4 Konzeption des Simulatorkerns

In der Anforderungsanalyse sind die Klassen und Prozesse zur Nachbildung des Transportsystems und der Arbeitsstationen aufgegriffen worden. Während bislang die Identifikation im Vordergrund stand, wird in diesem Kapitel das grundlegende Konzept des Simulatorkerns dargelegt. Zunächst wird in Abschnitt 4.1 der methodische Ansatz erläutert. Insbesondere wird die Auswahl einer prozessorientierten Simulationsmethode begründet. Zentraler Bestandteil dieser Methode sind Simulationsprozesse, die auf einer statischen Objektstruktur Zustandsveränderungen vornehmen. In Abschnitt 4.2 werden die Kriterien, die Simulationsprozesse auszeichnen, herausgestellt. Anschließend wird auf den Prozessablauf einzelner Modellelemente eingegangen, da sie die Erzeugung der Ereignisse und die damit verbundenen Zustandsänderungen wesentlich beeinflussen. Die Ereignisse werden sowohl von den Simulationsprozessen als auch von den Objekten ausgelöst, die nicht aktiv Zustandsänderungen veranlassen können. Die Verwaltung der Ereignisse in der Simulation wird in Abschnitt 4.3 thematisiert. Da der Simulatorkern auf einer gegebenen Objektstruktur Berechnungen zumeist ohne Ausgabe ausführt, benötigt der Kern eine flexible Möglichkeit, die Ein- und Ausgabe an externe Objekte weiterzuleiten. Diese Aspekte werden im letzten Abschnitt dieses Kapitels untersucht.

4.1 Methodischer Ansatz

Eine Simulationsmethode bestimmt die Art der Zustandsübergänge und die Fortschaltung der virtuellen Simulationszeit (vgl. Abschnitt 2.2.3). Zunächst ist zu ermitteln, welche Modellelemente zu Veränderungen des Systemzustands führen können. Danach ist zu prüfen, ob diese Übergänge kontinuierlich oder sprunghaft erfolgen. Die Analyse der dynamischen Prozesse des Transportsystems (Abschnitt 3.3.2) zeigt Zustandsänderungen bei:

- Der Position der Shuttles
- Der Einstellung variabler Streckenelemente
- Den Signalen der Kontrollelemente
- Den Arbeitsvorgängen an den Stationen

Das Auslösen der Signale an den Kontrollelementen wird als diskretes Ereignis betrachtet. Benötigt ein Signal in der realen Anlage das Übersteigen eines Schwellwertes, kann das Erreichen dieses Wertes durch eine Zeitverzögerung modelliert werden. Die relevanten Aktivitäten einer Arbeitsstation beschränken sich bei der Modellierung des Transportsystems auf aktiv oder nicht aktiv. Diese Zustände werden über Sensoren gesetzt und daher als diskrete Zustände angenommen. Analog ist der Einfluß der variablen Streckenelemente. Problematisch erscheint in diesem Zusammenhang die Behandlung der Position der Shuttles. In

der realen Anlage unterliegen diese kontinuierlichen Beschleunigungs- und Bremsvorgängen. Eine diskrete Simulationsmethode erlaubt hingegen nur sprunghafte Veränderungen des Systemzustandes. Der zu entwickelnde Simulator soll zur Validierung der Steuerungsprogramme eingesetzt werden. Zumal die Steuerung auf die Signale der Kontrollelemente reagiert, werden nur zwei Geschwindigkeitszustände berücksichtigt. Entweder fährt ein Shuttle mit voller Geschwindigkeit oder es hält. An dieser Stelle soll aufgezeigt werden, dass spätere Entwicklungen, die kontinuierliche Geschwindigkeitsvorgänge vorsehen, auch auf einer diskreten Simulationsmethode aufbauen können: Schienengebundene Systeme ermöglichen es, die Distanz zwischen einem Shuttle und der nächsten Systemkomponente, die eine Veränderung des Systemzustandes auslösen kann, genau zu ermitteln. Innerhalb statischer Streckenabschnitte ist diese Aussage offensichtlich, zumal ein gegenseitiges Überholen der Shuttles nicht möglich ist. Variable Abschnitte verändern den Streckenverlauf, dem ein Shuttle folgen muss. Daher kann sich eine variierende Distanz zum nächsten Ereignis ergeben. Durch die Steuerungskontrolle wird jedoch verhindert, dass ein Shuttle seine Bewegung fortsetzt, wenn die Streckenführung nicht eindeutig festgelegt ist. Nach Wiederaufnahme der Fahrt liegt eine statische Streckenführung vor. Durch Angabe einer Funktion zur Beschreibung der Brems- und Beschleunigungsvorgänge lässt sich der Zeitraum, der für das Zurücklegen der ermittelten Distanz erforderlich ist, berechnen. Da die Berechnungen des Simulatorkerns sich auf Zeitpunkte der Zustandsänderungen beziehen, können kontinuierliche Einflüsse berücksichtigt werden.

Ein Ansatz der diskreten Simulation besteht in der Zeitfortschaltung mit fixen Zeitinkrementen. Aus der technischen Spezifikation der Shuttles des vorliegenden Systems wird deutlich, dass Zeitinkremente von einer Millisekunde gewählt werden müssen. Dieses Zeitinkrement ergibt sich aus der Verfahrgeschwindigkeit des Shuttles von 0.5 m pro Sekunde und einer Anhaltegenauigkeit von 0.4 mm. Da zu jedem Zeitschritt neue Zustände für die Modellelemente berechnet werden müssen, kann bei dieser Methode mit einem schlechten Laufzeitverhalten gerechnet werden.

Bei der Simulation mit variablen Zeitinkrementen werden die ereignis-, aktivitäts- und prozessorientierten Methoden unterschieden. Jeder dieser Ansätze wird in der Literatur mit einer Objektorientierung in Verbindung gesetzt ([Pid95], [PP99], [FA96]) und weist Vor- und Nachteile auf, die in Abschnitt 2.2.3 aufgezeigt worden sind. Der prozessorientierte Ansatz wird dem Simulatorkern zugrunde gelegt, zumal die Systemkomponenten im realen System eigenständige Einheiten bilden, die auf äußere Signale reagieren. Beispielsweise reagiert ein Shuttle durch eine Sensorik auf zu einem zu geringen Abstand zum vorausfahrenden Shuttle. Die prozessorientierte Methode unterstreicht zudem den zyklischen Ablauf, nach dem sich die Komponenten verhalten [vgl. Lie95, S. 107]. Des Weiteren erlaubt dieser

Ansatz die Kapselung der Ereignisse, die ein Prozess im Simulationslauf ausführt und eine verteilte Simulation, bei geeigneter zeitlicher Koordination der Ereignisse. In [CDK94, S. 287ff] werden verschiedene Ansätze zur zeitlichen Koordination in verteilten Systemen behandelt. Im Hinblick auf die Anbindung einer realen Steuerung an die Simulation zeichnen sich Prozesse durch Nebenläufigkeit aus. Das Modell des Systems kann somit ähnlich wie das reale System reagieren.

4.2 Simulationsprozesse

Die prozessorientierte Simulation setzt zur Modellierung einzelner Systemkomponenten Prozesse ein. Zur zeitlichen Synchronisation wird eine Instanz (Scheduler) benötigt, die eine Verwaltung der Simulationsprozesse durchführt. Durch diese Instanz können Prozesse initiiert, aktiviert, unterbrochen oder beendet werden. Die Kontrollinstanz kann auf unterschiedliche Art realisiert sein. Erforderlich ist zum einen eine Datenstruktur, in der Referenzen auf die verwalteten Prozesse gehalten werden und zum anderen ein Algorithmus, der die Reihenfolge der Aktivierung bzw. Deaktivierung bestimmt. Der objektorientierte Ansatz bietet dabei die Möglichkeit, eine Klasse mit den entsprechenden Eigenschaften zu erzeugen. Eine Realisierung wird in Abschnitt 5.1 beschrieben.

Nach [Dud93] nehmen Prozesse einen der fünf Zustände: *initiiert, bereit, aktiv, blockiert oder beendet* an. Im Hinblick auf die Simulation werden im Folgenden die in Tabelle 4-1 angegebenen Zustände unterschieden. Die gewählten Bezeichnungen orientieren sich an [DCS99-ol].

Tabelle 4-1: Zustände der Simulationsprozesse

Zustand	**Bedeutung**
Aktiv	Der Prozess ist vom Ende der Warteschlange des Schedulers entfernt worden und wird zur Zeit ausgeführt.
Unterbrochen	Der Prozess ist in der Warteschlange eingereiht und wartet auf seine Aktivierung.
Passiv	Der Prozess ist nicht in der Warteschlange und wartet auf ein Ereignis, um in den Zustand *Unterbrochen* zu wechseln.
Beendet	Der Prozess ist nicht in der Warteschlange und kann auch nicht mehr aktiviert werden.

Erfolgt die Simulation als Reaktion auf reale Steuerungseinflüsse und reagiert das Simulationsmodell in Echtzeit, kann die Prozessverwaltung, des Betriebssystems eingesetzt werden und die Simulationsprozesse als nebenläufige Threads behandelt werden. Negativ wirkt sich bei diesem Ansatz das nicht deterministische Verhalten des Modells aus. Auf Grund dessen kann die Reproduktion der Simulationsergebnisse nicht gewährleistet werden. Insgesamt zeigt sich, dass unterschiedliche Kontrollinstanzen bereits implementiert worden sind, so dass in Bezug auf den zu entwickelnden Simulator auf eine bestehende Instanz zurückgegriffen wird.

Neben der Bereitstellung der Kontrollinstanz ist festzulegen, welche Komponenten durch Simulationsprozesse abgebildet werden müssen. Bevor in den nachfolgenden Abschnitten die einzelnen Prozessabläufe aufgezeigt werden, werden die Kriterien zur Auswahl der Prozesse erläutert, um anschließend die getroffene Auswahl zu begründen. Die Einschränkung der Simulationsprozesse soll ihre Anzahl während des Simulationslaufes gering halten.

Simulationsprozesse zeichnen sich durch folgende Punkte aus:

- Zyklische Bearbeitung zeitverbrauchender Aktivitäten
- Änderung des Systemzustandes
- Permanente Modellelemente

Die Anforderung einer zyklischen Folge der Aktivitäten ergibt sich aus dem prozessorientierten Ansatz (Abschnitt 2.2.3). Der zweite Aufzählungspunkt soll hervorheben, dass eine Abhängigkeit zwischen dem Simulationsprozess und dem Systemzustand besteht. Nach [Dud93] wird ein Prozess durch das Tripel $P=(S,f,s)$ definiert, wobei S ein Zustandsraum, f eine Aktionsfunktion und $s \subset S$ ein Anfangszustand ist. Der Zustandsraum bildet die Menge aller möglichen Zustände, die Zustandsvariablen annehmen können. Die Aktionsfunktion überführt durch Berechnungen die Anfangszustände in Folgezustände. Da der Zustandsraum durch die Attribute und Beziehungen der Klassen beschrieben wird, entspricht der Zustand, auf dem eine Aktionsfunktion ausgeführt wird, dem Systemzustand. Während bei [Lie95, S. 107] Prozesse temporäre Systembestandteile modellieren, die das System durchlaufen und danach wieder verlassen, bilden in dieser Arbeit Prozesse permanente Modellelemente des realen Systems. Die temporären Systembestandteile, die nach der Bearbeitung das Transportsystem verlassen, werden von den aktiven Prozessen generiert, transportiert und aus dem System entnommen.

Im Simulationsmodell sind, wie in dem nachfolgenden Abschnitt gezeigt wird, folgende Prozesse zu berücksichtigen:

- Shuttleprozess
- Variable Streckenmodule
- Schnittstelle zu den Steuerungsobjekten
- Arbeitsstationen

Die Kontrollelemente, die sich an der Fahrstrecke befinden, beeinflussen den Simulationslauf. Sie ändern den Systemzustand, indem sie die Position der Shuttles melden bzw. Fahrzeuge nach einem Signal von der Steuerung wieder starten. Weiterhin stellen sie feste Bestandteile des Systems dar. Damit werden zwei Kriterien, die zur Bestimmung der Simulationsprozesse dienen, erfüllt. Im Gegensatz zu den bisher aufgeführten Komponenten erfolgt die Änderung des Systemzustandes ohne zeitliche Verzögerung. Kontrollelemente erzeugen Ereignisse, die nach der Definition in Abschnitt 2.2.3 keine Zeit verbrauchen.

Die Meldungen der Kontrollelemente sowie die Kommunikation der Prozesse untereinander werden durch einen Mechanismus realisiert, der sich an das *Entwurfsmuster Befehl (Command)* aus [GHJ+96, S. 245] anlehnt. Dieses Konzept wird in Abschnitt 4.3 erklärt.

4.2.1 Der Shuttleprozess

Die Aktivitäten der Shuttles können auf zwei Ebenen betrachtet werden. Die übergeordnete Ebene ist der Materialtransport durch die Shuttles: Ein Shuttle wird durch die Steuerung einem Produkt zugeordnet und es fährt die Arbeitsstationen in der erforderlichen Reihenfolge an, bis das Produkt gefertigt ist. Diese Aktivität besteht aus mehreren Unteraktivitäten. Diese sind jeweils durch Ereignisse begrenzt, die zu Änderungen des Systemzustandes führen können. Erreicht ein Shuttle beispielsweise einen Haltepunkt, so wird es gestoppt und wartet, bis es wieder gestartet wird (Bild 4-1).

Der Verlauf eines Shuttleprozesses ist in Bild 4-2 dargestellt. Wird der Prozess vom Scheduler aktiviert, ruft das Objekt zunächst die Methode *calcNextTime* auf. Die Aufgabe dieser Methode ist zu ermitteln, wie groß die Zeitspanne bis zum nächsten Ereignis ist, das eine Zustandsänderung bewirken kann. Zur Berechnung der Zeit muss zum einen die Entfernung bis zum nächsten Objekt an der Strecke bekannt sein, zum anderen ist die Geschwindigkeit des Shuttles zu berücksichtigen. Während die aktuelle Geschwindigkeit in einem Attribut der Klasse *ShuttleType* gespeichert ist, hat das Shuttleobjekt keine Information über den Streckenverlauf. Im realen System folgt ein Shuttle der gegebenen Streckentopologie. Dieser Zusammenhang wird im Systemmodell durch eine Assoziation zwischen der Klasse *ShuttleType* und *AttachedTrack*, die Streckenabschnitte modelliert,

beachtet. Durch Aufruf der Methode *getNextElement* (Anhang A.2, Bild 8) erhält der Prozess ein Objekt, das sich aus einer eindimensionalen Sicht vor dem Shuttle auf dem Streckenabschnitt befindet. Der Abstand zu diesem Objekt entspricht der gesuchten Entfernung bis zum Eintritt des nächsten Ereignisses. Trifft das Shuttle bis zum Ende des Streckabschnitts auf kein weiteres Element mehr, tritt das nächste Ereignis erst beim Wechsel des Streckenabschnitts ein.

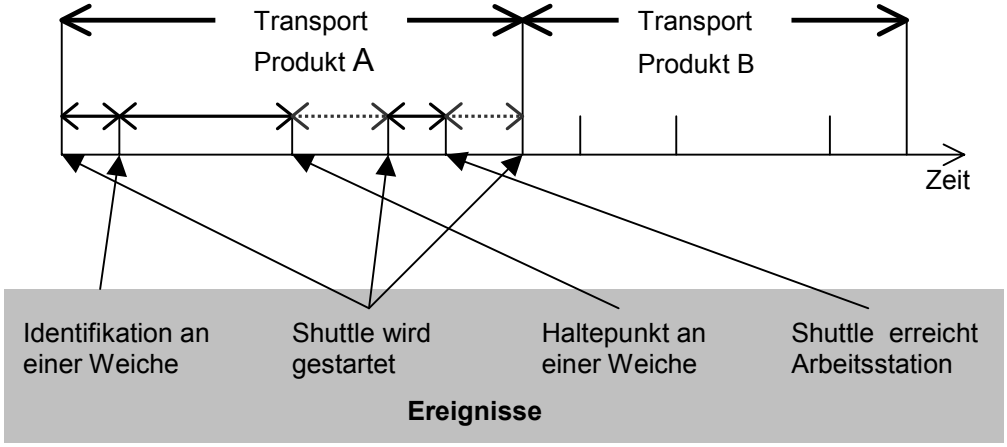

Bild 4-1: Aktivitäten eines Shuttles. Die Hauptaktivität des Transports der Produkte wird in Unteraktivitäten aufgeteilt, die durch Ereignisse begrenzt werden.

Einem Streckenabschnitt können Objekte verschiedener Klassen zugewiesen sein. Um eine Erweiterbarkeit der Klassentypen zu gewährleisten, müssen diese die Schnittstelle *IPositionEvent* implementieren. Diese Schnittstelle sieht Methoden vor, die es dem Shuttleprozess ermöglichen, den Zeitpunkt des Auslösens eines Ereignisses zu bestimmen und die Erzeugung eines Befehls zur Kommunikation zu veranlassen. Grundsätzlich stimmen die Werte der Position und der Ausführung des Ereignisses überein. Im Falle eines vorausfahrenden Shuttles ist jedoch zu beachten, dass ein gewisser Abstand zum vorausfahrenden Shuttle gehalten werden muss.

Hat der Shuttleprozess die Position des nächsten Ereignisses ermittelt, muss noch die Zeit bis zum Eintritt berechnet werden. Die aktuelle Version des Simulatorkerns bestimmt die Geschwindigkeit der Fahrzeuge mit den Kriterien *„mit maximaler Geschwindigkeit fahrend“* oder *„stehend“*. Die Zeitspanne bis zum nächsten Ereignis ergibt sich aus dem Quotienten des Weges und der Geschwindigkeit. Werden in zukünftigen Modellierungen Beschleunigungs- und Bremsvorgänge berücksichtigt, ist dieser Quotient durch eine Funktion zu ersetzen. Diese Funktion ermittelt abhängig von der momentanen Geschwindigkeit über das Integral einer Funktion, welche Beschleunigungs- bzw. Bremsvorgänge beschreibt, die zum Zurücklegen der Strecke bis zum nächsten Element benötigte Zeit. Proble-

matisch erweist sich in diesem Zusammenhang die Bestimmung der Zeitpunkte, an denen Ereignisse zur Geschwindigkeitsänderungen ausgelöst werden. Insbesondere erschwert sich die Verwaltung der Ereignisse bei differierenden Beschleunigungs- und Bremszeiten. Durch das Diskretisieren dieser kontinuierlichen Vorgänge erhöht sich die Zahl möglicher Zustandsänderungen und damit der Ereignisse und der Laufzeit. Es ist demnach ein Kompromiss zwischen der erforderlichen Genauigkeit der kontinuierlichen Vorgänge und dem Laufzeitverhalten zu finden.

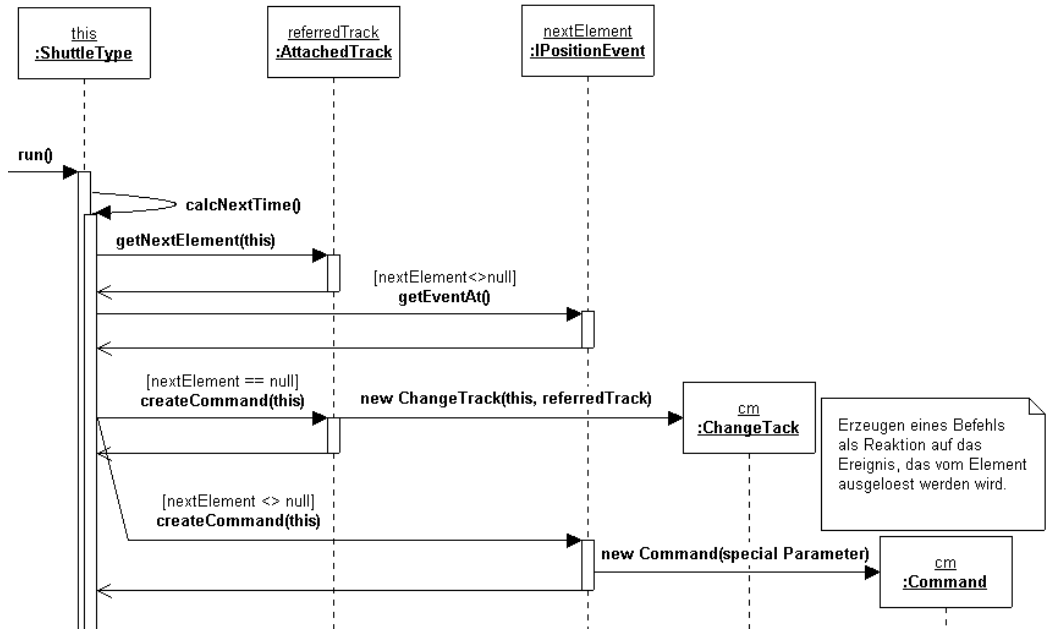

Bild 4-2: Ausschnitt des Sequenzdiagramms zur Beschreibung des zeitlichen Ablaufes eines Shuttleprozesses. Das vollständige Diagramm ist dem Anhang A.2 (Bild 7) zu entnehmen

Nachdem die Zeit berechnet worden ist, wird ein Befehl erzeugt, der eine Reaktion auf ein Ereignis am berechneten Zeitpunkt nachbildet (Abschnitt 4.3). Danach wird der aktuelle Prozess bis zum Eintritt des Zeitpunktes wieder deaktiviert.

Die Zustände des Prozesses sind mit denen der Systemkomponenten in Beziehung zu setzen. Der aktive *(acitve)* Zustand deutet an, dass ein Shuttle die Position eines möglichen Ereignisses erreicht hat und eine Änderung des Zustandes eintreten kann. Im passiven *(passive)* Status hält das Shuttle. Der Prozess muss von einem anderen Objekt das Signal zur Fortsetzung erhalten. Hat der Shuttleprozess die Zeitspanne bis zu seiner nächsten Aktivierung berechnet, erlaubt er der Kontrollinstanz, ihn zu unterbrechen *(Zustand suspended)*. Im realen System entspricht dieser Zustand der Bewegung des Shuttles von einer Systemkomponente zu einer anderen. Es ist zu beachten, dass nachfolgende Shuttles zur Berechnung des Weges bis zum nächsten möglichen Zustandswechsel die genaue Position des voraus-

fahrenden Shuttles benötigen. Daher ist der, seit dem Zeitpunkt der letzten Aktualisierung der Position des Shuttles, zurückgelegte Weg bei der Implementierung der Methode zur Sondierung der Shuttleposition zu beachten. Der Übergang der Prozesszustände eines Shuttleprozesses wird in Bild 4-3 gezeigt.

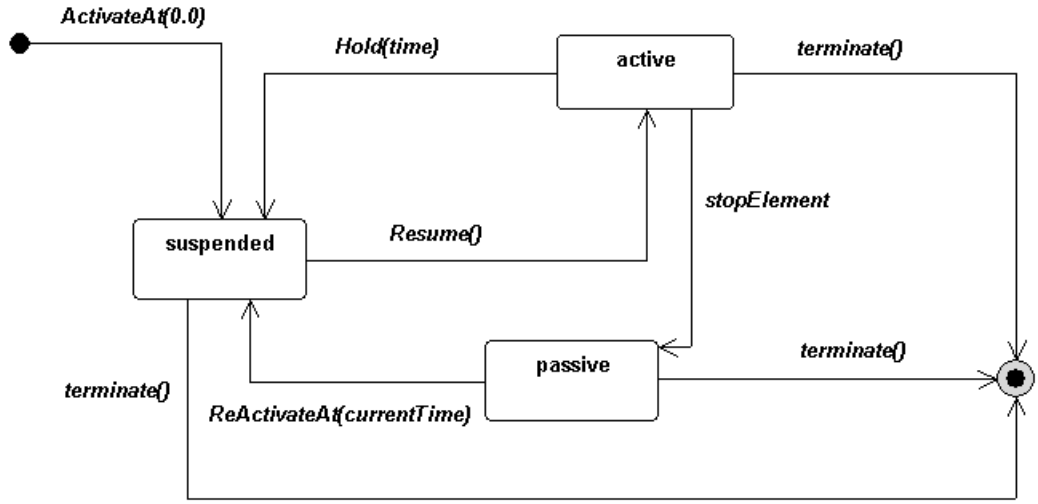

Bild 4-3: Zustandsdiagramm zur Darstellung der Phasen eines Shuttleprozesses. Die angeführten Methodenaufrufe werden von der Kontrollinstanz implementiert (vgl. Abschnitt 5.1).

4.2.2 Das Prozessinterface

Neben dem Datenaustausch ist die zeitliche Synchronisation der Steuerungsobjekte mit den Modellelementen des Kerns wichtig. Diese Aufgabe wird durch Prozesse erfüllt, die sich als Schnittstelle zwischen den beiden Architekturebenen befinden. Da die Steuerung auf Veränderungen des Systemzustandes reagiert, wird die Simulationszeit als grundlegende Zeit gewählt. Sendeereignisse der Sensoren, die im Simulationsmodell erzeugt und an die Steuerungsebene geleitet werden, sind durch die Koordination innerhalb des Kerns mit der Simulationszeit synchronisiert. Daher besteht die Aufgabe der Schnittstellenprozesse darin, Nachrichten von der Steuerung mit der virtuellen Zeit abzustimmen.

In diesem Abschnitt sollen die an der Kommunikation beteiligten Klassen und der zeitliche Verlauf erläutert werden. Ausgangspunkt ist das Klassendiagramm (Bild 4-4), das die Klassen mit ihren Beziehungen abbildet, welche dem Einfluss der Steuerung unterliegen. Wesentlich in der Abbildung ist die Klasse *IoInterface*. Instanzen dieser Klasse repräsentieren Objekte, die das Prozessinterface des realen Steuerungsknotens modellieren. Die Klasse *IoInterface* delegiert Informationen an die Objekte der Steuerung über die Klasse *ControlObject*. Die Sensoren als Systemkomponenten werden innerhalb der Modellelemente des Simulatorkerns

als Sensor Objekte modelliert. Jedes Objekt, das die *Actor*-Schnittstelle zur Verfügung stellt, kann auf eine Steuermeldung reagieren (Anhang A.1, Bild 4). Die Methode *startEvent* realisiert eine spezifische Reaktion eines Aktor-Objektes. So startet der Aufruf der Methode innerhalb eines Haltepunktes das wartende Shuttle, während die Methode für einen Antrieb eines Streckenmoduls die Einstellung des Moduls auf den übergebenen Wert setzt.

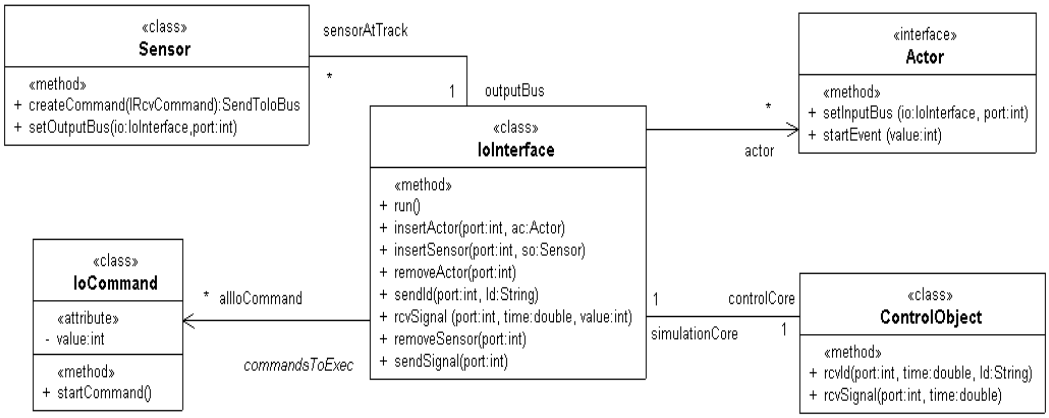

Bild 4-4: Klassendiagramm der Schnittstelle zwischen dem Simulatorkern und den Steuerungsobjekten. Das vollständige Diagramm ist im Anhang A.1 (Bild 5) dargestellt.

Erhält ein Schnittstellenprozess von der assoziierten Klasse *ControlObject* die Anweisung *rcvSignal*, wird eine Instanz der Klasse *IoCommand* gebildet. Dieses Objekt kapselt Informationen über den Empfänger der Nachricht und die auszuführende Operation (siehe Abschnitt 4.3). Abhängig von der virtuellen Simulationszeit und einer als Parameter übergebenen Verzögerungszeit wird die Ausführung des Sendeereignisses zurückgestellt. Nach Aufruf der *startCommand*-Methode wird der entsprechende Aktor veranlasst, die Aktion auszuführen, die von der Steuerung gewünscht ist. Das Bild 4-5 veranschaulicht den beschriebenen Vorgang.

Vor der Zeitgrenze im Sequenzdiagramm (Bild 4-6) wird ein Ereignis, das ein Signal in der realen Anlage modelliert, von einem Sensor über das Prozessinterface zum zugehörigen Objekt der Steuerung gesandt. Es werden Berechnungen vorgenommen, um auf den geänderten Systemzustand zu reagieren. Ist das Stellen eines Aktors erforderlich, liefert die Kontrollebene entsprechende Informationen an das Prozessinterface zurück. Ein entsprechender Befehl mit Operationen, die eine Reaktion auf das Ereignis im Simulationsmodell ausführen, wird instanziert und in eine Warteschlange eingefügt, die nach den Ausführungszeiten der noch nicht bearbeiteten Befehle sortiert ist. Dieser Schritt ist erforderlich, um mehrere Befehle im Prozessinterface puffern zu können und dabei nicht die Synchronisati-

on mit der Simulationszeit zu verletzen. Der Schnittstellenprozess wird unterbrochen, bis ein Befehl ausgeführt werden muss. Wird der Prozess reaktiviert, ruft die Hauptroutine die ausführende Methode des Befehlsobjektes auf, das in der Prioritätswarteschlange vorne steht. Nachdem der Befehl ausgeführt wurde, wird der verwaltende Prozess wieder passiviert. Das Zustandsdiagramm ist im Anhang A.3 (Bild 16) dargestellt. Dieses Szenario ist in Bild 4-6 nach der Zeitgrenze dargestellt. Erst nach einer erneuten Aktivierung kann der nächste Befehl abgearbeitet werden. Damit dieser Ablauf vollzogen werden kann, muss die Kontrollinstanz ein Umsetzen eines bereits innerhalb der Warteschlange eingereihten Prozesses unterstützen. Abhängig von der Spezifikation der verwendeten Instanz ist darauf zu achten, ob sich der Prozess nach dem Senden der Nachricht wieder neu einreihen muss. Wenn ein mehrfaches Vorkommen eines Prozesses in der Warteschlange unterstützt wird, darf sich der Prozess nicht wieder neu einreihen.

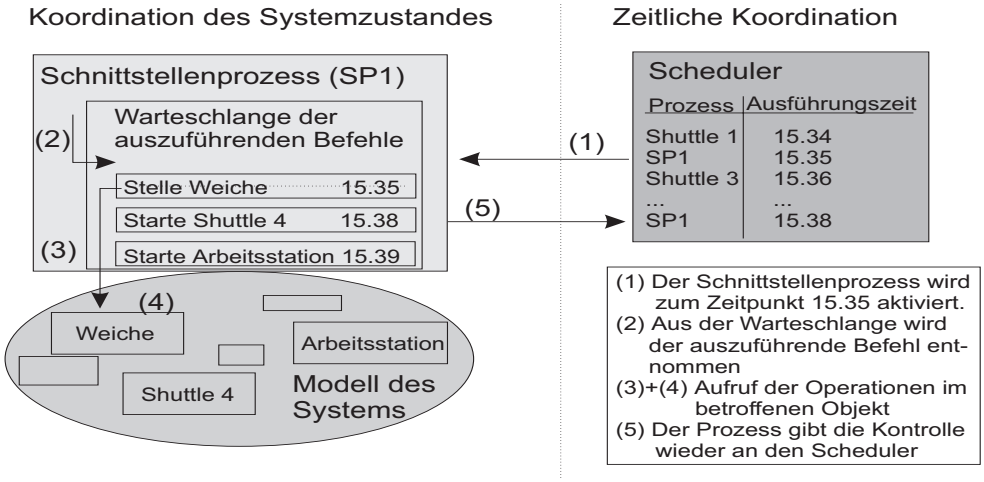

Bild 4-5: Unter der zeitlichen Kontrolle einer prozesskontrollierenden Instanz werden Veränderungen des Systemzustandes vorgenommen.

Das in Bild 4-5 dargestellte Szenario beruht auf der Annahme, dass der Simulationslauf keiner zeitlichen Beschränkung durch die Implementierung unterliegt. Dadurch werden die Ereignisse so schnell wie möglich berechnet. In diesem Fall kann eine Synchronisation mit den Objekten der Steuerung nur durch synchrone Methodenaufrufe erfolgen. Das Objekt der Klasse *IoInterface* wartet demnach, ob eine Reaktion hervorgerufen wird oder nicht. Dieser Ansatz benötigt zur Abbildung von Signalverzögerungen konstante Zeitspannen, die Objekte der Steuerung dem zugeordneten *IoInterface* mitliefern können.

Soll die Kommunikation asynchron erfolgen, ist es erforderlich, dass der Simulatorkern und die Modellelemente der Steuerung unabhängig voneinander zeitlich synchron ablaufen. Ein zeitliche Koordination der Änderungen des Systemzustandes auf Grund der Steuerung ist nicht erforderlich, da die Steuerung einer eigenen

Zeitverwaltung unterliegt. Die Schnittstelle sollte aber weiterhin als Simulationsprozess behandelt werden, um die Problematik der Nebenläufigkeit auf die Schnittstellenprozesse zu reduzieren.

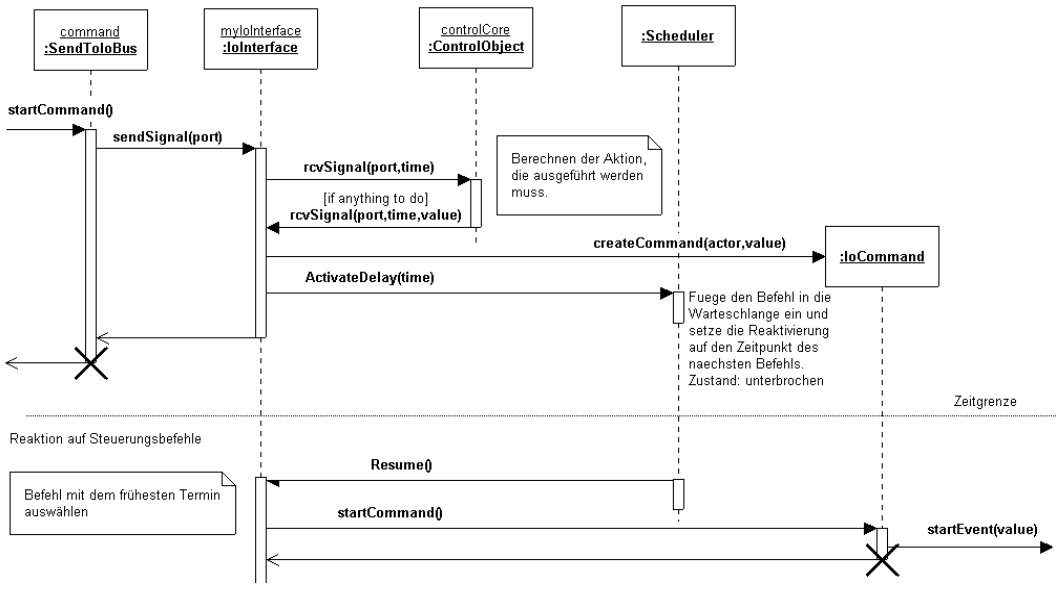

Bild 4-6: Sequenzdiagramm zur Kommunikation der Steuerung mit dem Kern

4.2.3 Variable Streckenelemente

Der Prozess zur Einstellung variabler Streckenelemente führt als Reaktion auf eine Nachricht von der Steuerung eine Veränderung der Streckenführung aus. Der Ablauf ist ähnlich wie der bei den im vorausgegangenen Abschnitt beschriebenen Schnittstellenprozessen. Erhält der Prozess ein Signal zum Einstellen der Streckenführung, wird er aktiviert. Daraufhin veranlasst er die zugeordneten Streckenteilabschnitte, ihre Streckenführung zu verändern und unterbricht sich selbst für die Zeitspanne, die zum Verstellen benötigt wird. Eine Unterbrechung des Prozesses führt dazu, dass der Prozess angehalten wird und unabhängig von einer Reaktivierung durch andere Prozesse an einem festen Zeitpunkt von der Kontrollinstanz wieder aktiviert wird. Nach der Reaktivierung wird ein Signal an die Steuerung geleitet, um das Ende des Vorgangs anzuzeigen.

Die Anforderungen der variablen Streckenelemente sind weniger an die Modellierung der zeitlichen Verzögerung als an die Aufrechterhaltung der Konsistenzbedingungen gestellt. Es ist Voraussetzung, dass nicht nur die Verbindungsstrukturen der Streckenteilabschnitte verändert werden, sondern auch die Reihenfolge der Shuttles verändert wird. Befindet sich ein Shuttle vor einer noch nicht richtig eingestellten Weiche, hat es zunächst keinen Vorgänger auf der vorliegenden Wegstrecke. Nachdem die Weiche eingestellt worden ist, muss das vorausfahren-

de Shuttle neu bestimmt werden. Andererseits kann die Veränderung der Streckenführung dazu führen, dass ein Fahrzeug keinen Vorgänger mehr hat. Erschwerend erweist sich, dass sich die Fahrzeuge nicht unmittelbar vor der Weiche befinden müssen. Zur Sicherung der Konsistenz ist demnach bei jeder Veränderung der Streckenführung für jeden betroffenen Teilabschnitt zu prüfen, ob sich ein Shuttle auf dem Streckenteil befindet. Im Fall einer Änderung der Eingangs- und Ausgangsverbindung sind mindestens vier Abschnitte zu testen. Wenn sich auf dem variablen Streckenabschnitt ein Shuttle befinden kann, beispielsweise bei einer Querverschiebung, ist auch für dieses Shuttle die Vorgängerbeziehung zu aktualisieren.

Die Aktivität des Prozesses zur Verwaltung eines variablen Streckenelements wird durch die Ereignisse zum Start und zum Beenden der Bewegung begrenzt, die in Hinblick auf den Simulator getrennt zu betrachten sind. Die Aktivität beginnt mit der Veränderung der Streckenführung. Dieses Ereignis ist an die Objekte der Benutzungsoberfläche weiterzuleiten, weil diese den Startpunkt einer Aktivität und die Dauer benötigen. Beide Informationen liegen zur Aktivierung des Prozesses vor. Das Bestätigungssignal am Ende des Vorgangs erlaubt es der Steuerung, wartende Shuttles wieder zu starten. Der beschriebene Ablauf ist im Anhang A.2 (Bild 10) abgebildet.

4.2.4 Arbeitsstationen

Die Arbeitsstationen arbeiten analog zu dem Prozess der variablen Streckenmodule. Aus dem in der Analysephase erstellten Klassendiagramm geht aber nicht hervor, wie ein Objekt einer Arbeitsstation Informationen an das Shuttle liefern kann, das gerade die zu bearbeitenden Teile transportiert hat. Der Zugriff der Arbeitsstation auf die zugrundeliegende Datenstruktur der Modellelemente des Simulatorkerns war maßgeblich für die Entscheidung, die Arbeitsstation als eigenständigen Prozess zu modellieren. Damit die Station diese Informationen einbringen kann, ist eine Erweiterung des Klassendiagramms erforderlich (Bild 4-7).

Voraussetzung für diese Anbindung ist, dass ein Shuttle an jeder Station anhalten muss. Nach Rücksprache mit einem Mitarbeiter der FASTEC ist diese Bedingung erfüllt. Veränderungen der Anzahl und Art der transportierten Teile während der Fahrt sind aus sicherheitstechnischen Gründen nicht durchzuführen. Somit kann einem Start-/Stoppelement eine Arbeitsstation zugeordnet werden. Das Start-/Stoppelement informiert die Station, sobald ein Shuttle dieses Element erreicht (eine Referenz auf das jeweilige Shuttle erhält das Start-/Stoppelement zur Generierung eines *Stopp-Befehls* (siehe 4.3)). Es ist zu beachten, dass diese Konstruktion nur in der Modellwelt vorgenommen wurde, um einem Objekt einer Arbeitsstation Zugriff auf das aktuelle Shuttle zu geben. In einer realen Anlage arbeiten

die Arbeitsstationen nebenläufig zum Transportsystem und sie kommunizieren über den lokalen Steuerungsknoten. Die Sequenzdiagramme zur Beschreibung des zeitlichen Verlaufes sind dem Anhang A.2 (Bilder 11 u. 12) zu entnehmen.

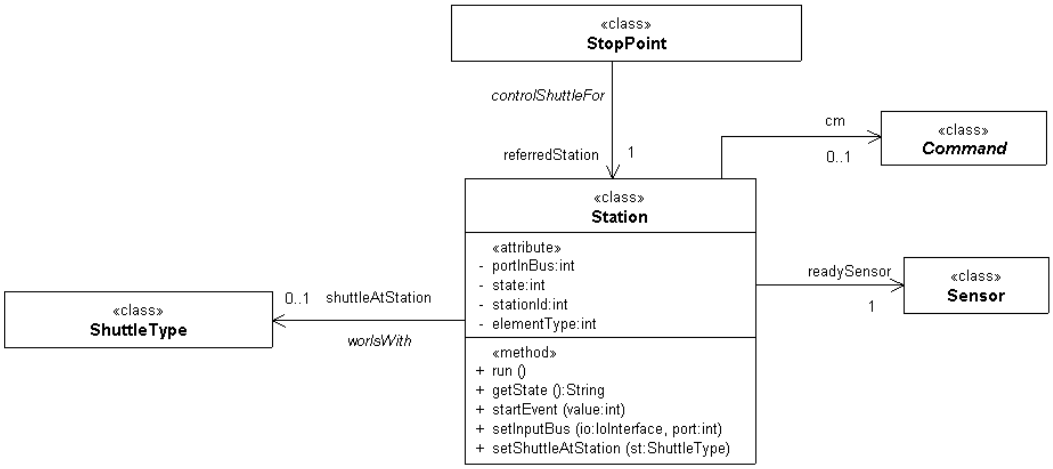

Bild 4-7: Das Klassendiagramm zeigt die Anbindung eines Prozesses zur Modellierung einer Arbeitsstation an die Modellelemente des Simulatorkerns.

4.3 Befehlsstruktur

Die Simulationsprozesse werden in der Prozesswarteschlange des Schedulers verwaltet. Tritt eine mögliche Zustandsänderung ein, werden sie aktiviert. Dagegen können Objekte, die nicht als Simulationsprozesse modelliert werden, nicht zeitlich koordiniert Ereignisse hervorrufen. Die Ereignisse sind aber für den weiteren Prozessverlauf bzw. für die Kommunikation der Prozesse miteinander entscheidend:

- Erreicht das Shuttle das Ende eines Streckenabschnitts, muss es auf den nächsten Abschnitt wechseln.
- Passiert es einen Sensor, muss den Steuerungsobjekten eine Nachricht übermittelt werden.
- An einem Haltepunkt muss das Shuttle anhalten und auf Signal der Steuerung wieder gestartet werden.

Aus dieser Aufzählung einzelner Ereignisse lässt sich folgern, dass viele Ereignisarten auszuführen sind, an denen unterschiedliche Objekte beteiligt sind. Des Weiteren sind die Ereignisse zeitlich koordiniert auszulösen. Da die Simulationsprozesse koordiniert ablaufen, werden die Ereignisse von ihnen in die Simulation eingebracht. Dieser Ansatz erhöht die Anforderungen, die an die Ereignisverwal-

tung gestellt werden, zusätzlich. Es kann nicht vorausgesetzt werden, dass dem aktiven Prozess eine Referenz auf das Objekt vorliegt, das vom Ereignis betroffen ist.

Nach den vorangegangenen Ausführungen ist ein Schema zu modellieren, das es passiven Objekten ermöglicht, Ereignisse zu erzeugen und zeitlich koordiniert in den Simulationslauf einzubringen. Ansatzpunkt zur Modellierung dieses Mechanismus ist ein Entwurfsmuster, das von [GHJ+96, S. 245] als *Befehlsmuster* beschrieben wird (Bild 4-8). Das Muster wird überwiegend zur Modellierung von Menüsteuerungen verwendet. Es erlaubt die Kapselung von Operationen. Außerdem benötigt das aufrufende Objekt keine Kenntnis über das empfangene Objekt. Weiterhin sollen durch den Einsatz des Entwurfsmusters die Modellelemente erweitert werden können. Wenn z. B. weitere Kontrollelemente eingeführt werden, kann die jeweilige Aktion durch einen geeigneten Befehl beschrieben werden. Um die allgemeine Erweiterbarkeit zu gewährleisten, wurden Schnittstellen in der Modellierung berücksichtigt, die Methoden für den Klienten und den Ausführer vorgeben (Anhang A.1, Bild 2).

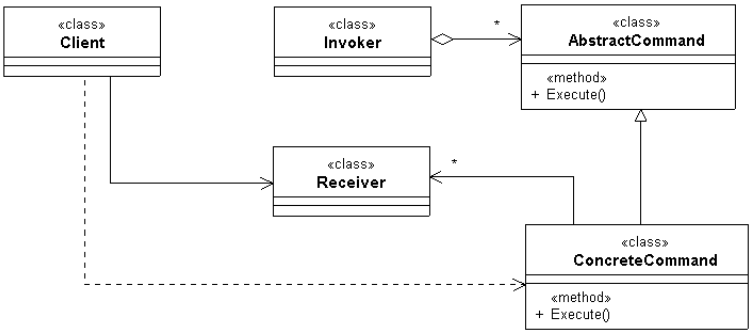

Bild 4-8: Entwurfsmuster Befehl (Command) [GHJ+96]

Ein Klient *(Client)* erzeugt einen konkreten Befehl und übergibt ihn dem *Aufrufer (Invoker)*. Der *Aufrufer* startet zum entsprechenden Zeitpunkt die *Execute*-Methode, die Aktionen innerhalb des Empfängerobjektes umsetzt. Im weiteren Verlauf dieses Abschnittes soll dargelegt werden, welche konkreten Befehle realisiert werden müssen und wer die Rollen des *Klienten*, *Empfängers (Receiver)* bzw. *Aufrufers* übernimmt.

Aus der bisherigen Darstellung ist ersichtlich, dass innerhalb des Simulatorkerns unterschiedliche Befehle implementiert werden müssen. In Bild 4-9 sind die konkreten Befehle aufgezeigt, die im Folgenden kurz beschrieben werden.

SendToIoBus und SendIdToIoBus

Die beiden Befehlstypen *SendToIoBus* und *SendIdToIoBus* werden durch Kontrollelemente, die an der Fahrstrecke liegen, erzeugt. Der Befehl *SendToIoBus*

wird von einem Sensor als Reaktion auf ein ankommendes Shuttle generiert, der *SendIdToIoBus*-Befehl von einer Identifikationseinheit. Die Ausführung des jeweiligen Befehls muss verzögert werden, da zum Zeitpunkt ihrer Instanzierung das Shuttle das Kontrollelement noch nicht erreicht hat (Abschnitt 4.2.1). Daher wird der Befehl im aktiven Shuttleprozess zwischengespeichert. Nachdem das Shuttle reaktiviert wurde, wird der abgelegte Befehl entnommen und durchgeführt. Der Empfänger der Operationen ist im diesem Fall das Prozessinterface, dem der Sensor zugeordnet ist und das durch ein Objekt der Klasse *IoInterface* modelliert wird. Andere Verwendung finden die Befehle, wenn eine Arbeitsstation das Beenden eines Auftrags an die Steuerungsobjekte meldet bzw. ein variabler Streckenabschnitt eingestellt worden ist.

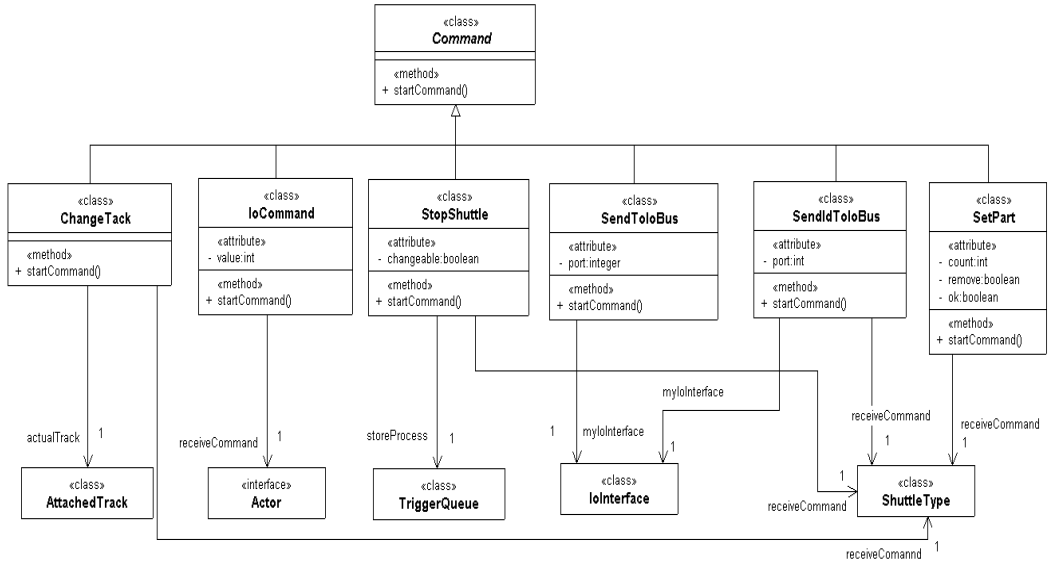

Bild 4-9: Das dargestellte Klassendiagramm zeigt die im Simulatorkern eingesetzten Befehle zur Reaktion auf Simulationsereignisse.

ChangeTrack

Der Befehl *ChangeTrack* bewirkt, dass ein Shuttle den aktuell befahrenen Streckenabschnitt verlässt und auf dem folgenden Abschnitt weiterfährt. Dieser Befehl wird erzeugt, wenn zwischen dem aktiven Shuttleprozess und dem Ende des aktuellen Streckenabschnitts kein anderes Element liegt. Da der Streckenabschnitt das Objekt generiert, fungiert dieser in der Rolle des Klienten. Der Empfänger ist das aktive Shuttle, zumal es bei der nächsten Aktivierung den Streckenabschnitt verlässt. Damit bei der Befehlsausführung eine Beziehung zum nächsten Abschnitt hergestellt werden kann, wird im Befehlsobjekt ein Verweis auf den aktuellen Abschnitt gespeichert. Nach der Aktivierung des Befehls, in diesem Fall durch den gleichen Shuttleprozess, wird im Shuttle-Objekt als Empfänger der Operationen die Methode *setNewTrack* ausgeführt. Nachdem der Befehl ausge-

führt worden ist, wird er gelöscht. Der beschriebene Ablauf wird in Bild 4-10 verdeutlicht.

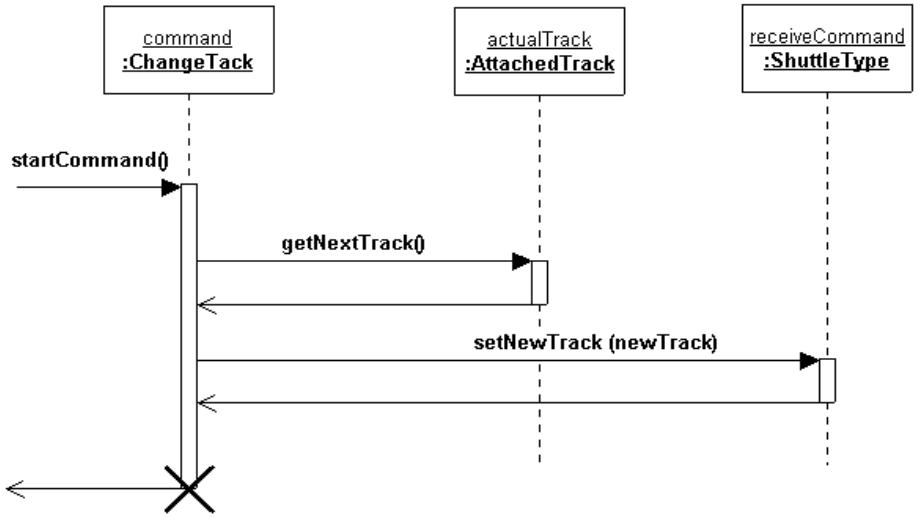

Bild 4-10: Sequenzdiagramm zur Bearbeitung des ChangeTrack Befehls

StopShuttle

Trifft ein Shuttle auf ein wartendes Shuttle bzw. muss es an einem Haltepunkt stoppen, sind Operationen zum Stoppen auszuführen. Bei der Umsetzung des Befehls *StopShuttle* wird der Shuttleprozess, der die Anordnung erhält, angehalten. Im Gegensatz zu der Unterbrechung des Prozesses für eine festgelegte Zeit kann ein angehaltener Shuttleprozess nicht wissen, wann er wieder reaktiviert wird. Daher wird dieser Prozess aus der Warteschlange des Schedulers entfernt. Damit der Prozess fortgeführt werden kann, wird er in einer Warteschlange des Objektes gehalten, das die Systemkomponente repräsentiert, auf die das Shuttle getroffen ist. Auf ein Signal der Steuerung hin wird der Prozess aus dieser Warteschlange wieder aktiviert.

Das Anhalten eines Shuttles ist ein bedingtes Ereignis. Vor der Ausführung ist nämlich zu prüfen, ob die Bedingungen, die zur Erzeugung des Ereignisses geführt haben, noch gegeben sind. Berechnet der Shuttleprozess, dass das Shuttle in einer bestimmten Zeit auf ein wartendes Shuttle trifft, ist nach dieser Zeitspanne zu prüfen, ob das vorausfahrende Shuttle immer noch hält. Ist diese Anforderung nicht mehr gegeben, wird der Befehl nicht ausgeführt.

IoCommand

Zur Kommunikation der Steuerung mit Objekten der Simulation wird ein *IoCommand* ausgelöst. Die Empfänger dieses Ereignisses müssen die Schnittstelle *Actor* implementieren. In der vorliegenden Modellierung erfüllen die Klassen *StopPoint*, *InnerActor* und *Station* diese Bedingung. Es ist ein Ziel, die Aktionen, die sich auf

Grund des Ereignisses ergeben, in den Klassen zu kapseln. Hierzu kann jede Klasse die Methode *startCommand* je nach Verwendung implementieren. Empfängt ein Objekt der Klasse *StopPoint* einen Aufruf dieser Methode, startet es alle Prozesse, die sich in der Warteschlange befinden und auf ihre Fortsetzung warten. Im Gegensatz hierzu führt dieses Ereignis bei einem *InnerActor*-Objekt dazu, dass eine Nachricht an die Benutzungsoberfläche versendet wird und sich der Prozess in die Warteschlange des Schedulers einreiht. Die Verzögerung entspricht der Zeit zum Einstellen der neuen Position. Eine analoge Reaktion ruft diese Nachricht bei einer Repräsentation einer Arbeitsstation hervor.

Zusammenfassung der Befehle

Die beschriebenen Befehle werden in Tabelle 4-2 noch einmal übersichtlich dargestellt. Die Sequenzdiagramme zu den jeweiligen Befehlen sind dem Anhang zu entnehmen.

Tabelle 4-2: Übersicht über die Befehle im Simulatorkern

Typ	Funktion	Empfänger	Aufrufer	Erzeuger
ChangeTrack	Wechsel der Fahrspur	Shuttle	Shuttle	AttachedTrack
StopShuttle	Anhalten	Shuttle	Shuttle	Shuttle (Vorgänger)/ StopPoint
IoCommand	Reaktion auf Steuerung	Actor	IoInterface	IoInterface
SendToIoBus	Signal an Steuerung	IoInterface	Shuttle	Sensor / InnerActor / Station
SendIdToIoBus	Lese die Identifizierung und liefere diese an die Steuerung	Shuttle IoInterface	Shuttle	IdentificationUnit
SetPart	Verändere die Anzahl der transportierten Teile	Shuttle	Station	Station

Das Befehlsmuster kann eingesetzt werden, um eine Anzahl von Operationen zu kapseln. Des Weiteren muss dem *Aufrufer* nicht bekannt sein, auf welchen Objekten die Operationen angewendet werden. Ausgehend von der Tabelle 4-2 erscheint der Einsatz dieses Musters nicht unbedingt notwendig. Zum einen sind die

ausgeführten Anweisungen noch wenig komplex, zum anderen ist dem *Aufrufer* das empfangene Objekt bekannt bzw. kann ermittelt werden. Erreicht bspw. ein Shuttle das Ende einer Fahrstrecke, kann der Shuttleprozess direkt eine Methode des Streckenobjektes aufrufen, die eine identische Anweisungsfolge des *Change-Track*-Befehls implementiert. Die Implementierung einer Operationsfolge im befehlsempfangenden Objekt erfüllt die Funktion der entsprechenden Anordnung, allerdings geht die Aufgabe der Methode nicht direkt aus ihrem Namen hervor. Verstärkt wird dieser Aspekt, wenn der Methodenname einer Schnittstellenspezifikation entsprechen muss. Durch das Entwurfsmuster wird die Struktur des Programms deutlicher. Der Name eines Befehls hebt unmittelbar seine Funktion hervor. Hierdurch wird die Wartbarkeit des Simulatorkerns erhöht. Ändern sich z. B. die Operationsfolgen auf ein Ereignis einer Identifizierungseinheit, kann der Entwickler intuitiv die Operationen in der zugehörigen Klasse *SendIdToIoBus* anpassen. Die Vorteile erhöhen sich, wenn der Befehl an mehreren Stellen benötigt wird. Weiterhin unterstützt das Entwurfsmuster die Erweiterbarkeit des Systems. Wenn Funktionen synchronisiert mit der Simulationszeit ausgeführt werden sollen, kann auf den selben Mechanismus zurückgegriffen werden. Ein solches Szenario ist denkbar, wenn der Benutzer zu einem bestimmten Zeitpunkt eine fest vorgegebene Einstellung eines variablen Streckenmoduls voraussetzen möchte. Soll der Anwender über eine interaktive Oberfläche auf den Simulationslauf Einfluss nehmen können, kann einem Menüeintrag ein Simulationsbefehl zugeordnet werden. Hierbei kann nicht mehr gewährleistet werden, dass der *Aufrufer* einen Verweis auf den *Empfänger* hat. Des Weiteren ermöglicht die Befehlsstruktur eine variable Reaktion auf Simulationsereignisse. So könnten in der Konfiguration des Simulators unterschiedliche Verhaltensweisen für die Modellelemente vorgegeben werden. Auf diese Weise wird über ein dynamisches Laden der Klassen das Verhalten des Simulationsmodells auf Ereignisse verändert, ohne den Quellcode des Programms anpassen zu müssen.

Ungeachtet der Realisierung durch ein Befehlsmuster oder durch Methodenaufrufe in einem Objekt entsteht durch den gewählten Ansatz ein Problem, das zu Verfälschungen in der Abarbeitung der Ereignisse führen kann. Die Ereignisse, welche durch die passiven Objekte ausgelöst werden, müssen im Voraus berechnet werden. Wenn zusätzlich das Auslösen der Ereignisse noch von der Aktivierung der Prozesse abhängig ist, können die Ereignisse zu falschen Zeitpunkten ausgeführt werden. Berechnet bspw. ein Shuttle, dass es in einer Entfernung von vier Metern auf einen Sensor trifft, speichert es das Ereignis und wird bei einer aktuellen Geschwindigkeit von einem Meter pro Sekunde für vier Sekunden deaktiviert. Nach der Reaktivierung startet es den zuvor erzeugten Befehl. Wurde der Zeitpunkt der Aktivierung des Shuttles durch einen anderen Prozess verändert, wird das Ereignis zu einem falschen Zeitpunkt durchgeführt. Es ist konsequent darauf zu achten, dass die Konsistenz zwischen der Aktivierung der Objekte und dem Auslösen der Ereignisse erhalten bleibt.

4.4 Anbindung der Benutzungsoberfläche

Der Simulatorkern verändert den Systemzustand abhängig von den Ereignissen, die im Simulationslauf ermittelt werden. Um die berechneten Werte darstellen bzw. auswerten zu können, müssen diese an entsprechende Objekte delegiert werden. Da der Kern in unterschiedlichen Projekten eingesetzt werden soll, erfordert diese Schnittstelle eine hohe Flexibilität. Es kann nicht im Voraus ermittelt werden, wann der Zustand eines Modellelements benötigt wird bzw. welches Objekt die Informationen erhalten soll. So kann zur Ermittlung statistischer Werte die Protokollierung zu frei wählbaren Zeitpunkten erfolgen. Währenddessen kann aber der Systemzustand auf einer grafischen Oberfläche animiert werden. Die Abbildung 4-11 zeigt die Klassenstruktur, die unter Beachtung der gestellten Forderungen das Weiterleiten der aktuellen Zustände erlaubt.

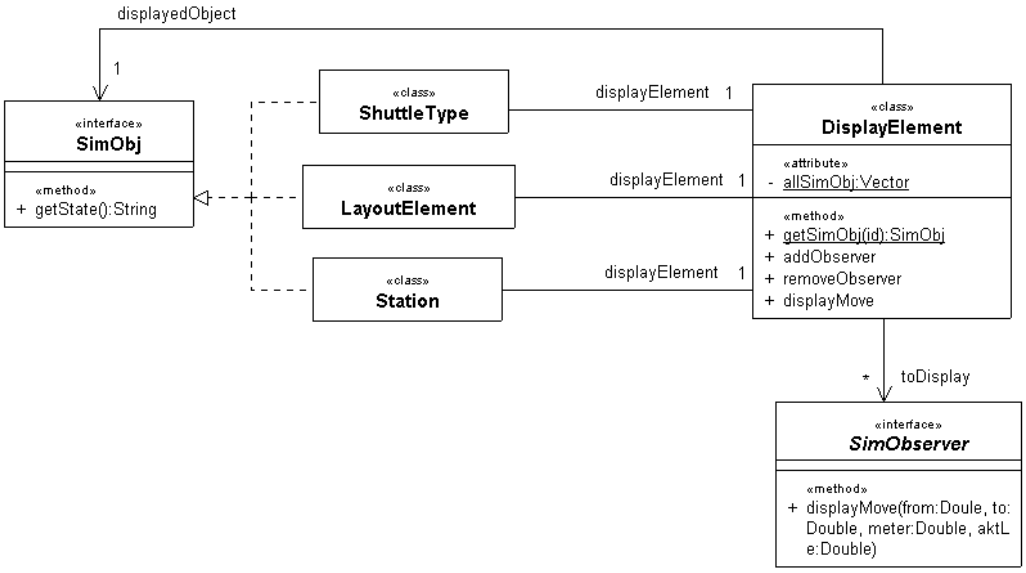

Bild 4-11: Klassendiagramm zur Anbindung externer Objekte an die des Simulatorkerns.

Das angegebene Klassendiagramm sieht vor, dass Instanzen der Klassen *ShuttleType*, *Station* und *LayoutElement* ihre Zustände nach außen sichtbar machen (vgl. Abschnitt 3.4.1). Objekte, die Informationen erhalten sollen, müssen die Schnittstelle *SimObserver* implementieren. Zwischen den sendenden und empfangenden Objekten vermittelt die Klasse *DisplayElement*. Objekte dieser Klasse erhalten die Veränderungen des Zustandes von den mit ihnen assoziierten Objekten des Simulatorkerns. Sie leiten die Information an alle *SimObserver*-Instanzen, die sich mit der Methode *addObserver* angemeldet haben. Zu Beginn der Verbindung kann der aktuelle Zustand der Modellelemente durch Aufruf der Methode *getState* ermittelt werden. Die Veränderung der Zustände wird als Text geliefert, der die in

der Anforderungsanalyse ermittelten Daten zur Beschreibung der Bewegung eines Simulationsobjektes enthält.

Es ist Aufgabe der Benutzungsoberfläche, dem Benutzer die Konfiguration der zu modellierenden Anlage zu erleichtern. Mit Hilfe der Benutzereingaben wird eine Objektstruktur erzeugt, auf der der Simulatorkern seine Berechnungen ausführt. Zur Speicherung der Benutzereingaben und damit der Konfigurationsdaten des Simulatorkerns sind unterschiedliche Ansätze denkbar. Neben der Speicherung der Daten in einer Datenbank (relational oder objektorientiert) können Dateien mit verschiedenen Satzformaten eingesetzt werden. Um auch in Zukunft anderen Alternativen gegenüber offen zu sein, wird auch zur Generation der Objektstruktur eine Schnittstelle verwendet (Bild 4-12). Zu diesem Zweck wird der Klasse *SimulationController*, die Kontrolle über einen Simulationslauf ausübt, ein Objekt, das die Schnittstelle *DataClassLoader* implementiert, übergeben. Im Gegensatz zu dem oben beschriebenen Verfahren, flexibel externe Objekte über Veränderungen des Systemzustandes zu informieren, erfordert die Anbindung einer neuen *Loader*-Klasse sehr spezielle Kenntnisse über den Aufbau des Simulatorkerns und der zu erzeugenden Objektstruktur. Z. Zt. ist zum Testen der Funktionsfähigkeit des Simulatorkerns der Klasse *MyFileLoader* implementiert, die eine Datei mit dem in Anhang B.1 beschriebenen Format einliest.

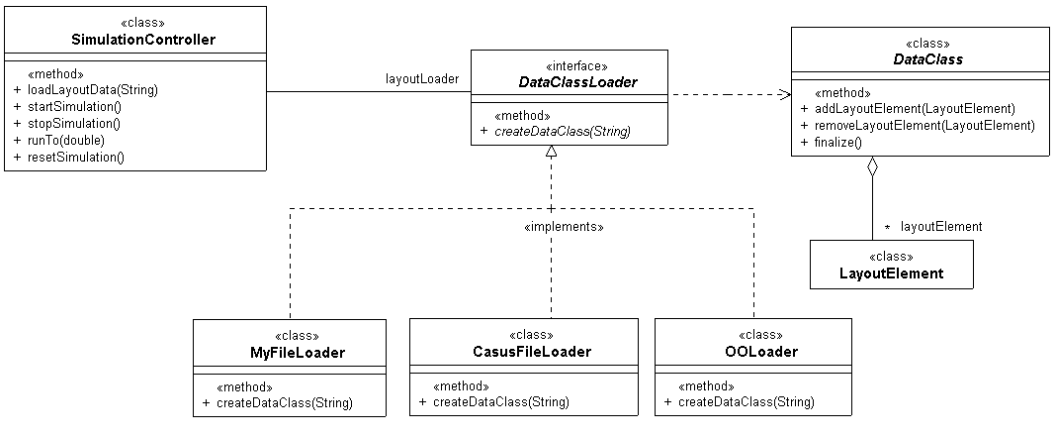

Bild 4-12: Klassendiagramm zum Erzeugen der Objektstruktur