Design-Dokument – Softwaretechnik Praktikumsprojekt "Verkehrsplaner" – Gruppe 03

Simon Franzen, Sven Hesse, Markus Tacker, Ramona Vehabovic

Version 1.0 vom 21.12.2010

Hochschule RheinMain
Wintersemester 2010

Inhaltsverzeichnis

Versionen

Version	Datum	Autor(en)	Änderungen
0.1	06.12.2010	Markus	Dokument in Trac angelegt, XML-Abschnitt dazu
0.2	15.12.2010	Sven	Unterpunkte hinzugefügt (Allgemeines, Systemstruktur), Gliederung, Todos als Kommentare
0.3	20.12.2010	Markus, Ramona, Simon, Sven	Methodenbeschreibungen als Javadoc, übersichtliches Klassendiagramm, Sequenzdiagramme, weitere Todos
0.4	21.12.2010	Sven	Texte zu den Use-Case-Realsierungen
0.5	21.12.2010	Markus	Umsetzung des Planers dazu, Dokument umstrukturiert
1.0	21.12.2010	Markus, Ramona, Sven	kleine Fehlerbehebungen

Die aktuelle Version dieses Dokumentes findet sich unter:
https://scm.mi.hs-rm.de/trac/2010swt/2010swt03/wiki/DesignDokument

1. Umsetzung des Planers

In diesem Dokument werden die Vorgaben zur technischen Umsetzung der Anforderungen aus dem PflichtenHeft definiert. Die Umsetzung des Verkehrsplaners erfolgt als Java-Anwendung, die sich in die folgenden logischen Bestandteile gliedert.

1.1 GUI

Die GUI des Planers wird in Swing umgesetzt. Der Aufbau und die einzelnen Elemente der GUI sind dem PflichtenHeft zu entnehmen.

1.2 Anwendung

Die Umsetzung der Anwendungsschicht erfolgt nach Grundlagen des MVC-Patterns und unterteilt sich in vier Schichten:

- GUI
  - Views
- Controller
  - GUI
  - Simulation
- Data
  - Datafactory
  - Models
- Persistenz
  - Datasource
  - XML

In der GUI-Schicht finden sich die einzelnen Views, die die GUI-Elemente enthalten. Die Controller verwenden die DataObjects auf die Models und steuern das Verhalten der GUI-Elemente sowie die Simulation.

In der Data-Schicht findet sich die DataFactory, die die DataObjects aus der Persistenzschicht erzeugt. In der Persistenzschicht findet sich die DataSource, die die Anwendungsdaten aus den XML-Dateien lädt und dort auch wieder speichert.

1.3 XML-Dateien

Die Anwendung verwendet folgende XML-Dateien:

1.3.1 Sprachen

Die Texte der Anwendung werden mit XLIFF übersetzt.

XLIFF is the XML Localization Interchange File Format designed by a group of software providers, localization service providers, and localization tools providers. It is intended to give any software provider a single interchange file format that can be understood by any localization provider. It is loosely based on the OpenTag version 1.2 specification and borrows from the TMX 1.2 specification. However, it is different enough from either one to be its own format.

- xliff-core-1.2-strict.xsd
- Beispiel XML

1.3.2 Kacheln und Fahrzeuge

- tiles.xsd
- Beispiel XML

1.3.3 Welten

- world.xsd
- Beispiel XML

1.4 Grafik-Dateien

Wie im PflichtenHeft festgelegt, werden SVG-Grafiken zur Darstellung der Kacheln, Fahrzeuge und Ampel verwendet.

Folgende Dateien werden benötigt:

- Basiskachel
- Gerade
- Kreuzung
- T-Kreuzung
- Kurve
- Doppelkurve
- Ampel (aus)
- Ampel (rote Phase)
- Ampel (rot-gelbe Phase)
- Ampel (grüne Phase)
- Ampel (gelbe Phase)
- LKW
- Auto
- Motorrad
- Papierkorb

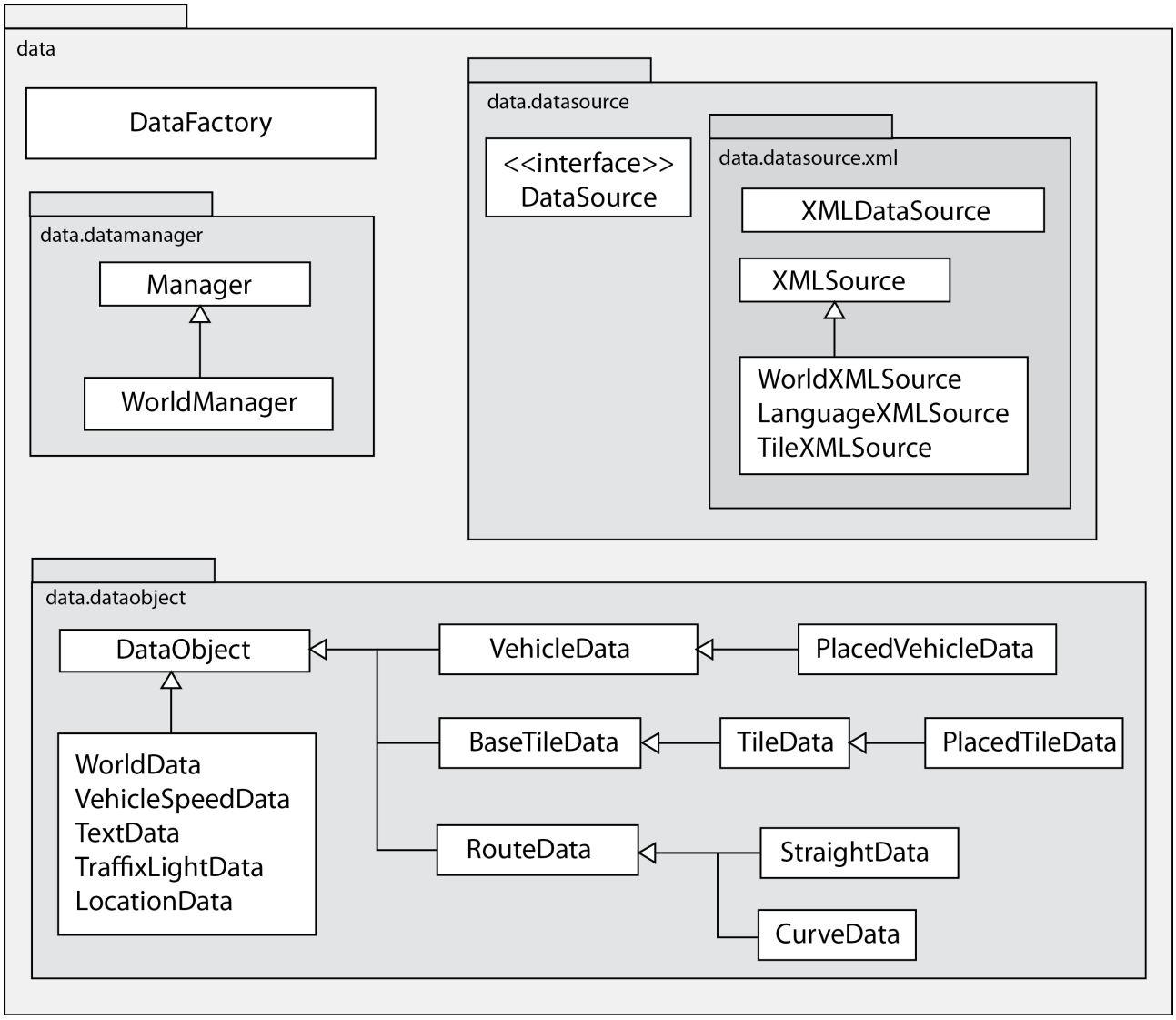
Die Kacheln finden sich unter Material/ in Form von SVG-Grafiken.

Da die Ampeln durch die Anwendung auf die Straßenkacheln gezeichnet werden, ist keine separate Ampel-Version jeder Straße nötig. Siehe hierzu auch JavaDoc in WorldTileView.

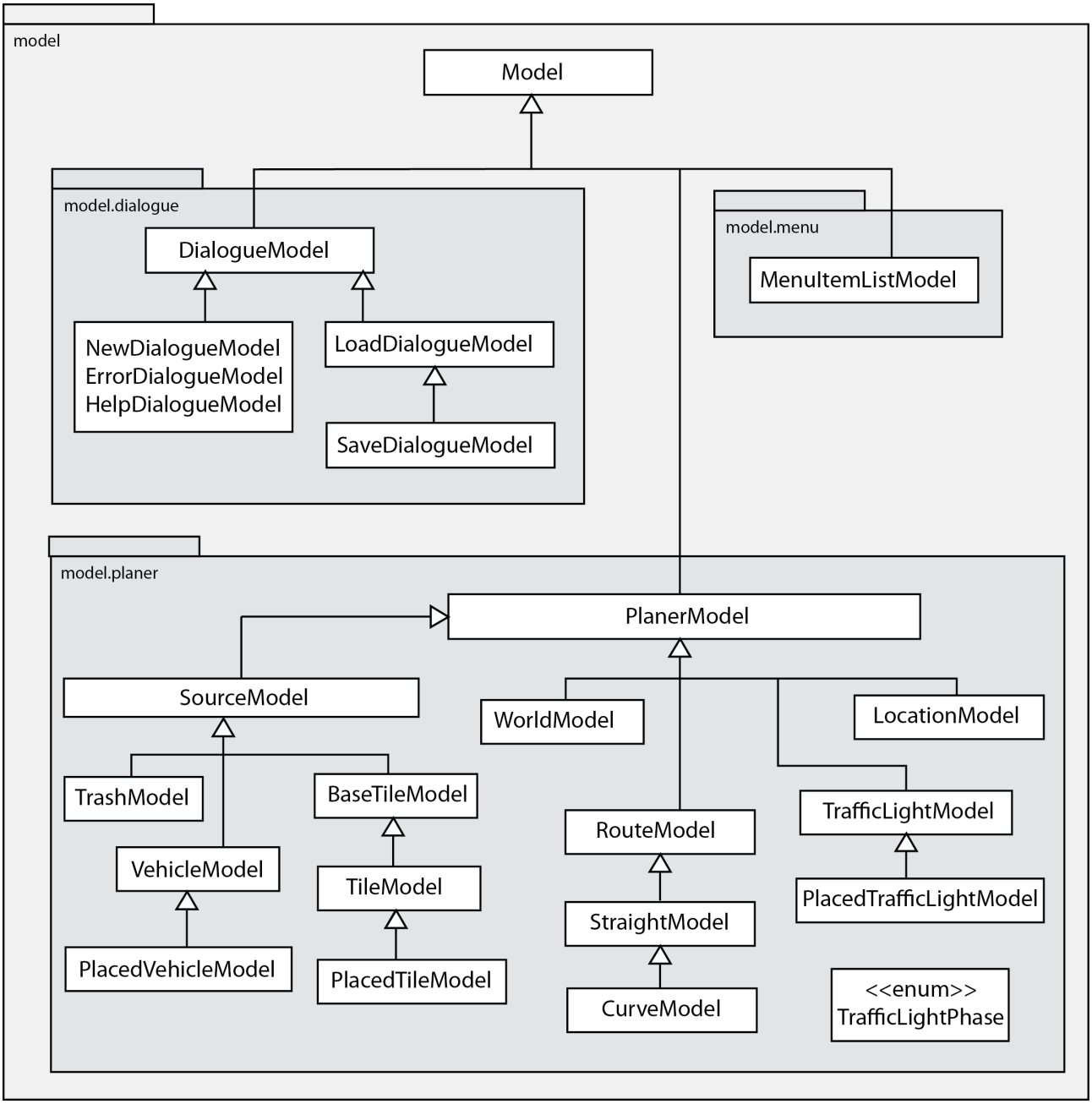
2. Klassen und Methoden

Die in der Systemarchitektur beschriebenen Bestandteile sind im angehangenen Dokument "Designtapete" mit Klassen und Methoden des Verkehrsplaners sowie deren Beziehungen ersichtlich. Zur besseren Veranschaulichung dient an dieser Stelle eine Übersicht, in der nur die Klassen der Pakete dargestellt sind:

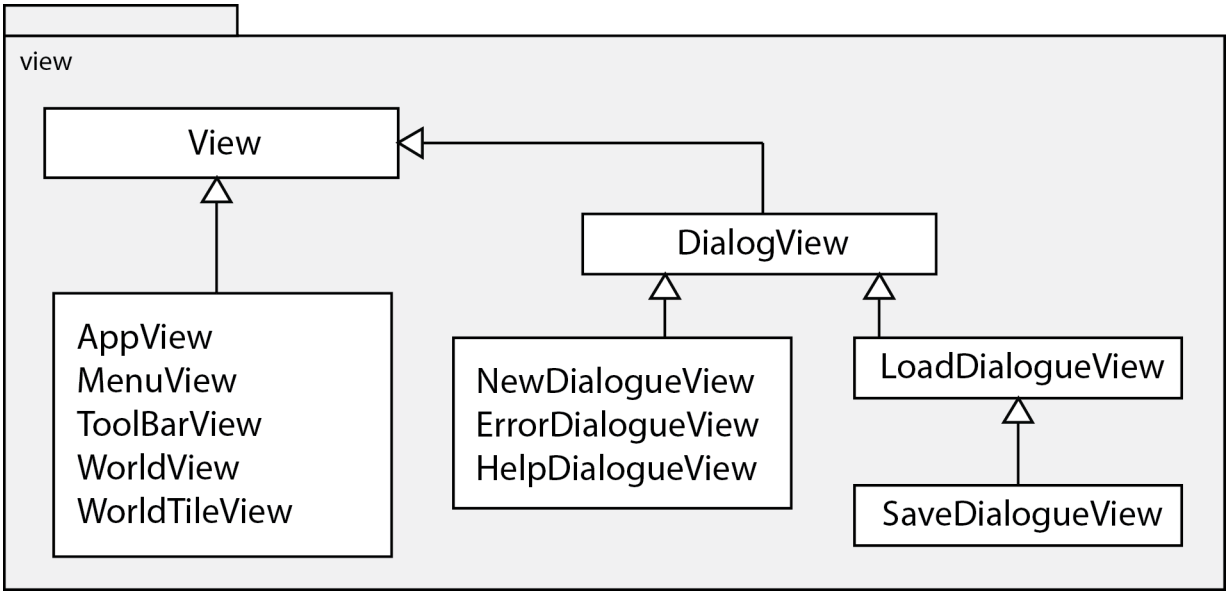
2.1 data


data
DataFactory
data.datamanager
Manager
WorldManager
data.datasource
<<interface>>
DataSource
data.datasource.xml
XMLDataSource
XMLSource
WorldXMLSource
LanguageXMLSource
TileXMLSource
data.dataobject
DataObject
VehicleData
PlacedVehicleData
WorldData
VehicleSpeedData
TextData
TraffixLightData
LocationData
BaseTileData
TileData
PlacedTileData
RouteData
StraightData
CurveData

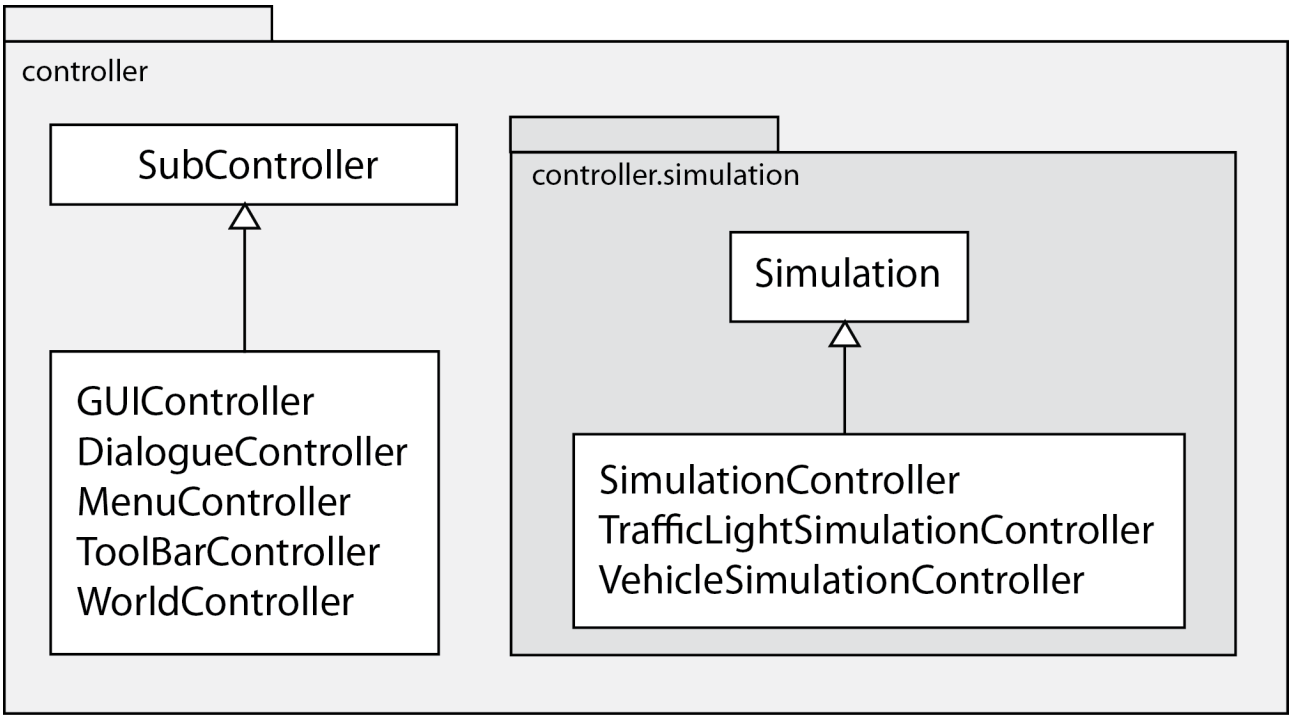
2.2 model


model
Model
model.dialogue
DialogueModel
NewDialogueModel
ErrorDialogueModel
HelpDialogueModel
LoadDialogueModel
SaveDialogueModel
model.menu
MenuItemListModel
model.planer
PlanerModel
SourceModel
WorldModel
LocationModel
TrashModel
BaseTileModel
VehicleModel
TileModel
PlacedVehicleModel
PlacedTileModel
RouteModel
StraightModel
CurveModel
TrafficLightModel
PlacedTrafficLightModel
<<enum>>
TrafficLightPhase

2.3 view


view
View
DialogView
AppView
MenuView
ToolBarView
WorldView
WorldTileView
NewDialogueView
ErrorDialogueView
HelpDialogueView
LoadDialogueView
SaveDialogueView

2.4 controller



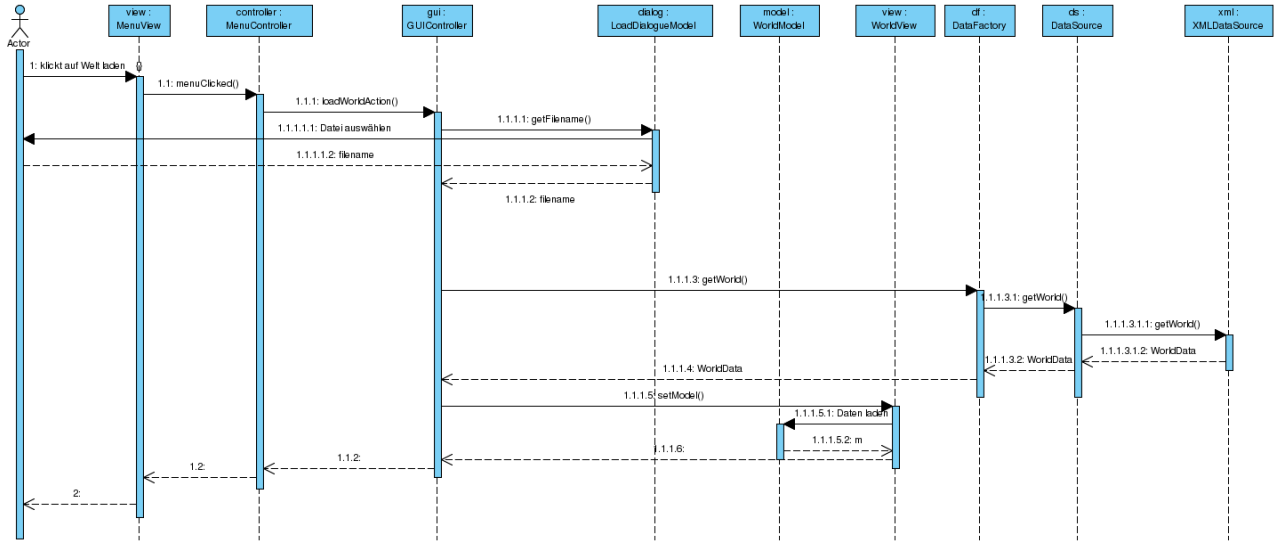
3. Use-Case-Realisierungen

Entsprechend der Anwendungsfälle im Pflichtenheft, erfolgt die Darstellung vier ausgewählter Use-Cases als Sequenzdiagramm:

- Welt aus Datei laden (siehe Eine Welt aus einer Datei laden)
- Kachel auf Welt legen per Drag&Drop (siehe Kachel platzieren)
- Ampel auf Straßenkachel ohne Ampel-Version ziehen (siehe Ampel platzieren)
- Fahrzeug-Simulation (siehe Simulation starten)

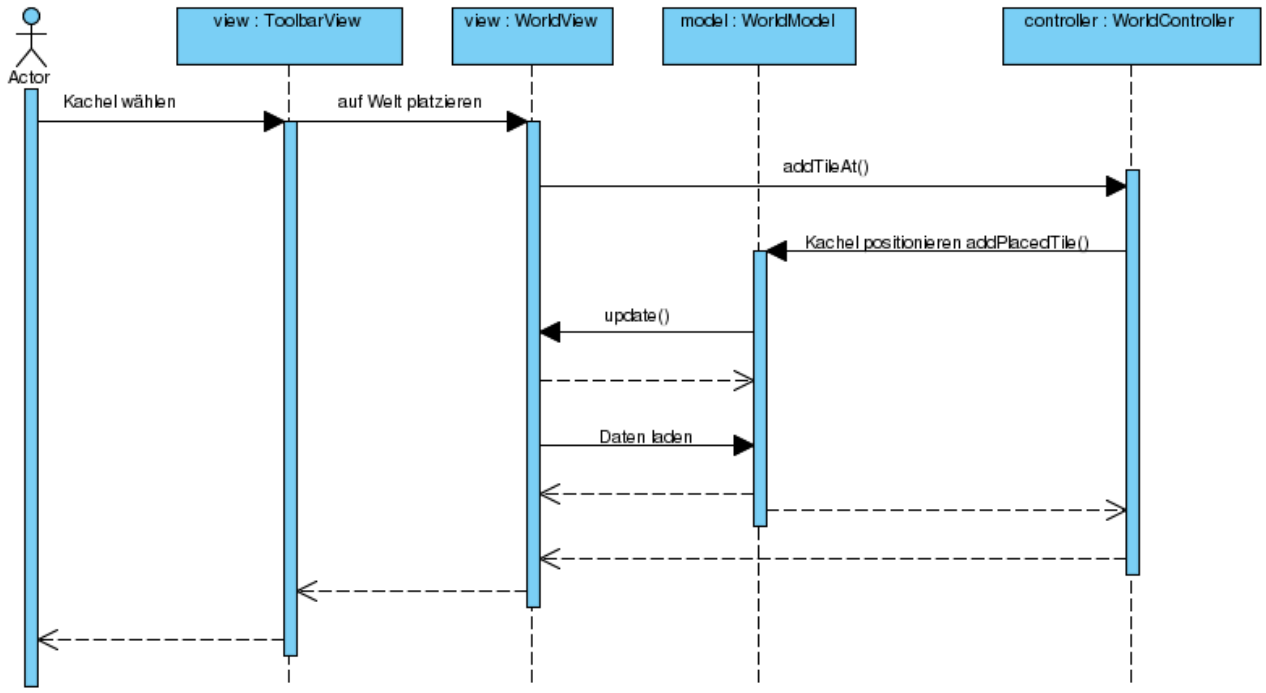
3.1 Welt aus Datei laden

Das Laden der Welt wird durch den Nutzer angestoßen, welcher über die Menüleiste "Welt laden" aufruft. Durch das Click-Event wird über den GUI-Controller ein Dialog zum Auswählen der Datei aufgerufen, welcher den Dateinamen an den GUI-Controller weiterreicht. Des Weiteren stößt dieser Controller das Laden der Welt an, deren Informationen letztendlich aus der gespeicherten Datei über die XMLDataSource geliefert werden. Vom GUI-Controller wird die Methode setModel() aufgerufen, welche über die WorldView aus dem entsprechenden Model alle Anzeige-Daten lädt und die Welt schließlich geladen ist.



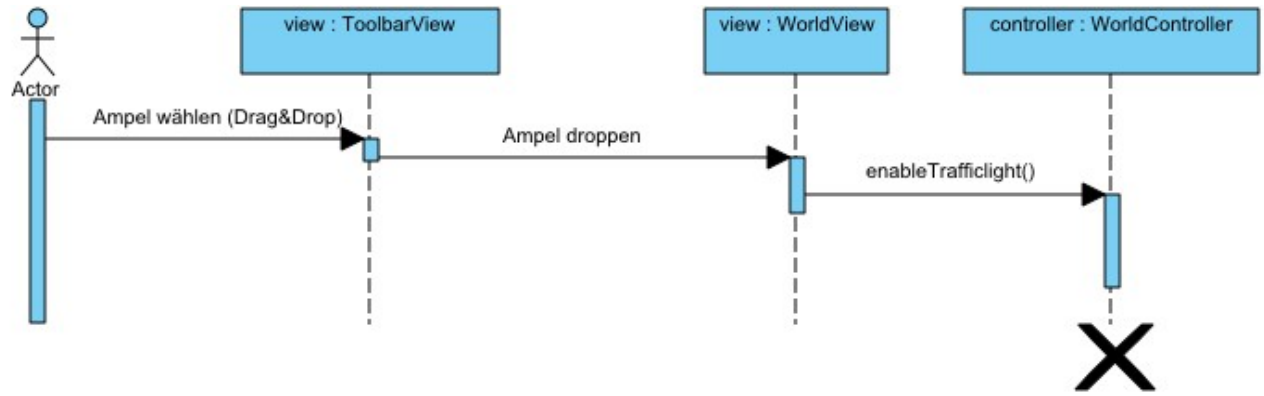
3.2 Kachel auf Welt legen per Drag&Drop

Beim Drag&Drop einer Kachel aus der Werkzeugleiste auf die Welt, wird durch das Loslassen der Kachel auf der WorldView der WorldController angestoßen. Sofern die Kachel an der Stelle platziert werden kann, wird dem WorldModel eine Kachel hinzugefügt und die WorldView aktualisiert sich. Sollte die Kachel in unzulässigen Bereichen losgelassen werden, wird die WorldView nicht aktualisiert und der Ausgangszustand tritt wieder ein. Die für Anzeige und Update nötigen Daten erhält die WorldView aus dem WorldModel.



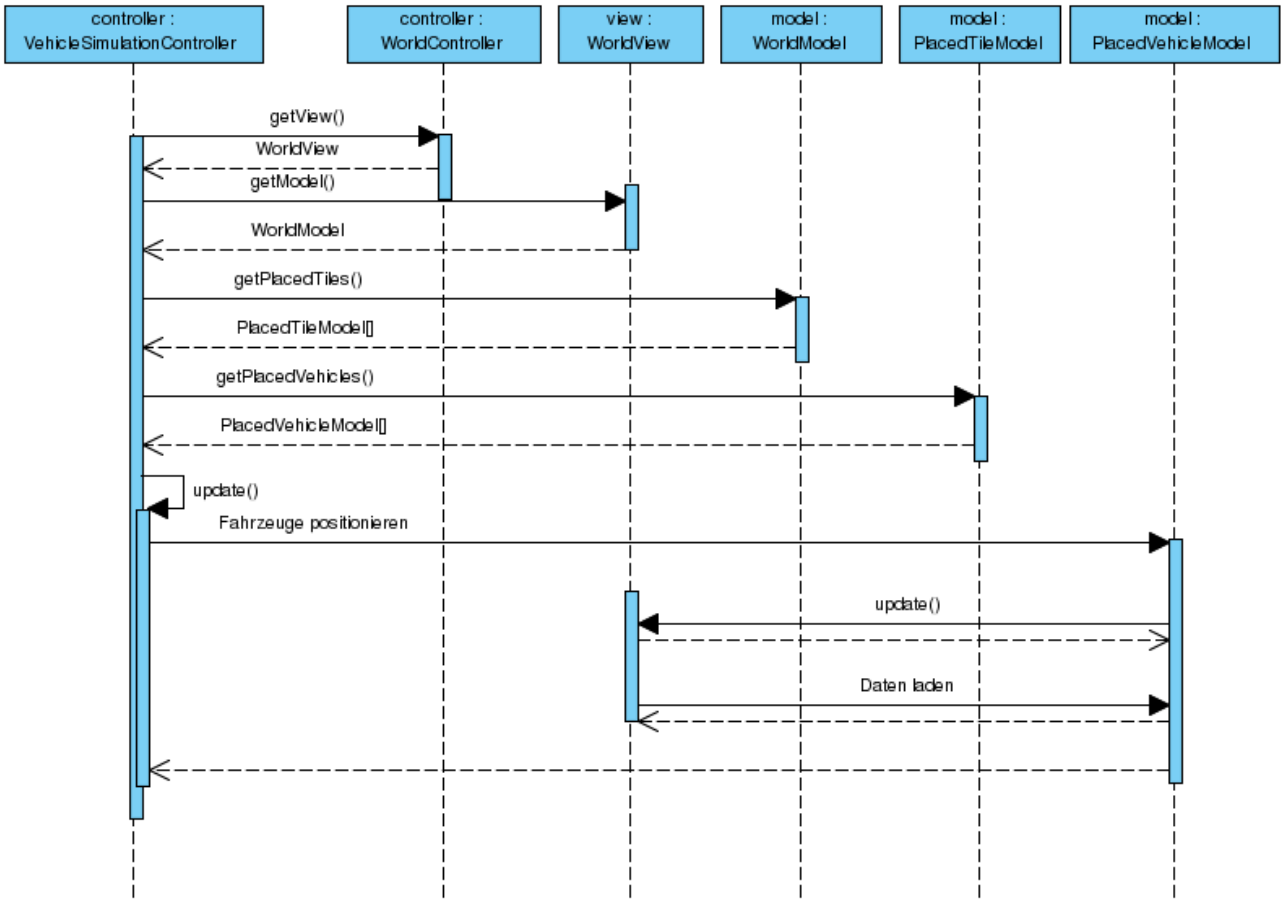
3.3 Ampel auf Straßenkachel ohne Ampel-Version ziehen

Wird eine Ampel auf eine Straßenkachel ohne Ampel-Version gezogen, so wird die Methode enableTrafficlight() WorldController angestoßen. Da sich keine Ampel-Version der Kachel finden lässt, auf der die Ampel gedroppt wird, wird der Vorgang beendet ohne dass eine Änderung der WorldView stattfindet - der Ausgangszustand bleibt bestehen.



3.4 Fahrzeug-Simulation

Die Simulation der platzierten Fahrzeuge lädt sich das aktuelle WorldModel samt View sowie die Daten der aktuell platzierten Kacheln und Fahrzeuge. In einem stetigen Update wird nun die WorldView mit neuen Positionsdaten der Fahrzeuge aktualisiert. Dies passiert, solange die Simulation läuft.



4. JavaDoc

Die detaillierten Beschreibungen der Methoden sind als JavaDoc exportiert, eine aktuelle Version findet sich unter http://swt.medieninformatiker.info/doc/. Die Daten werden alle 30 Minuten aus dem SVN-Repository aktualisiert. Auf eine ausgedruckte Abgabe dieser rund 120 Seiten fassenden Dokumentation wird zugunsten der Umwelt verzichtet.

Die Zusammenfassung des JavaDoc als PDF (mit Stand vom 22.12.2010) findet sich unter: Verkehrsplaner.pdf.

Zum tieferen Verständnis der Anwendung sei hier besonders auf die Dokumentation der folgenden Klassen verwiesen:

- TrafficLightSimulationController
- VehicleSimulationController
- WorldTileView

5. Erklärung

Das vorgelegte Design-Dokument wurde eigenständig von

Simon Franzen

Sven Hesse

Markus Tacker

Ramona Vehabovic

erstellt.

Wiesbaden, den 22.12.2010