Softwaretechnikpraktikum
SS 2004 – Gruppe 15

Datum: 31. Mai 2004

Verantwortlicher: Benedikt Schmidt

Pflichtenheft

Projekt: Shuttle Simulation

Auftraggeber: Universität Paderborn
AG−Softwaretechnik
Warburger Straße 97
33098 Paderborn

Auftragnehmer: **Softwaretechnik Praktikum SS2004, Gruppe 15**

Becker, Matthias	Test-Beauftragter
Horstmann, Christoph	Implementierungs-Beauftragter
Lüken, Frank	Projektleiter
Meißner, Roland	Reengineering-Beauftragter
Pottmeyer, Felix	(Re-)Design-Beauftragter
Schmelter, David	Qualitätsbeauftragter (CVS)
Schmidt, Benedikt	Pflichtenheft-Beauftragter
Wendland, Jan	Webmaster

Softwaretechnikpraktikum
SS 2004 – Gruppe 15

Datum: 31. Mai 2004

Autoren:	
Becker Matthias	Pottmeyer Felix
Horstmann Christoph	Schmelter David
Lüken Frank	Wendland Jan
Meißner Roland	Schmidt Benedikt

Über dieses Dokument

Bei diesem Dokument handelt es sich um das Pflichtenheft zur Aufgabenstellung des Softwaretechnikpraktikums SS 2004. Die in diesem Dokument verwendete Notation aller Diagramme orientiert sich an UML 2.0, allerdings sind die Interfaces einiger Komponentendiagramme aufgrund von Together 6.1 noch in der alten Notation dargestellt.

Softwaretechnikpraktikum
SS 2004 – Gruppe 15

Datum: 31. Mai 2004

Inhalt

Softwaretechnikpraktikum
SS 2004 – Gruppe 15

Datum: 31. Mai 2004

Softwaretechnikpraktikum
SS 2004 – Gruppe 15

Datum: 31. Mai 2004

1. Zielbestimmung

Der Schienenverkehr soll auch im Bereich des individuellen Güter- und Personenverkehrs im Vergleich zum Straßenverkehr wettbewerbsfähig werden. Eine Möglichkeit dieses Ziel zu erreichen, ist eine Umstrukturierung des Schienenverkehrs mittels autonom operierender Shuttles.

Der Wechsel von der bürokratisch-zentralistischen Organisation des Schienenverkehrs zur Dezentralisierung verläuft allerdings nicht reibungslos. Von einer Fahrplan orientierten Organisation des Schienenverkehrs muss zu einer Bedarfsorientierten Organisation umgestellt werden. Die Shuttles besitzen ein großes Autonomiepotential, das gerade innerhalb von Unternehmen auch der Regulierung bedarf. Es entsteht ein Kontrollproblem, das innerhalb einer Simulationsumgebung näher untersucht werden soll. Ein virtuelles Unternehmen besitzt eine Menge autonom operierender Shuttles, die mit einer beliebigen simulierten Umwelt gekoppelt werden können. Die Shuttles müssen sich nicht nur um die Abwicklung von Aufträgen und die Betriebsfähigkeit kümmern, sondern müssen außerdem innerhalb eines durch Makler organisierten Marktes selbstständig Angebote generieren.

Ziel dieses Projekts sind drei Entwicklungen. Zunächst eine Realisierung des angedachten Shuttle Betriebs. Darüber hinaus zwei Tools, die es ermöglichen die Realisierung im Hinblick auf ihre Funktionalität im Vergleich zu anderen Realisierungen einzuschätzen. Die Entwicklungen sind also:

- Erstens die Shuttlesteuerung, die eine Unternehmensstruktur berücksichtigt
- Zweitens eine Visualisierung der Unternehmensdaten
- Drittens ein Designer um Simulationsszenarien selbst zu erstellen und zu verändern.

Dabei kann in allen drei Bereichen auf früheren Entwicklungen aufgebaut werden. Es existiert bereits eine Shuttlesteuerung, die es einzelnen Shuttles ermöglicht auf Angebote zu reagieren. Außerdem gibt es eine Visualisierung der Topologie einer Simulationsumgebung. Auch gibt es bereits eine vorgegebene Schnittstelle um generierte Simulationsszenarien in die Simulation einzuspielen.

Die Entwicklungen müssen von jeweils unterschiedlichen Benutzergruppen bedient werden können. Die Shuttle Simulation muss nur mit der simulierten Umwelt verbunden werden. Die Visualisierung muss von weniger geschultem Personal bedient werden können. Der Szenariodesigner wird durch mit der Materie vertrautes Fachpersonal bedient werden.

2. Produkteinsatz

Der folgende Abschnitt beschreibt das Umfeld, in dem die zu entwickelnde Software eingesetzt werden soll. Darüber hinaus werden die für die Problemstellung relevanten Fachbegriffe erläutert.

2.1 Beschreibung des Problembereiches

Der betrachtete Problembereich ist die Simulation eines Bahnnetzes. Dieses Netz ist folgendermaßen aufgebaut: Bahnhöfe sind untereinander mit Schienen und Weichen verbunden (vgl. Abb. 2.1). Zwischen den Bahnhöfen fahren Shuttles, die Personen transportieren. Es gibt

verschiedene Unternehmen, die jeweils eine eigene Shuttleflotte haben.

Die Aufträge zur Personenbeförderung werden durch die Simulation ausgeschrieben (vgl. Abb. 2.1). Innerhalb des Unternehmens entscheiden die Shuttles untereinander, welcher Shuttle auf eine Ausschreibung reagiert und sich mittels eines Angebots um die Ausschreibung bewirbt. Aus jedem Unternehmen darf immer nur maximal ein beliebiger Shuttle ein Angebot abgeben. Dabei konkurrieren die Shuttles unterschiedlicher Unternehmen miteinander. Erhält ein Shuttle den angestrebten Auftrag, so muss es diesen innerhalb einer mit der Ausschreibung bekannt gegebenen Frist selbst erfüllen um den angebotenen Betrag zu erhalten. Solange die Transportkapazität nicht überschritten wird kann ein Shuttle auch mehrere Aufträge parallel ausführen. Erfüllt ein Shuttle seinen Auftrag nicht, oder überschreitet er die Frist, muss er eine Konventionalstrafe zahlen. Für das Befahren einer Strecke werden Mautgebühren erhoben, die von der Entfernung abhängig sind. Weiterhin fallen beim Betrieb der Shuttles Wartungskosten an. Die Wartung der Shuttles erfolgt an beliebigen Bahnhöfen und kostet Zeit und Geld.

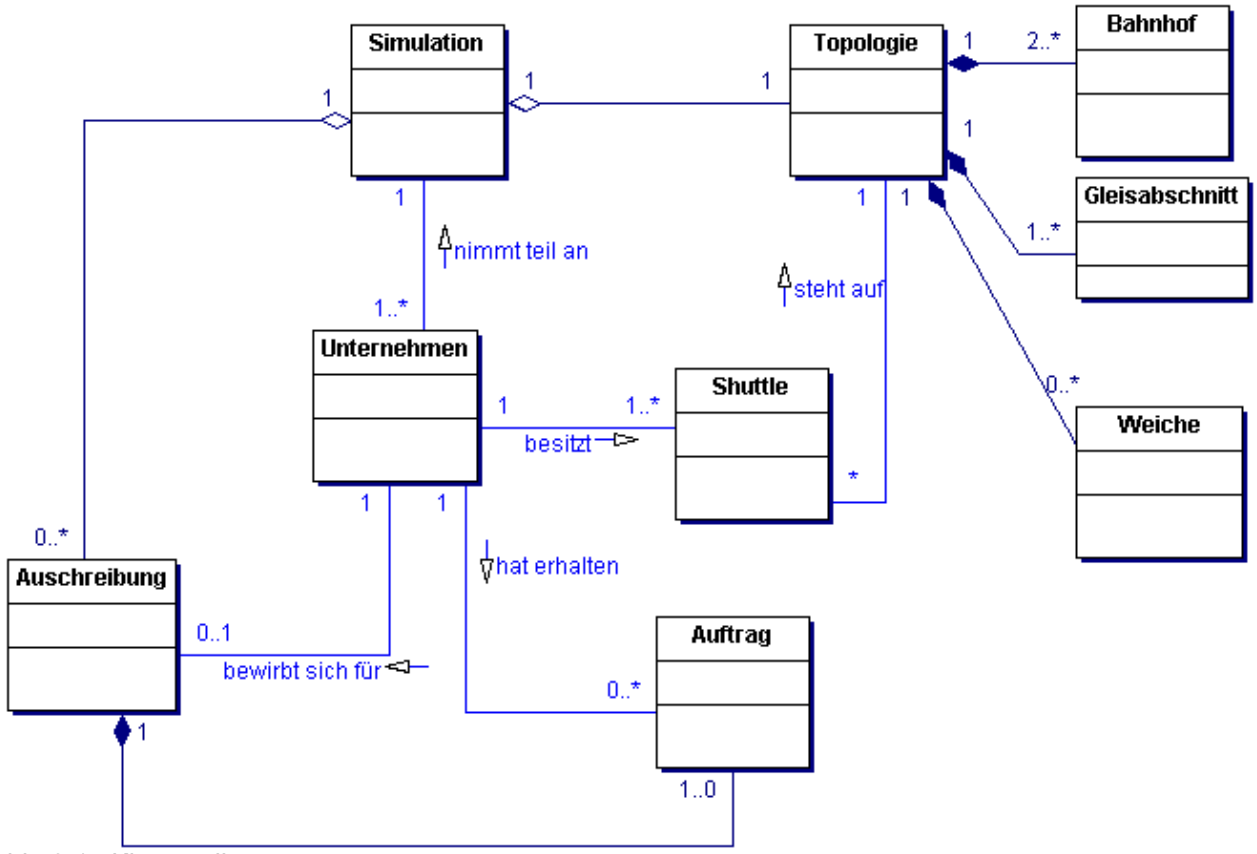

Abb. 2.1 : Klassendiagramm

2.1.1 Die Elemente der Simulation: Topologie und operative Einheiten

Wie aus Abb. 2.1.1 ersichtlich besteht die Topologie aus Bahnhöfen, Verbindungen und Weichen. Diese werden im Folgenden näher beschrieben. Dabei wird auch ihre Funktionalität innerhalb der Simulation erläutert.

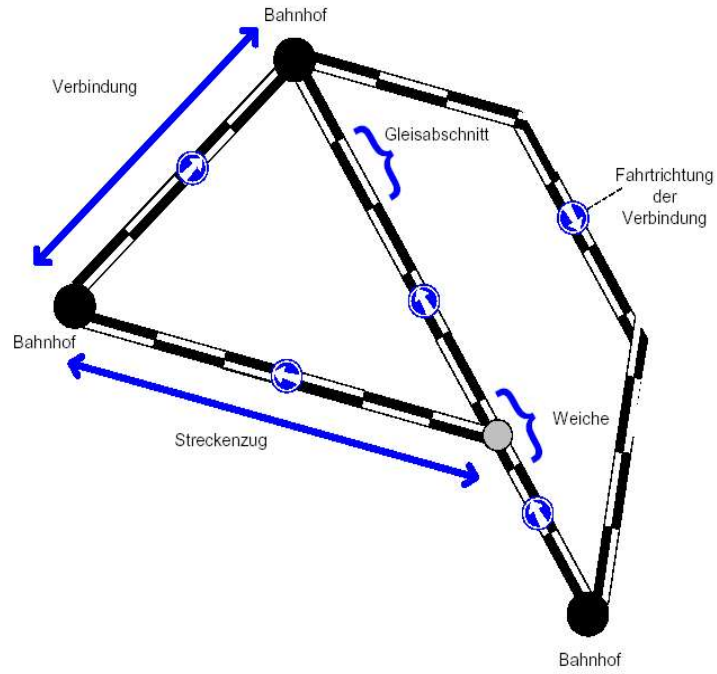

Abbildung 2.1.1 : Beispiel eines Streckennetzes

1) Bahnhöfe

Ein Bahnhof stellt den Knotenpunkt der Strecke dar. Durch ihn werden Verbindungen und Streckenzüge definiert, er ist der Ort, an dem Aufträge beginnen und enden. In einem Bahnhof können zur gleichen Zeit beliebig viele Shuttles parken. Nur hier kann eine Wartung erfolgen, die allerdings explizit in Auftrag gegeben werden muss.

2) Verbindungen

Verbindungen sind Streckenzüge zwischen zwei Bahnhöfen. Ein Streckenzug besteht aus beliebig vielen zusammenhängenden Gleisabschnitten und Weichen. Diese sind immer nur in eine Richtung befahrbar. Eine Weiche ist in Abhängigkeit der festgelegten Fahrtrichtung entweder als Verzweigung oder als Zusammenführung einer Strecke ausgelegt. Die Fahrtrichtung einer Verbindung ergibt sich aus der Richtung der zugrunde liegenden Gleisabschnitte. Es darf dabei immer nur eine direkte Verbindung pro Richtung zwischen zwei Bahnhöfen existieren.

Eine Verbindung zwischen zwei Bahnhöfen kann temporär ausfallen. Shuttles, die sich zum Zeitpunkt des Ausfalls bereits auf einem Gleisabschnitt dieser Verbindung befinden, werden von dem Ausfall nicht beeinflusst und können ihre Fahrt fortsetzen. Andere Shuttles können diese Verbindung nicht mehr nutzen und müssen eine Umleitung wählen.

3) Aufträge

Ein Auftrag besteht aus der Anzahl der Personen, einem Start- und Zielbahnhof und einer Bearbeitungszeit. Der Zeitpunkt zu dem der Auftrag erfolgreich beendet sein muss, ergibt sich aus

dem Zeitpunkt der Auftragsannahme und der vorgegebenen Bearbeitungszeit. Die Bearbeitungszeit beginnt mit der Vergabe des Auftrags an den Shuttle.

Die Auftragsvergabe erfolgt nach einem festgelegten Protokoll. Ein Element der Simulation ist der Makler(Broker Agent), der die Ausschreibungen ausgibt und sich um deren Vergabe kümmert. Zunächst werden alle Shuttles vom Makler über eine neue Ausschreibung informiert. Innerhalb eines in der Ausschreibung festgelegten Zeitfensters können die Shuttles ein Angebot abgeben. Der Shuttle mit dem günstigsten Angebot erhält vom Makler den Zuschlag. Werden von unterschiedlichen Shuttles gleiche Angebote abgegeben, so erhält dasjenige Shuttle den Zuschlag, das sein Angebot zuerst abgegeben hat. Ein Shuttle kann dabei mehrere Aufträge gleichzeitig annehmen.

4) Shuttle

Jedes Shuttle ist eine autonom agierende Einheit mit einer zu Beginn der Simulation festgelegten Transportkapazität. Diese darf nicht überschritten werden. Hat ein Shuttle einen Auftrag erhalten, muss dieser Shuttle den Auftrag bearbeiten und kann ihn nicht an einen anderen Shuttle übergeben. Jeder Shuttle muss mindestens einen Auftrag durchführen, damit das Unternehmen, zu dem es gehört, nicht disqualifiziert wird.

Zur Erfüllung des Auftrags muss der Shuttle zuerst den Startbahnhof anfahren, die Personen aufnehmen, diese anschließend zum Zielbahnhof bringen und dort entladen. Dieser Vorgang muss innerhalb der festgelegten Bearbeitungszeit erfolgen, da ansonsten eine Konventionalstrafe erhoben wird (siehe Abschnitt 2.2). Die Auftragsbearbeitung beginnt mit dem Beladen des Shuttles im Startbahnhof und endet mit dem Entladen im Zielbahnhof. Das Beladen, Entladen sowie Zwischenlagern an anderen Bahnhöfen ist nicht erlaubt.

Fährt ein Shuttle eine Verbindung zwischen zwei Bahnhöfen, so kann es weder seine Fahrtrichtung wechseln, noch einen anderen Zielbahnhof wählen. Eine Fahrtentscheidung ist nur in einem Bahnhof vor Antritt einer Fahrt möglich.

2.2 Glossar

Der Glossar enthält eine nähere Beschreibung der elementaren Begriffe des Problembereichs. So werden Definitionen abgegeben, die für das gesamte Pflichtenheft Gültigkeit besitzen.

Begriff	***Beschreibung***
Angebot	Wird vom Makler ein Transportauftrag ausgeschrieben, so kann jedes Unternehmen ein Angebot abgeben. Das Angebot enthält eine Geldsumme, für die ein Unternehmen bereit ist einen Auftrag durchzuführen.
Auftrag	siehe Transportauftrag
Auftragsdaten	Die Auftragsdaten bestehen aus der Anzahl der Personen, einem Start- und Zielbahnhof, so wie einer vorgegebenen Bearbeitungszeit.

Begriff	*Beschreibung*
Ausgaben	Mautgebühren, Wartungskosten, Konventionalstrafen, Strafen für mehrfache Angebotsabgabe, Kosten für Nachrichten (nicht für Nachrichten an Shuttles im eigenen Unternehmen)
Ausschreibung	Der Makler sendet eine Nachricht mit den Auftragsdaten an alle Shuttles. Die Shuttles eines Unternehmens können, innerhalb eines bestimmten Zeitfensters, ein Angebot für den Auftrag abgeben. Es dürfen niemals zwei Shuttles eines Unternehmens ein Angebot abgeben. Es erhält das Unternehmen den Auftrag, welches das günstigste Angebot innerhalb der Ausschreibungsdauer abgibt.
Bahnhof	In einem Bahnhof können zur gleichen Zeit beliebig viele Shuttles parken. Die Dauer des Aufenthaltes eines Shuttles im Bahnhof gilt nicht als Wartungszeit. Die Passagiere zu einem Auftrag werden an Bahnhöfen zu- oder aussteigen. Zwei Bahnhöfe können durch einen Streckenzug verbunden werden.
Bearbeitungszeit	Wird ein Auftrag nicht innerhalb der gegebenen Bearbeitungszeit ausgeführt, so muss eine Konventionalstrafe gezahlt werden. Die Bearbeitungszeit beginnt mit der Vergabe des Auftrags an ein Unternehmen.
Gleisabschnitt	Mehrere Gleisabschnitt bilden einen Streckenzuges. Die Anzahl der Gleisabschnitte in einem Streckenzug bestimmt dessen Länge. Die Gleisabschnitte sind immer nur in eine Richtung befahrbar.
Kernel	siehe Simulationsumgebung
Konventionalstrafe	Eine Konventionalstrafe muss gezahlt werden, wenn ein Auftrag nicht innerhalb der vorgegebenen Bearbeitungszeit abgegeben wurde.
Makler	Der Makler schreibt Transportaufträge aus und nimmt die Angebote entgegen. Er ist Teil der Simulationsumgebung.
Mautgebühr	Befährt ein Shuttle eine Verbindung zwischen zwei Bahnhöfen, so müssen Mautgebühren bezahlt werden. Die Kosten einer Verbindung errechnen sich aus der Summe der Kosten der benutzten Gleisabschnitte dieser Verbindung.
Nachricht	Shuttles können Nachrichten entweder an andere Shuttles versenden, oder an den Makler um ein Angebot abzugeben. Für das Versenden von Nachrichten, z. B. zur Abgabe von Angeboten, fallen Kosten an, unternehmensinterne Nachrichten von Shuttle zu Shuttle - bspw. zur Koordination - sind kostenfrei.
Shuttle	Ein Shuttle ist ein eigenständiges Schienenfahrzeug, welches Transportaufträge ausführen kann. Ein Shuttle gehört einem Unternehmen an.

Begriff	*Beschreibung*
Simulationsumgebung	In der Simulationsumgebung werden das Streckennetz, die Shuttles und die Daten für mögliche Aufträge geladen, so wie die Simulation durchgeführt. Eine Visualisierung kann sich mit der Simulationsumgebung verbinden, um relevante Daten anzuzeigen.
Streckennetz	Das Streckennetz besteht aus Bahnhöfen, Gleisabschnitten und Weichen.
Streckenzug	Ein Streckenzug besteht aus beliebig vielen zusammenhängenden Gleisabschnitten und Weichen. Ein Streckenzug zwischen zwei Bahnhöfen stellt eine Verbindung dar.
Szenario	Ein Szenario enthält Informationen über den Ablauf einer Simulation der Simulationsumgebung und die verwendete Topologie. Zum Ablauf gehören Ausschreibungen von Transportaufträgen und Ausfälle von Streckenzügen.
Topologie	siehe Streckennetz
Transportauftrag	Aufträge werden von einem Makler ausgeschrieben. Ein Auftrag definiert die Anzahl der Personen sowie einen Start- und Zielbahnhof und muss innerhalb einer vorgegebenen Bearbeitungszeit ausgeführt werden. Der Termin, zu dem der Auftrag erfolgreich beendet sein muss, ergibt sich aus dem Zeitpunkt der Auftragannahme und der vorgegebenen Bearbeitungszeit. Die Bearbeitungszeit beginnt mit der Vergabe des Auftrags an den Shuttle.
Unternehmen	Dem Unternehmen gehören mehrere Shuttles an. Die einzelnen Shuttles müssen durch Kommunikation und Strategie einen möglichst hohen Gewinn zu erwirtschaften.
Verbindung	siehe Streckenzug
Visualisierung	Eine Visualisierung kann sich mit der Simulationsumgebung verbinden, um relevante Daten zu empfangen und darzustellen. In diesem Dokument wird zwischen der Visualisierung der Topologie und der Datenvisualisierung unterschieden. Wird nur von einer Visualisierung gesprochen, so ist damit automatisch die Datenvisualisierung gemeint, da diese in diesem Dokument klar im Vordergrund steht.
Wartung	Nach dem Zurücklegen einer festgelegten Entfernung muss eine kostenpflichtige Wartung durchgeführt werden. Wird diese Entfernung überschritten, so kann das Shuttle nicht mehr weiter fahren, solange es nicht gewartet wurde.

Begriff	*Beschreibung*
Weiche	Eine Weiche ist in Abhängigkeit der festgelegten Fahrtrichtung entweder als Verzweigung oder als Zusammenführung einer Strecke ausgelegt.

3. Reverse Engineering (Ist-Zustand)

Wie bereits in der Zielbestimmung erwähnt exisietrt bereits Software, die im Problembereich angesiedelt ist. Dieses Kapitel beschäftigt sich mit dem Reverse-Engineering der bereits vorhandenen Produkte und Schnittstellen. Dies wird es ermöglichen bereits vorhandene Softwarekomponenten, deren Funktionalität insgesamt oder teilweise, für die angestrebte Entwicklung notwendig sind, zu benutzen.

Die bestehende Shuttle Steuerung wird im Hinblick auf ihre Kommunikation mit den einzelnen Teilen der Simulationsumgebung, insbesondere mit dem Makler untersucht. Die Visualisierung der Topologie ermöglicht eine Beschreibung der Weitergabe von Simulationsdaten an externe Softwarekomponenten. Die Struktur der XML Dateien, die bisher die Grundlage für den Entwurf von Simulationsszenarien bilden, wird genauer beschrieben.

Die verwendeten Hilfsmittel sind dabei neben einer textuellen Beschreibung Klassen- und Sequenzdiagramme.

3.1 Shuttlesteuerung

In diesem Abschnitt wird der Ist-Zustand der Shuttlesteuerung dargestellt. Zunächst wird auf die Klassen und Interfaces eingegangen, danach wird die Funktionsweise mit Hilfe eines Sequenzdiagrammes dargestellt

3.1.1 Erläuterung des Aufbaus der ShuttleSteuerung

Das Klassendiagramm Abbildung 3.1.1 zum Aufbau der Shuttlesteuerung stellt die wichtigsten Klassen aus dem Package „Agent“ dar.

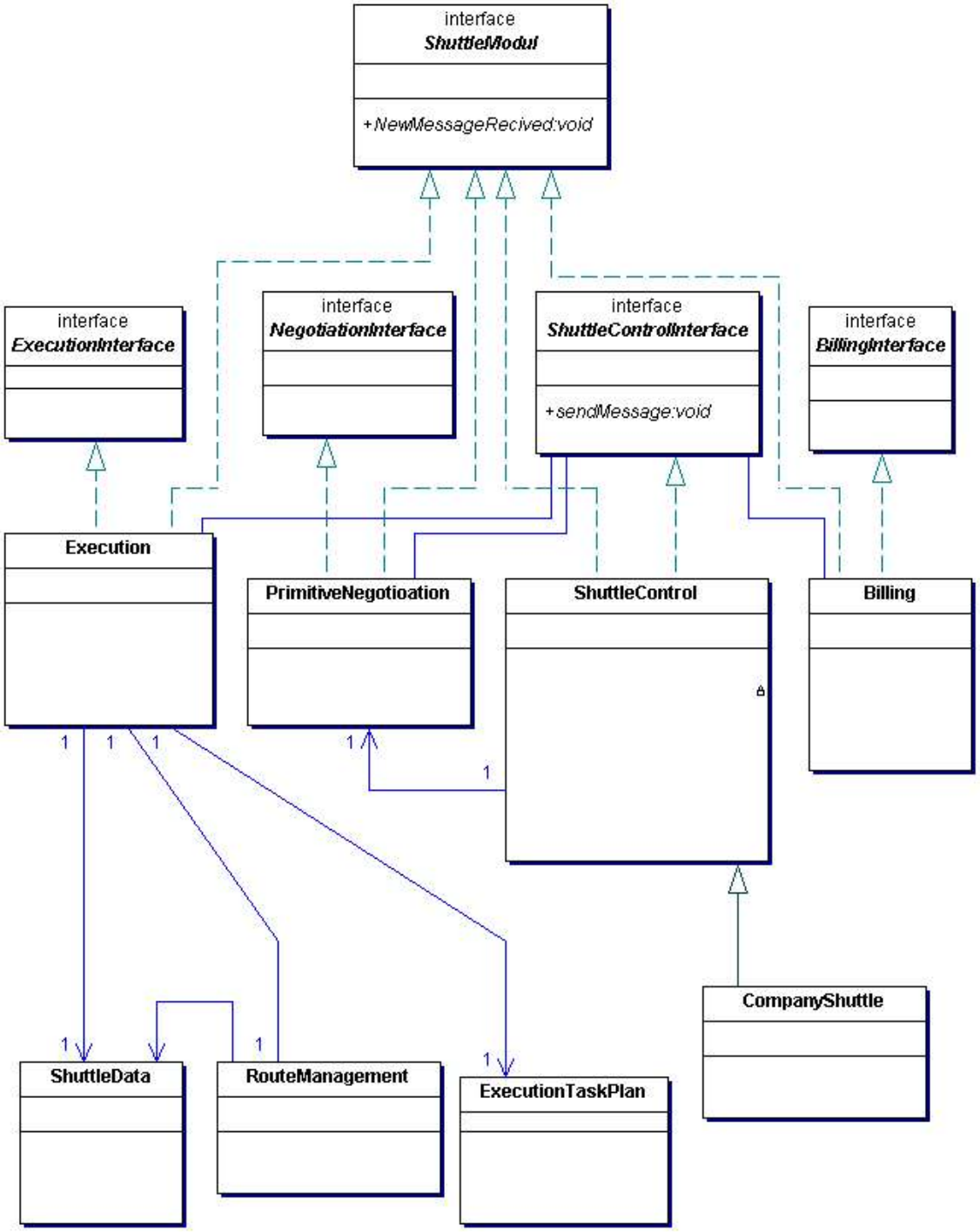

Abbildung 3.1.1 : Klassendiagramm der Shuttlesteuerung

Name	*Aufgabe*
Executioninterface:	Das Executioninterface Liefert eine Methode um neue Tasks für das Shuttle zu erteilen.
ShuttleControlInterface:	Das Shuttlecontrolinterface ist Hauptsächlich für das senden von Nachrichten zuständig. Außerdem liefert es Methoden um dem Shuttle mitzuteilen das ein Angebot abgelehnt wurde oder das es Repariert werden muss.
ShuttleModule:	Das Shuttlemodule ist für das Empfangen neuer Nachrichten verantwortlich.
BillingInterface	Ein Interface welchen jede BillingKlasse implimentieren muss sie ist für die Bezahlung zuständig.
NegotiationInterface	Gibt die Länge einer Verbindung , damit die kosten berechnet werden können.
CompanyShuttle:	Liefert den Konstruktor der im Kernel ein neuer Shuttle erzeugt.
Execution	Ist für die Ausführung von Aktionen des Shuttles verantwortlich.
ExecutionTaskPlan:	In dieser Klasse wird eine Liste erstellt, die alle benötgten Tasks in der Reihenfolge der Ausführung beinhaltet. Tasks sind: • LoadShuttleTask (Es werden Passagiere eingeladen) • UnloadShuttleTask (Es werden Passagiere ausgeladen) • GetConnectionOkTask (Es wird geprüft ob die Strecke zum Bahnhof ok ist) • DoNothingUntilTask .... • RepairShuttleTask (Das Suttle wird Repariert)
PrimitiveNegotation	In dieser Klasse wird für jeden ausgeschriebenen Auftrag der wurde ein Angebot berechnet und abgesandt.
RouteManagement	In der Klasse Routemanagement werden die Wege berechet die das Shuttle fährt.
ShuttleControl	Dies ist die Steuerzentrale des Shuttles.
ShuttleData	Die Klasse ShuttleData enthält wichtige Daten über die Eigenschaften des Shuttles.
Billing	Die Klasse Billing kümmert sich um die Finanzen und das Rechnungswesen des Shuttles

3.1.2 Sequenzdiagramm der Shuttlesteuerung

In diesem Abschnitt zeigen wir beispielelhaft den Ablauf einer Angebotsabgabe und nachfolgende Bearbeitung des erhaltenen Auftrags anhand eines Sequenzdiagramms(vgl. Abb. 3.1.2).

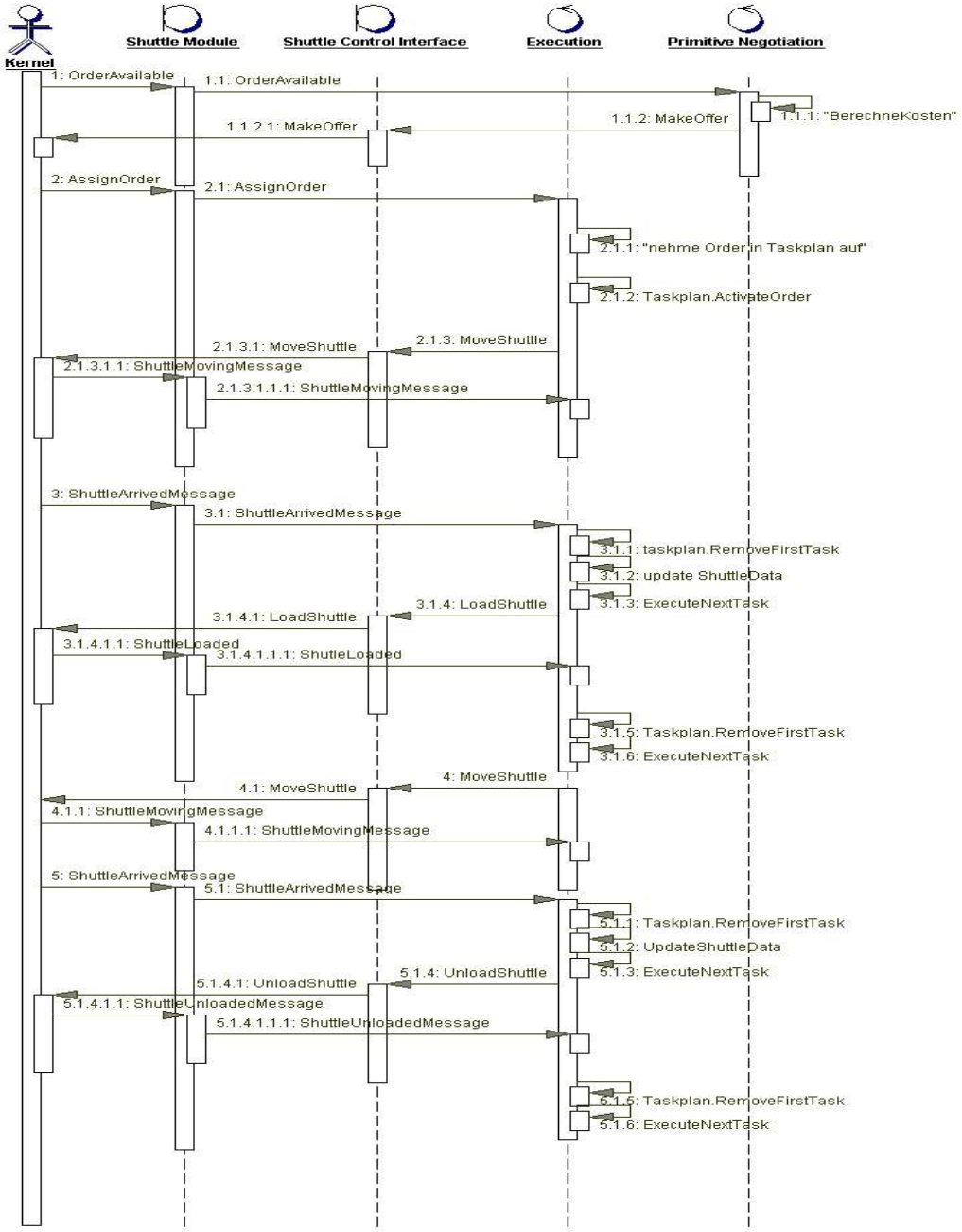

Abbildung 3.1.2 Sequenzdiagramm zur Angebotsabgabe und Abarbeitung eines Auftrags

Das Sequenzdiagramm stellt die Kommunikation zwischen der Shuttlesteuerung und dem Kernel, am Beispiel einer Auftragsausschreibung, der Reaktion darauf, der Angebotszuteilung und der Abarbeitung des Auftrags, dar.

Die Kommunikation zwischen Shuttlesteuerung und Kernel findet über die Interfaces ShuttleControlInterface (stellt die Methode *public void sendMessage (Message msg)* für vom Shuttle gesendete Nachrichten bereit) und ShuttleModule (stellt die Methode *public void newMessageRecieved(Message msg)* bereit, über die Nachrichten beim Shuttle ankommen) statt.

Der User ist in diesem Diagramm der Kernel. Er informiert den Shuttle über eine neue Ausschreibung, indem er eine Message vom Typ OrderAvailable sendet. Auf diese Message wird in der Klasse PrimitiveNegotiation reagiert. Der Shuttle ermittelt, welche Kosten bei der Abarbeitung der Ausschreibung entstehen würden und schickt dann sein Angebot in Form eines MakeOffer Objektes an den Kernel. Dieser vergibt mit einer AssignOrder Message den Auftrag an der Shuttle, welches das günstigste Angebot gemacht hat. Diese Message wird in der Klasse Execution verarbeitet, die auch für die gesamte Abarbeitung eines Auftrags verantwortlich ist. Hat der Shuttle einen Auftrag erhalten, nimmt es diesen in seinen Taskplan auf und beginnt mit der Abarbeitung. Sofern sich der Shuttle nicht am Startbahnhof befindet, muss es sich zuerst dorthin bewegen, weshalb es eine MoveShuttle Message an den Kernel sendet. Dieser antwortet mit einer ShuttleMovingMessage sobald der Shuttle sich losbewegt und mit einer ShuttleArrivedMessage sobald der Shuttle im Startbahnhof des Auftrags angekommen ist. Da der erste Task nun erfolgreich abgearbeitet wurde, kann er aus dem Taskplan entfernt und der nächste Task (LoadShuttle) ausgeführt werden.

3.2 Visualisierung

Dieser Abschnitt stellt den Ist-Zustand der bereits existierenden Visualisierung da. Insbesondere wird die Kommunikation zwischen Simulationskern und Visualisierung erläutert. Zu Beginn wird kurz auf den Aufbau der vom Kernel bereitgestellten Kommunikationsschnittstelle eingegangen.

3.2.1 Kommunikationschnittstelle des Kernels

Dieser Abschnitt gibt einen Überblick über die Funktionsweise der Kommunikationschnittstelle vom Kernel und den beteiligten Klassen.

1) Allgemeines zum Aufbau

Jegliche Kommunikation einer Visualisierung mit dem Kernel erfolgt über RMI. RMI steht für Remote Method Invocation und stellt seit dem JDK 1.1 einen Mechanismus zur Verfügung, der es ermöglicht, Objekte auf einfache Weise in einem Netzwerk zu verteilen und ihre Funktionalität anderen Arbeitsplätzen zur Verfügung zu stellen.

Die per RMI zur Verfügung gestellten, verteilten Objekte werden durch einen Name-Service, der RMI-Registry, bereitgestellt.

Der Kernel erstellt dazu ein Objekt vom Typ `Visualisation`, in dessen Konstruktor sowohl die

RMI-Registry gestartet, als auch das Visualisation-Objekt an dieser angemeldet wird.

Dieses Visualisation-Objekt läuft in einem eigenen Thread. In der `run` Methode werden die verschiedenen Remote-Events, die vom Kernel erzeugt werden, abgefragt und über "Daten"-Objekte an die sich verbundenen Visualisierungen verteilt, wo sie zur weiteren Verarbeitung in einer Remote-Queue abgelegt werden.

2) Übersicht über die RemoteEvents

Die Kommunikation mit den Visualisierungen erfolgt über Objekte, die alle von der serialisierbaren Klasse `de.upb.swtpra.kernel.visualisation.RemoteObj` erben und verschiedene Daten kapseln, die sich während der Simulation geändert haben. Die folgende Tabelle gibt eine Übersicht über diese RemoteEvents:

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteConnectionStatus`	Wird versendet, wenn sich der Status einer Verbindung zwischen zwei Bahnhöfen verändert hat.	`startStation`	Start-Station ID der Verbindung
		`destStation`	Ziel-Station ID der Verbindung
		`status`	Status der Verbindung

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteEndGame`	Wird versendet, wenn die Simulation beendet wurde.	*keine*	

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteGameConstants`	Übermittelt allgemeine Simulations-Parameter an die Visualisierung	`maxAllowedStretchWithoutRepair`	maximaler Weg eines Shuttles ohne Wartung
		`repairCosts`	Wartungskosten
		`repairTime`	Zeit, die eine Wartung in Anspruch nimmt
		`orderLoadTime`	Zeit, die das Beladen eines Shuttles in Anspruch nimmt
		`orderUnLoadTime`	Zeit, die das Entladen eines Shuttles in Anspruch nimmt.
		`timeFactor`	Simulationsgeschwindigkeit
		`startAmount`	Startkapital der Shuttles
		`maxShuttleCapacity`	maximale Beladung eines Shuttles

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteOrderAssigned`	Wird versendet, wenn ein Shuttle einen Auftrag zugeteilt bekommen hat.	`shuttleID`	ID des Shuttles, das den Zuschlag bekommen hat
		`companyId`	ID des Unternehmens

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteOrderCreated`	Wird versendet, wenn eine neue Ausschreibung erstellt wurde.	`orderID`	Auftrags-ID
		`destinationTimeLimit`	Frist, bei der der Auftrag abgeschlossen sein muss.
		`duration`	Dauer der Ausschreibung
		`penalty`	Strafe bei Nichterfüllung
		`startStationID`	ID Start-Bahnhof
		`destinationStationID`	ID Ziel-Bahnhof
		`destinationStation`	Name Ziel-Bahnhof
		`startStation`	Name Start-Bahnhof

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteOrderDelete`	Wird versendet, wenn eine Ausschreibung gelöscht wurde.	`orderID`	Auftrags-ID

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteShuttleBankrupt`	Wird versendet, wenn einem Shuttle nicht genug Geld zur Verfügung steht.	`money`	Kapital des Shuttles
		`shuttleId`	ID des Shuttles
		`companyId`	Unternehmens-ID, dem das Shuttle gehört

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteShuttleMoved`	Wird versendet, wenn ein Shuttle bewegt wurde.	`tdo`	Topologiedaten
		`shuttleId`	ID des Shuttles
		`companyId`	Unternehmens-ID, dem das Shuttle gehört
		`name`	

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteShuttleDisqualified`	Wird versendet, wenn ein Shuttle disqualifiziert wurde.	`shuttleId`	ID des Shuttles
		`companyId`	Unternehmens-ID, dem das Shuttle gehört

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteShuttleStatusChanged`	Wird versendet, wenn der Status eines Shuttles verändert wurde.	`stretchSinceLastRepair`	Gefahrene Strecke seit letzter Wartung
		`loadedOrdersIds`	Angenommene Aufträge
		`money`	Kapital des Shuttels
		`shuttleId`	ID des Shuttles
		`companyId`	Unternehmen, dem das Shuttle gehört
		`currentLoad`	aktuelle Beladung

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteShuttleStatusString`		`info`	Informations-String
		`shuttleId`	ID des Shuttles
		`companyId`	Unternehmens-ID, dem das Shuttle gehört

Name der Klasse	*Zweck*	*Attribute*	*Attribut-Beschreibung*
`RemoteTopologyData`	Senden von Topologie-Daten	`td`	Topologiedaten

3.2.2 Aufbau des Visualisation-Clients

Dieser Abschnitt erläutert den Aufbau der Visualisierung. Details zur Kommunikation zwischen Kernel und Visualisierung folgen in Abschnitt 3.2.3.

1) Allgemeines zum Aufbau

Die grundlegenden Klassen, aus denen jeder Visualisation-Client aufgebaut ist, liegen alle in dem Package `de.upb.swtpra.kernel.visualisation.client`. Diese sind:

Name der Klasse	*Funktion*
`RemoteMessageHandler`	Der RemoteMessageHandler läuft in einem eigenen Thread und ist dafür verantwortlich, ständig die neu angekommenden Events, die vom Kernel per RMI-Aufruf in der RemoteQueue abgelegt werden, auszulesen und über die `update` Funktion an den VisualisationClient weiter zu geben. Gestartet wird er, wenn der Client sich erfolgreich an den Kernel connecten konnte.

Name der Klasse	*Funktion*
`RemoteQueue`	In der RemoteQueue werden alle Events, die während der Simulation auftreten können und für den VisualisationClient von Bedeutung sind, vom Kernel abgelegt. Nach und nach werden diese dann vom RemoteMessageHandler abgearbeitet.
`RMIRemoteQueueInterface`	Das `RMIRemoteQueueInterface` stellt die für den Kernel wichtigen Funktionen `get` und `put` bereit, mit denen er die Events in der RemoteQueue ablegen kann und wird von der Klasse `RemoteQueue` implementiert.
`RMIRailwayPanelInterface`	Das `RMIRailwayPanelInterface` stellt die für den Kernel wichtigen Methoden `abortConnection` und `getQueue` bereit und wird vom VisualisationClient implementiert. `abortConnection` wird vom Kernel aufgerufen, wenn die Simulation beendet wurde. Mit der Methode `getQueue` kommt der Server an die RemoteQueue des Clients.
`ServerStatusChecker`	Der ServerStatusChecker fragt in gewissen Zeitabständen den Verbindungsstatus zum Server ab. Tritt hierbei eine Exception auf, so beendet er die Verbindung. Gestartet wird er, wenn der Client sich erfolgreich an den Kernel connecten konnte.

2) Allgemeines zur Funktionsweise

Für das Anzeigen der GUI des VisualisationClients ist die Klasse `VisualisationFrame` zuständig. Wird eine Verbindung zum Kernel aufgebaut, so werden RemoteQueue, RemoteMessageHandler und der ServerStatusChecker erstellt. Events, die vom Kernel in der RemoteQueue abgelegt werden, werden vom RemoteMessageHandler abgearbeitet, der die `update` Methode der Klasse VisualisationClientBase aufruft und das gelesene Remote-Objekt an diese weitergibt.

Innerhalb der `update` Methode wird das RemoteObjekt nun an die verantwortliche Klasse zur Datenhaltung -und Verarbeitung – `LUpdateData` - weitergereicht.

`LUpdateData` läuft in einem eigenen Thread und enthält eine eigene Queue, in der die vom RemoteMessageHandler weitergereichten Events erneut zwischengespeichert werden. In der `run` Methode von `LUpdateData` werden schließlich alle vom Kernel verschickten Events analysiert und zur Anzeige weiterverarbeitet.

Eine Besonderheit besteht noch beim RemoteMessageHandler: Wenn beim Start der Simulation zunächst die Topologie-Daten übermittelt werden, ruft er nicht die `update`-Methode des VisualisationClients auf, sonder dessen `init` Methode, die dann für das Zeichnen der Topologie zuständig ist.

Das folgenden Sequenzdiagramm gibt einen Überblick über die allgemeinen Abläufe im Client, nachdem eine Verbindung zum Kernel hergestellt wurde:

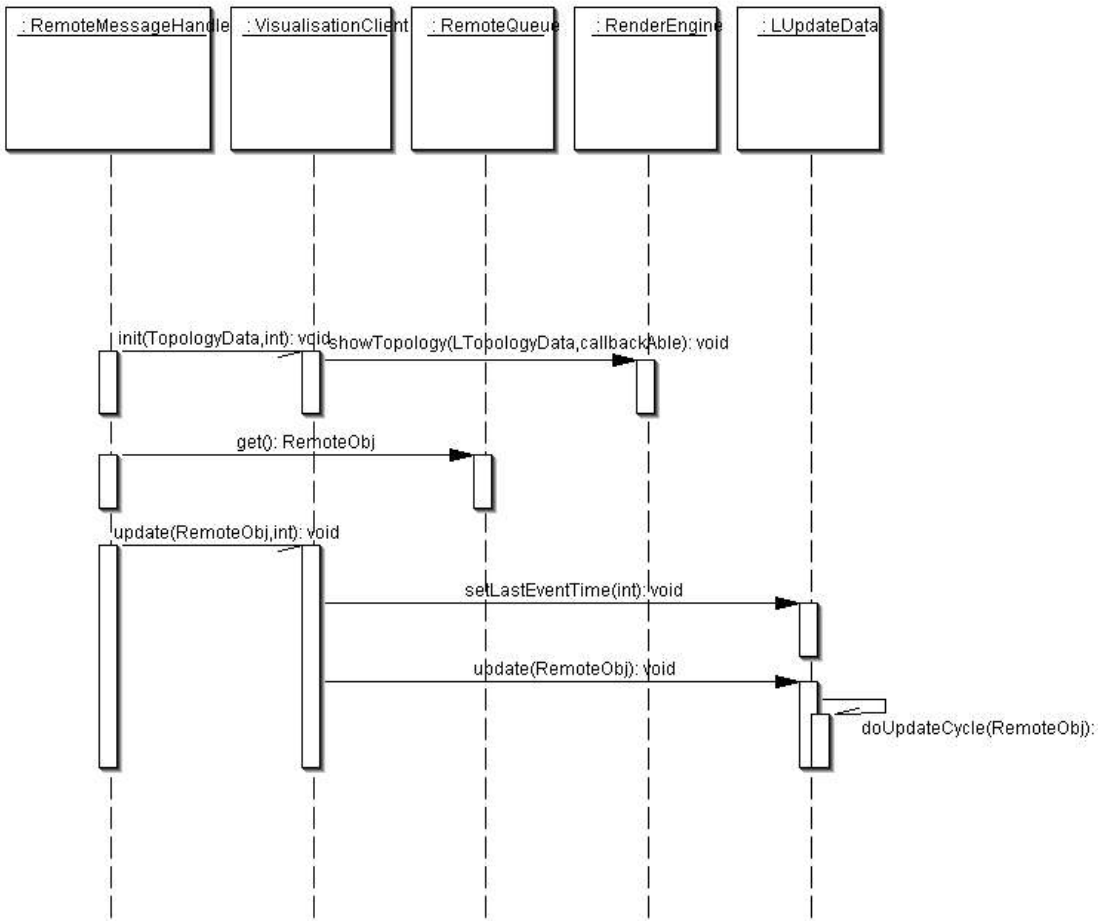

Abbildung 3.2.2.1 : Dieses Sequenzdiagramm soll die grundlegenden Abläufe im VisualisationClient darstellen, nachdem eine Verbindung zum Kernel aufgebaut wurde.

3) Klassendiagramm

Das folgende Klassendiagramm gibt einen Überblick über die Beziehungen zwischen den wichtigsten Klassen der Visualisierung:

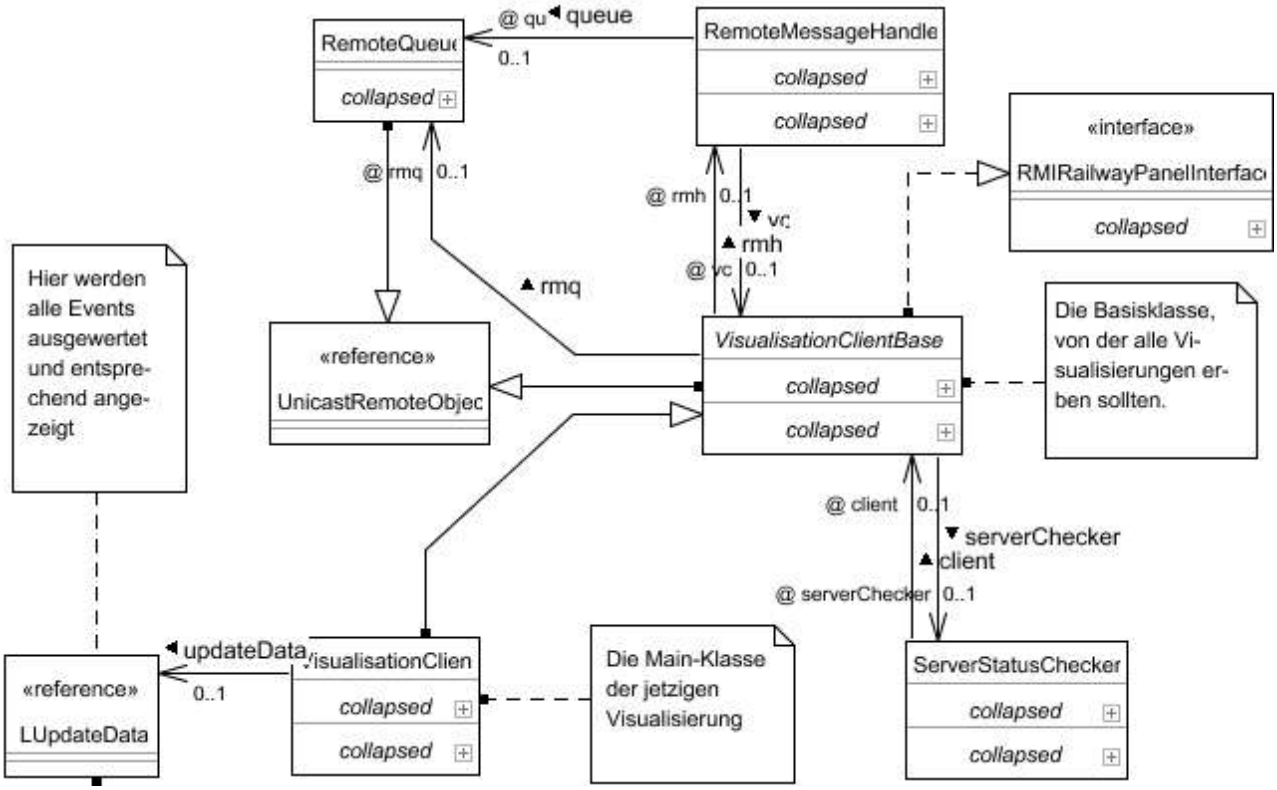

Abbildung 3.2.2.2 : Dieses Klassendiagramm verdeutlicht den Aufbau des Visualisation-Clients anhand der wichtigsten Klassen.Kommunikation zwischen Kernel und Visualisation-Client

Dieser Abschnitt erläutert detailliert die Kommunikation zwischen Kernel und Visualisierung. Es wird sowohl auf den Aufbau einer Verbindung, als auch auf die ordnungsgemäße oder unerwartete Trennung derselben eingegangen.

4) Die Kommunikation im Überblick

Die Verbindung des Visualisation-Clients mit dem Kernel wird vom User durch eine GUI aufgebaut. Hierzu wird als erstes die Methode `connectToServer` der Klasse `de.upb.swtpra.kernel.visualisation.client.VisualisationClientBase` aufgerufen.

Als Parameter bekommt der Client nun ein Objekt der Klasse `de.upb.swtpra.kernel.visualisation.Visualisation` zurück, auf dem die Methode `connect` aufgerufen werden kann.

Ist dies erfolgreich, so wird eine `RemoteQueue` erstellt, in die später der Kernel die in Absatz 3.2.1. beschriebenen Events abgelegt. Weiterhin wird ein `RemoteMessageHandler` in einem eigenen Thread gestartet, der dafür zuständig ist, ständig die in der `RemoteQueue` ankommenden Events auszulesen und an die Visualisierung zu übergeben.

Die Parameter von `connect` setzten sich zum Einen aus dem `RMIRailwayPanelInterface`

zusammen, durch das der Server an die RemoteQueue des entsprechenden Clients kommt und zum Anderen aus der Adresse des Clients.

Ist die Verbindung aufgebaut, werden vom Kernel zunächst die Topologiedaten an den Client übermittelt. Anschließend dann alle Änderungen, die während der Simulation auftreten und für den Client interessant sind. Dazu beschafft sich der Server über das `RMIRailwayPanelInterface` zunächst die `RemoteQueue` des Clients und schreibt die auftretenden Events dort hinein.

Der `RemoteMessageHandler` ist nun für das Weiterreichen dieser Events an die Visualisierung zuständig.

5) Überprüfung des Verbindungsstatus

In der Methode `connectToServer` der Klasse VisualisationClientBase wird weiterhin ein Objekt der Klasse `de.upb.swtpra.kernel.visualisation.client.ServerStatusChecker` in einem eigenen Thread gestartet. Hier wird (auf Clientseite) der Verbindungsstatus zwischen Client und Server in gewissen Zeitabständen überprüft.

6) Trennen der Verbindung

Dieser Abschnitt erläutert die unterschiedlichen Szenarien, die zu einer Trennung der Verbindung zwischen Visualisierung und Kernel führen.

Disconnect durch Server

Wird die Simulation auf der Seite des Kernels beendet, so wird am Ende der `run` Methode der Klasse `de.upb.swtpra.kernel.visualisation.Visualisation` bei jedem Client die Methode `abortConnection` aufgerufen (vgl. Abbildung 3.2.3.1).

Auf Client Seite wird in `abortConnection` nun das Verbindungsflag `connection` auf `false` gesetzt, der `RemoteMessageHandler` und `SeverStatusChecker` gestoppt und die abstrakte Methode `connectionRefused` aufgerufen.

Jede Klasse, die `de.upb.swtpra.kernel.visualisation.client.`

`VisualisationClientBase` erweitert, muss `connectionRefused` nun implementieren und weitere Schritte einleiten, was intern bei einem Verbindungsabbruch geschehen soll.

Disconnect durch Client

Bei einem Disconnect durch den (vgl. Abbildung 3.2.3.2) Client wird die Methode `disconnectFromServer` der Klasse `de.upb.swtpra.kernel.visualisation.client.VisualisationClientBase` aufgerufen. Das Verbindungsflag `connection` wird auf `false` gesetzt, der `RemoteMessageHandler` und `SeverStatusChecker` werden gestoppt und die Methode disconnect wird auf der Serverklasse `de.upb.swtpra.kernel.visualisation.Visualisation` aufgerufen.

Auf Seite des Servers wird nun der entsprechende Client einfach aus der Tabelle der sich verbundenen Clients entfernt, so dass keine Events mehr an diesen geschickt werden.

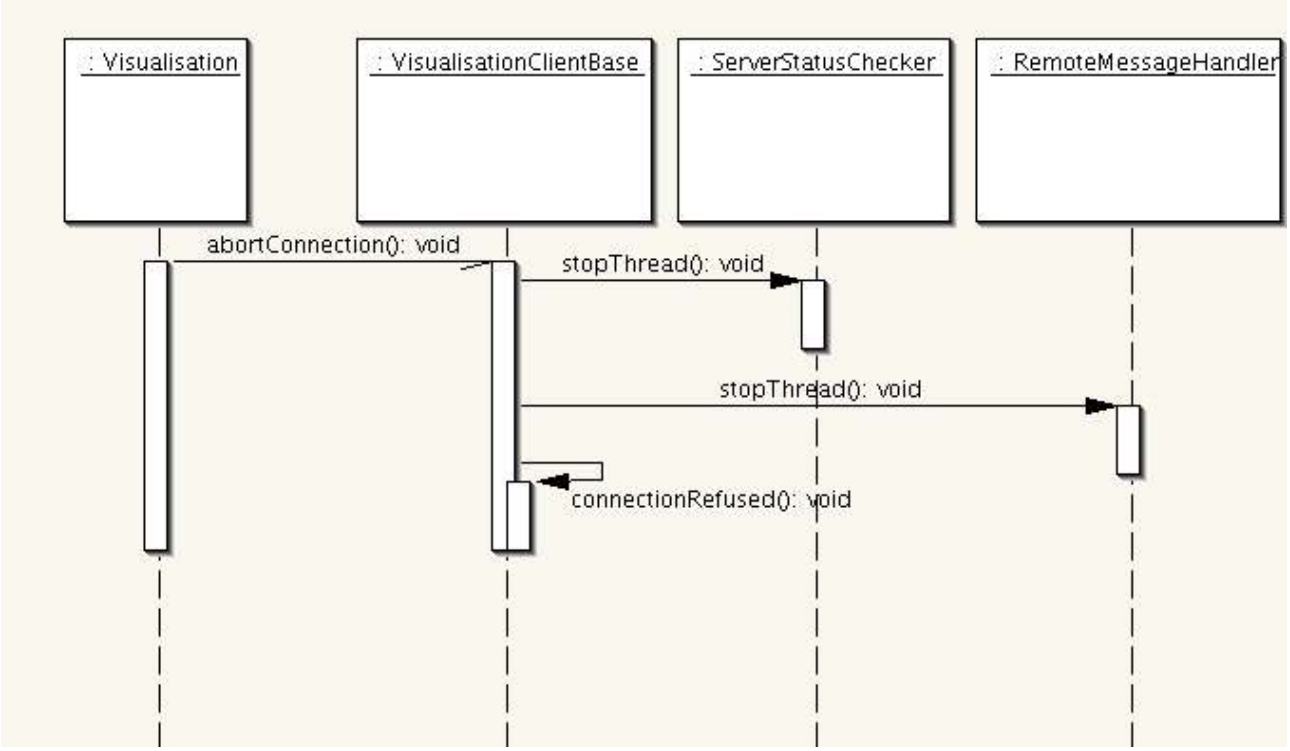

Abbildung 3.2.3.1 : Dieses Sequenzdiagramm verdeutlicht einen ordnungsgemäßen Verbindungsabbau durch den Server.

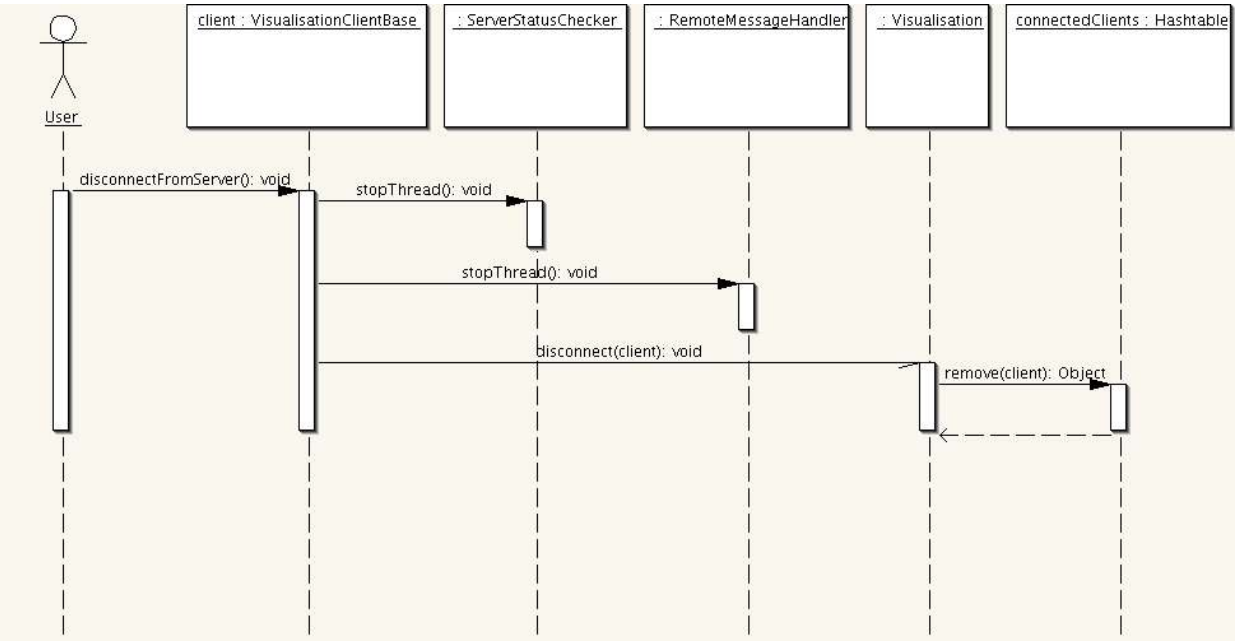

Abbildung 3.2.3.2: Dieses Sequenzdiagramm verdeutlicht einen ordnungsgemäßen Verbindungsabbau durch den Client.

Unerwarteter Disconnect

Ein unerwarteter Disconnect tritt auf, wenn z.B. eine physikalische Trennung der Verbindung von Client und Server erfolgt.

In diesem Fall wird auf der Clientseite im `ServerStatusChecker` eine `RemoteException` ausgelöst, die zur Folge hat, dass die abstrakte Funktion `connectionRefused` der Klasse `de.upb.swtpra.kernel.visualisation.client.`

`VisualisationClientBase` aufgerufen wird.

Tritt auf Serverseite beim Versuch, Events an den Client zu schicken, eine Exception auf, so wird erst versucht, mit einem Aufruf der Client-Methode `abortConnection` die Verbindung zu trennen.

In jedem Fall wird der entsprechende Client aus der Tabelle der verbundenen Clients gelöscht.

7) Klassendiagramm

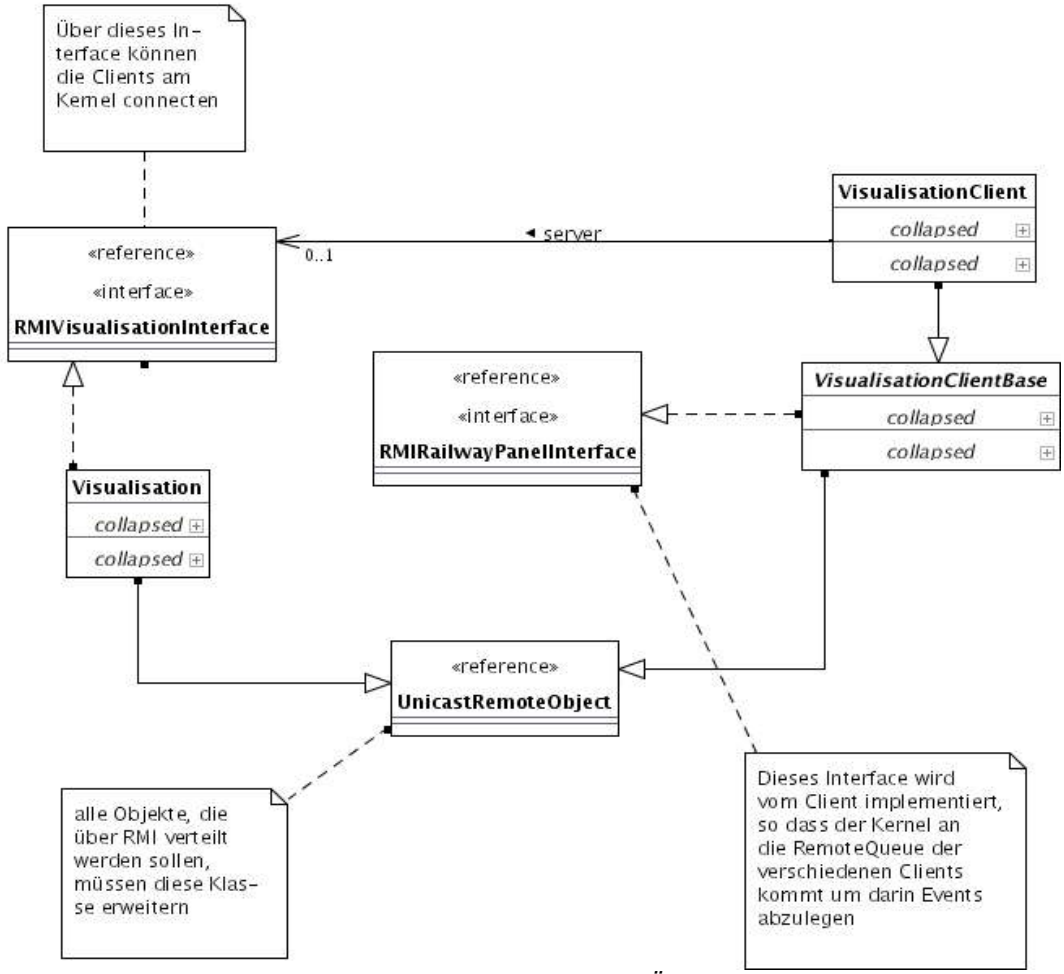

Abbildung 3.2.3.3 : Dieses Klassendiagramm gibt einen Überblick über die wesentllichen Klassen von Kernel und Visualisation-Client, die direkt an der Kommunikation beteiligt sind.

8) Sequenzdiagramm zur Kommunikation

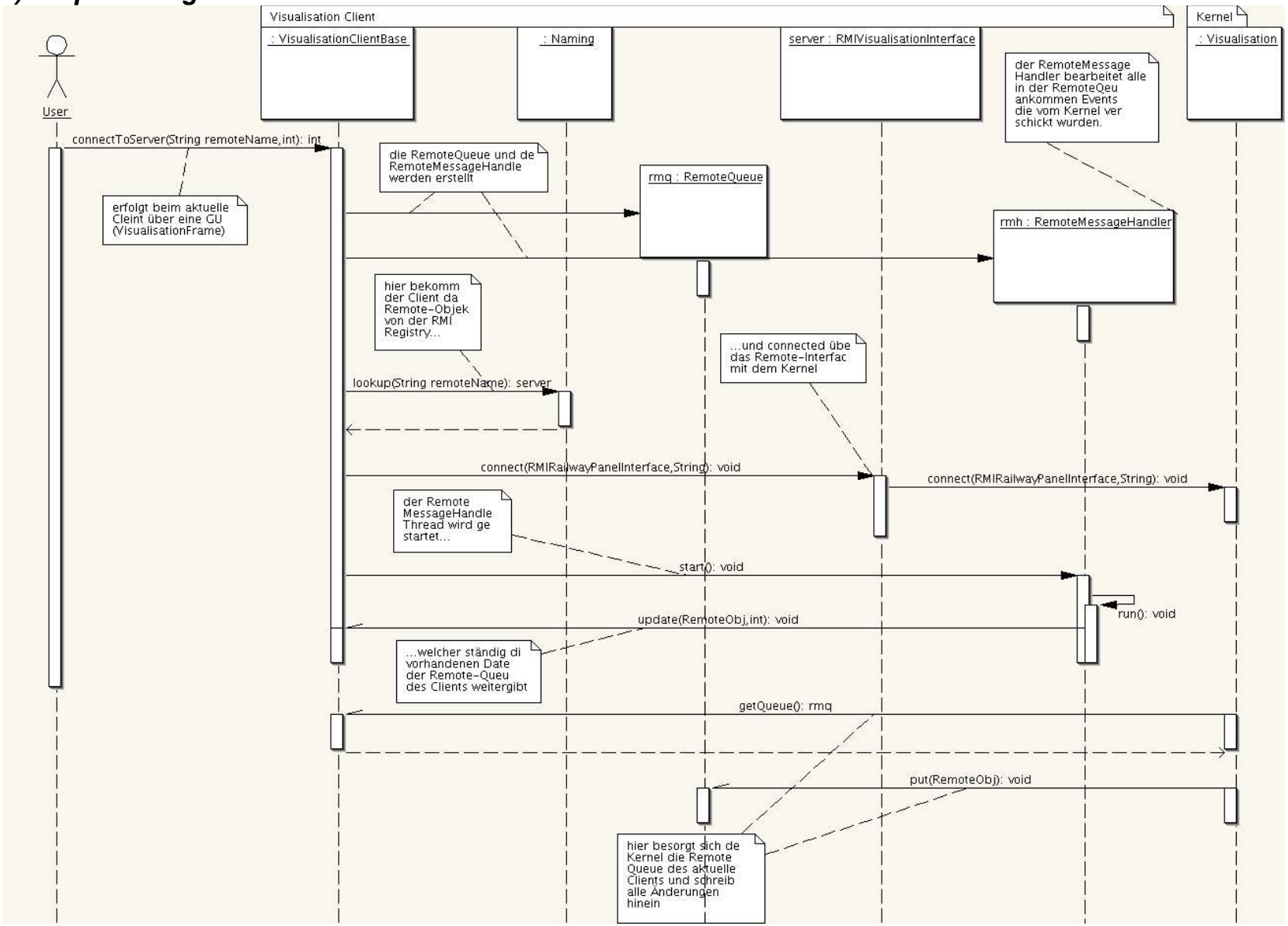

Abbildung 3.2.3.4 : Dieses Sequenzdiagramm verdeutlicht die Kommunikation zwischen Visualisation-Client und Kernel inkl. einem erfolgreichen Verbindungsaufbau.

3.3 Szenariodesigner

Ein Szenario ist eine Abfolge von Ereignissen und wird als XML-Datei standardmäßig unter dem Namen „orderScript.xml“ gespeichert. Ereignisse können Transportaufträge (order's) oder Verbindungsausfälle (disabledConnection's) sein. Diese Ereignisse treten dann zu bestimmten Zeitpunkten während der Simulation auf. In der „topologie.xml“-Datei sind alle Daten im XML-Format gespeichert die ein vollständiges Streckennetz beschreiben. Ein Streckennetz besteht aus Bahnhöfen (station's), Gleisabschnitten (track's) die Verbindungen, ggf. mit Weichen, zwischen Bahnhöfe realisieren und Weichen (switches) die Gleisabschnitte verzweigen oder vereinigen können. Beide XML-Dateien enthalten zusätzlich Konstanten, die für ein Szenario z.B den Namen der zugehörigen Topologie-Datei speichert und eine andere die Simulationszeit oder für eine Topologie z.B. wie lange das Beladen in einem Bahnhof dauert.
In den folgenden zwei Abschnitten wird nun der Aufbau dieser beiden Dateien durch UML-Diagramme und Tabellen, die die Funktionen der einzelnen Parameter erklären, beschreiben.

3.3.1 Dateibeschreibung der „orderScript“-XML Datei

In diesem Abschnitt wird der Aufbau der „orderScript“-XML Datei und die Funktion der einzelnen Parameter erläutert. Am Anfang der XML Datei steht die Document Type Definition (= DTD). In der DTD wird der Aufbau der XML-Datei formal beschrieben. Dieser Teil legt somit die Reihenfolge und den Aufbau der Daten fest. Standardmäßig wird in der DTD festgelegt, dass nach der DTD in der Datei zuerst alle Aufträge (orders) kommen, dann die Verbindungsausfälle (disabledConnections) und zum Schluß die Konstanten (constants). Das nachfolgende Diagramm zeigt von oben nach unten und von links nach rechts, wie nach der Standard-DTD, die Daten aufgebaut sind.

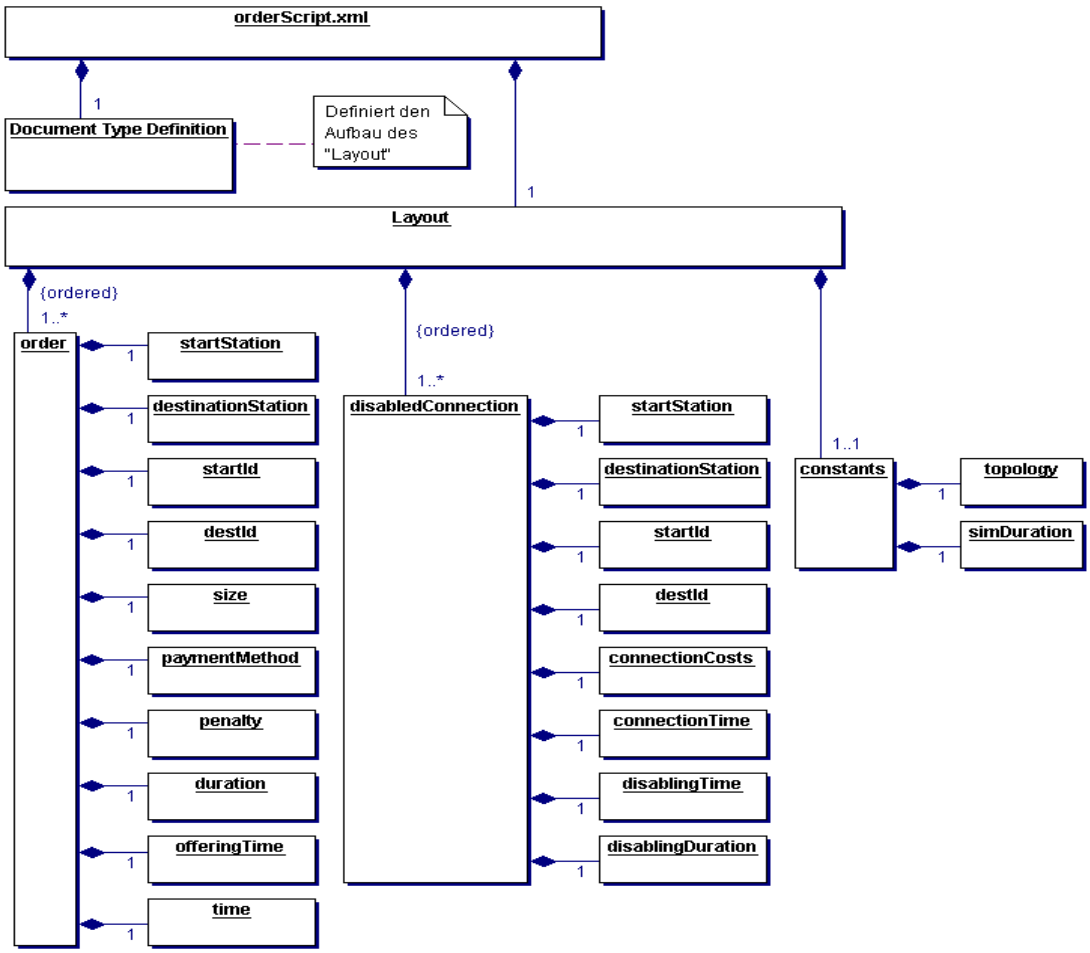

Abbildung 3.3.1.1 : Aufbau der DTD

Erklärung der Funktion der einzelnen Parameter:

order	Dies ist ein Transportauftrag und besteht aus mehreren Parametern
disabledConnection	Eine „disabledConnection“ ist ein Verbindungsausfall zwischen zwei Bahnhöfen (station's) und besteht aus mehreren Parametern
constants	„constants“ sind Konstanten, davon gibt es zwei (siehe: topology und simDuration)
startStation	Name eines Startbahnhofs (einer „station“), der Parametertyp ist: String
destinationStation	Name eines Zielbahnhofs (einer „station“), der Parametertyp ist: String
startId	ID eines Startbahnhofs (einer „station“), der Parametertyp ist: integer und gehört immer genau zu einem startStation
destId	ID eines Zielbahnhofs (einer „station“), der Parametertyp ist: integer und gehört immer genau zu einer destinationStation
size	Gibt die Personenanzahl an, die für den Auftrag transportiert werden muß, der Parametertyp ist: integer

paymentMethod	Beschreibt die Zahlungsmethode für einen Auftrag, zulässige Werte sind „Credit Card“ und „Invoice“, allerdings wird bis jetzt nur „Credit Card“ verwendet.
penalty	Dieser Wert gibt die Höhe der Strafe an, die bei Nichterfüllung des Auftrags anfällt, der Parametertyp ist: integer
duration	Gibt die Länge des Simulationszeitraum an, in dem der Auftrag nach der Erteilung ausgeführt sein muß, der Parametertyp ist: integer
offeringTime	Gibt die Länge des Simulationszeitraums an, in der Angebote für die Aufträge abgegeben werden können, der Parametertyp ist: integer
time / disablingTime	Zu diesem Zeitpunkt tritt das Ereignis in der Simulation auf, der Parametertyp ist: integer. Bei Aufträgen (order's) ist der Parameter „time“ und bei disabledConnection ist er „disablingTime“.
connectionCosts	Die Kosten der ausgefallenen Verbindung, diese ergeben sich aus den Einzelkosten der zur Verbindung gehörigen Gleisabschnitte (track's) und Weichen (switches), der Parametertyp ist: integer
connectionTime	Die Länge der ausgefallenen Verbindung, diese ergibt sich aus den Einzellängen der zur Verbindung gehörigen Gleisabschnitte (track's) und Weichen (switches), der Parametertyp ist: integer
disablingDuration	Gibt die Länge des Verbindungsausfalls in Simulationszeit an, der Parametertyp ist: integer
topology	Diese Konstante enthält den Namen der zugehörigen Topologie-Datei, der Parametertyp ist: String
simDuration	Gibt den Zeitpunkt an, zu dem die Simulation endet, der Parametertyp ist: integer

3.3.2 Dateibeschreibung der „topologies“-XML Datei

Dateibeschreibung der „topologies“-XML Dateien

In diesem Abschnitt wird der Aufbau der „topologies“-XML Datei und die Funktion der einzelnen Parameter erläutert. Am Anfang der XML Datei steht die Document Type Definition (= DTD). In der DTD wird der Aufbau der XML-Datei formal beschrieben. Dieser Teil legt somit die Reihenfolge und den Aufbau der Daten fest. Standardmäßig wird in der DTD festgelegt, dass nach der DTD in der Datei zuerst eine Liste aller Startbahnhöfe (startingPoints) kommt, falls man diese expliziet angeben möchte. Danach kommen die Konstanten (constants), dann die Bahnhöfe (station's), die Weichen (switches) und zum Schluß die Gleisabschnitte (track's). Das nachfolgende Diagramm zeigt von oben nach unten und von links nach rechts, wie nach der Standard-DTD, die Daten aufgebaut sind.

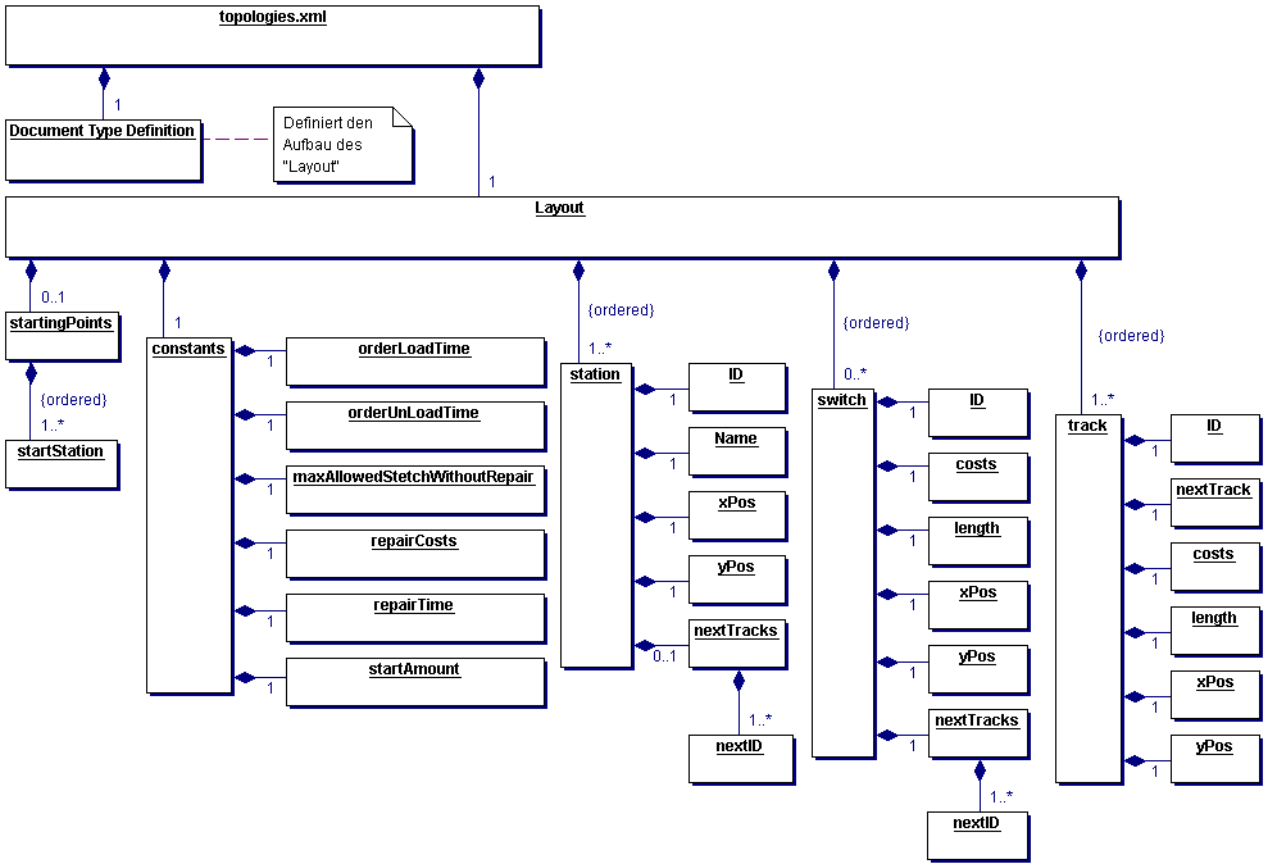

Abbildung 3.3.1.2 : Aufbau der DTD

Erklärung der Funktion der einzelnen Parameter:

startingPoints	Dies ist eine Liste vom möglichen Bahnhöfen (station's), die beim Simulationsstart mit Shuttle's besetzt werden dürfen, der Parametertyp ist eine Liste von startStation's.
startStation	ID eines Startbahnhofs (einer „station“), der Parametertyp ist: integer
constants	„constants“ sind Konstanten, davon gibt es mehrere (siehe Diagramm)
orderLoadTime	Gibt die Beladezeit eines Shuttles in Simulationszeit an, der Parametertyp ist: integer
orderUnLoadTime	Gibt die Entladezeit eines Shuttles in Simulationszeit an, der Parametertyp ist: integer
maxAllowedStretch WithoutRepair	Der Wert gibt die maximale Anzahl von Gleisabschnitten an, die ein Shuttle ohne Reparatur zurücklegen kann, der Parametertyp ist: integer
repairCosts	Der Wert gibt die Kosten für eine Shuttle-Reparatur an, der Parametertyp ist: integer
repairTime	Der Wert gibt die Zeit an, die für eine Reparatur benötigt wird, der Parametertyp ist: integer

startAmount	Dies ist das Startguthaben der Untermehmen zu beginn einer Simulation, der Parametertyp ist: integer
station	Dies ist ein Bahnhof, dieser wird durch mehrere Parameter beschreiben (siehe Diagramm)
ID	Die ID identifiziert einen Bahnhof (station), eine Weiche (switch) oder einen Gleisabschnitt (track) eindeutig, der Parametertyp ist: integer
Name	Name des Bahnhofs (station), der Parametertyp ist: String
xpos / yPos	Diese Parameter beschreiben die horizontale und vertikale Position eines Bahnhofs (station's), einer Weiche (switch) oder eines Gleisabschnittes (track's) in dem Streckennetz, der Parametertyp ist jeweils: integer
nextTracks	Dies ist eine Liste mit den ID's von nachfolgenden Bahnhöfen (station's), Gleisabschnitten (track's) oder Weichen (switches)
nextID / nextTrack	Dies ist die ID eines Bahnhofs (station's), Gleisabschnittes (track's) oder einer Weiche (switch), der Parametertyp ist: integer
costs	Gibt die Anzahl der Geldeinheiten an, die beim Befahren eines Gleisabschnittes (track's) oder einer Weiche (switch) bezahlt werden müssen, der Parametertyp ist: integer
length	Dieser Wert gibt die Länge (der Simulationszeit zum Befahren) des Gleisabschnittes (track's) oder der Weiche (switch) an, der Parametertyp ist: integer

4. Produktfunktionen

Im Folgenden werden die von der Software verlangten Grundfunktionalitäten dargestellt. Die verschiedenen Anwendungsfälle der Softwarekomponenten werden untersucht. Software muss vorgegebene Funktionalitäten erfüllen, die sich aus dem vorgesehenen Aufgabenbreich ableiten lässt. Der Ablauf dieser Funktionalitäten und ihre Verbindung zur Umwelt einer Software wird nun näher erläutert. Dies geschieht mittels Use Case Diagrammen, Beschreibung und Ablaufplan. Die Beschreibung der geforderten Funktionalitäten bildet die Basis für eine nähere Beschäftigung mit der Komponentenstruktur im Verlauf der Grobanalyse. Die bereits beim Reverse Engineering gemachte Trennung zwischen den einzelnen Softwarekomponenten wird auch im Weiteren beibehalten.

4.1 Shuttle-Steuerung

Die Shuttle Steuerung ist nur mit einem Nutzer konfrontiert, mit dem Kernel. Wie in Abbildung 4.1.1 ersichtlich können sechs Anwendungsfälle unterschieden werden. Der Kernel tritt meist in der Rolle des Maklers auf. Die Shuttle Steuerung ist mit allen Aspekten der Auftragskoordination beschäftigt.

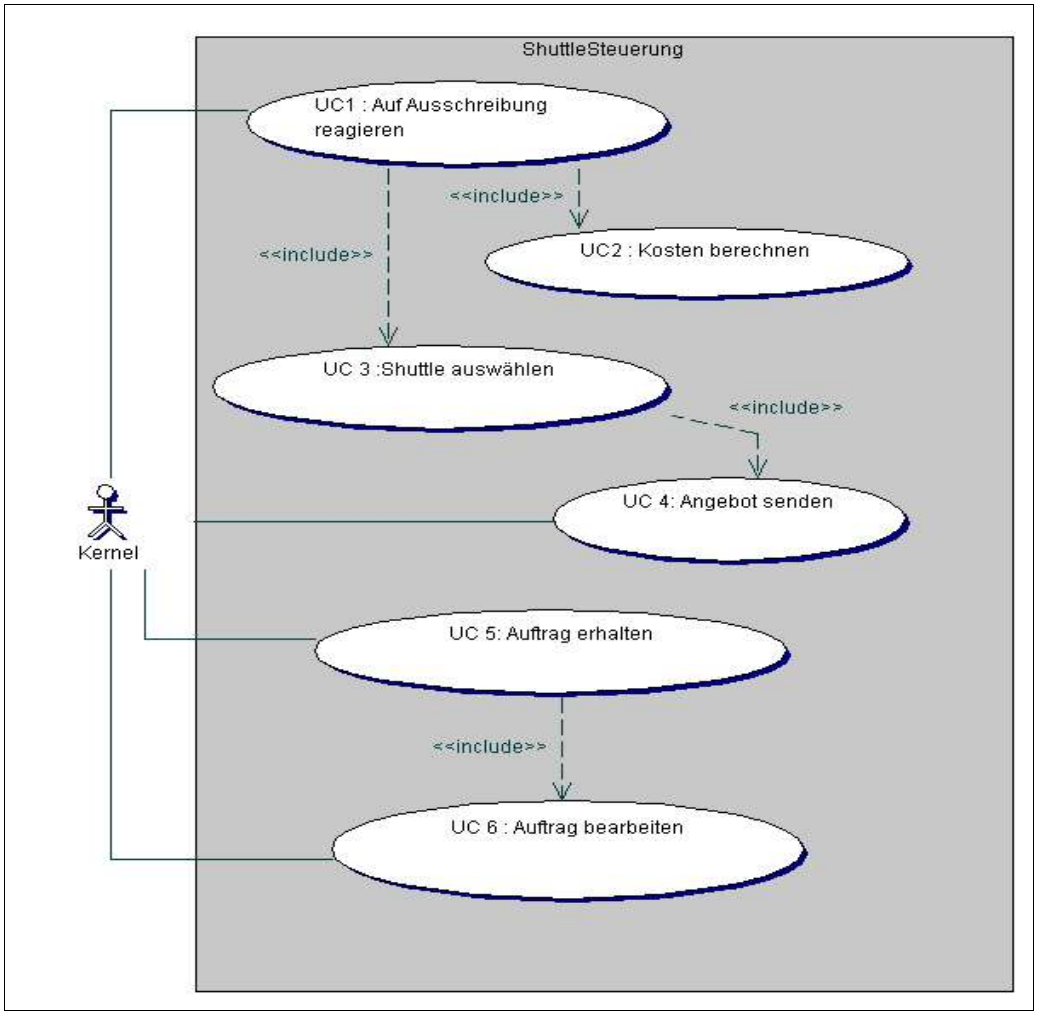

Abbildung 4.1.1 : UseCase Diagramm für die SchuttleStrg mit dem BrokerAgent als User

1) Beschreibung zu UC1 : Auf Ausschreibung reagieren

Charakterisierende Informationen

Die Shuttlesteuerung muss reagieren, wenn eine Ausschreibung vorliegt.

Ziel des Use Cases:	Shuttle nimmt Ausschreibung entgegen
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Kernel hat einen Auftrag ausgeschrieben
Nachbedingung bei erfolgreicher Ausführung:	Das Shuttle hat die Ausschreibung erhalten
Beteiligte Nutzer:	Kernel
Auslösendes Ereignis:	Kernel sendet Auftragsdaten

Szenario für den Standardablauf (Erfolg)

Sobald der Kernel einen Auftrag ausgeschrieben hat, liest die Shuttlesteuerung die Auftragsdaten ein.

Beschreibung des allgemeinen Ablaufs

Auf den Erhalt von Auftragsdaten durch den Kernel reagiert die Shuttlesteuerung mittels einlesen der Daten, berechnen der Kosten und auswählen des Shuttles (vgl. Abbildung 4.1.1).

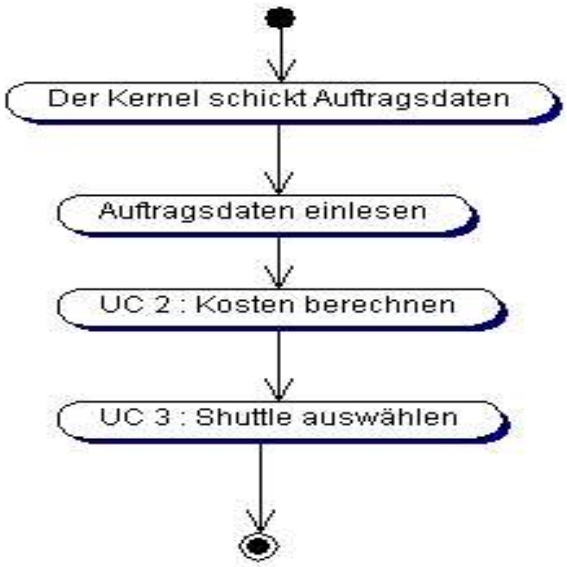

Abbildung 4.1.1: Aktivitätendiagramm für den Use Case 1 : Auf Ausschreibung reagieren

2) Beschreibung zu UC2 : KostenBerechung

Charakterisierende Informationen

Berechnet die Kosten, welche bei der Erfüllung des ausgeschriebenen Auftrags anfallen würden.

Ziel des Use Cases:	Shuttle berechnet die Kosten
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Shuttle hat Auftragsdaten eingelesen
Nachbedingung bei erfolgreicher Ausführung:	Shuttle hat Kosten berechnet
Beteiligte Nutzer:	
Auslösendes Ereignis:	Das Shuttle hat die Auftragsdaten eingelesen

Szenario für den Standardablauf (Erfolg)

Nachdem der Kernel einen Auftrag ausgeschrieben hat, berechnet der Shuttle den günstigsten Weg (Kombination von Verbindungen) von seiner jetzigen Position zum Startbahnhof und von dort zum Zielbahnhof und berechnet daraus die entstehenden Kosten. Dabei muss es berücksichtigen, dass eventuell andere, an der Shuttle vergebene, Aufträge zuerst bearbeitet werden müssen.

Alternative Abläufe für die Szenarien

Da es auch Aufträge gibt, die nicht erfüllbar sind, muss der Shuttle zwischen erfüllbaren und nicht erfüllbaren Aufträgen unterscheiden. Angenommene, nicht erfüllbare Aufträge führen zwangsläufig zu einer Konventionalstrafe.

Beschreibung des allgemeinen Ablaufs

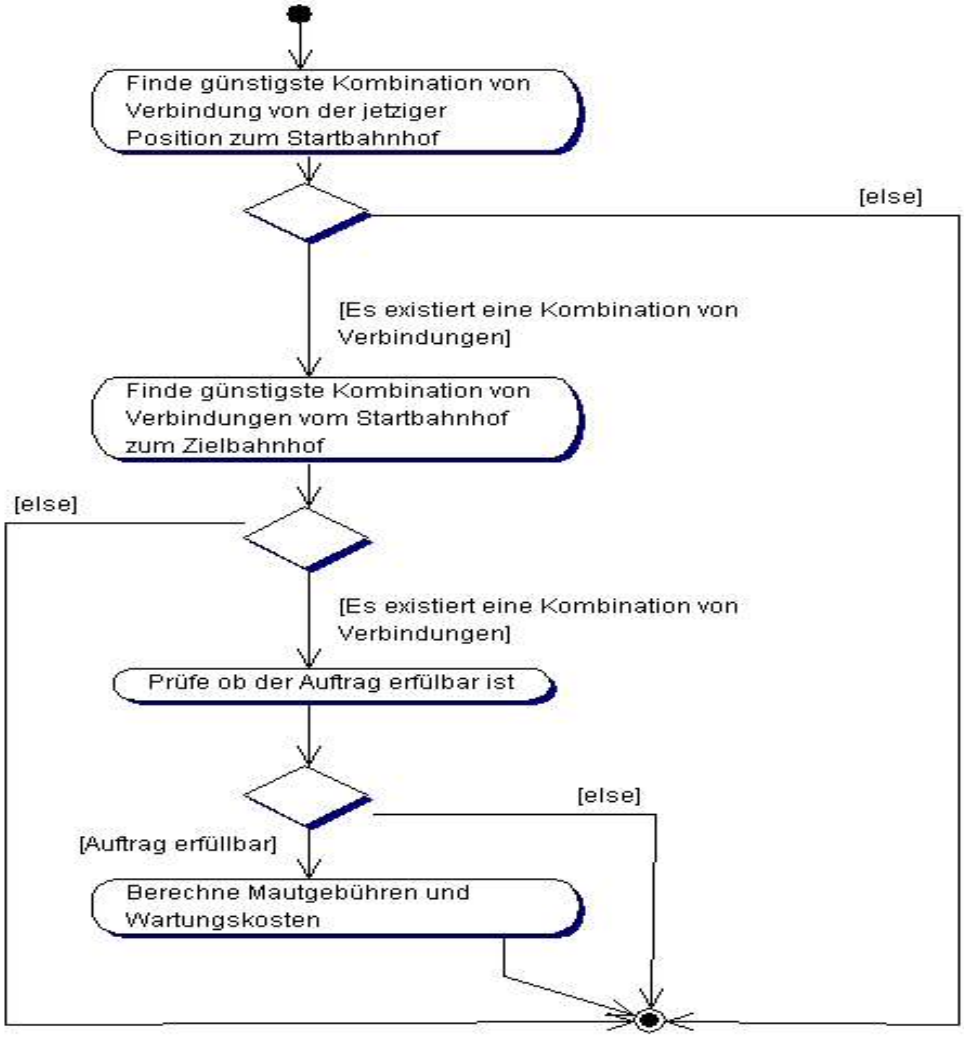

Abbildung 4.1.2: Aktivitätendiagramm für den Use Case 2: Kostenberechnung

3) Beschreibung zu UC3 : Shuttle auswählen

Charakterisierende Informationen

Die Shuttles eines Unternehmens bestimmen welches von ihnen ein Angebot senden soll.

Ziel des Use Cases:	Shuttle, welches ein Angebot senden soll, auswählen
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Shuttles haben Ausschreibung erhalten
Nachbedingung bei erfolgreicher Ausführung:	Genau ein Shuttle wurde ausgewählt und ist bereit, ein Angebot an den Kernel zu senden
Beteiligte Nutzer:	
Auslösendes Ereignis:	Das Shuttle hat eine Ausschreibung erhalten

Szenario für den Standardablauf (Erfolg)

Zuerst vergleichen die Shuttles eines Unternehmens ihre errechneten Kosten. Zusätzlich wird geprüft, ob es noch Shuttles ohne ausgeführten Auftrag gibt, da ein Unternehmen disqualifiziert wird, sofern nicht jeder Shuttle seiner Flotte wenigstens einen Auftrag ausgeführt hat. Danach wird ein Shuttle ausgewählt.

Alternative Abläufe für die Szenarien

Falls ein Auftrag nicht erfüllbar ist, und die Shuttles dies bemerkt haben, berechnen sie keine Kosten.

Beschreibung des allgemeinen Ablaufs

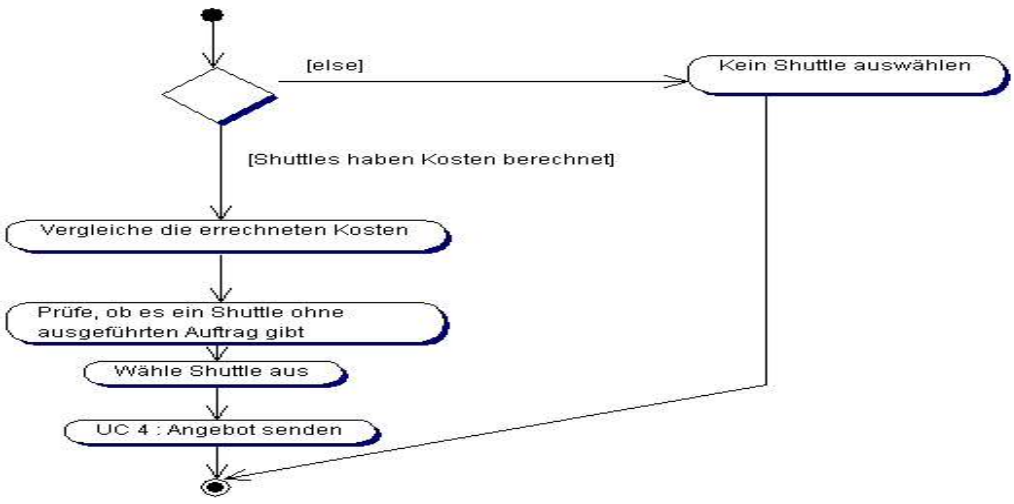

Abbildung 4.1.3: Aktivitätendiagramm für den Use Case 3: Shuttle auswählen

4) Beschreibung zu UC4 : Angebot senden

Charakterisierende Informationen

Das zuvor ausgewählte Shuttle sendet sein Angebot an den Kernel.

Ziel des Use Cases:	Das Shuttle sendet ein Angebot an den Kernel
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Innerhalb eines Unternehmens wurde ein Shuttle ausgewählt
Nachbedingung bei erfolgreicher Ausführung:	Der Kernel hat von einem Unternehmen genau ein Angebot erhalten
Beteiligte Nutzer:	Kernel
Auslösendes Ereignis:	Ein Shuttle wurde ausgewählt

Szenario für den Standardablauf (Erfolg)

Der Shuttle sendet sein Angebot an den Kernel. Es darf nur ein Shuttle pro Unternehmen ein Angebot senden.

Beschreibung des allgemeinen Ablaufs

Ist ein Angebot generiert, so wird dieses an den Kernel versandt (vgl. Abbildung 4.1.4)

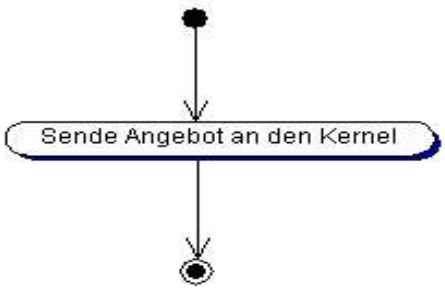

Abbildung 4.1.4: Aktivitätendiagramm für den Use Case 4 : Sende Angebot an den Kernel

5) Beschreibung zu UC5 : Auftrag erhalten

Charakterisierende Informationen

Der Shuttle erhält vom Kernel einen Auftrag für den es ein Angebot abgegeben hat.

Ziel des Use Cases:	In diesem UseCase wird das Shuttle vom Kernel über einen ihm zugeteilten Auftrag informiert. Dieser wird nun in die Liste der auszuführenden Aufträge hinzugefügt.
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Das Shuttle muss das günstigste Angebot an den Kernel gesendet haben.
Nachbedingung bei erfolgreicher Ausführung:	Es existiert ein zu bearbeitender Auftrag
Beteiligte Nutzer:	Kernel
Auslösendes Ereignis:	Das Shuttle erhält vom Kernel einen Auftrag

Szenario für den Standardablauf (Erfolg)

In diesem UseCase erhält der Shuttle vom Kernel einen Auftrag, für den es ein Angebot abgegeben hat. Dieser Auftrag wird dann in eine Liste aufgenommen, die der Shuttle abarbeitet. Hierbei kann es mehrere Aufträge parallel ausführen, wenn im Shuttle noch genug Platz für weitere Passagiere ist.

Beschreibung des allgemeinen Ablaufs

Die Behandlung eines erhaltenen Auftrags wird in Abbildung 4.1.5 dargestellt.

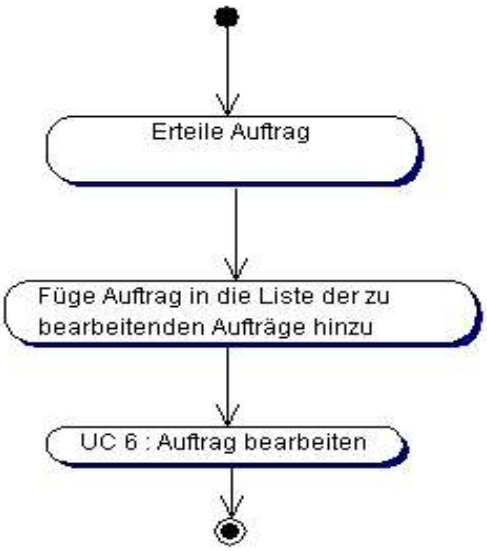

Abbildung 4.1.5: Aktivitätendiagramm für den Use Case 5 : Auftrag erhalten

Beschreibung zu UC6 : Auftrag bearbeiten

Charakterisierende Informationen

Der Shuttle bearbeitet einen Auftrag aus seiner Auftragsliste.

Ziel des Use Cases:	Das Shuttle bearbeitet einen Auftrag aus seiner internen Liste der ihm zugeteilten Aufträge.
Umgebende Systemgrenze:	ShuttleSteuerung
Vorbedingung:	Das Shuttle muss einen Auftrag haben
Nachbedingung bei erfolgreicher Ausführung:	Der Auftrag wurde bearbeitet
Beteiligte Nutzer:	
Auslösendes Ereignis:	Auftrag erhalten

Szenario für den Standardablauf (Erfolg)

In diesem UseCase gehen wir davon aus, dass der Shuttle einen einzelnen Auftrag bearbeitet. Es holt die Passagiere an einem Bahnhof ab und fährt sie zu einem anderen. In der Simulation können sich Aufträge später überschneiden, so dass z.B. zwei Mal Passagiere aufgenommen und dann erst beide Gruppen am gleichen Bahnhof abgesetzt werden.

Alternative Abläufe für die Szenarien

Die alternativen Abläufe beschreiben die Möglichkeit, das Strecken ausfallen und Shuttles den Bahnhof nur durch eine Umleitung oder gar nicht mehr erreichen können. Wenn sich der Shuttle beim Ausfall eines Streckenabschnitts gerade auf diesem befindet, kann es einfach weiter fahren.

Beschreibung des allgemeinen Ablaufs

Abbildung 4.1.6 verdeutlicht die Abarbeitung eines Auftrags. Dabei werden die alternativen Abläufe mitberücksichtigt.

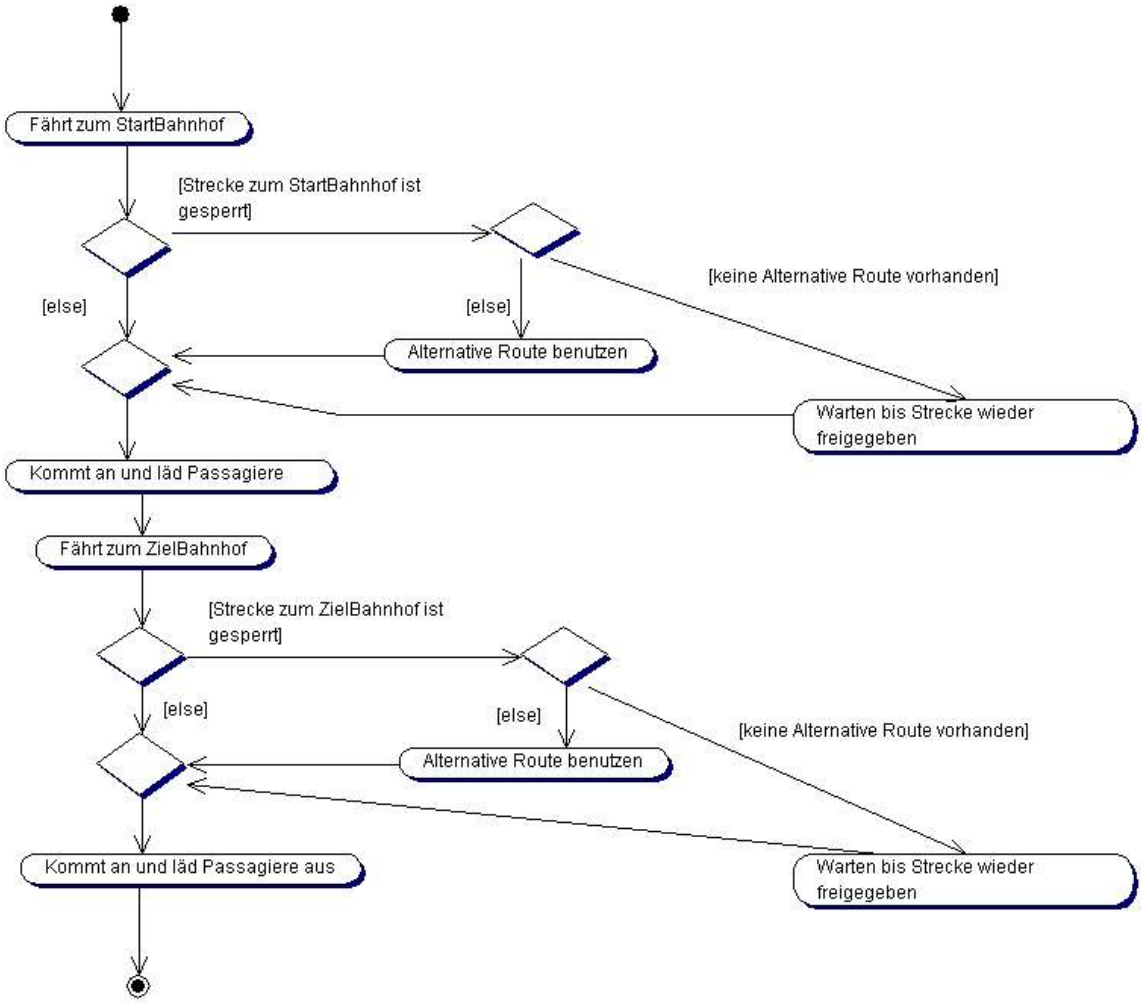

Abbildung 4.1.6: Aktivitätendiagramm für den Use Case 6 : Auftrag bearbeiten

4.2.Unternehmensvisualisierung

Im folgenden Abschnitt werden die Produktfunkitionen der Unternehmens Visualisierung aufgeführt. Die Unternehmens Visualisierung realisiert die in der Abbildung 4.1.2.1 dargestellten Use Cases. Als externe Nutzer treten hierbei einmal der `User` auf, welcher die Person darstellt, welche die Unternehmens Visualisierung benutzt, und der `Kernel`, welcher die vorhandene Simulationsumgebung darstellt. Die einzelnen Use Cases werden in den folgenden Abschnitten detailliert beschrieben.

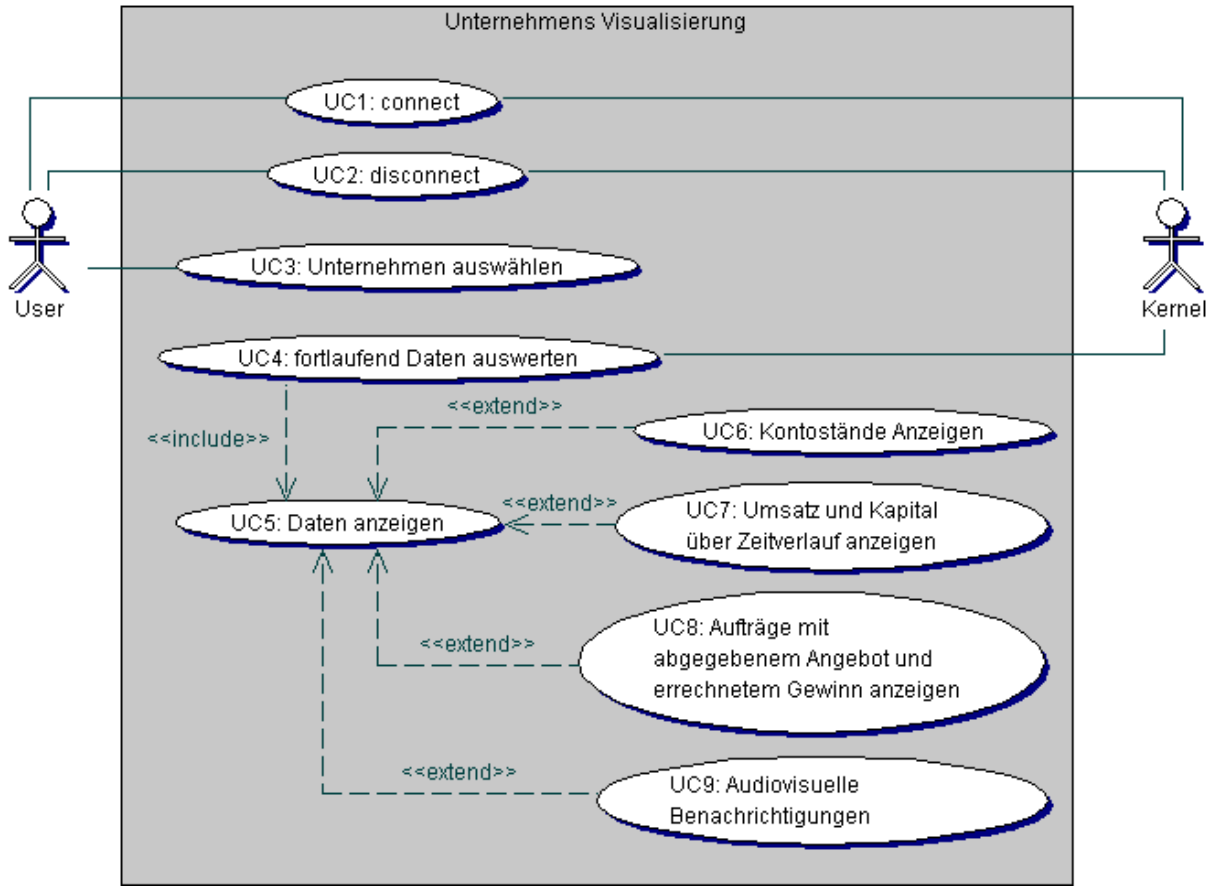

Abbildung 4.1.2.1 : Use Case Diagramm für die Visualisierug

GUI für die Visualisierung

Abbildung 4.1.2.2 zeigt eine Gesamtübersicht über die GUI der Unternehmens Visualisierung. Die einzelnen Elemente werden in der Beschreibung der Use Cases genauer erläutert.

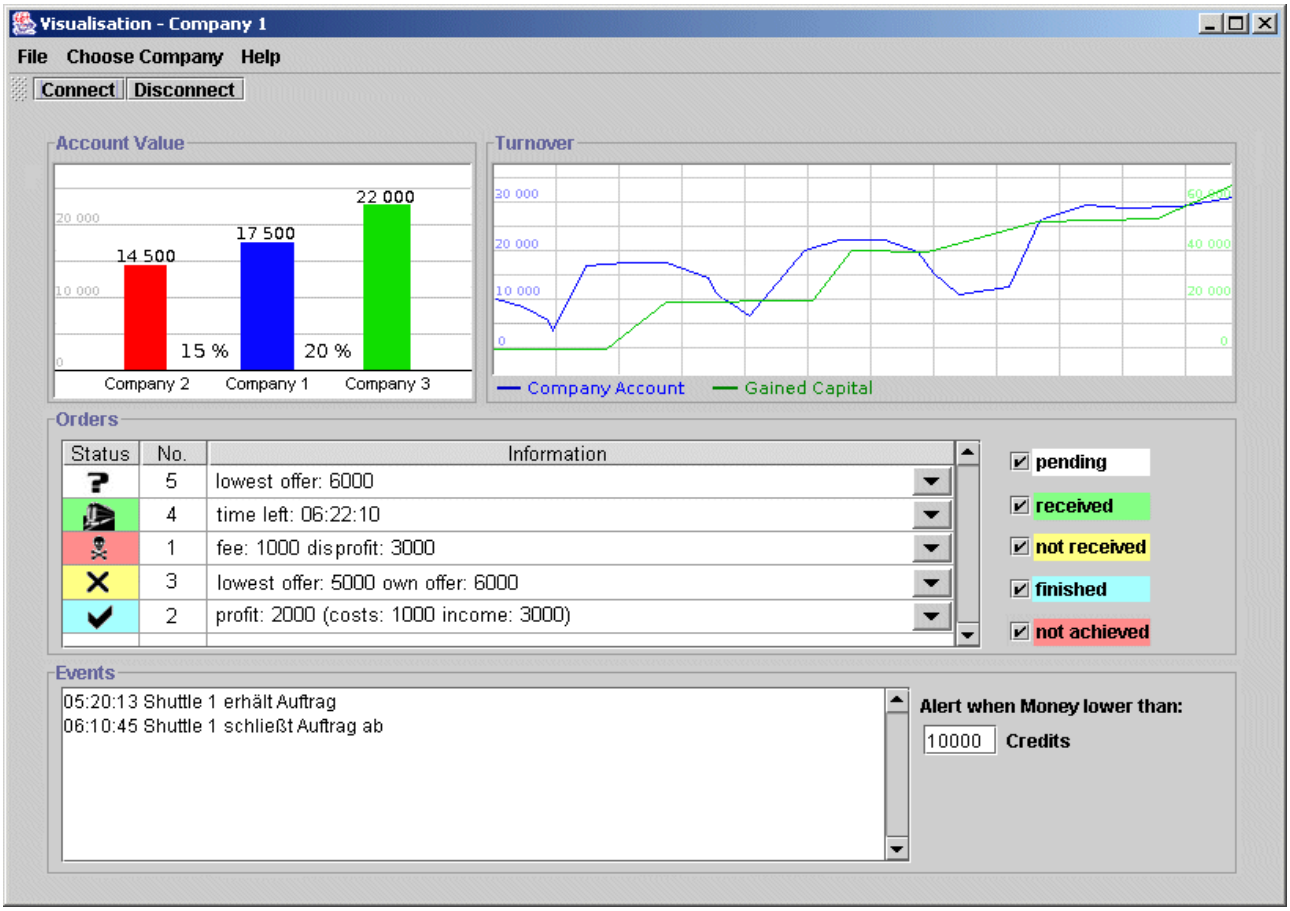

Abbildung 4.1.2.2 : GUI für die Visualisierug

1) Beschreibung zu UC1: connect

Charakterisierende Informationen

Der User kann in einer Dialog Box eine IP für einen Server eingeben. Die Visualisierung versucht sich dann mit dem Simulations-Kernel zu verbinden, um Daten von ihm zu empfangen.

Ziel des Use Cases:	Die Unternehmens Visualisierung mit dem Kernel verbinden.
Umgebende Systemgrenze:	Unternehmens Visualisierung
Vorbedingung:	Unternehmens Visualisierung gestartet.
Nachbedingung bei erfolgreicher Ausführung:	Die Unternehmens Visualisierung ist mit dem Kernel verbunden, hat die Topologiedaten empfangen und ist bereit, weitere Daten entgegen zu nehmen.
Beteiligte Nutzer:	User, Kernel
Auslösendes Ereignis:	User will die Unternehmens Visualisierung mit dem Kernel verbinden.

Beschreibung des allgemeinen Ablaufs

Die folgende Abbildung 4.1.2.3 zeigt den Ablauf des UC1: connect. Es wird zuerst eine Dialogbox für die IP-Abfrage geöffnet. Die Unternehmens Visualisierung verbindet sich dann mit dem Kernel und empfängt darauf die Topologiedaten der verwendeten Strecke. Schlägt die Verbindung zum Kernel fehl, so wird der User darüber informiert und aufgefordert eine neue IP einzugeben.

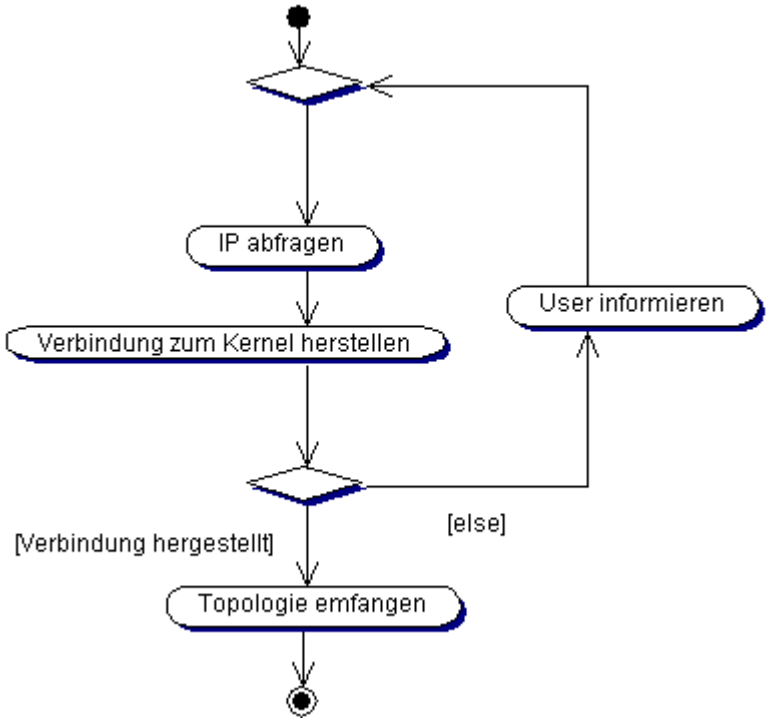

Abbildung 4.1.2.3 : Ablauf des UC1: Connect

GUI für den Ablauf

In dem in Abbildung 4.1.2.4 gezeigten Fenster kann der Nutzer die IP eingeben, unter der der Kernel zu finden ist.

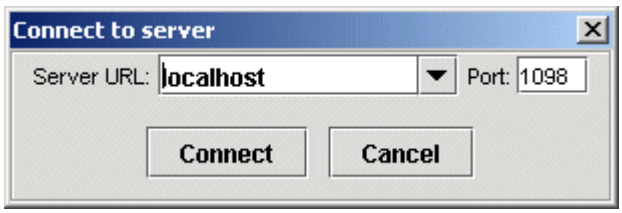

Abbildung 4.1.2.4 : GUI für UC1: Connect

2) Beschreibung zu UC2: disconnect

Charakterisierende Informationen

Der User muss zum einen die Verbindung zum Kernel trennen können. Zum anderen muss die Unternehmens Visualisierung darauf reagieren, wenn die Verbindung vom Kernel oder unerwartet getrennt wurde.

Ziel des Use Cases:	Die Verbindung zum Kernel abbrechen, bzw schließen.
Umgebende Systemgrenze:	Unternehmens Visualisierung
Vorbedingung:	Verbindung zum Kernel hergestellt.
Nachbedingung bei erfolgreicher Ausführung:	Die Unternehmens Visualisierung hat die Verbindung zum Kernel getrennt , stellt dies dar und ist bereit sich erneut mit einem Kernel zu verbinden.
Beteiligte Nutzer:	User, Kernel
Auslösendes Ereignis:	Der User will die Verbindung zwischen Unternehmens Visualisierung und dem Kernel trennen, die Verbindung wird vom Kernel getrennt oder sie wird physikalisch getrennt.

Beschreibung des allgemeinen Ablaufs

Im Normalfall beendet der User die Verbindung. Alternativ ist es zum einen möglich, dass der Kernel die Verbindung beendet, zum anderen kann die Verbindung durch äußere Einflüsse getrennt werden. Diese Funktionalität wird in Abbildung 4.1.2.5 dargestellt.

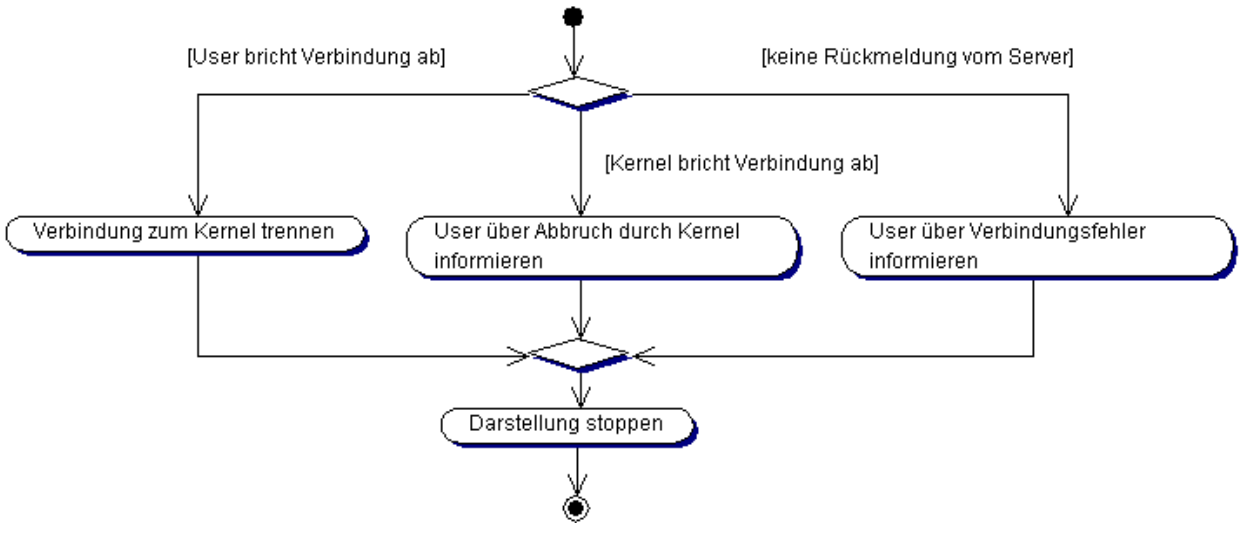

Abbildung 4.1.2.5 : Allgemeiner Ablauf des UC2: Disconnect

3) Beschreibung zu UC3: Unternehmen Auswählen

Charakterisierende Informationen

Da mehrere Unternehmen an einer Simulation im Kernel beteiligt sein können, kann der User das Unternehmen auswählen, welches dargestellt werden soll.

Ziel des Use Cases:	Die Daten für das ausgewählte Unternehmen darstellen.
Umgebende Systemgrenze:	Visualisierung
Vorbedingung:	Verbindung zum Kernel hergestellt, mehrere Unternehmen an der Simulation beteiligt.
Nachbedingung bei erfolgreicher Ausführung:	Die Daten für das ausgewählte Unternehmen werden Dargestellt.
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User wählt ein vom aktuellen Unternehmen verschiedenes aus.

Beschreibung des allgemeinen Ablaufs

Abbildung 4.1.2.6 verdeutlicht den allgemeinen Ablauf des UC3: Unternehmen Auswählen. Der User wählt ein Unternehmen aus, für welches die Daten dargestellt werden sollen, worauf diese dann dargestellt werden.

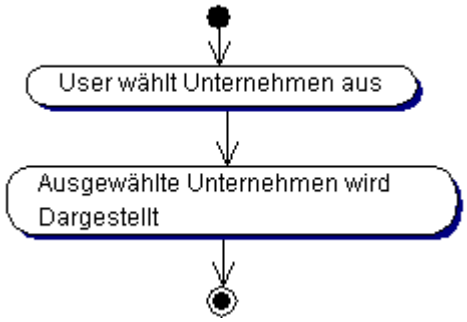

Abbildung 4.1.2.6 : Allgemeiner Ablauf des UC3: Unternehmen auswählen

Gui für den Ablauf

Die Darstellung wird durch ein Pulldown Menü realisiert, wie in Abbildung 4.1.2.7 zu sehen ist. In dieses Menü werden die an der Simulation des Kernels teilnehmenden Unternehmen eingefügt. Das aktuell dargestellte Unternehmen ist in der Titelleiste des Fensters zu sehen(siehe Abbildung 4.1.2.2).

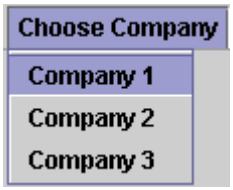

Abbildung 4.1.2.7: Gui Für den UC3: Unternehmen auswählen

4) Beschreibung zu UC4: fortlaufend Daten auswerten

Charakterisierende Informationen

Die Daten des Kernels werden Objekt für Objekt an die Unternehmens Visualisierung geschickt. Diese Daten müssen in interne Datenstrukturen übernommen, aufbereitet und dargestellt werden.

Ziel des Use Cases:	Die Daten, welche von dem Kernel geschickt werden, zu empfangen und auszuwerten.
Umgebende Systemgrenze:	Unternehmens Visualisierung
Vorbedingung:	Verbindung zum Kernel hergestellt.
Nachbedingung bei erfolgreicher Ausführung:	Die Daten werden dargestellt.
Beteiligte Nutzer:	Kernel
Auslösendes Ereignis:	Kernel schickt Daten.

Beschreibung des allgemeinen Ablaufs

Abbildung 4.1.2.6 zeigt den allgemeinen Ablauf des UC4: fortlaufend Daten auswerten. Zuerst treffen die Daten vom Kernel bei der Unternehmens Visualisierung ein, dann werden diese in interne Datenstrukturen übernommen und darauf die Anzeige aktualisiert.

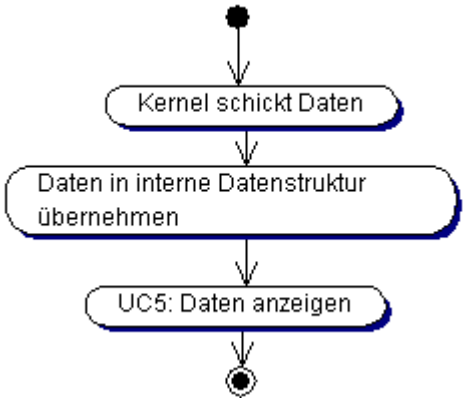

Abbildung 4.1.2.8 : Allgemeiner Ablauf des UC4: fortlaufend Daten auswerten

5) Beschreibung zu UC5: Daten anzeigen

Charakterisierende Informationen

Es wird überprüft ob genügend Daten vorhanden sind. Wenn ja, werden sie dargestellt, wenn nicht, wird dies kenntlich gemacht.

Ziel des Use Cases:	Die Daten aus der internen Datenstruktur darzustellen.
Umgebende Systemgrenze:	Unternehmens Visualisierung
Vorbedingung:	Verbindung zum Kernel hergestellt, es wurden Daten empfangen
Nachbedingung bei erfolgreicher Ausführung:	Die Daten sind dargestellt.
Beteiligte Nutzer:	-
Auslösendes Ereignis:	Es wurden neue Daten eingelesen.

Beschreibung des allgemeinen Ablaufs

Abbildung 4.1.2.9 beschreibt den Ablauf des UC5: Daten anzeigen. Sind die Daten zur Darstellung vorhanden, so werden sie dargestellt, ansonsten wird der User darüber informiert, dass die Daten nicht ausreichen.

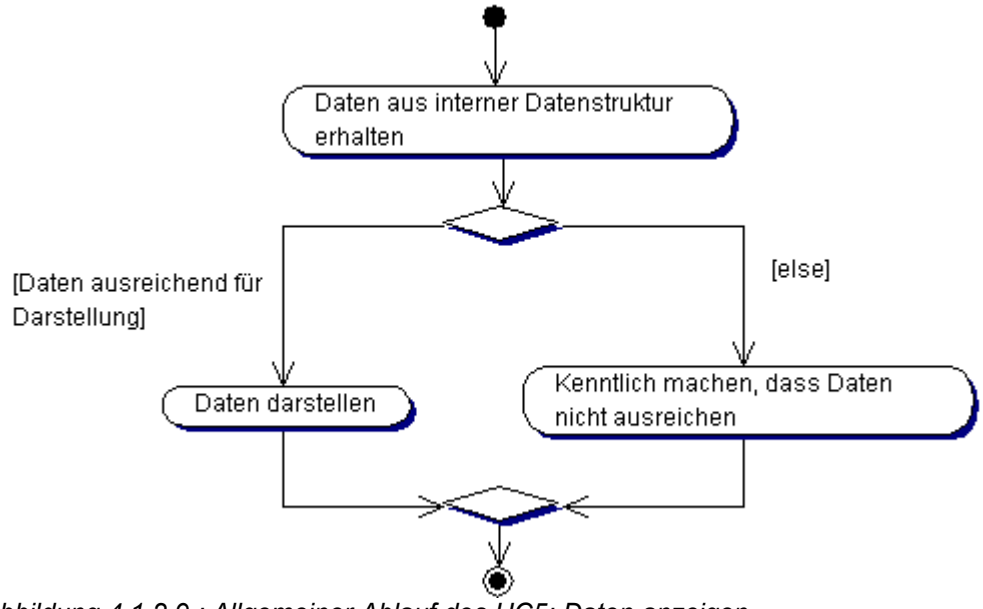

Abbildung 4.1.2.9 : Allgemeiner Ablauf des UC5: Daten anzeigen

6) Beschreibung zu UC6: Kontostände anzeigen

Charakterisierende Informationen

Es wird der Kontostand des aktuellen, des besten und schlechtesten als Balkendiagramm dargestellt, sowie die prozentuale Abweichung des aktuellen Unternehmens zum Besten und Schlechtesten.

Gui für den Ablauf

Abbildung 4.1.2.10 zeigt die Balkendiagramme, wobei der Mittlere den Kontostand des aktuell dargestellten Unternehmens zeigt, der linke den des niedrigsten Unternehmens und der rechte den des Höchsten.

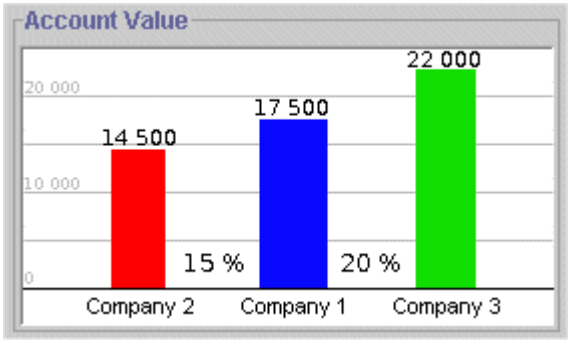

Abbildung 4.1.2.10 : GUI für UC6: Kontostände anzeigen

7) Beschreibung zu UC7: Umsatz und Kapital über Zeitverlauf anzeigen

Charakterisierende Informationen

Es werden die Einnahmen der Shuttles, sowie das Gesamtkapital des aktuellen Unternehmens als Liniendiagramm dargestellt.

Gui für den Ablauf

Abbildung 4.1.2.11 zeigt das Liniendiagramm für die Einnahmen(Gained Capital) und den Kontostand(Company Account). Die Messlatte für beide Kurven wird unabhängig angepasst.

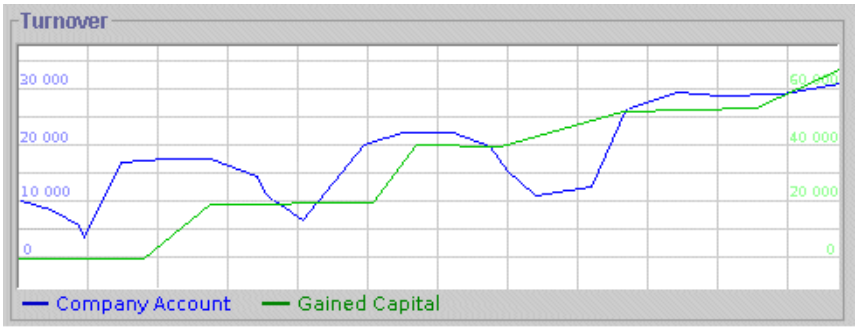

Abbildung 4.1.2.11 : GUI für UC7: Umsatz und Kapital über Zeitverlauf anzeigen

8) Beschreibung zu UC8: Aufträge mit abgegebenem Angebot und errechnetem Gewinn anzeigen

Charakterisierende Informationen

Es werden die ausgeschriebenen, erhaltenen, nicht erhaltenen und abgeschlossenen Aufträge der Shuttles angezeigt, mit Informationen über die abgegebenen Angebote, sowie der aktuelle Status der Aufträge. Es kann nach den oben genannten Kriterien gefiltert werden.

Gui für den Ablauf

Abbildung 4.1.2.12 zeigt die Liste zur Auftragsdarstellung. Es kann nach dem Status, in dem sich ein Auftrag befindet, gefiltert werden. Je nach Status, in dem sich ein Auftrag befindet, werden für den entsprechenden Status relevanten Informationen dargestellt. Zusätzlich soll noch bei erhaltenen Aufträgen der Shuttle, welches den Auftrag bearbeitet, dargestellt werden, was in Abbildung 4.1.2.12 noch nicht zu sehen ist.

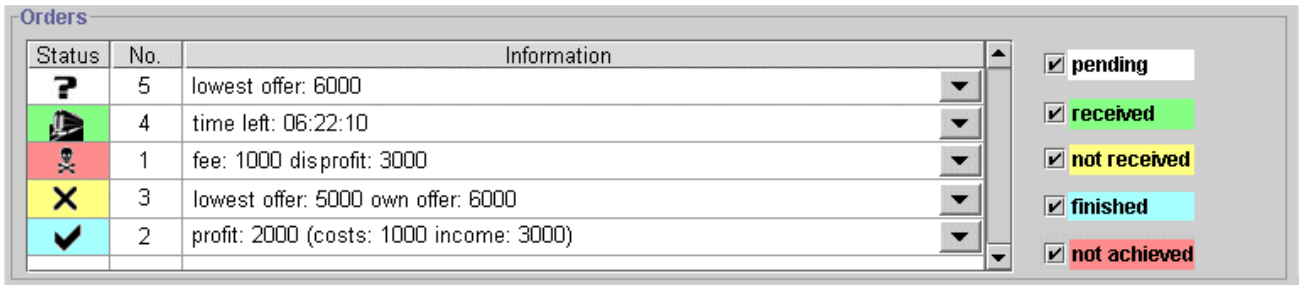

Abbildung 4.1.2.12 : GUI für UC8: Aufträge mit abgegebenem Angebot und errechnetem Gewinn anzeigen

9) Beschreibung zu UC9: Audiovisuelle Benachrichtigungen

Charakterisierende Informationen

Es werden akustische Signale und Meldungen über eine Konsole ausgegeben bei:

- Erhalt eines Auftrags
- Erfüllung eines Auftrags
- Nicht-Erfüllung eines Auftrags

- Unterschreiten eines zur Laufzeit konfigurierbaren Schwellenwertes des Unternehmenskapitals (Schwellenwert kann in eine Textbox eingestellt werden)

Gui für den Ablauf

Die Ereignisse werden über eine Textkonsole ausgegeben, was in Abbildung 4.1.2.13 zu sehen ist.

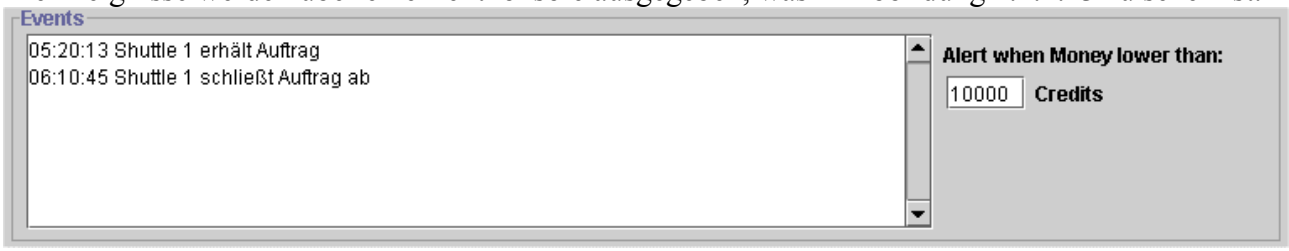

Abbildung 4.1.2.13 : GUI für UC9: Audiovisuelle Benachrichtigungen

4.3. Szenariodesigner

Wie aus Abbildung 4.1.3.1 ersichtlich wird lassen sich für den Szenariodesigner acht verschiedene Anwendungsfälle unterscheiden. Diese werden nun genauer untersucht.

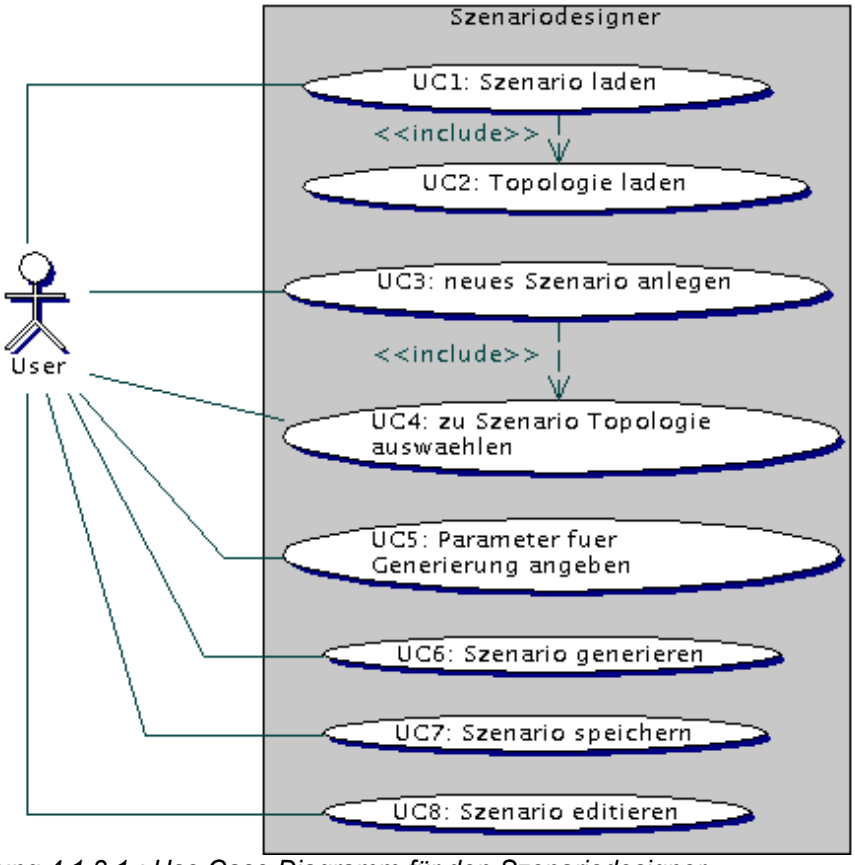

Abbildung 4.1.3.1 : Use Case Diagramm für den Szenariodesigner

1) Beschreibung zu UC1 : Szenario laden

Charakterisierende Informationen

Der User möchte ein vorhandenes Szenario in den Szenariodesigner laden können.

Ziel des Use Cases:	ein vorhandenes Szenario wird geladen
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodesigner gestartet
Nachbedingung bei erfolgreicher Ausführung:	eine Szenariodatei ist geladen
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will Szenario laden

Beschreibung des allgemeinen Ablaufs

Zunächst muss die bestehende Szenario Datei ausgewählt werden. Der entsprechende Dialog ist in Abbildung 4.1.3.3 dargestellt. Die weiteren Schritte von der Verifizierung der Datei bis zum Anzeigen des Szenarios lassen sich Abbildung 4.1.3.2 entnehmen.

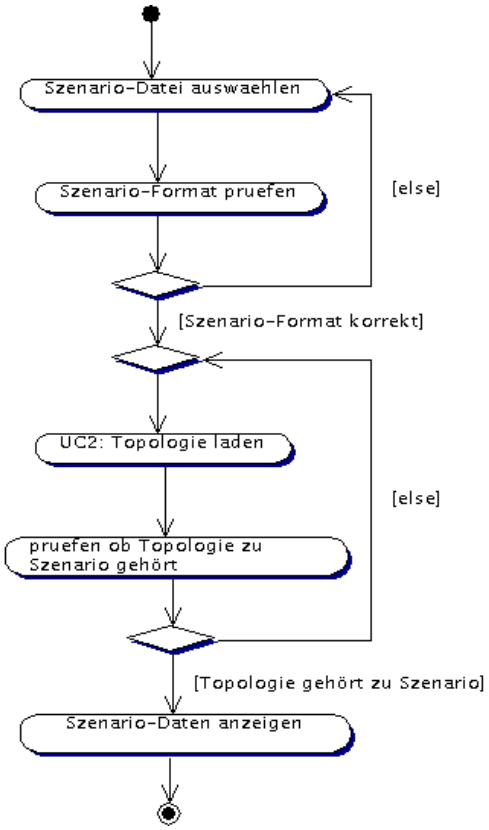

Abbildung 4.1.3.2 : Aktivitätendiagramm für Use Case 1

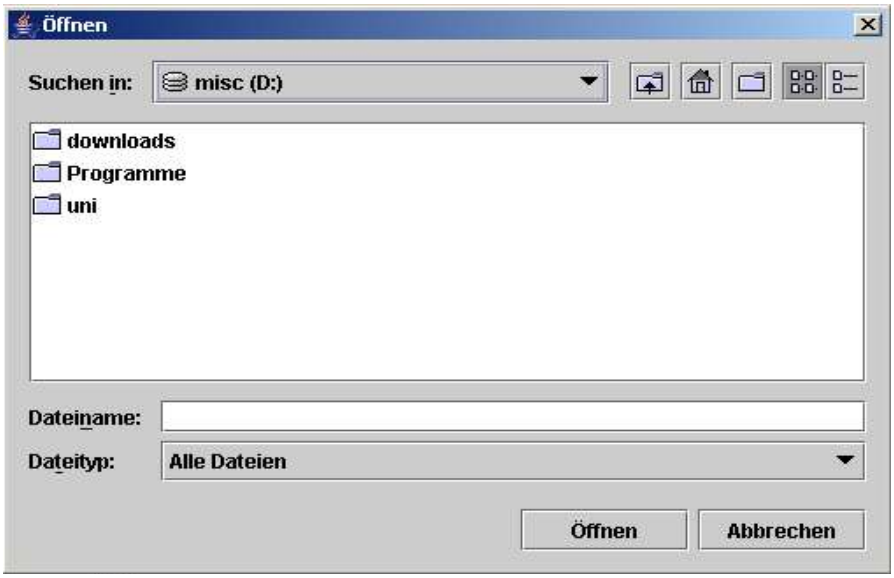

Abbildung 4.1.3.3 : GUI zum Öffnen einer Datei

2) Beschreibung zu UC2 : Topologie laden

Charakterisierende Informationen

Zu dem in Use Case 1 (UC1) ausgewählten Szenario wird vom Szenariodesigner die zugehörige Topologie geladen.

Ziel des Use Cases:	Topologiedatei zum aktuellen Szenario laden
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodatei geladen
Nachbedingung bei erfolgreicher Ausführung:	Die zugehörige Topologiedatei ist im Szenariodesigner geladen
Beteiligte Nutzer:	User
Auslösendes Ereignis:	Der User lädt ein bestehendes Szenario

Beschreibung des allgemeinen Ablaufs

In Abbildung 4.1.3.4 wird der Ablauf der Zugriffs- und Formatprüfung dargestellt.

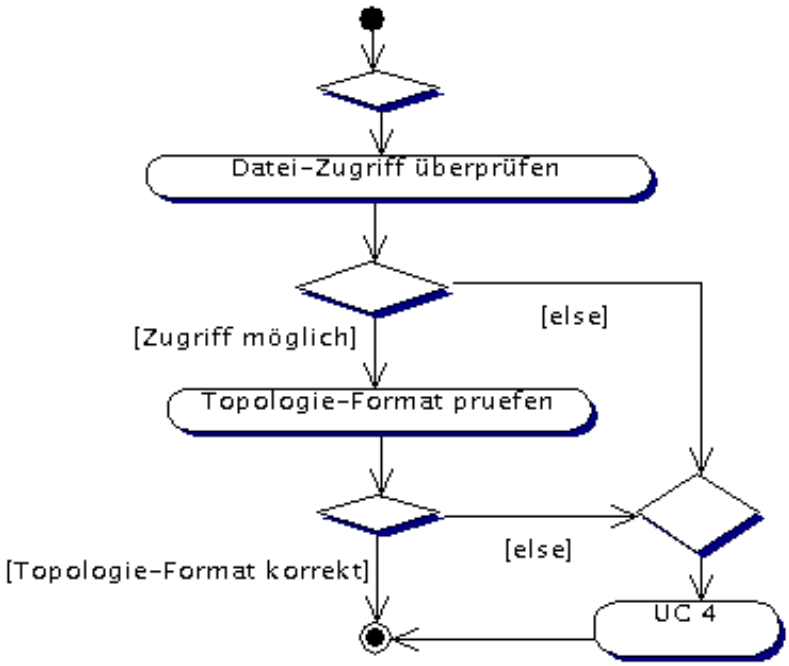

Abbildung 4.1.3.4 : Aktivitätendiagramm für Use Case 2

3) Beschreibung zu UC3 : neues Szenario anlegen

Charakterisierende Informationen

Ein neues Szenario wird angelegt.

Ziel des Use Cases:	ein neues leeres Szenario wird erstellt
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodesigner gestartet.
Nachbedingung bei erfolgreicher Ausführung:	Eine leere Szenariodatei ist geladen
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will neues Szenario erstellen

Beschreibung des allgemeinen Ablaufs

Das Anlegen eines neuen Szenarios wird in Abbildung 4.1.3.5 dargestellt. Eine wichtige Rolle nimmt dabei der „UC4: zu Szenario Topologie auswählen“ ein.

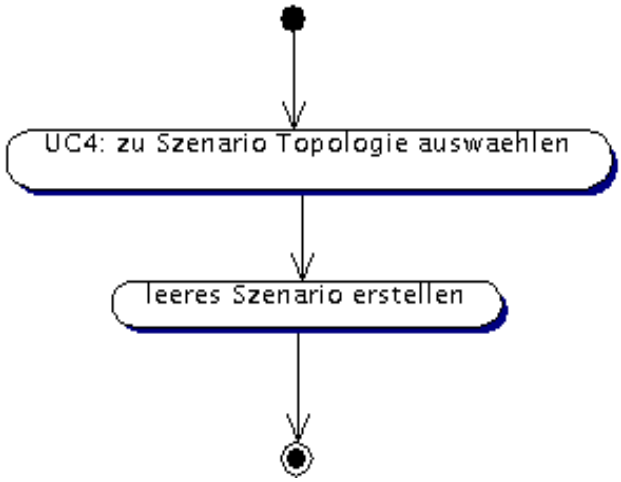

Abbildung 4.1.3.5 : Aktivitätendiagramm für Use Case 3

4) Beschreibung zu UC4 : zu Szenario Topologie auswaehlen

Charakterisierende Informationen

Zu einem Szenario wird vom User eine zugehörige Topologie ausgewählt.

Ziel des Use Cases:	eine Topologiedatei einem Szenario zuordnen
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodatei geladen
Nachbedingung bei erfolgreicher Ausführung:	eineTopologiedatei wurde einer Szenariodatei zugeordnet
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will neues Szenario erstellen oder User läd eine Szenariodatei deren zugehörige Topologiedatei Fehler aufweist.

Beschreibung des allgemeinen Ablaufs

Wird eine Topologie ausgewählt, so ist der korrekte Ablauf des Protokolls vom Format der Topologie abhängig. Falls eine Datei falschens Format gewählt wurde, wird der User aufgefordert eine neue Datei auszuwählen (vgl. Abbildung 4.1.3.6)

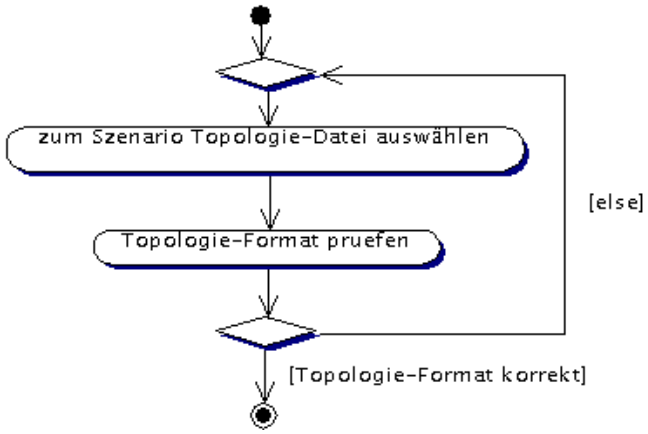

Abbildung 4.1.3.6 : Aktivitätendiagramm für Use Case 4

5) Beschreibung zu UC5 : Parameter für Generierung angeben

Charakterisierende Informationen

Es werden die folgenden Parameter, die für die Generierung eines Szenarios benötigt werden, eingegeben:

- Wert für die mittlere Dauer der Ausschreibungen
- Wert für die Auftragsgröße
- Werte für die relativen Anteile von Paaren von Start- und Zielbahnhöfen
- Wert für die Konventionalstrafe
- Wert für die Ausschreibungsfrist
- Wert für die Häufigkeit von Verbindungsausfällen
- Wert für die Simulationsdauer und den Wert für die Auftragsdauer

Ziel des Use Cases:	Parameter für die Generierung angeben
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodesigner gestartet
Nachbedingung bei erfolgreicher Ausführung:	Es sind für alle Parameter, die für die automatische Szenariogenerierung benötigt werden, gültige Werte vorhanden
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will für die automatische Szenariogenerierung die notwendigen Parameter eingeben.

Beschreibung des allgemeinen Ablaufs

Die Eingabe der Paramter erfolgt nach dem in Abbildung 4.1.3.7 dargestellten Muster und wird mittels eines GUI(siehe Abbildung 4.1.3.8) durchgeführt. Falls der User mit seinen Eingaben die zulässigen Wertebereiche überschreitet, wird er aufgefordert diese zu korrigieren. Die einzugebeben Daten werden mit Tooltip näher erläutert.

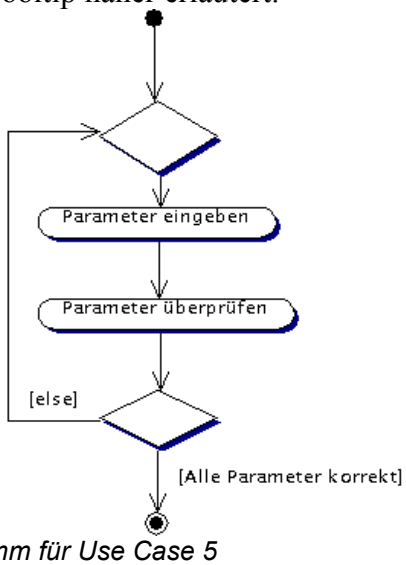

Abbildung 4.1.3.7 : Aktivitätendiagramm für Use Case 5

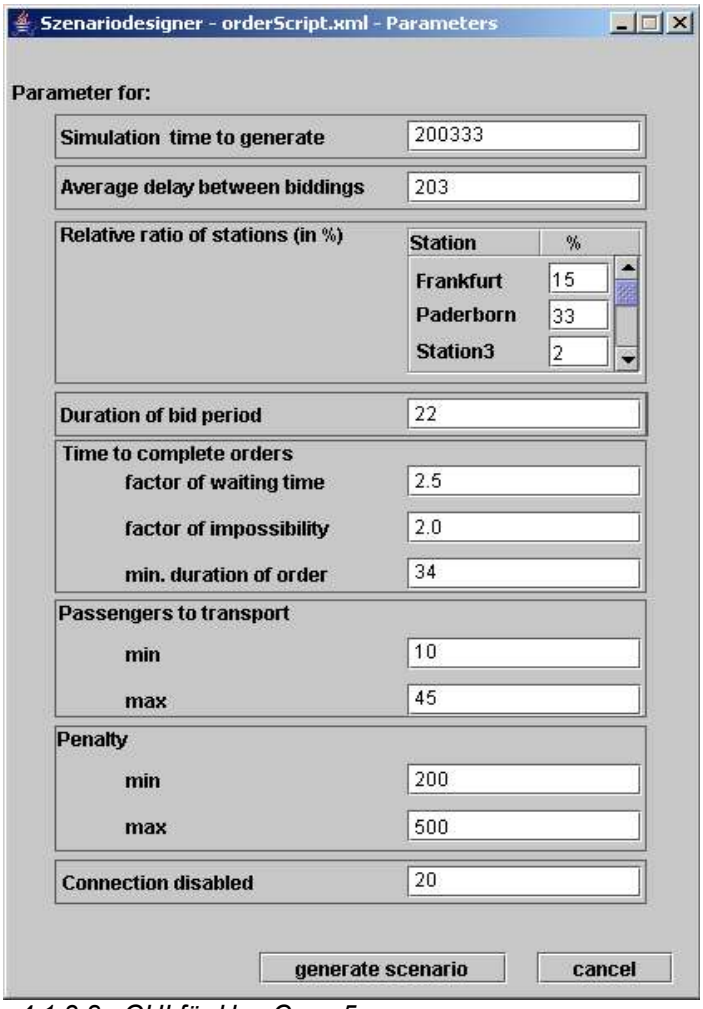

Abbildung 4.1.3.8 : GUI für Use Case 5

6) Beschreibung zu UC6 : Szenario generieren

Charakterisierende Informationen

Es werden Szenariodaten generiert.

Ziel des Use Cases:	Eine Szenario generieren
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodatei vorhanden und alle benötigten Parameter verfügbar
Nachbedingung bei erfolgreicher Ausführung:	Die generierten Ereignisse sind im Szenariodesigner aufgelistet
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will neues Szenario automatisch, unter Berücksichtigung der eingegeben Parameter (UC6), generieren

Beschreibung des allgemeinen Ablaufs

Die Generierung eines Szenarios erfolgt mittels der zuvor eingegebenen Paramter entsprechend Abbildung 4.1.3.9.

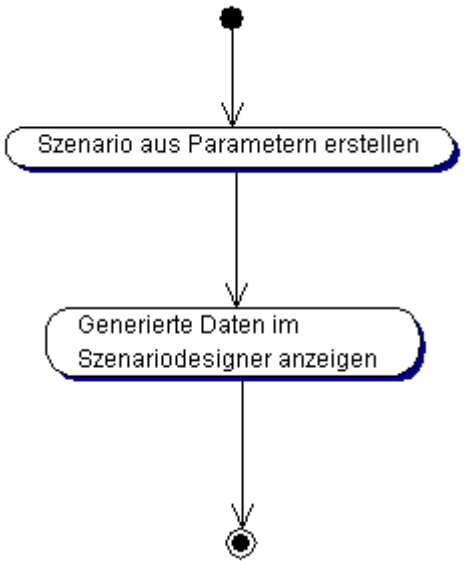

Abbildung 4.1.3.9 : Aktivitätendiagramm für Use Case 6

7) Beschreibung zu UC7 : Szenario speichern

Charakterisierende Informationen

Es werden die Szenariodaten aus dem Szenariodesigner als XML-Datei gespeichert.

Ziel des Use Cases:	Ein Szenario speichern
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Es sind gültige Szenariodaten im Szenariodesigner
Nachbedingung bei erfolgreicher Ausführung:	Eine Szenariodatei ist im XML-Format auf einem Datenträger gespeichert
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will das aktuell im Szenariodesigner befindliche Szenario für eine weitere Nutzung speichern.

Beschreibung des allgemeinen Ablaufs

Ein Szenario wird unter einem Namen in einer Datei abgespeichert (vgl. Abbildung 4.1.3.10).

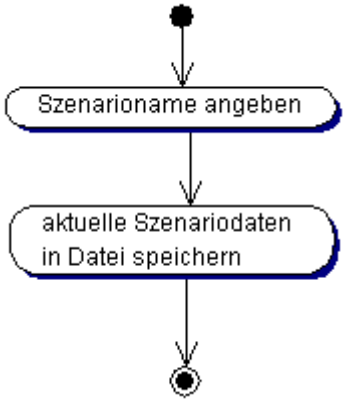

Abbildung 4.1.3.10 : Aktivitätendiagramm für Use Case 7

8) Beschreibung zu UC8 : Szenario editieren

Charakterisierende Informationen

Ein erzeugtes/geladenes Szenario wird vom Benutzer verändert. Diese Veränderung kann z.B. das löschen, verändern und hinzufügen von Ereignissen sein oder eine Änderung der Ereignisabfolge oder etwas Ähnliches. So dass der Aufbau der Szenariodatei bzw. der Ablauf der späteren Simulation verändert wird.

Ziel des Use Cases:	Eine Szenariodatei editieren
Umgebende Systemgrenze:	Szenariodesigner
Vorbedingung:	Szenariodatei vorhanden
Nachbedingung bei erfolgreicher Ausführung:	geänderte Szenariodatei vorhanden
Beteiligte Nutzer:	User
Auslösendes Ereignis:	User will Szenario editieren

Beschreibung des allgemeinen Ablaufs

Bei der Änderung eines bestehenden Szenarios, oder eines neu generierten Szenarios ist es notwendig die Zulässigkeit der Änderungen zu überprüfen (vgl. Abbildung 4.1.3.11). Die GUI (Abbildung 4.1.3.12) zeigt wie man Szenarien editieren kann.

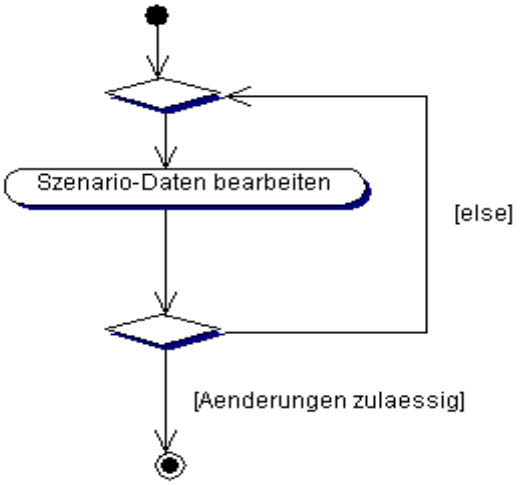

Abbildung 4.1.3.11 : Aktivitätendiagramm für Use Case 8

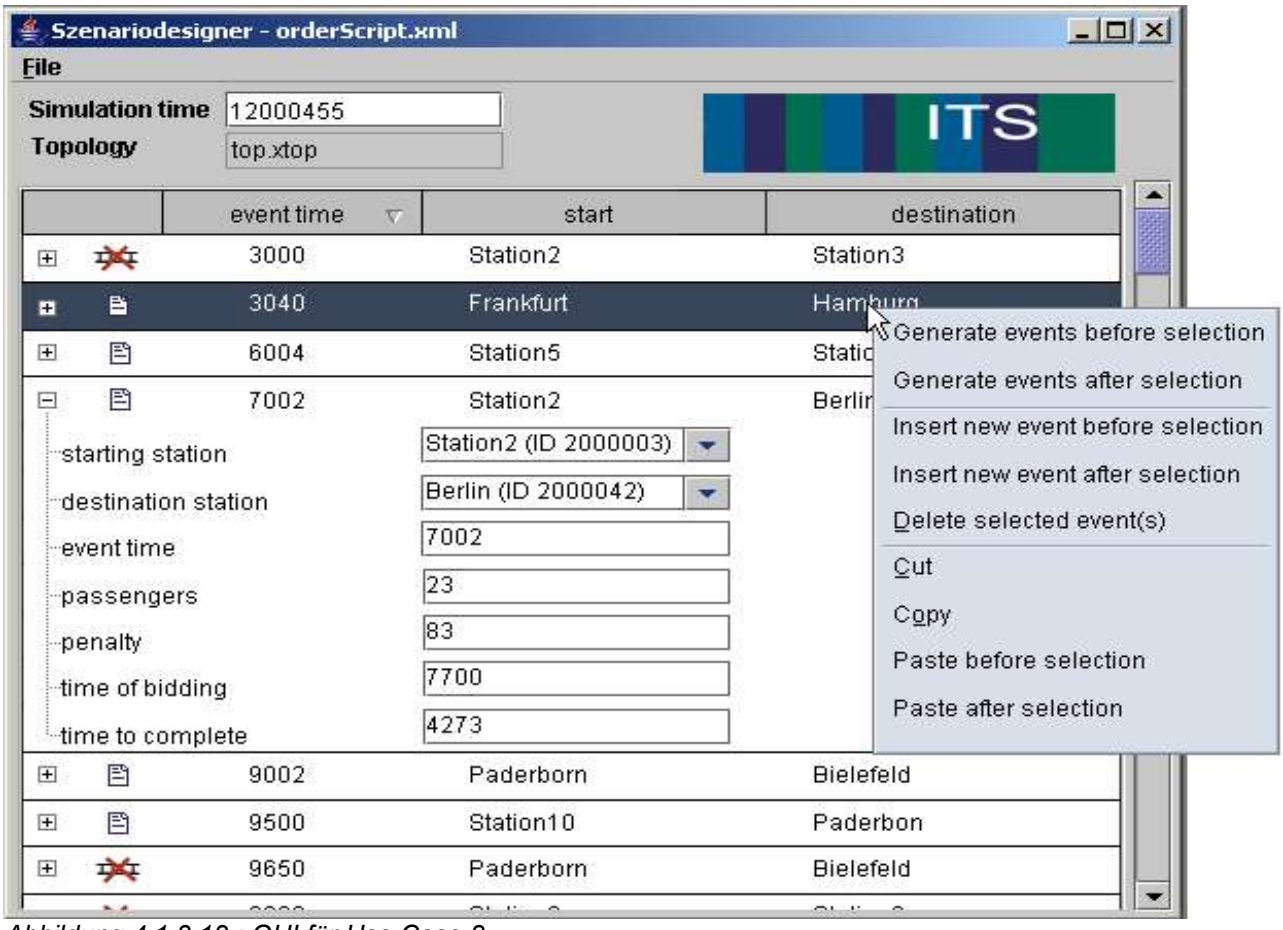

Abbildung 4.1.3.12 : GUI für Use Case 8

5. Grobentwurf (Soll-Zustand)

Die Aufgabe des Grobetwurfs ist es, den im Rahmen der Produktfunktionen beschriebenen Anwendungsfälle Komponenten zuzuordnen. So werden die Aufgaben entsprechend der 3-Schichten Architektur zerlegt und Komponenten zugeordnet, die mittels verschiedener Interfaces miteinander kommunizieren. Dies ermöglicht es die in Klassendiagrammen identifizierten Funktionsaufrufe Komponenteninteraktionsdiagrammen zuzuordnen. Diese Vorgehensweise wird zunächst auf der äußerst Abstrakten Ebene des gesamten Softwareprojekts zumindest als Zuodnung von Komponenten durchgeführt. Erst danach wird wieder auf die gewohnte Element orientierte Detailanalyse zurückgegriffen.

5.1 Komponentenstruktur des gesamten Produkts

Dieser Abschnitt beschreibt die Aufteilung der zu entwickelnden Anwendung in Komponenten, deren Schnittstellen und Interaktion.

Das System teilt sich in vier Hauptkomponenten (vgl. Abbildung 5.1) auf: die `Shuttle Steuerung`, die `Unternehmens Visualisierung`, den `Szenario Designer` und den Simulationskern(`Kernel`).

Wie im unten angeführten Komponentendiagramm(Abbildung 5.1.1) zu erkennen findet die Kommunikation zwischen `Shuttle Steuerung` und `Kernel` über die beiden Interfaces `ShuttleControlInterface` und `ShuttleModule` statt. Die `Unternehmens Visualisierung` kommuniziert über die Schnittstelle `RMIVisualisationInterface` mit dem `Kernel`, der wiederum über das `RMIRailwayPanelInterface` mit der `Unternehmens Visualisierung` kommuniziert. Die XML-Dateien, welche die Topologie-(`TopologyData`) oder Szenario-(`ScenarioData`)Daten enthalten, dienen als Schnittstelle zwischen `Szenario Designer` und `Kernel`.

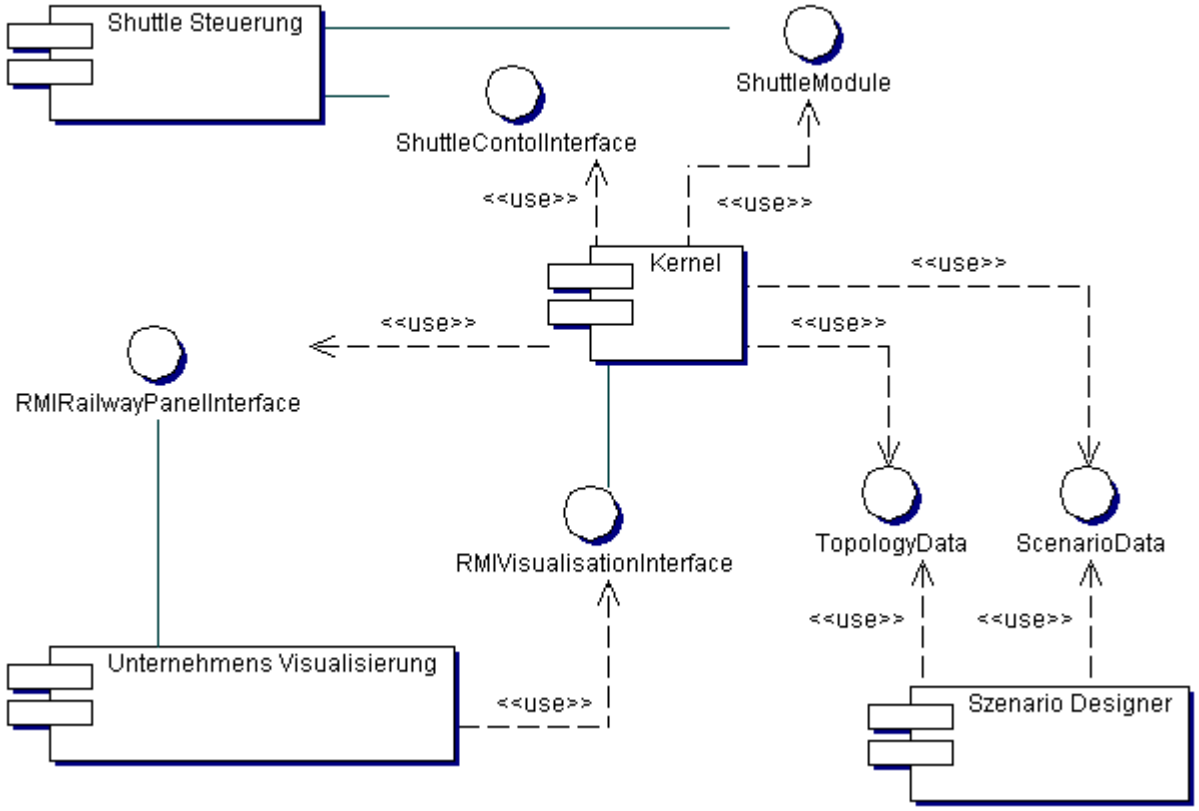

Abbildung 5.1.1 : Die gesamte Komponentenstruktur

5.2 Struktur und Interaktion der Shuttle Steuerung

Der folgende Abschnitt beschreibt einen groben Entwurf der ShuttleSteuerung anhand von Komponenten. Zunächst wird auf die Komponentenstruktur und benötigte Interfaces eingegangen.

Anschließend wird die Interaktion zwischen den Komponenten anhand von Sequenz-Diagrammen näher beschrieben.

5.2.1 Komponentenstruktur der Shuttle Steuerung

Die Shuttlesteuerung besteht aus sechs Komponenten:

1. `ShuttleControl`: Diese Komponente dient einmal zur Interaktion mit dem Kernel und ist intern die Steuerzentrale eines Shuttles. Als solche ist sie Verantwortlich für die Instanziirung der anderen Komponenten eines Shuttles.
2. `Execution`: In dieser Komponente werden die Aufträge abgearbeitet.
3. `Negotiation`: Diese Komponente ist verantwortlich für die Kostenberechnung einer Ausschreibung.

4. `Billing`: Komponente die die Finanzen eines Shuttles verwaltet. Hier können Verschiedene Bezahlungsmöglichkeiten implementiert werden.
5. `CompanyControl`: Repräsentiert ein Unternehmen und setzt die Unternehmensstrategie, die es aus der Komponente CompanyStartegy erhält, um.
6. `CompanyStrategy`: Hier wird die Strategie eines Unternehmens definiert. Diese Komponente sollte leicht editierbare Parameter enthalten um verschiedene Strategien testen zu können.

Zur Veranschaulichung folgt an dieser Stelle ein Komponentendiagramm(*Abbildung 5.2.1*). Hier erkennt man auch die Interfaces zwischen den einzelnen Komponenten:

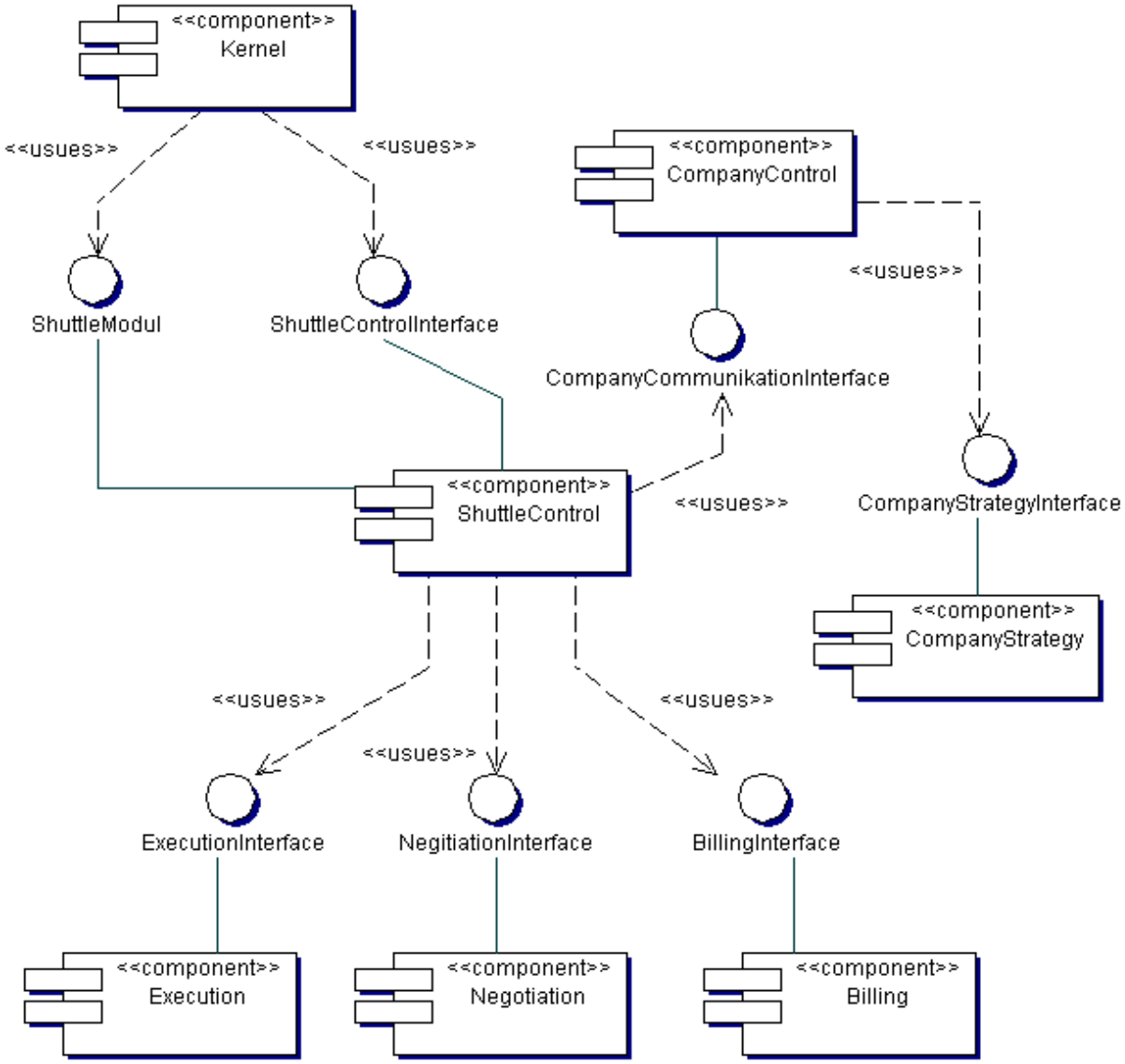

Abbildung 5.2.1 : Komponenternstrucktur der Shuttle Steurung

5.2.2 Komponenteninteraktion der Shuttle Steuerung

Der folgende Abschnitt beschreibt die Komponenteninteraktion zwischen den Komponenten der Shuttlesteuerung bei Ausführung der in Abschnitt 4.1.1 beschriebenen Use Cases mit Hilfe von Sequenz-Diagrammen. Dargestellt sind:

- UC1:Auf Ausschreibung reagieren
- UC3:Shuttle auswählen
- UC4:Angebot senden
- UC5:Auftrag erhalten

Die UseCases 2 (KostenBerechung) und 6 (Auftrag bearbeiten) sind hier nicht aufgeführt, da an dieser Stelle lediglich Operationen innerhalb einer Komponente durchgeführt werden. Eine genauere Beschreibung einer Auftragsabwicklung (UC 6) findet sich in Abschnitt 2.4.2. (Sequenzdiagramm der ShuttleStrg).

1) Komponenteninteraktion bei Ausführung von <UC1:Auf Ausschreibung reagieren>

Das folgende Sequenzdiagramm(Abbildung 5.2.2) zeigt die Komponenteninteraktion bei Ausführung von UC1:Auf Ausschreibung reagieren . Der Kernel schickt eine OrderAvailable Message an die ShuttleControl-Komponente. Diese leitet sie weiter an die Komponente Negotiation, in der sie weiterverarbeitet wird.

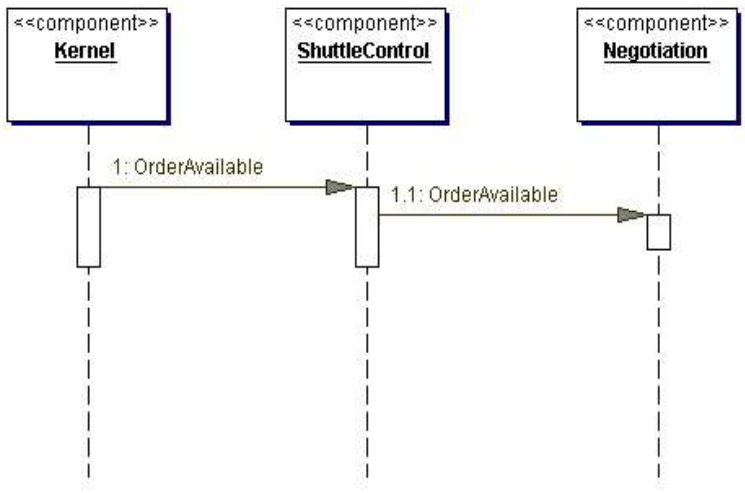

Abbildung 5.2.2 : Sequenzdiagramm der Shuttle Steuerung

2) Komponenteninteraktion bei Ausführung von <UC3: Shuttle auswählen>

Nun folgt ein Sequenzdiagramm(Abbildung 5.2.3) zur Ausführung von UC3: Shuttle auswählen.

Nachdem die Komponente Negotiation die Kosten für die Ausschreibung berechnet hat, sendet sie diese über die Komponente ShuttleControl an CompanyControl. Wenn die Komponente CompanyControl die Kosten erhalten hat, erfragt sie von ShuttleControl noch weitere Daten(z.B. Ob erst noch andere Aufträge abgearbeitet werden müssen). Wenn sie alle Daten von allen Shuttles bekommen hat, liest sie die aktuelle Strategie ein, mit deren Hilfe sie nun ein Shuttle auswählt.

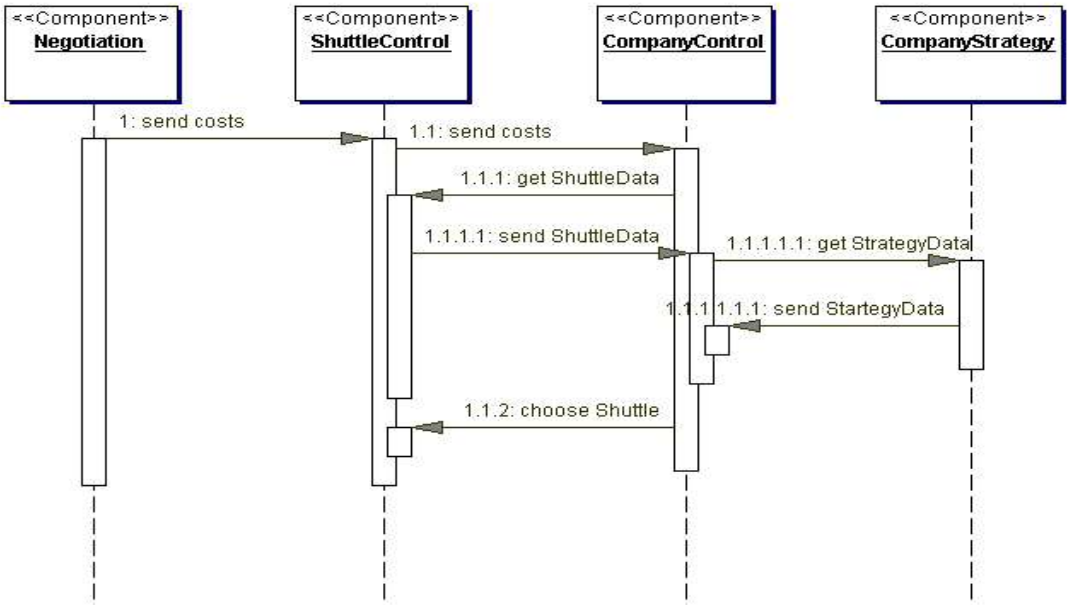

Abbildung 5.2.3 : Diese Abbildung zeigt die Interaktion der ShuttleSteuerung beim Ausführen von UC3: Shuttle auswählen

3) Komponenteninteraktion bei Ausführung von <UC4: Angebot senden >

Das folgende Sequenzdiagramm(Abbildung 5.2.4) zeigt die Komponenteninteraktion bei Ausführung von UC4: Angebot senden. Der in UC3:Shuttle auswählen ausgewählte Shuttle sendet nun sein Angebot an den Kernel. Dieser wählt das günstigste Angebot aus.

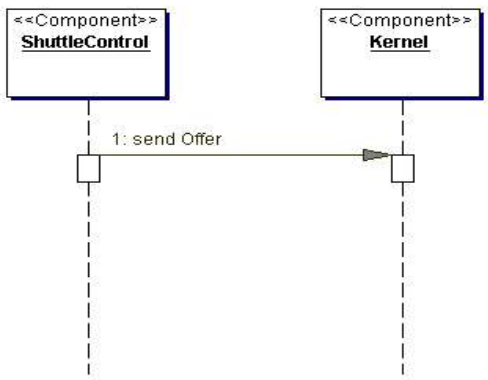

Abbildung 5.2.4 : Diese Abbildung zeigt die Interaktion der ShuttleSteuerung beim Ausführen von UC4

4) Komponenteninteraktion bei Ausführung von <UC5:Auftrag erhalten>

Das folgende Sequenzdiagramm zeigt die Komponenteninteraktion bei Ausführung von UC5:Auftrag erhalten. Der Kernel Schickt nun der Komponente ShuttleControl des Shuttles mit dem günstigstem Angebot eine AssignOrder Message.ShuttleControl übergibt den erhaltenen Auftrag nun der Komponente Execution zur Ausführung. Ein Beispiel für die Abarbeitung eines Auftrags findet sich in Abschnitt 2.4

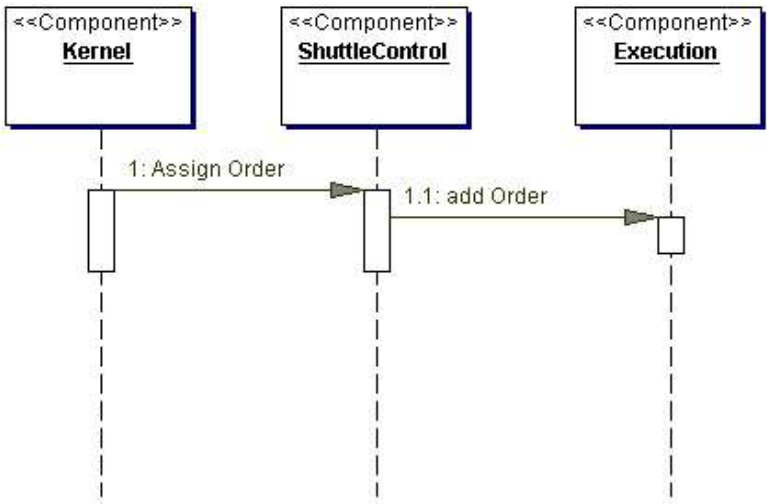

Abbildung 5.2.5 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC5:Auftrag erhalten

5.2.3 Grundzüge der Unternehmensstrategie

Nachdem die Struktur der Shuttlesteuerung geklärt ist, folgen nun einige Gedanken zum Thema Unternehmensstrategie. Diese teilt sich grob in zwei Teile: Erstens die Angebotsabgabe. Hier ist zu klären, wann welcher Shuttle eines Unternehmens ein Angebot senden darf, und in welcher Höhe.

Grundsätzlich setzt sich der Preis aus den entstehenden Kosten für Mautgebühren und Wartung des Shuttles, sowie einem Aufschlag, an dem das Unternehmen verdient, zusammen. Erhält man eine gewisse Zeit lang keinen Auftrag, da man von Shuttles anderer Unternehmen unterboten wurde, muss man seinen Aufschlag reduzieren. Andersherum kann man, sofern man viele Aufträge erhält, den Aufschlag vorsichtig erhöhen (dynamische Preisanpassung). Welches Shuttle ein Angebot senden darf, hängt in erster Linie davon ab, wie hoch seine Kosten zur Erfüllung des Auftrags wären. Dabei muss man berücksichtigen, dass Shuttles mehrere Aufträge parallel bearbeiten können, was zu einer Reduktion der Kosten führen kann. Beachtet werden muss außerdem, dass Aufträge in einer bestimmten Zeit ausgeführt werden müssen, da sonst Konventionalstrafe zu zahlen ist. Deshalb darf ein Shuttle nur ein Angebot für einen Auftrag abgeben, wenn es diesen

auch erfüllen kann. Außerdem müssen wir darauf achten, dass jeder Shuttle wenigstens einen Auftrag erfolgreich bearbeitet haben muss. Dies spielt mit zunehmender Simulationsdauer gegenüber den Kosten eine immer größere Rolle.

Der zweite Hauptaspekt der Strategie ist die Positionierung der einzelnen Shuttles auf der Karte. Es sollte eine möglichst gleichmäßige Verteilung der Shuttles auf der Karte angestrebt werden, da hierdurch die Wahrscheinlichkeit, dass sich kein Shuttle in der nähe eines Startbahnhofs einer Ausschreibung befindet, minimiert wird. Sofern es in der Topologie zentrale Bahnhöfe gibt, die besonders viele Verbindungen zu anderen Bahnhöfen haben, empfiehlt es sich dort Shuttles zu positionieren. Zusätzlich sollten die Ausschreibungen daraufhin analysiert werden, ob in manchen Bahnhöfen mehr Aufträge starten als in anderen.

5.3 Struktur und Interaktion der Unternehmens-Visualisierung

Der folgende Abschnitt beschreibt einen groben Entwurf der Unternehmens-Visualisierung anhand von Komponenten. Zunächst wird auf die Komponentenstruktur und benötigte Interfaces eingegangen, sowie den Aufbau dieser Interfaces.

Anschließend wird die Interaktion zwischen den Visualisierungs-Komponenten anhand von Sequenz-Diagrammen näher beschrieben.

5.3.1 Komponentenstruktur der Visualisierung

Die Unternehmensvisualisierung ist aus drei Komponenten aufgebaut:

1. `Communication`: Diese Kompomente dient als Schnittstelle zum Simulations-Kernel, über die die komplette Kommunikation zwischen Visualisierung und Kernel abgewickelt wird.
2. `DataManagment`: Diese Komponente bildet den Kern der Visualisierung. Hier werden einerseits die vom Kernel geschickten Daten gespeichert, andererseits aber auch zur Anzeige innerhalb der `GUI` Komponente aufbereitet.
3. `GUI`: Diese Komponente ist für die Anzeige der Unternehmens-Daten verantwortlich.

Die Komponenten sind durch Interfaces miteinander verbunden. Welche Komponente ein Interface implementiert oder welche Komponente über ein solches Interface mit einer anderen Komponente kommuniziert kann Abbildung 4.3.1 entnommen werden. Welche Funktionen die Interfaces bereitstellen und welche Datentypen verwendet werden ist in Abbildung 4.3.2 zu sehen.
Es folgt nun eine Auflistung der Interfaces und ihrer Funktion:

1. `CommunicationInterface`: Über dieses Interface kann die `Communication` Komponente angewiesen werden die Verbindung zum Kernel herzustellen oder abzubrechen. Weiterhin kann auf die Daten, welche vom Kernel geschickt wurden, zugegriffen werden. Diese Daten werden als `UpdateData` Objekte zurückgegeben.
2. `DataInterface`: Über dieses Interface kann die `GUI` Komponente der `DataManagement` Komponente mitteilen, für welches Unternehmen die Daten geschickt werden sollen.
3. `GUIInterface`: Über dieses Interface bekommt die `GUI` Komponente die Daten zur Darstellung von der `DataManagement` Komponente übergeben.

1) Komponentenstruktur

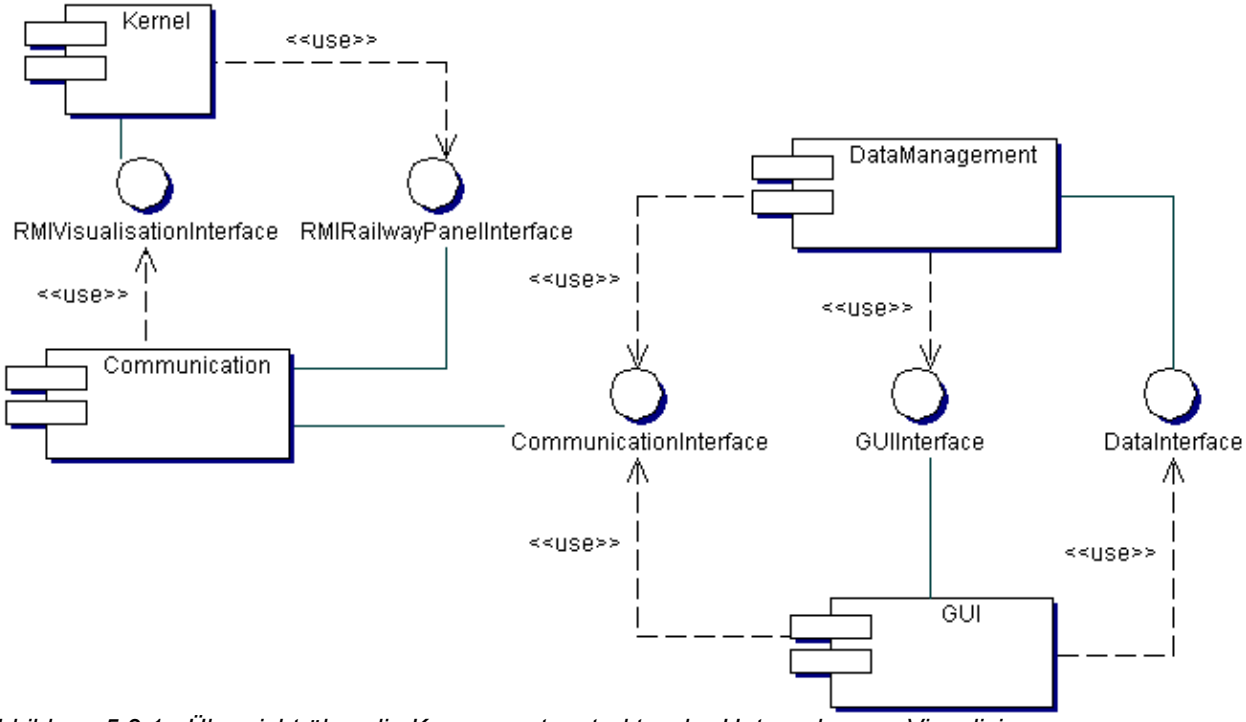

Abbildung 5.3.1 : Übersicht über die Komponentenstruktur der Unternehmens-Visualisierung.

2) Klassendiagramm

Die bereits dargestellten Interfaces stellen verschiedene Funktionen zur Verfügung. Diese werden im Klassendiagramm mit den evtl. Rückgabewerten dargestellt.

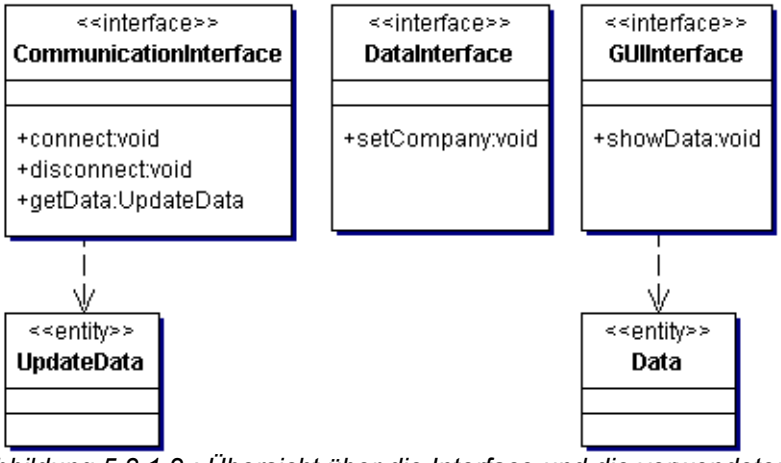

Abbildung 5.3.1.2 : Übersicht über die Interface und die verwendeten Parametertypen

5.3.2 Komponenteninteraktion der Visualisierung

Der folgende Abschnitt beschreibt die Komponenteninteraktion zwischen den Komponenten der

Unternehmens-Visualisierung bei Ausführung der in Abschnitt 4.1.2 beschriebenen Use Cases mit Hilfe von Sequenz-Diagrammen. Dargestellt sind:

- UC1:connect
- UC2:disconnect
- UC3:Unternehmen auswählen
- UC4:Fortlaufend Daten auswerten
- UC5:Daten anzeigen

Die Use Cases UC 6-9 sind hier nicht aufgeführt, da an dieser Stelle lediglich Operationen innerhalb der `GUI` Komponente durchgeführt werden.

1) Komponenteninteraktion bei Ausführung von <UC1:connect>

Das folgende Sequenzdiagramm (Abbildung 5.3.3) zeigt die Komponenteninteraktion bei Ausführung von UC1:connect. Über die GUI der `GUI` Komponente startet der User den Verbindungsaufbau. Die `GUI` Komponente leitet diese Anfrage dann an die `Communication` Komponente weiter, die für den Aufbau der Verbindung zum Simulationskern verantwortlich ist.

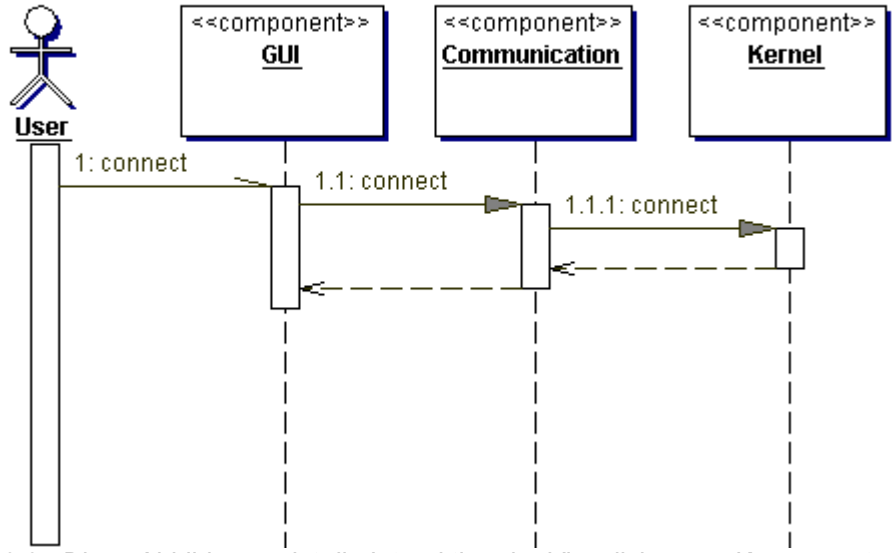

Abbildung 5.3.3 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC1:connect

2) Komponenteninteraktion bei Ausführung von <UC2:disconnect>

Das folgende Sequenzdiagramm (Abbildung 5.3.4) zeigt die Komponenteninteraktion bei Ausführung von UC2:disconnect. Über die GUI der `GUI` Komponente startet der User den Verbindungsabbau. Die `GUI` Komponente leitet diese Anfrage dann an die `Communication` Komponente weiter, die für den Abbau der Verbindung zum Simulationskern verantwortlich ist.

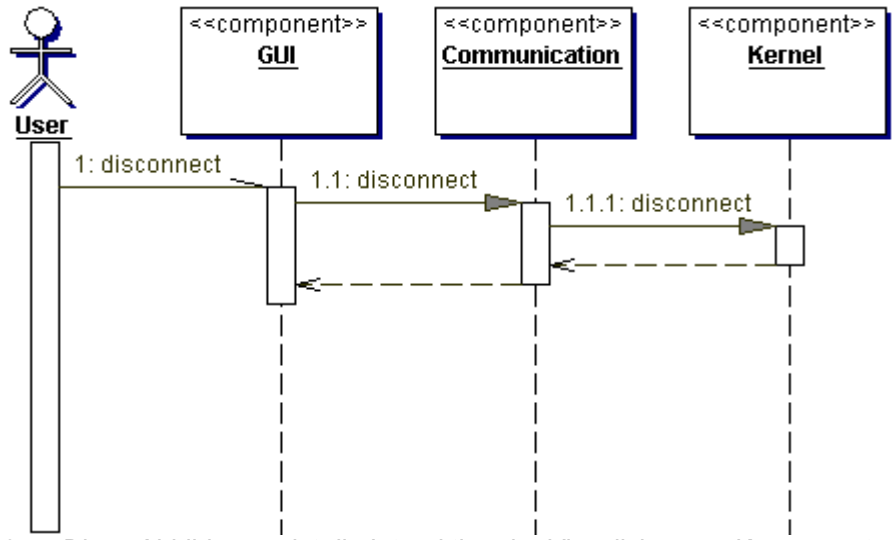

Abbildung 5.3.4 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC2:disconnect

3) Komponenteninteraktion bei Ausführung von <UC3:Unternehmen auswählen>

Das folgende Sequenzdiagramm (Abbildung 5.3.5) zeigt die Komponenteninteraktion bei Ausführung von UC3:Unternehmen auswählen. Über die GUI der `GUI` Komponente hat der User die Möglichkeit, ein Unternehmen auszuwählen, für das die Unternehmensdaten angezeigt werden sollen. Dazu wird in der `DataManagement` Komponente gespeichert, welche Daten an die `GUI` Komponente zu übertragen sind.

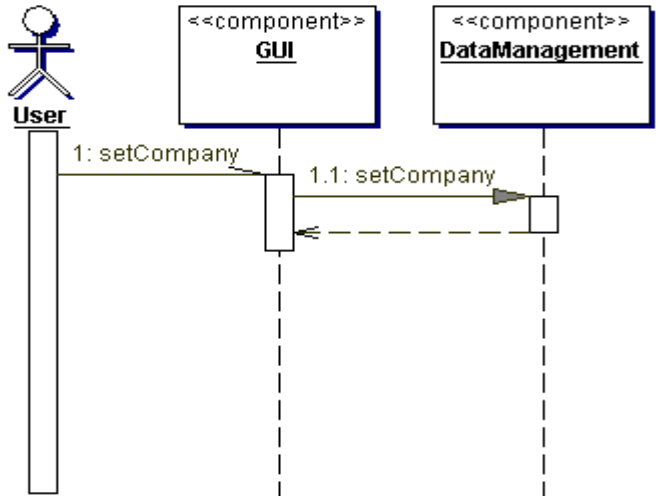

Abbildung 5.3.5 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC3:Unternehmen auswählen

4) Komponenteninteraktion bei Ausführung von <UC4:fortlaufend Daten auswerten>

Das folgende Sequenzdiagramm (Abbildung 5.3.6) zeigt die Komponenteninteraktion bei Ausführung von UC4:fortlaufend Daten auswerten. Für die Kommunikation mit dem Kernel ist die `Communication` Komponente verantwortlich. Die vom Kernel gesendeten Daten werden hier zwischengespeichert, so dass die `DataManagment` Komponente diese abholen und auswerten kann. Die so ausgewerteten und aufbereiteten Daten werden dann an die GUI Komponente zur Anzeige weitergegeben.

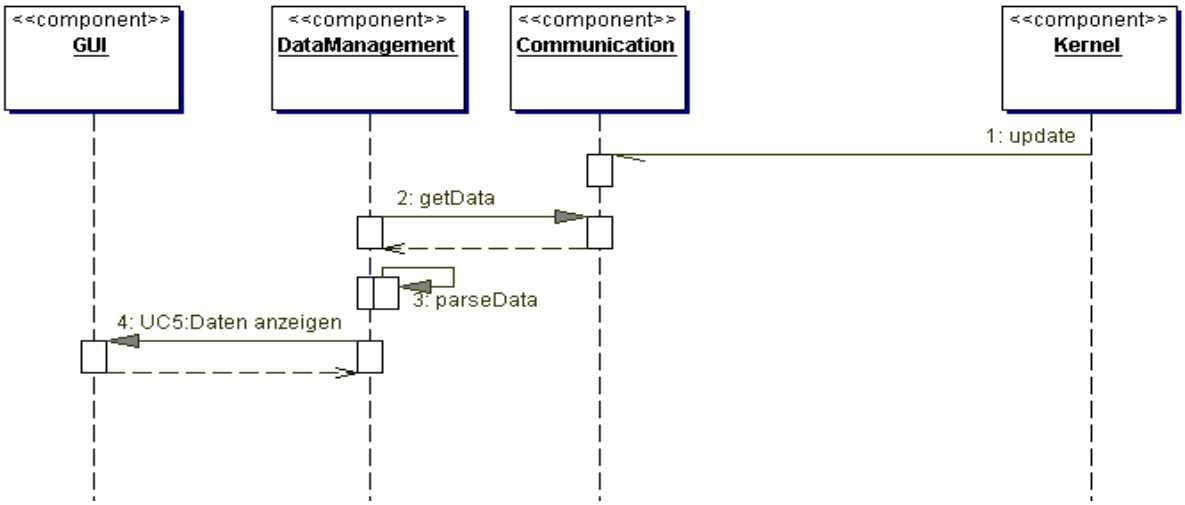

Abbildung 5.3.6 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC4:fortlaufend Daten auswerten

5) Komponenteninteraktion bei Ausführung von <UC5:Daten anzeigen>

Das folgende Sequenzdiagramm (Abbildung 5.3.7) zeigt die Komponenteninteraktion bei Ausführung von UC5:Daten anzeigen. Wie schon in 4) beschrieben, werden die in der `DataManagement` Komponente aufbereiteten Daten an die `GUI` Komponente zur Anzeige weiter gegeben. Die `GUI` Komponente ist nun dafür zuständig, die unterschiedlichen Daten entsprechend anzuzeigen (UC6-9).

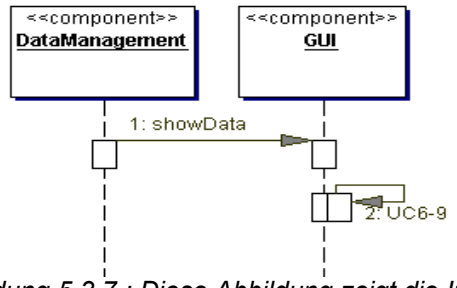

Abbildung 5.3.7 : Diese Abbildung zeigt die Interaktion der Visualisierungs-Komponenten beim Ausführen von UC5:Daten anzeigen

5.4 Komponentenstruktur des Szenariodesigners

5.4.1 Komponentenstruktur des Szenariodesigners

Der Szenariodesigners besteht aus drei Komponenten:

1. `ScenarioDataHandler`: Diese Kompomente läd und speichert die Szenario- und Topologiedateien. Sie überprüft die Dateiformate ud bietet die Inhalte der Dateien über ihre Schnittstelle `ScenarioDataInterface` an.
2. `ScenarioDesignerControl`: Diese Komponente bildet den Kern des Szenariodesigners. Hier werden die Szenarien generiert, Daten für die Anzeige aufbereitet und Nachrichten zwischen den zwei anderen Komponenten weitergeleitet.
3. `ScenarioDesigner`: Diese Komponente ist für die Anzeige der Szenario-Daten verantwortlich und für die Interaktion mit dem User.

1) Komponentenstruktur

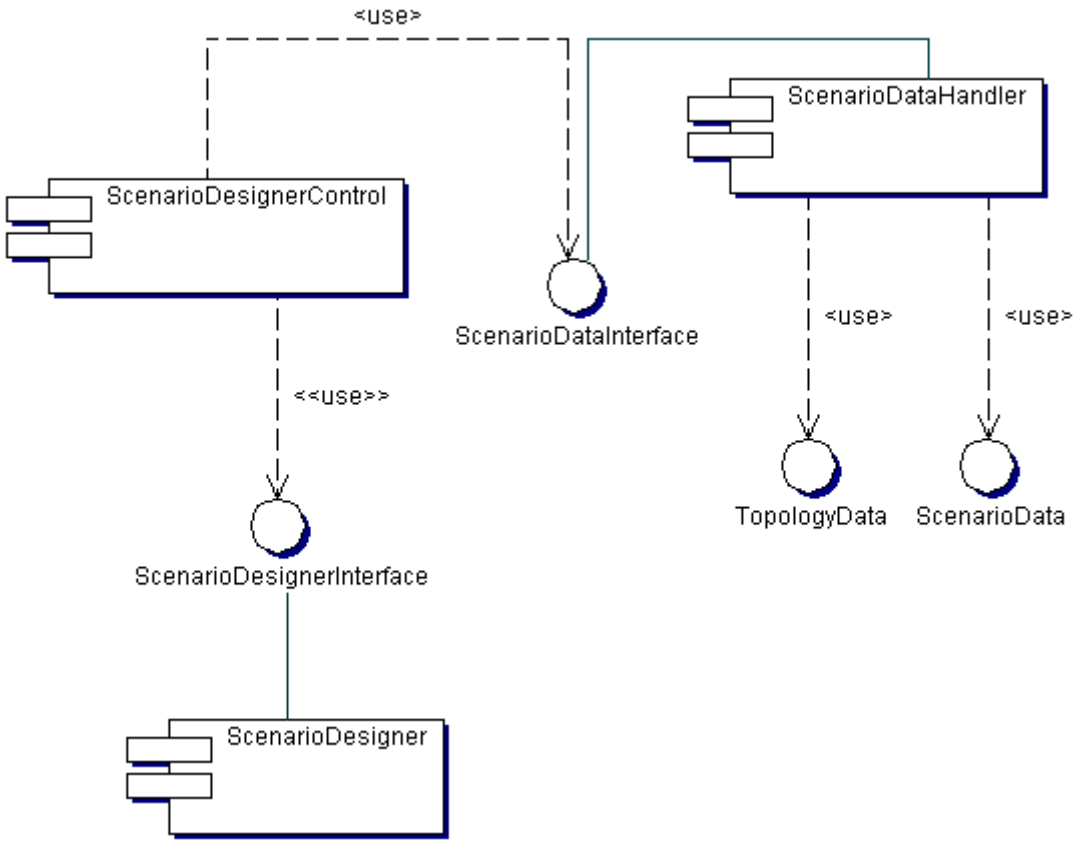

Abbildung 5.4.1 : Übersicht über die Komponentenstruktur des Szenarioders

5.4.2 Komponenteninteraktion

Der folgende Abschnitt beschreibt die Komponenteninteraktion zwischen den Komponenten des Szenariodesigners bei Ausführung der in Abschnitt 4.1.3 beschriebenen Use Cases mit Hilfe von Sequenz-Diagrammen.

1) Komponenteninteraktion bei Ausführung von UC1: Szenario laden und UC2: Topologie laden

In Abbildung 5.4.2 ist die Interaktion zwischen ScenarioDesigner, ScenarioDesignerControl und ScenarioDataHandler bei Ausführung des UC1: Szenario laden und UC2: Topologie laden dargestellt.Mittels der GUI Komponente werden ausgewählte Funktionalitäten an die Komponente ScenarioDesignerControl weitergerreicht, die der Aufgabe entsprechend mit der Klasse ScenarioDataHandler kommuniziert.

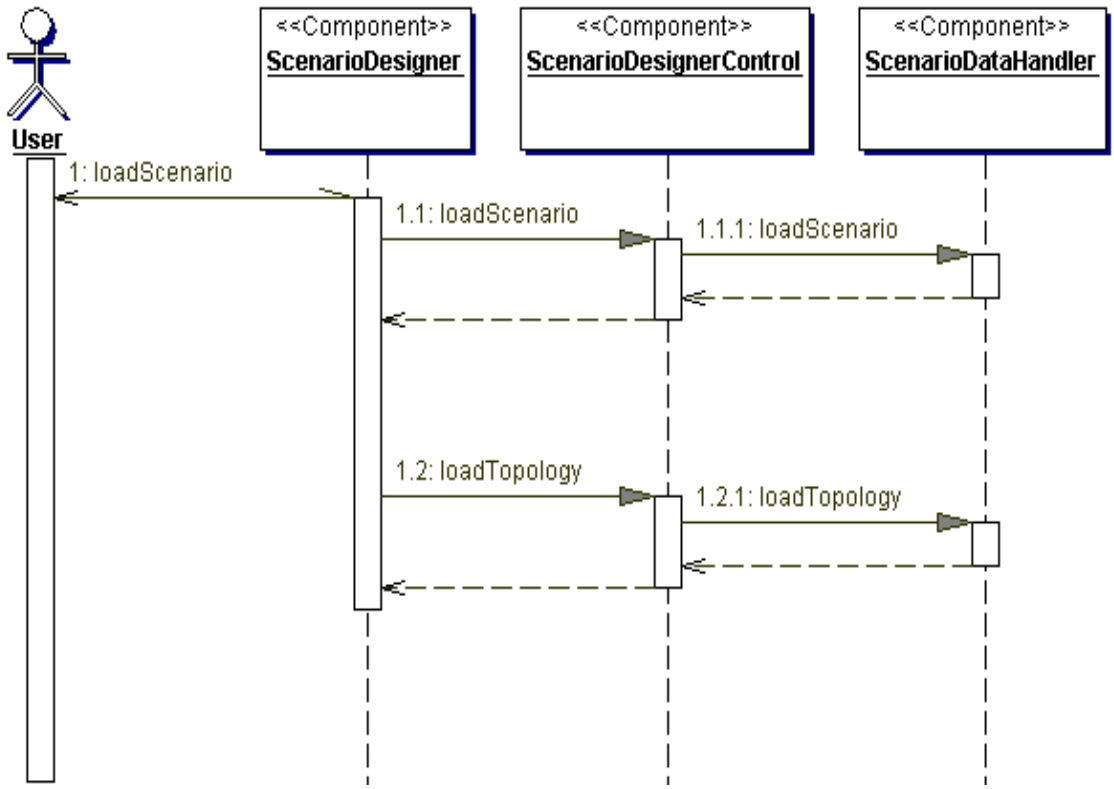

Abbildung 5.4.2 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten beim Ausführen von UC1: Szenario laden und UC2: Topologie laden

2) Komponenteninteraktion bei Ausführung von UC3: neues Szenario anlegen und UC4: zu Szenario Topologie auswaehlen

Wie aus dem Sequenzdaigramm 5.4.3 ersichtlich, wird die Funktionalität der UC3: neues Szenario anlegen und UC4: zu Szenario Topologie, wie bereits zuvor, letztlich von der Komponente ScenarioDataHandler bereitgestellt, deren Koordination mittels ScenarioDesignerControl und der GUI Komponente ScenarioDesigner gesteuert wird.

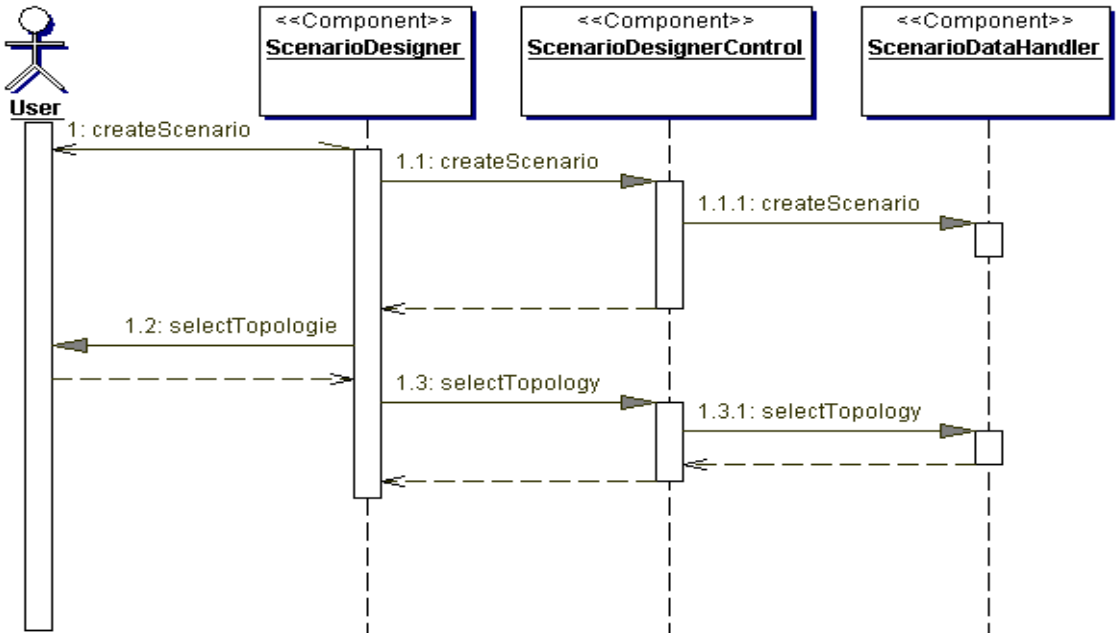

Abbildung 5.4.3 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten bei Aufsführungvon UC3: neues Szenario anlegen und UC4: zu Szenario Topologie

3) Komponenteninteraktion bei Ausführung von UC5: Parameter für die Generierung angeben

An der Generierung eines Szenarios sind nur die Komponenten ScenarioDesigner und ScenarioDesignerControl beteiligt. Die Aufrufe erfolgen nach dem aus Abbildung 5.4.4 ersichtlichen Muster.

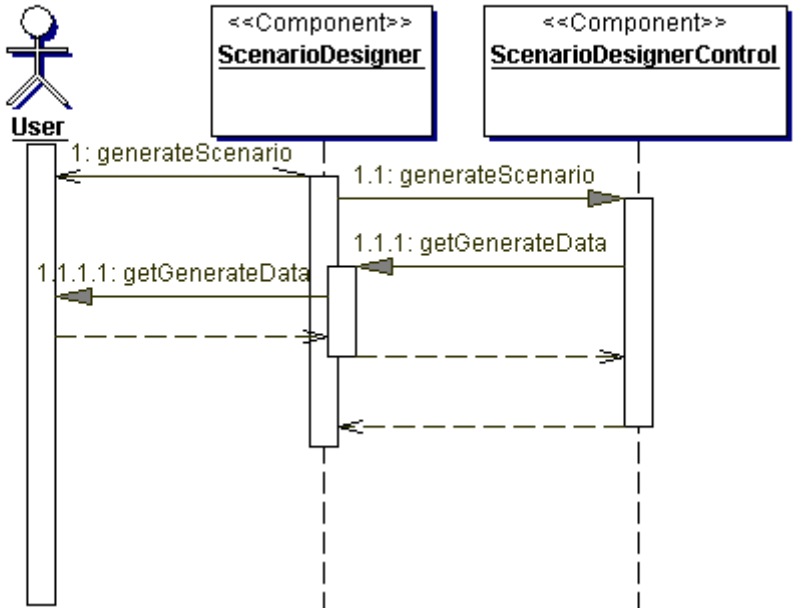

Abbildung 5.4.4 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten bei Aufsführungvon UC5: Parameter für die Generierung angeben

4) Komponenteninteraktion bei Ausführung von UC6: Szenario generieren

An der Generierung eines Szenarios sind die Funktionsaufrufe generate Scenario und set Scenario beteiligt. Ihr Aufruf ist entsprechend Abbildung 5.4.5 realisiert.

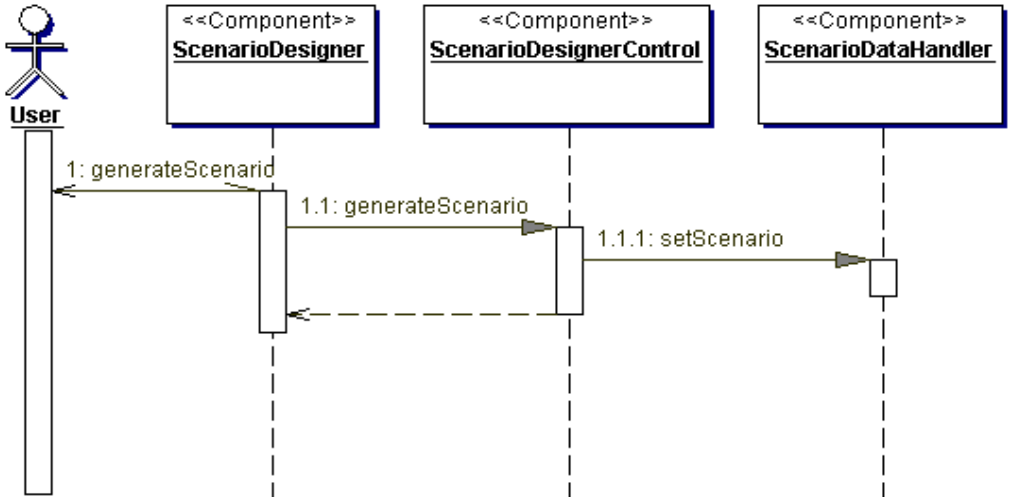

Abbildung 5.4.5 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten bei Aufsführungvon UC6: Szenario generieren

5) Komponenteninteraktion bei Ausführung von UC7: Szenario speichern

Das Speichern eines Szenarios wird über die GUI und Control Komponente bis zum ScenarioDataHandler weiterreicht (vgl. Abb. 5.4.6)

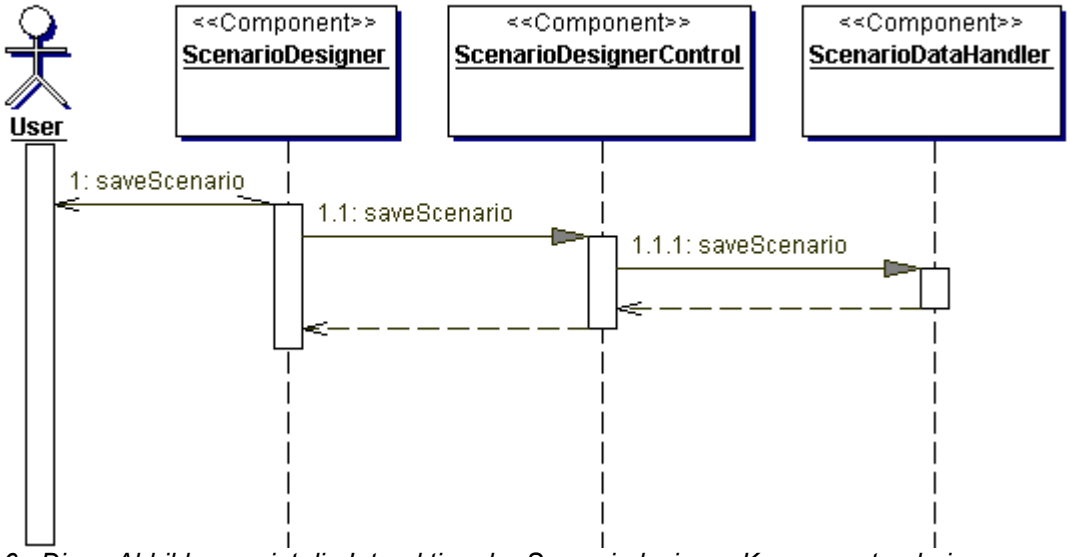

Abbildung 5.4.6 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten bei Aufsführungvon UC7:Szenario speichern

6) Komponenteninteraktion bei Ausführung von UC8: Szenario editieren

Die Interaktion erfolgt nach dem in Abbildung 5.4.7 ersichtlichen Muster.

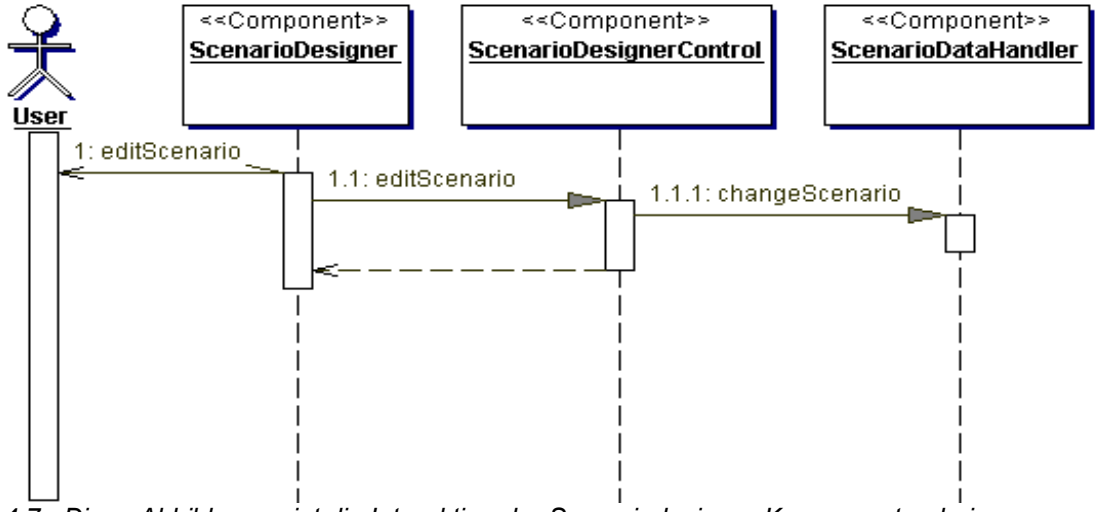

Abbildung 5.4.7 : Diese Abbildung zeigt die Interaktion der Szenariodesigner-Komponenten bei Aufsführungvon UC8Szenario editieren

6. Produktcharakteristiken

Für die Shuttlesteuerung, die Unternehmens-Visualisierung und den Szenariodesigner werden folgende System-Komponenten benötigt:

6.1 Systemumgebung

Die Produkte benötigen zur Ausführung folgende Systemumgebung:

6.1.1 Hardwareumgebung

50 Instsanzen Shuttlesteuerung sind auf folgendem System lauffähig:

Intel Pentium4 mit 2,4 Ghz
1024 MB Arbeitsspeicher

Zur Ausführung und Bedienung der Visualisierung und des Szenariodesigners wird jeweils folgende Hardware benötigt:

Intel Pentium3 mit 1.5 Ghz
512 MB Arbeitsspeicher
100 Mbit/s Ethernet
Tastatur/Maus
Farbbildschirm mit mindestens einer Auflösung von 800*600 Pixeln.

6.1.2 Softwareumgebung

Die Produkte benötigen zur Ausführung folgende Softwaremerkmale:

Java Virtual Machine 1.4.2 oder höher
TCP/IP Netzwerkprotokoll für die Visualisierung

6.2 Sonstige Anforderungen

Um die Unternehmens/Shuttlesteuerung betreiben zu können wird der Simulationskern benötigt.

7. Produktspezifische Qualitätsmerkmale

Die Entwicklungen dieses Projekts sind nun mittels verschiedener Verfahren auf die geforderten Anforderungen hin untersucht worden. Außerhalb des funktionalen Bereichs gibt es allerdings diverse nicht-funtkionale-Anforderungen, die nun aufgelistet werden. Auch hier liegt eine Unterscheidung nach Entwicklungskomponente vor.

7.1. Qualitätsmerkmale Shuttle Steuerung

Die nicht-funktionalen Anforderungen der Shuttle Steuerung beziehen sich meist auf eiffizienten Umgang mit den vorhandenenen Hardware Ressourcen.

Name:	Zeitl. optimierte Angebotsabgabe
Typ:	EFFIZIENZ
Beschreibung:	Die Bearbeitung einer Angebotsabgabe Soll so wenig Zeit, wie möglich in Anspruch nehmen
Zugeordnete(r) Use Case(s)	Auf Ausschreibung reagieren, Kosten berechnen, Shuttle auswählen, Angebot senden

Name:	Geringe Auslastung der Simulationsumgebung
Typ:	EFFIZIENZ
Beschreibung:	Die Steuerung der Shuttles darf die Simulationsumgebung nicht überlasten, wenn sie 50 mal geladen wird, verteilt auf 5 Unternehmen mit jeweils einer Flotte von 10 Shuttles
Zugeordnete(r) Use Case(s)	Alle UC der ShuttleSteuerung

Name:	Sichere Kommunikation
Typ:	SICHER
Beschreibung:	Kommunikation darf nur zwischen Shuttles eines Unternehmens stattfinden
Zugeordnete(r) Use Case(s)	Shuttle auswählen

7.2. Qualitätsmerkmale Visualisierung

Wie zu erwarten sind die meisten nicht-funktionalen-Anforderungen der Visualisierung auf die Nutzung ausgelegt. Darüberhinaus gibt es noch eine direkt aus der Aufgabenstellung zu entnehmende Anforderung, die eine Erweiterbarkeit dieser Komponente fordert.

Name:	Steuerung der Shuttles
Typ:	Pflege
Beschreibung:	Die Visualisierug soll später um die Möglichkeit der direkten Shuttle Manipulation erweitert werden.
Zugeordnete(r) Use Case(s)	Alle

Name:	Datensicherheit
Typ:	SICHER und USE
Beschreibung:	Sollten Daten für eine bestimmte Visualisierung nicht verfügbar sein wird dies erkannt und deutlich sichtbar angezeigt.
Zugeordnete(r) Use Case(s)	UC5:Daten anzeigen

Name:	Synchronizität
Typ:	SICHER
Beschreibung:	Es soll möglich sein mehrere Unternehmensvisualisierungen hintereinander und gleichzeitig an den Kern anzukoppeln.
Zugeordnete(r) Use Case(s)	UC1:connect

Name:	Visuelle Vielseitigkeit
Typ:	USE
Beschreibung:	Die Anzeige der Daten soll insgesamt sowohl in textueller, als auch in grafischer Form vertreten sein.
Zugeordnete(r) Use Case(s)	UC6:Kontostände Anzeigen, UC7: Umsatz und Kapital über Zeitverlauf anzeigen, UC8:Aufträge mit abgegebenem Angebot und errechnetem Gewinn anzeigen

7.3 Qualitätsmerkmale Szenariodesigners

Wie bereits bei der Visualisierug steht beim Entwurf des Szenariodesigners eine möglichst unkomplizierte und fehlerausschließende Nutzerführung im Vordergrund der nicht-funktionalen-Anforderungen. Darüber hinaus spielt aber auch das haushalten mit den Rechenressourcen eine wichtige Rolle.

Name:	Intuitivität
Typ:	USE
Beschreibung:	Die Bedinung des Szenariodesigner soll möglichst schnell zu verstehen sein, indem bekannte Realisierungsmuster beim Design nachvollzogen werden
Zugeordnete(r) Use Case(s)	Dies betrifft alle UC des Szenariodesigners

Name:	Szenario Editierung
Typ:	USE
Beschreibung:	Das Editieren von Szenarien im Szenariodesigner soll möglichst unkompliziert durchgeführt werden können, indem Regeln der Gestaltung von Benutzerobeflächen beachtet werden
Zugeordnete(r) Use Case(s)	UC8: Szenario editieren

Name:	Automatische Generierung
Typ:	EFFIZIENZ
Beschreibung:	Die automatische Erstellung eines Szenarios darf das System nicht zu sehr belasten
Zugeordnete(r) Use Case(s)	UC6: Szenario generieren

Name:	Sicherheitsabfrage
Typ:	SICHER
Beschreibung:	Das System muss beim beenden fragen, ob evtl. geänderte Szenariowerte gespeichert werden sollen.
Zugeordnete(r) Use Case(s)	UC3: neues Szenario anlegen, UC4: zu Szenario Topologie auswählen, UC6: Szenario generieren, UC8: Szenario editieren