- Systementwurf -

Software-Architektur für „Türme von Hanoi-Trainer“

Version: 1.2

Projektbezeichnung	Produkt/Projektbezeichnung	
Projektleiter		
Verantwortlich	SW-Architekt	
Erstellt am		
Zuletzt geändert	06.08.2021 10:18	
Bearbeitungszustand	X	in Bearbeitung
		vorgelegt
		fertig gestellt
Dokumentablage	C:\V-Modell-Editor\eclipse\workspace\weiter\Teilprojekte\TP14\templates\MasterTemplate.doc	

Änderungsverzeichnis

Änderung			Geänderte Kapitel	Beschreibung der Änderung	Autor	Zustand
Nr.	Datum	Version				
1		1.1	Alle	Initiale Produkterstellung		
2	5.8.2021	1.2	diverse	Änderung des Namens einer Schnittstelle: UIRepresentationBuilder → ModelRepresentationBuilder	KH	

Prüfverzeichnis

Die folgende Tabelle zeigt einen Überblick über alle Prüfungen – sowohl Eigenprüfungen wie auch Prüfungen durch eigenständige Qualitätssicherung – des vorliegenden Dokumentes.

Datum	Geprüfte Version	Anmerkungen	Prüfer	Neuer Produktzustand

Inhalt

1 Einleitung

Dieses Dokument soll ein Grundverständnis der Systemstruktur vermitteln ohne den Entwurf bis in letzte Einzelheiten darzulegen. Das Grundverständnis soll jedoch ausreichen, um sich ggf. anhand des Quellcodes in weitere Einzelheiten leicht einarbeiten zu können.

Kernthemen in diesem Dokument sind:

- Übersicht über die Zerlegung des Systems: Welche (größeren) Systemkomponenten gibt es? Wofür ist jede einzelne davon zuständig? Wie hängen diese Komponenten voneinander ab?
- Schnittstellenübersicht: Welche Schnittstellen stellt das System und jede Systemkomponente für seine/ihre Umgebung bereit?
- Systemkomponenten: Wie ist jede Systemkomponente aufgebaut?
- Designabsicherung: Zeigt für ausgewählte „architektur-relevante“ Use-Case-Szenarien, dass und wie diese mit dem gewählten Systementwurf realisierbar sind.

Der Systementwurf wird auf Grundlage der funktionalen und nicht-funktionalen Anforderungen sowie des konzeptuellen Datenmodells gewonnen, etwa indem man für ausgewählte „architektur-relevante“ Use-Case-Szenarien untersucht, welche Teile des Systems zur Realisierung in welcher Weise zusammenarbeiten müssen.

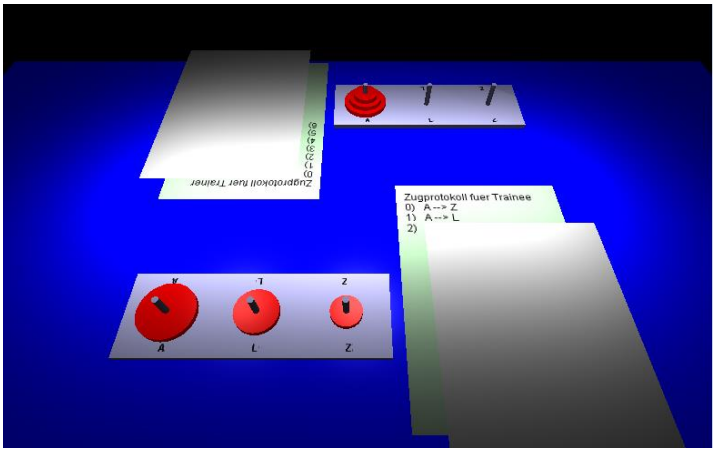
Die Gliederung dieses Dokuments orientiert sich grob am Aufbau der V-Modell-XT®[1]-Produkte „System-Architektur“ und „SW-Architektur“, ist jedoch zur Verwendung für die Veranstaltung **„Software-Projekte“** in Informatik-Curricula der **OTH-Amberg-Weiden** angepasst worden (und nicht konform zum V-Modell-XT).

Hinweis: Dieses Dokument nimmt Bezug auf das Ergebnis der Systemanalyse zum Türme-von-Hanoi-Trainer (vgl. [1]); insbesondere das dort dargelegte konzeptuelle Datenmodell und das Glossar.

2 Architekturprinzipien und Entwurfsalternativen

Das System ist unter strenger Einhaltung des **„model-view-separation“-Prinzips** realisiert, weil die Benutzungsschnittstelle (zu Demonstrationszwecken in der Lehre) ohne Änderungen am „model“ austauschbar sein soll. Zur Demonstration dieser Tatsache gibt es zwei unterschiedliche Benutzungsschnittstellen:

Grafische Benutzungsschnittstelle
Diese ist in Form einer interaktiven 3D-Szene realisiert, die den virtuellen Spieltisch mit den Spielbrettern und Zugprotokollen von Trainee und Trainer zeigt:



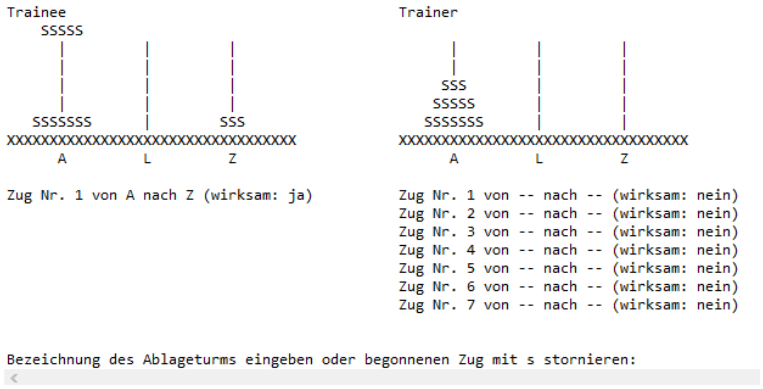
[1] V-Modell® ist eine geschützte Marke der Bundesrepublik Deutschland.

Der Benutzer kann neue Züge eingeben, indem er Entnahme- bzw. Ablageturm durch Mausclick oder durch Eingabe der Turmbezeichnung auf der Tastatur wählt. Das Zugprotokoll wird durch ein Notizblatt mit verschiebbarem Deckblatt dargestellt: im Bild oben enthält das Zugprotokoll des Trainees drei Züge, der letzte Zug ist unwirksam und deswegen durch das Deckblatt verdeckt. Der Benutzer kann im Zugprotokoll Züge unwirksam bzw. wirksam werden lassen, indem er das Deckblatt mit der Maus verschiebt oder das dafür vorgesehene Kommando über Tastatur eingibt.

Alternativ hätte die grafische Benutzungsschnittstelle auch technisch einfacher unter Verwendung von 2D-Grafik und von Standard-Bibliotheken wie Java-AWT oder Java-Swing realisiert werden können. Gründe für die aufwändigere Realisierung als 3D-Szene sind:

- Die Darstellung als 3D-Szene erscheint aus Benutzersicht „am natürlichsten“ – die Interaktionsmöglichkeiten mit der Oberfläche kann der Benutzer nahezu ohne weitere Erläuterung erkennen.
- Das Konzept eines 3D-Szenegraphen sollte für eine (inzwischen nicht mehr im Kursangebot der OTH enthaltene) Vorlesung beispielhaft an Java3D dargelegt werden; als motivierendes Beispiel, das nicht allzu komplex aber auch nicht völlig trivial ist, bot sich Türme von Hanoi an.

Kommandozeilenschnittstelle
Diese erlaubt (mit eingeschränktem Komfort) die Bedienung über die Kommandozeile. Das folgende Bild zeigt den Aufbau der Darstellung: in der linken Hälfte Spielbrett und Zugprotokoll des Trainees, in der rechten Hälfte entsprechend für den Trainer. Darunter die Eingabeaufforderung zur Eingabe des Ablageturms (das System wartet hier momentan auf den Ablageturm, der Entnahmeturm ist „A“, was an der abgehobenen Scheibe dieses Turms zu erkennen ist).



Die Kommandozeilenschnittstelle gibt es allein zur Demonstration, dass das „model“ bei Austausch der Benutzeroberfläche tatsächlich unverändert bleiben kann. Der Aufwand für Entwurf und Implementierung dieser Benutzerschnittstelle ist deswegen auf ein Minimum beschränkt und wird im Folgenden nicht weiter dargelegt.

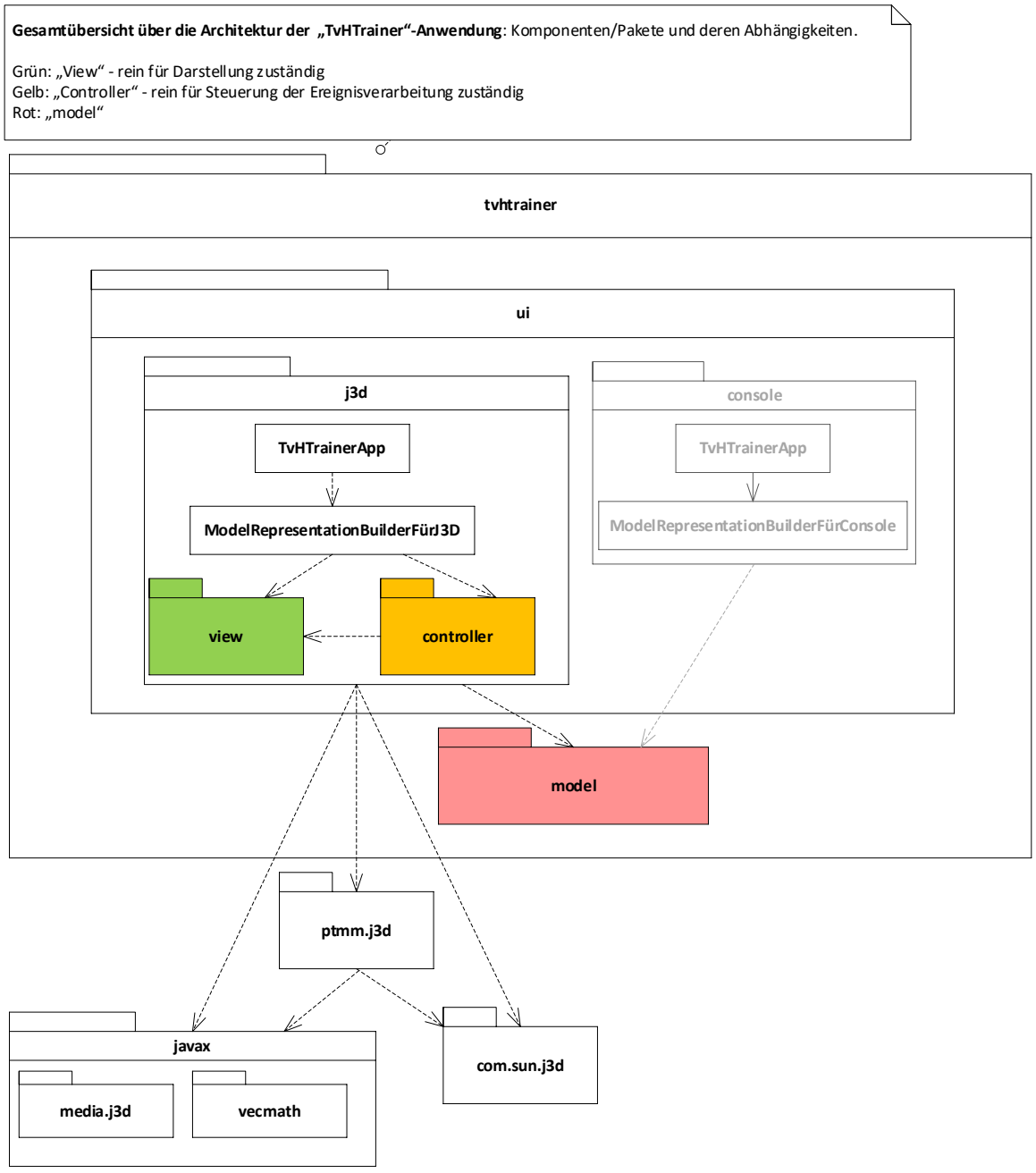
MVC-Architektur bei der grafischen Benutzungsschnittstelle

Innerhalb der Präsentationsschicht werden die Zuständigkeiten für Darstellung („view“) und Steuerung („controller“) nochmals strikt getrennt, so dass zusammen mit dem „model“ hier eine MVC-Architektur vorliegt. Mehr dazu weiter unten.

Weitere Entwurfsmuster

An der Schnittstelle zwischen model und Präsentationsschicht kommt das Entwurfsmuster „**Erbauer**“ („Builder“) der GoF zum Einsatz: Das model nutzt einen Erbauer, um eine Repräsentation von sich selbst bauen zu lassen. Das model definiert dazu eine Schnittstelle, in der festgelegt ist, was ein Erbauer können muss. Die Präsentationsschicht implementiert diese Schnittstelle, um eine Repräsentation des models bauen zu können, die für die Anzeige in der Benutzeroberfläche geeignet ist. Die Präsentationsschicht fordert das model auf, mit Hilfe des als Parameter übergebenen Erbauers eine Repräsentation zu erstellen. Während der Erstellung gibt das model dem Erbauer verschiedene Aufträge, Darstellungen für Scheiben, Türme, Protokolle, Züge etc. zu erstellen. Am Ende der Erstellung kann die Präsentationsschicht das fertige Gesamtergebnis von seinem Erbauer abholen und zur Anzeige verwenden. Mehr dazu weiter unten.

3 Übersicht über die Zerlegung des Systems



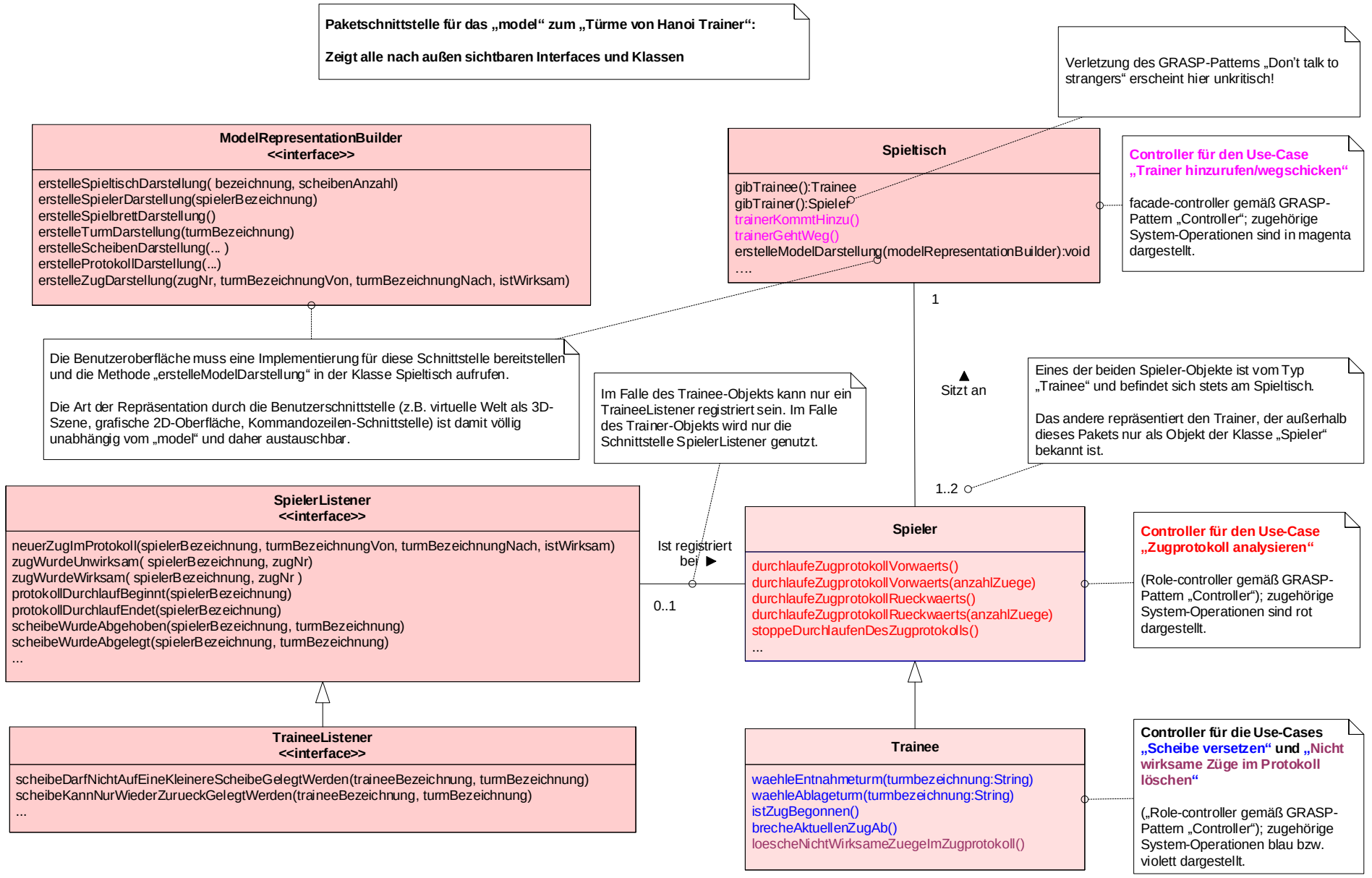
Die folgende Tabelle erläutert die Pakete mit ihren Zuständigkeiten:

Paket / Klasse	Zuständigkeit
tvhtrainer.model	Repräsentiert den Problembereich. Die Klassen dieses Pakets bilden Konzepte wie z.B. „Spielbrett“, „Turm“, „Scheibe“, „Zug“ und „Zugprotokoll“ ab und das damit verbundene Wissen. Zu diesem Wissen gehört z.B.: • Was ist ein regelkonformer Zug? • Welche Züge enthält das Protokoll und welche davon sind wirksam? • Welche Verteilung der Scheiben auf Türme ergibt sich aus den wirksamen Zügen des Protokolls? Die Präsentationsschicht soll daher bei der Eingabe neuer Züge die Entscheidung über die Regelwidrigkeit des Zugs dem model überlassen und die Entscheidungslogik nicht selbst implementieren.
tvhtrainer.ui.j3d.controller	Enthält alle Klassen zur Steuerung der Benutzerinteraktion: Ereignisse zu Maus- oder Tastatureingaben werden von diesen Klassen verarbeitet, indem sie die nötigen Reaktionen im „model“ bzw. „view“ veranlassen.
tvhtrainer.ui.j3d.view	Enthält alle Klassen zur Darstellung der 3D-Szene und zur Bewegung von Objekten in dieser Szene: eine Scheibe wird z.B. als 3D-Körper dargestellt und Scheiben werden von Turm zu Turm bewegt, während der Benutzer das Zugprotokoll analysiert. Sowohl die Scheibe selbst als auch ihre Bewegung von Turm zu Turm wird hier als etwas visuell Wahrnehmbares betrachtet und deshalb sind beide dem Paket „view“ zugeordnet.
tvhtrainer.ui.j3d.TvHTrainerApp	Anwendungsklasse für TvHTrainer als Java-3D-Anwendung - enthält „main“.
tvhtrainer.ui.j3d.ModelRepresentationBuilderFürJ3D	Die oben erwähnte Erbauerklasse. Erstellt für die Präsentationsschicht unter Regie des „models“ den Java3D-Szenegraphen. Dieser dient zur Visualisierung und zur Steuerung der Benutzerinteraktion.
tvhtrainer.ui.console	Enthält die Kommandozeilen-Schnittstelle (wird im Folgenden nicht weiter erläutert).
Externe Pakete bzw. Komponenten	
ptmm.j3d	OTH-eigene Klassen (Ersteller KH) für das bequemere Arbeiten mit Java3D-Szenegraphen.
javax und com.sun.j3d	Klassen der Java3D-Szenegraph-Bibliothek.Unter BSD-Lizenz (J3D-Utils in sun.j3d) bzw. GNU-General-Public-License (javax)

4 Schnittstellenübersicht

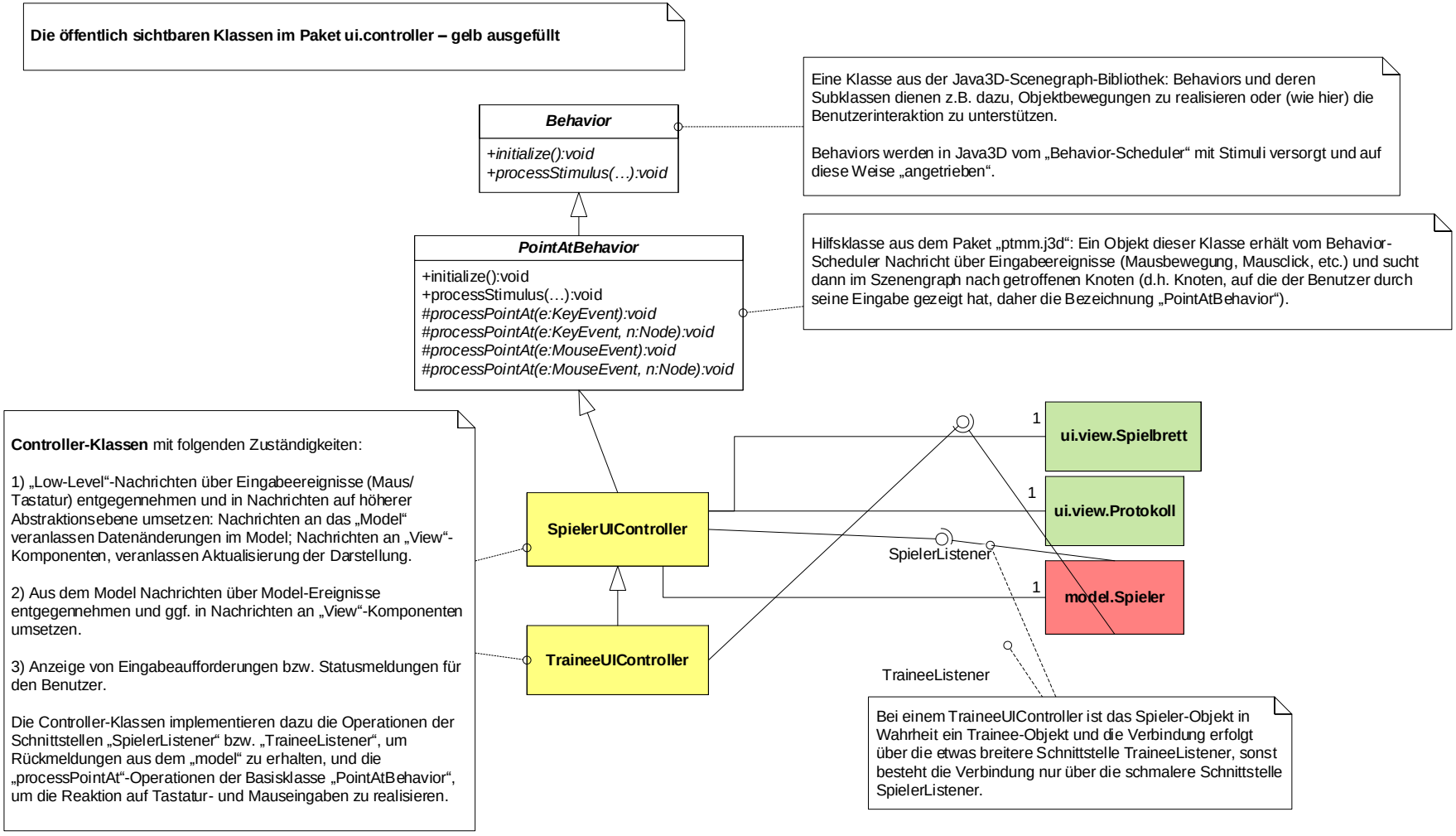
Schnittstelle des Pakets „tvhtrainer.model“

Das folgende Diagramm zeigt die öffentlich (d.h. nach außen) sichtbaren Klassen und Schnittstellen. Die Zuständigkeiten sind in UML-Notizelementen erläutert (die genannten GRASP-Muster stammen aus [2]). Nicht alle Operationen und Parameter sind dargestellt (Auslassungen sind kenntlich durch …). Die Präsentationsschicht kann sich über den Spieltisch das Trainee- und das Trainerobjekt beschaffen und dann direkt Nachrichten an diese senden. Das ist streng genommen ein Verstoß gegen das GRASP „Don’t talk to strangers“; die damit verbundenen Risiken erscheinen aber tragbar, weil die Beziehung „Trainee/Tainer sitzt an Spieltisch“ und auch die Schnittstellen der Klassen Trainee und Trainer hinreichend stabil (gegenüber künftigen Änderungen) erscheinen.


Paketschnittstelle für das „model“ zum „Türme von Hanoi Trainer“:
Zeigt alle nach außen sichtbaren Interfaces und Klassen
Verletzung des GRASP-Patterns „Don’t talk to strangers“ erscheint hier unkritisch!
ModelRepresentationBuilder
<<interface>>
erstelleSpieltischDarstellung(bezeichnung, scheibenAnzahl)
erstelleSpielerDarstellung(spielerBezeichnung)
erstelleSpielbrettDarstellung()
erstelleTurmDarstellung(turmBezeichnung)
erstelleScheibenDarstellung(...)
erstelleProtokollDarstellung(...)
erstelleZugDarstellung(zugNr, turmBezeichnungVon, turmBezeichnungNach, istWirksam)
Spieltisch
gibTrainee():Trainee
gibTrainer():Spieler
trainerKommtHinzu()
trainerGehtWeg()
erstelleModelDarstellung(modelRepresentationBuilder):void
....
Controller für den Use-Case „Trainer hinzurufen/wegschicken“
facade-controller gemäß GRASP-Pattern „Controller“; zugehörige System-Operationen sind in magenta dargestellt.
1
Sitzt an
Die Benutzeroberfläche muss eine Implementierung für diese Schnittstelle bereitstellen und die Methode „erstelleModelDarstellung“ in der Klasse Spieltisch aufrufen.
Die Art der Repräsentation durch die Benutzerschnittstelle (z.B. virtuelle Welt als 3D-Szene, grafische 2D-Oberfläche, Kommandozeilen-Schnittstelle) ist damit völlig unabhängig vom „model“ und daher austauschbar.
Im Falle des Trainee-Objekts kann nur ein TraineeListener registriert sein. Im Falle des Trainer-Objekts wird nur die Schnittstelle SpielerListener genutzt.
Eines der beiden Spieler-Objekte ist vom Typ „Trainee“ und befindet sich stets am Spieltisch.
Das andere repräsentiert den Trainer, der außerhalb dieses Pakets nur als Objekt der Klasse „Spieler“ bekannt ist.
1..2
SpielerListener
<<interface>>
neuerZugImProtokoll(spielerBezeichnung, turmBezeichnungVon, turmBezeichnungNach, istWirksam)
zugWurdeUnwirksam(spielerBezeichnung, zugNr)
zugWurdeWirksam(spielerBezeichnung, zugNr)
protokollDurchlaufBeginnt(spielerBezeichnung)
protokollDurchlaufEndet(spielerBezeichnung)
scheibeWurdeAbgehoben(spielerBezeichnung, turmBezeichnung)
scheibeWurdeAbgelegt(spielerBezeichnung, turmBezeichnung)
...
Ist registriert bei
0..1
Spieler
durchlaufeZugprotokollVorwaerts()
durchlaufeZugprotokollVorwaerts(anzahlZuege)
durchlaufeZugprotokollRueckwaerts()
durchlaufeZugprotokollRueckwaerts(anzahlZuege)
stoppeDurchlaufenDesZugprotokolls()
...
Controller für den Use-Case „Zugprotokoll analysieren“
(Role-controller gemäß GRASP-Pattern „Controller“); zugehörige System-Operationen sind rot dargestellt.
TraineeListener
<<interface>>
scheibeDarfNichtAufEineKleinereScheibeGelegtWerden(traineeBezeichnung, turmBezeichnung)
scheibeKannNurWiederZurueckGelegtWerden(traineeBezeichnung, turmBezeichnung)
...
Trainee
waehleEntnahmeturm(turmbezeichnung:String)
waehleAblageturm(turmbezeichnung:String)
istZugBegonnen()
brecheAktuellenZugAb()
loescheNichtWirksameZuegeImZugprotokoll()
Controller für die Use-Cases „Scheibe versetzen“ und „Nicht wirksame Züge im Protokoll löschen“
(„Role-controller gemäß GRASP-Pattern „Controller“); zugehörige System-Operationen blau bzw. violett dargestellt.

Schnittstelle des Pakets „tvhtrainer.ui.j3d.controller“

Das folgende Diagramm zeigt gelb ausgefüllt die öffentlich sichtbaren Klassen dieses Paktes. Die Zuständigkeiten sind in UML-Notizelementen erläutert:



Schnittstelle des Pakets „tvhtrainer.ui.j3d.view“

Paketschnittstelle für ui.j3d.view: Zeigt alle öffentlich sichtbare Interfaces und Klassen und deren öffentlich sichtbare Operationen

Die Klasse „Spielbrett“ unterstützt den Controller beim Versetzen von Scheiben und beim identifizieren des vom Benutzer angeclickten Turms.

Beim Aufbau der 3D-Szene sorgt das Spielbrett für die Erstellung seiner Bestandteile Turm- und Scheibendarstellung.

Die Klasse „Protokoll“ unterstützt den Controller beim interaktiven Ziehen des Deckblatts und beim Hinzufügen bzw. Löschen von Zügen.

Spielbrett

+Spielbrett(...)
+erzeugeScheibendarstellung(...):void
+erzeugeTurmdarstellung(...):void
+hebeScheibeAbVon(turmbezeichnung:string):void
+legeScheibeAbAuf(turmbezeichnung:String):void
+welcherTurmGehortZuDiesemKnoten(n:Node):String

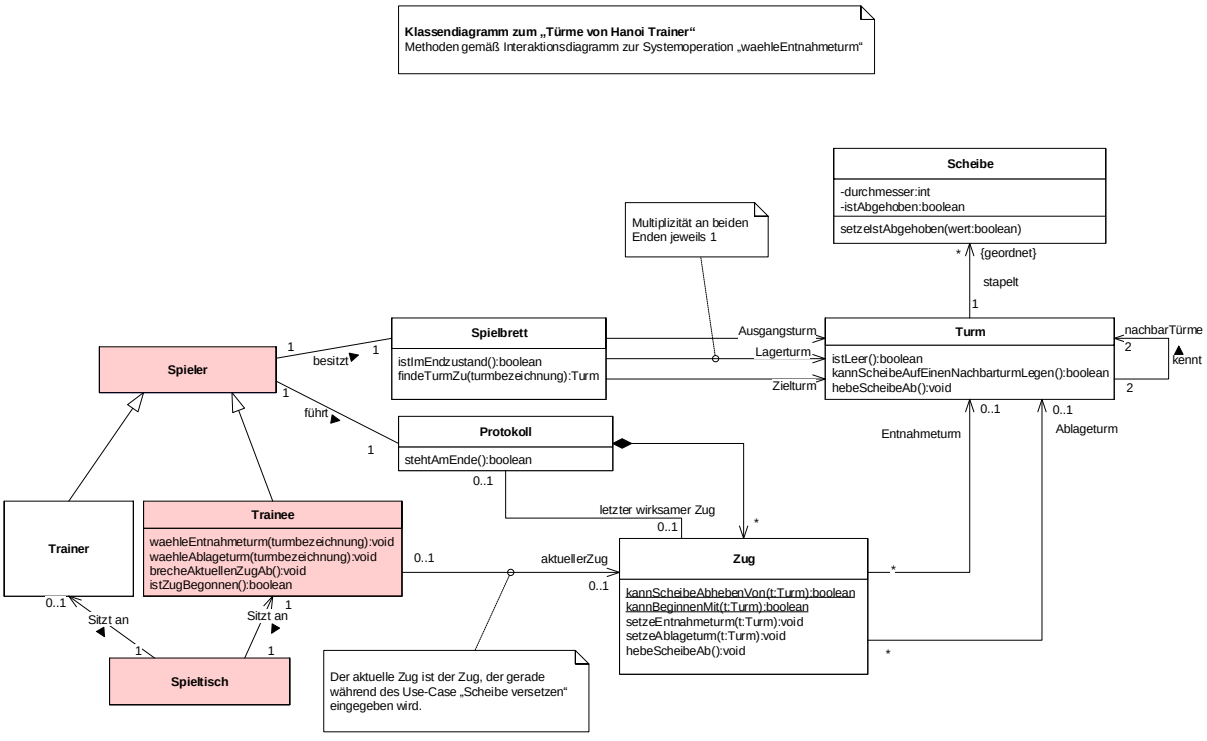
Protokoll

+Protokoll(...)
+beginneDeckblattziehen(wo:Point3d):void
+zieheDeckblatt(wohin:Point3d):void
+beendeDeckblattziehen():int
+deckeZugAuf():void
+deckeZugAb():void
+notiereZug(von:String, nach:String, istWirksam:boolean):void
+loescheNichtwirksameZuege():void

Einige öffentliche Operationen, die nur für das Detailverständnis wichtig sind, wurden hier weggelassen. Ebenso wurden an einigen Stellen die Parameter weggelassen – kenntlich durch (...)

5 Systemkomponenten

Paket tvhtrainer.model



Dargestellt sind die wichtigsten Klassen im model. Diese entsprechen den Klassen im konzeptuellen Datenmodell. Zusätzlich zum konzeptuellen Datenmodell sind Operationen ergänzt worden, die für die Interaktion in der System-Operation „waehleEntnahmeturm“ für den Use-Case „Scheibe versetzen“ gebraucht werden (vgl. auch den folgenden Abschnitt über Designabsicherung). Außerdem ist für jede Assoziation deren Navigierbarkeit festgelegt (im Interesse schwacher Kopplung alle nur in der angegebenen Richtung navigierbar). Öffentlich sichtbare Klassen sind hellrot ausgefüllt.

Die Zuständigkeiten sind:

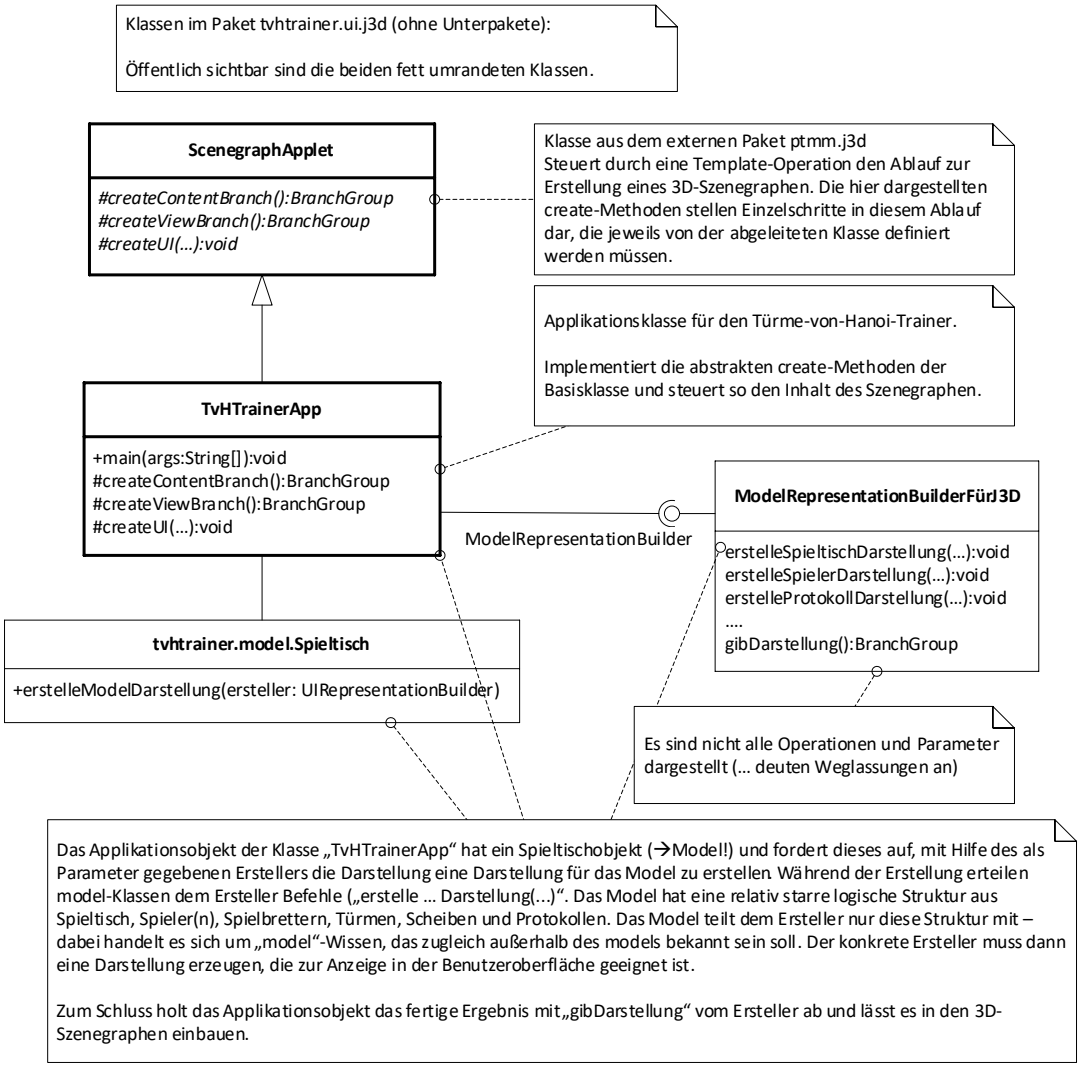
Klasse	**Zuständigkeit**
Spieler	Generalisierung von Trainee und Trainer - hat ein Spielbrett und ein Protokoll. Bietet Operationen zum Analysieren des Zugprotokolls (im Diagramm noch nicht dargestellt)
Trainee	Bietet zusätzlich zu Spieler Operationen zur Festlegung neuer Züge.
Trainer	Kann sein Zugprotokoll mit den optimalen Zügen befüllen, indem der bekannte rekursive Algorithmus ausgeführt wird.
Spieltisch	Keine weitere Erläuterung
Spielbrett	Keine weitere Erläuterung
Turm	Verwaltet einen der Größe nach geordneten Scheibenstapel und unterstützt die Prüfung auf Sinnhaftigkeit und Regelkonformität von Zügen sowie das Versetzen von Scheiben *so*, dass die Klasse „Scheibe“ für alle anderen Klassen hinter „Turm“ verborgen bleibt. Jeder Turm kennt seine beiden Nachbartürme auf dem Spielbrett.
Scheibe	Keine weitere Erläuterung
Protokoll	Verwaltet die geordnete Folge aller Züge und kennt den letzten wirksamen Zug. Bietet Operationen zum Analysieren des Zugprotokolls (im Diagramm noch nicht dargestellt)
Zug	Merkt sich Entnahme- und Ablageturm. Bietet Operationen zur Prüfung auf Sinnhaftigkeit und Regelkonformität von Zügen. Allein die Klasse „Zug“ hat das Wissen, ob ein Zug regelwidrig ist oder nicht; die Klasse Trainee soll dieses Wissen bei der Definition neuer Züge nutzen. Bietet Operationen zum Analysieren des Zugprotokolls (Versetzen der Scheibe vom Entnahme- auf den Ablageturm bzw. in umgekehrter Richtung).

Pakete tvhtrainer.ui.j3d.view und tvhtrainer.ui.j3d.controller

Keine weiteren Erläuterungen. Die wichtigsten Klassen sind bereits in der Schnittstellenübersicht erläutert. Weitere Einzelheiten in der Darlegung des Feinentwurfs (sofern erforderlich).

Paket tvhtrainer.ui.j3d:

Das folgende Diagramm zeigt die Klassen dieses Pakets (öffentlich sichtbare sind fett umrandet). Die Zuständigkeiten sind in UML-Notizelementen erläutert:



Externes Paket javax

Enthält Klassen zur Konstruktion eines 3D-Szenegraphen. Die Einzelheiten hierüber müssen aus der API-Dokumentation bezogen werden. Hier nur in Kürze das Wichtigste für ein Grundverständnis:

Ein Szenegraph ist ein azyklischer gerichteter Graph. Seine Knoten dienen der Beschreibung sichtbarer Objekte und von deren Verhalten (z.B. Bewegungen, Änderung der Erscheinung, etc.). Die Rendering-Einheit (um deren Implementierung sich der 3D-Anwendungsprogrammierer nicht kümmern muss) übernimmt den Szenegraphen und sorgt anschließend für das Rendern der Einzelbilder (Frames) – für jedes Einzelbild wird der Szenegraph von der Rendering-Einheit traversiert und die in den Knoten enthaltene Information auf den Bildschirm gebracht.

Einige sehr wichtige Klassen sind:

Klasse	**Zuständigkeit**
Shape3D	Beschreibt einen 3D-Körper. Enthält dazu Information über die Form des Körpers („geometry“) und über seine optische Erscheinung („appearance“).
BranchGroup	Alle Kind- und Kindeskindknoten werden zu einer größeren logischen Einheit zusammengefasst – das Protokoll eines Spielers wird z.B. als BranchGroup repräsentiert, die darunter befindliche Knotenhierarchie dient der Repräsentation von Deckblatt, Notizblatt und aller Textzeilen.

TransformGroup	Enthält eine 3D-Koordinaten-Transformation, die (automatisch) auf alle Kind- und Kindeskindknoten angewendet wird. Auf diese Weise erfolgen Platzierung, Ausrichtung, Größenänderung von Objekten der 3D-Szene. Auch bei animierten Bewegungen von Objekten werden TransformGroups zusammen mit speziellen Behaviors (s.u.) eingesetzt: das Behavior ändert in kurzen Zeitabständen die 3D-Transformation und nach solchen Änderungen erzeugt der Renderer automatisch(!) ein neues Einzelbild.
Behavior	Basisklasse für alle Szenegraphknoten, die „Verhalten“ repräsentieren. „Verhalten“ kann z.B. sein: animierte Bewegung oder Verformung von 3D-Objekten oder Änderung ihrer optischen Erscheinung. Behaviors können sich auch über Eingabeereignisse (Tastatur/Maus) benachrichtigen lassen und als Reaktion auf solche Ereignisse Änderungen an der 3D-Szene herbeiführen.
J3DBehavior-Scheduler	Software-Komponente, die zur Laufzeit alle Behaviors in einer Szene verwaltet und mit sog. „Stimuli“ versorgt. Beispiele für Stimuli sind Benachrichtigungen über Eingabeereignisse, Timer-Ereignisse, Eintreffen von Nachrichten, die andere Behaviors beim Scheduler „gepostet“ haben. Der 3D-Anwendungsprogrammierer muss sich um den BehaviorScheduler nicht weiter kümmern, muss aber ein paar grundlegende Dinge wissen: • Bevor das erste Einzelbild einer Szene gerendert wird, erhalten alle Behaviors die „initialize“-Nachricht. • Nachdem die 3D-Szene erstmalig gerendert wurde, schickt der Scheduler regelmäßig die „processStimulus“-Nachricht an alle Behavoirs, deren Aufweck-Bedingung erfüllt ist. • Bei der Verarbeitung einer processStimulus-Nachricht kann ein Behavior seine Aufweckbedingung neu festlegen und dem Scheduler mitteilen. • Ein Behavior kann beim Scheduler Nachrichten posten; alle Behaviors, die auf derartige Posts hin aufgeweckt werden wollen, erhalten dann vom Scheduler eine „processStimulus“-Nachricht. Behaviors sollen nicht direkt untereinander kommunizieren, sondern nur über den Scheduler durch „Posts“. • Der Scheduler muss alle Behaviors einer Szene gleichmäßig und hinreichend schnell mit Stimuli versorgen. Der Scheduler läuft in einem eigenen Thread, der die „processStimulus“-Methode ausführt. Diese Methode darf keine länger dauernden Aktivitäten oder gar Wartezustände hervorrufen, die den Scheduler verzögern oder gar blockieren (der Scheduler kann sonst seine Aufgabe nicht mehr erfüllen).

6 Designabsicherung

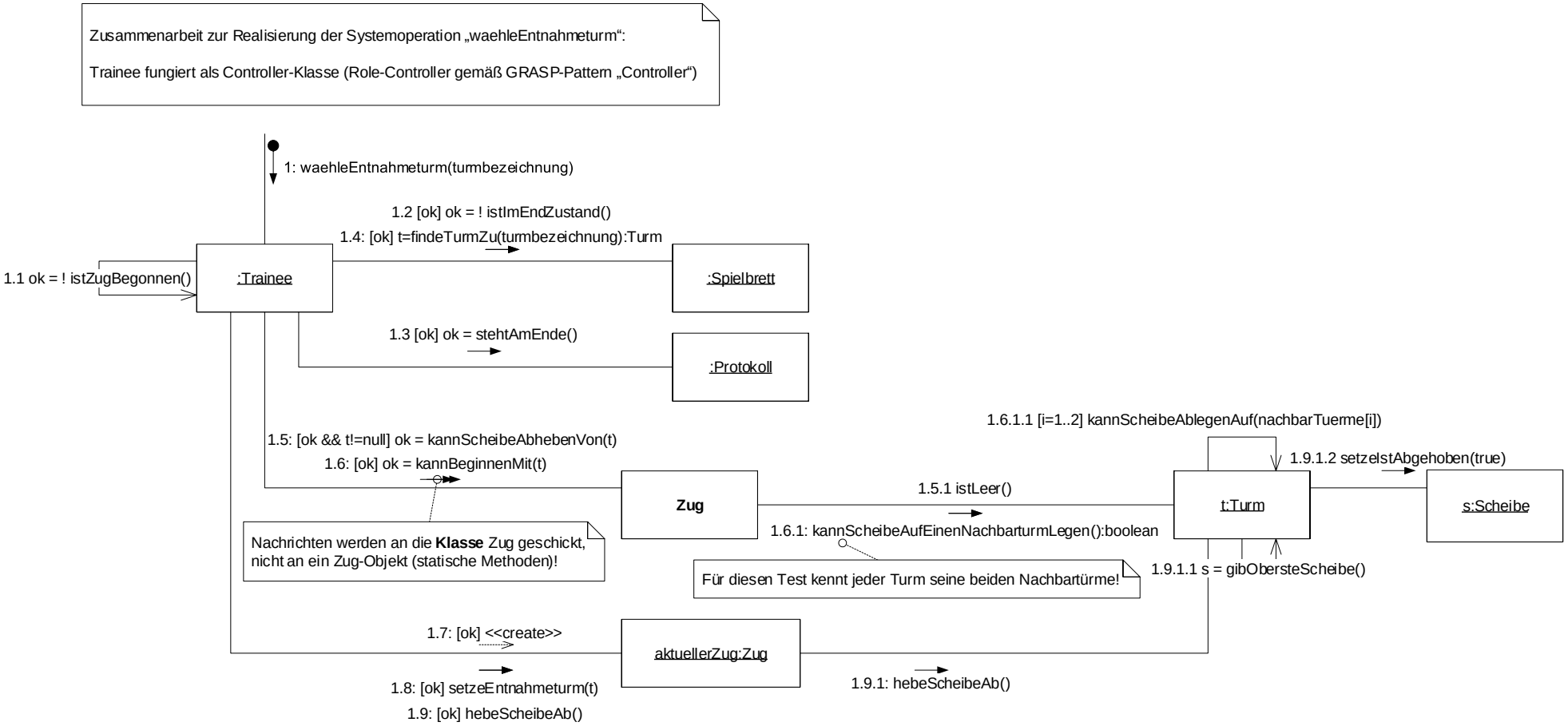
Aus dem Use-Case „Scheibe versetzen“ ergeben sich folgende System-Operationen:

waehleEntnahmeturm(turmBezeichnung)

waehleAblageturm(turmBezeichnung)

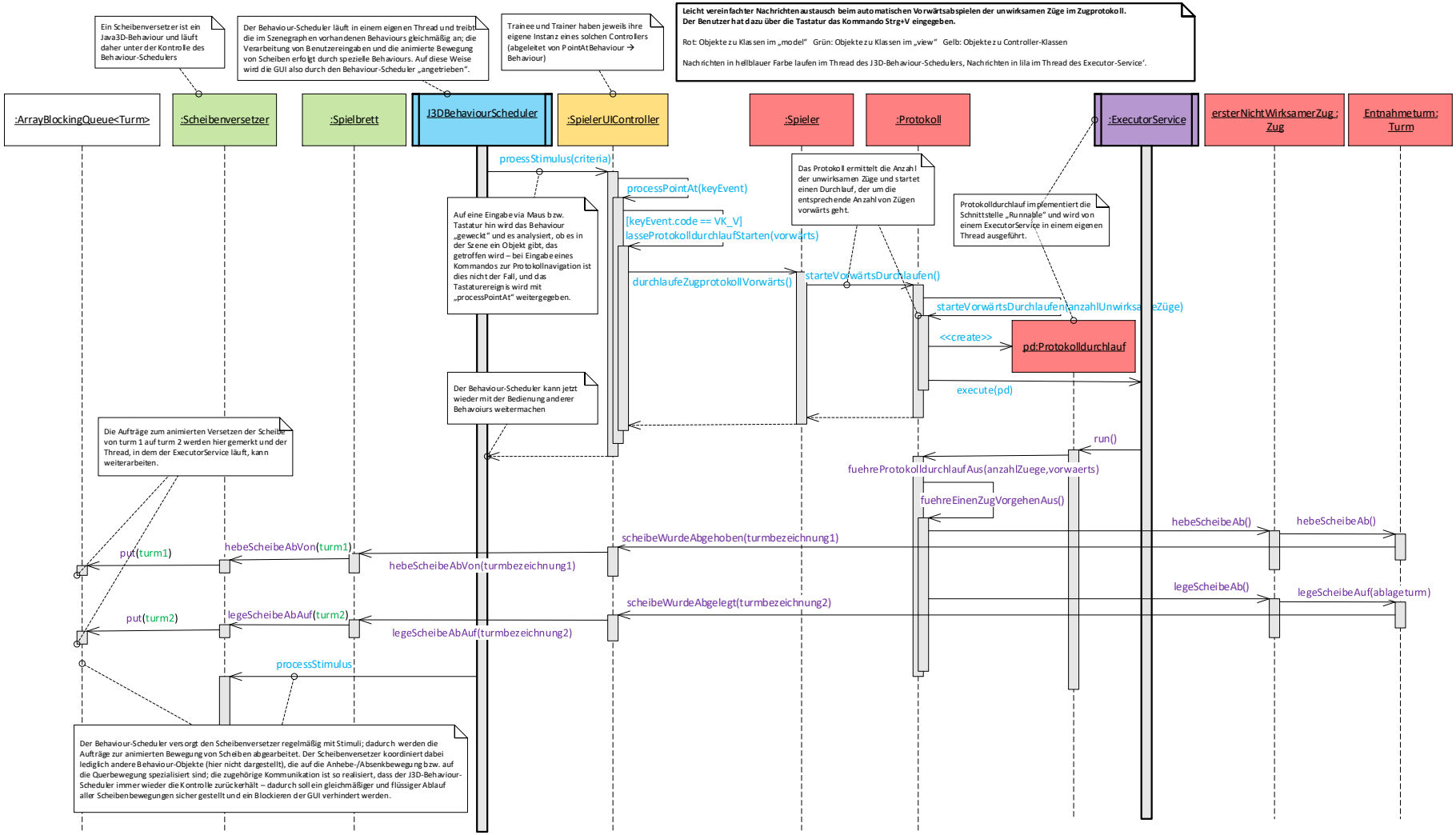
loescheNichtWirksameZuege()

Für die System-Operation waehleEntnahmeturm zeigt das folgende Diagramm die Interaktion (dargestellt sind nur die nötigen Überprüfungen und der Kommunikationsablauf, solange noch alles in Ordnung („OK“) ist. Ergibt eine Überprüfung, dass die Wahl des angegebenen Turms nicht möglich bzw. zulässig ist, so erfolgt eine Rückmeldung an die Präsentationsschicht über die Ablehnung der Eingabe und den Grund dafür; derartige Rückmeldungen sind hier nicht dargestellt). Die genannten GRASP-Muster stammen aus [2]:



Zur weiteren Absicherung scheinen Kommunikationsdiagramme zu waehleAblageturm und zu ausgewählten Systemoperationen für den Use-Case „Zugprotokoll analysieren“ nennenswerten Erkenntnisgewinn zu versprechen. Diese Diagramme wurden hier weggelassen.

Das folgende Sequenzdiagramm zeigt leicht vereinfacht für den Use-Case „Zugprotokoll analysieren“ und die System-Operation „durchlaufeZugprotokollVorwärts“ (mit animierter Scheibenbewegung im view) die paketübergreifende Kommunikation zwischen Controller, Model und View. Das Diagramm wurde erst im Laufe der Implementierung erstellt; da es aber die Stimmigkeit der Software-Architektur untermauern hilft und außerdem das Verständnis derselben wesentlich fördert, wurde es hier mit aufgenommen:



Erläuterung zum vorigen Sequenzdiagramm und zum Multi-Threading:

Im vorigen Sequenzdiagramm wird zu jedem Zug, der beim Vorwärtsdurchlaufen des Protokolls wieder wirksam wird, eine animierte Scheibenbewegung erzeugt. Eine derartige Bewegung ist eine länger andauernde Aktivität, die den Thread des BehaviorSchedulers in unzulässiger Weise verzögern würde. Deshalb wird im Model der Auftrag zum Vorwärtsdurchlaufen des Protokolls als Objekt der Klasse „Protokolldurchlauf“ in die Warteschlange eines ExecutorService-Objekts eingestellt. Der Thread des Behavior-Schedulers ist nach dem Einstellen des Protokolldurchlaufs in die Warteschlange sofort wieder frei für andere Aufgaben. Das Abarbeiten des Protokolldurchlaufs erfolgt durch einen separaten Thread, in dem der ExecutorService läuft. Dieser kann nach Bedarf verzögert werden, ohne dass dadurch der BehaviorScheduler behindert wird. Für jeden Spieler (Trainee und Trainer) gibt es einen eigenen ExecutorService, so dass gleichzeitige Protokolldurchläufe bei Trainee und Trainer möglich sind. Weil beide ExecutorServices auf separaten Zugprotokollen operieren, können durch ihre beiden Threads keine nebenläufigen Zugriffe entstehen. Nebenläufige Zugriffe auf ein Protokoll können nur entstehen, wenn ein Protokolldurchlauf im Gang ist, und dann nur durch die beiden Threads, in dem ExecutorService bzw. Behavior-Scheduler laufen.

7 Abkürzungsverzeichnis

Abkürzung	Erklärung
GoF	“gang of four” – bezeichnet die vier Autoren des bekannten Lehruchs über Entwurfsmuster (“Design Patterns”)
GRASP	“general responsibility assignment pattern” (vgl. das Buch [2] von Larman)

8 Literaturverzeichnis

[1] Dokument „Systemanalyse(TvHTrainer).docx“
[2] Larman Craig, Applying UML And Patterns. An Introduction to Object-Oriented Analysis And Design, Prentice Hall, 2nd ed., 2002

9 Abbildungsverzeichnis