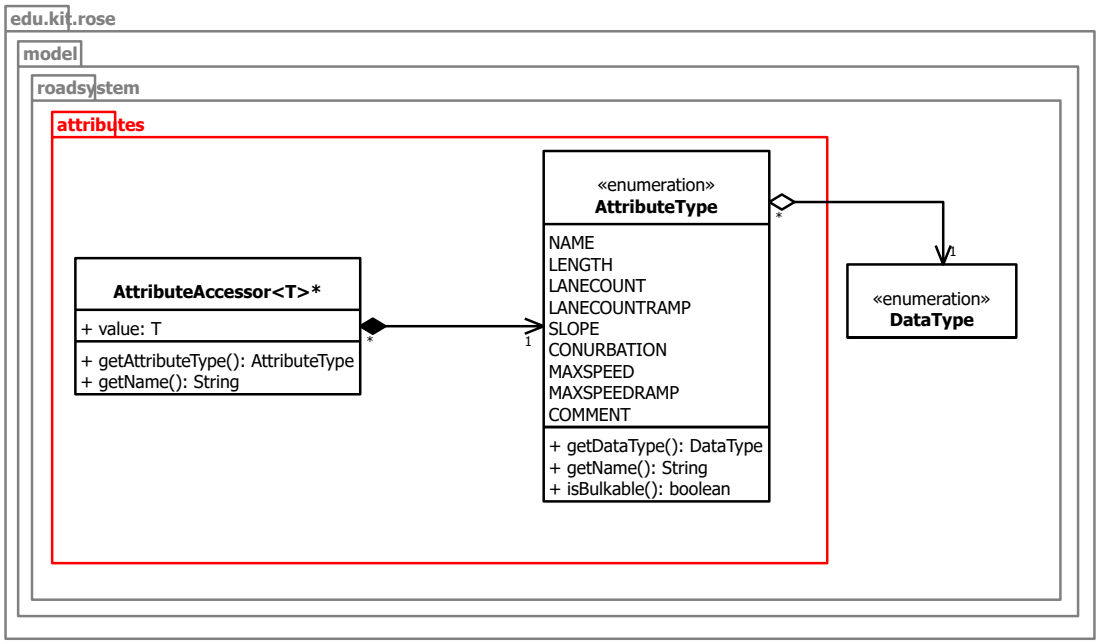

Abbildung 9: Class diagram for package edu.kit.rose.model.roadsystem.attributes

This package contains AttributeAccessors as well as an enum describing the different Types of attributes a Segment might have.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.3.1 Class AttributeAccessor

An accessor for an attribute. Has an AttributeType. Allows access to an attribute of an object without having to consult the object itself. This makes it possible to grant dynamic access to attributes of an object.
<T> the Class of the Attribute that this accessor is used for. Is to be observed by the view.

Declaration

- `public class AttributeAccessor<T> extends SimpleUnitObservable<AttributeAccessor<T>>`

Attributes

- none

Constructors

- none

Methods

- `AttributeType getAttributeType()`
  Returns the AttributeType of the underlying attribute.
- `String getName()`
  Returns the name of the underlying attribute.
- `public void setValue(T value)`
  Requires the value to set the attribute to.
- `public T getValue()`
  Returns the value of the attribute.

3.3.2 Enum AttributeType

Different types of AttributeAccessors. Used to determine the UI needed to configure an attribute. Provides information about whether an attribute should be bulkable, that is if it should be set to the same value as other attributes of the same type.

Declaration

- `public enum AttributeType`

Values

- `NAME`
- `LENGTH`
- `LANECOUNT`
- `LANECOUNT_RAMP`
- `DIRECTION`

- `SLOPE`
- `CONURBATION`
- `MAXSPEED`
- `MAXSPEEDRAMP`
- `COMMENT`

Attributes

- none

Constructors

- none

Methods

- `DataType getDataType()`
  Returns the DataType of the AttributeType.
- `String getName()`
  Returns a String holding the name of the AttributeType.
- `boolean isBulkable()`
  Returns a boolean describing if the AttributeType can be accessed for multiple Elements at once.

3.4 Package edu.kit.rose.model.roadsystem.elements

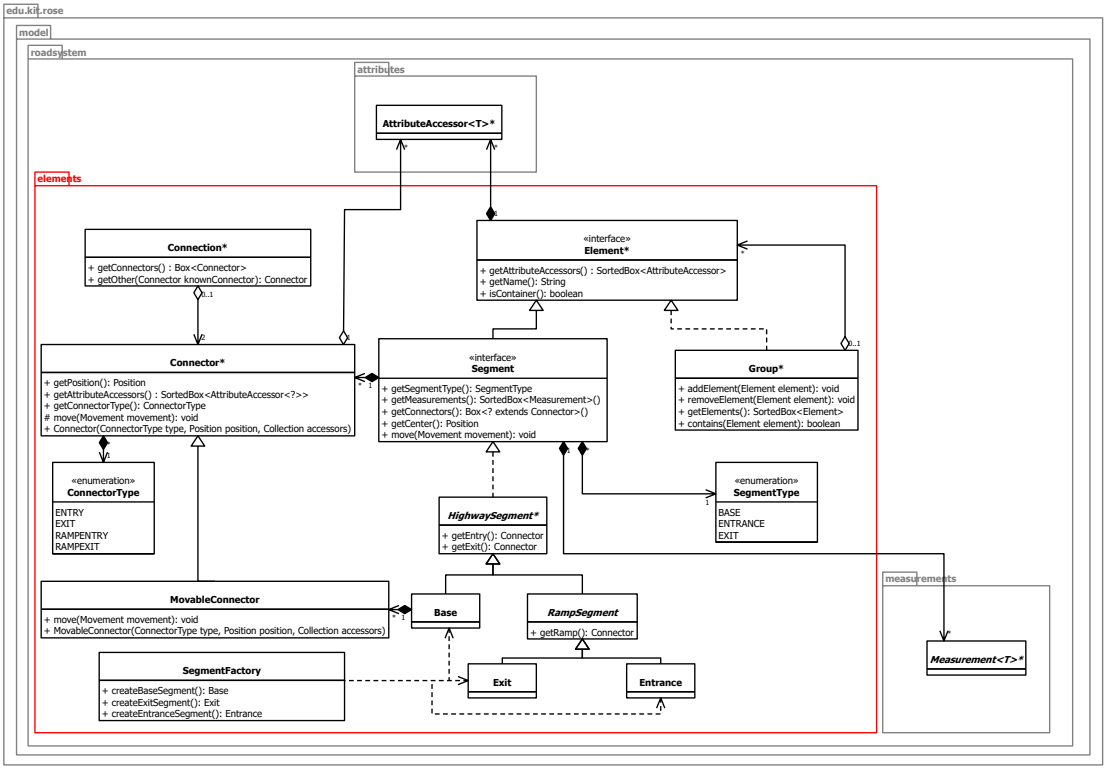

Abbildung 10: Class diagram for package edu.kit.rose.model.roadsystem.elements

This package contains all types of elements. E.g. Groups and different types of Segments as well as Connections.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.4.1 Class Base

Represents a one way road (as in part of a freeway).
A Base segment is a Segment that only has one entrance and one exit.

Declaration

- `class Base extends HighwaySegment`

Attributes

- none

Constructors

- none

Methods

- none

3.4.2 Class Connection

A connection between two Connectors. Is to be observed by the view.

Declaration

- `public class Connection extends SimpleUnitObservable<Connection>`

Attributes

- none

Constructors

- none

Methods

- `public Box<Connector> getConnectors()`
  Returns the Connectors that are connected by this Connection.
- `public Connector getOther(Connector knownConnector)`
  Provides the other Connector held within respectively.
  Requires the known Connector.
  Returns the other Connector.

3.4.3 Class Connector

A Connector is part of a Segment and represents the end points of the same (see Pflichtenheft: Straßensegment).
A Connector can also be part of a Connection. Is to be observed by the view as well as by a RoadSystem.

Declaration

- `public class Connector extends SimpleUnitObservable<Connector>`

Attributes

- none

Constructors

- `Connector(ConnectorType type, Position position, Collection<AttributeAccessor <?>> accessors)`
  Requires the ConnectorType for this Connector.
  Requires the Position that this Connector is supposed to be at.
  Requires the AttributeAccessors that this Connector is supposed to have.

Methods

- `public Position getPosition()`
  Returns the Position of the connector.
- `public SortedBox<AttributeAccessor<?>> getAttributeAccessors()`
  Gives the AttributeAccessors to the Attributes that are specific for this Connector. The referenced Attributes are part of the Segment this Connector is part of, though only the accessors for the Connector specific attributes will get returned. I.e. the lane count accessor returned by this method will give access to the lane count attribute of the end point of the segment represented by this connector.
  Returns a SortedBox containing the specific AttributeAccessors of this Connector.
- `public ConnectorType getConnectorType()`
  Returns the type of Connector this is.

3.4.4 Enum ConnectorType

An enum describing the different kinds of Connectors. Used to differentiate between Connectors of a Segment.

Declaration

- `public enum ConnectorType`

Values

- `ENTRY`
- `EXIT`

- `RAMPENTRY`
- `RAMPEXIT`

Attributes

- none

Constructors

- none

Methods

- none

3.4.5 Interface Element

Represents an Element (see Pflichtenheft: "Element"). Holds a set of AttributeAccessors. Implements the Composite in the Composition Pattern with Segments as leafs and Groups as containers. Is to be observed by the view.

Declaration

- `public interface Element extends UnitObservable<Element>`

Methods

- `SortedBox<AttributeAccessor<?>> getAttributeAccessors()`
  Returns AttributeAccessors that thus allow for access to attributes of this element.
  Returns the AttributeAccessors for attributes of this Element.
- `String getName()`
  Returns the name of this Element.
- `boolean isContainer()`
  Returns a boolean describing if this is a Group (contains other Elements) or a Segment
  Returns true if this Element can contain other Elements. False if it can not.

3.4.6 Class Entrance

Represents a freeway entrance.
An Entrance segment is a Segment that provides one entrance from the main road and

one exit from it as well as a ramp by which cars can enter the Road.

Declaration

- `class Entrance implements RampSegment`

Attributes

- none

Constructors

- none

Methods

- none

3.4.7 Class Exit

Represents a freeway exit.
An Exit Segment is a Segment that has one entrance to the main road and one exit from it as well as a singular ramp by which cars can leave the Road.

Declaration

- `class Exit implements RampSegment`

Attributes

- none

Constructors

- none

Methods

- none

3.4.8 Class Group

A container that holds several Elements. Might hold other groups. (see: Pflichtenheft: “Gruppe”)

Declaration

- `public class Group extends SimpleUnitObservable<Element> implements Element , Iterable<Element>`

Attributes

- none

Constructors

- none

Methods

- `public void addElement(Element element)`
  Adds a given Element to the Group. If the given Element is already in the Group nothing happens.
  Requires the Element that shall be added to the Group.
- `public void removeElement(Element element)`
  Removes a given Element from the Group. If the given Element does not exist nothing happens.
  Requires the Element that shall be removed from the Group.
- `public SortedBox<Element> getElements()`
  Returns a Box of all Elements in the Group.
- `public boolean contains(Element element)`
  Returns a boolean describing if a given Element is in this Group.
  Requires the Element to look for.
  Returns true if the given element is in the Group.

3.4.9 Class HighwaySegment

A simple Segment that only has one entrance and one exit.

Declaration

- `abstract class HighwaySegment extends SimpleUnitObservable<Element> implements Segment`

Attributes

- none

Constructors

- none

Methods

- `Connector getEntry()`
  Returns the Connector describing the entrance of the Base Segment.
- `Connector getExit()`
  Returns the Connector describing the exit of the base Segment.

3.4.10 Class MoveableConnector

A Connector that can be moved by anyone that dares.

Declaration

- `public class MoveableConnector extends Connector`

Attributes

- none

Constructors

- `MoveableConnector(ConnectorType type, Position position, Collection<AttributeAccessor <?>> accessors)`
  Requires the ConnectorType for this MoveableConnector.
  Requires the Position that this MoveableConnector is supposed to be at.
  Requires the AttributeAccessors that this MoveableConnector is supposed to have.

Methods

- none

3.4.11 Class RampSegment

A Base segment with a ramp Connector

Declaration

- `abstract class RampSegment extends HighwaySegment`

Attributes

- none

Constructors

- none

Methods

- `public Connector getRamp()`
  Returns the ramp Connector of the RampSegment.

3.4.12 Interface Segment

A Piece of a road in a RoadSystem. (see: Pflichtenheft: "Straßensegment"). Is additionally to be observed by PlausibilityCriteria.

Declaration

- `public interface Segment extends Element, Comparable<Segment>`

Methods

- `SegmentType getSegmentType()`
  Returns the SegmentType of the Segment.
- `SortedBox<Measurement<?>> getMeasurements()`
  Returns a SortedBox containing all Measurements of the Segment.
- `Box<Connector> getConnectors()`
  Returns a Box containing all Connectors of the Segment.
- `Position getCenter()`
  Returns the center Position of the Segment.
- `void move(Movement movement)`
  Moves Segment. Changes both the center point and the Positions of all included Connectors in order to preserve the shape of the Segment.
  Requires the Movement that is to be applied.

3.4.13 Class SegmentFactory

A Factory for different kinds of Segments.

Declaration

- `class SegmentFactory`

Attributes

- none

Constructors

- none

Methods

- `createBaseSegment`
  Creates a new Base segment
  Returns the new Base segment
- `createExitSegment`
  Creates a new Exit segment
  Returns the new Exit segment
- `createEntrySegment`
  Creates a new Entry segment
  Returns the new Entry segment

3.4.14 Enum SegmentType

Different Types of Segments. Used to differentiate between Segments.

Declaration

- `public enum SegmentType`

Values

- `BASE`
- `ENTRANCE`
- `EXIT`

Attributes

- none

Constructors

- none

Methods

- none

3.5 Package edu.kit.rose.model.roadsystem.measurements

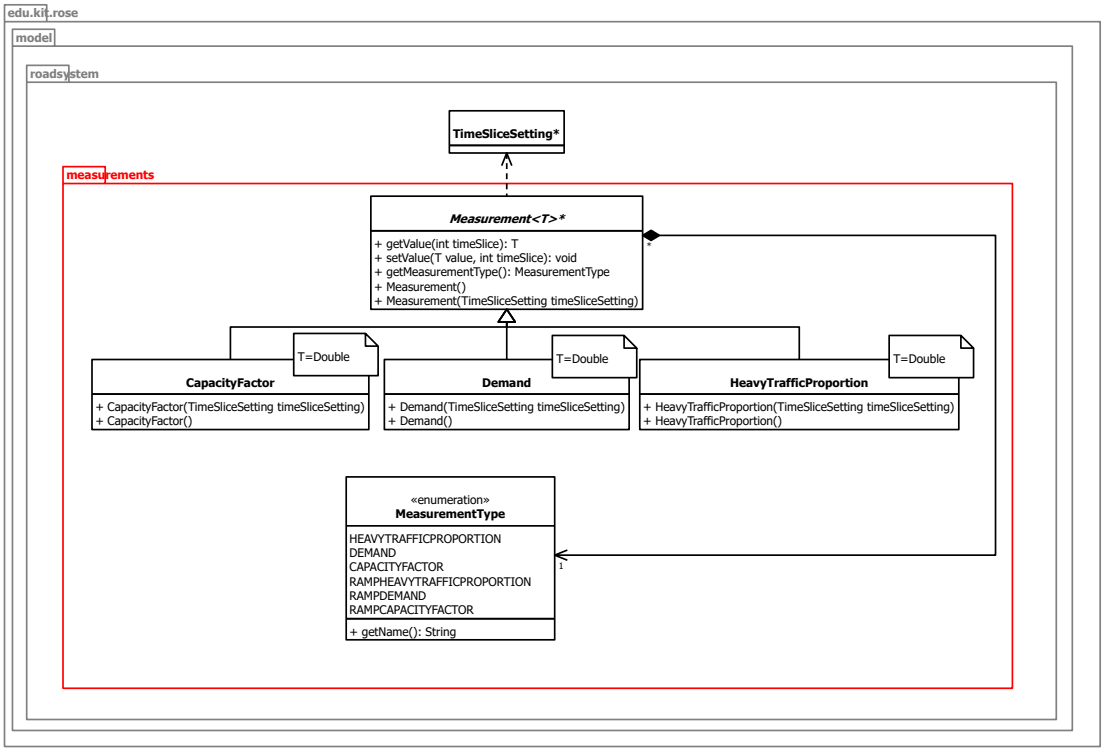

Abbildung 11: Class diagram for package edu.kit.rose.model.roadsystem.measurements

This package contains several types of Measurements as well as an enum for easy differentiation.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.5.1 Class CapacityFactor

A Measurement describing the Capacity Factor of a Segment

Declaration

- `class CapacityFactor extends Measurement<Double>`

Attributes

- none

Constructors

- `public CapacityFactor()`
  This creates a CapacityFactor with decoy values for both number of time slices and their length. Needs to be subscribed to a TimeSliceSetting to be consistent with other Measurements using the same. TimeSliceSetting has to notify after subscription.
- `public CapacityFactor(TimeSliceSetting timeSliceSetting)`
  The TimeSliceSetting provides the number of time slices as well as their length.

Methods

- none

3.5.2 Class Demand

A Measurement describing the Demand of a Segment

Declaration

- `class Demand extends Measurement<Double>`

Attributes

- none

Constructors

- `public Demand()`
  This creates a Demand with decoy values for both number of time slices and their length. Needs to be subscribed to a TimeSliceSetting to be consistent with other Measurements using the same. TimeSliceSetting has to notify after subscription.
- `public Demand(TimeSliceSetting timeSliceSetting)`
  The TimeSliceSetting provides the number of time slices as well as their length.

Methods

- none

3.5.3 Class HeavyTrafficProportion

A Measurement describing the Heavy Traffic Proportion of a Segment

Declaration

- `class HeavyTrafficProportion extends Measurement<Double>`

Attributes

- none

Constructors

- `public HeavyTrafficProportion()`
  This creates a HeavyTrafficProportion with decoy values for both number of time slices and their length. Needs to be subscribed to a TimeSliceSetting to be consistent with other Measurements using the same. TimeSliceSetting has to notify after subscription.
- `public HeavyTrafficProportion(TimeSliceSetting timeSliceSetting)`
  The TimeSliceSetting provides the number of time slices as well as their length.

Methods

- none

3.5.4 Class Measurement

A Measurement stores values for a given number of time slices. Measurements are to observe the TimeSliceSetting of the Project to configure themselves. (See Pflichtenheft: "Messwert" and "Zeitschritt"). Is to observed by the view. It to observe a TimeSliceSetting in order to be informed about global changes in time slice length and number of time slices per measurement.

Declaration

- `public abstract class Measurement<T> extends SimpleUnitObservable<Measurement <T>> implements UnitObserver<TimeSliceSetting>`

Attributes

- none

Constructors

- `public Measurement()`
  This creates a Measurement with decoy values for both number of time slices and their length. Needs to be subscribed to a TimeSliceSetting to be consistent with

other Measurements using the same. TimeSliceSetting has to notify after subscription.
- `public Measurement(TimeSliceSetting timeSliceSetting)`
  The TimeSliceSetting provides the number of timeSlices as well as their length.
  Requires the TimeSliceSetting to use.

Methods

- `public T getValue(int timeSlice)`
  Requires the int describing the timeSlot.
  Returns the value of the Measurement at the given time slice.
- `public void setValue(T value, int timeSlice)`
  Sets the given value at the given time slice.
  Requires the value to set.
  Requires an int describing the time slice.
- `getMeasurementType()`
  Returns the MeasurementType of this Measurement

3.5.5 Enum MeasurementType

Different Types of Measurement. Used for quick access to all possible measurement types.

Declaration

- `public enum MeasurementType`

Values

- `HEAVYTRAFFICPROPORTION`
- `DEMAND`
- `CAPACITYFACTOR`
- `RAMPHEAVYTRAFFICPROPORTION`
- `RAMPDEMAND`
- `RAMPCAPACITYFACTOR`

Attributes

- none

Constructors

- none

Methods

- `public String getName()`
  Returns a String containing the name of the MeasurementType.

3.6 Package edu.kit.rose.model.plausibility

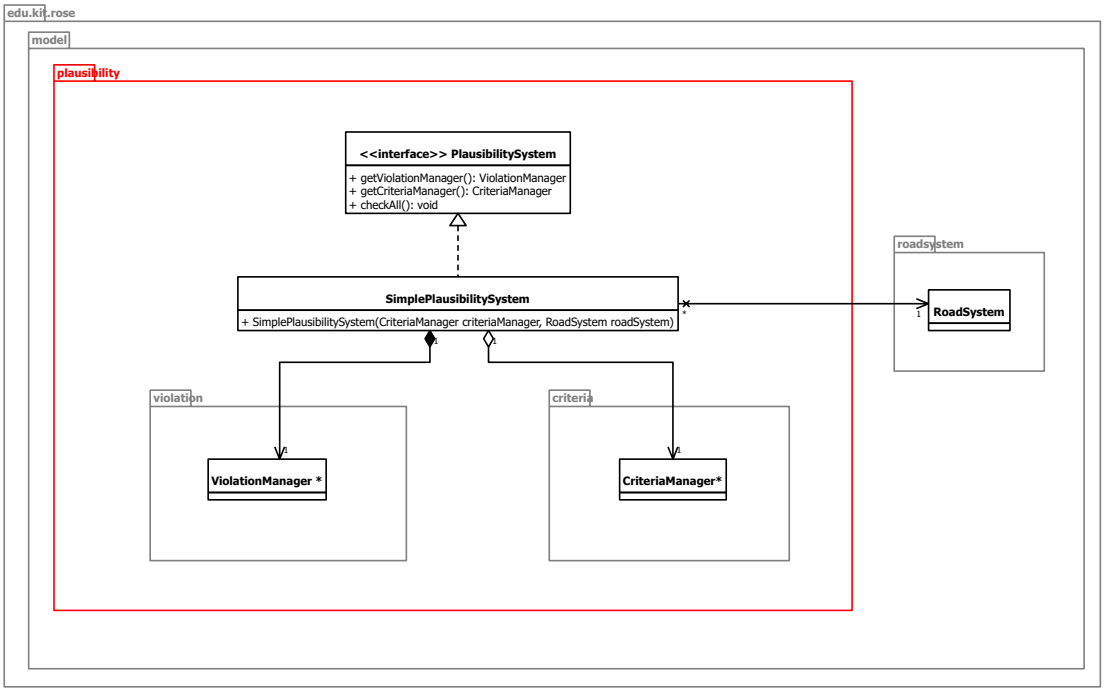

Abbildung 12: Class diagram for package edu.kit.rose.model.plausibility

This package contains the components needed for plausibility checks.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.6.1 Interface PlausibilitySystem

A Plausibility System consists of a ViolationManager and a CriteriaManager. The CriteriaManager holds PlausibilityCriteria that will be applied to a RoadSystem. Violations will be added to the ViolationManager.

Declaration

- `public interface PlausibilitySystem`

Methods

- `ViolationManager getViolationManager()`
  Returns the ViolationManager this PlausibilitySystem uses.
- `CriteriaManager getCriteriaManager()`
  Returns the CriteriaManager this PlausibilitySystem uses.
- `void checkAll()`
  Checks all PlausibilityCriteria in the CriteriaManager. This is only supposed to be used when importing a new RoadSystem or a new set of PlausibilityCriteria.

3.6.2 Class SimplePlausibilitySystem

A Standard Implementation of a PlausibilitySystem.

Declaration

- `public class SimplePlausibilitySystem implements PlausibilitySystem`

Attributes

- none

Constructors

- `public SimplePlausibilitySystem(CriteriaManager criteriaManager, RoadSystem roadSystem)`
  Requires the RoadSystem that this PlausibilitySystem is supposed to check.
  Requires the CriteriaManager holding the Criteria that are to be checked against.

Methods

- none

3.7 Package edu.kit.rose.model.plausibility.violation

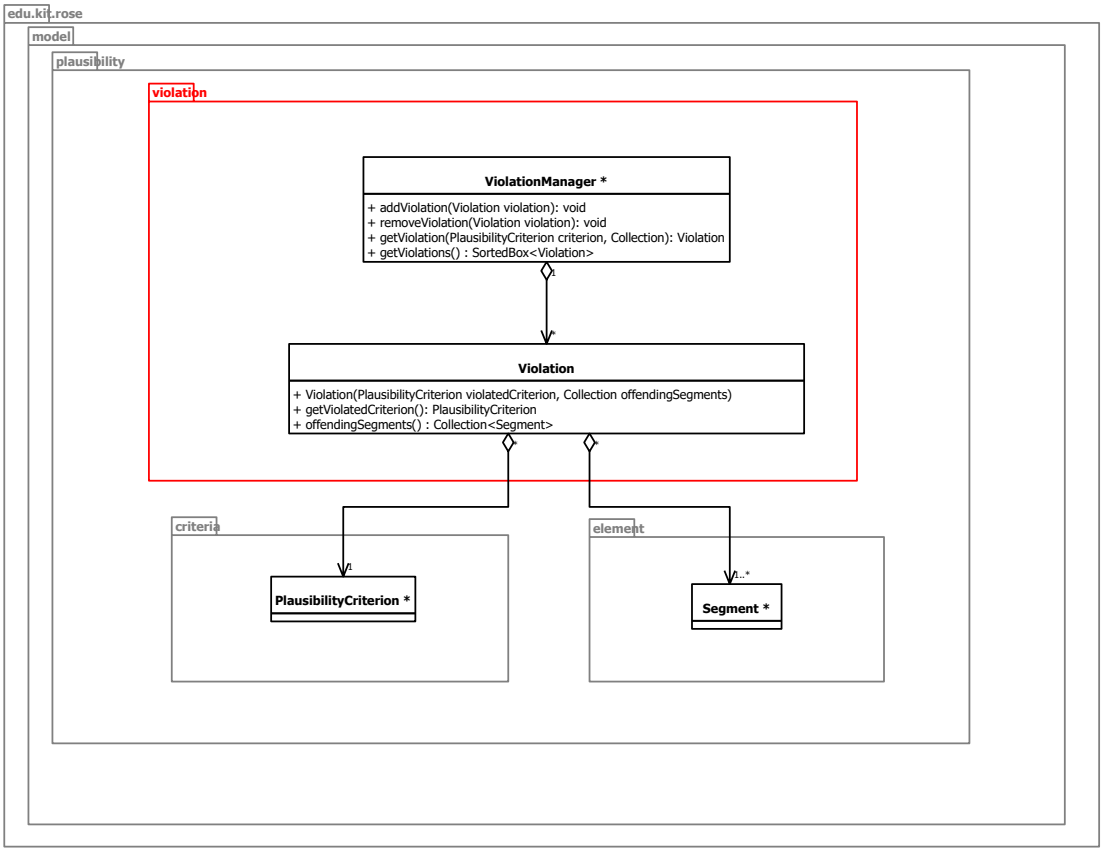

Abbildung 13: Class diagram for package edu.kit.rose.model.plausibility.violation

This package contains the ViolationManager as well as the Violations it is to hold.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.7.1 Class ViolationManager

A ViolationManager provides currently active Violations, violations can be added and removed. Violations held by this ViolationManager are mutually distinct. Is to be observed by the view.

Declaration

- `public class ViolationManager extends SimpleSetObservable<Violation, ViolationManager>, Iterable<Violation>`

Attributes

- none

Constructors

- none

Methods

- `public void addViolation(Violation violation)`
  Adds a given Violation to the ViolationManager. If the given Violation is already in the ViolationManager nothing happens.
  Requires the Violation to add.
- `public void removeViolation(Violation violation)`
  Removes a given Violation from the ones held by the violationManager. If the given Violation does not exist nothing happens.
  Requires the Violation to remove.
- `public Violation getViolation(PlausibilityCriterion criterion, Collection<Segment> offendingSegments)`
  Returns the Violation of a given PlausibilityCriterion that is caused by the given Segments.
  Requires the PlausibilityCriterion that the searched Violation offends against.
  Requires the Segments that cause the Violation.
  Returns the violation against the given Criterion by the given Segments.
- `public SortedBox<Violation> getViolations()`
  Returns all Violations currently held by the ViolationManager.
  Returns a SortedBox containing all Violations.

3.7.2 Class Violation

A Violation is an offense against a PlausibilityCriterion caused by one Segment or a Connection of multiple Segments.

Declaration

- `public class Violation`

Attributes

- none

Constructors

- `public Violation(PlausibilityCriterion violatedCriterion, Collection<Segment > offendingSegments)`
  Requires the PlausibilityCriterion this SimpleViolation offends.
  Requires the Segments that cause this SimpleViolation.

Methods

- `public PlausibilityCriterion getViolatedCriterion()`
  Returns the violated PlausibilityCriterion.
- `public Collection<Segment> offendingSegments()`
  Returns the Segments that caused this Violation

3.8 Package edu.kit.rose.model.plausibility.criteria

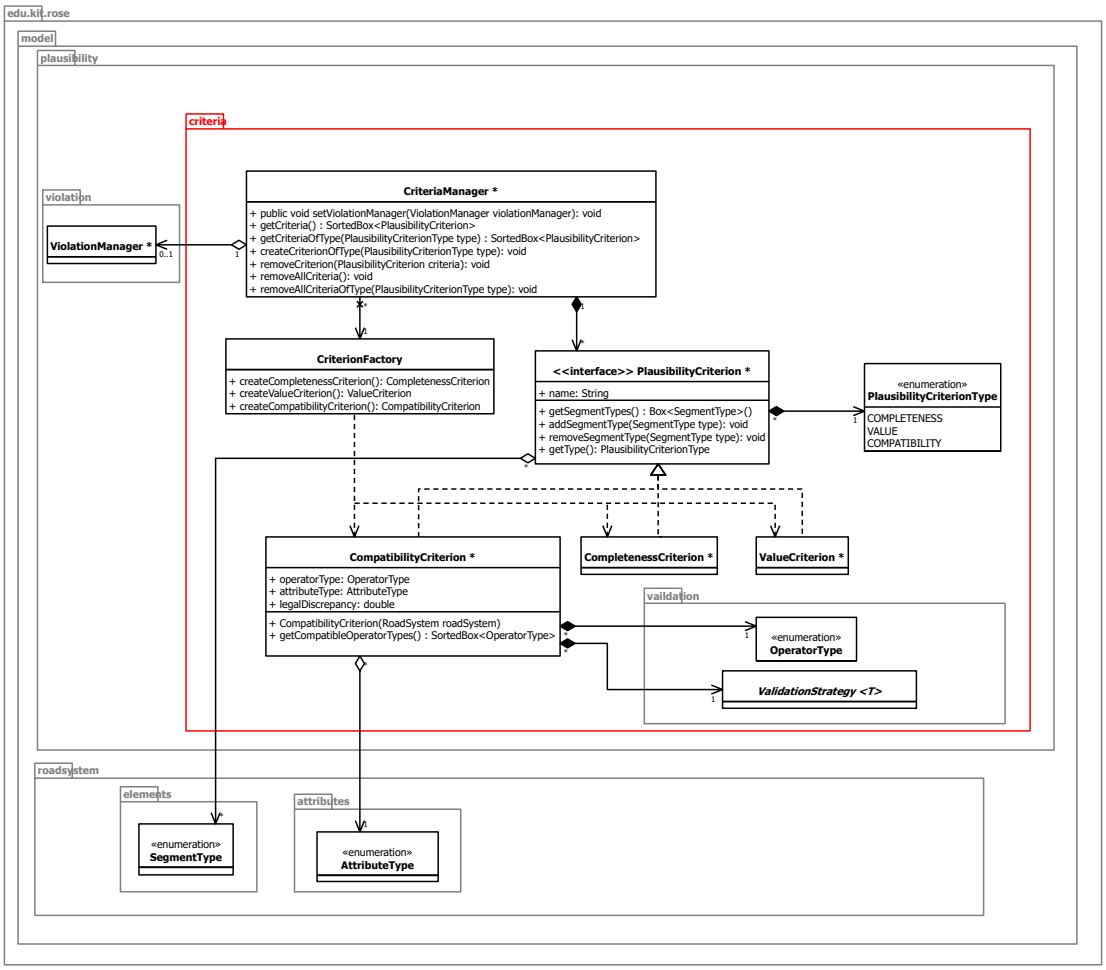

Abbildung 14: Class diagram for package edu.kit.rose.model.plausibility.criteria

This package contains the CriteriaManager as well as the different PlausibilityCriteria that it is to hold.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.8.1 Interface PlausibilityCriterion

A PlausibilityCriterion (see Pflichtenheft: “Plausibilitätskriterium”) is used to check if a RoadSystem fulfills logical requirements to attributes and Connections between Segments. It represents one such rule. Provided with a ViolationManager it will create Violations and add them to it. Is to be observed by both the view and the ApplicationDataSystem. Is to observe Segments as it needs to be notified whenever a Segment changes in order to check the new values.

Declaration

- `public interface PlausibilityCriterion extends UnitObserver<Segment>, SetObservable<SegmentType, PlausibilityCriterion>`

Methods

- `String getName()`
  Returns the name of this Criterion
- `void setName(String name)`
  Requires the new name this PlausibilityCriterion is supposed to have.
- `Box<SegmentType> getSegmentTypes()`
  Returns a Box containing all SegmentTypes this Criterion is applied to.
- `void addSegmentType(SegmentType type)`
  Adds a SegmentType that this Criterion should be applied to. If the given SegmentType was already added nothing happens.
  Requires the SegmentType that this Criterion should be applied to.
- `void removeSegmentType(SegmentType type)`
  Removes a SegmentType that this Criterion should no longer be applied to. If the given SegmentType is not present nothing happens.
  Requires the SegmentType that this Criterion should no longer be applied to.
- `PlausibilityCriterionType getType()`
  Returns the type of criterion this is.

3.8.2 Class CompatibilityCriterion

Modells a Compatiblity Criterion. (See Pflichtenheft: “Kompatibilitätskriterium”)

Declaration

- `public class CompatibilityCriterion extends SimpleSetObservable<SegmentType, PlausibilityCriterion> implements PlausibilityCriterion`

Attributes

- none

Constructors

- `CompatibilityCriterion(RoadSystem roadSystem)`
  Requires the Roadsystem this Criterion is applied to.

Methods

- `public AttributeType getAttributeType()`
  Returns the AttributeType that this Criterion is checking.
- `public void setAttributeType(AttributeType attributeType)`
  Requires the AttributeType that this Criterion is supposed to check.
- `public OperatorType getOperatorType()`
  Returns the type of Operator this Criterion is using.
- `public void setOperatorType(OperatorType operatorType)`
  Requires the Type of Operator this Criterion is supposed to use.
- `public SortedBox<OperatorType> getCompatibleOperatorTypes()`
  Gives a SortedBox of OperatorTypes that contains all Types that are compatible with this criterion. (This is depending on the AttributeType this criterion has)
  Returns a SortedBox containing all OperatorTypes that are compatible with this criterion.
- `public double getLegalDiscrepancy()`
  Returns the legal discrepancy numeric values can have to be accepted by this criterion.
- `public void setLegalDiscrepancy(double discrepancy)`
  Requires the legal discrepancy numeric values are supposed to have to be accepted by this criterion.

3.8.3 Class CompletenessCriterion

Models a completeness criterion (see Pflichtenheft: "Vollständigkeitskriterium").
This type of Criterion can be used to ensure certain Attributes are set to a value if necessary for export or similar.

Declaration

- `class CompletenessCriterion extends SimpleSetObservable<SegmentType, PlausibilityCrit > implements PlausibilityCriterion`

Attributes

- none

Constructors

- none

Methods

- none

3.8.4 Class ValueCriterion

A ValueCriterion represents a value criterion (see Pflichtenheft: "Wertebereichkriterium").
This checks if the value of an AttributeType of a Segment is legal.

Declaration

- `class ValueCriterion extends SimpleSetObservable<SegmentType, PlausibilityCriterion > implements PlausibilityCriterion`

Attributes

- none

Constructors

- none

Methods

- none

3.8.5 Class CriteriaManager

A CriteriaManager holds PlausibilityCriteria.
It provides functions to create and remove criteria as well as getters for them.

Declaration

- `public class CriteriaManager extends SimpleSetObservable<PlausibilityCriterion , CriteriaManager> implements UnitObserver<PlausibilityCriterion>`

Attributes

- none

Constructors

- none

Methods

- `public void setViolationManager(ViolationManager violationManager)`
  Sets the ViolationManager that will receive the Violations against PlausibilityCriteria in this CriteriaManager.Sets the ViolationManager that will receive the Violations of PlausibilityCriterion in this CriteriaManager.
  Requires the ViolationManager this CriteriaManager is supposed to use.
- `public SortedBox<PlausibilityCriterion> getCriteria()`
  Returns a SortedBox containing all PlausibilityCriteria that this CriteriaManager contains.
- `public SortedBox<PlausibilityCriterion> getCriteriaOfType(PlausibilityCriterionType type)`
  Returns a SortedBox containing all PlausibilityCriteria of the given PlausibilityCriterionType that this CriteriaManager contains.
  Requires the CriteriaType to look for.
  Returns a SortedBox containing all PlausibilityCriterion of the given type.
- `public void createCriterionOfType(PlausibilityCriterionType type)`
  Creates a new PlausibilityCriterion with the given PlausibilityCriterionType.
  Requires the Type of the new PlausibilityCriterion
- `public void removeCriterion(PlausibilityCriterion criteria)`
  Removes a given PlausibilityCriterion from this CriterionManager. If the given PlausibilityCriterion does not exist nothing happens.
  Requires the Criterion to remove.
- `public void removeAllCriteria()`
  Removes all PlausibilityCriteria from this CriterionManager.
- `public void removeAllCriteriaOfType(PlausibilityCriterionType type)`
  Removes all PlausibilityCriteria of the given PlausibilityCriterionType from this CriterionManager.
  Requires the type of PlausibilityCriterion to remove.

3.8.6 Class CriterionFactory

A Factory for different kinds of PlausibilityCriteria.

Declaration

- `class CriterionFactory`

Attributes

- none

Constructors

- none

Methods

- `public CompletenessCriterion createCompletenessCriterion()`
  Creates a new CompletenessCriterion
  Returns the new CompletenessCriterion
- `public ValueCriterion createValueCriterion()`
  Creates a new ValueCriterion
  Returns the new ValueCriterion
- `public CompatibilityCriterion createCompatibilityCriterion()`
  Creates a new CompatibilityCriterion
  Returns the new CompatibilityCriterion

3.8.7 Enum PlausibilityCriterionType

Different types of PlausibilityCriterions. Used to determine the concrete type of given PlausibilityCriterion. Allows quick access to all possible criterion types.

Declaration

- `public enum PlausibilityCriterionType`

Values

- `COMPLETENESS`
- `VALUE`
- `COMPATIBILITY`

Attributes

- none

Constructors

- none

Methods

- none

3.9 Package edu.kit.rose.model.plausibility.criteria.validation

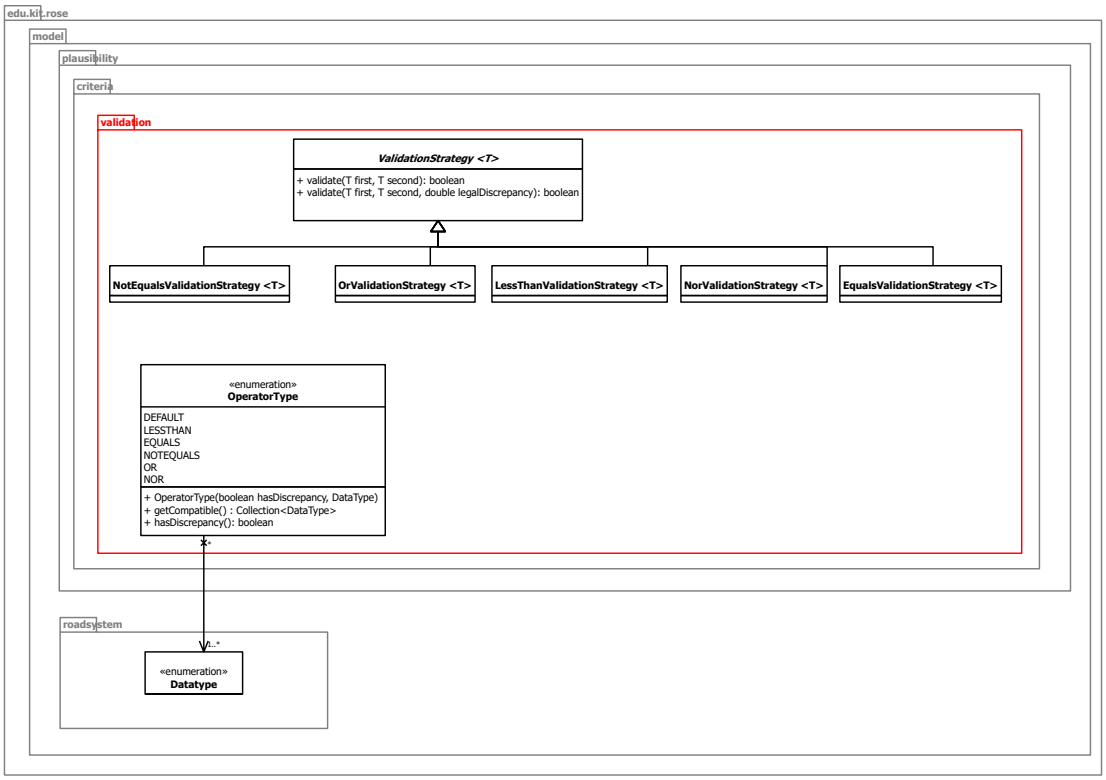

Abbildung 15: Class diagram for package edu.kit.rose.model.plausibility.criteria.validation

This package contains different ValidationStrategies as well as an enum describing the different yypes of them.
Classes in the UML Diagrams marked with '*' extend one of these classes: SimpleUnitObservable, SimpleSetObservable, DualSetObservable; and / or implement one of these Interfaces: UnitObserver, SetObserver, DualSetObserver;
These connections are not shown to improve readability but are documented in the Class signatures.

3.9.1 Class EqualsValidationStrategy

Describes an equals operator for different Objects. Uses the java.util.equals function.
This is a Strategy in the Strategy Pattern that these ValidationStrategies implement with the abstract Strategy ValidationStrategy and the CompatibilityCriterion as Client.
T is the Type of value this Operator uses.

Declaration

- `class EqualsValidationStrategy<T> extends ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.9.2 Class LessThanValidationStrategy

Describes a LessThan Operator for numerical values. "<"
This is a Strategy in the Strategy Pattern that these ValidationStrategies implement with the abstract Strategy ValidationStrategy and the CompatibilityCriterion as Client. T is the Type of value this Operator uses.

Declaration

- `class LessThanValidationStrategy<T> extends ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.9.3 Class NorValidationStrategy

An Operator describing the logical NOR function for two booleans.
This is a Strategy in the Strategy Pattern that these ValidationStrategies implement with the abstract Strategy ValidationStrategy and the CompatibilityCriterion as Client.

T is the Type of value this Operator uses.

Declaration

- `class NorValidationStrategy<T> extends ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.9.4 Class NotEqualsValidationStrategy

Describes a NotEquals Operator for different Objects. It uses the java.util.equals function. This is a Strategy in the Strategy Pattern that these ValidationStrategies implement with the abstract Strategy ValidationStrategy and the CompatibilityCriterion as Client.
T is the Type of value this Operator uses.

Declaration

- `class NotEqualsValidationStrategy<T> extends ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.9.5 Class OrValidationStrategy

An Operator describing the logical OR function for two booleans.
This is a Strategy in the Strategy Pattern that these ValidationStrategies implement with the abstract Strategy ValidationStrategy and the CompatibilityCriterion as Client. T is the Type of value this Operator uses.

Declaration

- `class OrValidationStrategy<T> extends ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.9.6 Enum OperatorType

An enum that describes different kinds of ValidationStrategys and the DataTypes they are compatible with.

Declaration

- `public enum OperatorType`

Values

- `DEFAULT`
- `LESSTHAN`
- `EQUALS`
- `NOTEQUALS`
- `OR`

- `NOR`

Attributes

- none

Constructors

- none

Methods

- `Collection<DataType> getCompatible()`
  Returns all DataTypes that are compatible with this OperatorType.
- `public boolean hasDiscrepancy()`
  Returns true if the OperatorType can be used with a discrepancy.

3.9.7 Class ValidationStrategy

A ValidationStrategy models a simple logical evaluation between two values of type T. It can also hold a double that holds a discrepancy for numerical Values. Then it will subtract the given values and check, if the absolute difference is smaller than the legalDiscrepancy.
This is the abstract Strategy in the Strategy Pattern with the CompatibilityCriteria as the Client.
Requires the Type of value this Operator uses.

Declaration

- `abstract class ValidationStrategy<T>`

Attributes

- none

Constructors

- none

Methods

- `abstract boolean validate(T first, T second)`
  Validates if the two given values are a legal combination in sense of the ValidationStrategy.
  Requires the first value for comparison.

Requires the second value for comparison.
Returns true if the given values are a legal combination in sense of the ValidationStrategy.

- `abstract boolean validate(T first, T second, double legalDiscrepancy)`
  Validates if the difference between the two given values are smaller than the given legalDiscrepancy.If this Validation Strategy does not support a Discrepancy, the third parameter is ignored.
  Requires the first value for comparison.
  Requires the second value for comparison.
  Requires if this Strategy supports a legalDiscrepancy it will use this one.
  Returns true if the given values are a legal combination in sense of the Strategy and the legalDiscrepancy.

3.10 Package edu.kit.rose.controller

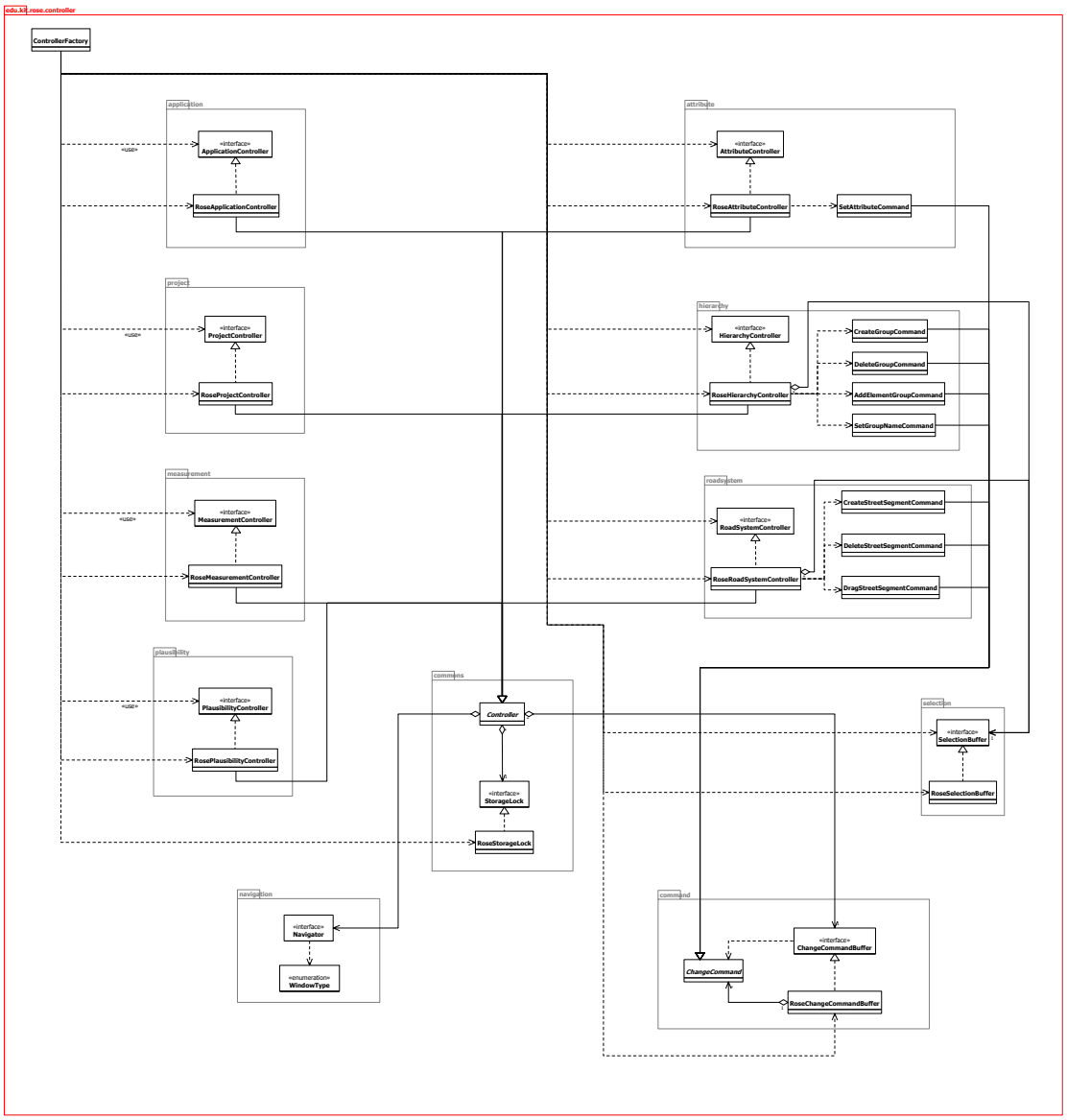

Abbildung 16: Class diagram for package edu.kit.rose.controller

This package manages the workflow between `edu.kit.rose.view` and `edu.kit.rose.model`.

3.10.1 Class ControllerFactory

Encapsulates the creation of all controllers. It also handles the constructor arguments which are necessary for multiple controllers to work together.

Declaration

- `public class ControllerFactory`

Attributes

- none

Constructors

- `public ControllerFactory(Navigator navigator, LanguageSelector languageSelector , ApplicationDataSystem applicationDataSystem, Project project)`
  Creates a new `ControllerFactory`. Requires `navigator`, the navigator to initialize all controller with. Requires `languageSelector`, component on which controllers can set the applications language. Requires `project`, the model facade for project data. Requires `applicationDataSystem` the model facade for application data.

Methods

- `public ApplicationController getApplicationController()`
  Creates and returns an ApplicationController instance.
- `public AttributeController getAttributeController()`
  Creates and returns an AttributeController instance.
- `public HierarchyController getHierarchyController()`
  Creates and returns a HierarchyController instance.
- `public MeasurementController getMeasurementController()`
  Creates and returns MeasurementController instance.
- `public PlausibilityController getPlausibilityController()`
  Creates and returns a PlausibilityController instance.
- `public ProjectController getProjectController()`
  Creates and returns a ProjectController instance.
- `public RoadSystemController getRoadSystemController()`
  Creates and returns RoadSystemController instance.

3.11 Package edu.kit.rose.controller.application

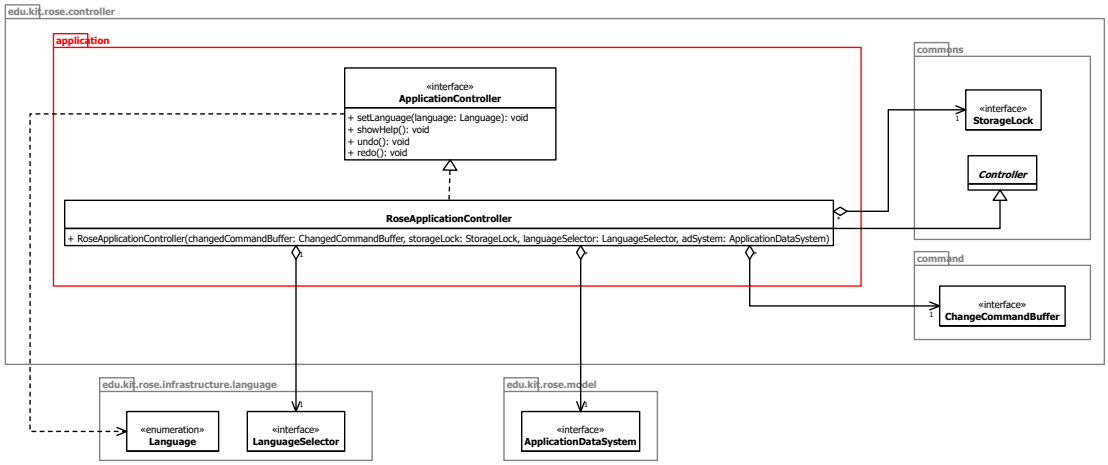

Abbildung 17: Class diagram for package edu.kit.rose.controller.application

This package contains the application controller.

3.11.1 Interface ApplicationController

Provides functionality for application settings and undoing/redoing of user actions.

Declaration

- `public interface ApplicationController`

Methods

- `void setLanguage(Language language)`
  Updates the applications displaying language to the given language.
- `void showHelp()`
  Shows a help-window to user.
- `void undo()`
  Undoes the last undoable action of the user.
- `void redo()`
  Redoes the last undoable action of the user.

3.11.2 Class RoseApplicationController

Provides functionality for application settings and undoing/redoing of user actions.

Declaration

- `public class RoseApplicationController extends Controller implements ApplicationContr`

Attributes

- none

Constructors

- `public RoseApplicationController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, LanguageSelector languageSelector, ApplicationDataSystem applicationDataSystem)`
  Creates a new `RoseApplicationController`. Requires the `ChangeCommandBuffer`. Requires `coordinator`, the coordinator for controller actions. Requires `languageSelector` , the class that configures the applications language. Requires `applicationDataSystem` , the model facade for application data

Methods

- none

3.12 Package edu.kit.rose.controller.command

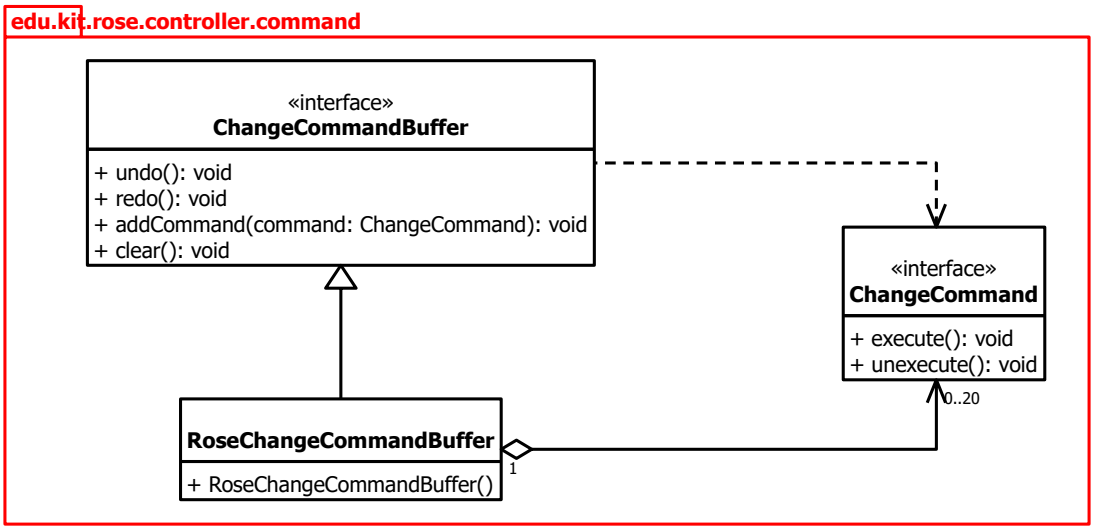

Abbildung 18: Class diagram for package edu.kit.rose.controller.command

This package contains the abstract change command and the change command buffer.

3.12.1 Interface ChangeCommand

Represents an action that can be undone and redone.

Declaration

- `public interface ChangeCommand`

Methods

- `void execute()`
  Executes or redoes the defined action.
- `void unexecute()`
  Undoes the defined action.

3.12.2 Interface ChangeCommandBuffer

Represents a container that stores a fixed number of `ChangeCommands` and provides the functionality to execute these commands in context of change.

Declaration

- `public interface ChangeCommandBuffer`

Methods

- `void undo()`
  Unexecutes/Undoes the newest `ChangeCommand` that has not bee undo.
- `void redo()`
  Rexecutes/Redoes the last undone `ChangeCommand`.
- `void addCommand(ChangeCommand changeCommand)`
  Adds a new `ChangeCommand`.

3.12.3 Class RoseChangeCommandBuffer

Implements a container that stores `ChangeCommands`. This container is responsible for executing the stored `ChangeCommands` in the correct order when it comes to undo/redo requests.

Declaration

- `public class RoseChangeCommandBuffer implements ChangeCommandBuffer`

Attributes

- none

Constructors

- none

Methods

- none

3.13 Package edu.kit.rose.controller.attribute

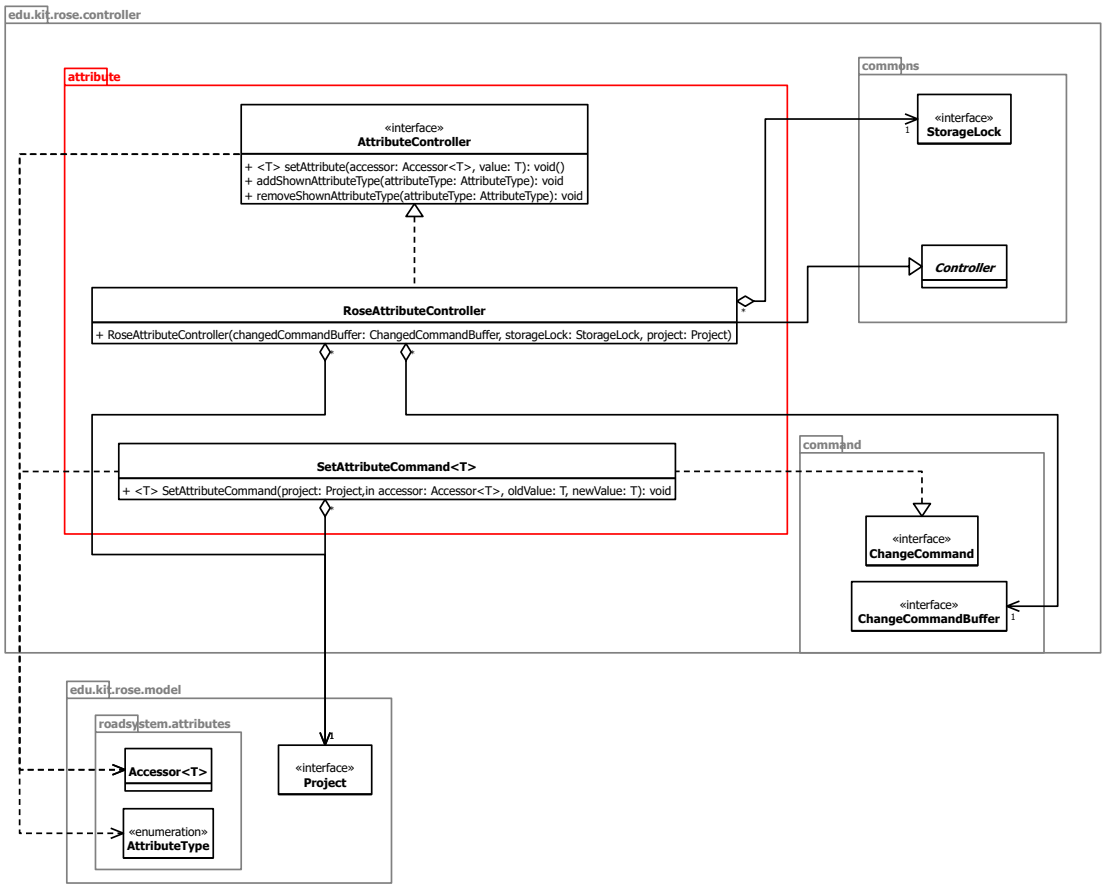

Abbildung 19: Class diagram for package edu.kit.rose.controller.attribute

This package contains the attribute controller and associated change commands.

3.13.1 Interface AttributeController

Provides methods for setting the values of attribute accessors.

Declaration

- `public interface AttributeController`

Methods

- `<T> void setAttribute(AttributeAccessor<T> accessor, T value)`

Sets the value of a given accessor to a given value.

- `void addShownAttributeType(AttributeType attributeType)`
  Adds an `AttributeType` to the list of `AttributeTypes`, that are displayed on `Segments`.
- `void removeShownAttributeType(AttributeType attributeType)`
  Removes an `AttributeType` from the list of `AttributeTypes`, that are displayed on `Segments`.

3.13.2 Class RoseAttributeController

Creates a new `RoseAttributeController`.

Declaration

- `public class RoseAttributeController extends Controller implements AttributeControlle`

Attributes

- none

Constructors

- `public RoseAttributeController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, Project project)`
  Creates a new `RoseAttributeController`. Requires `changeCommandBuffer`, the buffer for change commands. Requires `storageLock` the coordinator for controller actions. Requires `project`, the model facade for project data.

Methods

- none

3.13.3 Class SetAttributeAccessorCommand

Encapsulates the functionality of setting an attribute accessors value and makes it changeable.

Declaration

- `public class SetAttributeAccessorCommand<T> implements ChangeCommand`

Attributes

- none

Constructors

- `public SetAttributeAccessorCommand(Project project, AttributeAccessor< T> accessor, T oldValue, T newValue)`
  Creates a `SetAttributeAccessorCommand` that sets an accessor's value to a new value. Requires `project`, the model facade for project data. Requires `accessor`, the accessor with the value to be set. Requires `oldValue`, the previous value of the accessor. Reqires `newValue`, the value of the accessor

Methods

- none

3.14 Package edu.kit.rose.controller.commons

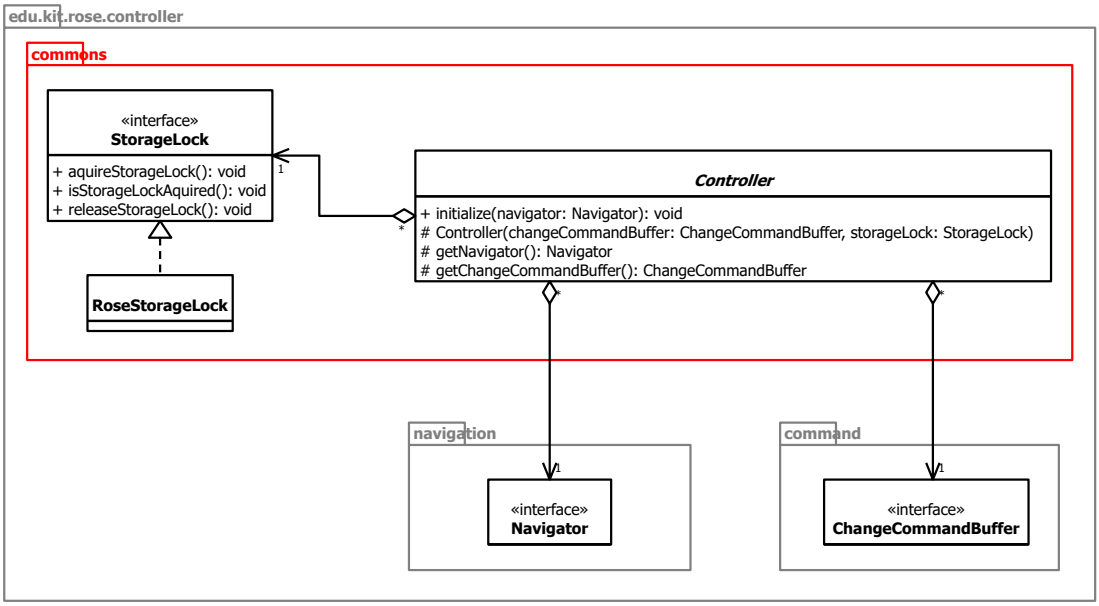

Abbildung 20: Class diagram for package edu.kit.rose.controller.commons

This package contains all common classes.

3.14.1 Interface StorageLock

Provides the functionality for `Controllers` to synchronize and coordinate their actions.

Declaration

- `public interface StorageLock`

Methods

- `void aquireStorageLock()`
  Aquires the storage lock for the caller.
- `boolean isStorageLockAquired()`
  Checks if the storage lock is aquired.
- `void releaseStorageLock()`
  Releases the storage lock if the caller has already aquired it.

3.14.2 Class RoseStorageLock

Provides the functionality for `Controllers` to synchronize and coordinate their actions.

Declaration

- `public class RoseStorageLock implements StorageLock`

Attributes

- none

Constructors

- none

Methods

- none

3.14.3 Class Controller

Base class for all controllers. Provides services for navigation handling, change handling and coordination of actions between multiple controllers.

Declaration

- `public abstract class Controller`

Attributes

- none

Constructors

- `protected Controller(ChangeCommandBuffer changeCommandBuffer, StorageLock storageLock)`
  Requires `ChangeCommandBuffer`, the buffer that contains the performed actions.
  Requires `StorageLock`.

Methods

- `public final void initialize(Navigator navigator)`
  Initializes the `Controller` with a `Navigator` and a `ChangeCommandBuffer`.
- `protected final Navigator getNavigatior()`
  Returns the `Navigator` instance of the `Controller`.

- `protected final ChangeCommandBuffer getChangeCommandBuffer()`
  Returns the `ChangeCommandBuffer` of the `Controller`.

3.15 Package edu.kit.rose.controller.hierarchy

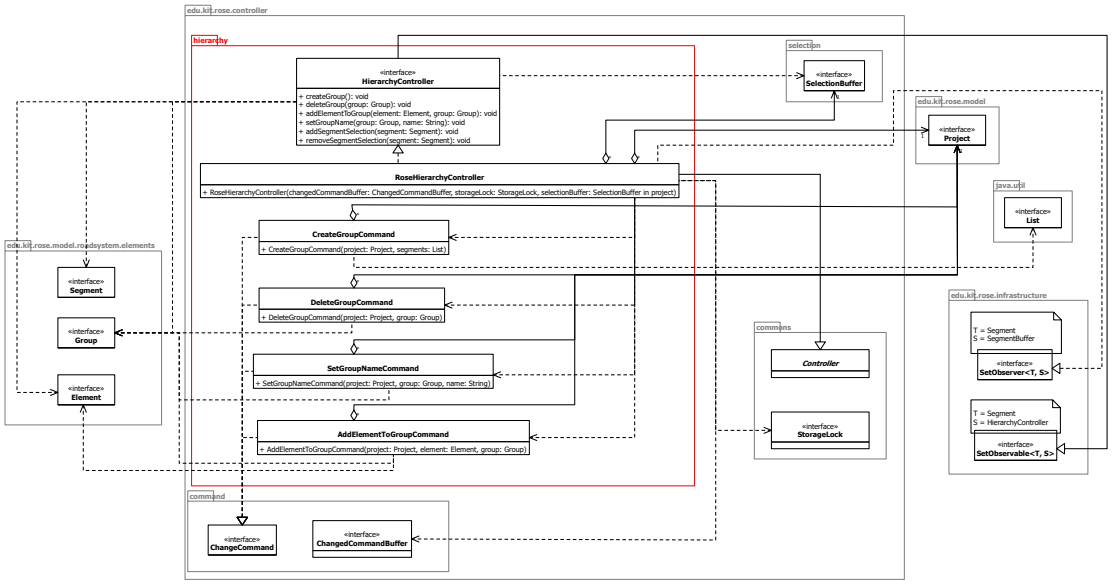

Abbildung 21: Class diagram for package edu.kit.rose.controller.hierarchy

This package contains the hierarchy controller and associated change commands.

3.15.1 Interface HierarchyController

Provides the functionality to manage the element hierarchy of the roadsystem.

Declaration

- `public interface HierarchyController extends SetObservable<Segment, HierarchyControll`
  `>`

Methods

- `void createGroup()`
  Creates a new `Group` that contains all selected `Segments` in the `Element` hierarchy.
- `void deleteGroup(Group group)`
  Deletes a `Group` in the `Element` hierarchy. The `Segments` the `Group` contains will not be deleted.
- `void addElementToGroup(Element element, Group group)`
  Adds an element to a group if the group does not already contain the element.
- `void setGroupName(Group group, String name)`
  Renames a given `Group` with a given name.

3.15.2 Class RoseHierarchyController

Provides the functionality to manage the `Element` hierarchy of the roadsystem.

Declaration

- `public class RoseHierarchyController extends Controller implements HierarchyControlle , SetObserver<Segment, SelectionBuffer>`

Attributes

- none

Constructors

- `public RoseHierarchyController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, Project project)`
  Creates a new `RoseHierarchyController`. Requires `changeCommandBuffer`, the buffer for change commands. Requires `storageLock`, the coordinator for controller actions. Requires `selectionBuffer`, the container that stores selected segments. Requires `project`, the model facade for project data.

Methods

- none

3.15.3 Class AddElementToGroupCommand

Encapsulates the functionality of adding an `Element` to a `Group` and makes it changeable.

Declaration

- `public class AddElementToGroupCommand implements ChangeCommand.`

Attributes

- none

Constructors

- `public AddElementToGroupCommand(Project project, Element element, Group group)`
  Creates a new `AddElementToGroupCommand` that adds an `Element` to a `Group`. Requires `project`, the model facade to execute the `AddElementToGroupCommand` on.

Requires `element`, the element to add. Requires `group`, the group to add an element to.

Methods

- none

3.15.4 Class CreateGroupCommand

Encapsulates the functionality of creating a new `Group` and makes it changeable.

Declaration

- `public class CreateGroupCommand implements ChangeCommand.`

Attributes

- none

Constructors

- `public CreateGroupCommand(Project project, List<Segment> segments)` Creates a `CreateGroupCommand` that creates a `Group` out of a list of `Segments`. Requires `Project` to execute the `CreateGroupCommand`. Requires `segments`, the `Segments` that will be in the `Group`.

Methods

- none

3.15.5 Class DeleteGroupCommand

Encapsulates the functionality of deleting a `Group` and makes it changeable.

Declaration

- `public class DeleteGroupCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public DeleteGroupCommand(Project project, Group group)`
  Creates a `DeleteGroupCommand` that deletes the given `Group`. Requires `project`, the model facade to execute the `DeleteGroupCommand` on. Requires `group`, the `Group` to be deleted.

Methods

- none

3.15.6 Class SetGroupNameCommand

Encapsulates the functionality of renaming a `Group` and makes it changeable.

Declaration

- `public class SetGroupNameCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public SetGroupNameCommand(Project project, Group group, String newName)`
  Creates a `SetGroupNameCommand` that assigns a new value to a `Group`. Requires `project`, the model facade to execute the `SetGroupNameCommand` on. Requires `group`, the `Group` with a name to change. Requires `newName`, the new name of the `Group`.

Methods

- none

3.16 Package edu.kit.rose.controller.measurement

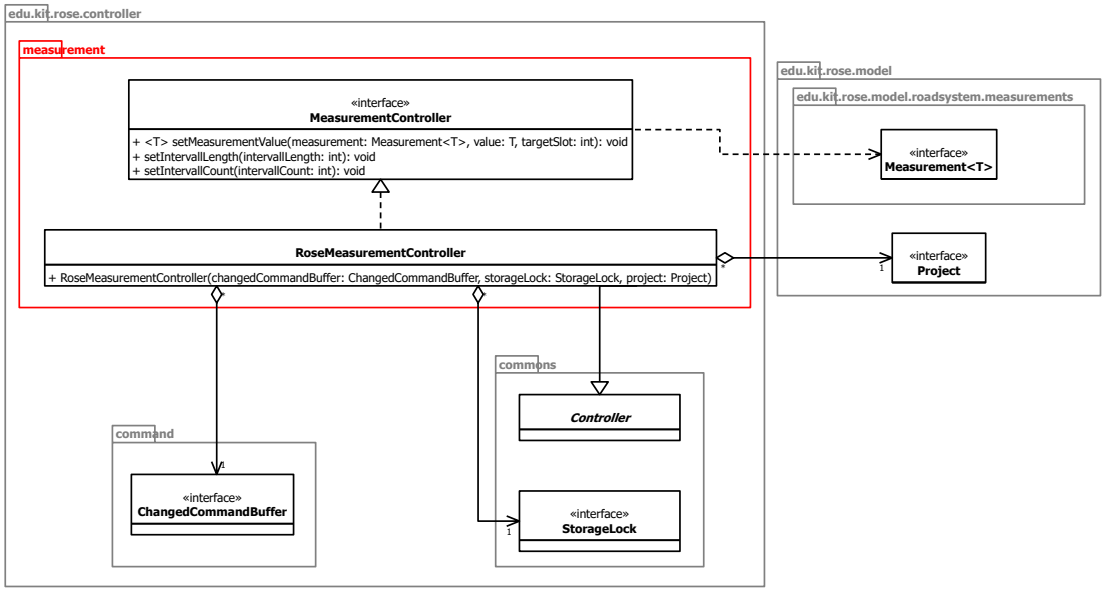

Abbildung 22: Class diagram for package edu.kit.rose.controller.measurement

This package contains the measurement controller.

3.16.1 Interface MeasurementController

Provides the functionality to set `Measurement` values and the settings which belong to the `Measurement`s.

Declaration

- `public interface MeasurementController`

Methods

- `<T> void setMeasurementValue(Measurement<T> measurement, T value, int targetSlot)`
  Sets the value of a `Measurement` to a given value.
- `void setIntervalLength(int intervalLength)`
  Sets the length of all measurement value's intervals.
- `void setIntervalCount(int intervalCount)`
  Sets the amount of measurement intervals per measurement value.

3.16.2 Class RoseStorageLock

Provides the functionality to set measurement values and the settings which belong to them.

Declaration

- `public class RoseMeasurementController extends Controller implements MeasurementContr`

Attributes

- none

Constructors

- `public RoseMeasurementController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, Project project)`
  Creates a new `link RoseMeasurementController`. Requires `changeCommandBuffer` , the buffer for change commands. Requires `storageLock`, the coordinator for `Controller` actions. Requires `project`, the model facade for project data

Methods

- none

3.17 Package edu.kit.rose.controller.navigation

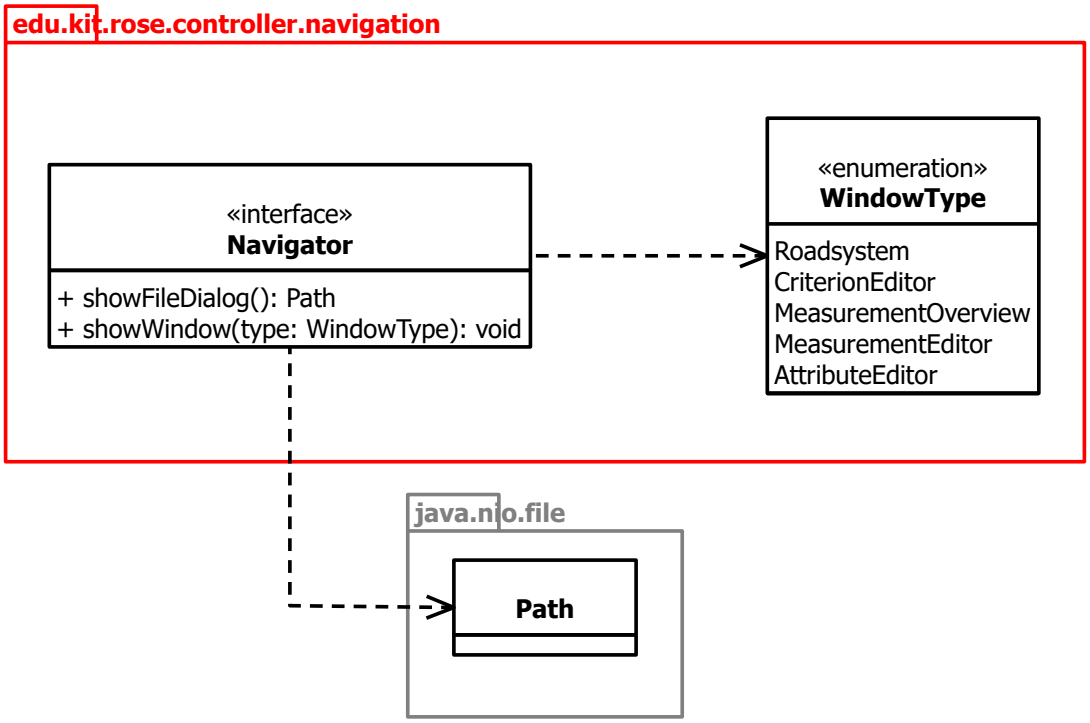

Abbildung 23: Class diagram for package edu.kit.rose.controller.navigation

This package contains the Navigator class and all needed enums for it.

3.17.1 Interface Navigator

Provides functionality to create new windows and file dialogs.

Declaration

- `public interface Navigator`

Methods

- `void showWindow(WindowType windowType)`
  Creates and shows a new window of specified type.
- `Path showFileDialog()`
  Creates and shows a file dialog, that provides to user the possibility to choose the path of a file and return that path.

3.17.2 Enum WindowType

Contains different types of views.

Declaration

- `public enum WindowType`

Values

- `Roadsystem`
  References the part of the GUI where the `RoadSystem` is shown.
- `Criterion`
  References the part of the GUI where the plausibility criteria are shown.
- `MeasurementOverview`
  References the part of the GUI where an overview over all `Measurements` is shown.
- `MeasurementEditor`
  References the part of the GUI where `Measurement` values can be edited.
- `Attribute`
  References the part of the GUI where attribute values can be edited.

Attributes

- none

Constructors

- none

Methods

- none

3.18 Package edu.kit.rose.controller.plausibility

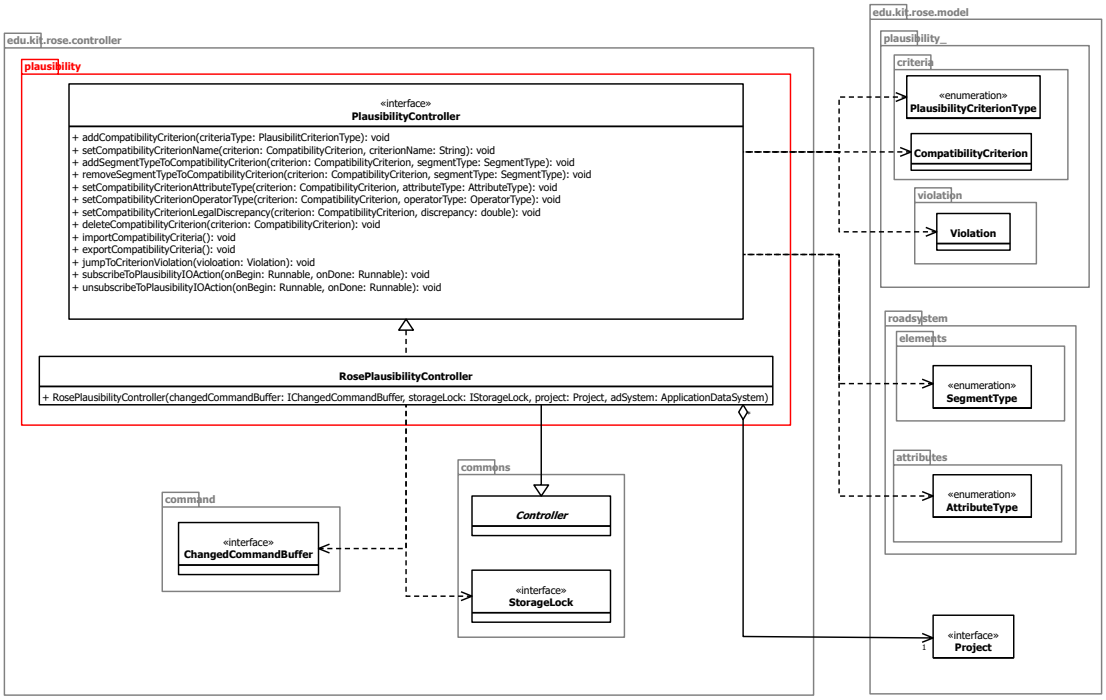

Abbildung 24: Class diagram for package edu.kit.rose.controller.plausibility

This package contains the plausibility controller and associated change commands.

3.18.1 Interface PlausibilityController

Provides functionality to work with plausibility criteria and to export and import these plausibility criteria.

Declaration

- `public interface PlausibilityController`

Methods

- `void addCompatibilityCriterion(PlausibilityCriterionType type)`
  Adds a `CompatibilityCriterion` of the given type.
- `void setCompatibilityCriterionName(CompatibilityCriterion criterion, String criterionName)`
  Set the name of a `CompatibilityCriterion`.

- `void addSegmentTypeToCompatibilityCriterion(CompatibilityCriterion criterion , SegmentType segmentType)`
  Adds a `SegmentType` to a `CompatibilityCriterion`.
- `void removeSegmentTypeToCompatibilityCriterion(CompatibilityCriterion criterion, SegmentType segmentType)`
  Removes a `SegmentType` from a `CompatibilityCriterion`.
- `void setCompatibilityCriterionAttributeType(CompatibilityCriterion criterion ,AttributeType attributeType)`
  Sets the `AttributeType` of an `CompatibilityCriterion`.
- `void setCompatibilityCriterionOperatorType(CompatibilityCriterion criterion , OperatorType operatorType)`
  Sets the `OperatorType` of an `CompatibilityCriterion`.
- `void setCompatibilityCriterionLegalDiscrepancy(CompatibilityCriterion criterion, double discrepancy)`
  Sets the discrepancy of an `CompatibilityCriterion`.
- `void deleteCompatibilityCriterion(CompatibilityCriterion criterion)`
  Deletes the specified `CompatibilityCriterion`.
- `void importCompatibilityCriteria()`
  Opens a file chooser and imports the compatibility criteria from a file chosen by the user.
- `void exportCompatibilityCriteria()`
  Opens a file chooser and exports the compatibility criteria into a file chosen by the user.
- `void jumpToCriterionViolation(Violation violation)`
  Changes the `Position` of the background surface to the `Segment` that causes the given `Violation`.
- `void subscribeToPlausibilityIOAction(Runnable onBegin, Runnable onDone )`
  Registers a `Runnable` that gets called before the `Controller` executes an import or an export and a runnable that runs when it is done.
- `void unsubscribeFromPlausibilityIOAction(Runnable onBegin, Runnable onDone )`
  Unregisters a `Runnable` that gets called before the `Controller` executes an import or an export and a runnable that runs when it is done.

3.18.2 Class RosePlausibilityController

Provides the functionality to manage the plausibility criteria of the `Roadsystem`.

Declaration

- `public class RosePlausibilityController extends Controller implements PlausibilityController`

Attributes

- none

Constructors

- `public RosePlausibilityController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, Project project, ApplicationDataSystem applicationDataSyst )`
  Creates a new `RosePlausibilityController`. Requires `changeCommandBuffer`, the buffer for change commands. Requires `storageLock`, the coordinator for `Controller` actions. Requires `project`, the model facade for project data. Requires `applicationDataSystem`, the model facade for application data.

Methods

- none

3.19 Package edu.kit.rose.controller.project

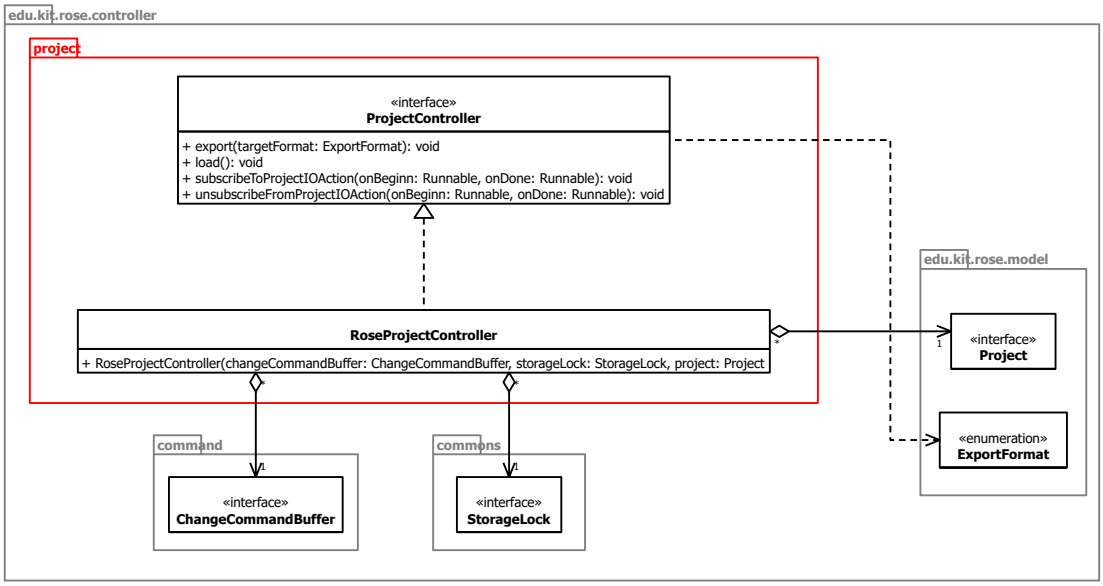

Abbildung 25: Class diagram for package edu.kit.rose.controller.project

This package contains the project controller.

3.19.1 Interface ProjectController

Provides functionality to save, load and export a project.

Declaration

- `public interface ProjectController`

Methods

- `void export(ExportFormat targetFormat)`
  Exports the `Roadsystem` into a specified target format.
- `void save()`
  Saves the `Project` to a file.
- `void subscribeToProjectIOAction(Runnable onBegin, Runnable onDone)`
  Registers a `Runnable` that gets called before the `Controller` executes a loading, saving or an export and a `Runnable` that runs when it is done.
- `void unsubscribeFromProjectIOAction(Runnable onBegin, Runnable onDone)`

Unregisters a `Runnable` that gets called before the `Controller` executes a loading, saving or an export and a `Runnable` that runs when it is done.

3.19.2 Class RoseProjectController

Provides functionality to save, load and export a project.

Declaration

- `public class RoseProjectController extends Controller implements ProjectController`

Attributes

- none

Constructors

- `protected RoseProjectController(ChangeCommandBuffer changeCommandBuffer , StorageLock storageLock, Project project)`
  Creates a new `RoseProjectController`. Requires `changeCommandBuffer`, the buffer for change commands. Requires `storageLock`, the coordinator for `Controller` actions. Requires `project`, the model facade for project data.

Methods

- none

3.20 Package edu.kit.rose.controller.roadsystem

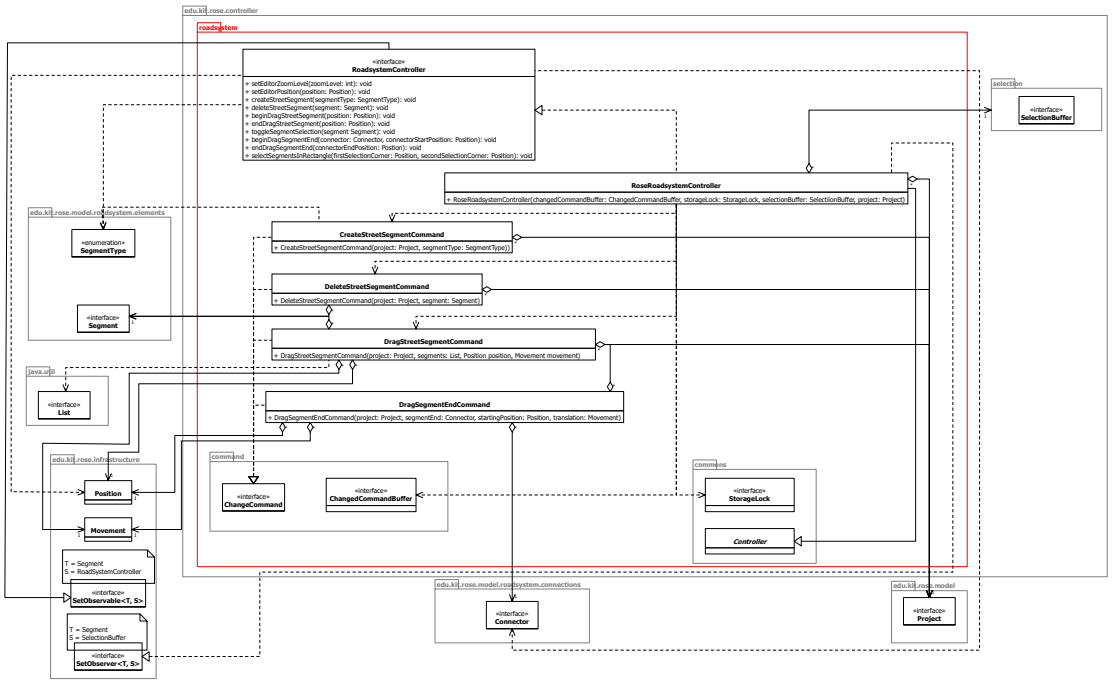

Abbildung 26: Class diagram for package edu.kit.rose.controller.roadsystem

This package contains the roadsystem controller and associated change commands.

3.20.1 Interface RoadSystemController

Provides the functionality for managing `Roadsystem`s and the `Segment`s.

Declaration

- `public interface RoadSystemController extends SetObservable<Segment, Roadsystem >`

Methods

- `void setZoomLevel(int zoomLevel)`
  Sets the zoomLevel of the road system editor.
- `void setEditorPosition(Position position)`
  Sets the `Position` of the `Roadsystem` editor on the background surface.
- `void createStreetSegment(SegmentType segmentType)`
  Creates a new `Segment` of a given `SegmentType`.

- `void deleteStreetSegment(Segment segment)`
  Deletes an existing street `Segment`.
- `void beginDragStreetSegment(Position segmentPosition)`
  Stores the starting `Position` of the `Segement` which will be dragged.
- `void endDragStreetSegment(Position segmentPosition)`
  Stores the end `Position` of the `Segment` which was dragged.
- `void toggleSegmentSelection(Segment segment)`
  Toggles the selection status of a `Segment`.
- `void selectSegmentsInRectangle(Position firstSelectionCorner, Position secondSelectionCorner)`
  Selects all `Segments` that have a `Connector` inside the rectangle which the given `Positions` determine. The `Positions` have to be located diagonally to each other on the rectangle.
- `void beginDragSegmentEnd(Connector connector, Position connectorStartPosition)`
  Begins a drag action for an end of a `Segment`. The `Controller` remembers the `Connector` of the last method call, until endDragStreetSegment is called.
- `void endDragSegmentEnd(Position connectorEndPosition)`
  Ends a drag action for an end of a `Segment`. Instructs the model to execute the dragging. If beginDragSegmentEnd has not been called, this method does nothing.

3.20.2 Class RoseRoadSystemController

Provides the functionality for managing `Roadsystems` and their `Segments`.

Declaration

- `public class RoseRoadSystemController extends Controller implements RoadSystemControl`

Attributes

- none

Constructors

- `public RoseRoadSystemController(ChangeCommandBuffer changeCommandBuffer, StorageLock storageLock, SelectionBuffer selectionBuffer, Project project)`
  Creates a new `RoseRoadSystemController`.Requires `changeCommandBuffer`, the buffer for change commands. Requires `storageLock`, the coordinator for controller actions. Requires `selectionBuffer`, the container that stores selected segments. Requires `project`, the model facade for project data.

Methods

- none

3.20.3 Class CreateStreetSegmentCommand

Encapsulates the functionality of creating a `Segment` and makes it changeable.

Declaration

- `public class CreateStreetSegmentCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public CreateStreetSegmentCommand(Project project, SegmentType segmentType )`
  Creates a new `CreateStreetSegmentCommand` that creates a new `Segment` of a given `SegmentType`. Requires `project`, the model facade to execute `CreateStreetSegmentCommand` on. Requires `SegmentType`, the type of the `Segment` to create.
- none

Methods

- none

3.20.4 Class DeleteStreetSegmentCommand

Encapsulates the functionality of deleting a `Segment` and makes it changeable.

Declaration

- `public class DeleteStreetSegmentCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public DeleteStreetSegmentCommand(Project project, Segment segment)`
  Creates a `DeleteStreetSegmentCommand` that deletes a `Segment`. Requires `project` , the model facade to execute `DeleteStreetSegmentCommand` on. Requires `segment`, the `Segment` to delete.

Methods

- none

3.20.5 Class DragSegmentEndCommand

Encapsulates the functionality of a `Segment` end dragging and makes it changeable.

Declaration

- `public class DragSegmentEndCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public DragSegmentEndCommand(Project project, Connector segmentEnd, Position startingPosition, Movement translation)`
  Creates a `DragSegmentEndCommand` that drags a given end of a `Segment` by a specified movement. Requires `project`, the model facade to execute `DragStreetSegmentCommand` on. Requires `segmentEnd` the `Segment` end to drag. Requires `startingPosition`, the starting `Position` of the `Segment` end. Requires `translation`, the `Movement` of the `Segment` end

Methods

- none

3.20.6 Class DragStreetSegmentCommand

Encapsulates the functionality of a `Segment` dragging and makes it changeable.

Declaration

- `public class DragStreetSegmentCommand implements ChangeCommand`

Attributes

- none

Constructors

- `public DragStreetSegmentCommand(Project project, List<Segment> segments , Position startingPosition, Movement movement)`
  Creates a `DragStreetSegmentCommand` that drags a given set of `Segment`s by a specified `Movement`. Requires `project`, the model facade to execute `DragStreetSegmentCommand` on. Requires segments, the segments to drag. Requires `startingPosition`, the starting `Position` of the `Segment`s. Requires `movement`, the translation of the `Segment`s

Methods

- none

3.21 Package edu.kit.rose.controller.selection

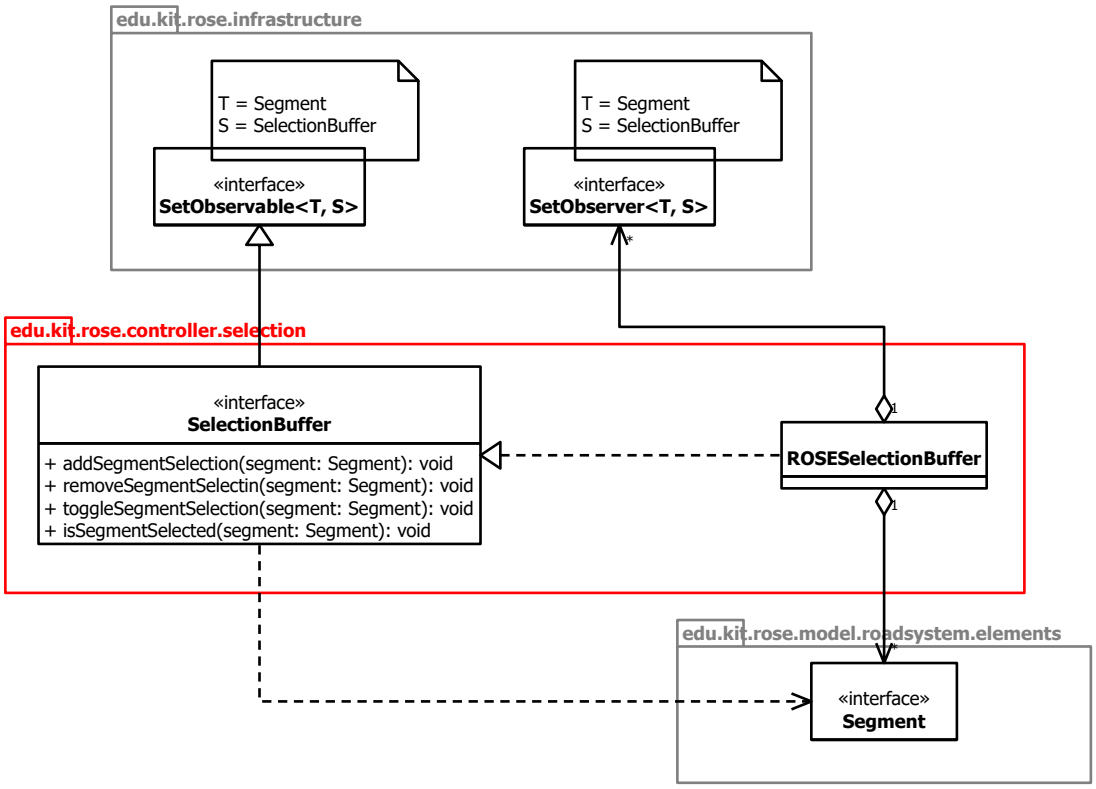

Abbildung 27: Class diagram for package edu.kit.rose.controller.selection

This package contains the selection buffer.

3.21.1 Interface SelectionBuffer

A `SelectionBuffer` is a container that stores selected `Segments` and notifies its `Observer` s when the selection state of one of the `Segments` changes. `Segments` that are stored in the `Selection` are implicitly assumed to be selected.

Declaration

- `public interface SelectionBuffer extends SetObservable<Segment, SelectionBuffer >`

Methods

- `void addSegmentSelection(Segment segment)`
  Adds a `Segment` to the `SelectionBuffer` and marks it as selected.
- `void removeSegmentSelection(Segment segment)`
  Removes a `Segment` from the `SelectionBuffer` and marks it as unselected.
- `void toggleSegmentSelection(Segment segment)`
  Toggle the selection status of a given `Segment`.
- `void isSegmentSelectino(Segment segment)`
  Determines if the given `Segment` is selected/in the `SelectionBuffer`.

3.21.2 Class RoseSelectionBuffer

A `RoseSelectionBuffer` is a container that stores selected `Segments` and notifies its `Observers` when the selection state of one of the `Segments` changes. `Segments` that are stored in the `Selection` are implicitly assumed to be selected.

Declaration

- `public class ROSESelectionBuffer implements SelectionBuffer`

Attributes

- none

Constructors

- none

Methods

- none

3.22 Package edu.kit.rose.view

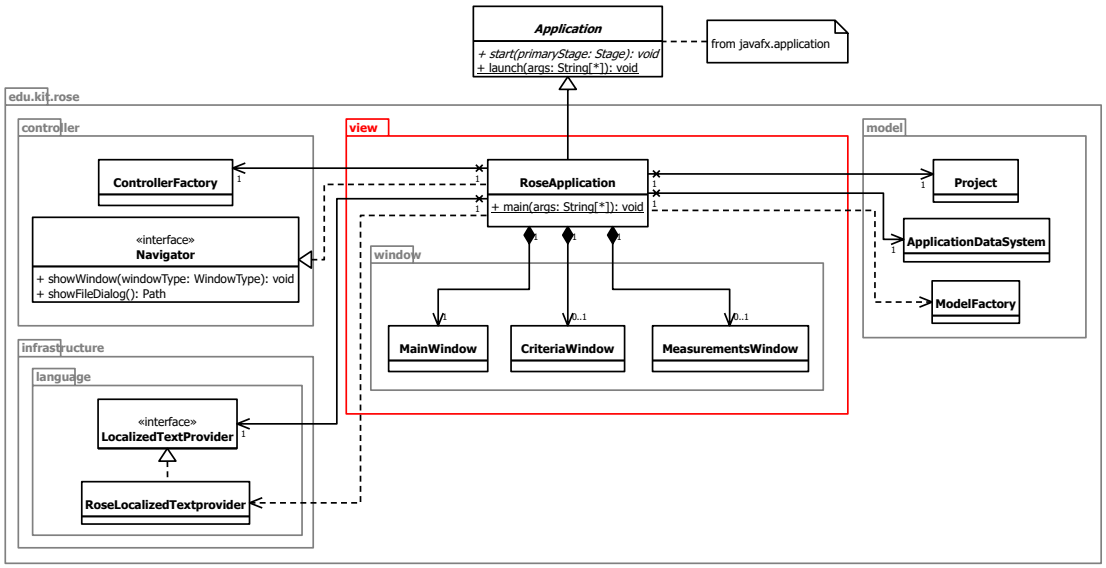

Abbildung 28: Class diagram for package edu.kit.rose.view

This package contains the view subsystem of ROSE. The `edu.kit.rose.view` package contains the `RoseApplication` class, which is the entry point to the entire application. Nested within this package are the `window` package (containing editor window components), the `panel` package (containing panel components) and the `commons` package (containing components shared between different panels). This package depends on the `edu.kit.rose.model`, `edu.kit.rose.controller` and the nested `window` package.

3.22.1 Class RoseApplication

This class is the entry point to the ROSE application. It is responsible for launching the JavaFX GUI framework with the window classes from the `window` sub-package, as well as initializing and connecting the `edu.kit.rose.controller` and `edu.kit.rose.model` packages.

Declaration

- `public class RoseApplication extends Application implements Navigator`

Attributes

- none

Constructors

- `public RoseApplication()`
  Instantiates a new RoseApplication object. This constructor is used by the JavaFX framework and should never be called manually.

Methods

- `public void start(Stage primaryStage)throws Exception`
  Entry point template method for the JavaFX application, inherited from Application.
- `public static void main(String[] args)`
  Entry point method that will be run when this Java application is started.

3.23 Package edu.kit.rose.view.commons

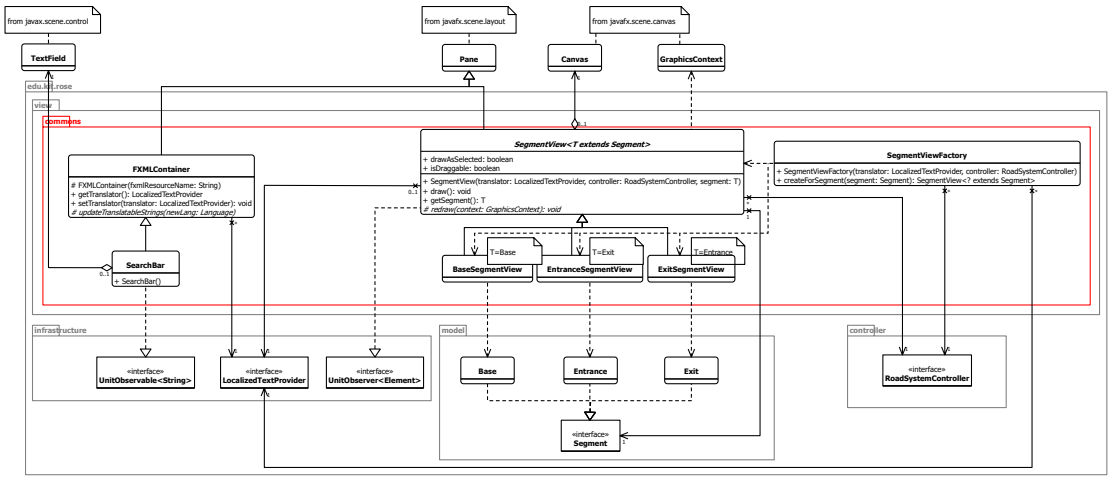

Abbildung 29: Class diagram for package edu.kit.rose.view.commons

This package contains view components that are shared between multiple panels.

3.23.1 Class BaseSegmentView

A base segment view is the visual representation of a base street segment. It allows dragging its ends independently of each other.

Declaration

- `class BaseSegmentView extends SegmentView<Base>`

Attributes

- none

Constructors

- `BaseSegmentView(Base segment, RoadSystemController controller, LocalizedTextProvider translator)`
  Creates a new base segment view for a given base `segment`.

Methods

- none

3.23.2 Class EntranceSegmentView

An entrance segment view is the visual representation of an entrance street segment.

Declaration

- `public class EntraceSegmentView extends SegmentView<Entrance>`

Attributes

- none

Constructors

- `EntranceSegmentView(Entrance segment, RoadSystemController controller, LocalizedTextProvider translator)`
  Creates a new entrance segment view for a given entrance `segment`.

Methods

- none

3.23.3 Class FXMLContainer

FXML containers mount components specified in an FXML file provided by the subclass into themselves.

Declaration

- `public abstract class FXMLContainer extends Pane`

Attributes

- none

Constructors

- `public FXMLContainer(String fxmlResourceName)`
  Creates a new FXMLPanel and immediately mounts the components specified in the given FXML resource (`fxmlResourceName`).

Methods

- `public LocalizedTextProvider getTranslator()`
  Returns the data source for string translation.

- `public void setTranslator(LocalizedTextProvider translator)`
  Sets the data source for string translation.
- `protected abstract void updateTranslatableStrings(Language newLang)`
  Template method that updates all visible strings in this container to the new translation.

3.23.4 Class SearchBar

A search bar is a text input component whose input is used to filter the content of another component.

Declaration

- `public class SearchBar extends FXMLContainer implements UnitObservable <String>`

Attributes

- none

Constructors

- `public SearchBar()`
  Creates a new search bar.

Methods

- none

3.23.5 Class SegmentView

The `SegmentView` is the base class for the visual representation of a street segment. It is responsible for drawing itself and receiving drag and drop operations.

Declaration

- `public abstract class SegmentView<T extends Segment> extends Pane implements UnitObserver<Element>`

Attributes

- none

Constructors

- `public SegmentView(T segment)`
  Creates a new segment view for a given `segment`.

Methods

- `public T getSegment()`
  Returns the segment that is represented by the segment view.
- `public void draw()`
  Draws the segment.
- `public void setDrawAsSelected(boolean drawAsSelected)`
  Sets if the segment view is drawn with a selection indicator.
- `public boolean getDrawAsSelected()`
  Returns whether the segment view is drawn with a selection indicator.
- `public boolean getIsDraggable()`
  Returns whether the segment view can drag itself.
- `public void setIsDraggable(boolean isDraggable)`
  Sets whether the segment view can drag itself.
- `protected LocalizedTextProvider getTranslator()`
  Returns the localized text provider instance of the segment view.
- `protected abstract void redraw(GraphicsContext context)`
  Draws the segment on a given graphical context.

3.23.6 Class SegmentViewFactory

The segment view factory can create a `SegmentView` for a given `Segment`.

Declaration

- `public class SegmentViewFactory`

Attributes

- none

Constructors

- `public SegmentViewFactory(LocalizedTextProvider translator, RoadSystemController controller)`
  Creates a new segment view factory.

Methods

- `public SegmentView<? extends Segment> createForSegment(Segment segment)`
  Creates a segment view for a given segment.

3.24 Package edu.kit.rose.view.window

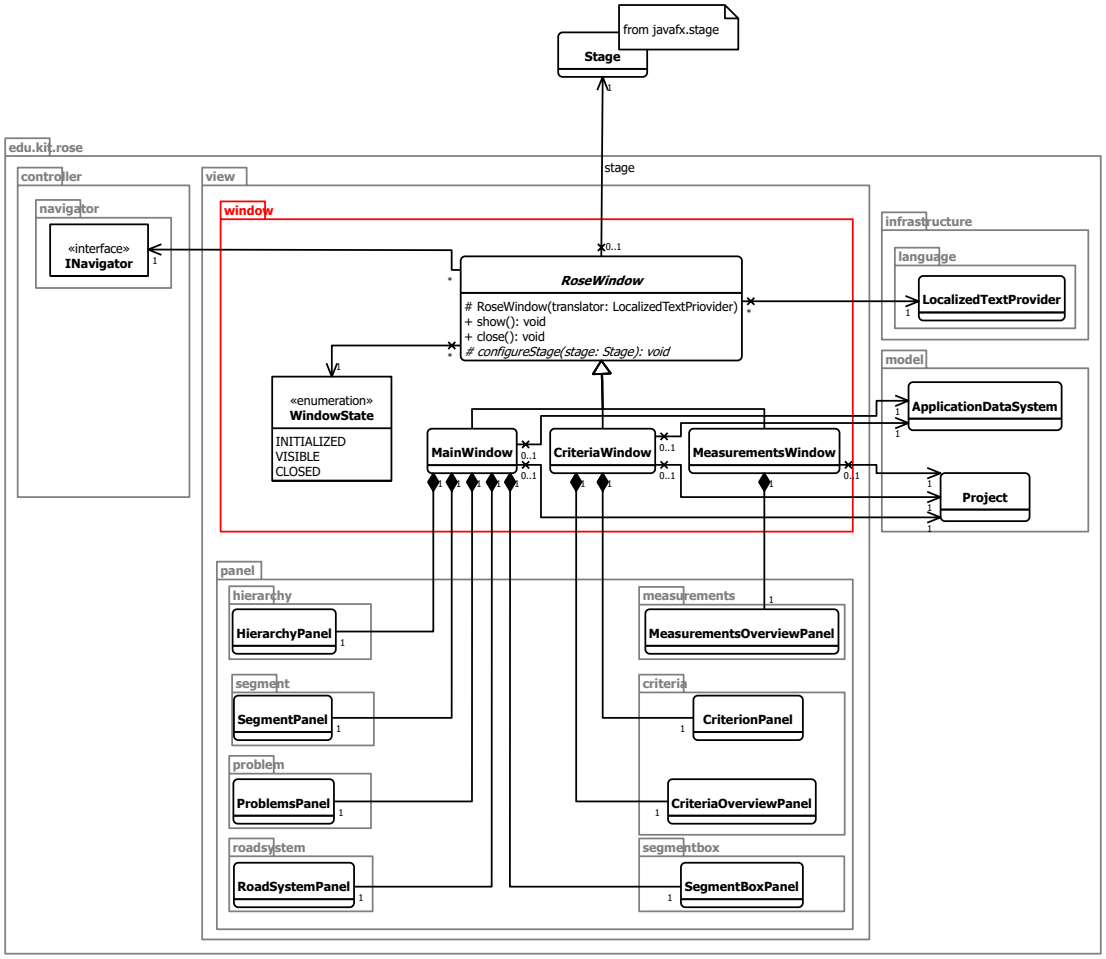

Abbildung 30: Class diagram for package edu.kit.rose.view.window

This package contains the classes responsible for setting up JavaFX windows. Each window in the ROSE application is an implementation of the abstract `RoseWindow` class. Windows usually lay out a set of panels from the `edu.kit.rose.view.panel` package.

3.24.1 Class CriteriaWindow

The criterion window allows the user to manage plausibility criteria, as specified in PF11.1.3. This class is responsible for laying it's contained panels and acts as a mediator between the selection of a criterion in the overview panel and the editor panel.

Declaration

- `public class CriteriaWindow extends RoseWindow`

Attributes

- none

Constructors

- `public CriteriaWindow(LocalizedTextProvider translator, PlausibilityController controller, ApplicationDataSystem applicationData)`
  Creates a new criterion window instance.

Methods

- none

3.24.2 Class MainWindow

This is ROSE's main window, as specified in PF11.1.1. This class lays out the contained panels and sets up the menu bar, as specified in PF11.1.2.

Declaration

- `public class MainWindow extends RoseWindow`

Attributes

- none

Constructors

- `public MainWindow(LocalizedTextProvider translator, ApplicationController applicationController, AttributeController attributeController, HierarchyController hierarchyController, MeasurementController measurementController, PlausibilityContro plausibilityController, RoadSystemController roadSystemController, Project project, ApplicationDataSystem applicationData, Stage stage)`
  Creates a new main window instance with the given primary `stage`

Methods

- none

3.24.3 Class MeasurementsWindow

The measurements window allows the user to look up and edit measurements of all street segments, as specified in PF11.1.4. This class is responsible for laying it's contained panels.

Declaration

- `public class MeasurementsWindow extends RoseWindow`

Attributes

- none

Constructors

- `public MeasurementsWindow(LocalizedTextProvider translator, MeasurementController controller, Project project)`
  Creates a new measurements window instance for a given `project`.

Methods

- none

3.24.4 Class RoseWindow

A ROSE window manages a JavaFX window. ROSE windows usually mount JavaFX panels from the `edu.kit.rose.view.panel` package into their stage.

Declaration

- `public abstract class RoseWindow`

Attributes

- none

Constructors

- `protected RoseWindow(LocalizedTextProvider translator)`
  Creates a new window for the ROSE application.
- `protected RoseWindow(LocalizedTextProvider translator, Stage stage)`
  Creates a new window for the ROSE application, using the given `stage`. This constructor should only be if this window should be set up in the primary stage.

Methods

- `public void show()`
  Makes this window visible and puts it into focus. The window can not be shown if it has already been closed.
- `public void close()`
  Closes this window; is also called when the user closes the window. Closed windows can not be `show()`n again.
- `protected abstract void configureStage(Stage stage)`
  Template method that allows the implementing class to populate the `stage` of this window.
- `public WindowState getState()`
  Returns the state of this window.
- `protected LocalizedTextProvider getTranslator()`
  Returns the data source for string translation.

3.24.5 Enum WindoowState

The window state describes the stages in the life cycle of a window (see 4.3.2).

Declaration

- `public abstract class RoseWindow`

Values

- `INITIALIZED`
  The window is initialized but not visible yet.
- `VISIBLE`
  The window is visible. This does not specify whether the window is in focus or minimized.
- `CLOSED`
  The window has been closed and can not be re-used.

Attributes

- none

Constructors

- none

Methods

- none

3.25 Package edu.kit.rose.view.panel

This package contains one nested package for each panel used within ROSE. The `edu.kit.rose.view.panel` package does not contain public classes itself.

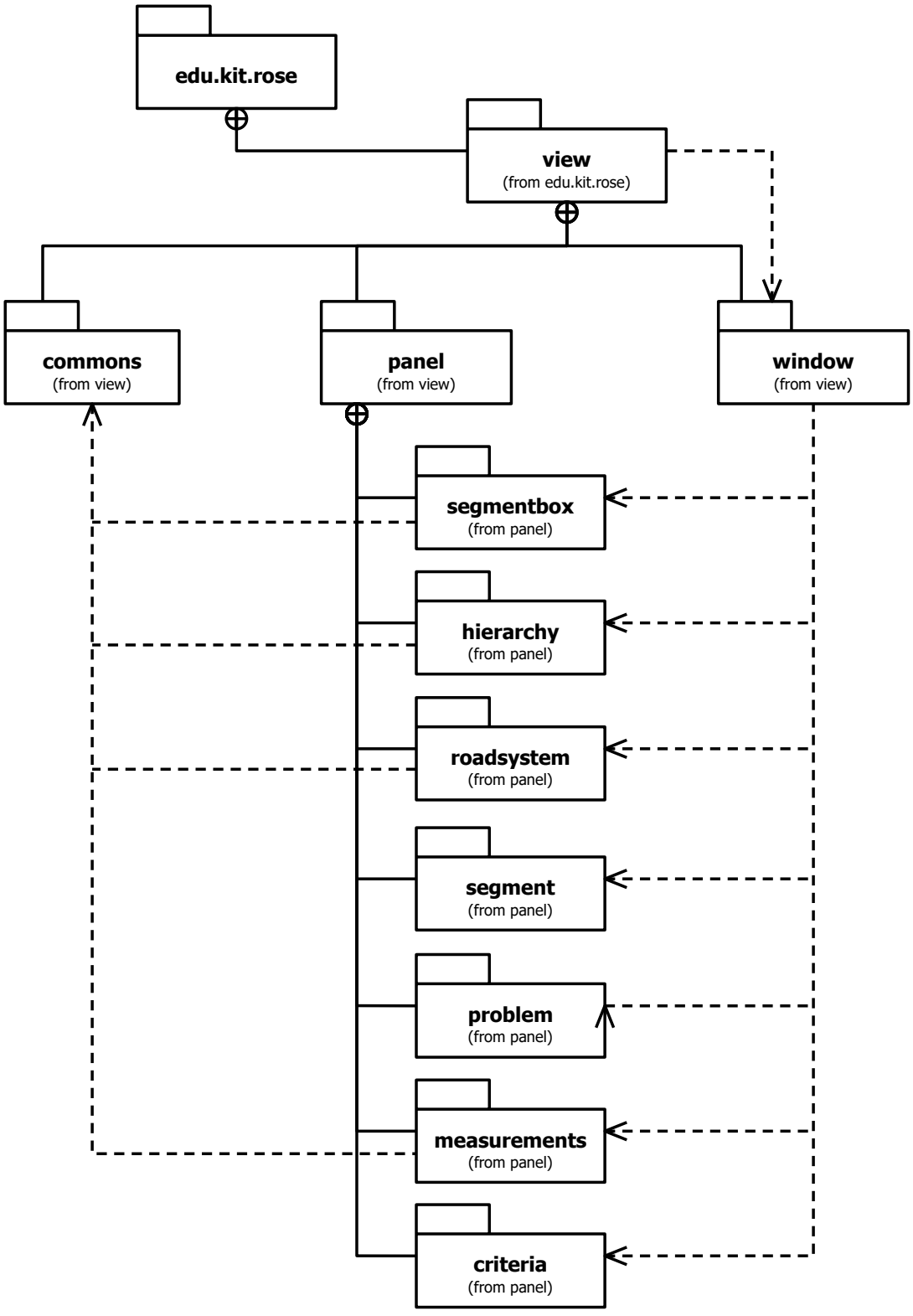

Abbildung 31: Class diagram for package edu.kit.rose.view.panel

3.26 Package edu.kit.rose.view.panel.criterion

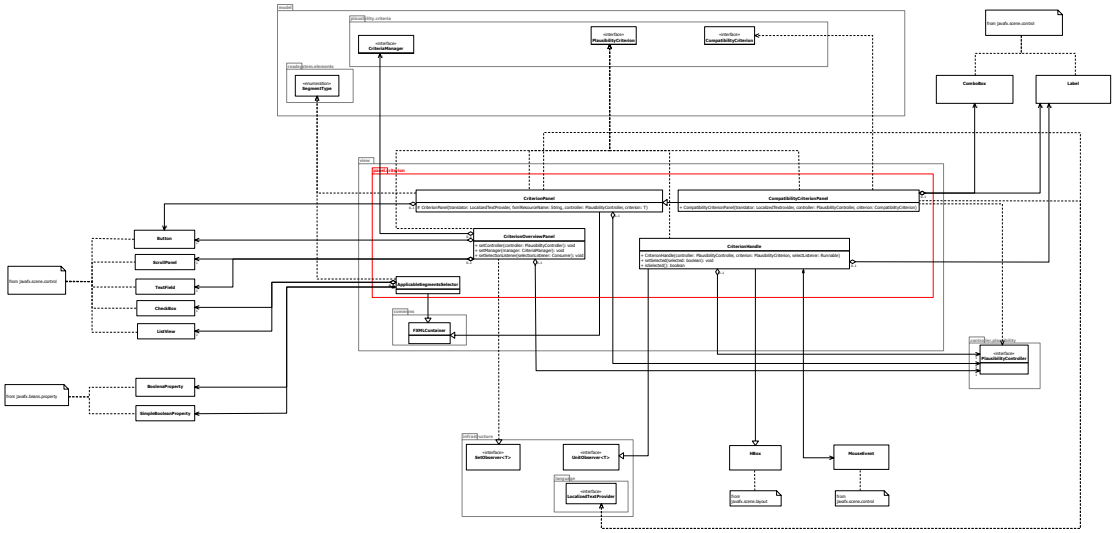

Abbildung 32: Class diagram for package edu.kit.rose.view.panel.criterion

This package provides the classes that are responsible for setting up and view the plausibility criteria.

3.26.1 Class ApplicableSegmentselector

The applicable segments selector allows the user to select which segment types a plausibility criterion applies to.

Declaration

- `class ApplicableSegmentsSelector extends FXMLContainer`

Attributes

- none

Constructors

- `public ApplicableSegmentsSelector((LocalizedTextProvider translator)`
  Requires `translator`, the component that translates displayed text.

Methods

- none

3.26.2 Class CompatibilityCriterionPanel

The compatibility criterion panel allows the user to configure a given compatibility criterion.

Declaration

- `class CompatibilityCriterionPanel extends CriterionPanel<CompatibilityCriteria >`

Attributes

- none

Constructors

- `public CompatibilityCriterionPanel(LocalizedTextProvider translator, PlausibilityCont controller, CompatibilityCriterion criterion)`
  Requires the compatibility criteria that will be configured. Requires the plausibility controller. Requires `LocalizedTextProvider`

Methods

- none

3.26.3 Class CriteriaOverviewPanel

The criteria overview panel allows the user to view the criteria.

Declaration

- `public class CriteriaOverviewPanel extends FXMLContainer implements SetObserver <PlausibilityCriteria>`

Attributes

- none

Constructors

- `public CriteriaOverviewPanel()`
  `#setController(PlausibilityController)` + `#setManager(CriteriaManager)` + `#setTranslator(LocalizedTextProvider)` + `#setSelectionListener(Consumer )` needs to be called!

Methods

- `public void setController(PlausibilityController controller)`
  controller is the PlausibilityController to be set
- `public void setManager(CriteriaManager manager)`
  manager is the CriteriaManager to be set
- `public void setSelectionListener(Consumer<PlausibilityCriterion> selectionListener)`
  selectionListener is the listener that is triggered by selection

3.26.4 Class CriterionHandle

Criterion handles are the entries in the `CriteriaOverviewPanel` that each represent one plausibility criterion.

Declaration

- `class CriterionHandle extends HBox implements UnitObserver<PlausibilityCriterion>`

Attributes

- none

Constructors

- `public CriterionHandle(PlausibilityController controller, PlausibilityCriterion criterion, Runnable selectListener)`
  Requires the controller that links to model. Requires the criteria to be handled. Requires the selectListener that will be called when this handle is clicked.

Methods

- `public void setSelected(boolean selected)`
  selected must be true if the criteria is selected and false otherwise
- `public boolean isSelected()`
  returns true if the criteria is selected and false otherwise

3.26.5 Class CriterionPanel

The criteria panel allows the user to sett up the criteria.

Declaration

- `public abstract class CriterionPanel<T extends PlausibilityCriterion> extends FXMLContainer implements SetObserver<SegmentType, PlausibilityCriterion >`

Attributes

- none

Constructors

- `protected CriterionPanel(LocalizedTextProvider translator, String fxmlResourceName , PlausibilityController controller, T criterion)`
  .Requires the name of fxml file that models this panel. Requires the criteria to be set up.

Methods

- `public T getCriterion()`
  returns the criteria that is being set up
- `public static CriterionPanel<? extends PlausibilityCriteria> forCriterion (ApplicationDataSystem applicationData, PlausibilityCriteria criterion )`
  Constructs a criterion panel for a given criterion, if it is configurable. For non-configurable criteria, null is returned.

3.27 Package edu.kit.rose.view.panel.hierarchy

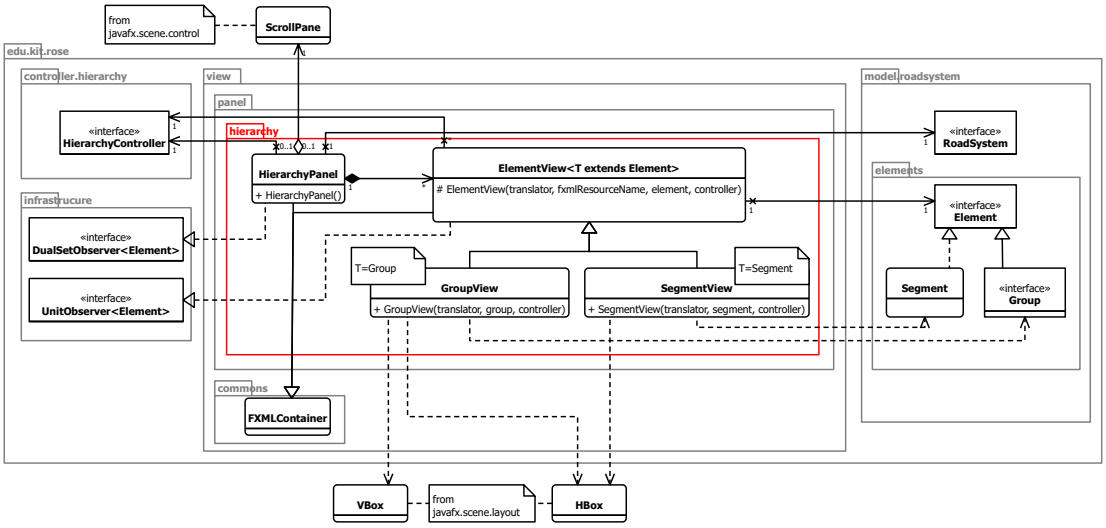

Abbildung 33: Class diagram for package edu.kit.rose.view.panel.hierarchy

This package exposes the `HierarchyPanel` class which allows the user to view and edit the hierarchical composition of the `Elements`.

3.27.1 Class ElementView

An element view represents an element in the hierarchy view and is responsible for showing itself and it's children.

Declaration

- `abstract class ElementView<T extends Element> extends FXMLContainer implements UnitObserver<Element>`

Attributes

- none

Constructors

- `protected ElementView(LocalizedTextProvider translator, String fxmlResourceName, T element, HierarchyController controller)`
  Creates a new element view for a given `element`.

Methods

- `protected HierarchyController getController()`
  Returns the controller of this element view.
- `public T getElement()`
  Returns the element that this element view represents.

3.27.2 Class GroupView

A group view represents a `Group` in the hierarchy view.

Declaration

- `class GroupView extends ElementView<Group>`

Attributes

- none

Constructors

- `GroupView(LocalizedTextProvider translator, Group group, HierarchyController controller)`
  Creates a new group view for a given `group`.

Methods

- none

3.27.3 Class HierarchyPanel

A group view represents a `Group` in the hierarchy panel.

Declaration

- `public class HierarchyPanel extends FXMLContainer implements DualSetObserver <Element, Connection, RoadSystem>`

Attributes

- none

Constructors

- `public HierarchyPanel()`
  Creates an empty hierarchy panel.

Methods

- `public void setController(HierarchyController controller)`
  Sets the controller that handles interactions with this panel.
- `public void setRoadSystem(RoadSystem roadSystem)`
  Sets the road system that this hierarchy panel should display.

3.27.4 Class SegmentView

A segment view represents a `Segment` in the hierarchy panel.

Declaration

- `class SegmentView extends ElementView<Segment>`

Attributes

- none

Constructors

- `SegmentView(LocalizedTextProvider translator, Segment segment, HierarchyController controller)`
  Creates a new segment view for a given `segment`.

Methods

- none

3.28 Package edu.kit.rose.view.panel.measurement

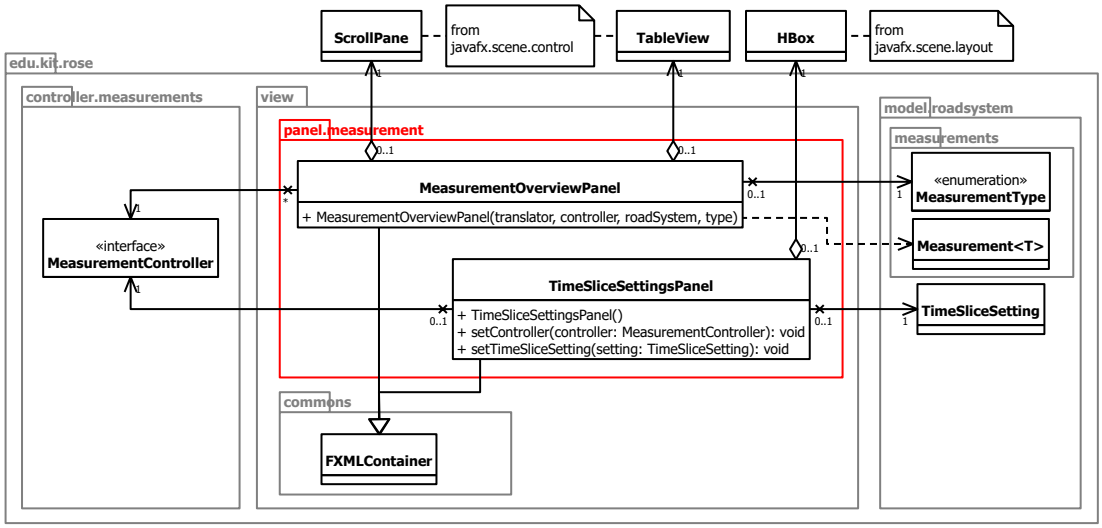

Abbildung 34: Class diagram for package edu.kit.rose.view.panel.measurement

This package exposes the `MeasurementsOverviewPanel` class which provides an overview over all measurements of all segments, as specified in 11.1.4.

3.28.1 Class MeasurementOverviewPanel

The measurement overview panel provides an editable overview over all measurement values of a given type for all segments of the road system.

Declaration

- `public class MeasurementOverviewPanel extends FXMLContainer`

Attributes

- none

Constructors

- `public MeasurementOverviewPanel(LocalizedTextProvider translator, MeasurementControll`
  `controller, RoadSystem roadSystem, MeasurementType type)`
  Creates a new measurement overview panel for the given measurement type.

Methods

- none

3.28.2 Class TimeSliceSettingPanel

The time slice setting panel allows the user to configure the measurement time slice settings, as specified in 11.1.4.

Declaration

- `public class TimeSliceSettingPanel extends FXMLContainer implements UnitObserver <TimeSliceSetting>`

Attributes

- none

Constructors

- `public TimeSliceSettingPanel()`
  Creates a new time slice settings panel.

Methods

- `public void setController(MeasurementController controller)`
  Sets the controller that handles measurement value updates.
- `public void setTimeSliceSetting(TimeSliceSetting timeSliceSetting)`
  Sets the time slice settings instance that is editable through this panel.

3.29 Package edu.kit.rose.view.panel.segment

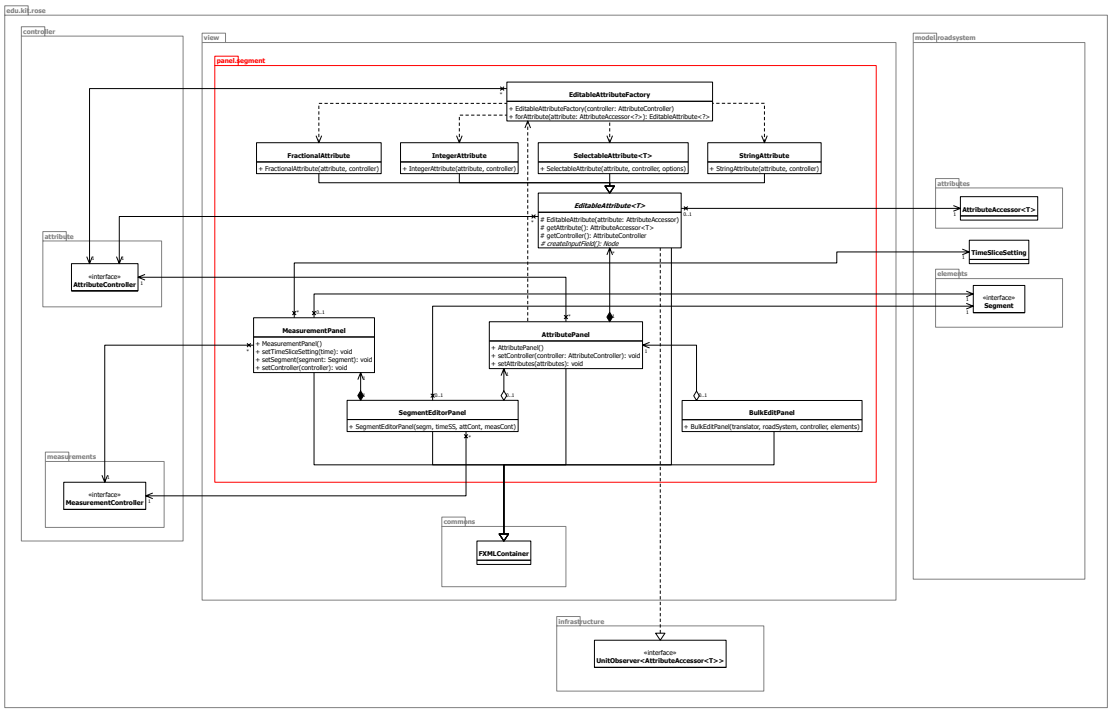

Abbildung 35: Class diagram for package edu.kit.rose.view.panel.segment

This package exposes the `SegmentEditorPanel` class which allows the user to edit a given segment, as specified in PF11.1.5.

3.29.1 Class AttributePanel

An attribute panel allows the user to see and configure attributes.

Declaration

- `public class AttributePanel extends FXMLContainer`

Attributes

- none

Constructors

- `public AttributePanel()`
  Creates an empty attribute panel.

Methods

- `public void setAttributes(SortedBox<AttributeAccessor<?>> attributes)`
  Sets which attributes are shown in this panel.
- `public void setController(AttributeController controller)`
  Sets the controller that handles attribute value updates.

3.29.2 Class BulkEditPanel

The bulk edit panel allows the user to edit attributes of multiple segments simultaneously.

Declaration

- `public class BulkEditPanel extends FXMLContainer`

Attributes

- none

Constructors

- `public BulkEditPanel(LocalizedTextProvider translator, RoadSystem roadSystem, AttributeController controller, Collection<Element> elements)`
  Creates a new bulk edit panel for a given collection of elements.

Methods

- none

3.29.3 Class EditableAttribute

An editable attribute is a JavaFX component that allows the user to see and edit the value of an attribute.

Declaration

- `public abstract class EditableAttribute<T> extends FXMLContainer implements UnitObserver<AttributeAccessor<T>>`

Attributes

- none

Constructors

- `protected EditableAttribute(AttributeAccessor<T> attribute, AttributeController controller)`
  Creates a new editable attribute component.

Methods

- `protected AttributeAccessor<T> getAttribute()`
  Creates an editable attribute component for a given attribute accessor.
- `protected AttributeController getController()`
  Returns the controller that is used for handling attribute value updates.
- `protected abstract Node createInputField()`
  Factory method that delegates the creation of the actual input field to the implementing class.

3.29.4 Class EditableAttributeFactory

The editable attribute factory can generate `EditableAttribute`s from `AttributeAccessor`s.

Declaration

- `public class EditableAttributeFactory`

Attributes

- none

Constructors

- `public EditableAttributeFactory(AttributeController controller)`
  Initializes the factory.

Methods

- `public EditableAttribute<?> forAttribute(AttributeAccessor<?> attribute)`
  Creates an editable attribute component for a given attribute accessor.

3.29.5 Class FractionalAttribute

This is the `EditableAttribute` implementation for the `DataType` `FRACTIONAL`.

Declaration

- `class FractionalAttribute extends EditableAttribute<Double>`

Attributes

- none

Constructors

- `FractionalAttribute(AttributeAccessor<Double> attribute, AttributeController controller)`
  Creates a new fractional attribute editor for the given `attribute`.

Methods

- none

3.29.6 Class IntegerAttribute

This is the `EditableAttribute` implementation for the `DataType INTEGER`.

Declaration

- `class IntegerAttribute extends EditableAttribute<Integer>`

Attributes

- none

Constructors

- `IntegerAttribute(AttributeAccessor<Integer> attribute, AttributeController controller)`
  Creates a new integer attribute editor for the given `attribute`.

Methods

- none

3.29.7 Class MeasurementPanel

A measurement panel allows the user to see and configure time-dependant measurements for a given segment.

Declaration

- `class MeasurementPanel extends FXMLContainer`

Attributes

- none

Constructors

- `MeasurementPanel()`
  Creates an empty measurement panel.

Methods

- `public void setController(MeasurementController controller)`
  Sets the controller that handles measurement value updates.
- `public void setSegment(Segment segment)`
  Sets the segment whose values should be displayed in the measurement panel.
- `public void setTimeSliceSetting(TimeSliceSetting timeSliceSetting)`
  Sets the time slice settings that the measurement should be displayed with.

3.29.8 Class SegmentEditorPanel

The segment editor panel allows the user to configure the attributes and measurements of a given segment, as specified in PF11.1.5.

Declaration

- `public class SegmentEditorPanel extends FXMLContainer implements UnitObserver <Element>`

Attributes

- none

Constructors

- `public SegmentEditorPanel(Segment segment, TimeSliceSetting timeSliceSetting , AttributeController attributeController, MeasurementController measurementControlle )`
  Creates a new segment editor panel for a given segment.

Methods

- none

3.29.9 Class SelectableAttribute

This is the `EditableAttribute` implementation for `DataTypes` whose value needs to be selected out of a small set of options (like enums and booleans).

Declaration

- `class SelectableAttribute<T> extends EditableAttribute<T>`

Attributes

- none

Constructors

- `SelectableAttribute(AttributeAccessor<T> attribute, AttributeController controller, Collection<T> options)`
  Creates a new selectable attribute editor for the given `attribute` with the given `options`.

Methods

- none

3.29.10 Class StringAttribute

This is the `EditableAttribute` implementation for the `DataType STRING`.

Declaration

- `class StringAttribute extends EditableAttribute<String>`

Attributes

- none

Constructors

- `StringAttribute(AttributeAccessor<String> attribute, AttributeController controller)`
  Creates a new string attribute editor for the given `attribute`.

Methods

- none

3.30 Package edu.kit.rose.view.panel.segmentbox

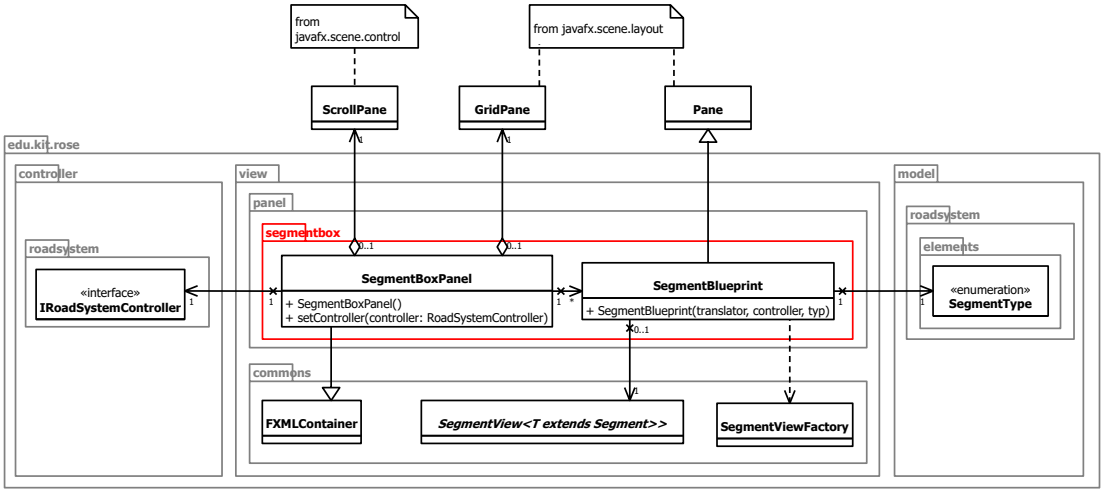

Abbildung 36: Class diagram for package edu.kit.rose.view.panel.segmentbox

This package exposes the `SegmentBoxPanel` class which provides an overview over the available segment types and allows the user to create new segments in the road system.

3.30.1 Class SegmentBlueprint

The segment box panel provides an overview over the available street segment types which can be created, as specified in PF11.1.7.

Declaration

- `class SegmentBlueprint extends Pane`

Attributes

- none

Constructors

- `public SegmentBlueprint(LocalizedTextProvider translator, RoadSystemController controller, SegmentType type)`
  Creates a new segment blueprint for the given type.

Methods

- none

3.30.2 Class SegmentBoxPanel

The segment box panel provides an overview over the available street segment types which can be created, as specified in PF11.1.7.

Declaration

- `public class SegmentBoxPanel extends ScrollPane`

Attributes

- none

Constructors

- `public SegmentBoxPanel()`
  Creates a new segment box panel.

Methods

- `public void setController(RoadSystemController controller)`
  Sets the controller of this panel.

3.31 Package edu.kit.rose.view.panel.problem

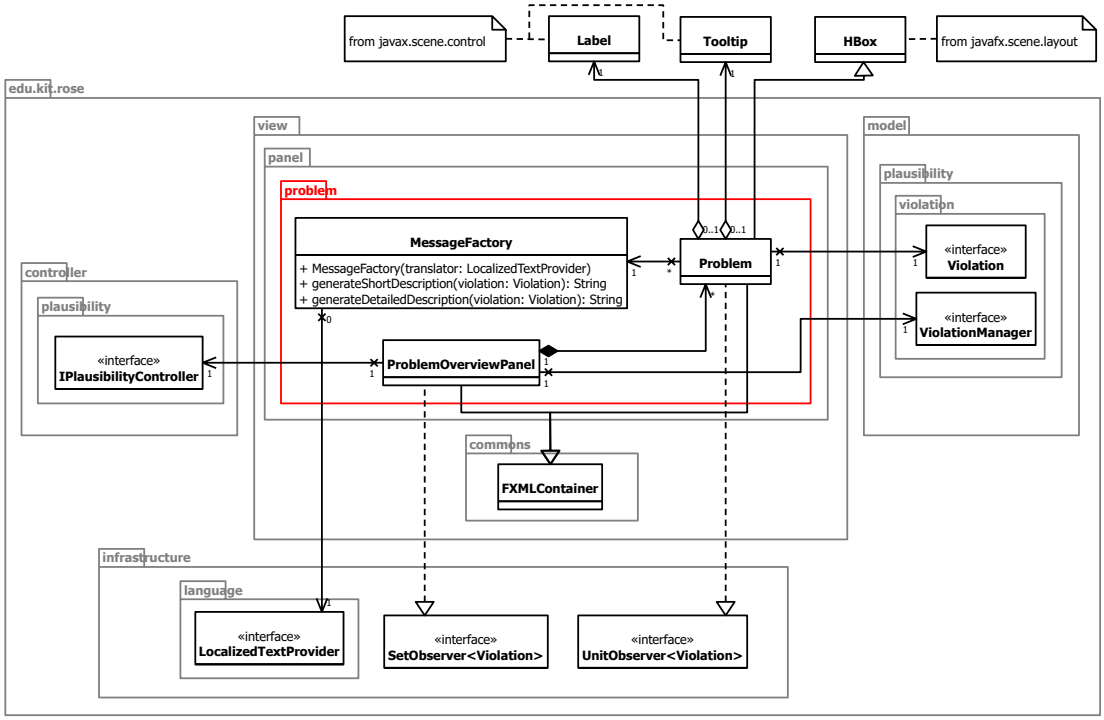

Abbildung 37: Class diagram for package edu.kit.rose.view.panel.problem

This package exposes the `ProblemOverviewPanel` class which provides an overview over all current violations

3.31.1 Class Problem

A problem informs the user about a violation against a plausibility criterion.

Declaration

- `class Problem extends FXMLContainer implements UnitObserver<Violation>`

Attributes

- none

Constructors

- `public Problem(PlausibilityController controller, Violation violation)`

  Creates a new problem for a given violation.

Methods

- none

3.31.2 Class ProblemOverviewPanel

The problem overview panel provides an overview over all current violations of the current road system against any plausibility criteria. This realizes the probelm overview part of specification PF11.1.2.

Declaration

- `public class ProblemOverviewPanel extends FXMLPanel implements SetObserver <Violation, ViolationManager>`

Attributes

- none

Constructors

- `public ProblemOverviewPanel()`
  Creates a new problem overview panel.

Methods

- `public void setController(PlausibilityController controller, ViolationManager manager)`
  Sets the controller that handles plausibility input.
- `public void setManager(ViolationManager manager)`
  Sets the violation manager.

3.31.3 Class MessageFactory

The message factory generates short and detailed descriptions of plausibility criterion violations.

Declaration

- `public class MessageFactory implements Consumer<Language>`

Attributes

- none

Constructors

- `public MessageFactory(LocalizedTextProvider translator)`
  Creates a new message factory with a given translator.

Methods

- `public String generateShortDescription(Violation violation)`
  Generates a short and localized description of the given `violation`, that informs the user about which segment caused a violation against which plausibility criterion.
- `public String generateDetailedDescription(Violation violation)`
  Generates a detailed and localized description of the given `violation`, that informs the user about what exactly is causing it.

3.32 Package edu.kit.rose.view.panel.roadsystem

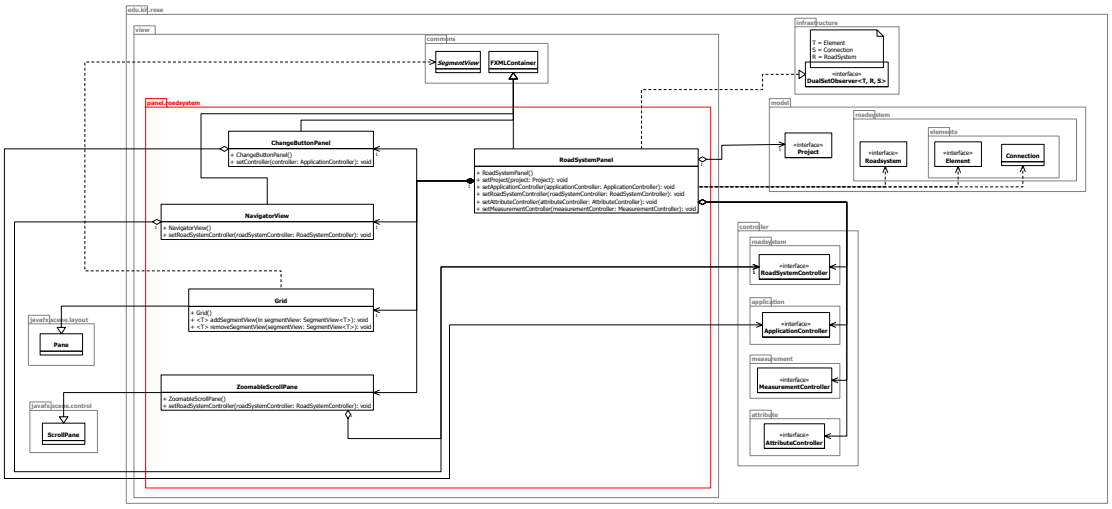

Abbildung 38: Class diagram for package edu.kit.rose.view.panel.roadsystem

This package exposes the `RoadSystemPanel` class, which provides the graphical editing surface for a roadsystem.

3.32.1 Class RoadSystemPanel

The `RoadSystemPanel` is a view component that display a panable and zoomable editor, where `SegmentViews` can be placed, moved and edited.

Declaration

- `public class RoadSystemPanel extends FXMLContainer implements DualSetObserver <Element, Connection, RoadSystem>`

Attributes

- none

Constructors

- `public RoadSystemPanel()`
  Creates a new `RoadSystemPanel`.

Methods

- `public void setProject(Project project)`
  Sets the project.
- `public void setApplicationController(ApplicationController applicationController)`
  Sets the `ApplicationController`.
- `public void setApplicationController(RoadSystemController roadSystemController)`
  Sets the `RoadSystemController`.
- `public void setAttributeController(AttributeController attributeController)`
  Sets the `AttributeController`.
- `public void setMeasurementController(MeasurementController measurementController)`
  Sets the `MeasurementController`.

3.32.2 Class ChangeButtonPanel

The `ChangeButtonPanel` contains the buttons for undoing and redoing a changeable action.

Declaration

- `public class ChangeButtonPanel extends FXMLContainer`

Attributes

- none

Constructors

- `public ChangeButtonPanel()`
  Creates a new `ChangeButtonPane`.

Methods

- `public void setController(ApplicationController)`
  Sets the `ApplicationController`.

3.32.3 Class NavigatorView

The `NavigatorView` is a set of buttons which can be used to navigate on the `RoadSystemPanel` .

Declaration

- `public class NavigatorView extends FXMLContainer`

Attributes

- none

Constructors

- `public NavigatorView()`
  Creates a new `NavigatorView`.

Methods

- `public void setRoadSystemController(RoadSystemController)`
  Sets the `RoadSystemController`.

3.32.4 Class Grid

A `Grid` is a surface that shows a grid, on which `SegmentViews` can be drawn.

Declaration

- `public class Grid extends Pane`

Attributes

- none

Constructors

- `public Grid()`
  Creates a new `Grid`.

Methods

- `public void addSegmentView(SegmentView<? extends Segment> segmentView)`

  Adds a `SegmentView` and displays it on the `Grid`.
- `public void removeSegmentView(SegmentView<? extends Segment> segmentView)`
  Removes a `SegmentView` from the `Grid`.

3.32.5 Class ZoomableScrollPane

The zoomable ScrollPane is a ScrollPane that adds pan and zoom gesture support to its content.

Declaration

- `public class ZoomableScrollPane extends ScrollPane`

Attributes

- none

Constructors

- `public ZoomableScrollPane()`
  Creates a new `ZoomableScrollPane`.

Methods

- `public void setRoadSystemController(RoadSystemController roadSystemController)`
  Sets the `RoadSystemController`.

3.33 Package edu.kit.rose.infrastructure

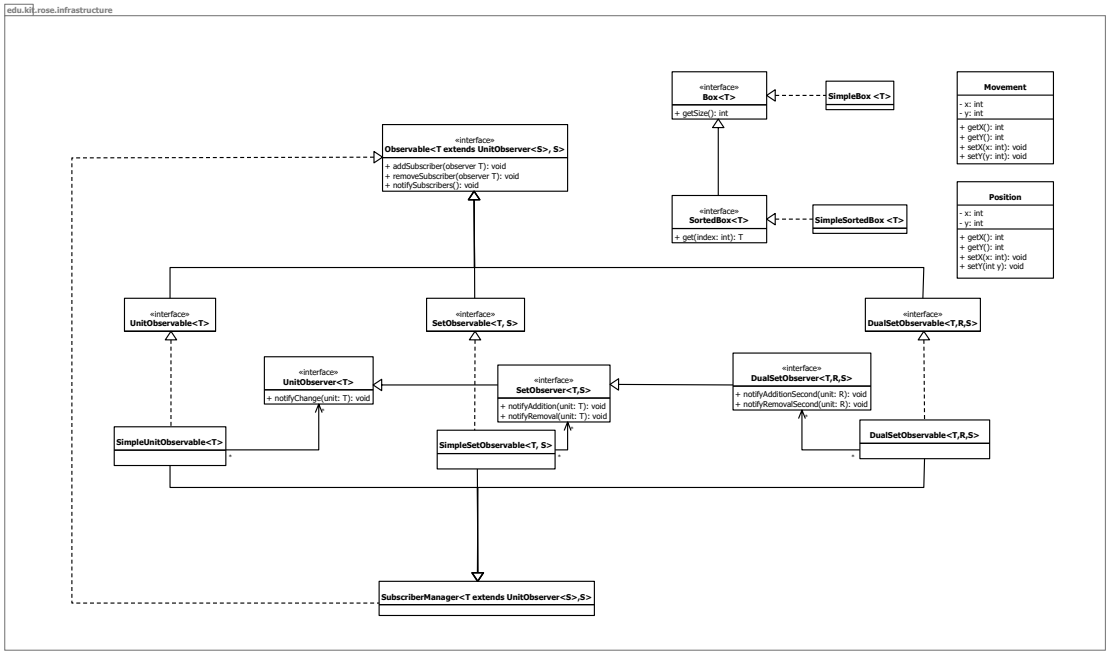

Abbildung 39: Class diagram for package edu.kit.rose.infrastructure

This package contains the application logic that does not belong to domain logic. The whole observer modeling can be found here.

3.33.1 Interface Box

A Box is a read-only container that contains unsorted Objects of the Type T.

Declaration

- `public interface Box<T> extends Iterable<T>`

Methods

- `int getSize()`
  returns an int describing the Number Objects in the Box.

3.33.2 Interface SortedBox

A SortedBox is a Read only container that contains sorted Objects of the Type T.

Declaration

- `public interface SortedBox<T> extends Box<T>`

Methods

- `T get(int index)`
  Provides the Object of type T at the given index in the Box.

3.33.3 Interface Observable

An Observable provides the methods that make it possible to add and remove subscribers as well as to notify them. Subscribers will get notified whenever the Observable undergoes a change. An Observer is either an `UnitObserver` or an `SetObserver`.

Declaration

- `public interface Observable<T extends UnitObserver<S>, S>`

Methods

- `addSubscriber(T observer))`
  Adds a given Observer so that it will be notified by the Observable in case of change to the Observable.
- `void removeSubscriber(T observer)`
  Removes a given Observer so that it will no longer be notified by the Observable.
- `void notifySubscribers()`
  Notifies all Observers. The Observable provides itself as the Parameter in the notify-call of the Observer.

3.33.4 Interface UnitObservable

A UnitObservable notifies its subscribers in case of a change. It provides a method to allow for explicit notification of all subscribers.

Declaration

- `public interface UnitObservable<T> extends Observable<UnitObserver<T>, T>`

Methods

- none

3.33.5 Interface SetObservable

A SetObservable is a Container that notifies its subscribers in case of a change to itself or to the units held within.

Declaration

- `public interface SetObservable<T, S> extends Observable<SetObserver<T, S>,S>`

Methods

- none

3.33.6 Interface DualSetObservable

A DualSetObservable is a Container that notifies its subscribers in case of a change to itself or to the units held within. It holds two types of units.

Declaration

- `public interface DualSetObservable<T,R,S> extends Observable<DualSetObserver <T,R,S>, S>`

Methods

- none

3.33.7 Interface UnitObserver

A UnitObserver is an Observer for a single Object.

Declaration

- `public interface UnitObserver <T>`

Methods

- `void notifyChange(T unit)`
  Called to inform the Observer of changes to an Observable.

3.33.8 Interface SetObserver

A SetObserver observes a set of type S that holds elements of type T.

Declaration

- `public interface SetObserver <T,S> extends UnitObserver<S>`

Methods

- `void notifyAddition(T unit)`
  To be called when a new Object is added to the set.
- `void notifyRemoval(T unit)`
  To be called when an Object is removed from the set.

3.33.9 Interface DualSetObserver

A DualSetObserver observes a set of type S that holds both units of type T and units of type R.

Declaration

- `public interface DualSetObserver<T,R,S> extends SetObserver<T,S>`

Methods

- `void notifyAdditionSecond(R unit)`
  To be called when a new Object is added to the set.
- `void notifyRemovalSecond(R unit)`
  To be called when an Object is removed from the set.

3.33.10 Class SubscriberManager

Default implementation of the observable subscriber management logic.

Declaration

- `abstract class SubscriberManager<T extends UnitObserver<S>,S> implements Observable<T,S>`

Attributes

- none

Constructors

- none

Methods

- none

3.33.11 Class SimpleUnitObservable

A SimpleUnitObservable notifies its subscribers in case of a change. It provides a method to allow for explicit notification of all subscribers.

Declaration

- `public class SimpleUnitObservable<T> extends SubscriberManager<UnitObserver<T>, T> implements UnitObservable<T>`

Attributes

- none

Constructors

- none

Methods

- none

3.33.12 Class SimpleSetObservable

A SimpleSetObservable is a Container that notifies its subscribers in case of a change to itself or to the units held within.

Declaration

- `public class SimpleSetObservable<T,S> extends SubscriberManager<SetObserver <T,S>, S> implements SetObservable<T,S>`

Attributes

- none

Constructors

- none

Methods

- none

3.33.13 Class SimpleDualSetObservable

A SimpleDualSetObservable is a Container that notifies its subscribers in case of a change to itself or to the units held within. It holds two types of units.

Declaration

- `public class SimpleDualSetObservable<T,R,S> extends SubscriberManager< DualSetObserver<T,R,S>, S> implements DualSetObservable<T,R,S>`

Attributes

- none

Constructors

- none

Methods

- none

3.33.14 Class Movement

A movement in form of a 2D Vector.

Declaration

- `public class Movement`

Attributes

- none

Constructors

- none

Methods

- `int getX()`
  Gives the x axis value of the Vector.
- `int getY()`
  Gives the y axis value of the Vector.
- `void setX(int x)`
  Sets the x axis value of the Vector.
- `void setY(int y)`
  Sets the y axis value of the Vector.

3.33.15 Class Position

A Position in a two Dimensional Plane.

Declaration

- `public class Position`

Attributes

- none

Constructors

- none

Methods

- `int getX()`
  Gives the x axis value of the Position.
- `int getY()`
  Gives the y axis value of the Position.
- `void setX(int x)`
  Sets the x axis value of the Position.

- `void setY(int y)`
  Sets the y axis value of the Position.

3.34 Package edu.kit.rose.infrastructure.language

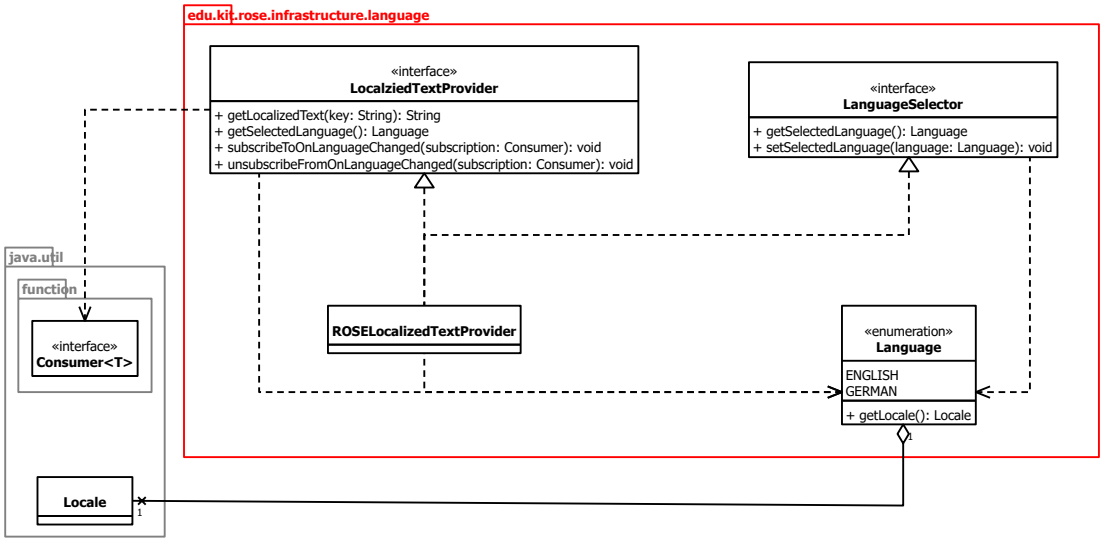

Abbildung 40: Class diagram for package edu.kit.rose.infrastructure.language

This package contains the localization functionality.

3.34.1 Enum Language

Contains all languages supported by ROSE.

Declaration

- `public enum WindowType`

Values

- `ENGLISH`
  References the english language.
- `GERMAN`
  References the german language.

Attributes

- none

Constructors

- `Langauge(Locale locale)`
  Constructor.
  Requires a Locale that wraps the associated iso-language-code.

Methods

- `public Locale getLocale()`
  Returns the java.util.Locale associated with the language.

3.34.2 Interface LanguageSelector

Provides the functionality to get and set the selected language.

Declaration

- `public interface LanguageSelector`

Methods

- `Language getSelectedLanguage()`
  Returns the currently selected language.
- `void setSelectedLanguag(Language language)`
  Sets the currently selected language.

3.34.3 Interface LocalizedTextProvider

Provides the functionality to retrieve localized Strings, get their language and subscribe to get notified when the language changes.

Declaration

- `public interface LocalizedTextProvider`

Methods

- `String getLocalizedText(String key)`
  Returns the localized string/translated text that belongs to a given key.
- `Language getSelectedLanguage()`
  Returns the currently selected language.
- `void subscribeToOnLanguageChanged()`
  Registers a function that will be called when the selected language changes.

- `void unsubscribeFromOnLanguageChanged()`
  Removes the registration of a given function that was registered with the subscribeToOnLanguageChanged method. If the function is not registered, this method does nothing.

3.34.4 Class RoseLocalizedTextProvider

A window of the ROSE application.

Declaration

- `public class RoseLocalizedTextProvider implements LocalizedTextProvider , LanguageSelector`

Attributes

- none

Constructors

- none

Methods

- none

4 Dynamisches Modell

In diesem Abschnitt befindet sich das dynamische Modell des ROSE-Entwurfs. Es umfasst UML-Aktivitätsdiagramme, UML-Sequenzdiagramme und UML-Zustandsdiagramme.

4.1 Aktivitätsdiagramme

4.1.1 Change-Funktionalität beim Ändern von Attributwerten

Abbildung 4.1.1 ist ein Aktivitätsdiagramm, das den Ablauf der Change-Funktionalität, beim Ändern des Wertes eines Segmentattributes, darstellt.

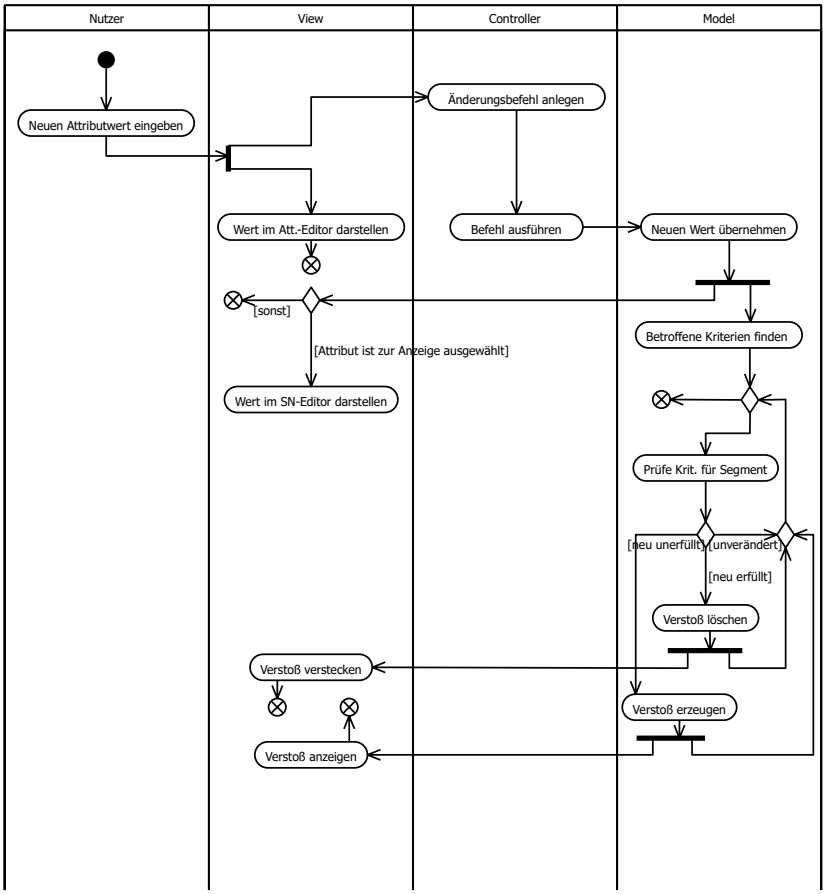

Abbildung 41: A sequence of setting an attribute followed by an undo and redo action

4.2 Sequenzdiagramme

4.2.1 Change-Funktionalität beim Ändern von Attributwerten

Abbildung 4.2.1 zeigt den Ablauf, wie im edu.kit.rose.controller.attribute.AttributeController ein Attribut auf einen neuen Wert gesetzt wird , bzw. diese Aktion durch die Change-Funktionalität rückgängig gemacht und wiederhergestellt wird.

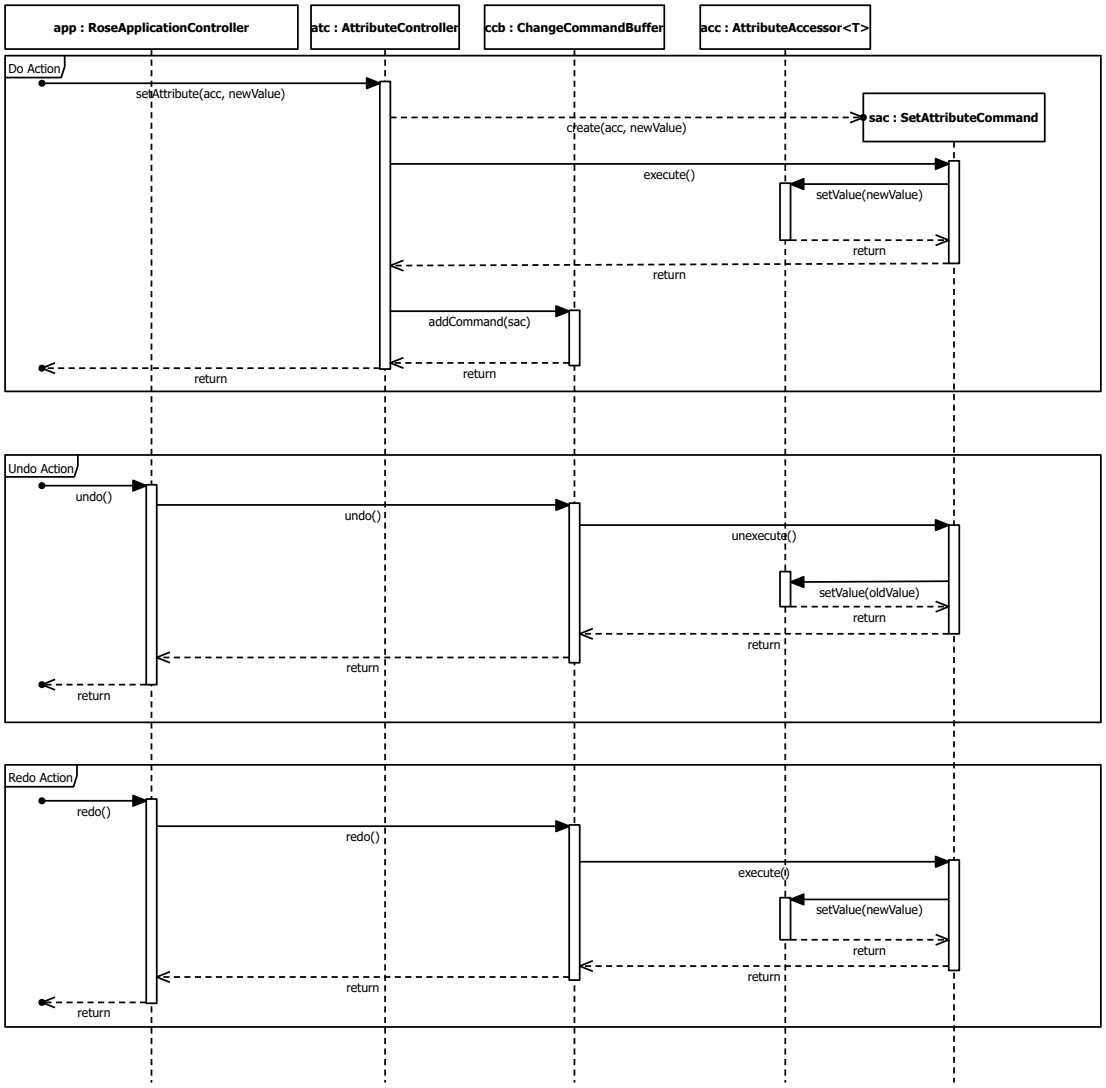

Abbildung 42: A sequence of setting an attribute followed by an undo and redo action

4.2.2 Verbinden von zwei Connectoren

Abbildung 4.2.2 zeigt den Ablauf, wie im Model zwei Connectoren verbunden werden und anschließend sofort durch einem Kompatibilitätskriterium überprüft werden.

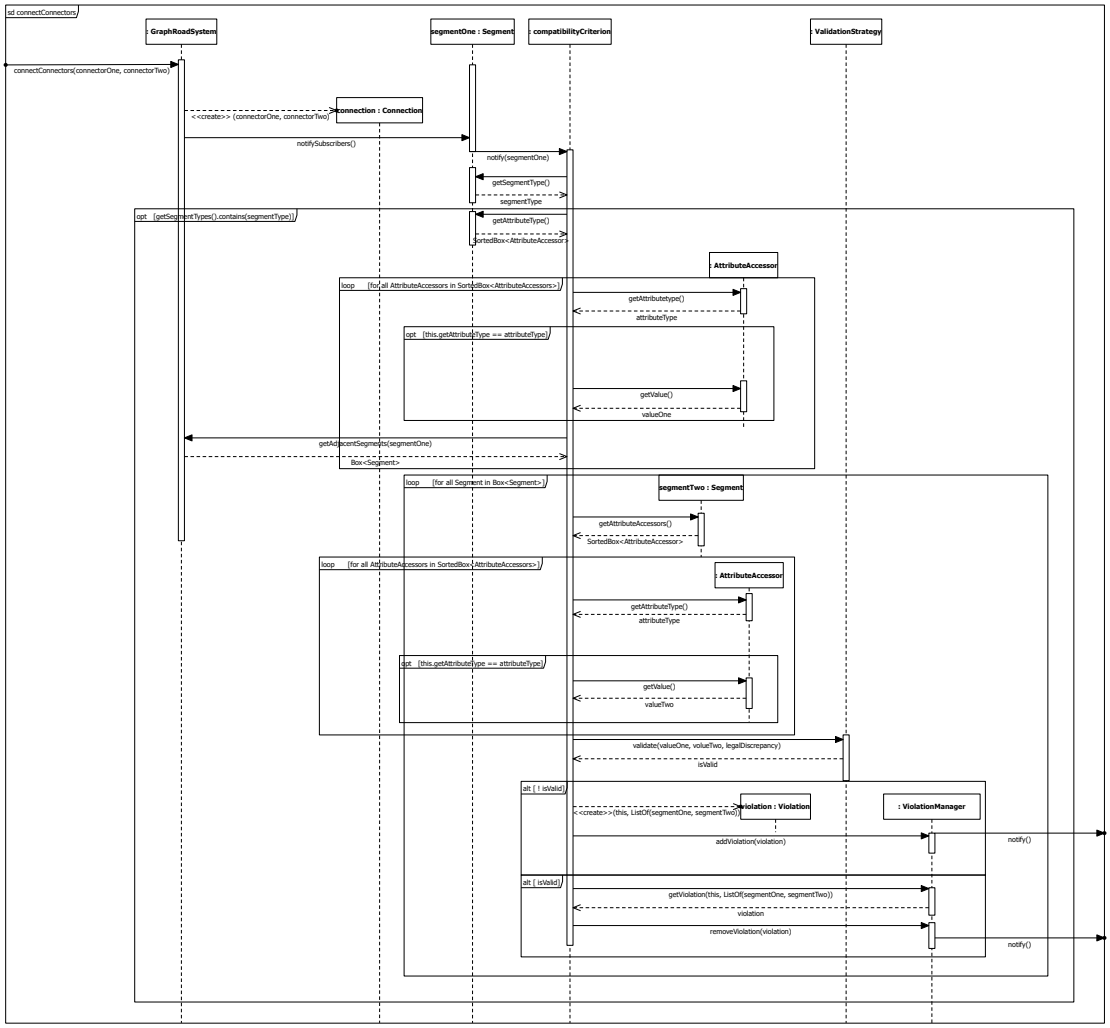

Abbildung 43: A sequence of connecting two Connectors

4.2.3 Segment verschieben

Abbildung 4.2.3 zeigt den Ablauf, wie in der View ein Segment per drag and drop verschoben wird. Die Änderung wird zunächst an den RoadSystemController weitergegeben

und dort in einem Command gespeichert. Anschließend wird die Änderung ins Model eingepflegt und das Model löst eine notification an alle Observer aus. Dadurch wird insbesondere die SegmentView dazu gebracht, sich mit der neuen Position neu zu zeichnen.

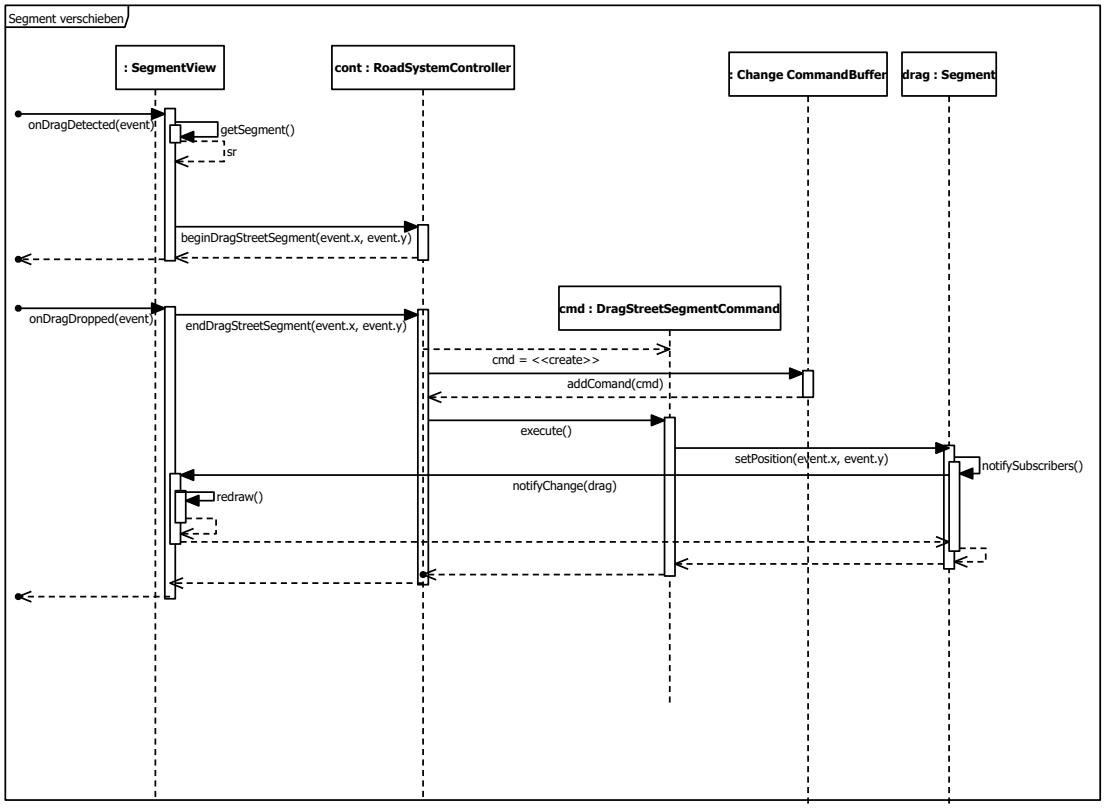

Abbildung 44: A sequence of dragging a segment

4.3 Zustandsdiagramme

4.3.1 Segmentzustand beim Dragging

Abbildung 4.3.1 zeigt den Zustand eines Straßensegments während es mit Dragging verschoben wird.

Abbildung 45: The states of a segment while dragging

4.3.2 Zustand eines RoseWindows

Abbildng 4.3.2 zeigt die möglichen Zustände eines RostWindows während dessen Lebenszyklus. Die Zustände sind linear angeordnet und es ist nicht möglich, zu einem früheren Zustand zurückzuspringen.

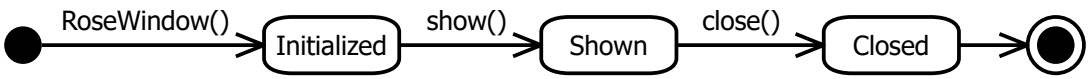

Abbildung 46: The states of a ROSEWindow during its lifecycle

5 Änderungen

Dieses Kapitel enthält die Änderungen an den Spezifikationen des ROSE-Pflichtenheftes, die sich durch den Entwurf ergeben haben.

5.1 Änderung an der Scrollbar der Hierarchieübersicht

- Betrifft:
  - Mockup 11.1.1
  - Mockup 11.1.2
  - Mockup 11.1.8
- Stand Pflichtenheft:

  Die Abbildungen der Hierarchieübersicht in den betroffenen Mockups zeigen, dass die angezeigte Scrollbar auch den Button "Gruppe aus Selektionünd die Searchbar in das Scrolling mit einbezieht.
- Änderung:

  Die angezeigte Scrollbar bezieht nur die Auflistung der Elemente in das Scrolling mit ein. Die Searchbar ist am oberen Rand der Hierarchieübersicht fixiert und der Button ist am unteren Rand der Hierarchieübersicht fixiert.

5.2 Änderung an der Scrollbar des Kriteriumseditors

- Betrifft:
  - Mockup 11.1.3
- Stand Pflichtenheft:

  Die Abbildung der Kriterienübersicht (linke Hälfte) des Kriteriumseditors zeigt, dass die angezeigte Scrollbar die Buttons für ”Alles löschen“, ”Neu“, ”Import“ und ”Export“ mit in das Scrolling einbezieht.

- Änderung:

  Die angezeigte Scrollbar bezieht nur die Auflistung der Kriterien in das Scrolling mit ein. Die Buttons sind am unteren Rand des Kriteriumseditors fixiert.

5.3 Änderung an den Zeitschritteinstellungen in der Messwertübersicht

- Betrifft:
  - Mockup 11.1.4
- Stand Pflichtenheft:

  In der Abbildung der Messwertübersicht sind die Searchbar, sowie die Einstellungen für Zeitschrittanzahl und Zeitschrittlänge unterhalb der Tab-Auswahl für die verschiedenen Messwerte positioniert.

- Änderung:

  Die Searchbar, sowie die Zeitschritteinstellungen sind oberhalb der Tab-Auswahl für die verschiedenen Messwerte positioniert.

5.4 Streichung des Tutorials

- Betrifft:
  - FA6030W
  - 11.3
- Stand Pflichtenheft:

  Das Programmführt den Nutzer durch eine interaktive Anleitung in ihre Hauptfunktionalität ein. (“Tutorial”, siehe §11.3)

- Änderung:

  Das Tutorial wird nicht umgesetzt.

Glossar

Change-Funktionalität Change-Funktionalität bezeichnet die Funktionalität der Anwendung einzelne Nutzeraktionen rückgängig machen zu können und wiederherstellen zu können. Alternativ bezeichnet man diese Funktionalität als Undo/Redo. . 11, 146, 147

Domänenlogik Die Logik der nicht softwaretechnischen Anwendungsdomäne. Im Fall von "ROSE" ist die Domänenlogik die Logik der Domäne Verkehrswesen.. 9, 10, 14