Architekturdokumentation

JASI

Simulationsrahmensoftware

Ersteller:	Thomas Rieth	Erstellt am:	23.01.2006
Geändert:	Thomas Rieth	Geändert am:	05.12.2008
Dokument:	simframe-architecture.odt	Gedruckt am:	

Inhaltsverzeichnis

1 Architekturtreiber (Architecture Driver) 5
1.1 Architekturziele (Goals for the Architecture) 5
1.2 Aufgabenstellung (Business Context) 5
1.3 Stakeholder 6
2 Randbedingungen (Architecture Constraints) 6
2.1 Technische Randbedingungen 6
2.2 Organisatorische Randbedingungen 7
2.3 Konventionen 7
3 Kontextsicht 7
3.1 Logischer Kontext 8
3.1.1 Erweiterung von Simulationsobjekten 8
3.1.2 Einbindung externer Komponenten 9
3.2 Physischer Kontext 11
4 Bausteinsicht 12
4.1 Gesamtsystem (Whitebox-Beschreibung der Ebene 1) 12
4.1.1 Überblicksdiagramm des Pakets jasi.sim 13
4.1.2 Das Paket jasi.sim.basic (BlackBox-Beschreibung) 13
4.1.3 Beschreibung der Beziehungen 13
4.1.4 Offene Punkte 14
4.2 Ebene 2 14
4.2.1 Das Paket jasi.sim.basic (Whitebox-Beschreibung) 14
4.2.1.1 Übersichtsdiagramm jasi.sim.basic 15
4.2.1.2 jasi.sim.basic.value (BlackBox Beschreibung) 15
4.2.1.3 jasi.sim.basic.id (Black Box Beschreibung) 16
4.2.1.4 jasi.sim.basic.element (Black Box Beschreibung) 17
4.2.1.5 jasi.sim.basic.event (Black Box Beschreibung) 18
4.2.1.6 jasi.sim.basic.table (Black Box Beschreibung) 19
4.2.1.7 jasi.sim.basic.control (Blackbox-Beschreibung) 19
4.2.1.8 Beschreibung der Beziehungen 20
4.2.1.9 Offene Punkte 21
4.2.2 Das Paket jasi.sim.geo (Blackbox- und Whitebox-Beschreibung) 21
4.2.2.1 Übersichtsdiagramm 22
4.2.3 Das Paket jasi.sim.user (Blackbox- und Whitebox-Beschreibung) 22
5 Laufzeitsicht 23
5.1 Der Simulationsprozess 23
5.2 Lesen und Ändern der Attribute von Simulationselementen 23
5.3 Verhalten von Simulationsereignissen 25
5.4 Speichern und Lesen des Simulationsarchivs 26
6 Verteilungssicht 28
6.1 Infrastruktur des Gesamtsystems 28
7 Entwurfsentscheidungen 28
8 Szenarien zur Architekturbewertung 29
8.1 Anwendungsfälle 29

8.2 Anwendungsbeispiel einer Luftverkehrssimulation 29
9 Architekturaspekte 31
9.1 Persistenz 31
9.2 Benutzungsoberfläche 31
9.3 Ablaufsteuerung 31
9.4 Transaktionsbehandlung 31
9.5 Sessionbehandlung 31
9.6 Sicherheit 31
9.7 Kommunikation und Integration mit anderen IT-Systemen 31
9.8 Verteilung 31
9.9 Ausnahme-/Fehlerbehandlung 31
9.10 Management des Systems & Administrierbarkeit 31
9.11 Logging, Protokollierung, Tracing 32
9.12 Konfigurierbarkeit 32
9.13 Parallelisierung und Threading 32
9.14 Internationalisierung 32
10 Projektaspekte 32
10.1 Change Request 32
10.2 Technische Risiken 32
10.3 Offene Punkte 32
10.4 Erwartete Änderungen 32
10.5 Migration 32
11 Auswirkungen 32
12 Glossar 32

Abbildungsverzeichnis

Abbildung 1: Kontextsicht auf den Simulationsrahmen 7
Abbildung 2: Neue Arten von Simulationselementen durch Erweiterung der Basisklassen des Simulationsrahmens und Implementieren der Schnittstellen der Simulationsmodelle 8
Abbildung 3: Neue Arten von Simulationsereignissen durch Erweiterung der Basisklassen des Simulationsrahmens 9
Abbildung 4: Anbindung externer Komponenten mittels Beobachter- und Kontrollschnittstellen des Simulationsrahmens 10
Abbildung 5: Verwendung von Codegeneratoren 12
Abbildung 6: Paketdiagramm der Simulationsrahmensoftware im Paket sim. Im den Unterpaketen der 2. Ebene sind die weiteren untergeordneten Paketen aufgeführt 13
Abbildung 7: Paketdiagramm mit den Klassen unterhalb jasi.sim.basic. 15
Abbildung 8: Klassendiagramm für das Paket jasi.sim.basic.value 16
Abbildung 9: Klassendiagramm für das Paket jasi.sim.basic.id 17
Abbildung 10: Klassendiagramm für das Paket jasi.sim.basic.element 17
Abbildung11: Klassendiagramm für jasi.sim.basic.element.relation 18
Abbildung 12: Klassendiagramm für das Paket jasi.sim.basic.event 19
Abbildung 13: Klassendiagramm für das Paket jasi.sim.basic.table 19
Abbildung 14: Klassendiagramm für das Paket jasi.sim.basic.control 20

Abbildung 15: Klassendiagramm des Simulationsrahmens mit den Beziehungen.....20
Abbildung 16: Objektdiagramm des Simulationsprozesses.....21
Abbildung 17: Klassendiagramm des Paketes jasi.sim.geo und seiner Unterpakete.....22
Abbildung 18: Sequenzdiagramm über das Fortschreiten des Simulationsprozesses.....23
Abbildung 19: Sequenzdiagramm für das Lesen von Attributen der Simulationselemente. 24
Abbildung 20: Sequenzdiagramm für das Ändern der Attribute von Simulationselementen. 24
Abbildung 21: Sequenzdiagramm für das Verarbeiten von Simulationsereignissen.....25
Abbildung 22: Zustandsdiagramm von einfachen Ereignissen.....25
Abbildung 23: Zustandsdiagramm für wiederholbare Ereignisse.....26
Abbildung 24: Sequenzdiagramm für das Lesen eines Simulationsarchivs.....26
Abbildung 25: Sequenzdiagramm für das Schreiben eines Simulationsarchivs.....27
Abbildung 26: Verteilungsdiagramm der Ebene 1.....28
Abbildung 27: Anwendungsfälle des Simulationsrahmens.....29
Abbildung 28: Das Paket jasi.airtraffic.kernel.....30
Abbildung 29: Das Paket jasi.airtraffic.model.....30
Abbildung 30: Das Paket jasi.airtraffic.defi.....30

Tabellenverzeichnis

Tabelle 1: Schnittstelle (sim.basic.control.SimListener) für Beobachter der Simulation.....10
Tabelle 2: Schnittstelle (sim.basic.control.SimController) zur Simulationssteuerung.....11
Tabelle 3: Aufstellung der Paketstruktur auf Ebene 1.....12
Tabelle 4: Aufstellung der Klassen im Paket jasi.sim.basic.....14
Tabelle 5: Aufstellung der weiteren Unterpakete im Paket jasi.sim.basic.....14
Tabelle 6: Aufstellung der weiteren Pakete im Paket jasi.sim.basic.....16
Tabelle 7: Klassen aus dem Paket jasi.sim.basic.element.....17
Tabelle 8: Klassen aus dem Paket jasi.sim.basic.element.relation.....18
Tabelle 9: Klassen aus dem Paket jasi.sim.basic.event.....18
Tabelle 10: Klassen aus dem Paket jasi.sim.geo.....21
Tabelle 11: Klassen aus dem Paket jasi.sim.geo.cartesian.....21
Tabelle 12: Klassen aus dem Paket jasi.sim.geo.spherical.....22
Tabelle 13: Abstrakte Methoden zum Speichern und Lesen des Simulationszustandes.. .27

1 Architekturtreiber (Architecture Driver)

Die Entwicklung einer Simulationssprache, d.h. eines Softwarerahmens, ist die wesentliche Forderung an diese Architektur. Auf der Grundlage dieser Simulationssprache sollen konstruktive, ereignisorientierte Simulationssysteme entwickelt werden können. Es soll ein möglichst allgemeiner Simulationsrahmen entwickelt werden, der individuell für verschiedene Simulationssysteme angepasst werden kann. Anwendungsfelder der Simulationssysteme sollen sowohl Übungen als auch für Analysen sein. Für die Anwendung in Übungen sind Funktionalitäten wie die Darstellung der Dynamik des Simulationszustandes – die Entwicklung des Lagebildes – und die interaktive Befehlseingabe notwendig, während bei Problemanalysen mit Unterstützung durch Simulationen die Betonung mehr auf nachvollziehbare Simulationsmodelle und eine geeignete Auswertekomponente liegt, die auch eine statistische Analyse der Ergebnisse von stochastischen Simulationen erlaubt.

1.1 Architekturziele (Goals for the Architecture)

Eine Liste der Architekturziele und Randbedingungen der Simulationsrahmensoftware lautet wie folgt:

- Eine stabile Simulationssprache in Form einer Simulationsrahmensoftware.
- Einfache Integration der Simulationsrahmensoftware als Bibliothek in ein Simulationssystem.
- Automatische Testumgebung für die Komponenten des Simulationsrahmens.
- Kopplung zu externen Komponenten, wie grafische Benutzeroberflächen und Simulationsföderationen, über eine wohl-definierte Schnittstelle durch Datenbanken und geeignete Middleware.
- Identifizieren der in der Simulation abgebildeten Objekte (Simulationsobjekte) durch einen eindeutigen Schlüssel.
- Definition und Handhabung des Zustandsvektors der Simulation, einer Liste von Daten, die den Simulationszustand eineindeutig beschreibt.

1.2 Aufgabenstellung (Business Context)

Ein Simulationssystem baut auf einer Simulationssprache auf. Der hier vorliegende Simulationsrahmen soll für die Entwicklung ereignisorientierter Simulationen die folgenden Funktionalitäten zur Verfügung stellen:

- Verwaltung des Simulationszustandes und dessen Dynamik durch Schnittstellen:
  - Empfänger (Listener) von Änderungen des Simulationszustandes,
  - Steuerung (Controller) der dynamischen Entwicklung der Simulation,
  - Datenstrukturen zur Abbildung des Simulationszustandes;
- Ein Simulationsarchiv,
- Schnittstellen zu externen Simulationssystemen.

Ein Simulationssystem enthält einen Simulationskernel, der den Simulationszustand und dessen Änderungen verwaltet. Der Simulationszustand, auch Zustandsvektor der Simulation genannt, wird durch eine Liste von eindeutig identifizierbaren Daten, den Spekulationsobjekten, abgebildet. Während die elementaren Datenstrukturen vom Simulationsrahmen zur Verfügung gestellt werden, müssen die Datenstrukturen durch den Anwender definiert werden, welche die relevanten Daten der innerhalb der Simulationsmodelle referenzierten Schnittstellen auf Simulationsobjekte enthalten. Der Simulationskernel benötigt auch die Funktionalitäten, um den Simulationszustand wie in einer Datenbank zu ändern, sowie in eine hierarchisch organisierte Datenstruktur zu speichern und wieder einlesen zu können.

Bei einem Simulationsarchiv handelt es sich um eine chronologische Ordnung von Simulationszuständen.

Die dynamische Entwicklung des Simulationszustandes wird durch die zeitliche Veränderung der Komponenten des Zustandsvektors abgebildet. Die Einführung einer Simulationszeit bewirkt die Unterscheidung dieser Simulationsobjekte in persistente Simulationselemente und transiente Simulationsereignisse. Persistente Datenstrukturen, wie Simulationselemente, existieren über eine gewisse Zeitdauer, und ihr Inhalt ändert sich gewöhnlich während der dynamischen (zeitlichen) Entwicklung der Simulation. Sie speichern die Änderungen des Simulationszustandes. Transiente Datenstrukturen, wie Simulationsereignisse, sind nur zu einem bestimmten Zeitpunkt aktiv, und ihr Inhalt ist im Allgemeinen zeitlich konstant. In ereignisorientierten Simulation werden Änderungen des Simulationszustandes durch Ereignisse bewirkt. Der Simulationskalender steuert das Auftreten der Ereignisse und damit die Dynamik der Simulation. Zeitschrittsimulationen sind ein Spezialfall von Ereignisfolgesimulationen mit periodisch wiederkehrenden Ereignissen.

Ein Simulationssystem besteht neben der durch den Simulationsrahmen abgebildeten Simulationssprache aus einer in den meisten Fällen hierarchischen Struktur von Simulationsmodellen. Simulationsmodelle sind Empfänger von Simulationsereignissen und generieren die Zustandsänderungen der Simulation beim Verarbeiten der Simulationsereignisse. Sie definieren Schnittstellen, die von Simulationselementen implementiert werden, damit sie den Zustand der Simulationselemente ändern können. Dadurch wird eine komponentenorientierte Struktur für die Simulationsmodelle und das Verhalten der Simulationselemente und -Ereignisse bereitgestellt.

1.3 Stakeholder

Stakeholder sind Personen oder Organisationen, die von der Architektur betroffen sind oder zur Gestaltung beitragen können. Dies sind prinzipiell alle Entwickler von Simulationssystemen, die diesen Simulationsrahmen zur Entwicklung konstruktiver Simulationen anwenden. In konstruktiven Simulation werden bis auf einige Teilnehmer simulierte Mittel in einer simulierten Umgebung für die Anwender dargestellt. Konstruktive Simulationen werden manchmal auch als Wargaming bezeichnet.

2 Randbedingungen (Architecture Constraints)

2.1 Technische Randbedingungen

Zur Implementierung des Simulationsrahmens wird die objektorientierte Programmiersprache Java[1] verwendet. Innerhalb der Java-Programmierumgebung stehen zahlreich moderne Implementierungen für verteilte Architekturen (Middleware) und grafische Oberflächen zur Verfügung.

Weitere Randbedingungen bei der Entwicklung des Simulationsrahmens sind: Der Simulationsrahmen wird als Bibliothek (jar-Archiv) in ein Simulationssystem eingebunden. Individuelle Anpassungen sollen keine Softwareänderungen im Simulationsrahmen erfordern, sondern nur über bereitgestellte Schnittstellen erfolgen. Die Kopplung zu externen Komponenten soll flexibel über Datenbanken und Middleware, wie Corba, HLA/RTI möglich sein. Für das Simulationsarchiv dient XML als Datenformat zum Datenaustausch, und ein binäres Datenformat für die interne Anwendung.

Zum Testen wird das JUnit-Framework[2] verwendet. Dadurch kann eine automatische Testumgebung für den Simulationsrahmen entwickelt werden. Simulationsmodelle sind Empfänger für Simulationsereignisse und verwalten die Zustandsänderungen der Simulation durch Erzeugen und Verarbeiten von Simulationsereignissen. Sie definieren Schnittstellen,

1 Homepage http://java.sun.com/

2 Homepage http://junit.org/

die von Simulationselementen implementiert werden, damit sie den Zustand der Simulationselemente ändern können. Dadurch wird eine komponentenorientierte Struktur für die Simulationsmodelle und das Verhalten der Simulationselemente und -Ereignisse bereitgestellt.

2.2 Organisatorische Randbedingungen

Zurzeit bestehen keine organisatorischen, strukturellen und auf Ressourcen bezogene Randbedingungen.

2.3 Konventionen

Hier werden alle Konventionen zusammengefasst, die für die Entwicklung der Software-Architektur relevant sind. Zu den Konventionen gehören beispielsweise Programmierrichtlinien, Dokumentationsrichtlinien, Richtlinien für Versions- und Konfigurationsmanagement, sowie Namenskonventionen.

Durch die Referenzierung auf Schnittstellen anstelle von implementierten Klassen kann eine gewisse Unabhängigkeit zwischen Simulationsmodellen und Simulationsrahmen erreicht werden; dies bedeutet auch, dass Änderungen und Korrekturen in der Implementierung die Schnittstellen nicht verändern dürfen.

Der Zugriff auf die Attribute der Simulationselemente erfolgt auf der Grundlage der Konvention für JavaBeans[3] mittels der get/set-Methoden. Die JavaBeans-Architektur wird auch dazu verwendet den Simulationszustand zu speichern und zu lesen. Dies hat die Konsequenz das alle Schnittstellen, die von Simulationselementen implementiert werden, keine get/set-Methoden enthalten sollten. Mit Hilfe der von der JavaBeans Component API bereitgestellten Funktionalitäten können solche Probleme aber vermieden werden.

3 Kontextsicht

Die Kontextsicht grenzt den Simulationsrahmen von allen Nachbarsystemen ab. Die wesentlichen Schnittstellen des Simulationsrahmens zu den Nachbarsystemen sind in Abbildung 1 dargestellt.

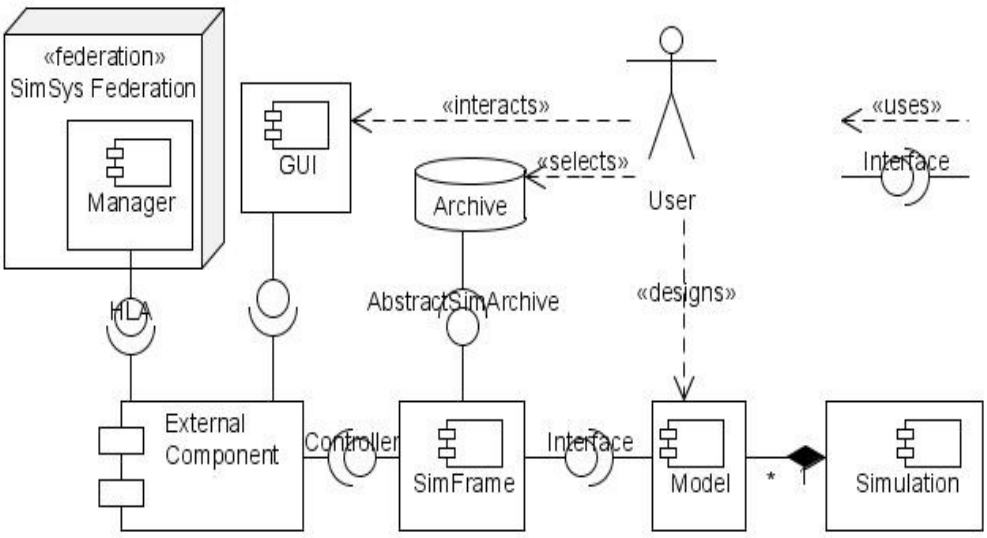

Abbildung 1: Kontextsicht auf den Simulationsrahmen.

Der Simulationsrahmen hat die Aufgabe eine Verbindung zwischen den Simulationsmodellen und den externen Komponenten des Simulationssystemen herzustellen. Durch Aus-

3 Dokumentation der JavaBeans Component API http://java.sun.com/j2se/1.5.0/

wahl und Implementierung von Modellen auf der Grundlage der vom Simulationsrahmen bereitgestellten Schnittstellen und Komponenten wie Elementen und Ereignissen wird die Simulation durch den Anwender definiert und zusammengesetzt.

Die externe Komponente des Simulationssystems ermöglicht dem Anwender mit dem Simulationssystem zum Beispiel über eine grafische Anwenderschnittstelle (GUI) zu interagieren. Zum Einbinden des Simulationssystems in eine Simulationsföderation können mit Hilfe einer externen Komponente über die Schnittstellen der Föderation gebildet werden. Die High-Level Architecture (HLA) bildet einen geeigneten Rahmen zum Erstellen einer Simulationsföderationen.

3.1 Logischer Kontext

Im logischen Kontext werden alle Nachbarsysteme des betrachteten Systems durch Spezifikation aller logischen Daten festgelegt, die mit diesen ausgetauscht werden. Zur Einbindung der Modelle stellt der Simulationsrahmen Basisklassen und Schnittstellen bereit, die in den Modellen erweitert oder implementiert werden müssen, wie in den Klassendiagrammen von Abbildung 2 und 3 dargestellt.

3.1.1 Erweiterung von Simulationsobjekten

Neue Simulationselemente werden durch eine zweifache Vererbung in die Simulation eingebracht. Die erste direkte Unterklasse, deren Bezeichnung mit Handler endet, ist für die Verwaltung der Elementattribute zuständig, die ein Bestandteil des gesamten Simulationszustandes bilden. Die nachfolgende Unterklasse verknüpft die Simulationsmodelle mit den Simulationselementen durch Implementierung der Schnittstellen der Methoden der in den Modellen definierten Objekte. Zum Beispiel in Abbildung 2 sind dieses die SimpleObject- und Movable-Schnittstellen zweier verschiedener Simulationsmodelle. Auf diese Weise erhält man eine Trennung der Daten und des Verhaltens von Simulationselementen. Die wechselseitige Abhängigkeit der Modelle und des Simulationsrahmens wird dadurch wesentlich reduziert. Eine hierarchische Erweiterung mit neuen Simulationselementen ist durch eine weitere, zweistufige Vererbung über jeweils eine Klasse zur Datenverwaltung und zum Abbilden des Verhaltens für eine Simulationselementklasse möglich.

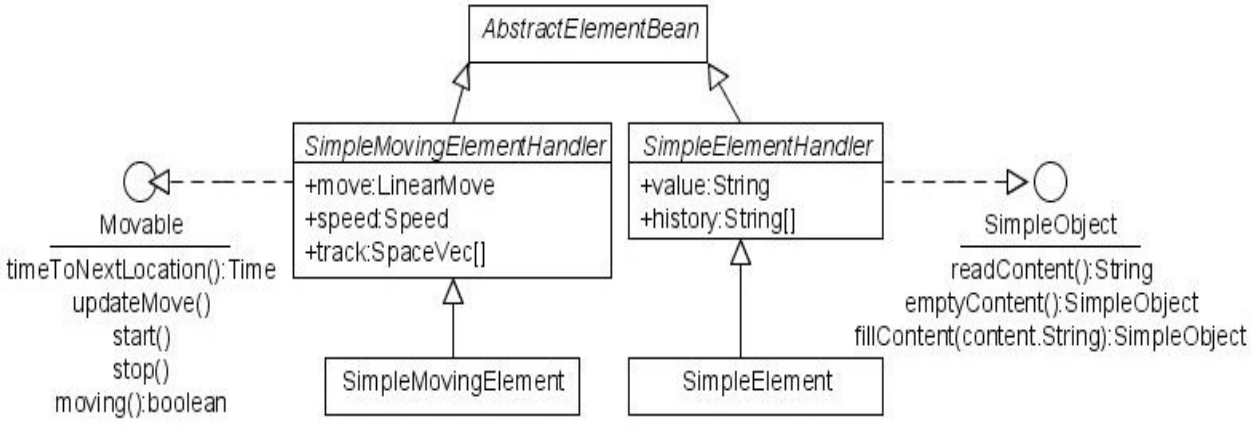

Abbildung 2: Neue Arten von Simulationselementen durch Erweiterung der Basisklassen des Simulationsrahmens und Implementieren der Schnittstellen der Simulationsmodelle.

Dieses Rohbau-Softwaremuster realisiert eine strikte Aufgabentrennung bei Modellierung, Generierung und Implementierung von Klassen in der folgenden Form. Eine Blaupause dient als Schnittstelle der zu entwickelnden Komponente. Ein Codegenerator implementiert diese Schnittstelle in einer Rohbauklasse (siehe 3.2Physischer Kontext). Im Entwicklungsprozess laufen die folgenden Tätigkeiten ab:

1. Modellierung (manuell): Entwurf der Blaupause(Schnittstelle), Modellierung der Fach- / Modelllogik.
2. Generierung (in der Regel durch Codegeneratoren): Erstellung des Rohbaus (abstrakte Handler-Klasse) auf Basis der Blaupause und der Spezifikation der Attribute auf Basis des Simulationszustandes.
3. Implementierung (manuell): Implementierung der Umsetzung Modelllogik auf Simulationszustand.

Auf diese Weise können die Struktur der zugrunde liegenden Modelle und die Implementierung des Simulationszustandes unabhängig voneinander geändert werden, sofern die Schnittstellen erhalten bleiben. Die Anzahl der Klassen wird dadurch aber möglicherweise sehr groß. Dieser Nachteil kann durch strikte Einhaltung von Namenskonventionen eingeschränkt werden.

In analoger Weise werden neue Simulationsereignisse definiert (siehe Abbildung 3). Im Unterschied zu den Simulationselementen besitzen Simulationsereignisse kein spezifisches Verhalten sondern nur Daten, die in den Simulationsmodellen zur Verarbeitung der Ereignisse benötigt werden. Das Verhalten der Simulationsereignisse wird in den Simulationsmodellen abgebildet.

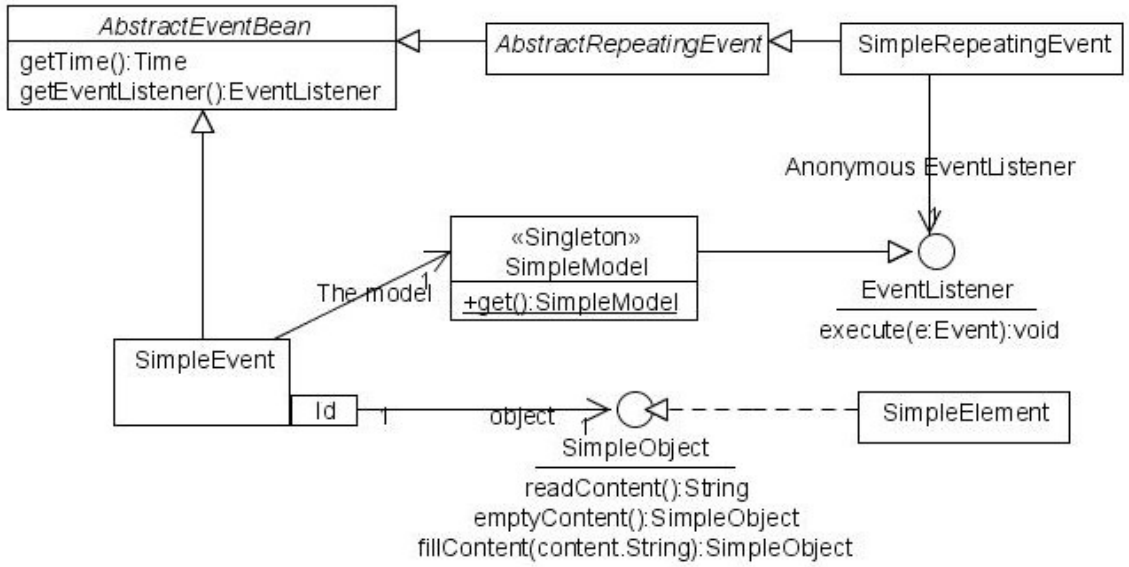

Abbildung 3: Neue Arten von Simulationsereignissen durch Erweiterung der Basisklassen des Simulationsrahmens.

3.1.2 Einbindung externer Komponenten

In Abbildung 4 sind die Schnittstellen des Simulationsrahmens dargestellt, über denen die Anwendung der Simulation über alle Änderung des Simulationszustandes informiert werden und die Dynamik des Simulationsprozesses gesteuert werden kann, der die zeitliche Änderung des Simulationszustandes bewirkt. Neben den Schnittstellen stellt der Simulationsrahmen auch abstrakte Basisklassen bereit, bei denen die leeren vordefinierten Methoden individuell überladen werden können.

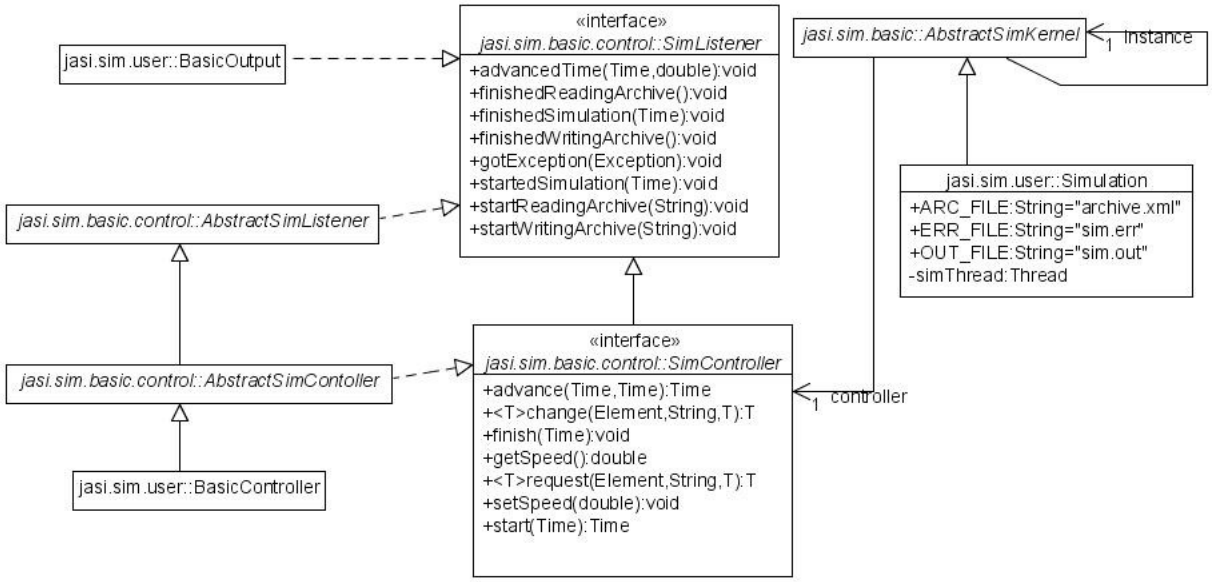

Abbildung 4: Anbindung externer Komponenten mittels Beobachter- und Kontrollschnittstellen des Simulationsrahmens.

Der Simulationskernel enthält einen Proxy, bei dem alle Simulationsbeobachter registriert werden können. Über diesen Proxy werden alle beim ihm registrierten Beobachter über jede Änderung des Simulationszustandes informiert. Die Methoden dieser Schnittstelle eines Simulationsbeobachters sind in Tabelle 1 aufgeführt.

*Tabelle 1: Schnittstelle (*sim.basic.control.SimListener*) für Beobachter der Simulation.*

Methode	***Beschreibung***
void startReadingArchive(java.lang.String file)	Der Simulationszustand wird aus dem Archiv eingelesen
void finishedReadingArchive()	Der Simulationszustand wurde eingelesen
void startedSimulation(Time time)	Der Simulationsprozess wurde gestartet
void advancedTime(Time time, double speed)	Die Simulation schreitet weiter mit der Simulationszeit
void createdElement(Element element)	Ein Simulationselement wurde erzeugt
<T> void requestedElement(Element element, java.lang.String property, T value)	Ein Simulationselementattribut wurde gelesen
<T> void changedElement(Element element, java.lang.String property, T value)	Ein Simulationselementattribute hat sich verändert
void deletedElement(Element element)	Ein Simulationselement wurde gelöscht
void pushedEvent(Event event)	Ein Simulationsereignis wurde auf den Kalender gelegt
void removedEvent(Event event)	Ein Simulationsereignis wurde gelöscht
void poppedEvent(Event event)	Ein Simulationsereignis wurde vom Kalender geworfen
void gotException(java.lang.Exception e)	Eine Ausnahme ist im Simulationsprozess aufgetreten
void gotMessage(java.lang.String m)	Der Simulationsprozess hat eine Nachricht erzeugt
void finishedSimulation(Time time)	Der Simulationsprozess wurde beendet
void startWritingArchive(java.lang.String file)	Der Simulationszustand wird in das Archiv geschrieben
void finishedWritingArchive()	Der Simulationszustand wurde geschrieben

Typische Anwendung dieses Proxys sind beispielsweise:

- Organisation der Ausgabe von Nachrichten des Simulationsprozesses in z.B. Datenbanken,
- Steuerung der Ankopplung in Simulationsföderationen,
- Aktualisieren von Darstellungen des Simulationszustandes in Anwendungen, wie z.B. grafische Oberflächen.

Zustandsänderungen der Simulation initiieren sowohl Nachrichten des Simulationssystemen in die Föderation als auch beispielsweise die Erneuerung der Darstellung in einer grafischen Oberfläche.

Eine Spezialisierung der Schnittstelle eines Simulationsbeobachters bildet die Schnittstelle zur Kontrolle des Simulationsprozesses. Von dieser Schnittstelle kann jeweils nur eine Instanz einer Klasse, die diese Kontrollschnittstelle implementiert, bei dem Simulationskernel registriert werden. Die Methoden dieser Schnittstelle sind in Tabelle 2 aufgeführt.

*Tabelle 2: Schnittstelle (*sim.basic.control.SimController*) zur Simulationssteuerung.*

Methode	***Beschreibung***
Time start(Time time)	Der Simulationsprozess soll gestartet werden
double getSpeed()	Lese Geschwindigkeit des Simulationsprozesses
void setSpeed(double speed)	Setze Geschwindigkeit des Simulationsprozesses
Time advance(Time time, Time next)	Die Simulationszeit soll fortschreiten und kann extern vorgegeben werden
<T> T request(Element element, java.lang.String name, T value)	Ein Simulationsattribut soll gelesen werden und kann von extern noch aktualisiert werden
<T> boolean change(Element element, java.lang.String name, T value)	Ein Simulationsattribute soll verändert werden und ein externes Veto kann noch erfolgen
void finish(Time time)	Der Simulationsprozess soll beendet werden

Die folgenden Methoden erlauben es der Simulationsanwendung die Änderungen des Simulationszustandes zu kontrollieren:

1. *<T> boolean change(Element element, String name, T value);* Das Attribut eines Simulationselementes soll im Simulationsprozess geändert werden, und die externe Komponente kann mit einem Rückgabewert false ein Veto gegen diese Änderung einlegen.
2. *<T> T request(Element element, String name, T value);* Auf das Attribut eines Simulationselementes wird im Simulationsprozess lesend zugegriffen und, die externe Komponente hat die Möglichkeit dessen Wert zu aktualisieren.
3. *Time advance(Time time, Time next);* Die Simulationszeit will im Simulationsprozess von der Zeit *time* zur Zeit *next* fortschreiten, und die Größe der Änderung kann von der externen Komponente neu bestimmt werden.

Diese Steuerung des Simulationsprozesses ist vor allem für die Integration in eine Simulationsföderation notwendig, in der verteilte Simulationselemente von den verschiedenen Simulationen verwaltet werden und der Zeitfortschritt der beteiligten Simulationssystemen in kausaler Weise erfolgen soll.

3.2 Physischer Kontext

Im physischer Kontext werden die Datenformate, Protokolle, sowie Medien/Kanäle der Kommunikation mit Nachbarsystemen festgelegt. Da die Erweiterung der Basisklassen für Simulationsereignisse und Simulationselemente entsprechend eines bestimmten Musters

erfolgt, besteht die Möglichkeit Codegeneratoren zur Erzeugung von Klassen für Simulationselemente und --Ereignisse zu verwenden (siehe Abbildung 5).

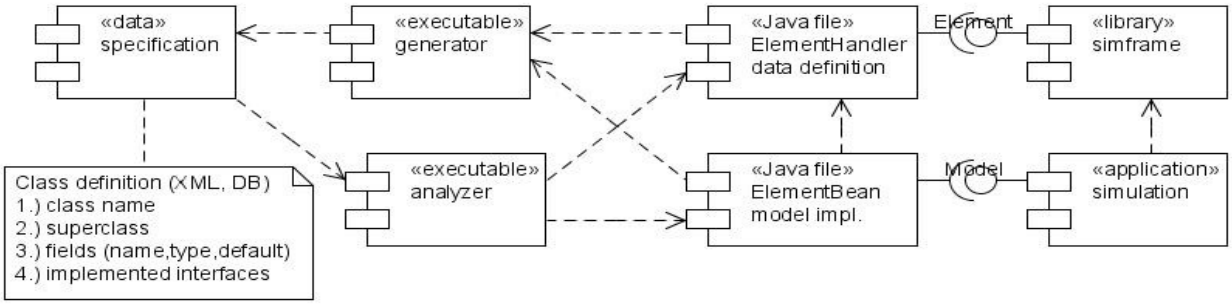

Abbildung 5: Verwendung von Codegeneratoren.

Da die Simulationsobjekte auf der Grundlage des Java Beans Component API zusammengebaut werden, kann mit Hilfe des Java API ein Analysator entwickelt werden, mit dessen Hilfe die Spezifikation für die Simulationsobjekte generiert werden kann. Auf diese Weise kann ein geschlossener Kreislauf zwischen der Spezifikation und der Implementierung der Simulationsobjekte erreicht werden.

4 Bausteinsicht

In der Bausteinsicht wird die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen beschrieben. Mit der Whitebox-Beschreibung werden das Innenlebens eines einzelnen Architekturbausteines mit allen für die Architektur relevanten Aspekten, Hintergrundinformationen, und Verweis dargestellt. Die Blackbox-Beschreibung erläutert einen Baustein aus der Sicht eines Nutzers oder Klienten, dies bedeutet, welche Aufgabe dieser Baustein übernimmt beziehungsweise welche Verantwortung er im Rahmen des Gesamtsystems wahrnimmt.

4.1 Gesamtsystem (Whitebox-Beschreibung der Ebene 1)

In Tabelle 3 ist die Blackbox-Beschreibung der Pakete aufgeführt, die Bestandteil der Simulationsrahmensoftware sind. Der Simulationsrahmen wurde in Pakete unterteilt, denen die in dieser Tabelle beschriebenen Verantwortlichkeiten zugeordnet sind. Die zentrale Komponente des Simulationsrahmen bildet der Simulationskernel im Paket *sim.basic*. Die anderen Pakete stellen ergänzende Funktionalität zur Verfügung oder stellen Beispiele für die Anwendungen von Simulationen bereit.

Tabelle 3: Aufstellung der Paketstruktur auf Ebene 1.

Paket	***Beschreibung***
jasi.sim.basic	Simulationskernel mit allen grundlegenden Datenstrukturen zur Veraltung des Simulationszustandes wie Simulationselemente, Ereignisse und Kalender, Tabellen und Schnittstellen für Simulationsmodelle und externe Komponenten.
jasi.sim.geo	Klassen zur Abbildung geometrischer Objekte (Ort, Geschwindigkeit) und Bewegungen im dreidimensionalen kartesischen Raum und einer Kugeloberfläche.
jasi.sim.user	Klassen die allgemein verwendbare Implementierungen von Funktionalitäten, die vom Simulationskernel benötigt werden, sowie allgemeinen, komponentenorientierten Simulationsmodellen, die Simulationsanwendungen zur Verfügung gestellt werden.
jasi.simple	Exemplarisches Simulationsanwendung zur Demonstration der Nutzung der Simulationsrahmensoftware.

Paket	*Beschreibung*
jasi.airtraffic	Beispiel einer Simulationsanwendung zur Abbildung von Luftverkehr.

4.1.1 Überblicksdiagramm des Pakets *jasi.sim*

In Abbildung 6 sind die Abhängigkeiten der untergeordneten Pakete dargestellt. Es wurde darauf geachtet, dass keine wechselseitigen Abhängigkeiten zwischen diesen Paketen bestehen.

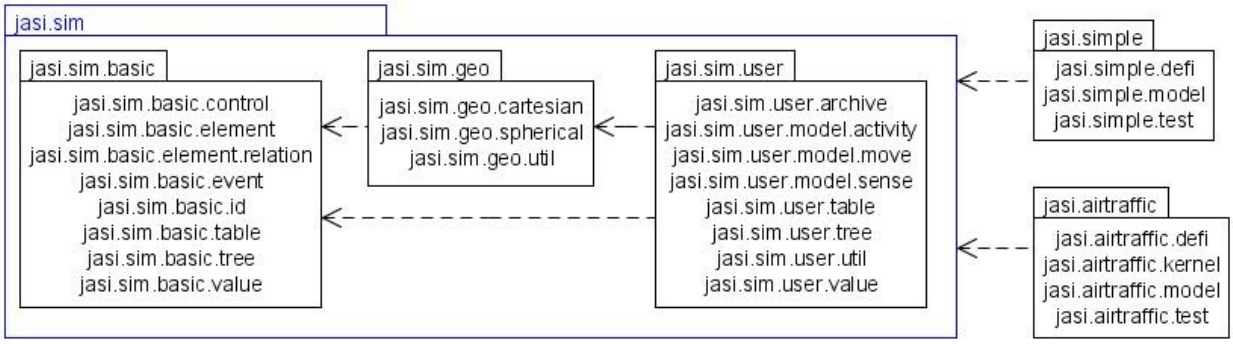

Abbildung 6: Paketdiagramm der Simulationsrahmensoftware im Paket jasi.sim*. Im den Unterpaketen der 2. Ebene sind die weiteren untergeordneten Paketen aufgeführt.*

4.1.2 Das Paket *jasi.sim.basic* (BlackBox-Beschreibung)

Das Paket *jasi.sim.basic* stellt zusammen mit seinen untergeordneten Paketen den Simulationskernel mit allen grundlegenden Datenstrukturen zur Veraltung des Simulationszustandes wie Simulationselemente, Ereignisse und Kalender, Tabellen und Schnittstellen für Simulationsmodelle und externe Komponenten zur Verfügung. Wichtige Schnittstellen, die bereitgestellt werden, sind:

- Schnittstellen zur Einbindung externer Komponenten, siehe hierzu auch:
  - 3.1.2Einbindung externer Komponenten,
  - 4.2.1.7sim.basic.control (Blackbox-Beschreibung);
- Schnittstellen und Basisklassen für die Implementierung konkreter Simulationsmodelle, siehe hierzu auch:
  - 3.1.1Erweiterung von Simulationsobjekten
  - 4.2.1.4sim.basic.element (Black Box Beschreibung),
  - 4.2.1.5sim.basic.event (Black Box Beschreibung).

Dieses Paket erfüllt die Anforderungen an eine stabile Simulationssprache in Form einer Simulationsrahmensoftware und die Definierbarkeit des Zustandsvektors der Simulation.

4.1.3 Beschreibung der Beziehungen

Die beiden Pakete *jasi.sim.geo* (siehe 4.2.2Das Paket sim.geo (Blackbox- und Whitebox-Beschreibung)) und *jasi.sim.user* (siehe 4.2.3Das Paket sim.user (Blackbox- und Whitebox-Beschreibung)), sowie ihre untergeordneten Pakete stellen Datenstrukturen bereit, die in Form von Attributen der Simulationselemente und Simulationsereignisse Bestandteile des Simulationszustandes bilden können.

Zur Beschreibung der Abhängigkeiten siehe 4.1.1Überblicksdiagramm des Pakets sim.

4.1.4 Offene Punkte

Es fehlt noch die Implementierung eines Codegenerators, der die Erzeugen des Zustandsvektors unterstützt (siehe 3.2Physischer Kontext).

4.2 Ebene 2

Ebene 2 bildet einen vergrößerten Ausschnitt auf Sicht in die Bausteine der Ebene 1 hinein und ist somit die Sammlung aller White-Box- Beschreibungen der Bausteine der Ebene 1 zusammen mit den Black-Box-Beschreibungen der Bausteine der Ebene 2.

4.2.1 Das Paket *jasi.sim.basic* (Whitebox-Beschreibung)

In Tabelle 4 werden die Klassen beschrieben, die unterhalb des Paketes *jasi.sim.basic* angeordnet sind. In Tabelle 5 werden die Pakete beschrieben, die unterhalb des Paketes *jasi.sim.basic* angeordnet sind.

Tabelle 4: Aufstellung der Klassen im Paket jasi.sim.basic*.*

Klasse	***Beschreibung***
AbstractSimArchive	Abstrakte Basisklasse für alle Simulationsarchive zum Speichern aller Simulationsobjekte. Erweiterungen dieser Klasse werden verwendet, um das Simulationsarchive in verschiedenen Formaten zu speichern (siehe auch 5.4Speichern und Lesen des Simulationsarchives).
AbstractSimKernel	Abstrakte Basisklassen des Simulationskernels.
SimProxy	Simulationsproxy (siehe auch 3.1.2Einbindung externer Komponenten)

Tabelle 5: Aufstellung der weiteren Unterpakete im Paket jasi.sim.basic*.*

Paket	***Beschreibung***
jasi.sim.basic.control	Beobachterschnittstelle und Kontrollschnittstelle zur Information über Änderungen des Zustandsvektors und Steuerung des Simulationsprozesses.
jasi.sim.basic.element	Schnittstelle und abstrakte Basisklasse für Simulationselemente.
jasi.sim.basic.element.relation	Klassen für Relationen zwischen Simulationselementen.
jasi.sim.basic.event	Schnittstelle und abstrakte Basisklasse für Simulationsereignisse.
jasi.sim.basic.id	Identifikatoren für Simulationsobjekte.
jasi.sim.basic.table	Datentabellen als Komponente des Zustandsvektors der Simulation.
jasi.sim.basic.tree	Graphen und Bäume als Komponente des Zustandsvektors der Simulation.
jasi.sim.basic.value	Klassen für Attribute von Simulationsobjekten.

Eine Simulationsanwendung wird durch Erweiterung der AbstractSimKernel-Klasse erzeugt und muss eine run-Methode in der folgenden Form implementieren:

```
Time end = ...;
Time current = startSimulation();
try {
   while (isRunning()) {
      if (current.isGE(end)) {
         break; // The requested end time has been reached
      }
```

```
            current = continueSimulation(current, end);
        }
    } finally {
        finishSimulation();
    }
```

Der Simulationsprozess kann in der folgenden Form gestartet und beendet werden:

```
sim.start();
try {
    sim.join();
} catch (InterruptedException e) {
    System.err.println(e);
} finally {
    sim.writeArchive();
}
```

Eine genauere Beschreibung ist in der Dokumentation der Laufzeitsicht zu finden (5.1Der Simulationsprozess)

4.2.1.1 Übersichtsdiagramm *jasi.sim.basic*

Die Abhängigkeiten zwischen den Paketen unterhalb *jasi.sim.basic* und die in den Paketen enthaltenden Klassen sind im Paketdiagramm von Abbildung 7 darstellt.

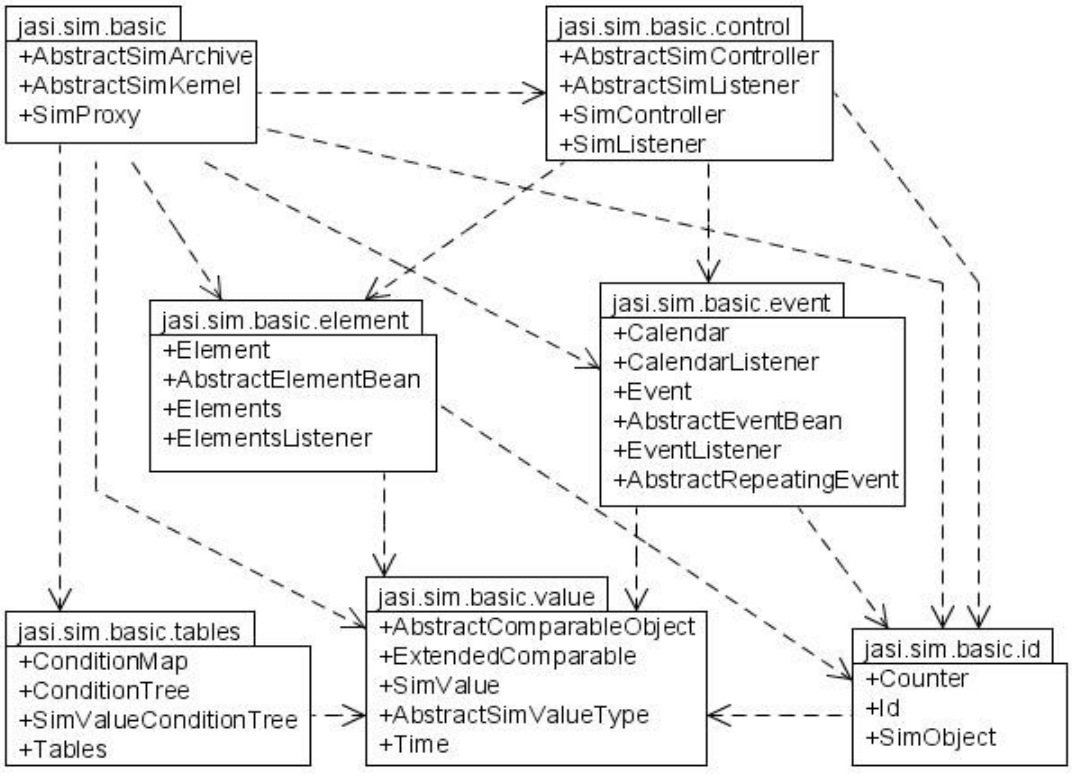

Abbildung 7: Paketdiagramm mit den Klassen unterhalb jasi.sim.basic*.*

4.2.1.2 jasi.sim.basic.value (BlackBox Beschreibung)

Dieses Paket (siehe Abbildung 8) stellt die Methoden von Standarddatentypen für den Simulationsrahmen bereit. Die Klassen *TFloat*, *TString* und *TInteger* sind Bestandteil des Paketes *jasi.sim.user.value*.

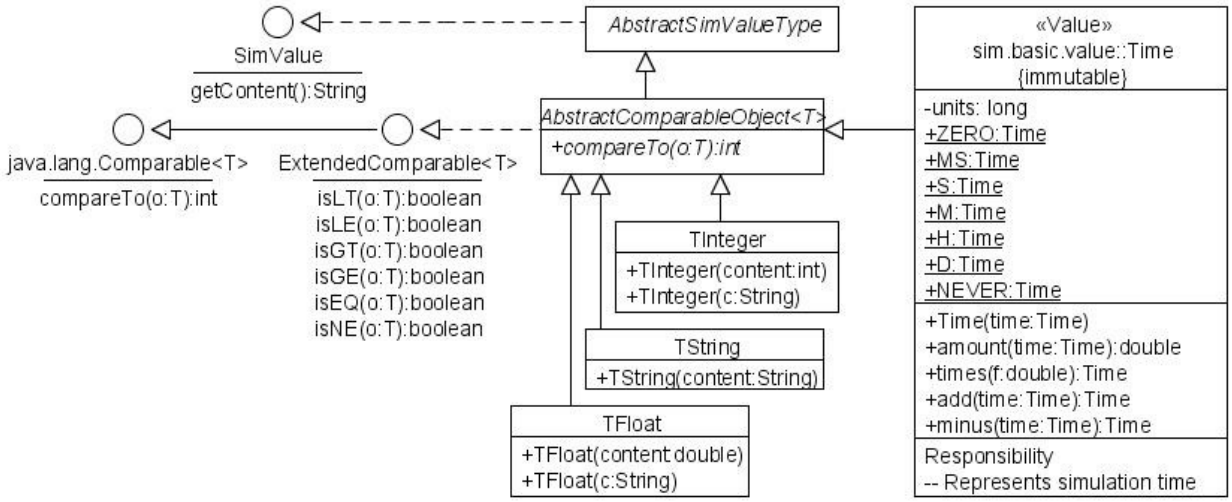

Abbildung 8: Klassendiagramm für das Paket jasi.sim.basic.value*.*

Eine wichtige Funktionalität ist das Speichern und Lesen dieser Datentypen als String, um beispielsweise in Archiven im XML-Format gespeichert werden zu können. Alle Klassen, die die Schnittstelle *AbstractSimValueType* implementieren, müssen einen Konstruktor mit String-Argument und die Methode *getContent():String* besitzen. Alle Unterklassen dieser Objektklasse müssen die folgende Bedingung erfüllen.

```
AbstractSimValueType a = ...;
AbstractSimValueType b = new ...(a.toString());
assert(a.equals(b); // always true
```

Die Schnittstelle *ExtendedComparable* stellt im Vergleich zu der Schnittstelle *java.lang.-Comparable* zusätzliche Methoden zum Vergleich von Wertobjekten bereit. Der Vergleich basiert auf den Attributwerten und nicht auf der Objektidentität, wie bei einfachen Objekten. An den Stellen im Code, an denender Vergleich zweier Wertobjekte stattfindet, vereinfacht sich die Impleentierung. Das Wissen um den attributbasierten vergleich wird zentral in der Klasse implementiert, und Wissen über den inneren Aufbau des Objektes wird nicht benötigt. Die unbedachte Änderung des Vergleichsverhaltens kann zu unerwünschten Seiteneffekten führen, deshalb sollte auch die folgende Randbedingung immer berücksichtigt werden. Wenn zwei Objekte gleich sind bezüglisch der Methode *equals(Object)*, dann muss auch die Methode *hashCode()* für beide Objekte denselben ganzzahligen Wert (integer) erzeugen.

4.2.1.3 jasi.sim.basic.id (Black Box Beschreibung)

Dieses Paket stellt eindeutige Schlüssel (Identifikatoren) für alle Simulationsobjekte, Elemente und Ereignisse, zur Verfügung (siehe Tabelle 6 und Abbildung 9).

Tabelle 6: Aufstellung der weiteren Pakete im Paket jasi.sim.basic*.*

Klasse	***Beschreibung***
SimObject	Schnittstelle für Simulationsobjekte mit eindeutigem Schlüssel.
Id	Eindeutiger Schlüssel.
Counter	Schlüsselzähler, Fabrikklasse für eindeutige Schlüssel.

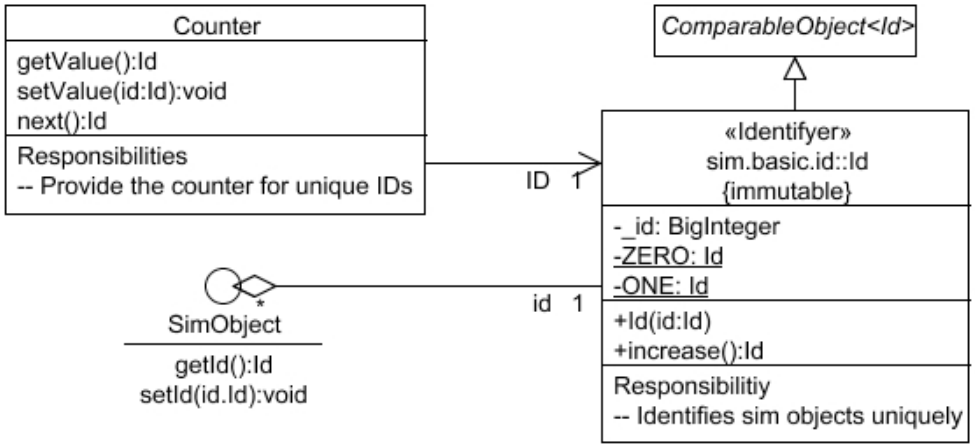

Abbildung 9: Klassendiagramm für das Paket jasi.sim.basic.id.

4.2.1.4 jasi.sim.basic.element (Black Box Beschreibung)

Dieses Paket enthält die grundlegenden Schnittstellen und Basisklassen für die Implementierung von Simulationselementen. Änderungen im Simulationszustand spiegeln sich in den Änderungen der Attribute von Simulationselementen nieder (siehe auch 5.2Lesen und Ändern von Simulationselementen).

Im Paket *jasi.sim.basic.element* sind die grundlegenden Basisklassen und Schnittstellen für alle Simulationselemente enthalten (siehe Abbildung 10 und Tabelle 7).

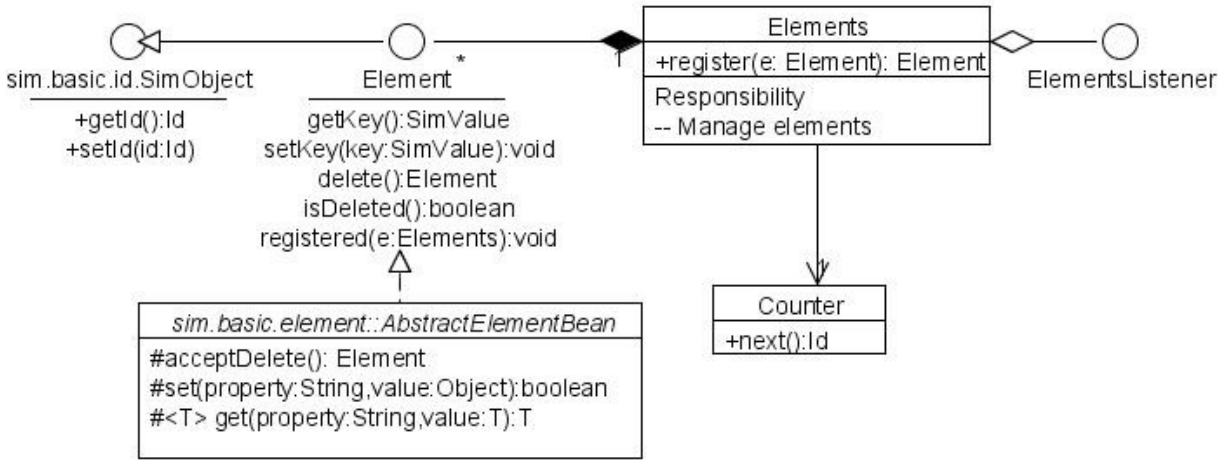

Abbildung 10: Klassendiagramm für das Paket jasi.sim.basic.element.

Tabelle 7: Klassen aus dem Paket jasi.sim.basic.element.

Klasse	***Beschreibung***
Element	Schnittstelle für Simulationselemente.
AbstractElementBean	Zentrale Verwaltung der Änderung der Attribute von Simulationselementen.
Elements	Verwaltung und Registrierung von Simulationselementen.
ElementsListener	Beobachterklassen für die Änderung von Simulationselementen.

Die im Paket *jasi.sim.basic.element.relation* enthaltenen Klassen sollen Relationen zwischen Simulationselementen abbilden (siehe Abbildung 11 und Tabelle 8).

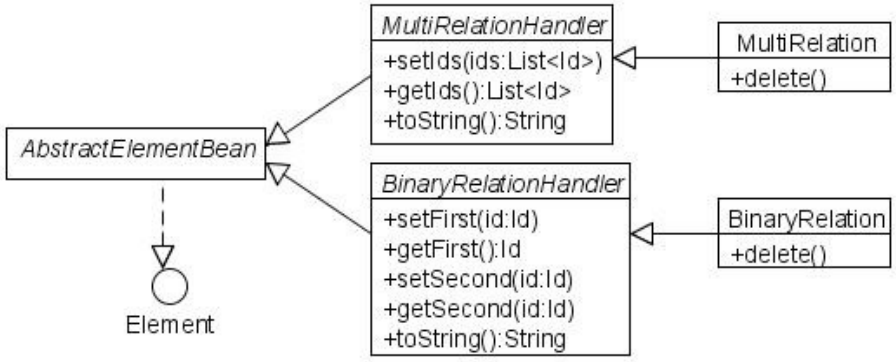

Abbildung11: Klassendiagramm für jasi.sim.basic.element.relation.

Tabelle 8: Klassen aus dem Paket jasi.sim.basic.element.relation.

Klasse	***Beschreibung***
MultiRelation	Relationen von Simulationen
MultiRelationHandler	Verwaltung der Relationen von Simulationselementen.
BinaryRelation	Binäre Relationen von Simulationselementen.
BinaryRelationHandler	Verwaltung der binären Relationen von Simulationselementen.

4.2.1.5 jasi.sim.basic.event (Black Box Beschreibung)

Diese Paket stellt die Schnittstellen und Basisklassen für Simulationsereignisse und deren Verarbeitung zur Verfügung. (siehe Tabelle 9 und Abbildung 12). Der Simulationskalender enthält eine sortierte Liste der noch zu bearbeitenden Simulationsereignisse. Sortierungskriterium der Simulationsereignisse bilden die Simulationszeit des Ereignisses und als weiteres Kriterium dessen Priorität.

Jedes Simulationsereignis enthält eine Methode *inform():boolean*, deren Rückgabewert bei der Abarbeitung eines Ereignisses bestimmt, ob angeschlossene externe Komponenten über den Simulationsproxy über dieses Ereignis informiert werden. Eine nähere Beschreibung der Verarbeitung und der Zustandsänderungen der Simulationsereignisse findet sich in der Laufzeitsicht (siehe auch 5.3Verarbeitung von Simulationereignissen).

Tabelle 9: Klassen aus dem Paket jasi.sim.basic.event.

Klasse	***Beschreibung***
Calendar	Simulationskalender.
CalendarListener	Beobachter von Änderungen im Simulationskalender.
Event	Schnittstelle für alle Simulationsereignisse.
AbstractEventBean	Verwaltung der Simulationsereignisse (siehe auch Abbildung 22 auf Seite 25).
EventListener	Verarbeitung der Simulationsereignisse.
AbstractRepeatingEvent	Wiederholbare Simulationsereignisse (siehe auch Abbildung 23 auf Seite 26).

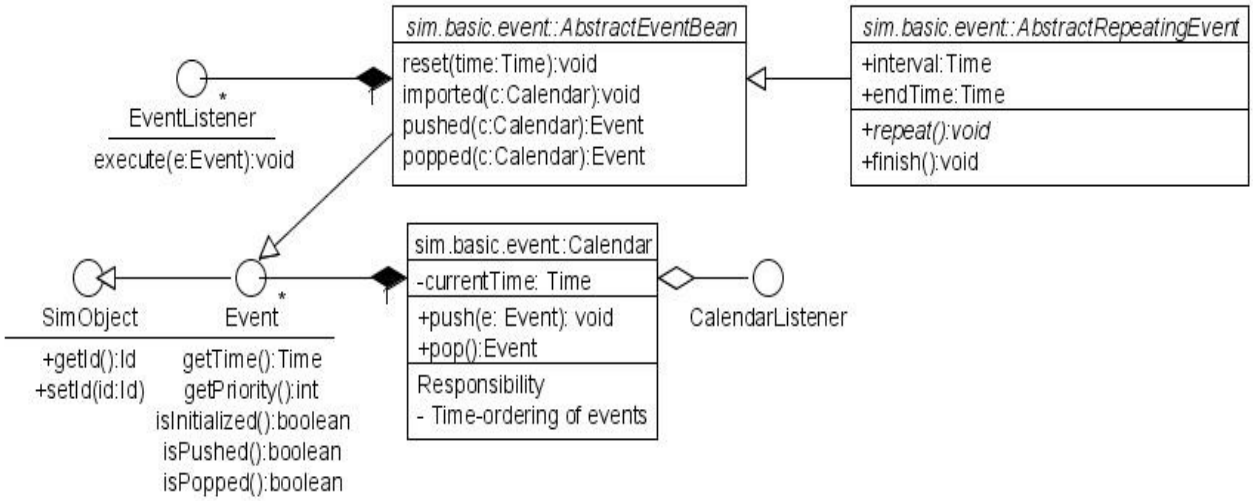

Abbildung 12: Klassendiagramm für das Paket jasi.sim.basic.event*.*

4.2.1.6 jasi.sim.basic.table (Black Box Beschreibung)

Die Klassen dieses Paketes stellen Datenstrukturen bereit, auf die innerhalb des Simulationsprozesses zugegriffen werden kann, und die als Bestandteil des Simulationszustandes im Archiv gespeichert werden (siehe Abbildung 13). Eine Entscheidungstabelle *ConditionTree* enthält Datenstrukturen als eine Abbildung von hierarchischen Schlüsseln zu Daten. Falls beim Zugriff auf die Daten einer Entscheidungstabelle ein Schlüssel nicht vorhanden ist, besteht die Möglichkeit für diesen Fall einen Ersatzwert auf allen Hierarchieebenen bereitzustellen. Mit Hilfe von Objekten der geschachtelten Klasse *ConditionTree.Line* werden die Entscheidungstabellen mit Schlüsseln und Werten befüllt, während die Entscheidungstabellen mit indizierten Listen von indizierten Listen *ConditionMap* verwaltet werden.

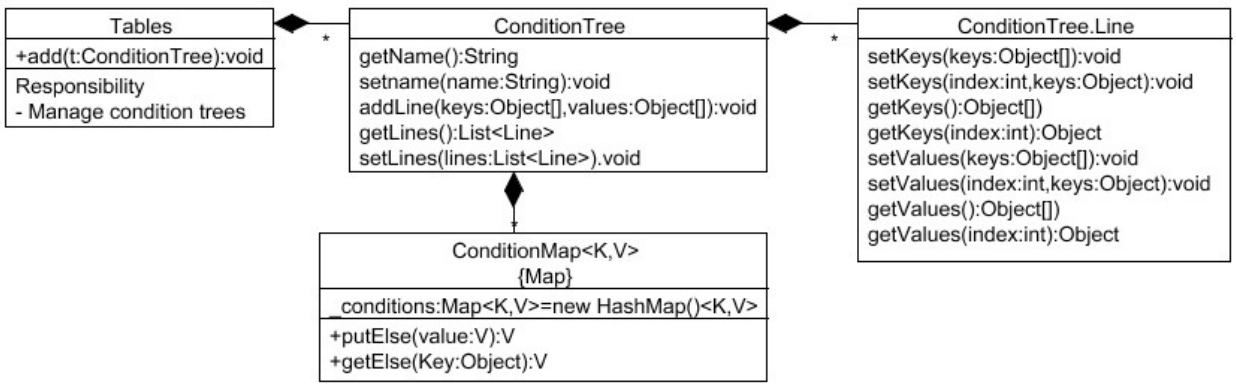

Abbildung 13: Klassendiagramm für das Paket jasi.sim.basic.table*.*

4.2.1.7 jasi.sim.basic.control (Blackbox-Beschreibung)

Die Schnittstellen und Basisklassen (siehe Abbildung 14) dieses Paketes erlauben das Einbinden externer Komponenten in den Simulationsrahmen und dessen Steuerung. Eine ausführliche Beschreibung findet sich in der Kontextsicht (siehe auch 3.1.2Einbindung externer Komponenten).

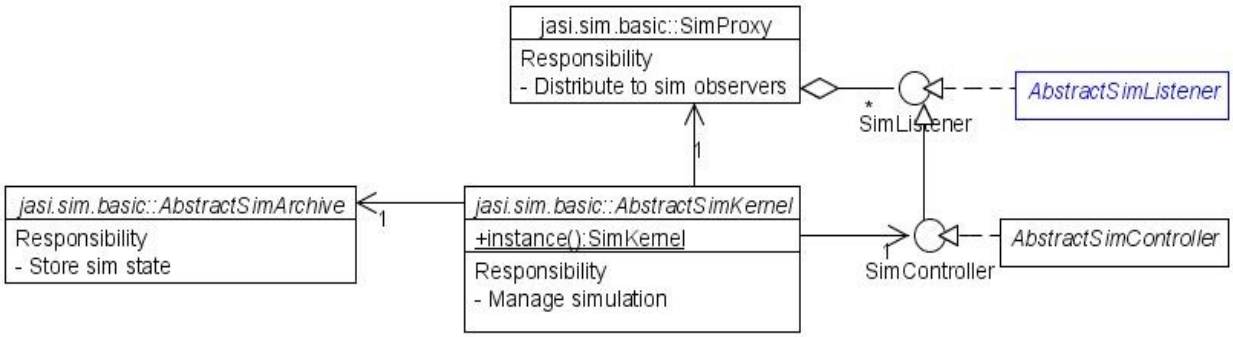

Abbildung 14: Klassendiagramm für das Paket jasi.sim.basic.control*.*

4.2.1.8 Beschreibung der Beziehungen

In diesem Kapitel werden die Beziehungen der Komponenten untereinander in dem Paket *sim.basic* beschrieben. Diese sind im Klassendiagramm der wesentlichen Klassen des Simulationsrahmens (siehe Abbildung 15) dargestellt.

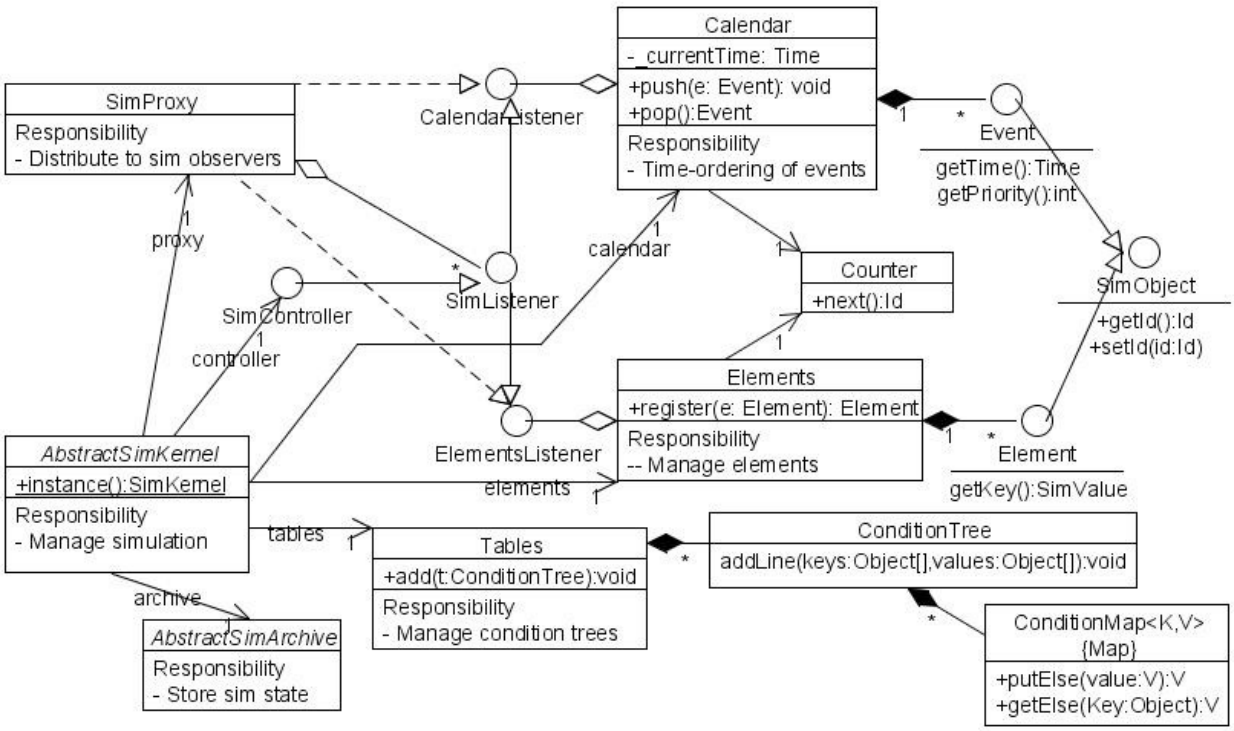

Abbildung 15: Klassendiagramm des Simulationsrahmens mit den Beziehungen.

Im Objektdiagramm des Simulationsprozesses (siehe Abbildung 16) werden die Beziehungen zwischen den Objekten bei der Abarbeitung von Simulationsereignissen beschrieben. Über sen Simulationsproxy werden die externen Komponenten der Simulation über die Bearbeitung jedes Ereignisses informiert, wenn es nicht über die *inform()*-Methode als ein internes Ereignis definiert ist.

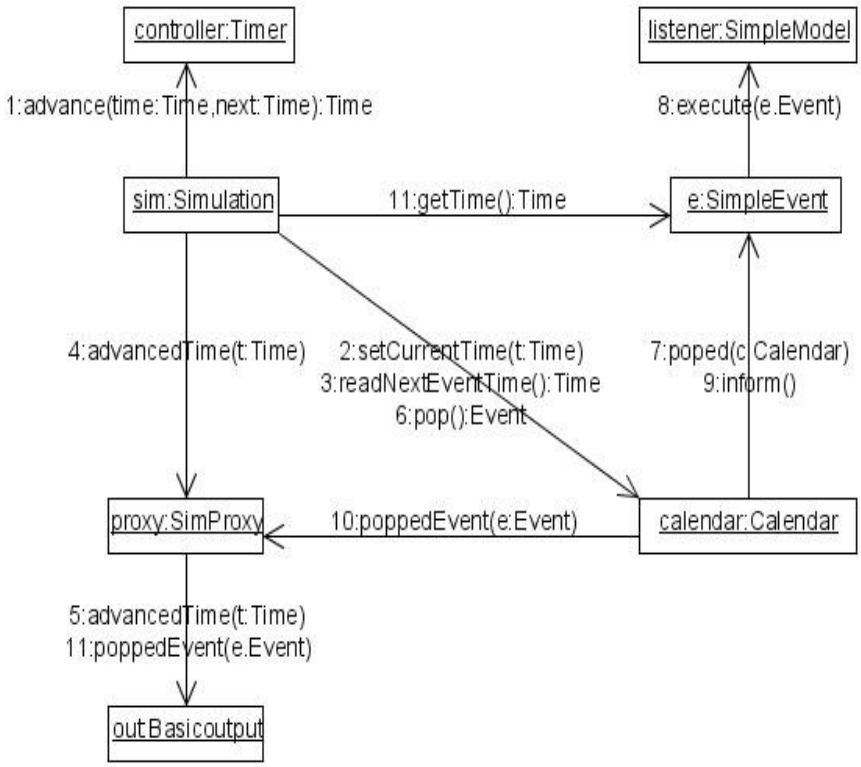

Abbildung 16: Objektdiagramm des Simulationsprozesses.

4.2.1.9 Offene Punkte

Zurzeit keine offenen Punkte.

4.2.2 Das Paket *jasi.sim.geo* (Blackbox- und Whitebox-Beschreibung)

Im Paket *jasi.sim.geo* werden Klassen zur Abbildung geometrischer Objekte (Ort, Geschwindigkeit) und Bewegungen im dreidimensionalen kartesischen Raum und auf einer Kugeloberfläche bereitgestellt. In Tabelle 10, 11 und 12 sind die wichsten Klassen diese Paketes und seiner Unterpakete beschrieben

Tabelle 10: Klassen aus dem Paket jasi.sim.geo*.*

Klasse	***Beschreibung***
ComparableGeoObject	Basisklasse von vergleichbaren geometrischen Objekten.
Geometric	Definition von Konstanten
GeoObject	Basisklasse von geometrischen Objekten.

Tabelle 11: Klassen aus dem Paket jasi.sim.geo.cartesian*.*

Klasse	***Beschreibung***
Direction	Einheitsvektoren im dreidimensionalen euklidischen Raum.
LinearMove	Geradlinige Bewegung im dreidimensionalen euklidischen Raum.
Move	Allgemeine Schnittstellen für Bewegungen.

Klasse	*Beschreibung*
Space	Allgemeiner Raumkoordinate, Abstand.
SpaceVec	Dreidimensionaler Raumvektor im dreidimensionalen euklidischen Raum.
Speed	Allgemeine Geschwindigkeit.
SpeedVec	Dreidimensionaler Geschwindigkeitsvektor im dreidimensionalen euklidischen Raum..

Tabelle 12: Klassen aus dem Paket jasi.sim.geo.spherical*.*

Klasse	*Beschreibung*
Arc	Kreisbogen.
Latitude	Breitengrad auf der Kugeloberfläche.
Longitude	Längengrad auf der Kugeloberfläche.
Coordinate	Sphärische Koordinaten auf der Kugeloberfläche.
SphericalMove	Bewegung auf der Kugeloberfläche.

4.2.2.1 Übersichtsdiagramm

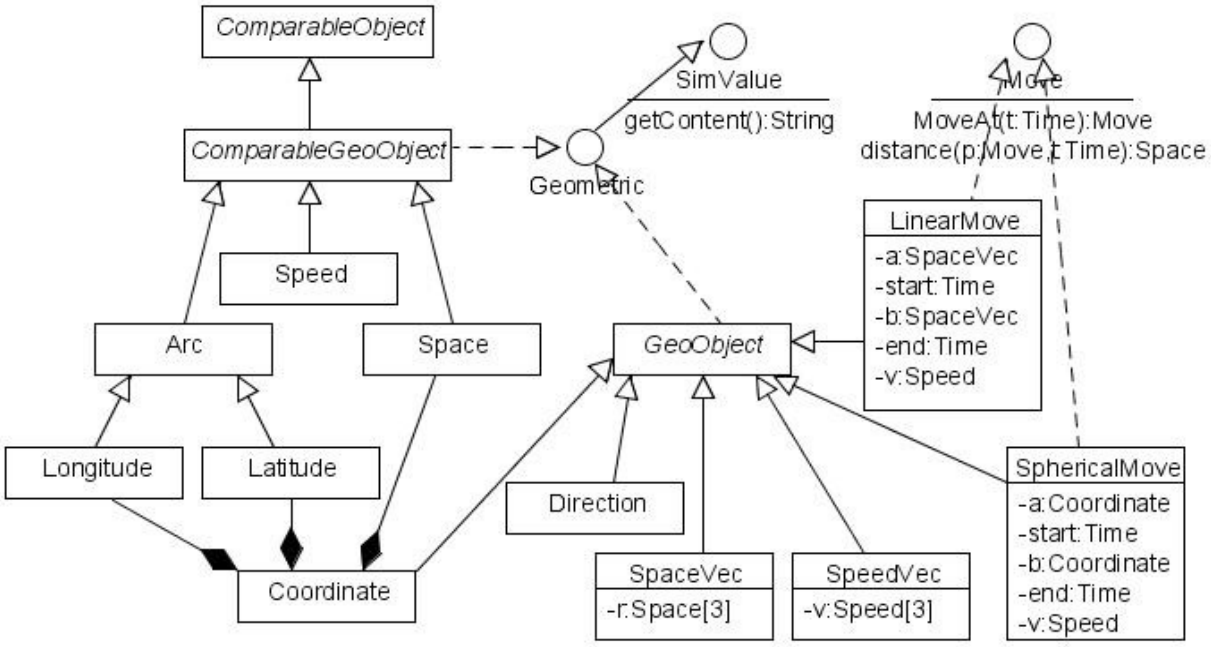

Abbildung 17: Klassendiagramm des Paketes jasi.sim.geo *und seiner Unterpakete.*

4.2.3 Das Paket *jasi.sim.user* (Blackbox- und Whitebox-Beschreibung)

Im Paket *jasi.sim.user* werden einfache, beispielhafte und möglichst allgemein verwendbare Implementierungen von Schnittstellen aus *jasi.sim.basic* bereitgestellt. Auf eine genauere Beschreibung wird hier im Rahmen dieser Dokumention verzichtet, da sie nicht zum wesentlich zum Verständnis des Simulationsrahmen beiträgt.

Fügen Sie hier die grafische Darstellung des Innenlebens von <Name-des-Bausteins> ein. Verwenden Sie für die Diagramme zum Beispiel UML-Klassen-, Komponenten- und Paketdiagramme.

5 Laufzeitsicht

Diese Sicht beschreibt, wie sich die Bausteine des Systems als Laufzeitelemente (Prozesse, Tasks, Activities, Threads, ...) verhalten und wie sie zusammenarbeiten .

5.1 Der Simulationsprozess

Der zeitliche Ablauf des Simulationsprozesses ist im Sequenzdiagramm von Abbildung 18 beschrieben. Über die Simulationssteuerung (*Timer*) wird zuerst der Fortschritt in der Simulationszeit bestimmt. Dieser ist durch Vorgabe einer oberen Schranke, durch das nächste Simulationsereignis oder durch externe Komponenten wie einer Simulationsföderation beschränkt. Eine Simulationsföderation hat hier die Möglichkeit externe Ereignisse in den Simulationskalender eintragen zu lassen. Hat die Simulationszeit den Zeitpunkt des nächsten Ereignisses erreicht, wird dieses vom Kalender geworfen und abgearbeitet (siehe auch 5.3Verhalten von Simulationereignissen).

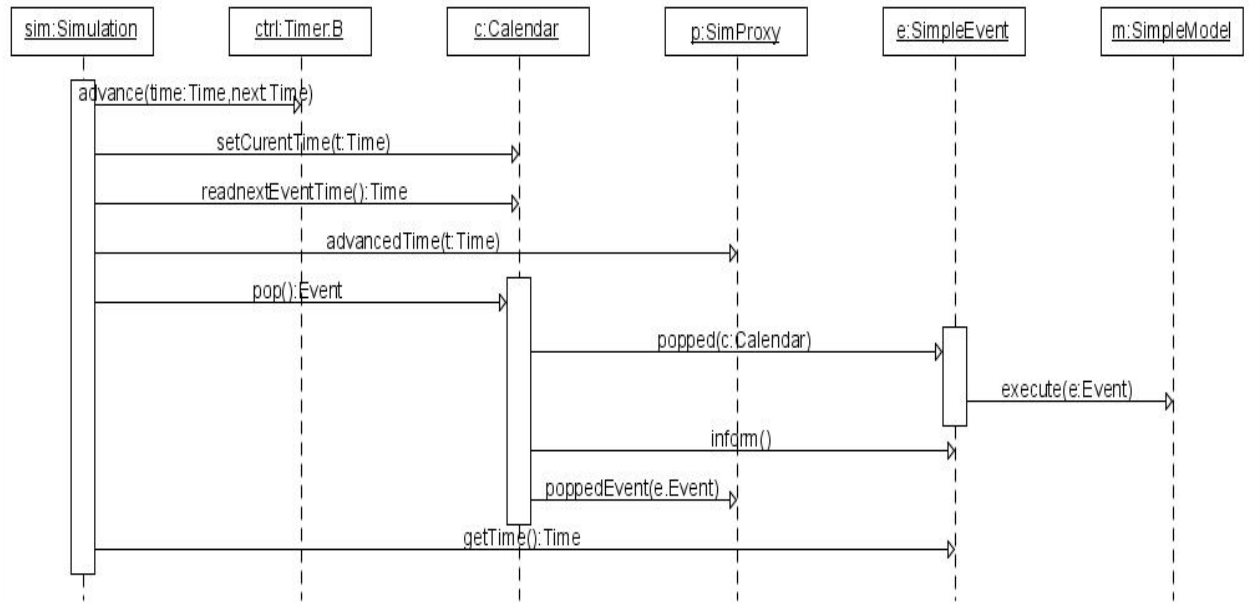

Abbildung 18: Sequenzdiagramm über das Fortschreiten des Simulationsprozesses.

5.2 Lesen und Ändern der Attribute von Simulationselementen

Das Lesen und Ändern der Attribute von Simulationselementen wird transparent für den Anwender von der Simulationssteuerung kontrolliert. Der Anwender hat die Möglichkeit eigene Implementierungen einer Simulationssteuerung bereitzustellen (siehe 3.1.2Einbindung externer Komponenten), und über den Simulationsproxy können ihm alle Zustandsänderungen mitgeteilt werden.

Die zeitliche Abfolge der Methodenaufrufe beim Lesen von Simulationselementattributen ist im Sequenzdiagramm von Abbildung 19 dargestellt. Beim Lesen von Attributen von Simulationselementen wird über die Simulationssteuerung ermittelt, ob der Wert des Attributes von einer externen Komponente zuvor aktualisiert werden soll. Wenn der Wert des Attributen dann geändert wurde, wird dieser neue Wert geliefert. Über den Simulationsproxy werden externe Komponenten darüber informiert.

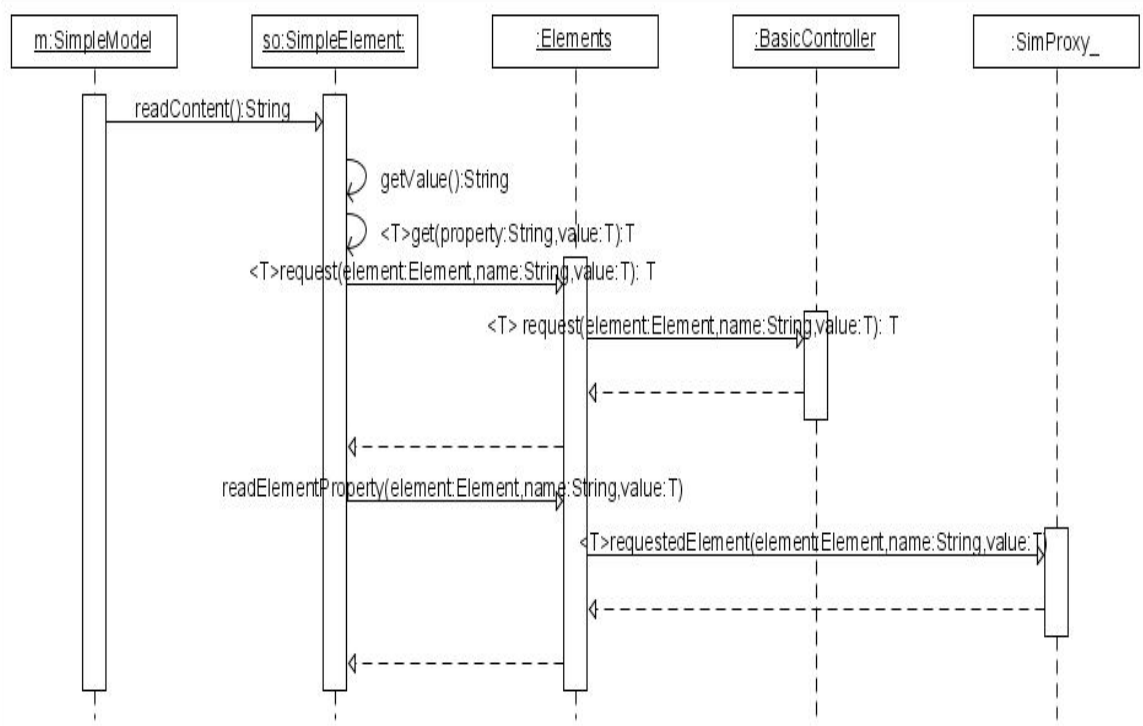

Abbildung 19: Sequenzdiagramm für das Lesen von Attributen der Simulationselemente.

Das Ändern von Simulationselementattributen ist im Sequenzdiagramm von Abbildung 20 dargestellt. Wie zuvor wird über die Simulationssteuerung geprüft, ob eine Änderung des Attributes durchgeführt werden kann, und wenn dies von der Simulationssteuerung abgelehnt wird, bleibt der Wert des Attributes unverändert.

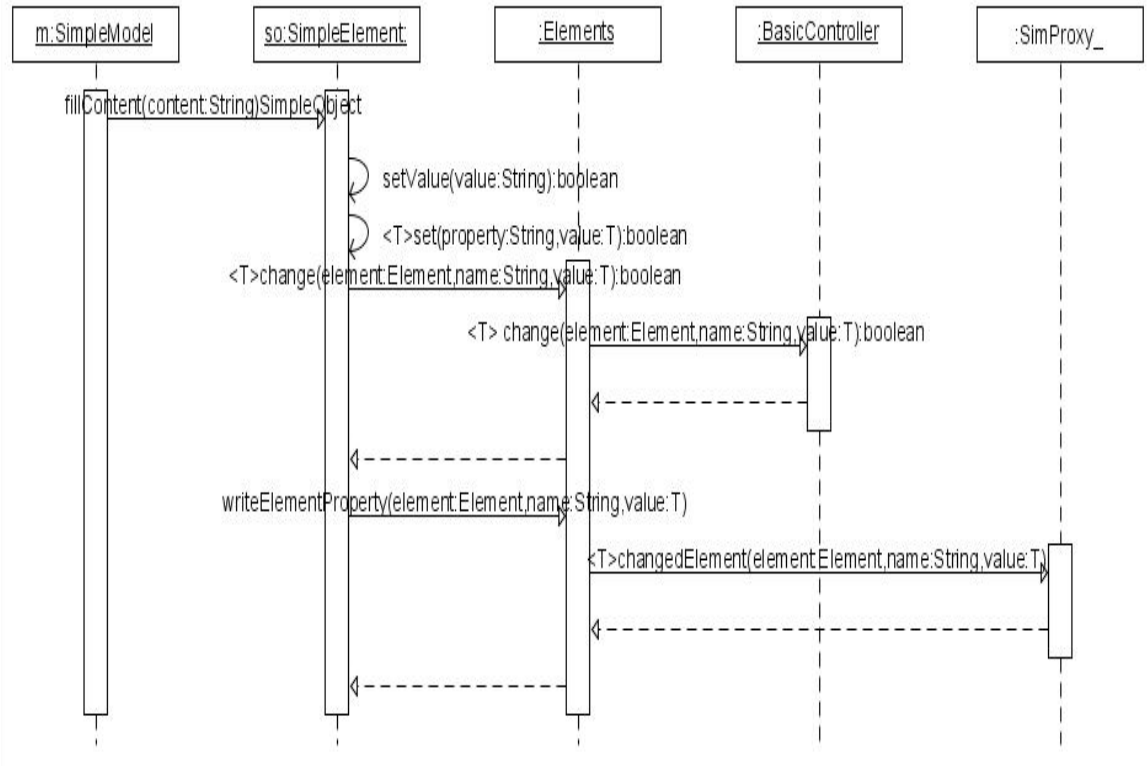

Abbildung 20: Sequenzdiagramm für das Ändern der Attribute von Simulationselementen.

5.3 Verhalten von Simulationsereignissen

Der zeitliche Ablauf der Wechselwirkungen (Methodenaufrufe) zwischen den Objekten ist im Sequenzdiagramm von Abbildung 21 dargestellt. Jedes Ereignis wird über das Beobachtermuster mindestens ein Simulationsmodell (Beobachter) zugewiesen, der beim Werfen des Ereignisses vom Kalender (*pop()*-Methode) für die Abarbeitung des Ereignisses zuständig ist. Wenn das Ereignis im Kalender eingetragen wird (*push(e:Event)*-Methode), werden die Ereignisse in einer Liste des Kalenders abgespeichert, nach Simulationszeit des Ereignisses und in zweiter Reihe nach deren Priorität sortiert und dann der reihe nach abgearbeitet.

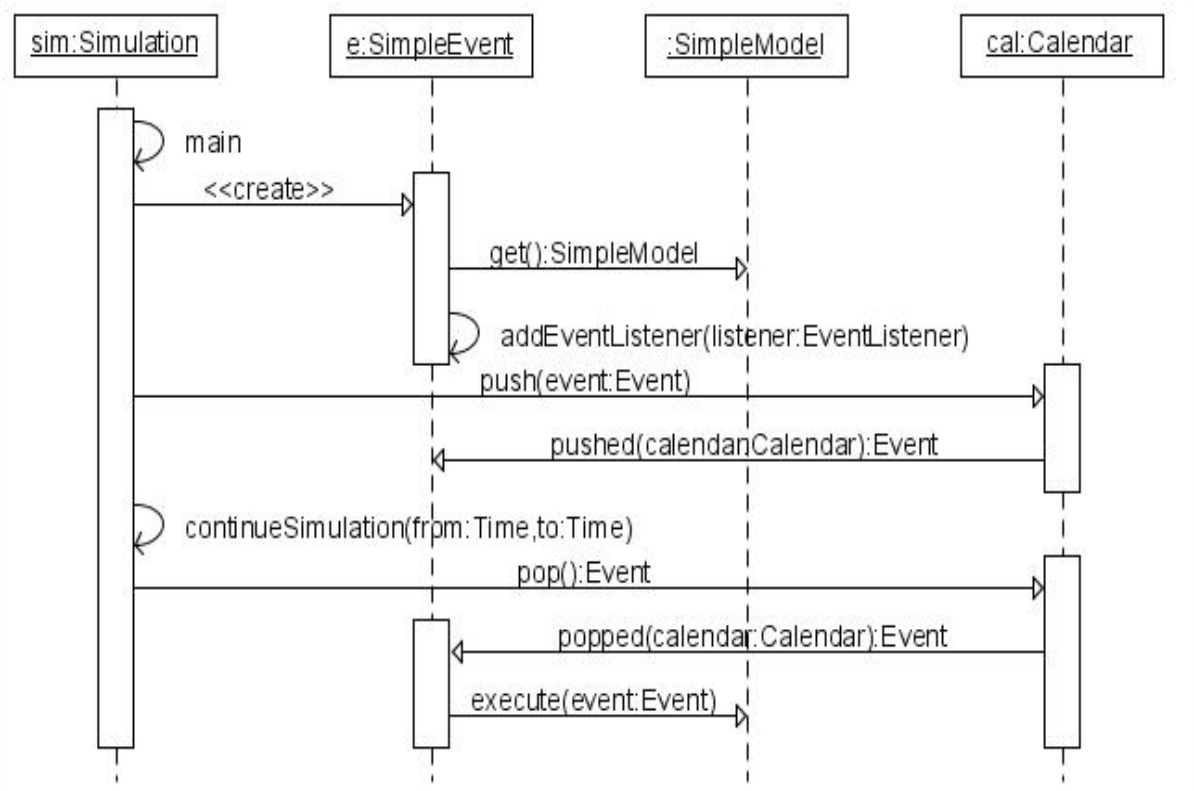

Abbildung 21: Sequenzdiagramm für das Verarbeiten von Simulationsereignissen.

Die Zustandsänderungen eines einfachen Simulationsereignisses sind im Zustandsdiagramm von Abbildung 22 dargestellt. Sobald ein Ereignis vom Kalender geworfen wurde, kann es nicht wieder verwendet und im Kalender eingetragen werden.

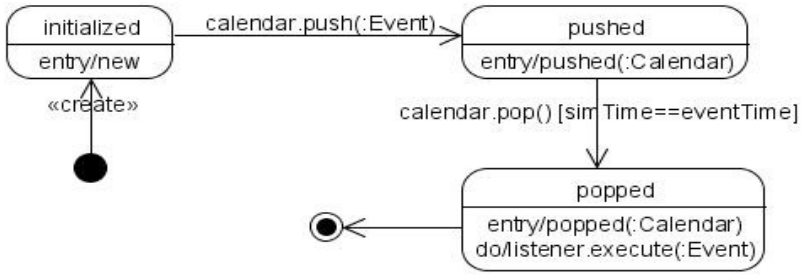

Abbildung 22: Zustandsdiagramm von einfachen Ereignissen.

Wiederholbare Simulationsereignisse dagegen können wieder im Kalender eingetragen werden. Anhand des im Ereignis eingetragenen Zeitintervalls wird der nächste Zeitpunkt für das Event berechnet und das Ereignis erneut im Kalender eingetragen. Mit Hilfe der *repeat()*-Methode eines Ereignisses können das Zeitintervall und die maximale Simulations-

zeit zuvor geändert werden, so dass das wiederholte Bearbeiten des Simulationsereignisse vorzeitig abgebrochen werden kann. Die Zustandsänderungen der wiederholbaren Ereignisse sind in Abbildung 23 dargestellt.

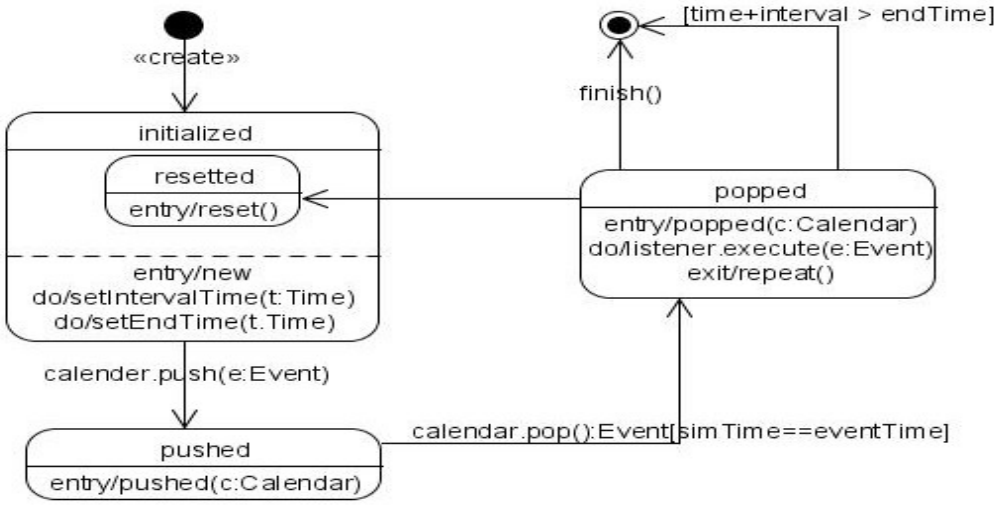

Abbildung 23: Zustandsdiagramm für wiederholbare Ereignisse.

5.4 Speichern und Lesen des Simulationsarchivs

Jeweils in einem Sequenzdiagramm werden die zeitliche Abfolge der Methodenaufrufe von eigenen Implementierungen eines Simulationsarchivs für das Lesen in Abbildung 24 und für das Schreiben in Abbildung 25 beschrieben.

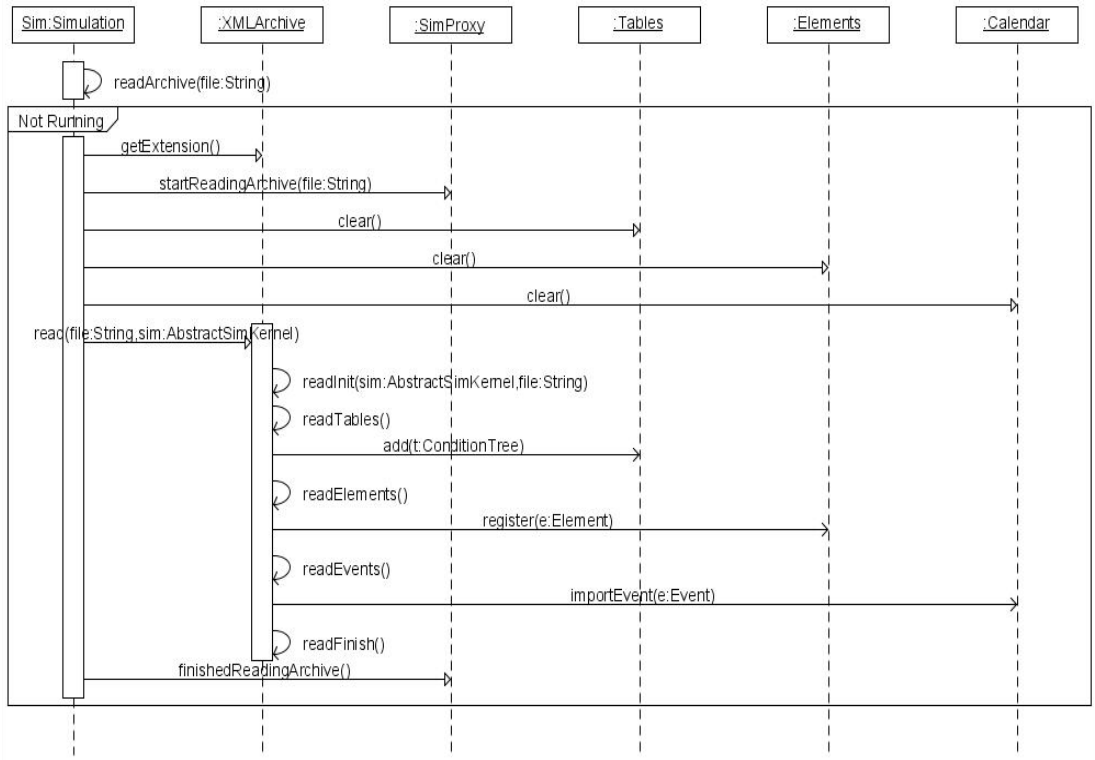

Abbildung 24: Sequenzdiagramm für das Lesen eines Simulationsarchivs.

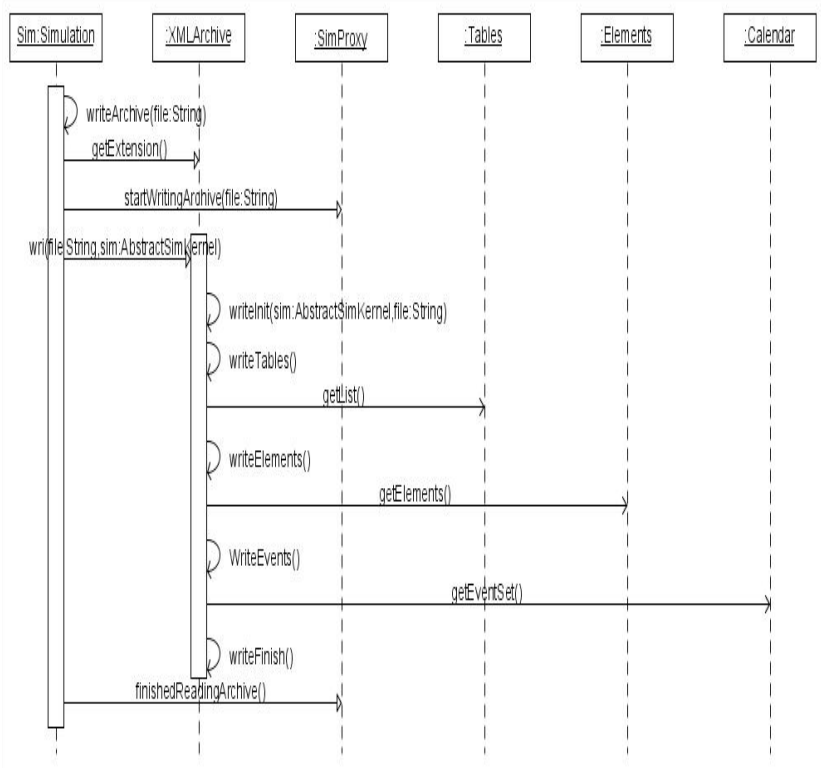

Abbildung 25: Sequenzdiagramm für das Schreiben eines Simulationsarchivs.

Die von der Simulationsrahmensoftware bereitgestellte Klasse *AbstractSimArchive* selbst ist abstrakt und alle Unterklassen müssen die abstrakten Methoden aus Tabelle 13 implementieren, die das Speichern und Lesen des Simulationszustandes in ein vom Anwender wählbares Format gewährleisten. Die Klasse *AbstractSimArchive* organisiert den Aufruf dieser Methoden.

Tabelle 13: Abstrakte Methoden zum Speichern und Lesen des Simulationszustandes.

Methode	***Beschreibung***
writeInit(SimKernel sim, String file)	Initialization of writing this archive, when exported to a file.
writeTables()	Write condition tables into archive.
writeElements()	Write simulation elements into archive.
writeEvents()	Write simulation events into archive.
writeFinish()	Finish writing this archive to a file.
readInit(SimKernel sim, String file)	Initialization of reading this archive, when imported from a file.
readTables()	Read condition tables from archive.
readElements()	Read simulation elements from archive.
readEvents()	Read simulation events from archive.
readFinish()	Finish reading this archive from a file.

6 Verteilungssicht

Diese Verteilungsicht beschreibt, in welcher Umgebung das System abläuft, die geographische Verteilung des Systems oder die Struktur der Hardwarekomponenten, auf denen die Software abläuft. Sie dokumentiert Rechner, Prozessoren, Netztopologien und Kanäle, sowie sonstige Bestandteile der physischen Systemumgebung. Die Verteilungssicht zeigt das System aus Betreibersicht.

Zur Entwicklung des Simulationsrahmens ist die folgende Software notwendig:

- Java 2 Platform Standard Edition 5.0/6.0 Development Kit (JDK 5.0/6.0)[4],
- Eclipse 3.1 oder höher, integrierte Entwicklungsumgebung[5],
- Apache Ant 1.6.2 oder höher, Java-basiertes Erzeugungswerkzeug[6],
- JUnit 3.8.1, einfacher Softwarerahmen für wiederholbares Testen[7],

Zum Betrieb des Simulationssystems sind mindestens der JDK 5.0, sowie die entsprechende Middleware oder Datenbanken eine Voraussetzung.

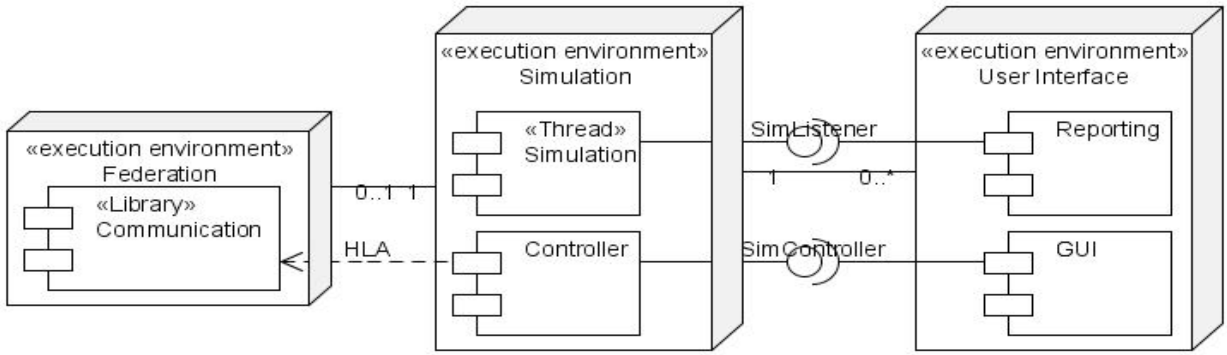

Abbildung 26: Verteilungsdiagramm der Ebene 1.

6.1 Infrastruktur des Gesamtsystems

Der Simulationsrahmen ist so konstruiert, dass verschiedene Komponenten auf andere Knoten verteilt werden können. In Java API werden zahlreiche Möglichkeiten einer Middleware, angefangen über JAVA RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture) bis zu den verschiedenen Web-Services. Bei Ankopplung von grafischen Anwenderoberflächen empfiehlt es sich eine Datenbank zur Speicherung der in der Oberfläche benötigten Daten zu verwenden, mit dem Ziel auch in diesem Fall eine weitgehende Unabhängigkeit zwischen der Simulation und den Oberflächen zu erhalten.

7 Entwurfsentscheidungen

Der Simulationssoftwarerahmen soll eine saubere Trennung zwischen der Modellierungssicht (domain model) und der Entwurfssicht (design model) im objektorientierten Entwicklungsprozess garantieren. Eine wesentliche Entwurfsentscheidung des Simulationsrahmens war die Trennung von Daten und Verhalten bei den Simulationsobjekten.

Die Daten der Simulationsobjekte bilden die eigentlichen Komponenten des Zustandsvektors der Simulation. Der Simulationsrahmen muss Funktionalitäten bereitstellen, diesen

4 Homepage http://java.sun.com/

5 Homepage http://www.eclipse.org

6 Homepage http://ant.apache.org/

7 Homepage http://junit.org/

Zustand vollständig und eindeutig in einer Datenbasis speichern und wieder lesen zu können. Die Trennung von Verhalten und Daten der Simulationsobjekte erfolgt bei:

- Simulationselementen durch Handler-Klasse (Daten) und Elementartklasse (Verhalten), siehe Abbildung 2,
- Simulationsereignisse durch Ereignisklassen (Daten) und Simulationsmodelle, den Ereignisbeobachtern, (Verhalten), siehe Abbildung 3.

Durch die vorgenommen Trennung von Daten und Verhalten kann die Entkopplung der Simulationsmodelle von den konkret implementierten Datenstrukturen zur Beschreibung des Simulationszustandes erreicht werden.

In den Simulationsmodellen ist das Verhalten der Simulationsobjekte von vorrangiger Bedeutung. Dazu werden innerhalb der Simulationsmodelle Schnittstellen definiert, auf die von den Modellen referenziert wird. Simulationselemente implementieren diese Schnittstellen und erhalten auf diese Weise ein durch die Simulationsmodelle induziertes Verhalten. Auf diese Weise eine komponentenorientierte Struktur der Simulationsmodelle. Die Simulationsmodelle sind so weitgehend unabhängig von den konkret implementierten Datenstrukturen.

8 Szenarien zur Architekturbewertung

8.1 Anwendungsfälle

In Abbildung 27 sind die Anwendungsfälle für diesen Simulationsrahmen dargestellt. Eine zentrale Anwendung ist das Setzen des Simulationszustandsvektors, der eindeutig eine Simulationszustand beschreibt. Innerhalb dieses Simulationsrahmens wird ein Simulationszustand eindeutig durch die Parameter von Simulationselementen und -ereignissen und den Inhalten von Datentabellen definiert. Zur Interaktion mit einer Simulation stellt der Simulationsrahmen die Fähigkeiten bereit, den Simulationszustand sowie Simulationsparameter, die das Laufzeitverhalten der Simulation beschreiben, zu Beginn der Simulation zu setzen und im Laufe der Simulation zu ändern.

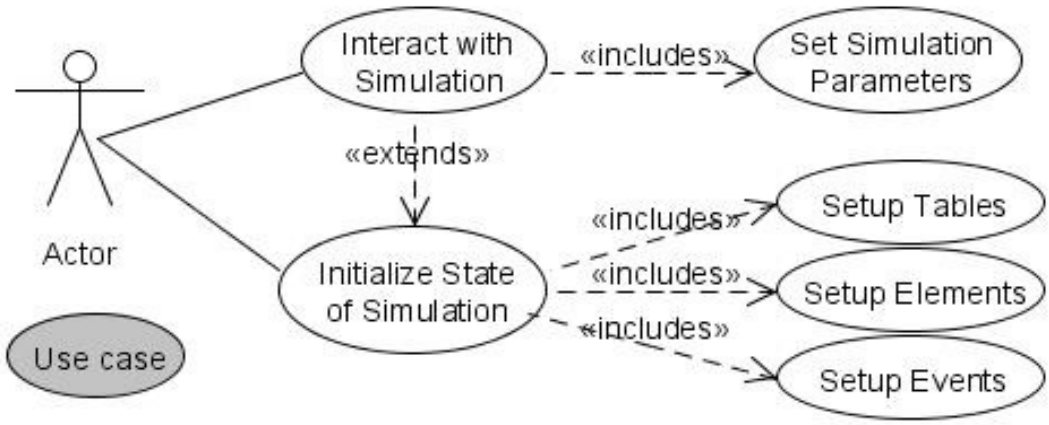

Abbildung 27: Anwendungsfälle des Simulationsrahmens.

8.2 Anwendungsbeispiel einer Luftverkehrssimulation

In diesem Kapitel ist exemplarisch die Anwendung der Simulationsrahmensoftware am Beispiel einer einfachen Simulation der Bewegungen im Luftverkehr zwischen Flugplätzen dargestellt. In Abbildung 28 ist die Erweiterung des Simulationskernels für diese Anwendung dargestellt. Die Klasse *TestAirtraffic* enthält die eigentliche Simulationsanwendung

und instantiiert den Simulationskernel, wie im Paket *jasi.sim.user* bereitgestellt wird. Die Anwendung stellt über die Klasse *AirtrafficArchive* die Archivierung der neuen Simulationsobjekte, wie sie in Abbildungen 29 und 30 dargestellt sind, im XML-Format bereit.

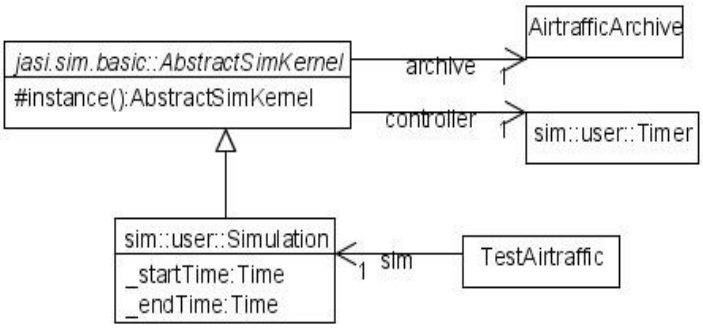

Abbildung 28: Das Paket jasi.airtraffic.kernel*.*

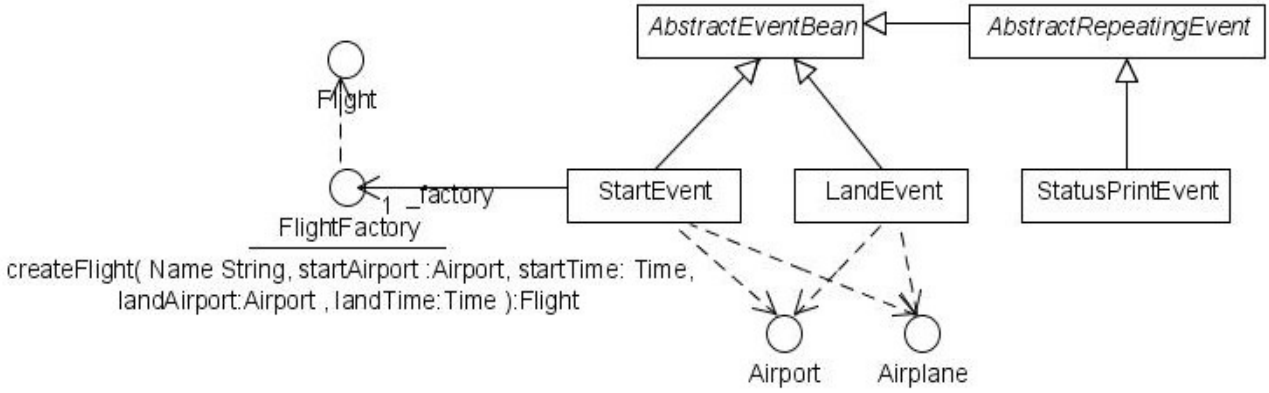

Abbildung 29: Das Paket jasi.airtraffic.model*.*

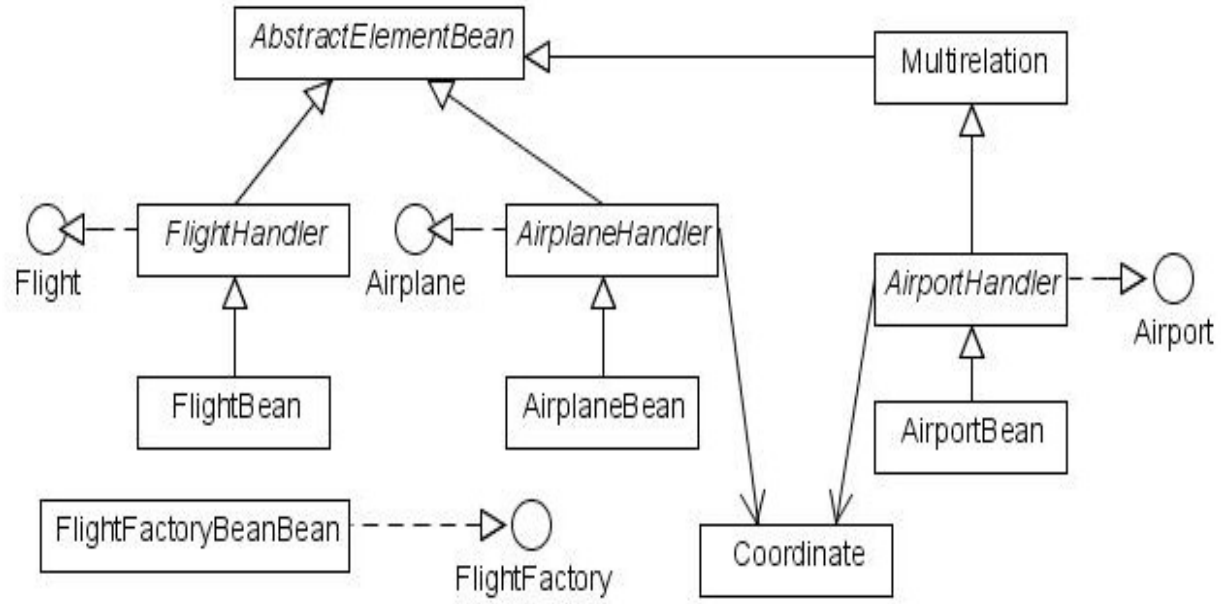

Abbildung 30: Das Paket jasi.airtraffic.defi*.*

9 Architekturaspekte

9.1 Persistenz

Der Simulationsrahmen muss die Speicherung des Simulationszustandsvektors ermöglichen, der aus den Attributen der Simulationselementen, Tabellen und im Kalender wartende Simulationsereignisse gebildet wird. Dies bedeutet, das die Tabellen und Simulationsobjekte persistent gespeichert werden müssen.

9.2 Benutzungsoberfläche

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.3 Ablaufsteuerung

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.4 Transaktionsbehandlung

Nicht zutreffend.

9.5 Sessionbehandlung

Nicht zutreffend.

9.6 Sicherheit

Sicherheitsaspekte spielen keine Rolle für diese Simulationsrahmensoftware, sondern sind möglicherweise ein Thema des Simulationssystems oder der Simulationsanwendung.

9.7 Kommunikation und Integration mit anderen IT-Systemen

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.8 Verteilung

Siehe 6Verteilungssicht.

9.9 Ausnahme-/Fehlerbehandlung

Durch die bei dem Simulationsproxy registrierten Simulationsbeobachter werden alle während des Simulationsprozesses im Simulationskernel auftretenden Ausnahmen und Fehler zur Anwendung des Simulationsrahmens weitergeleitet, und dort kann individuell und dem jeweiligen Kontext der Simulationsanwendung entsprechend auf diese regiert werden.

Siehe auch: 3.1.2Einbindung externer Komponenten.

9.10 Management des Systems & Administrierbarkeit

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.11 Logging, Protokollierung, Tracing

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.12 Konfigurierbarkeit

Siehe 3.1.2Einbindung externer Komponenten und 4.2.1.7sim.basic.control (Blackbox-Beschreibung).

9.13 Parallelisierung und Threading

Der eigentliche Simulationsprozess kann nicht auf einfache Weise parallelisiert werden, da in einem solchem Fall immer die Gefahr der Verletzung der Kausalität besteht, wenn zu einem Zeitpunkt Ereignisse zu verschiedenen Simulationsereignissen bearbeitet werden.

9.14 Internationalisierung

Nicht zutreffend für die Simulationsrahmensoftware, kann aber für Standardwerkzeuge und grafische Oberflächen von Bedeutung sein.

10 Projektaspekte

10.1 Change Request

Ein Konzept für Change Requests wurde noch nicht formuliert.

10.2 Technische Risiken

Die prinzipielle Verfügbarkeit der Java-Entwicklungswerkzeuge kann ein Risiko bilden. Da Java im Prinzip unabhängig von Plattform und Betriebssystem konzipiert ist, besteht im Bezug auf Plattform und Betriebssystem kein Risiko.

10.3 Offene Punkte

Sehe 10.1Change Request.

10.4 Erwartete Änderungen

Fehlerkorrekturen aus der Testphase.

10.5 Migration

Nicht zutreffend.

11 Auswirkungen

Auswirkungen des neuen Systems auf Betriebsumgebung, andere IT-Systeme, Umwelt und Benutzer werden zurzeit nicht erwartet.

12 Glossar

GUI (Graphical User Interface): Grafische Anwenderschnittstelle

HLA/RTI (High-Level Architecture, Run-Time Infrastructure): Middleware für Simulationsföderationen

Simulationssystem: Software zur Anwendung einer Simulation, die aus den Modellen, den Simulationen und den externen Komponenten, wie Visualisierungen, Anwenderschnittstellen, Auswertung oder Kopplungen mit anderen Systemen gebildet wird.

Simulationsarchiv: Chronologische Ordnungen von Simulationszuständen.

Simulationselement: Persistente Simulationsobjekte, deren Attribute Komponenten des Zustandsvektors der Simulation bilden.

Simulationsereignis: Temporäres Simulationsobjekt, das Änderungen des Simulationszustandes durch Berechnung in den Simulationsmodellen induziert.

Simulationsföderation: Kopplung von Simulationssystemen untereinander oder mit anderen Systemen.

Simulationskalender: Organisation und zeitliche Ordnung der Simulationsereignisse.

Simulationsmodell: Komponente eines Simulationssystems, das die Änderung eines Simulationszustandes mit Hilfe einer mathematischen Beschreibung nachvollziehbar ermöglicht.