Entwurf und Implementierung eines Shader und Special Effects Management Systems für die Open-Source 3D Grafik-Engine Horde3D

Bachelorarbeit im Studiengang Informatik und Informationswirtschaft

von

Axel Habermaier

31. Juli 2009

Erstgutachter: Prof. Dr. Elisabeth André
Zweitgutachter: Prof. Dr. Bernhard Bauer

Universität Augsburg

Universität Augsburg
Fakultät für Angewandte Informatik
Lehrstuhl für Multimedia-Konzepte und Anwendungen

Ich versichere, dass die Bachelorarbeit von mir selbstständig verfasst wurde und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Zitate habe ich klar gekennzeichnet.

Augsburg, den 31. Juli 2009

Inhaltsverzeichnis

1. Einleitung

Die Entwicklungskosten für Videospiele nehmen seit vielen Jahren kontinuierlich zu. Heute kostet ein modernes Spiel für Sonys PlayStation 3 im Durchschnitt 15 Millionen Dollar [1]. Aber auch das technisch weniger aufwendige Braid[1] für Microsofts Xbox Live Arcade, das mit Ausnahme des Artworks und des Soundtracks allein von Jonathan Blow entwickelt wurde, kostete in seiner Entwicklung bereits 200.000 Dollar [2].

Aufgrund des Kostendrucks wird bei der Entwicklung eines Spiels oftmals auf bereits vorhandene Subsysteme zurückgegriffen, wie beispielsweise die 3D Grafik-Engines Source[2], id tech 4[3], Unreal Engine 3[4] und Gamebryo[5]; die Physik-Engines Havok[6] und PhysX[7]; oder auch SpeedTree[8], eine Library zur Erstellung animierter 3D-Bäume. Da die Erstellung von Spielen auch für Hobby- und Open Source-Entwickler immer aufwendiger wird, unterstützen auch im Open Source-Bereich eine Vielzahl an Bibliotheken die Entwicklung; unter anderem die 3D Grafik-Engines Irrlicht[9], OGRE[10] und Horde3D[11].

Nicolas Schulz veröffentlichte am 25. September 2006 die erste Version von Horde3D. Unterstützt wird die Entwicklung durch den Lehrstuhl für Multimedia-Konzepte und Anwendungen der Universität Augsburg. Der Lehrstuhl verwendete die Engine bereits als Grundlage für einige Forschungsprojekte wie etwa das Facial Animation System Alfred[12], oder auch für das Spiel SheepMeUp[13], das im Rahmen des Multimedia Praktikums im Sommersemester 2008 von Studenten der Universität Augsburg entwickelt wurde.

Horde3D grenzt sich gegenüber den anderen kommerziellen und frei verfügbaren Engines durch ein besonderes Design-Ziel ab:

> „One of the most important design goals of Horde3D is to keep things simple and avoid complexity where possible without sacrificing flexibility or productivity. [...] Much of the power and flexibility of the engine comes from its shader driven architecture and customizable pipeline that makes it possible to apply nearly all modern rendering and post processing techniques.“ [3]

Den Anwendungsprogrammierern bietet Horde3D eine API, die einfach zu verwenden ist und dennoch größtmögliche Flexibilität gewährleistet. Jedoch beinhaltet die Engine keine

[1] http://braid-game.com
[2] http://source.valvesoftware.com
[3] http://www.idsoftware.com/business/idtech4
[4] http://www.unrealtechnology.com
[5] http://www.emergent.net
[6] http://www.havok.com
[7] http://www.nvidia.com/object/nvidia_physx.html
[8] http://www.speedtree.com
[9] http://irrlicht.sourceforge.net
[10] http://www.ogre3d.org
[11] http://www.horde3d.org
[12] http://mm-werkstatt.informatik.uni-augsburg.de/alfred-virtual-character.html
[13] http://mm-werkstatt.informatik.uni-augsburg.de/SheepMeUp.html

Tools, um zum Beispiel die angesprochenen *Post Processing* Techniken zu entwickeln und mit dem *Look and Feel* der Anwendung abzustimmen. Nur für das Zusammenstellen von Szenen gibt es Volker Wiendls Horde3D Scene Editor[14].

Bei der Entwicklung von SheepMeUp sind Mängel in der Toolunterstützung von Horde3D deutlich geworden: Das Spiel verwendet einige Shader- und Partikel-basierte Special Effects zur Darstellung von Kraftfeldern, Schockwellen und Zaubersprüchen. Dank der flexiblen Rendering-Pipeline der Engine konnten diese Effekte leicht eingebunden werden. Schwierigkeiten ergaben sich erst bei der Feinabstimmung, da die Rendering-Pipeline, Materials, Shader- und Partikeleffekte nur durch Bearbeiten der zugrundeliegenden XML-Dateien geändert werden konnten. Die Wichtigkeit der Effekte als Teil der Ästhetik des Spiels sollte aber nicht unterschätzt werden; die Ästhetik ist ein Grundstein des *Elemental Tetrad* [4, S. 41ff] und somit eines der vier Basis-Elemente eines Spiels. Die Ästhetik beeinflusst die drei anderen Basis-Elemente – Spielemechanik, Story und verwendete Technologie – und kann ausschlaggebend sein, ob ein Spiel als gut empfunden wird oder nicht. Umgekehrt sollte die verwendete Technologie das Finetuning der Effekte unterstützen und vereinfachen, um die Ästhetik des Spiels möglichst einfach und schnell – und damit kostengünstig – perfektionieren zu können.

Aufbauend auf den gewonnen Erfahrungen bei der Entwicklung von SheepMeUp wird im Rahmen dieser Bachelorarbeit ein Shader und Special Effects Management Tool, das Horde3D Development Environment, entworfen und implementiert. Das Tool soll es erleichtern, die Pipeline-Konfiguration, Materials, sowie Shader- und Partikeleffekte zu optimieren, anzupassen und abzustimmen. Außerdem soll das Tool mit jeder Horde3D-Anwendung zusammenarbeiten. Es soll jedoch nicht möglich sein, Anwendungs- oder Shader Code zu debuggen. Dafür gibt es bereits eine Reihe ausgereifter Standardentwicklungswerkzeuge.

Diese Arbeit beschreibt den Entwurf und die Implementierung der Anwendung in den drei Phasen der Softwareentwicklung: zunächst die Anforderungsanalyse, dann das Design und die Struktur des Systems und schließlich die Implementierung in C++ und C#. Es werden die jeweils getroffenen Entscheidungen und ausgewählte Teile der erstellten Artefakte jeder Phase erläutert. Die einzelnen Entwicklungsphasen wurden jedoch, angelehnt an den *Unified Process*, in mehreren Iterationen durchlaufen. Sollte eine wichtige Entscheidung erst in einer späteren Iteration getroffen worden sein, so wird dies an der entsprechenden Stelle erwähnt.

Abschließend wurde das Tool für die Entwicklung eines neuen Effekts für ein Raumschiff-Spiel eingesetzt und von einigen Entwicklern von Horde3D und SheepMeUp getestet und bewertet. Davon ausgehend werden Erweiterungen und Verbesserungsmöglichkeiten für das Horde3D Development Environment vorgeschlagen.

[14] `http://mm-werkstatt.informatik.uni-augsburg.de/project_details.php?id=45`

2. Phase I: Analyse

Bevor mit der eigentlichen Entwicklung des Systems begonnen wurde, mussten zunächst die Anforderungen an das Tool analysiert werden. Die Anforderungen ergaben sich aus der Betrachtung des Aufbaus und der Funktionsweise von Horde3D und beziehen die gewonnenen Erfahrungen bei der Entwicklung von SheepMeUp mit ein. Außerdem mussten die Ziele, die bei der Entwicklung des Horde3D Development Environments verfolgt wurden, von bereits vorhandener Software abgegrenzt werden.

In der Analyse-Phase wurden ein *Use Case* Modell und ein Konzeptmodell erstellt, welche die Grundlage für die Design-Phase bildeten.

2.1. Überblick über Horde3D

Zu Projektbeginn wurde deutlich, dass die Struktur des Horde3D Development Environments maßgeblich vom Aufbau und der Funktionsweise von Horde3D bestimmt wird. Bevor die genauen System-Anforderungen untersucht wurden, wurde ein konzeptuelles Modell von Horde3D entwickelt. Dabei wurde die interne Repräsentation der Daten – also die Klassen und Funktionen, die Horde3D intern verwendet – nicht betrachtet. Wichtig ist nur der von außen festzustellende Aufbau, denn nur davon wird die Struktur des Tools beeinflusst. Da die Dokumentation [3] allerdings keine Klassendiagramme enthält, musste die Klassenstruktur aus der Beschreibung der API [3, „Engine API Reference“], der Datenformate [3, „Data Format Reference“] und der Rendering-Pipeline [3, „Rendering Pipeline Documentation“] ermittelt werden.

2.1.1. Aufbau des Szenengraphs

Ein Szenengraph repräsentiert die logische oder räumliche Zusammensetzung der dargestellten Szene [3, „Basic Concepts, Scene Graph“]. Der Horde3D Szenengraph ist als Baum organisiert, dessen Wurzel als *Root Node* bezeichnet wird. Jeder Knoten, *Scene Node* genannt, kann beliebig viele Kinder haben und besitzt Transformationswerte für Verschiebung, Rotation und Skalierung, die jeweils relativ zum Vaterknoten sind. Knoten werden über ihren *Node Handle* eindeutig identifiziert.

Der Szenengraph wird aus folgenden Knotentypen aufgebaut:

- **Group**: Ein *Group Node* hat keine Repräsentation in der Szene, sondern fasst beliebig viele untergeordnete Knoten zusammen. Damit können beispielsweise alle untergeordneten Knoten auf einmal in der Szene verschoben, rotiert und ein- oder ausgeblendet werden. *Root Node* ist von diesem Typ.
- **Camera**: In einer Szene können beliebig viele virtuelle Kameras vorhanden sein, die zum Zeichnen der Szene verwendet werden können. Insbesondere ist es möglich, die Kamera einem animierten *Joint Node* unterzuordnen, wobei die Kamera dann dem Bewegungsablauf der Animation folgt.

- **Light**: Ein *Light Node* repräsentiert eine Lichtquelle in der Szene. Derzeit werden von Horde3D nur *Spotlights* unterstützt. Ein Lichtknoten kann durch verschiedene Parameter – wie Lichtfarbe, Reichweite und das An- oder Abschalten des Schattenwurfs – an die Szene angepasst werden.

- **Model**: Ein *Model Node* repräsentiert ein 3D-Modell, welches aus einer Modellierungssoftware exportiert wurde. Es ist eine Menge von *Mesh Node*s und *Joint Node*s, welche das Aussehen und die Animation des Modells definieren.

- **Mesh**: Ein *Mesh Node* enthält verschiedene Polygone, die zum Zeichnen eines 3D-Modells verwendet werden. Alle Polygone werden dabei mit dem gleichen Material gezeichnet.

- **Joint**: Eine Hierarchie von *Joint Node*s repräsentiert ein animierbares Skelett.

- **Emitter**: Ein *Emitter Node* ist der Ursprungsort von Partikeln und kann verschiedene Eigenschaften der erzeugten Partikel beeinflussen.

- **Overlay**: Ein *Overlay* ist eigentlich kein Teil des Szenengraphs. Es repräsentiert eine zweidimensionale Grafik, die über die dargestellte Szene gezeichnet wird. Damit kann zum Beispiel ein *Heads Up Display* realisiert werden. Auch das Zeichnen von Text wird von Horde3D durch Verwendung von *Overlay*s umgesetzt.

2.1.2. Ressourcen-Verwaltung

Ressourcen sind Daten, die zum Zeichnen der Szene benötigt werden. Dabei können mehrere *Scene Node*s die gleichen Ressourcen verwenden [3, „Basic Concepts, Resource Management“]. Der Horde3D Ressourcen Manager lädt Ressourcen bei Bedarf und vergibt eindeutige *Res Handles* zur Identifikation. Ressourcen können jederzeit neu geladen werden.

Horde3D kennt folgende Arten von Ressourcen:

- **Scene Graph**: Eine *Scene Graph Resource* enthält einen Szenengraph, der zum aktuellen Szenengraph als Ast hinzugefügt werden kann.

- **Geometry**: Eine *Geometry Resource* enthält Polygon-Daten für *Mesh Nodes*. Neben den eigentlichen Eckpunkt-Koordinaten können auch weitere Daten – wie Normalen, Tangenten, Texturkoordinaten, etc. – enthalten sein.

- **Animation**: Eine *Animation Resource* stellt Animationsdaten für *Mesh* und *Joint Nodes* bereit.

- **Pipeline**: Eine *Pipeline Resource* ist ein XML-Dokument, das die erforderlichen Schritte zum Zeichnen der Szene beschreibt. Dafür können zunächst *Render Targets* angelegt und die Engine-Konfiguration angepasst werden. Danach werden die einzelnen Schritte genau beschrieben: Zeichenreihenfolge der Material-Klassen, Lichtberechnungen, Verwenden von *Render Targets* als Quelle und Ziel für Zeichenoperation, usw. Die Pipeline ist sehr flexibel und unterstützt sowohl *Forward* als auch *Deferred Shading*.

- **Material**: Eine *Material Resource* definiert das Aussehen einer Oberfläche. Sie legt fest, welche Texturen und welcher Shader zum Zeichnen verwendet werden.

- **Shader**: Eine *Shader Resource* legt einen Kontext für die Ausführung der Vertex und Fragment Shader auf der Grafikkarte fest. So kann ein Shader zum Beispiel einen Kontext für den *Ambient Pass* und einen für den *Deferred Lighting Pass* enthalten. Die Kontexte unterscheiden sich beim Code für Vertex und Fragment Shader und bei den Rendering-Optionen wie *Depth Writes*, *Alpha Writes*, etc.

- **Code**: Eine *Code Resource* enthält Shader Code, der von mehreren Shadern verwendet werden kann.

- **Texture**: Eine *Texture Resource* ist eine 2D-Textur oder *Cube Map*, die beim Zeichnen von Oberflächen verwendet wird.

- **Particle Effect**: Eine *Particle Effect Resource* legt die Größe, Farbe, Geschwindigkeit und Lebensdauer von Partikeln fest.

2.1.3. Eigenschaften der Horde3D-API

Anwendungen können Horde3D durch Einfügen einer *Header*-Datei und Linken gegen eine *Dynamic Link Library* (DLL) einbinden. Obwohl die Engine in objekt-orientiertem C++ entwickelt wurde, orientiert sich die öffentliche Schnittstelle am Design der C-basierten Win32-API. Die Beta 3 von Version 1.0.0 hat gerade einmal 78 öffentliche Funktionen. Dies erleichtert das Erlernen der API und das Erstellen von *Bindings* für verschiedenen Programmiersprachen. Momentan werden C, C++ und alle .NET-Sprachen offiziell unterstützt, weitere *Bindings* werden derzeit von der Community entwickelt[1]. Gegenüber einer objekt-orientierten Schnittstelle, wie sie z.B. OGRE bietet, wird für manche Aufgaben jedoch mehr Code benötigt. Beispielsweise sind drei Funktionsaufrufe nötig, um die Farbe eines *Light Nodes* zu setzen; die Position, Rotation und Skalierung eines *Nodes* können nicht einzeln geändert werden und es gibt keine Vektor-Datentypen:

```
// Horde3D::setNodeParamf muss dreimal aufgerufen werden,
// um die Lichtfarbe auf Rot zu setzen.
Horde3D::setNodeParamf(light, LightNodeParams::Col_R, 1.0f);
Horde3D::setNodeParamf(light, LightNodeParams::Col_G, 0.0f);
Horde3D::setNodeParamf(light, LightNodeParams::Col_B, 0.0f);

// Diese Funktion ändert die Position, Rotation und Skalierung von
// 'node'. Dafür müssen immer alle Werte skalar übergeben werden.
Horde3D::setNodeTransform(node, 0, 20, 50, -30, 0, 0, 1, 1, 1);
```

Die Kern-API wurde schlank gehalten und ist ausreichend, um alle Features der Engine zu verwenden. Es gibt jedoch zusätzlich die Horde3D *Utility Library*, die verschiedene spezifische Funktionen zur Produktivitätssteigerung bereitstellt. Mit dieser Bibliothek können zum Beispiel Ressourcen von der Festplatte geladen, Frame Statistiken angezeigt, der OpenGL-Kontext initialisiert oder Objekte der dargestellten Szene selektiert werden. Der Stil der Horde3DUtils-API entspricht dem der Kern-API [3, „Utility Library API Reference“].

Horde3D bietet außerdem einen Mechanismus an, um die Features der Engine durch Plugins zu erweitern. Diese verwenden nicht die öffentliche API, sondern haben Zugriff auf die

[1] Momentan entwickelt die Community zum Beispiel *Bindings* für die Sprachen Python, D und Lua (`http://www.horde3d.org/wiki/index.php5?title=Language_Bindings`). Sogar für die funktionale Sprache Haskell gibt es *Bindings* (`http://www.horde3d.org/forums/viewtopic.php?f=1&t=550`).

interne Klassenstruktur von Horde3D. Um das zu ermöglichen, werden die Plugins zur Kompilierungszeit statisch in die Horde3D DLL hineinkompiliert. Die öffentlichen Schnittstellen der Plugins erweitern dann die Kern-API und können von allen Horde3D-Anwendungen verwendet werden, die gegen die erweiterte DLL gelinkt werden [3, „Basic Concepts“].

2.1.4. Fehlerbehandlung und Debug-Informationen

Da C keine Ausnahmen unterstützt, zeigen Horde3D-Funktionen Fehler durch spezielle Fehlerwerte auf; so gibt beispielsweise die Funktion `int getNodeType(NodeHandle)` im Fehlerfall den Wert `ResourceTypes::Undefined` zurück. Zusätzlich verwaltet Horde3D intern eine Liste aller aufgetretenen Fehler. Auch Ereignisse und Debug-Informationen werden protokolliert, wie zum Beispiel das Hinzufügen eines neuen *Scene Nodes*, das Laden einer Ressource oder Probleme beim Kompilieren des Shader Codes. Die protokollierten Meldungen können aus der Engine ausgelesen werden, beziehungsweise über eine Funktion der *Utility Library* direkt in eine Datei im HTML-Format auf die Festplatte geschrieben werden. Dies sind hilfreiche Informationen beim Entwickeln und Debuggen einer Horde3D-Anwendung.

2.1.5. Shader und Rendering-Pipeline

Horde3D baut auf OpenGL 2.0 auf, das eine hardware-unabhängige Schnittstelle für hardwarebeschleunigte Rasterisierung von 3D-Grafik auf der Grafikkarte (GPU) bietet. Abbildung 2.1 gibt einen Überblick über die Pipeline-Architektur moderner GPUs. `Vertex Data` sind die untransformierten Daten der Eckpunkte der Geometrie der aktuellen Szene. `Primitive Data` verbindet diese Vertex-Daten zu einer Menge von Punkten, Linien, Dreiecken oder Polygonen. Die `Vertex Processing`-Stufe verwendet die Szenen- und Projektionsmatrizen zum Transformieren der Eckpunkte und berechnet gegebenenfalls weitere Daten pro Vertex, wie etwa Texturkoordinaten. In der `Geometry Processing`-Stufe findet die eigentliche Rasterisierung statt. Beim `Fragment Processing` werden die Farbwerte der Fragmente[2] berechnet und in der `Fragment Rendering`-Stufe schließlich die Tiefen-, Alpha- und *Stencil*-Tests durchgeführt. Sind die Tests für ein Fragment erfolgreich, so wird es in die Ausgabe-Textur geschrieben oder hineingemischt [5].

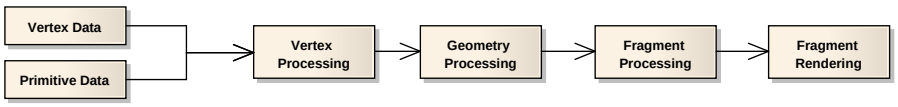

Abbildung 2.1.: Die Grafik-Pipeline von Direct3D 9 und OpenGL 2.0

Bei modernen Grafikkarten können Fragmente nicht nur in den *Backbuffer* geschrieben werden, sondern auch in *Render Targets* (RTs). *Render Targets* können im weiteren Verlauf wiederum als Eingabe für weitere Zeichenoperationen dienen. Dieses Vorgehen ist für eine Vielzahl moderner Effekte relevant; beispielsweise gibt es Effekte, die die zuvor berechneten

[2] Ein Pixel ist ein Bildpunkt auf dem Bildschirm. Ein Fragment hingegen ist ein Pixel, der eventuell nicht sichtbar ist. Im Allgemeinen werden pro Bild mehr Fragmente berechnet, als Pixel überhaupt sichtbar sein können, weil manche Fragmente später unter gewissen Umständen wieder überschrieben werden.

Tiefeninformationen der Szene kennen müssen. Es ist auch möglich, dass eine Zeichenoperation gleichzeitig verschiedene Werte in verschiedene RTs schreibt, also *Multiple Render Target*s (MRTs) verwendet.

Seit DirectX 8 und den `ARB_vertex_program`- und `ARB_fragment_program`-Erweiterungen für OpenGL ist es möglich, die `Vertex` und `Fragment Processing`-Stufen der Grafik-Pipeline mit Vertex respektive Fragment Shadern[3] frei zu programmieren. Allerdings waren diese frühen APIs stark eingeschränkt; die Syntax der Shadersprache war an Assembler angelehnt und die Anzahl der Instruktionen pro Shader war sehr stark begrenzt. Durch die Weiterentwicklung der Grafik-Hardware konnten die Limitierungen jedoch zunehmend aufgehoben und mit HLSL, Cg und GLSL C-ähnliche Sprachen für die Shader-Programmierung entwickelt werden.

Die freie Programmierbarkeit der Vertex und Fragment Shader ermöglicht die Implementierung komplexer, hardware-beschleunigter Effekte. Ohne Shader wären die Effekte gar nicht oder nur mit sehr viel Programmieraufwand realisierbar. Der Erfolg der Shader zeigt sich auch dadurch, dass mit DirectX 10 sogar noch eine weitere Shader-Art, Geometry Shader, hinzugekommen ist. Zusätzlich wurde die *Fixed Function Pipeline*, also die Pipeline ohne frei programmierbare Shader, komplett entfernt. DirectX 11 wird noch drei weitere Shader-Arten hinzufügen und HLSL um die Unterstützung von objekt-orientierten Konzepten erweitern.

Horde3D verwendet GLSL als Shadersprache und unterstützt derzeit offiziell nur Vertex und Fragment Shader. Wichtig für die Verwendung von Shadern ist das Verständnis der Funktionsweise und der Kommunikationsmöglichkeiten zwischen den beiden Shader-Arten.

Der Shader Code wird für jedes Vertex und jedes Fragment einzeln ausgeführt, insbesondere kennt ein Shader keine anderen Eckpunkte oder Fragmente. Nur durch diese Einschränkung kann der hohe Grad der Parallelisierbarkeit der Shader gewährleistet werden – moderne Grafikkarten berechnen hunderte Shader gleichzeitig.

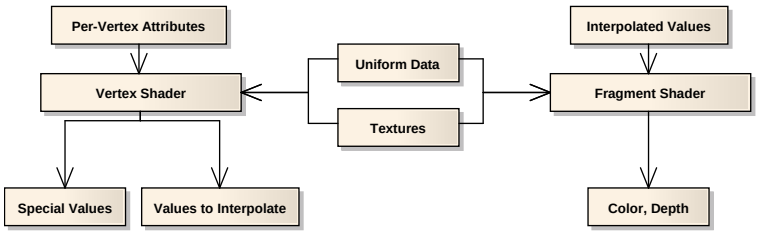

Abbildung 2.2.: Datenfluss zwischen Vertex und Fragment Shadern in OpenGL 2.0

Abbildung 2.2 zeigt den Datenfluss zwischen den Shadern. Der Vertex Shader erhält Attribute für jedes Vertex, beispielsweise die Position und die Textur-Koordinaten. Über *Uniform*s können Vertex-unabhängige Daten, wie etwa die Transformations- und Projektionsmatrizen, Position der Lichtquelle, etc., abgerufen werden. Manche *Uniform*s werden von OpenGL auch als *built-in* Variablen zur Verfügung gestellt[4]. Auch Texturen können als Eingabe verwendet werden. Die Ausgabe erfolgt an spezielle Ausgabevariablen oder an Variablen, die über alle

[3] Was unter OpenGL Fragment Shader heißt, wird von Direct3D als Pixel Shader bezeichnet. Da die Fragment/Pixel Shader aber auf Fragmenten und nicht auf Pixeln arbeiten, ist die OpenGL-Terminologie zutreffender.

[4] *Built-in* Variablen wie beispielsweise Nebel- oder Lichtquellen-Einstellungen sind aus Effizienzgründen seit OpenGL 3.1 kein Teil des Standards mehr [6].

Eckpunkte eines Primitivs interpoliert und an den Fragment Shader weitergereicht werden. Der Fragment Shader kann mit diesen interpolierten Werten weiterrechnen und hat ebenfalls Zugriff auf die *Uniforms*. Die Ausgabe des Fragment Shaders ist ein Farbwert und implizit auch ein Tiefenwert. Bei der Verwendung von MRTs werden mehrere Farbwerte ausgegeben [7, S. 52].

Horde3D verwendet zwar GLSL, führt aber ein XML-ähnliches Format ein, um die Ausdrucksstärke der Shader zu erweitern. Die Erweiterung ist an die *HLSL Effects* von Direct3D angelehnt. Dieses Framework ermöglicht es, für einen Effekt verschiedene Stufen zu definieren – von Horde3D als Kontext bezeichnet –, die jeweils unterschiedlichen Shader Code ausführen. Es können auch *Uniforms* global definiert und Einstellungen für verwendete Texturen, wie zum Beispiel der anzuwendende Texturfilter, festgelegt werden. Jeder Kontext kann außerdem die Vergleichsfunktion für den Tiefen- und Alphatest auswählen und eine Funktion für das *Blending* spezifizieren.

Das Pipeline-Konzept von Horde3D geht jedoch über die *HLSL Effects* hinaus. Pipelines ermöglichen die Definition verschiedener *Render Targets* und die Festlegung deren Verwendung in den einzelnen Zeichenschritten als Ein- oder Ausgabe für die Shader. Es kann genau spezifiziert werden, welche Material-Klassen mit welchen Shader-Kontexten gezeichnet werden, wann *Overlays* über die Szene gelegt werden und welche Art der Lichtberechnung – *Forward* oder *Deferred Shading* – verwendet wird.

2.2. Anforderungen an das System

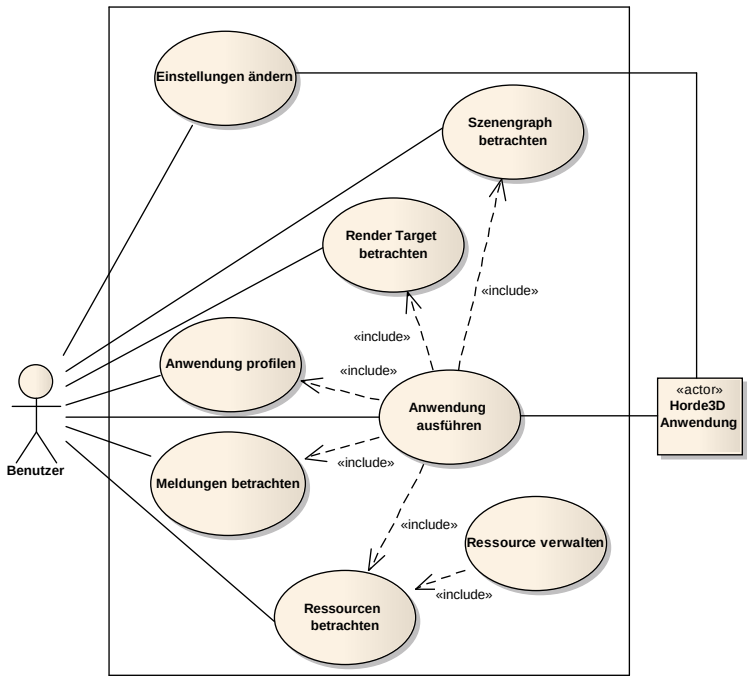

Abbildung 2.3.: Das *Use Case* Modell des Horde3D Development Environments

Nach der Betrachtung der Funktionsweise von Horde3D mussten die Anforderungen an

das Horde3D Development Environment konkretisiert werden. Es kamen viele Möglichkeiten in Betracht, die Entwickler eines Spiels beim Erstellen, Optimieren und Feinabstimmen der Spieleeffekte zu unterstützen. Im Rahmen dieser Bachelorarbeit wurden deshalb ausschließlich diejenigen Anforderungen betrachtet, die speziell bei der Entwicklung von SheepMeUp von Nutzen gewesen wären. Weitere denkbare Anforderungen und Erweiterungen des Systems werden in Kapitel 5.4 vorgestellt.

Abbildung 2.3 zeigt das *Use Case* Modell des Systems. Die Anwendungsfälle werden in den folgenden Abschnitten genauer beschrieben. Aktivitätsdiagramme für die *Use Cases* sind im Anhang abgebildet.

2.2.1. Unabhängigkeit und Eigenständigkeit

Eine übergeordnete Anforderung an das System war die Unabhängigkeit von der konkreten Horde3D-Anwendung und von der Implementierung von Horde3D. Um das Horde3D Development Environment beim Entwickeln einer Anwendung verwenden zu können, sollen keine Änderungen an der Anwendung nötig sein; es sollen keine speziellen Funktionen aufgerufen oder gegen zusätzliche Programmbibliotheken gelinkt werden müssen.

Das System soll aber auch unabhängig von der Implementierung von Horde3D sein. Dies verhindert das Überladen der Kern-API der Engine mit Tool-spezifischen Funktionen und erleichtert die Weiterentwicklung in getrennten Entwicklergruppen.

Einzige Voraussetzung an Horde3D-Anwendungen ist, dass eine unmodifizierte Horde3D DLL verwendet wird[5].

2.2.2. Absicherung vor unerwünschtem Reverse-Engineering

Mit dem Horde3D Development Environment können Einblicke in den Aufbau und in die Funktionsweise der Effekte von Horde3D-Anwendung gewonnen werden. Dies könnte möglicherweise von manchen Anwendungsentwicklern unerwünscht sein. Das System soll seine Dienste verweigern, wenn die Anwendung die Verwendung des Horde3D Development Environments nicht explizit erlaubt. Um die Verwendung nur explizit zuzulassen, ist jedoch eine Modifizierung der Anwendung notwendig. Die damit einhergehende Verletzung des Ziels der Unabhängigkeit wird aber bewusst in Kauf genommen. So soll es für die Entwickler von Horde3D-Anwendungen möglich sein, das Horde3D Development Environment nur mit *Debug Builds* ihrer Anwendung zu verwenden, wohingegen das Tool bei *Release Builds* nicht verwendet werden kann.

Diese Anforderung wurde erst in einer späteren Iteration aufgenommen. Ursprünglich regte Nicolas Schulz, der Entwickler von Horde3D, die Entwicklung des Schutzmechanismus an.

2.2.3. Konfigurieren und Ausführen der Horde3D-Anwendung

Das System soll als eigenständig ausführbares Programm entwickelt werden, aus dem heraus die Horde3D-Anwendung gestartet werden kann. Dadurch wird es erleichtert, die Anforderung der Unabhängigkeit zu erfüllen. Eine Alternative wäre, während der Ausführung der Anwendung beim Drücken einer bestimmten Taste das Horde3D Development Environment zu laden und anzuzeigen. Dies hätte jedoch Änderungen an der Anwendung oder an Horde3D

[5] Die derzeitige Implementierung unterstützt ohne erneutes Kompilieren keine modifzierten Horde3D DLLs oder Horde3D-Extensions.

notwendig gemacht. Ein weiterer Vorteil der gewählten Lösung ist die Möglichkeit, das Tool später auch „offline“ – also ohne laufende Horde3D-Anwendung – verwenden zu können (siehe auch Kapitel 5.4).

Für den Start der Horde3D-Anwendung sollen verschiedene Parameter eingestellt und gespeichert werden können: der Pfad zur *Executable* der Anwendung, das Arbeitsverzeichnis, Kommandozeilenparameter sowie verschiedene implementierungsspezifische Details. Das Horde3D Development Environment soll eine Eingabemaske für diese Parameter bereitstellen und die Pfade auf Korrektheit überprüfen. Abbildung B.1 zeigt das Aktivitätsdiagramm für die Konfiguration der Anwendungsparameter.

Abbildung B.2 erläutert die Verwendung des Horde3D Development Environments. Das System soll auf Wunsch des Benutzers die Anwendung starten. Der Benutzer soll grundsätzlich jederzeit die Möglichkeit haben, die Anwendung wieder zu beenden. Während das Programm läuft, soll das System sofort alle von Horde3D generierten Fehler-, Debug- und Informationsmeldungen in einer Übersicht anzeigen. Damit kann der Benutzer Probleme seiner Anwendung identifizieren, ohne die von den Horde3DUtils generierte HTML-Datei durchzusehen.

Hat die Anwendung einen Zustand erreicht, in dem der Benutzer die Anwendung anhalten möchte, teilt er dies dem System mit. Das System soll die Anwendung anhalten und es dem Benutzer ermöglichen, die aktuelle Szene zu betrachten und Ressourcen zu ändern, während sich die Szene selbst nicht ändert. Dies kann im Allgemeinen, also ohne ein spezielles Anhalten der Anwendung, nicht garantiert werden: Ein Explosionseffekt in einem Spiel wird beispielsweise nur wenige Sekunden andauern. Möchte man den Effekt ändern, so kann das Anhalten der Explosionsanimation beim Finetuning der Details der Explosion hilfreich sein.

Während die Szene eingefroren ist, soll der Benutzer den Zustand der Anwendung analysieren können:

- Das System soll alle bekannten Ressourcen auflisten. Falls eine Ressource editierbar ist, soll auf Wunsch die XML-Datei der Ressource geladen werden können. Der Benutzer soll die XML-Datei frei ändern und die Änderungen speichern können. Anschließend soll das System die Szene mit der geänderten Ressource automatisch neu zeichnen.

- Während der Ausführung der Anwendung werden von Horde3D Fehler- und Debug-Meldungen erzeugt. Das System soll diese bereits während der Ausführung anzeigen und eine Filterung nach Wichtigkeit unterstützen.

- Mit dem Tool soll es möglich sein, den Inhalt von aktiven *Render Targets* zu betrachten. Da sich deren Daten, während die Anwendung nicht eingefroren ist, in jedem Frame ändern können, soll das System automatisch in kurzen Zeitintervallen den dargestellten Inhalt aktualisieren.

- Das Horde3D Development Environment soll dem Benutzer eine einfache Profiling-Funktion anbieten. Das System soll die Aufrufe aller Horde3D-Funktionen aufzeichnen und dem Benutzer anzeigen. Es soll dann eine Auswertung der gewonnenen Daten möglich sein.

- Der Benutzer soll einen Überblick über den aktuellen Zustand des Szenengraphs erhalten können. Dazu soll der komplette Szenengraph als Baum dargestellt werden. Zu jedem *Scene Node* sollen auf Wunsch weitere Details, wie etwa Typ, Name, Transformationswerte, etc., angezeigt werden.

Anschließend kann der Benutzer entweder die Anwendung beenden oder mit der Ausführung fortfahren. Anders als in Abbildung B.2 dargestellt, soll das System aber auch während der Weiterausführung das Ändern von Ressourcen und Anzeigen des Inhalts von *Render Targets* erlauben.

2.2.4. Bearbeiten von Ressourcen und sofortiges Anzeigen der Änderungen

Das System soll das Neuladen von geänderten Ressourcen ohne explizite Unterstützung der Anwendung ermöglichen. Das ist der wichtigste *Use Case* des Horde3D Development Environments: Jeder beliebige Shader- oder Partikel-basierte Spezialeffekt, jedes Material und jede Pipeline soll mit dem Tool zur Laufzeit der Anwendung, aber völlig unabhängig von dieser, geändert werden können. Die Anwendung soll beim nächsten Zeichnen der Szene sofort die aktualisierten Ressourcen verwenden. Abbildungen B.3 und B.4 zeigen den Ablauf dieses Anwendungsfalls. Zunächst sollen alle Ressourcen in einer Übersicht angezeigt werden. Das System soll eine Filtermöglichkeit nach Ressourcen-Typ und -Name anbieten. Falls der Benutzer eine editierbare Ressource auswählt, soll das System die zugrundeliegende XML-Datei der Ressource in einem Texteditor anzeigen. Änderungen an der XML-Datei sollen gespeichert und sofort an die Horde3D-Anwendung übermittelt werden können.

Beim Entwickeln der Kraftfeld- und Schockwellen-Effekte für SheepMeUp war es wichtig, verschiedene Parameter der Effekte fein zu justieren und an das *Look and Feel* des Spiels anzupassen. Das Ausblenden des Effekts, die Farbgebung, die Intensität und die Abspieldauer mussten von Hand eingestellt werden. Da weder Horde3D noch SheepMeUp ein automatisches Neuladen der Ressourcen unterstützen, war nur folgendes Vorgehen möglich:

1. Die XML-Datei des Effekts ändern.
2. Eventuell muss die Anwendung neu kompiliert und gelinkt werden[6].
3. Das Spiel starten.
4. Bis zur entsprechenden Stelle im Spiel spielen.
5. Das Aussehen des Effekts beurteilen.
6. Bei Nichtgefallen zurück zu Schritt 1.

Dieser Prozess war zeitintensiv und unproduktiv. Das Horde3D Development Environment soll ein kontinuierliches Testen und Anpassen der Effekte ermöglichen und somit die Optimierung der Effekte leichter und schneller gestalten. Dies führt zu einer entsprechenden Verbesserung der Ästhetik des Spiels, und somit auch des Spiels insgesamt [4, nach S. 89]. Das verbesserte Vorgehen lässt sich wie folgt zusammenfassen:

1. Das Spiel starten.
2. Bis zur entsprechenden Stelle im Spiel spielen.

[6] Bei der Entwicklung von SheepMeUp war eine Neukompilierung erforderlich, wenn der Effekt von Werten aus der Anwendung abhing, die ebenfalls angepasst werden mussten. Dieses Problem kann das Horde3D Development Environment nicht lösen. Wäre SheepMeUp nicht in C++, sondern in C# entwickelt worden, hätte dieses Problem durch die *Edit And Run*-Funktionalität von Visual Studio vermieden werden können.

3. Das Aussehen des Effekts beurteilen.
4. Die XML-Datei des Effekts ändern, gegebenenfalls sogar unterstützt durch einen Designer.
5. Bei Nichtgefallen zurück zu Schritt 3.

Während der Entwicklung des Systems zeigte sich außerdem, dass das Shader-Format von Horde3D bei komplexen Shadern unübersichtlich werden kann. Aus diesem Grund wurde die Entwicklung eines visuellen Designers für Horde3D-Shader in die Anforderungen aufgenommen. Im Rahmen dieser Bachelorarbeit soll die nötige Infrastruktur für den Shader-Designer implementiert sowie die automatische Synchronisation des Designers und der XML-Datei umgesetzt werden. Der Designer soll das Ändern von bereits vorhandenen Shadern unterstützen; neue Shader sollen damit nicht angelegt werden können. In Kapitel 5.4 wird kurz auf die Limitierungen der derzeitigen Implementierung des Designers eingegangen.

2.2.5. Anzeigen von Fehlern und Debug-Informationen

Horde3D generiert laufend Fehler- und Debug-Meldungen. Diese Meldungen sind die einzige Möglichkeit, mehr über aufgetretene Probleme beziehungsweise den derzeitigen internen Zustand der Engine zu erfahren. So traten bei der Entwicklung von SheepMeUp mehrere Probleme auf, die nur mit Hilfe von Horde3D-Fehlermeldungen gelöst werden konnten. Beispielsweise kam es in unregelmäßigen Zeitabständen vor, dass scheinbar zufällig ausgewählte Schafe aus der Szene entfernt wurden. Im Anwendungscode gab es keine Hinweise, die dieses Problem erklären konnten, da Schafe nur an einer genau definierten Stelle gelöscht wurden. Der entsprechende Code wurde beim Auftreten des Bugs aber nicht ausgeführt. Erst ein Blick in die von den Horde3DUtils generierte HTML-Datei führte zur Lösung des Problems. Die Datei enthielt mehrere Warnungen über Versuche, *Scene Nodes* über ungültige *Node Handles* aus dem Szenengraph zu löschen. Es stellte sich heraus, dass die Schafe sowohl vom Anwendungscode gelöscht wurden als auch von der verwendeten Game Engine der Universität Augsburg. Da Horde3D nach dem Löschen eines Knoten dessen *Node Handle* neu vergibt, wurde manchmal beim zweiten Löschen ein neu erzeugter Knoten im Szenengraph gelöscht. Mit diesem Wissen war das Problem leicht zu beheben und die HTML-Datei war anschließend frei von Fehlermeldungen.

Die Wichtigkeit der Horde3D-Meldungen beim Entwickeln von Anwendungen ist nicht zu unterschätzen. Problematisch ist, dass die Meldungen standardmäßig gar nicht angezeigt werden und von den Horde3DUtils nur in eine HTML-Datei geschrieben werden. Dort aber übersieht man wichtige Informationen leicht. Im Zusammenhang mit dem Neuladen von aktualisierten Ressourcen sind die Meldungen aber auch noch aus einem weiteren Grund sehr wichtig. Sie zeigen nämlich Probleme beim Neuladen der Ressourcen an, beispielsweise Syntaxfehler im GLSL-Code oder falsche Pfade zu referenzierten Ressourcen.

Das System soll zum einen alle generierten Meldungen anzeigen und zum anderen auch das Filtern der Meldungen nach Wichtigkeit – also Fehler, Warnung, Debug-Information – unterstützen. Abbildung B.5 zeigt das Aktivitätsdiagramm für diesen *Use Case*.

2.2.6. Anzeigen von Render Targets

Für die Kraftfeld- und Schockwellen-Effekte verwendet SheepMeUp mehrere *Render Targets*, die die aktuellen Szenen-Daten, den Abstand eines Pixels zur Kamera und Ergebnisse

verschiedener Zwischenberechnungen enthalten. Beim Programmieren der Shader wäre es hilfreich gewesen, direkt den Inhalt der RTs betrachten zu können. Als Workaround musste ein spezieller Schritt in die Pipeline-Konfiguration von SheepMeUp eingeführt werden, der den Inhalt eines *Render Targets* in den *Backbuffer* kopiert und anzeigt. Das Spiel musste dann neu gestartet und bis zur entsprechenden Stelle gespielt werden, um den Inhalt des RTs betrachten zu können.

Render Targets werden generell für die Grafikentwicklung immer wichtiger. *Post Processing* Shader benötigen die gezeichnete Szene als Eingabe und das immer mehr in Mode kommende *Deferred Shading* benötigt MRTs, um die Position jedes Pixels in der 3D-Welt, die Normale des Pixels und Materialeigenschaften zu speichern [7, S. 255ff]. Daher soll das Horde3D Development Environment das Betrachten des RT-Inhalts vereinfachen. Wie Abbildung B.6 zeigt, soll das System dem Benutzer alle bekannten *Render Targets* der eingefrorenen Szene zur Auswahl anbieten. Nachdem der Benutzer eines ausgewählt hat, soll das Tool den Inhalt des RTs darstellen. Da sich dieser im Allgemeinen in jedem Frame ändert, soll das System immer die aktuellen Daten anzeigen. Dies soll auch während der Ausführung der Anwendung möglich sein. Bei der Implementierung dieses Features zeigte sich jedoch, dass ein Auslesen der RT-Daten in Echtzeit zu unperformant ist. Stattdessen soll ein Intervall eingestellt werden können, innerhalb dessen die angezeigten Daten aktualisiert werden.

2.2.7. Profiling von Horde3D-Funktionen

Grafikberechnungen sind sehr leistungshungrig, daher kann eine Optimierung des Codes lohnend sein – eventuell gestützt durch Profiling-Tools wie Intels VTune[7]. Es ist keine Anforderung an das Horde3D Development Environment, etablierte Profiling-Tools zu ersetzen. Es soll aber möglich sein, einen generellen Überblick über die Aufrufkosten von Horde3D-Funktionen zu erhalten und entsprechende Auswertungen vorzunehmen. Das System soll eine Szene profilen können, indem es für einen Frame die Aufrufdaten der Horde3D-Funktionen – den Zeitpunkt des Aufrufs, die Ausführungsdauer, sowie die Aufrufsreihenfolge und damit die Aufrufshäufigkeit – protokolliert und dem Benutzer anzeigt. Der Benutzer soll die gewonnen Daten nach folgenden Kriterien analysieren können: Durchschnittliche Ausführungsdauer einer Funktion, absolute Ausführungsdauer einer Funktion und Anzahl der Aufrufe einer Funktion. Es soll außerdem eine Aufruf-Historie dargestellt werden, die den zeitlichen Ablauf der Funktionsaufrufe repräsentiert. Abbildung B.7 zeigt diesen Anwendungsfall als Aktivitätsdiagramm.

2.2.8. Anzeigen des Szenengraphs

Im Rahmen der Bachelorarbeit soll eine Visualisierung des Zustands des Szenengraphs einer eingefrorenen Szene implementiert werden. Die Visualisierung soll dabei die Baumstruktur des Szenengraphs widerspiegeln. Wie Abbildung B.8 illustriert, sollen zu jedem *Scene Node* auf Wunsch des Benutzers weitere Details angezeigt werden, wie beispielsweise der Typ des Knotens, sein Name oder seine Transformationswerte. Für das Auslesen und Anzeigen der Daten soll die benötigte Infrastruktur entworfen und implementiert werden. In Kapitel 5.4 werden Ideen und Möglichkeiten für zukünftige Versionen des Tools vorgestellt, um diese Daten für interessante und hilfreiche Features zu verwenden.

[7] `http://software.intel.com/en-us/intel-vtune`

2.3. Konzeptmodell

Abbildung B.9 zeigt das Konzeptmodell des Horde3D Development Environments. Die meisten Konzepte, ihre Abhängigkeiten und Spezialisierungshierarchien wurden von der Struktur und vom Aufbau von Horde3D vorgegeben. Das Modell musste nur um die Konzepte `Horde3DApplication`, `FunctionCall` und `EditableResource` erweitert werden. Ersteres repräsentiert die Anwendung, die vom Tool gestartet wird. `FunctionCall` wurden gegenüber den System-Anforderungen noch um die Attribute `ReturnValue` und `Parameters` erweitert. In diesen beiden Attributen werden die Werte, die an die Funktion übergeben beziehungsweise von ihr zurückgegeben werden, aufgezeichnet. Diese Daten wurden erst in einer späteren Iteration ins Konzeptmodell aufgenommen. Damit wird eine der in Kapitel 5.4 beschriebenen Erweiterungen bereits vorbereitet.

Im Konzeptmodell wurden alle Typen von *Scene Nodes* als Spezialisierungen des abstrakten Konzepts `SceneNode` dargestellt. Bei den Ressourcen wurde zusätzlich noch das Konzept der editierbaren Ressource hinzugefügt, da das Horde3D Development Environment nur gewisse Ressourcentypen – Materials, Pipelines, Shaders, Particle Effects und Code Ressourcen – verändern kann. Alle editierbaren Ressourcen sind durch das abstrakte Konzept `EditableResource` generalisiert, während alle anderen Ressourcen und `EditableResource` selbst Spezialisierungen des allgemeineren abstrakten Konzepts `Resource` sind.

Es wurden außerdem die Beziehungen der einzelnen Ressourcen untereinander untersucht. Die Pipeline Konfiguration wurde auf einzelne Konzepte aufgeteilt, um möglichst flexibel auf Änderungen und Erweiterungen der Pipeline in neuen Horde3D-Versionen reagieren zu können. Außerdem wurden die Assoziationen zwischen den *Scene Nodes* und ihren benötigten Ressourcen eingezeichnet. Damit wird es später möglich sein, für einen *Scene Node* die verwendeten Ressourcen zu betrachten beziehungsweise für eine Ressource herauszufinden, von welchen Teilen des Szenengraphs sie verwendet wird. Diese detaillierte Analyse der Assoziationen zwischen den Konzepten war zwar nicht aufgrund der *Use Cases* erforderlich, verbesserte aber das Verständnis des Problembereichs und wird für einige der vorgeschlagenen Erweiterungen in Kapitel 5.4 benötigt.

Die Horde3D-Konzepte, ihre Attribute und ihre Abhängigkeiten untereinander wurden aus der API-Beschreibung der Horde3D-Dokumentation [3, „Engine API Reference“] ermittelt. Da während der Entwicklung der Anwendung eine neue Horde3D-Version erschien, mussten in einer späteren Iteration einige wenige Details der Konzepte verändert werden. Hierbei machte sich allerdings die starke Modularisierung des Konzeptmodells positiv bemerkbar.

2.4. Verwandte Softwaretools

Neben dem Horde3D Development Environment gibt es eine Reihe weiterer Tools mit dem Ziel, die Entwicklung von 3D-Anwendungen zu vereinfachen und zu beschleunigen. Dabei unterscheiden sich die Tools nicht nur in ihrer Umsetzung, sondern vor allem auch darin, welches Problem sie zu lösen versuchen.

Nachdem die erste Iteration der Analyse abgeschlossen war, wurden die gefundenen Anforderungen mit bereits vorhandenen Tools verglichen. Teilweise dienten Ideen anderer Tools als Grundlage für Erweiterungen und Ergänzungen der Anforderungen während der zweiten Iteration der Analyse-Phase. Es konnten aber auch Ideen für die Design- und Implementierungs-Phasen gewonnen werden.

Die folgenden Tools wurden eingehender betrachtet. Es wird kurz auf den Verwendungszweck, die Unterschiede zum Horde3D Development Environment und auf die für die weitere Entwicklung hilfreichen Ideen eingegangen.

2.4.1. Horde3D Scene Editor

Volker Wiendl entwickelte den Horde3D Scene Editor [8] zum Erstellen und Manipulieren von Szenen. Die erstellten Szenen können als *Scene Graph Resource* gespeichert und von anderen Horde3D-Anwendung geladen und angezeigt werden. Die Hauptaufgabe des Tools unterscheidet sich somit grundlegend von den Anforderungen an das Horde3D Development Environment. Es gibt jedoch auch Überschneidungen, da der Editor auch Unterstützung für die Entwicklung von Effekten bietet:

- Ressourcen, die außerhalb des Editors verändert werden, werden sofort neu geladen und die Szene mit den geänderten Ressourcen gezeichnet. Das Horde3D Development Environment bemerkt externe Änderungen der Ressourcen-Dateien derzeit nicht.
- Es ist ein Designer zum Ändern von Materialeigenschaften integriert. Das Horde3D Development Environment bietet hierfür keinen Designer an, sondern erlaubt nur die direkte Änderung der XML-Datei des Materials. Allerdings bietet der Szenen-Editor keinen Designer für Shader an.
- Auch der Editor erlaubt das Anzeigen des Inhalts eines beliebigen *Render Targets*. Jedoch wird der dargestellte Inhalt nicht automatisch aktualisiert, wenn sich die Szene ändert.
- Der Editor verwendet Horde3D zum Darstellen der Szene. Allerdings läuft dabei nicht der Code der Anwendung mit, für die die Szene erstellt wird. Dies ist problematisch, wenn Shader-Effekte von Parametern abhängen, die vom Anwendungscode in jedem Frame neu gesetzt werden müssen. In SheepMeUp betraf dies die Kraftfeld- und Schockwellen-Effekte, die von einem Zeitparameter abhingen. Da der Editor diesen Parameter nicht aktualisierte, wurden die Effekte innerhalb des Editors nicht korrekt dargestellt.
- Die Szene kann nicht eingefroren und in Ruhe betrachtet werden, was zum Beispiel das Entwickeln von kurzlebigen Partikeleffekten erschweren kann. Auch der Schockwellen-Effekt von SheepMeUp wird nur etwa eine Sekunde lang angezeigt. Das Finetuning solcher Effekte wird durch das Einfrieren der Szene erleichtert.
- Der Szenen-Editor kann durch Plugins erweitert werden, wodurch prinzipiell auch der Anwendungscode im Editor mitlaufen könnte. Allerdings erfordert dies eine Anpassung der Anwendung an den Editor, was in vielen Fällen nicht erwünscht ist. Als Alternative gibt es die Game Engine der Universität Augsburg[8], die eng mit dem Szenen-Editor verknüpft ist und beispielsweise Plugins für Physik und Sound bietet. Um von diesen Plugins profitieren zu können, müsste die Anwendung neben Horde3D aber auch die Game Engine verwenden. Beim Horde3D Development Environment hingegen sollen alle Features unabhängig von der Anwendung verwendbar sein; es kann mit jeder beliebigen Horde3D-Anwendung – auch mit dem Szenen-Editor und der Game Engine – zusammenarbeiten, ohne Änderungen am Code nötig zu machen. Die Horde3D-Anwendung

[8] `http://mm-werkstatt.informatik.uni-augsburg.de/projects/GameEngine/doku.php`

wird dabei normal ausgeführt und bekommt von der Verwendung des Horde3D Development Environments nichts mit.

- Der Szenen-Editor ist plattformunabhängig in C++ und Qt implementiert. Das Horde3D Development Environment hingegen läuft nur unter Windows, da das .NET Framework 3.5 Service Pack 1 vorausgesetzt wird und einige Windows-spezifische Funktionen und Bibliotheken verwendet werden. Jedoch ist eine Implementierung in C# und .NET im Allgemeinen weniger fehleranfällig, einfacher und schneller.

2.4.2. PIX for Windows

Microsofts DirectX SDK[9] enthält das Direct3D Debugging-Tool PIX for Windows. Die Hauptaufgabe von PIX ist das Auffinden von Problemen bei der Verwendung der Direct3D-API. Es wird eine Historie aller Direct3D Funktionsaufrufe angezeigt, sowie wichtige Daten über alle allozierten Direct3D-Ressourcen wie *Vertex Buffer*, *Index Buffer*, Texturen oder *Render Targets*. Man kann sich den Inhalt eines *Render Targets* anzeigen und die Szene schrittweise – also *Render Call* für *Render Call* – zeichnen lassen. Besonders hilfreich ist die Möglichkeit, Vertex, Geometry und Pixel Shader zu debuggen. Dazu wählt man einfach ein Vertex, Polygon oder Pixel in der Szene aus. PIX zeigt dann den Source Code des Shaders an und ermöglicht ein zeilenweises Durchlaufen des Codes und Anzeigen der Variablenwerte, wie man es vom Visual Studio Debugger gewohnt ist. Mit entsprechender Unterstützung des Grafikkartentreibers können auch verschiedene Metriken – wie Auslastung der *Raster Operation Units*, der *Texture Mapping Units* sowie der Shadereinheiten der Grafikkarte – für einen Frame untersucht und entsprechende Performanceoptimierungen an der Direct3D-Anwendung vorgenommen werden.

Horde3D hat derzeit nur einen OpenGL Renderpfad und somit ist die Verwendung von PIX für Horde3D-Anwendungen unmöglich. Die Ziele des Horde3D Development Environments unterscheiden sich allerdings hauptsächlich im Abstraktionslevel von den Zielen des Microsoft Tools: Beide Tools erlauben die Betrachtung des Zustands einer Software, die eine entsprechende API verwendet. Bei PIX ist diese API Direct3D, beim Horde3D Development Environment ist es Horde3D. Insbesondere stammt die Idee der Funktionsaufruf-Historie und -Performanceauswertung aus PIX.

2.4.3. NVIDIA PerfHUD

NVIDIA PerfHUD[10] ist ebenfalls ein Tool zum Debuggen von Direct3D-Anwendungen. Allerdings ist der Fokus nicht der gleiche wie bei PIX, sondern die Tools ergänzen sich in ihren Möglichkeiten. So erlaubt PerfHUD ebenfalls ein schrittweises Zeichnen der Szene, unterstützt allerdings kein Shaderdebugging. Auch ist PerfHUD keine eigenständige Anwendung wie PIX, sondern läuft direkt in der zu debuggenden Anwendung mit. PerfHUDs Hauptaufgabe ist die Unterstützung der Performanceoptimierung und Auffinden von Flaschenhälsen in der Grafik-Pipeline. Alle Performancecounter der NVIDIA Grafikkarten können mit dem Tool ausgelesen werden. Dies ermöglicht es beispielsweise, für einen *Draw Call* die Auslastung der Shader Einheiten der Grafikkarte, die CPU-Auslastung durch den Grafikkartentreiber usw. zu ermitteln und auszuwerten.

[9] `http://msdn.microsoft.com/en-us/directx/default.aspx`
[10] `http://developer.nvidia.com/nvperfhud_home.html`

Im Vergleich zum Horde3D Development Environment gibt es fast keine Überschneidungen in der Funktionalität. Jedoch macht es die Funktionsweise von PerfHUD erforderlich, die Szene einzufrieren. Der User Guide [9, S. 11] gibt einen Hinweis darauf, dass dies technisch über ein „Anhalten der Zeit“ gelöst wurde. Das Horde3D Development Environment verwendet einen analogen Lösungsansatz. Des Weiteren wurde Nicolas Schulz von diesem Tool inspiriert, als er den Schutz vor unerwünschtem *Reverse-Engineering* vorschlug. PerfHUD lässt sich nur mit Anwendungen verwenden, die beim Starten ein spezielles Direct3D-*Device* auswählen. Wählt die Anwendung in *Release Builds* das PerfHUD-*Device* nicht aus, verweigert das Tool seinen Dienst. Damit können Konkurrenzfirmen nicht herausfinden, wie die Anwendung eine Szene berechnet und auf welche Performancecharakteristika die Anwendung optimiert wurde.

2.4.4. NVIDIA FxComposer und AMD RenderMonkey Toolsuite

NVIDIA FxComposer[11] und AMD RenderMonkey Toolsuite[12] sind Entwicklungsumgebungen für Shader und Materials. Sie bieten eine Vorschau der erstellten Effekte an und können diese in verschiedene Formate für OpenGL- und Direct3D-Anwendungen exportieren. Allerdings gibt es derzeit keinen Exporter für das Horde3D-Shaderformat und GLSL wird von beiden Tools nicht direkt unterstützt.

Im Gegensatz zum Horde3D Development Environment ist jedoch keine direkte Integration mit der Anwendung vorgesehen; die erstellten Shader und Materials können also nicht sofort in der Anwendung betrachtet werden.

2.5. Konklusion

Es stellte sich als die richtige Entscheidung heraus, die Funktionsweise und den Aufbau von Horde3D vor der eigentlichen Anforderungsanalyse genau zu untersuchen. Dadurch konnten wichtige Einblicke in den Problembereich gewonnen werden, da das Konzeptmodell fast komplett vorgegeben wurde. Zusammen mit den Erfahrungen aus der Entwicklung von SheepMeUp konnten viele Systemanforderungen gefunden werden, die später als hilfreich und wichtig bestätigt wurden.

Die Betrachtung ähnlicher und verwandter Softwaretools bestätigte außerdem, dass es für Horde3D kein zum Horde3D Development Environment vergleichbares Tool gibt. Des Weiteren konnten einige Ideen der Tools in die Anforderungen, das Design oder die Implementierung des Systems integriert werden.

[11] `http://developer.nvidia.com/object/fx_composer_home.html`
[12] `http://developer.amd.com/GPU/RENDERMONKEY/Pages/default.aspx`

3. Phase II: Design

In der Design-Phase wurde die Architektur des Systems entworfen. Ausgehend von den Erkenntnissen der Analyse-Phase wurden zunächst wichtige, grundlegende Entscheidungen gefällt und schließlich ein Designklassen-Diagramm erstellt. Für einige wichtige Operationen wurden außerdem Sequenzdiagramme angefertigt, um weitere Einblicke in die Funktionsweise des Systems zu gewinnen. Für die Anbindung der GUI wurde ein eigenständiges Framework entwickelt, für das es ein eigenes Designklassen-Diagramm gibt.

3.1. Grundlegende Entscheidungen

Eine Anforderung an das System war die Unabhängigkeit von Horde3D und der Anwendung des Benutzers. Bereits in der Design-Phase musste daher sichergestellt werden, dass die Implementierung keine Änderungen an Horde3D oder der Benutzeranwendung voraussetzt. Die *Use Cases* fordern zusätzlich, dass das System die Anwendung des Benutzers starten und beenden kann. Eine Client-Server-Architektur zwischen Horde3D Development Environment und Anwendung, welche service-orientiert entworfen wurde, erfüllt diese Anforderungen. Was nun als Server bezeichnet wird, ist in diesem Fall allerdings nicht klar. Für die Horde3D-Anwendung spricht, dass sie die Daten hält, die von dem System ausgelesen und angezeigt werden. Für das System spricht hingegen, dass es die ganze Zeit läuft und die Horde3D-Anwendung erst starten muss. Das Festlegen einer genauen Terminologie wird zusätzlich durch den Einsatz eines DLL-*Replacement*-Mechanismus erschwert. Die originale Horde3D DLL wird durch eine modifizierte Version ersetzt, die alle Horde3D-Funktionsaufrufe – für die Anwendung völlig transparent – an die Original-DLL weiterleitet und intern noch zusätzlichen Code ausführt. Dadurch schleust das Horde3D Development Environment Code in die Horde3D-Anwendung ein; das System läuft sowohl server- als auch client-seitig.

In den nächsten Abschnitten wird nun folgende Terminologie verwendet: Mit „Horde3D-Anwendung“ ist die Anwendung des Benutzers gemeint, ohne DLL-*Replacement*. „Server“ bezeichnet den Code des Horde3D Development Environments, der innerhalb des Prozesses der Horde3D-Anwendung läuft. „Client“ oder „Shell“ ist der Teil des Systems, der als eigenständige Applikation lauffähig ist und Server-Instanzen starten und beenden kann.

Abbildung 3.1 verdeutlicht die Interaktionen zwischen Server und Client an einem Beispiel. Der Client wird dabei durch die Lebenslinie des linken Akteurs repräsentiert, die rechte Lebenslinie repräsentiert den Server. Die Shell führt zunächst ein paar vorbereitende Schritte durch – unter anderem das DLL-*Replacement* sowie einige Konfigurationsaufgaben – und startet schließlich die Horde3D-Anwendung. Anstelle der Original-DLL wird jedoch die modifizierte Horde3D DLL geladen und initialisiert. Während der Initialisierung wird schließlich die originale Horde3D DLL in den Prozess geladen. Anschließend wird die Anwendung normal ausgeführt und alle von Horde3D generierten Meldungen sofort an den Client geschickt.

Zu einem beliebigen Zeitpunkt weist der Benutzer die Shell an, die Horde3D-Anwendung zu pausieren. Nun können alle relevanten Daten – beispielsweise der aktuelle Zustand des Szenengraphs oder die derzeit bekannten Ressourcen – aus dem Server ausgelesen werden.

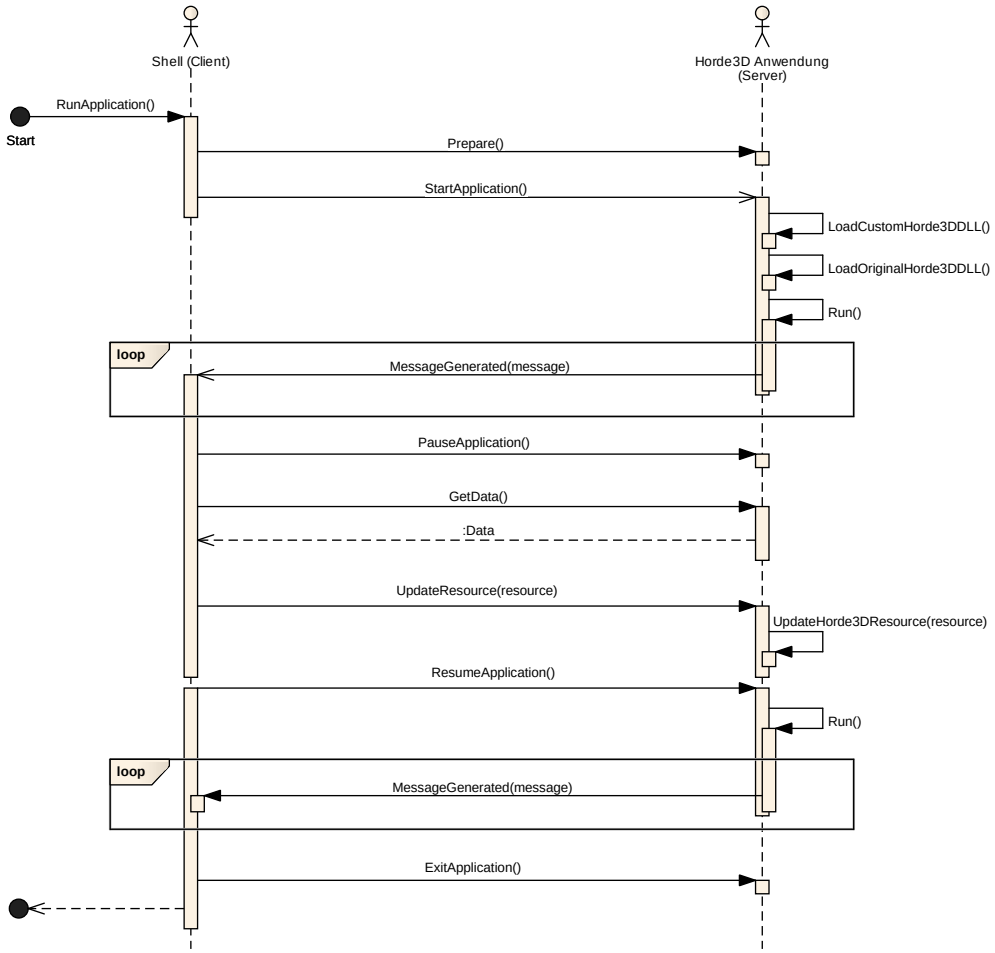

Abbildung 3.1.: Interaktionen zwischen Client und Server an einem Beispiel

Anschließend wird eine Ressource verändert und deren aktualisierte Daten wieder an den Server geschickt, der die entsprechende Horde3D-Ressource aktualisiert. Danach wird die Anwendung fortgesetzt und beim Zeichnen die aktualisierte Ressource verwendet, bis die Shell den Server schließlich beendet.

Auch über die grafische Benutzeroberfläche wurde bereits am Anfang der Design-Phase nachgedacht. Das *User Interface* sollte sich an GUIs bekannter Entwicklungsumgebungen wie Visual Studio und Eclipse orientieren. Abbildung 3.2 verdeutlicht die geplante GUI am Beispiel von Visual Studio 2008: In der Mitte des *Workspaces* werden mehrere Dokumente oder grafische Designer angezeigt. Oben gibt es die bereits von Windows bekannten Menüs und *Toolbars*. An der rechten und an der unteren Seite sind verschiedene *Tool Windows* versteckt, die erst beim Berühren mit der Maus sichtbar werden. Links ist das *Solution Explorer Tool Window* „angedockt“. Die *Tool Windows*, auch *Dock Panes* genannt, können vom Benutzer frei positioniert sowie versteckt und wieder sichtbar gemacht werden. Es besteht auch die Möglichkeit, die *Dock Panes* als eigenes Fenster (*Floating Window*) über die Anwendung zu legen. Das vom Benutzer frei gewählte Layout wird gespeichert und beim Starten der Anwendung automatisch wiederhergestellt.

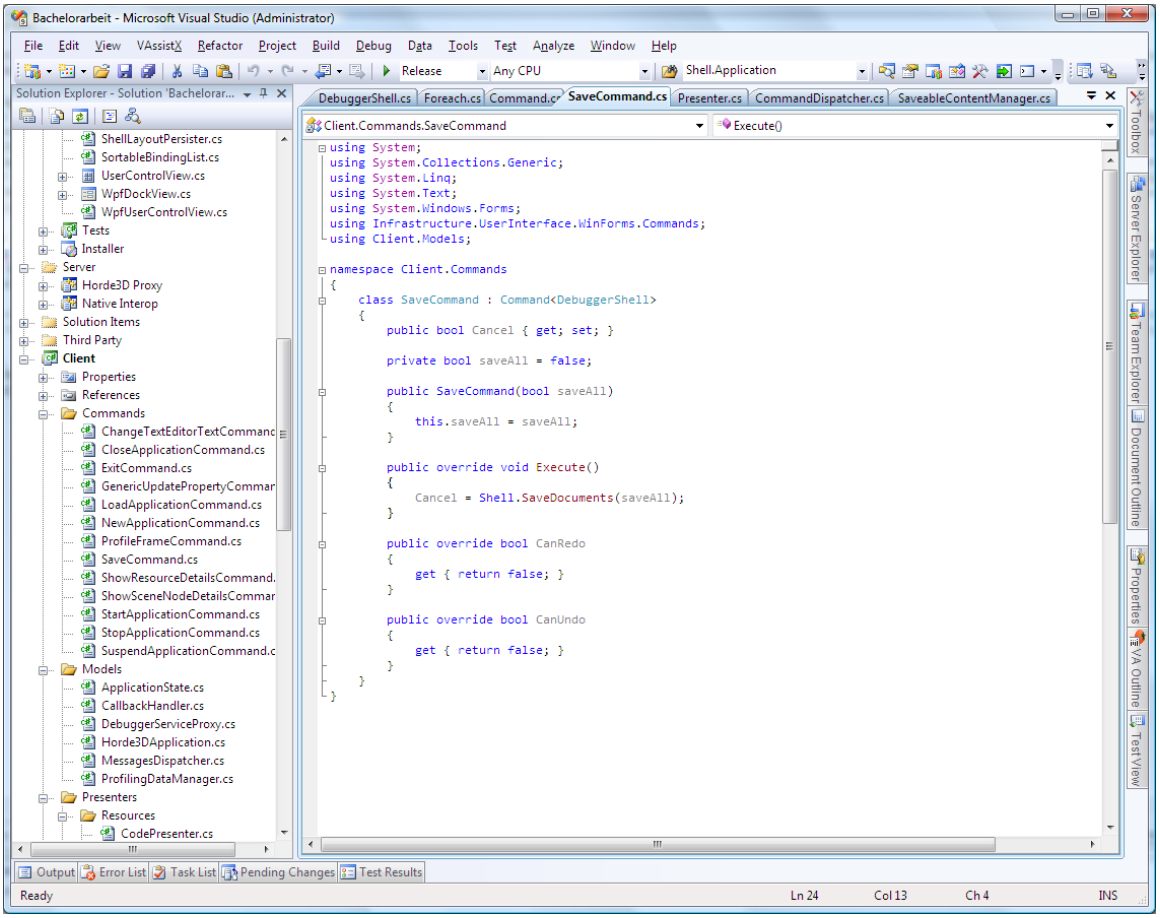

Abbildung 3.2.: Die grafische Benutzeroberfläche von Visual Studio 2008

Das Horde3D Development Environment wurde im Hinblick auf ein solches GUI-Design entworfen, da es weit verbreitet und bekannt ist und es dem Benutzer viele Möglichkeiten lässt, die Oberfläche an seine Wünsche und Vorlieben anzupassen.

Zu Beginn der Design-Phase wurde auch entschieden, dass das System in C# mit dem .NET Framework 3.5 Service Pack 1 implementiert wird. Dadurch ist das System zwar nur unter Windows lauffähig, aufgrund der höheren Produktivität gegenüber einer Implementierung mit C++ und Qt konnte bei der Entwicklung jedoch mehr Zeit in die eigentlich zu lösenden Probleme investiert werden. Diese Entscheidung wurde bereits zu diesem Zeitpunkt getroffen, damit schon in den Design-Artefakten .NET *Properties* und *Events* verwendet werden können, was die Darstellung erleichtert und verkürzt. Da UML jedoch keine Unterstützung für diese Konzepte anbietet, wurden die entsprechenden UML-Attribute und -Operationen mit den Stereotypen `property` (*Getter* und *Setter* für Attribute), `event` (ein Ereignis, das auftreten kann) und `event property` (an diesem kann sich ein Objekt für ein *Event* registrieren und deregistrieren) versehen.

All diese Entscheidungen hatten einen großen Einfluss auf den Gesamtentwurf des Systems. Deshalb war es wichtig zu überprüfen, ob die vorgestellten Konzepte überhaupt wie geplant umsetzbar sind. Vor dem Entwurf der Design-Dokumente wurde daher ein Prototyp entwickelt, der die Machbarkeit der Konzepte überprüfte und als durchführbar bestätigte.

3.2. Angewandte Entwurfsmuster

In der Softwareentwicklung gibt es für ein spezielles Architektur-Problem oftmals mehrere mögliche Lösungswege. Diese sind je nach Situation besser oder schlechter geeignet. Eine der wichtigsten Fähigkeiten eines Softwarearchitekten ist es daher, unterscheiden zu können, was für ein gegebenes Problem derjenige Lösungsansatz ist, der zum bestmöglichen Design führt [10, S. 4]. Auf der anderen Seite gibt es in der Softwareentwicklung eine Vielzahl an wiederkehrenden Entwurfsproblemen, für die es bewährte Lösungsschablonen gibt. Die Schablonen, *Design Patterns* oder Entwurfsmuster genannt, stellen eine allgemeine Vorlage zur Problemlösung dar, die nur noch auf den spezifischen Kontext des Problems angepasst werden muss. Gut strukturierte, objekt-orientierte Softwarearchitekturen verwenden eine Vielzahl an unterschiedlichen *Design Patterns* [11].

Beim Design des Horde3D Development Environments wurden verschiedene *Design Patterns* angewandt, die im folgenden kurz vorgestellt werden. Bei der Vorstellung des Systemdesigns wird immer wieder auf die Verwendung der *Patterns* zur Lösung eines Design-Problems hingewiesen.

3.2.1. General Responsibility Assignment Software Patterns

Für die Zuweisung der Verantwortlichkeiten auf die einzelnen Klassen wurden einige der *General Responsibility Assignment Software Patterns* verwendet [12]. Da die *Patterns* aber die Grundlage für objekt-orientierte Entwürfe bilden und somit vielfach angewandt wurden, wird auf eine Instanziierung eines GRAS Entwurfsmuster im Designmodell nicht hingewiesen. Im folgenden seien die verwendeten *GRAS Patterns* aber kurz vorgestellt:

- **Expert:** *Expert* ist das allgemeinste *Pattern*. Diejenige Klasse sollte die Verantwortlichkeit erhalten, die alle benötigten Informationen dafür besitzt.
- **Creator:** Das *Creator Pattern* gibt Hinweise darauf, welche Klasse eine Instanz einer anderen Klasse erzeugen sollte. Ein Objekt B sollte ein Objekt A erzeugen, wenn beispielsweise B eine Aggregation von A ist, oder B die benötigten Initialisierungsdaten für A besitzt.
- **Low Coupling:** Mit Kopplung bezeichnet man das Maß der Abhängigkeit einer Klasse von anderen Klassen. Eine niedrige Kopplung ist eines der wichtigsten Ziele von gutem objekt-orientierten Design, da es die Wiederverwendbarkeit erhöht und auch die Verständlichkeit und Wartbarkeit des Codes verbessert.
- **High Cohesion:** Die Kohäsion einer Klasse bezeichnet den semantischen Zusammenhang der Verantwortlichkeiten einer Klasse. Eine Klasse sollte möglichst wenige verschiedene Aufgaben enthalten, um eine hohe Kohäsion zu erreichen. Dadurch werden die Klassen kleiner, die Verantwortlichkeiten einer Klasse genauer definiert und der Code damit insgesamt besser wartbar, verständlich und wiederverwendbar. Jedoch steht das *High Cohesion Pattern* im Widerspruch zu *Low Coupling*. Beim Softwareentwurf muss daher eine angemessene Balance gefunden werden.
- **Polymorphismus:** Polymorphismus ist ein grundlegendes *Pattern* der objekt-orientierten Entwicklung, das von allen modernen objekt-orientierten Programmiersprachen direkt unterstützt wird. Durch die Verwendung von virtuellen und polymorphen Funktionen kann das Verhalten abhängig vom konkreten Typ der Klasse(n) geändert werden.

- **Pure Fabrication:** Mit *Pure Fabrication* bezeichnet man die Einführung von Design-Klassen, die im Design eine spezielle Aufgabe übernehmen, im Konzeptmodell aber nicht vorhanden sind.

3.2.2. Gang of Four Design Patterns

Der Entwurf des Horde3D Development Environments verwendet einige der bekannten und oft eingesetzten *Design Patterns* der *Gang Of Four* [11]. Diejenigen dieser Entwurfsmuster, die für das Systemdesign eingesetzt wurden, seinen hier kurz erläutert; an den entsprechenden Stellen wird auf ihre genaue Verwendung hingewiesen.

- **Composite:** Das *Composite Pattern* [11, S. 163ff] fügt mehrere Objekte in einer Baumstruktur zusammen, um eine Teil-Ganzes-Hierarchie zu repräsentieren. Ein Knoten muss nicht unterscheiden, ob eines seiner Kinder ein Blatt oder wiederum ein Knoten mit weiteren Kindern ist. Primitive Objekte und Behälter können also uniform behandelt werden.

- **Facade:** Das *Facade Pattern* [11, S. 185ff] fasst verschiedene Interfaces unter einem gemeinsamen, einfacher benutzbaren Interface zusammen. Damit werden Komplexität und Abhängigkeiten reduziert, und somit das System besser wartbar.

- **Command:** Beim *Command Pattern* [11, S. 233ff] wird eine Operation durch ein Objekt gekapselt. Durchgeführte Operationen können parametrisiert und protokolliert werden. Dadurch wird es möglich, Operationen später rückgängig zu machen.

- **Observer:** Das *Observer Pattern* [11, S. 293ff] wird direkt von zwei C#-Sprachfeatures unterstützt: *Delegates* und *Events*. Objekte können einen speziellen *Event Handler Delegate* für ein *Event* eines (anderen) Objekts registrieren. Löst das Objekt das Ereignis aus, werden alle registrierten *Delegates* automatisch ausgeführt. Das Objekt muss dafür nicht wissen, welche Objekte oder ob überhaupt Objekte an diesem Ereignis interessiert sind. Mit Hilfe dieses *Patterns* kann eine *One-To-Many*-Beziehung definiert werden, ohne zusätzliche Abhängigkeiten einzuführen.

- **Singleton:** Das *Singleton Design Pattern* [11, S. 127ff] stellt sicher, dass es nur eine Instanz einer Klasse geben kann. Für diese Instanz gibt es einen einzigen, globalen Zugriffspunkt.

- **Strategy:** Eine Menge von Algorithmen zur Lösung eines Problems kann mit dem *Strategy Pattern* [11, S. 315] gekapselt werden. Zur Laufzeit kann dann der am besten passende Algorithmus zur Lösung des Problems ausgewählt werden.

- **Decorator:** Das *Decorator Pattern* [11, S. 175] ermöglicht es, Verantwortlichkeiten dynamisch zur Laufzeit zu einem Objekt hinzuzufügen und wieder zu entfernen. Mit der klassische Vererbung hingegen können Verantwortlichkeiten nur statisch hinzugefügt werden.

- **Template Method:** Beim *Template Method Pattern* wird in einer Methode ein Algorithmus definiert, wobei manche Schritte als abstrakte oder virtuelle Methoden gekapselt sind. Abgeleitete Klassen müssen oder können diese Methoden überschreiben

und so einzelne Schritte, nicht aber die Struktur des Algorithmus, verändern. Die Unterstützung von Lambda-Funktionen in C# ermöglicht eine Variation des *Patterns*: Dem Algorithmus werden Lambda-Funktionen übergeben, die der Algorithmus an ausgewiesenen Stellen ausführt. Die Lambda-Funktionen ersetzen somit die abstrakten oder virtuellen Methoden.

3.2.3. Model-View-Presenter Design Pattern

Die Anbindung der GUI an das System ist über das *Model-View-Presenter Design Pattern* umgesetzt. Leider gibt es für dieses Entwurfsmuster keine genaue Definition; es gibt mehrere verschiedene Varianten und die Abgrenzung zum *Model-View-Controller Pattern* ist ebenfalls nicht ganz klar [13, Abschnitt „Model-View-Controller and Model-View-Presenter Confusion“]. Der gemeinsame Nenner ist lediglich die Separation der Anwendung in drei Bereiche:

- **Models:** Die Modelle sind die eigentlichen Businessobjekte der Anwendung aus dem Designmodell. Sie können gegebenenfalls mit dem *Facade* oder *Controller Pattern* gekapselt sein, um das Zugriffsinterface für die GUI-Anbindung zu vereinfachen.
- **Views:** Die *Views* sind für die grafische Anzeige zuständig. Sie sind die einzigen Klassen, die Abhängigkeiten zum gewählten GUI-Framework wie Windows Forms oder der Windows Presentation Foundation haben. Die *Views* zeigen die Daten der Modelle an; wie sie allerdings an die Daten kommen, ist nicht genau definiert.
- **Presenters:** Die Rolle der *Presenter* ist ebenfalls nicht klar festgelegt. Fest steht, dass die *Views* auf Benutzereingaben nicht selbst reagieren, sondern diese an ihren jeweiligen *Presenter* weiterreichen. Ob ein *Presenter* Daten an seine *View* schicken darf, hängt von der verwendeten Variante des *Model-View-Presenter Patterns* ab.

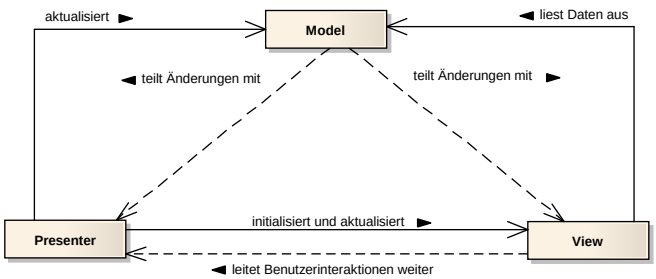

Abbildung 3.3.: Überblick über die *Supervising Controllers* Variante des *MVP Patterns*

Abbildung 3.3 zeigt die *Supervising Controllers* Variante des *MVP Patterns* [14], welche vom Horde3D Development Environment verwendet wird. Jeder *Presenter* hat genau eine *View* und umgekehrt. Der *Presenter* kennt seine *View* und kann diese jederzeit aktualisieren und auf deren Daten zugreifen. Die *View* hingegen kennt ihren *Presenter* nicht. Um die Benutzereingaben dennoch an den *Presenter* weiterleiten zu können, werden *Events* verwendet. Der *Presenter* reagiert darauf und aktualisiert gegebenenfalls die Daten des Modells.

Die *View* erhält ihre Daten entweder vollständig vom *Presenter*, oder kann diese aus dem Modell auslesen. Ändert sich das Modell, so werden die Änderungen über *Events* sowohl an

den *Presenter* als auch an die *View* geschickt. Im Regelfall bindet sich die *View* per *Databinding* an das Modell und wird so – ohne zusätzlichen Code – automatisch aktualisiert, wenn sich das Modell ändert. Der *Presenter* kann ebenfalls auf Änderungen des Modells reagieren, um komplexere Präsentationslogik auszuführen und dann die *View* zu aktualisieren.

3.3. System-Architektur

In der ersten Iteration der Design-Phase wurde bereits die gesamte Architektur des Horde3D Development Environments entworfen. Der erste Entwurf erwies sich als recht stabil und erforderte in späteren Iterationen nur wenige Modifikationen. Die erstellten Artefakte, ein Designmodell und mehrere Sequenzdiagramme, werden in den folgenden Abschnitten besprochen und die getroffenen Entscheidungen begründet. Das Designklassen-Diagramm ist allerdings zu groß, um es im Anhang dieser Arbeit abzubilden. Daher werden nur relevante Ausschnitte des Modells im Anhang beigelegt; das komplette Modell ist als .pdf-Datei auf der CD-ROM enthalten.

Da UML keine Syntax für .NET *Properties* anbietet, wurde der Stereotyp `property` eingeführt. In der Implementierung wurden dann `Get/SetX`-Methoden mit `property`-Stereotyp als entsprechende *Properties* `X { get; set; }` umgesetzt. Aus Gründen der Übersichtlichkeit wurden im Designmodell für viele öffentliche Attribute keine *Properties* definiert; sie werden später aber als öffentliche *Properties* implementiert, da laut den .NET Richtlinien der Zugriff auf öffentliche Klassenattribute immer durch *Properties* gekapselt sein sollte [15].

3.3.1. Horde3D-Klassen

Die aus dem Konzeptmodell bekannten Horde3D-Klassen wurden in das Designmodell übernommen; alle Klassenattribute, die Assoziationen zwischen den Klassen und die Vererbungshierarchien sind unverändert. Es wurden allerdings mehrere Funktionen ergänzt. So erhält die abstrakte Klasse `SceneNode` die virtuelle Methode `InitializeFromHorde3DState`, die aus dem aktuellen Horde3D-Zustand beispielsweise die Transformationswerte und den Vaterknoten ausliest. Alle abgeleiteten Klassen können diese Methode überschreiben, um ihre eigenen Daten auszulesen. Dabei können sie auf die `SceneNode`-Implementierung zurückgreifen, um die von `SceneNode` geerbten Attribute und Assoziationen zu füllen beziehungsweise aufzubauen.

Ein analoges Vorgehen wurde für die Ressourcen-Hierarchie angewandt. Die abstrakte Klasse `Resource` besitzt ebenfalls eine virtuelle Methode `InitializeFromHorde3DState`, die in ihrer Standardimplementierung die `Resource`-Attribute und -Assoziationen ausliest und von den Subklassen überschrieben werden sollte. Die Klasse `EditableResource` führt zwei weitere abstrakte Methoden ein: `SaveToDisk` und `LoadFromDisk`. Konkrete Subklassen müssen diese Funktionen so implementieren, dass ihre Attribute und Assoziationen in das Horde3D-XML-Format übersetzt werden, beziehungsweise aus diesem ausgelesen und aufgebaut werden. Jedoch können manche Assoziationen nur dynamisch aus dem aktuellen Horde3D-Zustand ermittelt werden, da sie in den XML-Dateien nicht gespeichert sind.

3.3.2. Client-Server-Schnittstelle

Wie bereits in Abschnitt 3.1 ausgeführt, zerfällt das System in einen Client- und einen Serverteil. Da in der Design-Phase bereits die Windows Communication Foundation (WCF)

als Netzwerk-Framework ausgewählt wurde, orientiert sich das Design der Client-Server-Schnittstelle an der Terminologie und an den Fähigkeiten von WCF.

Abbildung C.2 zeigt einen Ausschnitt aus dem Designmodell für die Client-Server-Schnittstelle. Zunächst wurde das Interface `IDebuggerService` spezifiziert, über das der Client auf den Server zugreifen kann. Der Service hat je eine Funktion für jede Systemanforderung: `Suspend` und `Resume`, um die Anwendung anzuhalten und wieder fortzusetzen; `Profile`, um die Profiling-Daten zu generieren; `GetRenderTargetData`, um den Inhalt eines *Render Targets* auszulesen; und `UpdateResource`, um aktualisierte Ressource-Daten zu übermitteln und an Horde3D weiterzureichen.

Ursprünglich lieferten die Funktionen bereits die jeweilig benötigten Daten zurück. In der zweiten Iteration wurde hingegen das `IDebuggerServiceCallback`-Interface eingeführt. Der Server verwendet dieses Interface, um dem Client Daten und Ereignisse zu übermitteln. Das *Callback*-Interface vereinfachte das Einfrieren, Fortsetzen und Profilen der Anwendung aus der Anwendung selbst heraus. Wenn der Benutzer eine bestimmte Taste innerhalb der Anwendung drückt, schickt der Server per *Callback* eine Meldung an den Client, und dieser kann darauf entsprechend reagieren. Ohne *Callbacks* wäre das in Sequenzdiagramm C.1 gezeigte Vorgehen nicht umsetzbar gewesen. Dort werden die *Service*- und *Callback*-Interaktionen beim Anhalten der Anwendung gezeigt. Der Client erhält vom Benutzer den Befehl zum Anhalten der Anwendung und ruft den *Service* auf, welcher wiederum den Server informiert. Alternativ erteilt der Benutzer den Befehl durch einen Tastendruck innerhalb Anwendung direkt an den Server. In beiden Fällen ruft der Server die `OnSuspended`-Methode des *Callbacks* auf. Eine Implementierung des *Callback*-Interfaces würde den Client beispielsweise durch Auslösen eines *Events* darüber informieren, dass die Anwendung angehalten wurde; der Client geht bis zu diesem Zeitpunkt in beiden Fällen immer noch davon aus, dass die Anwendung noch läuft. Kam der `Suspend`-Aufruf direkt vom Server, so kann der Client davon nichts wissen. Kam der Aufruf hingegen vom Client selbst, so wurde er nur an den Server weitergereicht, aber sonst nicht darauf reagiert. Dadurch kann der Client unabhängig vom Ursprung des `Suspend`-Befehls die gleichen Aktionen durchführen.

Erst wenn der Client durch den *Callback* über das Einfrieren der Anwendung informiert wurde, fordert er mit der `RequestDebugData`-Methode des *Services* die aktuellen Szenengraph- und Ressourcendaten an. Der Server schickt dann jede *Scene Node* und jede Ressource einzeln über das *Callback* Interface an den Client, der mit den Objekten entsprechend weiterarbeitet. Beim Fortsetzen oder Profilen der Anwendung wäre das Verfahren analog, es würden nur andere beziehungsweise gar keine Daten per *Callback* an den Client zurückgeschickt werden. Eine Ausnahme bildet hingegen das Auslesen des Inhalts eines *Render Targets*. Da die Anfrage ausschließlich vom Client kommen kann, bringt das *Callback*-Verfahren hier keine Vorteile. Aus diesem Grund liefert die Funktion `GetRenderTargetData` das Ergebnis direkt zurück.

Ein weiterer Vorteil des *Callback*-Interfaces ist die Möglichkeit, generierte Horde3D-Meldungen unverzüglich an den Client schicken zu können. Dies erspart ein kontinuierliches Nachfragen des Clients beim Server, ob neue Nachrichten erstellt wurden. Zusätzlich werden alle *Scene Nodes*, Ressourcen, Horde3D-Meldungen und Profiling-Objekte einzeln übertragen, was bei großen Datenmengen die Reaktionszeit des Clients verbessert. Statt auf die Übertragung des gesamten Datensatzes warten zu müssen, kann der Client nach und nach jedes Objekt, das er bereits erhalten hat, einzeln darstellen.

Problematisch an der Einführung des *Callback*-Interfaces war hingegen, dass es prinzipiell möglich sein könnte, dass *Callbacks* vor dem Verbindungsaufbau aufgerufen werden. Aus

diesem Grund stellt die Implementierung einen Sicherungsmechanismus bereit, der *Callback*-Aufrufe abfängt und erst dann an den Client übermittelt, wenn die Verbindung aufgebaut wurde.

Auf der Clientseite kennt nur die Klasse `Horde3DApplication` eine Instanz der Klasse `DebuggerServiceProxy`, über die der Client mit dem Server kommunizieren kann. Dabei repräsentiert `Horde3DApplication` wie bereits im Konzeptmodell eine konkrete Horde3D-Anwendung. Um eine Anwendung starten zu können, werden gewisse Informationen benötigt, wie der Pfad zur ausführbaren Datei, die *Endpoint*-Konfiguration für WCF, das Arbeitsverzeichnis etc. `Horde3DApplication` kennt diese Daten und die Methode `Start` startet eine neue Serverinstanz. Sobald der Server läuft, dient die Klasse als Fassade für den Service *Proxy*. Zum Beenden des Servers muss die Methode `ShutDown` aufgerufen werden.

3.3.3. Umsetzung des DLL-Replacements und des Profilings

Der gewählte DLL-*Replacement*-Mechanismus macht die Einführung von *Proxy*-Funktionen erforderlich. Wenn die Anwendung eine Horde3D-Funktion aufruft, so wird nicht sofort der Code der originalen Horde3D DLL ausgeführt, sondern eine *Proxy*-Funktion. Diese ruft die ursprüngliche Funktion auf, führt aber auch noch zusätzlichen Code aus. Dieser Code hängt jedoch davon ab, ob der Server gerade die Horde3D-Funktionsaufrufe für die Profiling-Funktion protokolliert.

Um mit den unterschiedlichen Codepfaden zurechtzukommen, wurden die in Abbildung C.4 gezeigte Klassenstruktur entworfen. `Horde3DProxyBase` ist eine abstrakte Basisklasse, die für jede Horde3D-Funktion einen Funktionszeiger sowie eine gleichnamige, abstrakte Methode mit gleichen Rückgabe- und Parametertypen wie die Horde3D-Funktion besitzt. Aus Gründen der Übersichtlichkeit sind im Klassendiagramm C.4 nur der Funktionszeiger und die *Proxy*-Funktion für `Horde3D::getVersionString` eingezeichnet.

Die beiden Klassen `Horde3DNoProfilingProxy` und `Horde3DProfilingProxy` spezialisieren `Horde3DProxyBase`. Alle Funktionszeiger der Basisklasse werden geerbt und durch die ebenfalls geerbte `Initialize`-Methode initialisiert. Die beiden *Proxy*-Klassen überschreiben die abstrakten Horde3D-Methoden: Es wird jeweils die ursprüngliche Horde3D-Funktion sowie die entsprechende Methode der statischen `Horde3DCall`-Klasse aufgerufen und gegebenenfalls die Profiling-Daten protokolliert. `Horde3DCall` besitzt für jede Horde3D-Funktion eine statischen Methode und ein statisches *Event*, welche jeweils die an die Funktion übergebenen Parameter und deren Rückgabewert als Parameter erhalten. Wird eine Horde3D-Methode der `Horde3DCall`-Klasse aufgerufen, so wird das entsprechende spezifische *Event* sowie die beiden generischen *Events* `Before-/AfterFunctionCalled` ausgelöst, wobei letztere jeweils vor – respektive nach – dem Auslösen des speziellen Ereignisses ausgelöst werden. Objekte des Servers können sich an diesen *Events* registrieren und werden somit informiert, wann und mit welchen Parameter- und Rückgabewerten eine Horde3D-Funktion aufgerufen wurde.

Die `Horde3DProxyHandler`-Klasse verwaltet die zwei verschiedenen *Proxies* und hält eine Referenz auf die originale Horde3D DLL, mit der die Funktionszeiger auf die Horde3D-Funktionen ermittelt werden können. Mit `Initialize` werden die beiden *Proxy*-Klassen initialisiert und eine Instanz des `Horde3DNoProfilingProxy` als Wert der `CurrentProxy`-Assoziation gesetzt. Die Methode `EnableProfiling` (de)aktiviert den Profiling-Modus, indem die `CurrentProxy`-Assoziation auf den entsprechenden *Proxy* gesetzt wird.

Die API von Horde3D verwendet allerdings keine Klassen, sondern organisiert die Funktionen im Namensraum `Horde3D`. Die *Replacement*-DLL benötigt daher neben dieser Klassen-

struktur noch *Proxy*-Funktionen für alle Horde3D-Funktionen im Namensraum `Horde3D` und eine globale Instanz der `Horde3DProxyHandler`-Klasse. Die *Proxy*-Funktionen können dann mit der `GetCurrentProxy`-Methode der `Horde3DProxyHandler`-Klasse auf den derzeitigen *Proxy* zugreifen und die entsprechende Horde3D-Methode des *Proxies* aufrufen. Dieser Vorgang wird am Beispiel des Profiling-*Proxies* für die Horde3D-Funktion `getVersionString` in Abbildung C.5 gezeigt, wobei zusätzlich noch `FunctionCall`-Objekte für die Profiling-Funktionalität erzeugt werden.

Das beschriebene Design wurde gewählt, weil es eine klare Trennung zwischen den beiden verschiedenen *Proxies* ermöglicht. Die Trennung wurde nötig, da der Profiling-*Proxy* für jeden Funktionsaufruf einige zusätzliche Schritte durchführen und mehrere Objekte anlegen muss, was einen negativen Einfluss auf die Performance in CPU-limitierten Szenen hatte. Daher sollte der Profiling-Code, wenn er nicht benötigt wird, auch nicht ausgeführt werden.

Es hätte zwei denkbare Alternativen zur umgesetzten Lösung gegeben:

- Man hätte direkt in den *Proxy*-Funktionen im `Horde3D`-Namensraum die Original-Funktionen aufrufen und den zusätzlichen Code ausführen können, wobei man per `if-then-else` den Profiling-Code ein- und ausgeschaltet hätte. Dadurch hätte man sich den Aufruf der `GetCurrentProxy`-Methode gespart – der aber im Regelfall vom Compiler sowieso *inline* kompiliert wird – und den virtuellen Horde3D-Methodenaufruf des *Proxies* durch die Auswertung einer Bedingung ersetzt. Der Performancevorteil der `if-then-else`-Lösung, falls überhaupt messbar, wäre aber im Vergleich zur Ausführungsdauer der eigentlichen Funktionen so gering, dass statt dieser Lösung die im objekt-orientierten Sinne bessere Alternative gewählt wurde.

- Eine weitere Variante betrifft die `Horde3DCall`-Klasse. Diese bietet neben den beiden generischen *Events*, die nur den Namen der aufgerufenen Funktion übergeben, ein spezielles Ereignis mit Ein- und Ausgabewerten für jede Horde3D-Funktion. Dies erfordert viel Code, der durch die Beschränkung auf die generischen *Events* vermieden werden könnte. Dann müsste man allerdings die Ein- und Ausgabewerte der aufgerufenen Funktionen als `Object`-*Array* übergeben. Das führt aber bei jedem Funktionsaufruf zur Erzeugung vieler temporärer Objekte, erfordert *Boxing* und *Unboxing* für Parameter primitiver Datentypen und beim Zugriff auf das Parameter-*Array* eine Überprüfung der *Array*-Grenzen sowie des Parametertyps. Da diese Lösung unperformant, belastend für den *Garbage Collector* und zudem nicht einmal typsicher ist, wurde die aufwändigere, aber elegantere Variante gewählt.

3.3.4. Anhalten der Anwendung

In der ersten Iteration der Design-Phase war nicht klar, wie das Anhalten der Anwendung technisch genau lösbar ist. Es wurde daher das *Strategy Pattern* verwendet, um verschiedene Ansätze möglichst einfach ausprobieren und austauschen zu können. Abbildung C.3 zeigt den relevanten Ausschnitt des Designmodells. Das Interface `ISuspendApplicationStrategy` stellt die beiden Funktionen `Suspend` und `Resume` bereit, nach deren Ausführung die Anwendung entweder angehalten ist oder weiterläuft. Bei der Implementierung zeigte sich, dass insgesamt drei verschiedene Strategien zum Anhalten der Anwendung benötigt werden: `StopTimeSuspendStrategy`, um der Anwendung ein Anhalten der Zeit vortäuschen zu können; `CursorSuspendStrategy`, um der Anwendung die Kontrolle über den Mauszeiger gezielt entziehen und wieder zurückgeben zu können; und `WndProcSuspendStrategy`, um das

Window Procedure der Anwendung ersetzen und damit Benutzereingaben über Maus und Tastatur abfangen zu können. Die Umsetzung dieser Strategien und ihre Probleme werden in Kapitel 4.4 genauer erläutert.

Um diese Objekte nicht einzeln verwalten zu müssen, wurde in einer späteren Iteration die Container-Klasse `SuspendApplicationStrategy` hinzugefügt, die ebenfalls vom *Suspend*-Interface erbt. Einzige Aufgabe dieser Klasse ist es, beim Aufruf ihrer `Suspend`- und `Resume`-Methoden die entsprechenden Methoden aller Objekte im `Strategies`-*Property* aufzurufen. In Kapitel 5.4 findet sich ein Vorschlag, wie man aufbauend auf dieser Instanz des *Strategy Patterns* und der `SuspendApplicationStrategy`-Klasse eine weitere Strategie zum Anhalten der Anwendung integrieren könnte.

3.3.5. Starten und Beenden des Servers

Die Abbildungen C.6 und C.7 zeigen, was beim Starten des Servers nach Aufruf der `Start`-Methode der `Horde3DApplication`-Klasse geschieht. Zunächst muss die originale Horde3D DLL im Anwendungsverzeichnis umbenannt werden. Dann werden die benötigten DLLs des Horde3D Development Environments in das Anwendungsverzeichnis kopiert, einschließlich der modifizierten Horde3D DLL mit den *Proxy*-Funktionen. Anschließend wird die Konfigurationsdatei für den Server generiert, die ebenfalls in das Anwendungsverzeichnis kopiert wird. Dannach wird der Prozess gestartet. Da es in der modifizierten Horde3D DLL eine globale `Horde3DProxyHandler`-Variable gibt, wird beim Starten der Anwendung und nach dem Laden der modifizierten DLL automatisch eine *Proxy Handler*-Instanz erzeugt. Der Konstruktor ruft die `Initialize`-Methode auf, die wiederum zunächst den `Horde3DDebugger`-*Singleton* initialisiert. Clientseitig wird sofort eine Instanz des `DebuggerServiceProxy` erzeugt und mit dem Verbindungsaufbau zum Server begonnen. Da es keine Möglichkeit gibt, vom Server über den Abschluss der Initialisierung informiert zu werden, wird einfach so lange ein Verbindungsaufbau versucht, bis der Serverprozess hochgefahren und initialisiert ist und die Verbindung erfolgreich aufgebaut werden konnte.

Auf der Serverseite werden Instanzen der beiden Horde3D-*Proxies* erzeugt und der Profiling-Modus standardmäßig deaktiviert. Die Initialisierungsroutine des `Horde3DDebugger`-*Singletons* liest zunächst die Konfigurationsdaten aus der *Settings*-Datei ein und überprüft die Horde3D-Version. Der gewählte DLL-*Replacement*-Mechanismus macht es erforderlich, dass die Anwendung und der Server die gleiche Version von Horde3D verwenden. Sollte das nicht der Fall sein, wird sich die Anwendung in vielen Fällen mit einer Windows-Fehlermeldung „Prozedureinsprungspunkt „*ProzedurName*“ wurde in der DLL „Horde3D.dll“ nicht gefunden.“ beenden, bevor der Server initialisiert wird. Es gibt aber auch Fälle, in denen Windows die fehlenden oder zusätzlichen Einsprungspunkte nicht entdeckt. Dann überprüft der Server anhand der Horde3D-Versionskennung beider DLLs, ob unterschiedliche Versionen vorliegen. Sind die Versionen unterschiedlich, wird die Anwendung mit einer Fehlermeldung beendet.

Der `Horde3DDebugger`-*Singleton* erzeugt neben der `DebuggerService`-Instanz noch zwei weitere Objekte: `Horde3DMessagesHandler` und `Horde3DStateWatcher`. Die Aufgabe des `Horde3DMessagesHandler`s ist es, nach jedem Funktionsaufruf eventuell neu generierten Meldungen abzufangen. Wurde eine oder mehrere Meldungen generiert, so werden die Daten ausgelesen, für jede neu generierte Nachricht ein `Horde3DMessage`-Objekt erzeugt und dieses per *Callback* an den Client gesendet.

Zu beachten ist hierbei die Phase der Initialisierung und Deinitialisierung des Servers.

Der `Horde3DMessagesHandler` darf nur nach neuen Meldungen fragen, solange Horde3D korrekt initialisiert ist. Wird beispielsweise `Horde3D::getVersionString` vor dem Aufruf von `Horde3D::init` aufgerufen, würde ohne korrekte Horde3D-Initialisierung bereits nach dem Aufruf von `Horde3D::getVersionString` nach neuen Nachrichten gefragt. Auch nach dem Aufruf von `Horde3D::release` würde nach neuen Meldungen gefragt werden; zu diesem Zeitpunkt ist Horde3D allerdings bereits vollständig deinitialisiert. In beiden Fällen könnte es zu einem Absturz des Programms kommen.

Es ist allerdings kein Problem, wenn bereits Meldungen erzeugt werden, bevor der Client die Verbindung zum Server aufgebaut hat. Alle *Callback*-Aufrufe werden abgefangen und erst ausgeführt, wenn die Verbindung steht.

Die `Horde3DStateWatcher`-Klasse hat derzeit nur die Aufgabe, unerwünschtes *Reverse-Engineering* zu unterbinden. Die Klasse könnte aber in Zukunft erweitert werden, um beispielsweise die Kamera, mit der die Szene gezeichnet wurde, zu protokollieren. Das könnte in Szenen mit mehreren Kameras und mehreren Aufrufen von `Horde3D::render` wichtig sein. Momentan überprüft die Klasse allerdings nur, ob vor dem Aufruf von `Horde3D::render` die Horde3D-Funktion `checkExtension` mit dem Parameter `AllowDebugging` aufgerufen wurde. Ist dies nicht der Fall, so wird beim Versuch die Szene zu zeichnen eine Fehlermeldung angezeigt und der Server beendet.

Das Beenden des Servers ist im wesentlichen eine Umkehr des Startprozesses. Die `ShutDown`-Methode der `Horde3DApplication`-Klasse schließt zunächst die Verbindung zum Server und beendet den Prozess durch Schließen des Hauptfensters. Falls der Prozess nach einem kurzen *Timeout* von wenigen Sekunden nicht beendet wurde, wird er zwangsweise terminiert. Anschießend wird der ursprüngliche Zustand des Anwendungsverzeichnis wiederhergestellt. Die in das Verzeichnis kopierten DLLs des Horde3D Development Environments werden gelöscht und auch die erstellte Konfigurationsdatei wird entfernt. Zum Schluss wird der umbenannten, originalen Horde3D DLL wieder ihr ursprünglicher Name gegeben.

3.3.6. Aktualisieren von Ressourcen-Daten

Das Sequenzdiagramm C.8 stellt das Vorgehen für die Aktualisierung von geänderten Ressourcen dar. Der Vorgang wird über die `Horde3DApplication`-Klasse angestoßen, da nur diese Klasse die `UpdateResource`-Operation des Servers aufrufen kann. Der Server führt die interne `UpdateHorde3D`-Methode der übergebenen `EditableResource` aus. Diese Methode ist virtuell und kann somit von den Subklassen überschrieben werden. Das Sequenzdiagramm zeigt die Funktionsweise der Standardimplementierung. Zunächst wird überprüft, ob die Ressource, die Horde3D gerade unter dem gleichen `ResHandle` kennt, den gleichen Typ und gleichen Namen hat. Implizit wird damit auch überprüft, ob überhaupt noch eine Ressource mit diesem `ResHandle` existiert. Dies ist notwendig, weil es das Horde3D Development Environment erlaubt, die Anwendung nach dem Einfrieren der Szene und nach dem Auslesen der bekannten Ressourcen weiter auszuführen. Der Benutzer könnte nun eine Ressource bearbeitet haben und diese an die `UpdateResource`-Operation des Servers übergeben. Zwischenzeitlich könnte die Anwendung die Ressource aber gelöscht und ihren `ResHandle` neu vergeben haben. Durch die Überprüfung des Ressourcennamens und -typs soll verhindert werden, dass die übergebene Ressource eine eventuell neu hinzugefügte Ressource überschreibt. Ansonsten würde die derzeit geladene Ressource durch eine eventuell völlig andere ersetzt. Bei unterschiedlichen Typen würde Horde3D eine Fehlermeldung erzeugen, da die übergebenen Ressourcen-Daten dann nicht korrekt interpretiert werden können.

Ist die Überprüfung jedoch erfolgreich, werden die derzeit geladenen Ressourcen-Daten gelöscht. Die Ressource selbst bleibt aber erhalten, insbesondere ändert sich ihr `ResHandle` nicht. Nun können die Daten aus der übergebenen `EditableResource` in ein `byte`-*Array* kopiert und an Horde3D übergeben werden. Anschließend werden alle zur Zeit nicht geladenen aber bekannten Ressourcen geladen. Das ist erforderlich, weil die aktualisierte Ressource auf neue, noch nicht geladene Ressourcen verweisen könnte. Bevor die Szene mit der aktualisierten Ressource gezeichnet werden kann, müssen diese Abhängigkeiten geladen werden.

Wie bereits erwähnt ist die `UpdateHorde3D`-Methode der `EditableResource`-Klasse virtuell. Werden nämlich Shader- oder Material-Ressourcen aktualisiert, müssen noch einige zusätzliche Schritte ausgeführt werden. Horde3D erlaubt das Klonen von Materials, das heißt, die Material-Ressource wird kopiert und der Kopie ein neuer `ResHandle` zugewiesen. Dadurch können zum Beispiel die *Uniform*-Parameter des Materials für verschiedene Objekte unterschiedlich gesetzt werden. Allerdings werden die Kopien nicht aktualisiert, wenn das Original-Material aktualisiert wird. Übergibt man also eine `MaterialResource` an `UpdateResource`, so müssen alle Kopien des Materials gesucht und aktualisiert werden. Die Kopien sind an ihrem Namen erkennbar: hieß das Original-Material `Material1.material.xml`, dann heißen die Klone `Material1.material.xml|X`, wobei X der `ResHandle` der Kopie ist.

Wird eine `ShaderResource` an `UpdateResource` übergeben, müssen ebenfalls noch weitere Ressourcen aktualisiert werden. Der GLSL-Code des Shaders wird von Horde3D in zusätzlichen Code-Ressourcen gespeichert. Wird ein Shader aktualisiert, werden allerdings die Code-Ressourcen nicht automatisch neu geladen. Dies muss manuell geschehen. Dazu werden die abhängigen Code-Ressourcen über den Namen ermittelt, der als Präfix den Namen der zugehörigen Shader-Ressource enthält. Allerdings kann diese Sonderbehandlung der Shader-Ressourcen für zukünftigen Horde3D-Versionen entfallen. Alle Versionen nach Horde3D 1.0.0 Beta 3 aktualisieren automatisch auch alle abhängigen Code-Ressourcen.

3.4. Anbindung der Benutzeroberfläche an das System

Die Anbindung der grafischen Benutzeroberfläche an den Anwendungscode wurde bislang nicht weiter betrachtet, ist jedoch ein komplexes Problem. Für das Horde3D Development Environment wurde eine eigene Bibliothek entwickelt, die die GUI-Anbindung an das System kapselt und einige Grundfunktionalitäten bereitstellt. Dabei standen die Wiederverwendbarkeit der Bibliothek sowie eine saubere Trennung zwischen GUI- und Anwendungscode im Vordergrund.

3.4.1. Alternativen zur Eigenentwicklung eines GUI-Frameworks

Bevor mit der Entwicklung eines eigenen GUI-Frameworks begonnen wurde, wurden zwei alternative Ansätze in Betracht gezogen: Die Integration des Clients in Visual Studio oder in SharpDevelop[1] als Plugin. Beide Tools haben eine ausgereifte, mächtige und bewährte Plugin-Architektur und bieten viele Schnittstellen zu Subsystemen – wie einen Texteditor mit *Syntaxhighlighting*, eine Konsole für Textausgabe, ein *Undo/Redo*-Framework, ein Framework zum automatischen Speichern geänderter Dateien, etc. –, die das Horde3D Development Environment ebenfalls benötigt. Es bestand also ein großes Potential, bereits vorhandene und

[1] `http://www.icsharpcode.net/OpenSource/SD`

erwiesenermaßen gut funktionierende Lösungen wiederzuverwenden. Dennoch wurde auf diesen Vorteil verzichtet, da die Nachteile überwogen. Visual Studio ist ein extrem komplexes und im Kern auch sehr altes System, das zu weiten Teilen noch in C++ und COM implementiert ist. Plugins können zwar in C#/.NET entwickelt werden, allerdings sind die Interfaces, die Visual Studio bereitstellt, aufgrund ihres Alters überladen, schwer verständlich und nicht mehr zeitgemäß. Außerdem muss ein Plugin nach jeder Änderung neu integriert werden, was die Entwicklung verzögert und ein schnelles Debuggen unmöglich macht. Ein weiteres Problem ist, dass nicht jeder potenzielle Benutzer des Horde3D Development Environments eine Visual Studio Lizenz besitzt; die kostenlosen Express-Versionen unterstützen leider keine Plugins. Die eigenständig lauffähige Visual Studio Shell würde diese Probleme zwar lösen, wäre aber aufgrund ihrer Größe (ca. 150 MByte) und Komplexität eine sehr belastende Abhängigkeit für das Horde3D Development Environment.

SharpDevelop hingegen wurde von Anfang an komplett in C# entwickelt und ist Open Source. Die Einbindung von Plugins gestaltet sich angenehmer als unter Visual Studio und es gibt moderne objekt-orientierte APIs für den Zugriff auf die Standardkomponenten der Entwicklungsumgebung. Leider ist die Entwicklung von Plugins für SharpDevelop sehr schlecht dokumentiert. Es gibt zwar einführende Erläuterungen, aber eine prototypische Implementierung des Horde3D Development Environments zeigte schnell, dass bei komplexeren Fragen und Problemen oft keine Hilfestellung in der Dokumentation, im Internet oder im inzwischen veralteten Buch [16] existiert. Da die IDE Open Source ist, konnte die Antwort zwar immer durch Durchsehen des Quellcodes gefunden werden, dies nahm aber sehr viel Zeit in Anspruch. Visual Studio hingegen ist teilweise deutlich besser dokumentiert. Aber auch hier gibt es noch einige Defizite, insbesondere bei der Erweiterung der Standard-Projektverwaltung mit dem *Visual Studio Managed Package Framework for Projects*[2].

3.4.2. Architektur des GUI-Frameworks

Um eine klare Trennung zwischen GUI- und Anwendungscode zu erreichen, ist das GUI-Framework eine Instanz des *Model-View-Presenter Patterns*. Es wurde außerdem von Anfang an auf die Entwicklung einer Visual Studio-ähnlichen Oberfläche ausgelegt, kann aber prinzipiell auch für andere UI-Designs verwendet werden.

Abbildung C.9 zeigt das Designklassen-Diagramm des GUI-Frameworks. Die Verwendung des *Model-View-Presenter Patterns* wird durch die abstrakte Klasse `Presenter` und das Interface `IView` deutlich. Im GUI-Framework selbst kommen keine Modelle vor; diese sind anwendungsspezifisch und werden von den Applikationen, die das Framework verwenden, bereitgestellt.

Es gibt insgesamt vier Klassen, die das `IView`-Interface implementieren. Diese Klassen ergeben sich aus dem Design moderner GUI-Frameworks wie Windows Forms und der Windows Presentation Foundation. Dort gibt es zum einen `Form`-Klassen, welche ein eigenständiges Fenster repräsentieren. Diesen Fenstern können mehrere Instanzen von `UserControl`-Klassen untergeordnet sein, beispielsweise für Eingabefelder oder Schaltknöpfe. Um das Visual Studio-ähnliche Design zu realisieren, gibt es noch eine weitere spezielle Klasse, `DockContent`, deren Instanzen ebenfalls einer `Form` untergeordnet sein müssen. Dabei kann eine `DockContent`-Instanz sowohl ein eventuell verstecktes *Dock Pane* an der Seite der Oberfläche, ein *Floating Window* oder ein Dokument sein. Für all diese Klassen wird eine `IView`-Implementierung

[2] `http://www.codeplex.com/mpfproj`

benötigt. `UserControlView` erbt von `UserControl` und `FormView` ist von `Form` abgeleitet, wobei hier zwischen modalen und nicht-modalen Fenstern unterschieden wird. `DockView` erbt von `DockContent` und fügt zwei Attribute hinzu, um den standardmäßigen `DockState` und die Menge der erlaubten `DockState`s festzulegen. `DocumentView` ist eine Subklasse von `DockView`, wobei hier der standardmäßige `DockState` und die Menge der erlaubten `DockState`s fest auf `Document` beschränkt sind.

Die abstrakte Klasse `Presenter` ist die Basisklasse für alle *Presenter* im System. Instanzen von `Presenter` werden über einen Namen identifiziert; für einen gegebenen Namen kann es immer nur eine `Presenter`-Instanz geben. `Presenter` hat eine private Assoziation zu einem `IView`, auf die abgeleitete Klassen nur über die Funktion `UpdateView` zugreifen können. Damit wird zum einen die Kopplung zur *View* verringert und zum anderen müssen in dieser Funktion noch einige implementierungsspezifische Details (Zugriff über korrekten Thread, kein Zugriff auf bereits gelöschte *Views*, etc.) geregelt werden. Zu beachten ist, dass nur die abstrakte `Presenter`-Klasse ihre zugehörige *View* kennt; insbesondere kennen die `IView`-Instanzen nicht ihre zugehörige `Presenter`-Instanzen. Dies wird durch das `Request`-Ereignis, das alle `IView`-Implementierungen besitzen, ermöglicht. Ein *Presenter* registriert sich an den `Request`-Ereignissen seiner *View* und kann darüber über Eingaben und Aktionen des Benutzers informiert werden.

Die GUI-*Controls* bilden eine Hierarchie aus `Form`-, `DockContent`- und `UserControl`-Instanzen. Diese Hierarchie müssen auch die *Presenter* widerspiegeln. Aus diesem Grund kann ein `Presenter` einen `Parent` und beliebig viele `Children` haben, die über die Funktionen `Add-/RemoveChildPresenter` hinzugefügt und wieder entfernt werden können. `Presenter` hat außerdem abstrakte Methoden für die Initialisierung und Deinitialisierung. Abgeleitete `Presenter`-Klassen müssen diese Methoden mit einer speziellen Implementierung überschreiben. Die *Presenter*- und *View*-Hierarchien sind eine Instanz des *Composite Patterns*.

Da eine *Undo/Redo*-Funktionalität benötigt wird, werden vom Benutzer ausgeführte Aktionen gemäß des *Command Patterns* durch Objekte gekapselt, die von der abstrakten Klasse `Command` erben müssen. Ein `Command` hat Funktionen zum Ausführen der Aktion (`Execute`), sowie zum Rückgängigmachen (`Undo`) und Wiederholen (`Redo`); wobei `Redo` standardmäßig einfach noch einmal `Execute` aufruft und `Execute` und `Undo` abstrakt sind. Es ist auch möglich, die *Undo/Redo*-Funktionalität für einen `Command` zu deaktivieren, indem die `CanUndo`- und `CanRedo`-Attribute auf `false` gesetzt werden. Instanzen von `Command` werden immer von einem `Presenter` erzeugt und jeder `Command` kennt seinen Erzeuger. Über diese Assoziation können alle `Command`-Objekte eines `Presenter`s beim Entfernen des `Presenter`s aus der GUI gelöscht werden, wodurch die Aktionen dieser `Command`s nicht mehr rückgängig gemacht werden können.

`Command`-Objekte werden vom `CommandDispatcher` verwaltet. Dessen `HandleCommand`-Methode ruft die `Execute`-Methode des übergebenen `Command`s auf und verwaltet intern eine *Undo-/Redo*-Liste von bereits ausgeführten Aktionen. Der `CommandDispatcher` kann über die Funktionen `Undo` und `Redo` angewiesen werden, eine Aktion rückgängig zu machen oder zu wiederholen. Dies ist allerdings nur möglich, wenn das `CanUndo`- respektive das `CanRedo`-Attribut des `CommandDispatcher`s auf `true` gesetzt ist. Ändert sich der Wert eines dieser Attribute wird ein entsprechendes Ereignis ausgelöst, ebenso nach dem Rückgängigmachen und Wiederholen einer Aktion.

Das Horde3D Development Environment stellt editierbare Ressourcen als `DocumentView`-Instanzen dar. Nach dem Ändern einer Ressource sollen die Änderungen gespeichert werden können. Außerdem soll das Framework einen visuellen Hinweis geben, falls ein Do-

kument geändert wurde, aber die Änderungen noch nicht gespeichert sind. Schließt der Benutzer ein geändertes und noch nicht gespeichertes Dokument, so soll das Framework den Benutzer fragen, ob die Änderungen gespeichert oder verworfen werden sollen. Diese Aufgaben übernimmt die `SaveableContentManager`-Klasse, an der `Presenter`, die das `ISaveableContentPresenter`-Interface implementieren, registriert und wieder entfernt werden können. Registrierte `ISaveableContentPresenter`-Instanzen können einzeln oder alle auf einmal gespeichert werden. Wird ein geänderter, aber noch nicht gespeicherter `Presenter` geschlossen, so wird der Benutzer gebeten, die Änderungen zu speichern oder zu verwerfen.

Das `ISaveableContentPresenter`-Interface stellt Methoden zum Laden und Speichern des zu persistierenden Objekts bereit. Auch der aktuelle `SaveState` wird dort verwaltet. Das Interface wird von der abstrakten Klasse `SaveableContentPresenter` implementiert, die die normale `Presenter`-Klasse erweitert.

Die `Shell`-Klasse ist schließlich das Hauptelement des Frameworks. In ihr laufen alle Vorgänge zusammen. `CommandDispatcher` und `SaveableContentManager` sind mit einer 1:1-Assoziation zur `Shell` verbunden und hätten somit auch direkt in die `Shell`-Klasse integriert werden können. Dies hätte jedoch zu einer niedrigen Kohäsion und damit zu einer Überladung der `Shell`-Klasse geführt, was durch die Verteilung der Zuständigkeiten vermieden wurde. Da der `SaveableContentManager` nur der `Shell` bekannt ist, ist die `Shell` hier eine Instanz des *Facade Patterns*. Auch das *Singleton Pattern* wird von der `Shell` verwendet: Mit dem `Current`-Attribut kann global auf die `Shell`-Instanz zugegriffen werden, wobei es nur eine `Shell`-Instanz pro *AppDomain* geben darf. Dies ist konsistent mit dem Vorgehen der `Application`-Klasse der Windows Presentation Foundation.

Die Aufgabe der `Shell`-Klasse ist neben der Initialisierung der gesamten Oberfläche auch die Verwaltung aller `Presenter`-Instanzen. `Presenter` können hinzugefügt, wieder entfernt und Dokumente einzeln oder gemeinsam gespeichert oder geschlossen werden. Auch das Layout der *Dock Panes* wird von der `Shell`-Klasse automatisch persistiert und beim Starten der Anwendung wiederhergestellt.

3.4.3. Identifizierung der benötigten Modelle

Modelle tauchen in Abbildung C.9 nicht auf, da das GUI-Framework kein Interface und keine Zugriffswege für Modelle vorgibt. Dies ermöglicht einem *Presenter*-*View*-Paar die Verwendung mehrere Modelle, wovon die Implementierung des Horde3D Development Environments auch Gebrauch macht. Insgesamt werden zur Umsetzung der Systemanforderungen sechs Modelle benötigt, die im Designmodell mit dem Stereotyp `model` gekennzeichnet sind.

- `Horde3DApplication`: Die `Horde3DApplication`-Klasse repräsentiert die Horde3D-Anwendung, in der gerade der Server läuft oder die gestartet werden kann. Sie ist die einzige Zugriffsmöglichkeit auf Operationen des Servers.

- `CallbackHandler`: Der `CallbackHandler` nimmt die *Callbacks* des Servers entgegen und generiert für jede Nachricht ein entsprechendes *Event*. Die *Presenter* können sich an der `CallbackHandler`-Instanz an verschiedenen Ereignissen registrieren, um über Server-Nachrichten informiert zu werden.

- `MessagesDispatcher`: Die Aufgabe des `MessagesDispatcher`-Modells ist die Verwaltung aller erzeugten Nachrichten. Die Nachrichten können zum einen aus der Horde3D-Anwendung stammen oder vom Client generierte Fehler- oder Debugmeldungen sein.

Die Nachrichten werden nach ihrer Herkunft (Client oder Server) kategorisiert und nach Wichtigkeit (Debug, Information, Warnung oder Fehler) unterschieden. Der *Dispatcher* unterstützt auch das Entfernen aller Nachrichten einer bestimmten Kategorie. Nachdem Meldungen entfernt oder hinzugefügt wurden, wird ein spezielles Ereignis ausgelöst, damit die interessierten *Presenter* und *Views* entsprechend reagieren können.

- `SceneGraph` und `ResourceGraph`: Diese beiden Modelle dienen zum Verwalten aller bekannten *Scene Nodes* und Ressourcen. Werden neue Objekte eingefügt oder Objekte entfernt, wird ein `GraphChanged`-*Event* ausgelöst, um die Änderungen bekannt zu machen.
- `ProfilingDataManager`: Dieses Modell funktioniert ähnlich der beiden zuvor genannten Modelle und verwaltet die Profiling-Daten. Das `ProfilingDataChanged`-*Event* wird ausgelöst, wenn Profiling-Daten empfangen oder gelöscht werden.

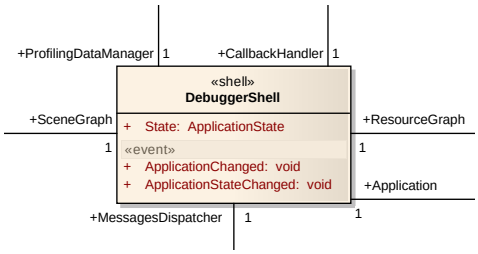

Abbildung 3.4.: Die Shell des Horde3D Development Environments im Designmodell

Alle Modelle werden von der `Shell`-Klasse verwaltet, wodurch alle *Presenter* Zugriff auf alle Modelle erhalten. Die `Shell`-Klasse überwacht außerdem den aktuellen Zustand des Servers, da einige *Presenter* und *Views* bei Zustandsänderungen spezielle Aktionen durchführen müssen. Die möglichen Zustände sind `Unloaded`, wenn derzeit kein `Horde3DApplication`-Modell geladen ist und somit der Server nicht gestartet werden kann; `Stopped`, wenn der Server zur Zeit nicht läuft aber gestartet werden kann; `Suspended`, wenn der Server gerade läuft und die Szene eingefroren ist; und `Running` sonst. Die `Shell`-Klasse löst ein *Event* aus, sobald sich der Zustand des Servers ändert.

3.5. Konklusion

In der Design-Phase wurden viele generische Konzepte verwendet, um flexibel auf mögliche Änderungen von Horde3D oder eventuell auftauchende Probleme in der Implementierungs-Phase reagieren zu können. Aufgrund der detaillierten Analyse der Anforderungen und dem Aufbau von Horde3D konnte ein Systemdesign erstellt werden, das bei der Implementierung nur wenige Probleme verursachte. Die notwendigen Änderungen am Design beschränkten sich jedoch immer auf einzelne Klassen; die grundlegende Architektur blieb bestehen. Eine größere Änderung ergab sich lediglich durch die Einführung der Client-Server-*Callbacks*.

Im Rahmen dieser Bachelorarbeit wurde aber nicht das komplette Design implementiert. So gibt es gerade bei den Horde3D-Klassen einige Attribute und Assoziationen, die zwar im Designmodell vorhanden sind, für die Umsetzung der Systemanforderungen aber nicht erforderlich waren. Sie wurden dennoch in das Konzept- und Designmodell aufgenommen,

um die Modelle zu vervollständigen. Im Code können diese bei Bedarf einfach hinzugefügt werden.

Bei der Entwicklung des Horde3D Development Environments wurde auch deutlich, dass der verwendete DLL-*Replacement*-Mechanismus sowie die Einführung der `Horde3DCall`-Klasse richtige Entscheidungen waren. Insbesondere die beiden Klassen `Horde3DMessagesHandler` und `Horde3DStateWatcher` zeigten, warum das Auslösen von generischen als auch spezifischen Ereignissen mit den genauen Aufrufsparametern und Rückgabewerten nach einem Horde3D-Funktionsaufruf sinnvoll ist. Diese Informationen sind für unterschiedlichste Aktionen nützlich; so wurde beim Entwurf dieses Verfahrens nicht an die Verwendung eines *Reverse-Engineering*-Schutzes gedacht. Sowohl die `Horde3DStateWatcher`-Klasse als auch die Anforderung vor dem Schutz vor unerwünschtem *Reverse-Engineering* wurden erst in einer späteren Iteration ins Horde3D Development Environment aufgenommen und fügten sich nahtlos in das Systemdesign ein.

Die Entwicklung des GUI-Frameworks war zeitaufwändig und hätte vermieden werden können, wenn das Horde3D Development Environment als Plugin für Visual Studio oder SharpDevelop entwickelt worden wäre. Aufgrund verschiedener Unzulänglichkeiten der Plugin-Infrastruktur der IDEs hat sich die Eigenentwicklung schließlich doch als die bessere Lösung herausgestellt, da sich während der Implementierung des Systems die Stärken des Frameworks zeigten und ein zügige und unkomplizierte Umsetzung des Designs ermöglichten. So ist der GUI- und Anwendungscode stets klar voneinander abgegrenzt und es ist einfach, neue Features durch Implementieren weiterer `Presenter`- und `View`-Klassen hinzuzufügen. Die Wiederverwendbarkeit und Erweiterbarkeit des Frameworks konnte bereits im Rahmen der Bachelorarbeit überprüft werden. So wurde zu einem späteren Zeitpunkt eine weitere `DockView`-Subklasse, `WpfDockView`, hinzugefügt, mit der Windows Presentation Foundation `UserControl`s in Windows Forms `DockView`s dargestellt werden können. Der Einsatz des Frameworks bei der Entwicklung des Code Generators, siehe Abschnitt 4.2, bestätigte die Wiederverwendbarkeit der Bibliothek.

4. Phase III: Implementierung

Nach der ersten Iteration der Design-Phase war die grundsätzliche Architektur und Funktionsweise des Horde3D Development Environments festgelegt. In der Implementierungs-Phase wurde das Design in C++/CLI und C# mit Visual Studio 2008 Service Pack 1 umgesetzt. Es wurde eine Vielzahl an *Third Party* Bibliotheken und Frameworks eingesetzt, um den Programmierungsaufwand möglichst gering zu halten.

Zunächst wurde mit der Umsetzung des DLL-*Replacement*-Mechanismus begonnen, dann wurde das GUI-Framework implementiert. Anschließend wurden die einzelnen Systemanforderungen inkrementell programmiert. Durch das inkrementelle Vorgehen konnten schnell die ersten Features umgesetzt und getestet und dadurch auch früh mögliche Designfehler entdeckt werden. Die Einführung der *Callbacks* wurde beispielsweise während der Umsetzung des Szenengraph-Explorers vorgenommen, bevor die erste Zeile für die Ressourcen oder die Profiling-Daten geschrieben worden war.

Da die wichtigsten Entscheidungen bereits während der Analyse- und Design-Phasen getroffen worden waren, konnte das Design zügig umgesetzt werden. In den folgenden Abschnitten werden nun wichtige technische Details erläutert, die in den vorherigen Phasen noch keine Rolle spielten. Dabei wird allerdings weitestgehend auf das Abdrucken von Code verzichtet und nur die verwendeten Konzepte erläutert. Der komplette Source Code befindet sich auf der beiliegenden CD-ROM.

4.1. Verwendete Technologien und Frameworks

Zur Umsetzung des Horde3D Development Environments wurde auf kommerzielle und frei verfügbare Open Source Bibliotheken und Frameworks zurückgegriffen, um gewisse Standardlösungen nicht selbst entwickeln zu müssen. Insbesondere basiert das System auf dem .NET Framework von Microsoft[1].

4.1.1. .NET Framework

2002 veröffentliche Microsoft das .NET Framework, welches seither stark erweitert und verbessert wurde. Ähnlich wie bei Java wird der Code zunächst in eine Zwischensprache, die *Common Intermediate Language*, übersetzt und dann zur Laufzeit von der *Common Language Runtime* (CLR) mit einem *Just-in-Time-Compiler* in Maschineninstruktionen umgewandelt. Das .NET Framework spezifiziert ein vereinheitlichtes Typsystem, das *Common Type System*, bietet die Möglichkeit für *Reflection* und Code Generierung zur Laufzeit und verwendet einen *Garbage Collector* zum Aufräumen und Verwalten des Speichers. Inzwischen gibt es viele Sprachen, die in die *Common Intermediate Language* übersetzt werden können und somit Zugriff auf alle Klassen des .NET Frameworks besitzen. Die wichtigsten sind C#, Visual Basic .NET und C++/CLI [17].

[1] `http://www.microsoft.com/NET`

Das Framework und C# sind ECMA- und ISO-standardisiert. Es gibt verschiedene Implementierungen des Frameworks, wobei Microsoft offiziell nur Windows, mobile Geräte wie Handys oder Zune und die XBOX 360 unterstützt. Für Linux- und Mac-Systeme ist Mono[2] weit verbreitet.

Das Horde3D Development Environment ist in C# 3.0 und C++/CLI implementiert. Es ist jedoch nur unter Windows lauffähig, da sowohl der Client als auch der Server native, Windows-spezifische Funktionen und Bibliotheken verwenden. Dieser Nachteil wurde jedoch bewusst in Kauf genommen, da die Verwendung des .NET Frameworks und der plattform-abhängigen Bibliotheken eine einfachere und schnellere Implementierung ermöglichte.

Neben der *Base Class Library*, die nur grundlegende Klassen wie Ein-/Ausgabestreams, Dateizugriffe, Strings, Listen, Hash-Tabellen und Warteschlangen anbietet, verwendet das Horde3D Development Environment vor allem auch Windows Forms (WinForms) für die Umsetzung der GUI, die Windows Communication Foundation (WCF) für die Interprozesskommunikation und Language Integrated Queries (Linq) für Filterung und Sortierung von Listen. Windows Forms basiert auf der mit Windows 95 eingeführten Win32-API und wurde mit dem Update auf .NET 3.0 durch die modernere Windows Presentation Foundation (WPF) abgelöst. Dennoch wird Windows Forms aufgrund seiner Einfachheit und hohen Effizienz – sowohl bei der Entwicklung als auch beim Laufzeitverhalten – gerne noch für Anwendungen eingesetzt, die keine speziellen Anforderungen, wie *Themes* oder Animationen, an das grafische Design stellen. Da die GUI des Horde3D Development Environments sich am Standard-Design von Windows orientiert, wurde WinForms als UI-Framework gewählt. Obwohl Mono vollständig kompatibel zu Windows Forms ist [18], ist der Client nur unter Windows lauffähig. Zur Umsetzung der GUI wurden zwei Bibliotheken verwendet, Weifen Luo Winforms Docking und das Krypton Toolkit, die native Win32-Funktionen aufrufen und somit von Mono unter Linux und Mac-OS nicht unterstützt werden können.

Die Windows Communication Foundation ist Microsofts Ansatz, eine einheitliche API für die plattformunabhängige, service-orientierte Netzwerkprogrammierung mit dem .NET Framework zu schaffen; insbesondere ersetzt WCF TCP-Sockets und das .NET Remoting. WCF kann sehr flexibel an die Anforderungen des Systems angepasst werden und ist somit sowohl für Webservices als auch für die Interprozesskommunikation geeignet. Da Client und Server des Horde3D Development Environments auf dem gleichen Rechner laufen, werden *Named Pipes* zur Kommunikation eingesetzt. Die Verwendung von *Named Pipes* mit WCF garantiert eine performante, verlässliche und geordnete Übertragung der Nachrichten in einem Binärformat. *Callback* Kontrakte werden ebenfalls unterstützt [19].

Linq wird durch eine Reihe neuer Klassen im .NET-Framework realisiert, deren Verwendung durch neue C# 3.0-Sprachfeatures vereinfacht wird [20]. Es erlaubt auf eine deklarative Weise, Listen, XML-Dateien und Datenbanktabellen zu durchsuchen, zu sortieren und zu gruppieren. Diese Operationen sind in herkömmlichen objekt-orientierten oder imperativen Sprachen oft mit viel Code verbunden; durch Linq wird der Code kürzer und präziser, wie man es von Datenbankanfragen mit SQL gewöhnt ist. Zusammen mit Linq verwendet das Horde3D Development Environment auch weitere neue Sprachfeatures von C# 3.0, wie Extension-Methoden und Lambda-Funktionen. Mit Extension-Methoden kann man beliebige Klassen um Funktionen erweitern, ohne den Source Code der Klasse zu ändern oder von ihr zu erben. Lambda-Funktionen bieten eine kurze Syntax zum Definieren anonymer Methoden. Das folgende Beispiel verdeutlicht diese Konzepte:

[2] `http://www.mono-project.com/Main_Page`

```
// Diese Extension-Methode ruft auf jedem Element der Liste "source"
// die Funktion "func" auf.
public static void Foreach<T>(this List<T> source, Action<T> func)
{
  if (source == null)
    return;

  foreach (var item in source)
    func(item);
}

// Mit diesem Linq-Query werden alle Ressourcen, deren ResHandle
// größer 10 ist, nach ihrem Namen sortiert ausgewählt.
var resources = from r in ResourceManager.Resources
                where r.ResHandle > 10
                order by r.Name
                select r;

// Anwendung der Extension-Methode "Foreach". Das Argument für
// "source" ist "resources", das Argument für "func" ist eine
// Lambda-Funktion.
resources.Foreach(r => r.Reload());
```

4.1.2. Verwendete Third Party Bibliotheken

Die Implementierung des Horde3D Development Environments verwendet die folgenden Bibliotheken, die alle kostenlos eingesetzt werden dürfen. Sofern bekannt wurden die Lizenzen der Bibliotheken ebenfalls aufgelistet.

- **Weifen Luo Winforms Docking** (*MIT License*)[3]: Die Bibliothek wird zur Umsetzung des Visual Studio *User Interfaces* aus Abbildung 3.2 verwendet. Alle für die Umsetzung der GUI benötigten Features sind vorhanden; es wurden lediglich einige kosmetische Verbesserungen durchgeführt. Auch SharpDevelop verwendet diese Bibliothek.

- **Superlist** (*Microsoft Permissive License v1.1*)[4]: Dieses *User Control* erweitert die `ListView`- und `Grid`-Klassen aus Windows Forms um einige neue Features und bietet eine einfachere Schnittstelle als die Standardklassen.

- **Krypton Toolkit** (*kommerziell/kostenlos)*[5]: Das Toolkit ist Teil der kommerziellen Krypton Suite, darf aber frei verwendet werden. Es enthält Erweiterungen für viele WinForms-Standard-*Controls*, um eine grafisch ansprechendere Oberfläche zu erstellen.

- **Detours Express** (*kommerziell/kostenlos*)[6]: Mit dieser Bibliothek von Microsoft Research können Aufrufe nativer Funktionen auf andere Funktionen umgeleitet werden. Allerdings werden in der Express-Version nur 32 Bit Prozesse unterstützt.

[3] `http://sourceforge.net/projects/dockpanelsuite`
[4] `http://www.codeplex.com/Superlist`
[5] `http://www.componentfactory.com`
[6] `http://research.microsoft.com/en-us/projects/detours`

- **ExceptionReporter** (*GNU Library General Public License*)[7]: Tritt während der Ausführung des Clients eine unerwartete Ausnahme auf, so zeigt der Exception Reporter einen Dialog mit einer ausführlichen Fehlerbeschreibung an. Der Benutzer kann den Entwickler dann per Email über den aufgetretenen Fehler informieren.
- **SharpDevelop TextEditor** (*GNU Library General Public License*)[8]: Der Text Editor der SharpDevelop IDE unterstützt *Syntaxhighlighting* für C#, C++/CLI und XML.
- **ScalablePictureBox** (*Open Source/unbekannt*)[9]: Das `ScalablePictureBox`-*Control* erlaubt es, ein dargestelltes Bild mit einem Klick zu verkleinern und wieder zu vergrößern und den dargestellten Teil des Bildes zu verschieben.
- **Microsoft Chart Controls** (*proprietär/unbekannt*)[10]: Das .NET Framework enthält keine vorgefertigten APIs zum Darstellen von Diagrammen. Die Chart Controls Bibliothek von Microsoft füllt diese Lücke und ermöglicht eine grafische Auswertung der Profiling-Daten im Horde3D Development Environment.
- **TGAReader** (*The Code Project Open License*)[11]: Die `Image`-Klasse des .NET Frameworks kann keine Bilder im .tga-Format laden. Die `TargaImage`-Klasse fügt die .tga-Unterstützung hinzu.
- **Horde3D** (*GNU Lesser General Public License*)[12]: Das Horde3D Development Environment verwendet die C- und C#-Bindings von Horde3D. Allerdings ist der Client nicht direkt von der DLL, sondern nur vom Aufbau und der Funktionsweise der Engine abhängig. Der Server hingegen ruft direkt Funktionen von Horde3D auf.

4.2. Implementierung des DLL-Replacement-Mechanismus

Die Klassenstruktur in Abbildung C.4 zusammen mit der Klasse `Horde3DCall` bilden die Grundlage des DLL-*Replacement*-Mechanismus. Die *Proxy*-Klassen und der *Handler* wurden in C++/CLI geschrieben, `Horde3DCall` hingegen in C#. Daher muss die DLL, die die originale Horde3D DLL der Anwendung ersetzt, sowohl nativen Code als auch *managed* Code enthalten. Die *Proxy*-Funktionen im `Horde3D`-Namensraum sind nativer Code innerhalb der DLL und können daher von der unmodifizierten Anwendung ohne Neukompilierung – also völlig transparent – aufgerufen werden. Diese nativen Funktionen wiederum rufen den *Handler* auf, der *managed* ist und somit vollen Zugriff auf alle öffentlichen C#-Klassen hat.

Problematisch ist die Menge des Codes, der für die Umsetzung des Designs erforderlich ist. Horde3D 1.0.0 Beta 3 hat 78 öffentliche Funktionen, für die jeweils folgender Code notwendig ist: Ein *Delegate*-Typ, ein Ereignis und eine Funktion in `Horde3DCall`; ein Funktionszeiger und dessen Initialisierung sowie in eine virtuelle Funktion in `Horde3DProxyBase`; eine Implementierung der virtuellen Funktion in `HordeProfilingProxy` und `Horde3DNoProfilingProxy`; und eine native *Proxy*-Funktion. In der derzeitigen Implementierung des Horde3D Development Environments sind das rund 4800 Zeilen Code, die zunächst geschrieben und dann für

[7] `http://www.codeplex.com/ExceptionReporter`
[8] `http://www.icsharpcode.net/OpenSource/SD`
[9] `http://www.codeproject.com/KB/miscctrl/ScalablePictureBox.aspx`
[10] `http://microsoft.com/downloads/details.aspx?FamilyId=130F7986-BF49-4FE5-9CA8-910AE6EA442C`
[11] `http://www.codeproject.com/KB/graphics/TargaImage.aspx`
[12] `http://www.horde3d.org`

jede API-Änderung von Horde3D von Hand angepasst werden müssten. Um diese uninteressante und fehleranfällige Arbeit zu vermeiden, wurde ein Code Generator entwickelt, der den kompletten Code des DLL-*Replacement*-Mechanismus aus dem Horde3D-*Header* erzeugt.

4.2.1. Code Generator Phase I: Analyse

In der Analyse-Phase für den Code Generator wurde untersucht, wie das Tool arbeiten soll und was für Probleme bei der automatischen Typumwandlung von nativen C++-Typen in .NET-Typen entstehen können. Die Typ-Konvertierungen wurden in folgende Kategorien eingeteilt:

- Primitive Typen wie `int` und `float` können ohne manuelle Konvertierung sowohl in *managed* als auch *unmanaged* Code verwendet werden.
- Aufzählungstypen müssen über ihre `Integer`-Repräsentation konvertiert werden.
- Die `NodeHandle`- und `ResHandle`-Datentypen von Horde3D sind nur Umbenennungen von `int`. Da man in .NET aber keine globalen Typ-Aliase einführen kann, muss eine Konvertierung in `int` vorgenommen werden.
- Horde3D verwendet `const char*` als String-Repräsentation. Über den Konstruktor der .NET `String`-Klasse können die `const char*`-*Arrays* konvertiert werden.
- Im Gegensatz zu `const char*` sind alle anderen Zeiger problematisch. Es ist nicht klar, ob ein Zeiger einen *out*-Parameter einer Funktion repräsentiert oder ob er auf eine einzelne Variable oder auf ein ganzes *Array* verweist. Daher kann bei einem Zeiger-Parameter oder -Rückgabewert keine automatische Typumwandlung vorgenommen werden.

Der Code Generator soll eine Horde3D-*Header*-Datei parsen und den nötigen Code aus Abbildung C.4 sowie die Klasse `Horde3DCall` generieren. Er soll eine mit dem GUI-Framework erstellte Oberfläche bieten, die problematische Konvertierungen hervorhebt. Eine Konvertierung ist problematisch, wenn die Umwandlung in einen .NET-Typ nicht automatisch bestimmt werden kann, was im wesentlichen nur Zeiger betrifft. Für Zeiger auf primitive Typen kann allerdings vermutet werden, dass sie *out*-Parameter sind und entsprechend einfach nur dereferenziert werden müssen. Diese Standardkonvertierung für Zeiger auf primitive Typen muss aber als problematisch markiert werden, weil auch ein *Array* des primitiven Typs gemeint sein könnte. Völlig unklar hingegen ist die Konvertierung von `void`-Zeigern.

Der Benutzer soll die Möglichkeit haben, eigenen Code zum Umwandeln des Typs anzugeben oder eine der oben aufgelisteten Standardkonvertierungen auszuwählen. Er soll außerdem in der Lage sein, das Konvertierungsproblem als gelöst zu markieren. Alle manuellen Änderungen sollen aber bei der Neugenerierung des Code nach Updates der Horde3D-API übernommen werden können. Der Code Generator soll nur für die Generierung des Codes zuständig sein; die syntaktische Korrektheit des Codes, den der Benutzer eingegeben hat, muss nicht überprüft werden. Der generierte Code soll für unproblematische Typumwandlungen sowohl syntaktisch als auch semantisch korrekt sein.

Die im *Use Case* Modell 4.1 gezeigten Anwendungsfälle des Code Generators sind weitestgehend selbsterklärend. Mit dem System soll man die vorgenommenen Einstellungen speichern und später wieder laden können. Der Benutzer soll außerdem den Namen der generierten Code-Dateien angeben können und auf Wunsch alle Einstellungen verwerfen und von vorne beginnen können. Interessanter sind die Anwendungsfälle für das Ändern der Einstellungen der Code Generierung und für das Parsen der *Header*-Datei. Ersteres wird durch das

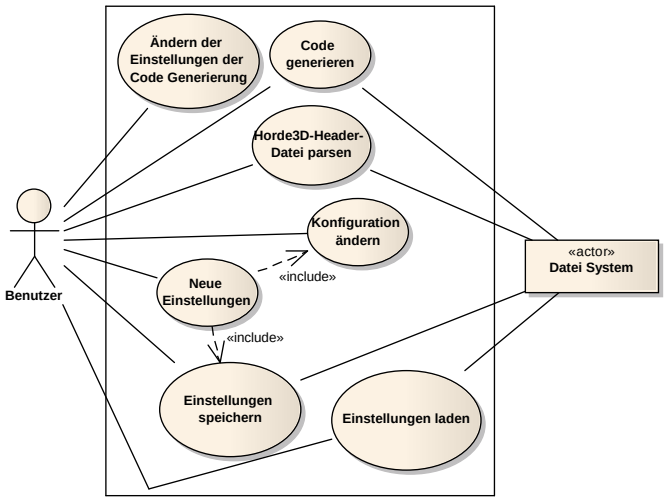

Abbildung 4.1.: Das *Use Case* Modell des Code Generators

Aktivitätsdiagramm D.2 dargestellt. Wenn der Benutzer die Einstellungen ändern möchte, zeigt ihm das System zunächst alle problematischen Funktionen an. In der Liste sind alle Funktionen, für die die Konvertierung mindestens eines Parameters oder des Rückgabewertes problematisch ist. Auf Wunsch kann sich der Benutzer aber auch alle Horde3D-Funktionen anzeigen lassen. Der Benutzer kann nun eine der Funktionen auswählen, und das System zeigt die ausgewählten Typumwandlungen für alle Parameter und den Rückgabetyp an. Problematische Konvertierungen werden hervorgehoben. Ein Konvertierungsproblem kann nun als gelöst markiert werden, oder die automatisch gewählte Umwandlung geändert werden. Das System speichert die Änderungen, und der Benutzer kann weitere Einstellungen der gewählten Funktion oder weiterer Funktionen ändern.

Beim Parsen der *Header*-Datei von Horde3D sollen alle Funktionen und ihre Parameter- und Rückgabetypen ausgelesen werden. Es soll versucht werden, eine automatische Typumwandlung für alle Typen der Funktion zu finden. Gelingt dies nicht, soll die Umwandlung als problematisch markiert werden. Falls eine aktualisierte *Header*-Datei geparst wird und der Benutzer vor dem Update bereits manuelle Einstellungen vorgenommen hat, sollen diese übernommen werden. Es muss dazu für jede zuvor manuell geänderte Funktion überprüft werden, ob die Änderungen auf die neu geparste Funktion übertragen werden können. Sollte dies nicht gelingen, so soll die Funktion als problematisch markiert werden und die alten Einstellungen zusätzlich zu den neu generierten gespeichert werden.

Abbildung D.1 zeigt das Konzeptmodell des Code Generators. `CodeGenerationSettings` speichert den Dateinamen der zu generierenden C++- und C#-Dateien, sowie eine Liste der Horde3D-Funktionen. Die Einstellungen können in einer Datei gespeichert und wieder geladen werden. Der programmiersprachliche Aufbau der Funktionen wurden in seine Bestandteile zerlegt: Das `Function`-Konzept enthält den Funktionsnamen und hat für den Rückgabetyp und alle Parameter der Funktion eine Assoziation zu einem `Type`-Konzept. Dort wird der C++-Typ gespeichert und festgehalten, ob die Konvertierung problematisch ist, das Problem als gelöst markiert wurde oder ob vom Benutzer manuelle Änderungen vorgenommen wurden. Die gewählte Typumwandlung eines `Type`s wird durch die Assoziation zu einem Konzept der `TypeConversion`-Hierarchie beschrieben. Diese Hierarchie entspricht im wesentli-

chen den oben beschriebenen Kategorien der Typumwandlung. `ToStringConversion` macht aus einem `const char*` einen `System.String`, `ToEnumConversion` konvertiert eine C++-Aufzählung in eine C#-Aufzählung, bei einer `DirectConversion` bleibt der Typ unverändert, `DereferencePointerConversion` dereferenziert einen Zeiger und `CodeConversion` erlaubt beliebigen Code. Der Grund für diese genaue Aufschlüsselung der Konvertierungsmethoden liegt in dem unterschiedlichen Code begründet, der für die jeweiligen Methoden generiert werden muss. Die `TypeConversion`-Hierarchie erlaubt für die Fallunterscheidungen virtuelle Funktionen zu verwenden, anstatt explizit mit `if-then-else`-Konstruktionen arbeiten zu müssen. Aufgrund dieser sehr implementierungsnahen Argumentation hätte die Hierarchie auch erst in der Design- oder sogar erst in der Implementierungs-Phase eingeführt werden können. Da jedoch die unterschiedlichen Konvertierungskonzepte bereits in dieser Phase untersucht wurden und im Mittelpunkt der Anwendung stehen, wurden sie bereits ins Konzeptmodell aufgenommen.

Der `AutomaticTypeConverter` versucht über die ihm bekannten `ConversionRule`s für einen gegebenen `Type` die beste `TypeConversion` zu finden. Eine Umwandlungsregel kann als problematisch markiert sein, wenn ihre Richtigkeit nicht garantiert werden kann. Welche Regeln der Code Generator kennt und verwendet, wird in Abbildung 4.2 aufgelistet.

4.2.2. Code Generator Phase II: Design

Wie aus Abbildung D.3 ersichtlich ist, gibt es nur wenige Unterschiede zwischen dem Designmodell und dem Konzeptmodell des Code Generators. Die Verantwortlichkeiten wurden auf die Klassen verteilt und die `TypeConversion`-Hierarchie noch etwas verfeinert. Es wird nun zwischen der aus dem Konzeptmodell bekannten `CodeConversion` und den neuen `InlineConversion`s unterschieden. Außer der `CodeConversion` erben alle anderen Konvertierungsklassen von der Basis-Klasse `InlineConversion`, da die jeweiligen Typumwandlungen in nur einer Zeile Code ausgeführt werden können. `CodeConversion` erlaubt im Unterschied dazu beliebig langen Code über mehrere Zeilen, wohingegen `InlineCodeConversion` beliebigen Code in nur einer einzigen Zeile zulässt.

Ebenfalls im Design hinzugekommen ist die Klasse `Horde3DHeaderFileParser`, welche die Funktionen in der *Header*-Datei parst und in die Objekt-Struktur der Funktionen aufbaut.

Der Code für die Code Generierung wird aus den `Function`-Objekten ausgelagert. Diese Mischung aus *Decorator* und *Strategy Pattern* wird durch die `FunctionCodeGenerator`-Hierarchie umgesetzt. Abhängig von der Art des zu generierenden Codes wird jedes `Function`-Objekt an eine `C++FunctionGenerator`- oder `C#FunctionGenerator`-Instanz übergeben. Die Klassen enthalten die benötigten Funktionen zur Generierung des Codes und zum Anzeigen einer Code-Vorschau. Erstellt werden die `FunctionCodeGenerator`-Objekte durch eine `CodeGenerator`-Instanz, die Funktionen zur Generierung des gesamten Codes anbietet.

Einziges Modell des Code Generators ist die `CodeGenerationSettings`-Klasse, die im Designklassen-Diagramm mit dem Stereotyp `model` hervorgehoben wurde.

Die Abbildungen D.4 und D.5 zeigen die Vorgehensweise zur Ermittlung aller Funktionen mit problematischen Typ-Konvertierungen. Es werden alle Funktionen zurückgeliefert, für die die Umwandlung des Rückgabetyps oder mindestens eines Parametertyps problematisch ist. Falls beim Aktualisieren der Funktion manuelle Benutzereingaben nicht automatisch übernommen werden konnten, wird die Funktion ebenfalls zurückgeliefert.

Das Parsen der *Header*-Datei wird durch das Sequenzdiagramm D.6 dargestellt. Zunächst wird eine Instanz der `Horde3DHeaderFileParser`-Klasse erzeugt und der Pfad zur *Header*-

Datei übergeben. In der übergebenen Datei werden alle Funktionen extrahiert und deren Rückgabetyp und Parameter ermittelt. Für alle Typen der Funktion wird die passende Typumwandlung gesucht. Die `GuessTypeConversion`-Methode der `Type`-Klasse verwendet die *Singleton*-Instanz des `AutomaticTypeConverters`, um basierend auf dem C++-Typen und den Konvertierungsregeln ein geeignetes `TypeConversion`-Objekt zu erzeugen. Nachdem für alle Funktionen alle benötigten Objekte erstellt wurden, wird versucht, alle manuellen Änderungen des Benutzers in die neuen Funktionsobjekte zu kopieren. Sollte es dabei zu Problemen kommen, wird das alte Funktionsobjekt mit all seinen Einstellungen über die `OldFunction`-Assoziation mit dem neuen Funktionsobjekt verbunden. Die Funktion wird dann als problematisch gelistet, und der Benutzer kann seine Änderungen überprüfen und nachziehen.

4.2.3. Code Generator Phase III: Implementierung

Während der Implementierung des Code Generators erwies sich die hierarchische Einteilung der `TypeConversion` als vorteilhaft. Der Code zum Generieren des C#-Codes konnte mit Ausnahme der `ToEnumConversion` komplett durch die abstrakte `TypeConversion`-Klasse selbst abgedeckt werden. Bei der Umsetzung der Generierung des C++-Codes hingegen gab es größere Unterschiede. Durch die Verwendung virtueller Funktionen konnte der Code jedoch auf die einzelnen `TypeConversion`-Klassen aufgeteilt werden, wodurch die `FunctionCodeGenerator`-Klassen keine Fallunterscheidungen auf Grundlage der verwendeten Typ-Konvertierungen durchführen müssen.

C++-Typ	TypeConversion-Klasse	C#-Typ	Problematisch	Bemerkung
`bool`	`DirectConversion`	`System.Boolean`	Nein	
`int`	`DirectConversion`	`System.Int32`	Nein	
`float`	`DirectConversion`	`System.Single`	Nein	
`double`	`DirectConversion`	`System.Double`	Nein	
`void`	`DirectConversion`	`System.Void`	Nein	Rückgabetyp, wird ignoriert
`const char*`	`ToStringConversion`	`System.String`	Nein	
`EngineOptions`	`ToEnumConversion`	`Horde3D.EngineOptions`	Nein	
`EngineStats`	`ToEnumConversion`	`Horde3D.EngineStats`	Nein	
`ResourceTypes`	`ToEnumConversion`	`Horde3D.ResourceTypes`	Nein	
`SceneNodeTypes`	`ToEnumConversion`	`Horde3D.SceneNodeTypes`	Nein	
`NodeHandle`	`InlineCodeConversion`	`System.Int32`	Nein	Alias für `int`
`ResHandle`	`InlineCodeConversion`	`System.Int32`	Nein	Alias für `int`
`NodeHandle*`	`InlineCodeConversion`	`System.Int32`	Ja	Alias für `int`, out-Parameter?
`const float**`	`CodeConversion`	`System.IntPtr`	Ja	
`const float*`	`DereferencePointerConversion`	`System.Single`	Ja	out-Parameter?
`const void**`	`CodeConversion`	`System.IntPtr`	Ja	
`const void*`	`CodeConversion`	`System.IntPtr`	Ja	
`int*`	`DereferencePointerConversion`	`System.Int32`	Ja	out-Parameter?
`float*`	`DereferencePointerConversion`	`System.Single`	Ja	out-Parameter?

Abbildung 4.2.: Konvertierungsregeln des Code Generators

Für das Parsen der Funktionen in der Horde3D-*Header*-Datei wurden *Regular Expressions* verwendet, wodurch der Parsing-Code kompakt gehalten werden konnte. Auch für die Generierung des C#-Codes wurde auf bereits vorhandene .NET-Funktionalitäten zurückgegriffen; .NET enthält einen Code Generator für C#, Visual Basic .NET und C++/CLI. Der C++-Code musste jedoch von Hand durch Konstruktion eines Strings generiert werden, da der .NET Code Generator keinen nativen C++-Code erzeugen kann.

Die Bestimmung der benötigten `CodeConversionRules` orientierte sich an den verwendeten Datentypen der Horde3D-Funktionen und ist somit nicht allgemein vollständig, sondern nur ausreichend für Version 1.0.0 Beta 3 von Horde3D. Die Konvertierungsregeln sind in Abbildung 4.2 tabellarisch aufgelistet.

4.2.4. Code Generierung nach einem Update der Horde3D-API

Wenn sich die öffentliche API einer neuen Horde3D-Version geändert hat, muss die Code Generierung erneut ausgeführt werden. Um Änderungen, die an den Einstellungen problematischer Funktionen durchgeführt wurden, nicht immer wieder machen zu müssen, verfügt der Code Generator über einen automatischen Update-Mechanismus. Tabelle 4.3 zeigt, wie dieser Mechanismus mit API-Updates zurechtkommt.

	Horde3D Version			
	0.0.15	1.0.0 Beta 1	1.0.0 Beta 2	1.0.0 Beta 3
Horde3D-Funktionen	67	68	75	78
Problematische Funktionen	9	9	10	10
... bereits als gelöst markiert	-	9	9	9
... eigentlich unproblematisch	3	0	1	0
Notwendige Änderungen	6	0	0	1
Änderungen übernommen	-	9	9	9
Änderungen verworfen	-	0	0	1

Abbildung 4.3.: Evaluation des Update-Mechanismus des Code Generators

Ausgehend von Version 0.0.15 wurden die API-Updates auf die Betas 1, 2 und 3 von Horde3D 1.0.0 untersucht. Version 0.0.15 hatte 67 öffentliche Funktionen, von denen neun als problematisch markiert wurden. Von diesen neun waren wiederum drei eigentlich unproblematisch. Die Annahme, bei Zeiger-Parametern handle es sich eigentlich um *out*-Parameter, war für diese drei Funktionen korrekt. Die verbleibenden sechs problematischen Funktionen mussten allerdings wirklich von Hand modifiziert werden. In zwei Fällen konnten `void`-Zeiger sogar nur als `System.IntPtr` weitergereicht werden. Ansonsten war es möglich, `float`-Zeiger, die Matrizen repräsentierten, als `float`-*Arrays* der Länge 16 weiterzugeben.

Beim Update auf Version 1.0.0 Beta 1 kam eine neue, unproblematische Funktion hinzu. Die manuellen Änderungen konnten alle erfolgreich übernommen werden, sodass nichts weiter zu tun war und der Code sofort neu generiert werden konnte. Mit Beta 2 kamen weitere Funktionen hinzu, von denen eine als problematisch erkannt wurde. Allerdings war auch hier die *out*-Parameter-Annahme korrekt. Alle manuellen Änderungen von Version 0.0.15 wurden auch bei diesem Update korrekt übernommen. Beta 3 führte weitere neue Funktionen ein. Beim Update wurde allerdings eine manuelle Änderung verworfen, da die Funktion umbenannt wurde. Umbenannte Funktionen erkennt der Update-Mechanismus nicht. Die Änderung musste also manuell aus den vorherigen Einstellungen kopiert werden; ansonsten waren aber auch in diesem Update-Schritt keine weiteren Anpassungen nötig.

4.3. Client-Server-Kommunikation

Bei der Implementierung der Client-Server-Kommunikation mit der Windows Communication Foundation konnte das `IDebuggerService`- und das `IDebuggerServiceCallback`-Interface

aus dem Design unverändert übernommen werden. Aus technischen Gründen musste allerdings noch die Service-Operation `RegisterClient` hinzugefügt werden. Der Client ruft diese Funktion sofort nach dem Zustandekommen der Verbindung auf, um seinen *Callback Channel* beim Server zu registrieren. Dann erst kann der Server *Callbacks* an den Client schicken.

Es mussten besondere Maßnahmen getroffen werden, um *Deadlocks* und *Race Conditions* zu vermeiden. Die Horde3D-Anwendung kann mehrere Threads verwenden; der Server hat keine Möglichkeit dies herauszufinden und sollte davon auch nicht abhängen, da die Horde3D-API sowieso nicht *thread-safe* ist. Die Implementierung geht deshalb davon aus, dass alle Horde3D-Funktionen vom gleichen Thread aus aufgerufen werden. WCF führt aber jede Anfrage des Clients in einem gerade zur Verfügung stehenden Thread des *Thread Pools* aus. Da die Service-Operationen zumindest teilweise auf Horde3D-Funktionen zugreifen, könnte es somit zu *Race Conditions* kommen oder auf inkonsistenten Zuständen gearbeitet werden. Es musste daher sichergestellt werden, dass jeder Server-Aufruf im gleichen Thread wie Horde3D läuft.

Das .NET Framework bietet die `SynchronizationContext`-Klasse für Threadwechsel an. Der `Post`-Methode dieser Klasse kann ein *Delegate* übergeben werden, der in einem ganz bestimmten Thread ausgeführt wird. Leider stellt die Standard-Implementierung der Klasse nur die Funktionen bereit, das Wechseln des Threads ist nicht implementiert[13]. Es wurde daher der `Horde3DSynchronizationContext` entwickelt, der von `SynchronizationContext` erbt. Wird die `Post`-Methode der Klasse aufgerufen, wird die übergebene Funktion in einer `Queue` *thread-safe* protokolliert, aber noch nicht ausgeführt.

Bei jedem Aufruf einer Horde3D-Funktion löst `Horde3DCall` das `BeforeFunctionCalled`-Ereignis aus. Der `Horde3DDebugger` registriert einen *Event Handler*, der beim Auslösen dieses Ereignisses die `Execute`-Methode des `Horde3DSynchronizationContext`s aufruft. Da die Horde-Funktion und somit auch das Ereignis im Horde3D-Thread laufen, werden alle Methoden, die derzeit in der `Queue` des Synchronisationskontexts liegen, der Reihe nach im Horde3D-Thread ausgeführt.

Dieses Prinzip stellt sicher, dass alle Aufrufe der WCF-Serverfunktionen im Horde3D-Thread laufen. Da WCF sehr flexibel und konfigurierbar entwickelt wurde, kann WCF automatisch eine Instanz des `Horde3DSynchronizationContext`s zum Wechseln des Threads verwenden. Dazu musste das `Horde3DThreadAffinityAttribute`-Attribut definiert werden, welches das `IContractBehavior`-Interface von WCF implementiert. Mit diesem Attribut kann der Synchronisationskontext von WCF programmatisch gesetzt werden [19].

Das Design sah auch vor, dass *Callback*-Aufrufe garantiert an den Client geschickt werden, auch, wenn dieser noch gar nicht verbunden ist. Die `DebuggerService`-Klasse wurde daher um statische Funktionen für jede *Callback*-Operation erweitert, die den *Callback*-Aufruf abfangen und in einen anderen Thread verlagern. Dieser Thread versucht solange den *Callback* auszuführen, bis eine Verbindung aufgebaut wurde und der *Callback* erfolgreich durchgeführt werden konnte.

Client-seitig werden Serveroperationen immer aus einem *Backgroundthread* aufgerufen, damit die GUI nicht blockiert wird und weiterhin auf Benutzereingaben reagieren kann. Die `DebuggerServiceProxy`-Klasse kümmert sich intern um die Verbindung zum Server und die Thread-Verwaltung. Falls der Client noch nicht zum Server verbunden ist, wird vor dem Aufruf der Operation eine Verbindung aufgebaut.

[13] Der Grund dafür ist, dass es verschiedene Möglichkeiten für die Implementierung des Threadwechsels gibt. .NET bietet zwei Implementierungen für WinForms und WPF an, die allerdings auch nur von WinForms- und WPF-Anwendungen verwendet werden können. Horde3D-Anwendung können aber auch die native Win32-API zum Erzeugen des Anwendungsfensters benutzen.

Ein Problem ergab sich durch die Assoziationen zwischen den Ressourcen und insbesondere der *Scene Nodes* untereinander. WCF serialisiert immer den kompletten Objektgraph. Schickt man also beispielsweise einen *Scene Node* an den Client, wird auch dessen Vaterknoten mitgeschickt. Aber auch der Vater des Vaters wird mit übertragen. Das endet erst beim Erreichen der Wurzel. Um dieses Problem zu vermeiden, werden die Assoziationen über die `NodeHandle`s beziehungsweise `ResHandle`s identifiziert. Der Client muss dann über diese IDs das eigentliche Objekt beim `SceneGraph`- oder `ResourceGraph`-Modell erfragen. Die Zuordnung von den IDs zu den Objekten erfolgt über eine Hashtabelle, hat also nur die Komplexität $O(1)$. Um den Zugriff syntaktisch möglichst einfach zu halten, ist in der `Graph`-Klasse ein Indexoperator definiert. Das folgende Beispiel zeigt die Implementierung des *Properties* `SceneNode.Parent`, das über die `SceneGraph`-Instanz und den `NodeHandle` des Vaters die Vater-Instanz zurückliefert:

```
public class SceneNode
{
  internal int ParentHandle { get; }

  public SceneGraph SceneGraph { get; set; }

  public SceneNode Parent
  {
    get { return SceneGraph[ParentHandle]; }
  }

  // ...
}
```

Der Zugriff auf die Vater-Instanz über das `ParentHandle`-*Property* wird komplett gekapselt. Für den Benutzer der Klasse macht es keinen Unterschied, ob die Vater-Instanz direkt übertragen wird, oder aus dem `SceneGraph`-Objekt stammt.

4.4. Anhalten der Anwendung

Das Horde3D Development Environment verwendet drei verschiedene Implementierungen des `ISuspendApplicationStrategy`-Interfaces zum Anhalten der Anwendung. Eine Strategie täuscht dem Prozess vor, dass keine Zeit mehr vergeht. Eine andere leitet alle Tastatur- und Mauseingaben des Benutzers an eine andere Funktion um. Die dritte Klasse zum Fangen und Freilassen des Mauszeigers kam erst nach der zweiten Iteration der Design-Phase hinzu. Dank der Verwendung des *Strategy Patterns* musste jedoch nur eine Zeile Code im Konstruktor der `SuspendApplicationStrategy`-Klasse hinzugefügt werden, um die dritte Strategie einzubinden.

Die ursprüngliche Idee war, den Prozess der Anwendung wirklich anzuhalten. Dazu gibt es die Funktionen `SuspendThread` und `ResumeThread` der Win32-API. Mit Hilfe dieser Funktionen kann ein Thread pausiert und wieder fortgesetzt werden. Ruft man diese Funktionen für alle Threads eines Prozesses auf, so wird der komplette Prozess eingefroren und wieder fortgesetzt. Dieses Verfahren ist jedoch nicht problemlos anwendbar, da Teile des Prozesses noch weiterlaufen sollten; so sollte WCF in der Lage sein, Anfragen vom Client zu bearbeiten und *Callbacks* auszulösen. Um das zu gewährleisten, darf der Thread, in dem WCF läuft, nicht angehalten werden. Die Win32-Funktionen arbeiten aber auf nativen Threads,

während WCF in einem *managed* Thread läuft. Es kann aber passieren, dass die CLR mehrere *managed* Threads im gleichen nativen Thread laufen lässt. Noch problematischer wird es, wenn die Horde3D-Anwendung ebenfalls in .NET geschrieben wurde oder mehrere *AppDomains*[14] im Prozess laufen. Es ist also schwierig genau festzustellen, welche Threads angehalten werden dürfen. Des Weiteren kommt noch hinzu, dass es verschiedene Versionen der `Suspend-/ResumeThread`-Funktionen für native 32 Bit Prozesse, 32 Bit Prozesse unter einem 64 Bit Betriebssystem und 64 Bit Prozesse gibt.

Derzeit ist für das richtige Anhalten des Prozesses keine `ISuspendApplicationStrategy`-Implementierung vorhanden. In vielen Fällen sind die drei im folgenden vorgestellten Strategien ausreichend. Es gibt aber auch Fälle, in denen ein richtiges Anhalten des Prozesses notwendig wäre. Im folgenden Abschnitt wird auf diese Problematik näher eingegangen.

4.4.1. Anhalten der Zeit

Um den Prozess nicht wirklich anhalten zu müssen, wurde eine Idee aus NVIDIAs PerfHUD übernommen: Das Anhalten der Zeit. Im Benutzerhandbuch gibt es einen Hinweis darauf, wie NVIDIA dieses Problem löst:

> „PerfHUD “freezes” your application by returning the same value every time your application asks for the current time.“ [9, S. 11]

Das ist auch die Vorgehensweise des Horde3D Development Environments. Die meisten Anwendungen verwenden die hochauflösende *Timer*-Funktion `QueryPerformanceCounter`, um die vergangene Zeit zu messen. Jeder Aufruf dieser Funktion wird an eine Funktion des Servers umgeleitet. Dort wird der wirkliche *Timer* aufgerufen und der Rückgabewert aufgezeichnet. Ist die Anwendung angehalten, wird immer der Zeitwert zum Zeitpunkt des Anhaltens der Anwendung zurückgeliefert; die Anwendung meint, es sei seit dem letzten Frame keine Zeit vergangen. Dadurch ist die Zeit angehalten und die Anwendung pausiert. Wird die Anwendung wieder fortgesetzt, so wird wieder der richtige Zeitwert geliefert, abzüglich der Zeitdauer, während der die Anwendung pausiert war.

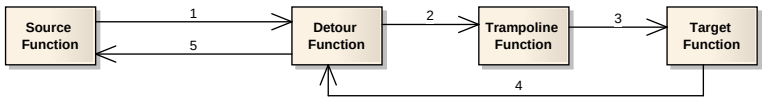

Abbildung 4.4.: Kontrollfluss bei einem abgefangenen Funktionsaufruf

Technisch wird das Umleiten des Funktionsaufrufs durch Microsofts Detours Express Bibliothek gelöst. Die Bibliothek kann in einer Anwendung verwendet werden, um einen Aufruf einer beliebigen, nativen Funktionen (*Target Function*) abzufangen und auf eine beliebige selbst definierte Funktionen (*Detour Function*) umzuleiten. Zur Laufzeit schreibt Detours das Abbild des in den Prozess geladenen Binärcodes der *Target Function* um. Abbildung 4.4 zeigt den Kontrollfluss beim abgefangenen Aufruf der *Target Function*. Die ersten Instruktionen der *Target Function* werden durch einen Sprung zur *Detour Function* ersetzt. Es wird also als erstes die *Detour Function* ausgeführt. Dort kann beliebiger Code abgearbeitet werden. Detours erzeugt außerdem eine *Trampoline Function*, welche die ersetzen Instruktionen

[14] Eine .NET-*AppDomain* kann man sich als einen .NET-Prozess vorstellen. Es kann in einem nativen Prozess mehrere *AppDomains* geben. Das Betriebssystem kennt das Konzept der *AppDomains* nicht.

der *Target Function* enthält und anschließend zur *Target Function* zurückspringt. Die *Detour Function* kann die *Trampoline Function* aufrufen, um die *Target Function* auszuführen. Verlässt der Kontrollfluss die *Target Function*, ist wieder die *Detour Function* aktiv und kann weiterhin beliebigen Code ausführen. Schließlich kehrt der Kontrollfluss zur aufrufenden Funktion (*Source Function*) zurück.

Das Anhalten der Zeit hat einige wichtige Konsequenzen: Zum einen funktioniert die Vorgehensweise gar nicht, wenn die Anwendung nicht `QueryPerformanceCounter` zum Messen der Zeit verwendet. Da es aber keine anderen hochauflösenden Zeitmesser für Windows gibt, wird dies nicht oft der Fall sein. Zum anderen kommt es zu Problemen, wenn die Anwendung nicht so schnell arbeitet wie sie kann, sondern auf eine gewisse Zahl an Updates pro Sekunde festgelegt ist. Falls die Anwendung beispielsweise maximal 60 mal pro Sekunde ein Update ausführt, aber scheinbar seit dem letzten Frame keine Zeit vergangen ist, wird sie einfach gar nichts mehr tun; sie sollte aber wenigstens noch `Horde3D::render` aufrufen. Gegebenenfalls muss zur Verwendung des Horde3D Development Environments die Frameraten-Limitierung abgeschaltet werden. Da beim Starten der Anwendung Kommandozeilenparameter übergeben werden können, kann die Abschaltung der Limitierung prinzipiell auch ohne eine Änderung der Anwendung erfolgen, falls die Anwendung per Kommandozeile entsprechend konfigurierbar ist.

Das Vorgehen funktioniert auch nicht, wenn für zeitabhängige Berechnungen nicht die vergangene Zeit Δt sondern die aktuellen *Frames Per Second* (FPS) verwendet werden. Da FPS $= 1/\Delta t$ wird die FPS-Berechnung für $\Delta t = 0$ entweder nicht ausgeführt, oder sie liefert keinen vernünftigen Wert zurück.

Die Methode hat auch keinen Erfolg, wenn die Anwendung oder Teile der Anwendung zeitunabhängig arbeiten. Denkbar wäre zum Beispiel eine Client-Server-Anwendung, in der der Server kontinuierlich neue Positionen der Objekte berechnet und an den Client schickt. Der Client empfängt die Nachrichten des Servers, baut die Szene entsprechend um und zeichnet sie neu. Für den Benutzer scheint die Anwendung weiterhin zu laufen, obwohl der Client eigentlich angehalten ist und für ihn keine Zeit mehr vergeht.

All diese Probleme würden nicht auftauchen, wenn der Prozess der Anwendung wirklich angehalten werden würde, wie oben beschrieben. Es wurde dennoch aus zwei Gründen nur die Zeit-Methode umgesetzt: Erstens ist wie beschrieben beim Anhalten des Prozesses nicht klar, welche Threads angehalten werden dürfen, wohingegen die Implementierung der Zeit-Methode einfach ist. Zweitens wird die Zeit-Methode in jedem Fall benötigt. Hält man die Threads der Anwendung an, so laufen diese zwar nicht weiter, die Zeit aber schon. Setzt man nun die Threads fort, so können seit dem letzten Frame mehrere Sekunden oder Minuten vergangen sein. Die Physik- und Animationssysteme rechnen dann mit einem viel zu großen Δt von mehreren Sekunden oder Minuten und können eventuell keinen stabilen Zustand mehr erreichen.

4.4.2. Ersetzen des Window Procedures

Die Win32-API verwendet Nachrichten, um mit einem Anwendungsprozess zu kommunizieren. Mit den Nachrichten kann die Anwendung über alle wichtigen Ereignisse informiert werden: Mausklicks, Tastatureingaben, Mausbewegungen, Schließen der Anwendung, Verschieben des Anwendungsfensters, Ablaufen eines *Timers*, und so weiter. Alle Nachrichten an einen Prozess beziehungsweise an ein Fenster werden an eine ausgewiesene Funktion, das *Window Procedure* (*WndProc*), geschickt. Dort wird als Reaktion auf spezielle Nachrichten anwendungsspezifi-

scher Code ausgeführt.

Das Horde3D Development Environment muss auf zwei Arten mit dem *Window Procedure* der Anwendung interagieren: Einerseits muss es Nachrichten abfangen, die für den Server eine Bedeutung haben. Drückt der Benutzer beispielsweise die Taste F7, so soll der Server die Anwendung anhalten. Auf der anderen Seite darf das *Window Procedure* der Anwendung keine Benutzereingaben mehr erhalten, wenn die Anwendung angehalten ist. Es soll aber dennoch möglich sein, das Anwendungsfenster zu verschieben oder zu schließen.

Bei der Initialisierung des Servers wird mit der Win32-Funktion `SetWindowsHookEx` ein prozessweiter *Hook* aktiviert, der alle Nachrichten an das *WndProc* der Anwendung abfängt. Innerhalb des *Hooks* wird überprüft, ob für den Server eine relevante Nachricht – also zum Beispiel `KeyDown` für die Taste F7 – geschickt wurde. In diesem Fall reagiert der Server entsprechend. Anschließend wird die Nachricht an das *WndProc* weitergeleitet.

Mit der Funktion `SetWindowLongPtr` kann der *WndProc*-Zeiger eines Fensters auf eine beliebige benutzerdefinierte Funktion gesetzt werden. Der Server ersetzt das *Window Procedure* des Anwendungsfensters, wenn die Anwendung angehalten wird. Das *WndProc* des Servers ignoriert alle eingehenden Nachrichten über Maus- und Tastatureingaben und behandelt nur Fenster-Nachrichten wie Verschieben oder Schließen. Beim Fortsetzen der Anwendung wird der Zeiger wieder auf das ursprüngliche *WndProc* gesetzt.

4.4.3. Freigabe des Mauszeigers

Viele Anwendungen beschränken den Mauszeiger auf ihre Fenstergröße und verstecken ihn. Beim Testen des Horde3D Development Environments mit den Beispielapplikationen des Horde3D-SDKs zeigten sich verschiedene Probleme mit dieser exklusiven Benutzung der Maus. Der Server fängt daher – wiederum mit der Detours Express Bibliothek von Microsoft – alle Aufrufe der Win32-Funktionen `ShowCursor`, `ClipCursor` und `SetCursorPos` ab. Aufrufe dieser Funktionen werden protokolliert und anschließend die ursprünglichen Funktionen aufgerufen. Dadurch kann der Mauszeiger beim Anhalten der Anwendung freigegeben werden und beim Fortsetzen der Anwendung der von der Anwendung gewünschte Zustand wiederhergestellt werden.

4.5. Aufbau der Visual Studio Solution

Die Visual Studio Projektmappe für das Horde3D Development Environment besteht aus mehreren Projekten, die in mehreren Kategorien gruppiert sind. Abbildung 4.5 zeigt einen Screenshot des *Solution Explorers* von Visual Studio und gibt einen Überblick über die verschiedenen Projekte.

Die Verzeichnis-Struktur des Projekts im Dateisystem entspricht nicht dem Visual Studio-Standard. Unterhalb des Hauptverzeichnisses gibt es drei weitere Verzeichnisse: bin, in dem alle kompilierten DLLs, die *Executable* des Clients und das Knight-Sample liegen; Dependencies, in dem alle benötigten Bibliotheken liegen; und src, in dem der komplette Source Code liegt.

Infrastructure

In der Infrastruktur-Kategorie befinden sich die Projekte, die vom Client und vom Server gemeinsam benutzt werden, sowie der Code Generator und der *Installer*. Alle Projekte dieser

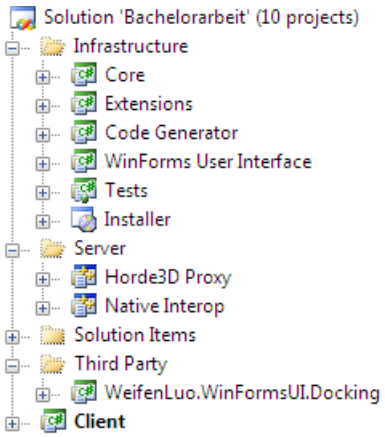

Abbildung 4.5.: Aufbau der Visual Studio 2008 *Solution*

Kategorie sind in C# geschrieben.

- **Core** (DLL): Diese DLL enthält die Kern-Klassen des Systems. Dort befinden sich die Klassen und Interface-Definitionen für die Client-Server-Kommunikation, die Horde3D-Klassen, die Klassen für das Profiling und die Horde3D-Meldungen sowie `Horde3DCall` und `Horde3DDebugger`.
- **Extensions** (DLL): Dieses Projekt stellt einige nützliche Erweiterungen für das .NET Framework bereit, die vom Horde3D Development Environment an vielen Stellen verwendet werden.
- **Code Generator** (*Executable*): Der Code Generator ist eine eigenständige Anwendung zur Generierung der Horde3D-*Proxy*-Funktionen und -Klassen. Zur Laufzeit des Systems wird dieses Projekt nicht benötigt.
- **WinForms User Interface** (DLL): Diese *Assembly* enthält die Implementierung des GUI-Frameworks unter Verwendung der Windows Forms-Klassen.
- **Tests** (DLL): Diese Bibliothek enthält *Unit Tests* für das System. Derzeit gibt es allerdings nur Tests für das Parsen der XML-Dateien von Shader-Ressourcen.
- **Installer** (*Executable*): Das Projekt erzeugt eine .msi-Datei für den Windows *Installer*. Der *Installer* kann das Horde3D Development Environment auf einem Computer installieren, aktualisieren und wieder löschen. Alle Voraussetzungen – .NET 3.5 SP1 und die Visual C++ *Redistributable* für 32 Bit Systeme – werden automatisch installiert.

Server

Die Server-Projekte sind .NET-*Assemblies*, die in C++/CLI geschrieben wurden. In C++/CLI ist der Zugriff auf native Funktionen sowie die Verwendung von Zeigern einfacher und performanter als unter C#. Möchte man diese *Assemblies* debuggen, so benötigt man zunächst eine Horde3D-Anwendung, die vom Horde3D Development Environment gestartet wurde. Anschließend kann man die „*Attach Debugger*“-Funktionalität von Visual Studio verwenden, um den Prozess der Anwendung zu debuggen. Zu beachten ist, dass die DLLs sowohl *managed* als

auch nativen Code enthalten. Der Debugger muss daher unbedingt auf „*mixed mode*“ gesetzt werden, da er ansonsten nur den nativen oder nur den *managed* Teil des Codes sieht.

- **Horde3D Proxy** (DLL): In dieser DLL befinden sich die Horde3D-*Proxy*-Funktionen sowie die globale Instanz der `Horde3DProxyHandler`-Klasse.
- **Native Interop** (DLL): Der Server führt einige Aktionen durch, die sehr stark von der Win32-API abhängen und daher in C++/CLI einfacher implementierbar sind. In dieser DLL befindet sich der komplette Code des *Strategy Patterns* zum Anhalten der Anwendung und auch der Code zum Auslesen und Konvertieren des Inhalts eines *Render Targets*.

Third Party

Zur Zeit befindet sich bloß das Projekt der Weifen Luo Winforms Docking Bibliothek in dieser Kategorie. Das Horde3D Development Environment verwendet zwar noch weitere Open Source Bibliotheken, diese wurden aber nicht modifiziert. In der *Docking*-Bibliothek wurden hingegen ein paar kleine Änderungen vorgenommen, um das Aussehen der Shell an das von Visual Studio anzugleichen.

Client

Der Client ist das eigentliche ausführbare Projekt des Horde3D Development Environments. Vor der Kompilierung des Projekts werden die DLLs, die beim Starten des Servers in das Anwendungsverzeichnis kopiert werden müssen, in das *Resources*-Verzeichnis des Client-Projekts kopiert. Der Compiler kompiliert dann die DLLs in die .exe-Datei als .NET-Ressource hinein. Soll der Server gestartet werden, so können die benötigten DLLs aus den Ressourcen ausgelesen werden. Damit entfällt das fehleranfällige Kopieren direkt aus dem Anwendungsverzeichnis der Shell.

Da die Verzeichnis-Struktur der *Solution* nicht der Standardstruktur von Visual Studio entspricht, müssen zum Debuggen des Clients zwei Projekt-Einstellungen geändert werden. In den Einstellungen muss im Abschnitt „*Debug*“ die „*Start Action*“ auf „*Start external program*“ gesetzt werden. Außerdem muss der Pfad zur .exe-Datei im bin-Verzeichnis angegeben werden. Das „*Working Directory*“ muss ebenfalls auf das bin-Verzeichnis gesetzt werden.

4.6. Konklusion

Für die Implementierung des Horde3D Development Environments wurde eine Mischung aus nativem C++, C++/CLI und C# verwendet. In vielen Bereichen konnten bereits vorhandene Bibliotheken gewinnbringend verwendet werden; insbesondere wurde die Umsetzung der GUI durch die Verwendung der Weifen Luo Winforms Docking Bibliothek und einiger anderer *Controls* deutlich erleichtert.

Insgesamt wurden weite Teile des Designs erfolgreich umgesetzt. Es wurde jedoch nur der Code entwickelt, der zum Erfüllen der Anforderungen erforderlich war. Gerade bei den Horde3D-Klassen sind das Konzept- und auch das Designmodell jedoch detaillierter, als die Anforderungen eigentlich verlangten. An diesen Stellen wurden nur die erforderlichen Teile umgesetzt; die fehlenden Codeteile können jederzeit ergänzt werden.

Beim Auslesen der Ressourcen-Daten fiel ein Problem mit der Horde3D-API auf: Es gab keine einfache Möglichkeit, alle Horde3D derzeit bekannten Ressourcen zu ermitteln. Jedoch wurde auf Nachfrage bei den Entwicklern eine entsprechende Funktion in Horde3D 1.0.0 Beta 3 eingeführt. Die API wurde in dieser Version zusätzlich um eine Funktion ergänzt, die den Abschluss eines Frames markiert. Der Profiling-Mechanismus ist auf das Vorhandensein dieser Funktion angewiesen. Somit sind diese beiden Funktionen der Grund, warum das Horde3D Development Environment nur mit Beta 3 kompatibel ist.

Die Implementierung des GUI-Frameworks benötigte gegenüber dem Designmodell einige zusätzliche Klassen. Die neuen Klassen waren jedoch nur notwendig, um die `Shell`-, `Presenter`- und `IView`-Klassen durch Verwendung von *Generics* typsicher zu machen. Das ermöglichte ein schnelleres Umsetzen der notwendigen *Presenter* und *Views* des Horde3D Development Environments.

Die Erstellung des Code Generators nahm einige Zeit in Anspruch, da zunächst die Analyse- und Design-Phasen durchlaufen werden mussten. Es zeigte sich jedoch, dass rund 4800 Zeilen Code automatisch generiert werden konnten und nicht von Hand eingetippt werden mussten. Da der Code Generator auch gut mit Updates der Horde3D-API zurecht kommt, hat sich auch die detaillierte Analyse der Anforderungen und der möglichen Probleme bei der Code Generierung gelohnt. Durch die Verwendung des GUI-Frameworks bei der Implementierung der Benutzeroberfläche des Code Generators hielt sich der Implementierungsaufwand in Grenzen. Zudem konnten früh erste Erfahrungen im Umgang mit dem GUI-Framework gesammelt werden und einige Unschönheiten bei der Typsicherheit, der Sichtbarkeit von eigentlich internen *Properties* sowie dem thread-sicheren Zugriff auf die *View* eines *Presenters* behoben werden.

5. Evaluation und Ausblick

Während der Entwicklung des Horde3D Development Environments wurden bereits verschiedene Stärken, eventuelle Schwächen und Erweiterungsmöglichkeiten offensichtlich. Es wurde daher nach Abschluss der Entwicklungsarbeiten überprüft, ob das Tool die Anforderungen erfüllt. Dazu wurde zunächst am Beispiel von SheepMeUp getestet, wie gut das Tool mit bereits bestehenden Anwendungen zusammenarbeitet. Anschließend wurde für einen Prototyp eines Spiels ein Special Effect für den Schild eines Raumschiffes umgesetzt. Es wurden außerdem einige Entwickler aus der Horde-Community gebeten, das Tool zu testen und an einer Meinungsumfrage teilzunehmen, deren Ergebnisse im folgenden ebenfalls vorgestellt werden. Zuletzt wird noch auf einige Ideen für Erweiterungsmöglichkeiten und Verbesserungen eingegangen, die während der Entwicklung des Tools, der Umsetzung des Schild-Effekts oder beim Testen durch die Community-Mitglieder aufgekommen sind.

5.1. Zusammenarbeit mit SheepMeUp

SheepMeUp wurde ursprünglich für die Horde3D-Version 1.0.0 Beta 1 entwickelt und musste zunächst auf Beta 3 portiert werden. Die Portierung erwies sich als schwierig, umfangreich und zeitaufwändig, da das komplette Shader-System und große Teile des Material-Systems mit Beta 3 geändert wurden. Es war innerhalb eines vernünftigen Zeitrahmens daher nicht möglich, das Spiel voll funktionsfähig und spielbar auf Beta 3 zu portieren. Die Portierung war jedoch soweit erfolgreich, dass das Spiel wieder lief und nur einige grafische Effekte – wie Kraftfelder, Fackeln, sowie alle Menü-Elemente – fehlten. Dies reichte aber bereits aus, um das Horde3D Development Environment mit SheepMeUp verwenden zu können.

Es traten jedoch einige Kompatibilitätsprobleme mit SheepMeUp auf. Diese lagen zwar im Spiel selbst begründet, zeigten aber auch, dass die idealisierten Annahmen, die bei der Entwicklung des Horde3D Development Environments getroffen wurden, in der Realität nicht immer zutreffen. Im folgenden werden die Probleme und ihre Lösung kurz aufgeführt:

- SheepMeUp verwendete keine Originalversion der Horde3D DLL, sondern eine um die Funktion `castRayAABB` erweiterte Variante. Da das Spiel diese Funktion jedoch nicht aufruft, konnte einfach die Original-DLL verwendet werden. Andernfalls hätte mit dem Code Generator eine angepasste Version des DLL-*Replacement*-Codes generiert werden müssen.

- Anstatt die vergangene Zeit seit dem letzten Frame für zeitabhängige Berechnungen zu verwenden, wurden die aktuellen *Frames Per Second* (FPS) verwendet. Die FPS wurden jedoch über einen Zeitraum von einigen Frames gemittelt, spiegelten also nicht exakt die vergangene Zeit wider. Wird die Anwendung mit dem Horde3D Development Environment pausiert, so ist die vergangene Zeit seit dem letzten Frame $\Delta t = 0$. In diesem Fall wurden jedoch die FPS nicht aktualisiert. Das führte dazu, dass Schafe, Schockwelleneffekte, die Manaregeneration, zufällig abgespielte Sounds und Partikeleffekte nicht

mit $\Delta t = 0$ sondern mit einem falschen FPS-Wert berechnet wurden. Die Physik-Engine wurde jedoch angehalten; so bewegten sich die Schafe beispielsweise weiter, aber flogen durch Hyänen und Wände hindurch. Aufgrund der Ungenauigkeit der FPS-Berechnung sollte immer der wirkliche Δt-Wert für zeitabhängige Berechnungen verwendet werden. Da alle Klassen des Spiels auf diesen Wert zugreifen können, war eine Anpassung von SheepMeUp innerhalb kürzester Zeit umgesetzt.

- Die Bewegung der Käfer auf dem Boden des Levels waren überhaupt nicht von der Zeit abhängig; die Käfer wurden also je nach Höhe der FPS schneller oder langsamer. Da die Bewegung zeitunabhängig war, konnte das Horde3D Development Environment die Käfer auch nicht anhalten. Das war ein Bug des Spiels, der ebenfalls schnell behoben werden konnte.

- Ein weiterer Fehler steckte in der Sound-Komponente der verwendeten Game Engine der Universität Augsburg. Dort wurde die Geschwindigkeit des *Listeners* unter anderem aus der Differenz zwischen aktueller Zeit und der Zeit des letzten Frames berechnet. Ist die Anwendung angehalten, ist diese Differenz $\Delta t = 0$. Innerhalb der Sound-Bibliothek wurde dann zur Berechnung der Geschwindigkeit des *Listeners* durch $\Delta t = 0$ dividiert, was bei manchen Soundkarten-Treibern zu Abstürzen führte. Dieser Bug der Game Engine konnte durch eine Überprüfung auf den Wert 0 vor dem Funktionsaufruf schnell behoben werden.

Nach Behebung dieser Bugs arbeitete das Horde3D Development Environment problemlos mit SheepMeUp zusammen. Da jedoch aufgrund der Portierung des Spiels auf die aktuelle Horde3D-Version die Special Effects nicht mehr funktionierten, konnte SheepMeUp nicht als Grundlage für die Implementierung eines neuen Effekts mit Hilfe des Horde3D Development Environments dienen.

5.2. Umsetzung eines Special Effects für Raumschiff-Schilde

Für einen Prototyp eines Spiels – ein Raumschiff-Shooter in Top-Down-Ansicht – wurde ein Effekt für das Auftreffen eines Geschosses auf den Schild eines Raumschiffs umgesetzt. Abbildung 5.1 zeigt eine Momentaufnahme des Effekts. Das Raumschiff in der Mitte wurde von rechts von einem Geschoss getroffen. Die beiden blauen Ringe stellen Energiewellen dar, die durch die Absorption des Schusses durch den Schild laufen.

Es stellte sich heraus, dass das Horde3D Development Environment in diesem Szenario seine Stärken sehr gut ausspielen kann. Zunächst wurde der Effekt in das Spiel mit einem Standard-Shader, der alles einfach weiß zeichnete, eingebaut. Anschließend wurde das Spiel mit dem Horde3D Development Environment gestartet und der Effekt implementiert. Während der Umsetzung des Effekts musste das Spiel kein einziges mal neu gestartet oder neu kompiliert werden.

Der Shader des Effekts besteht im wesentlichen aus der Berechnung von Sinus- und Cosinus-Wellen, die in Abhängigkeit von der Treffposition des Geschosses und der seit dem Treffer vergangenen Zeit berechnet werden. Die Wellen werden zusätzlich in Abhängigkeit von der Zeit und von der Entfernung zum Treffpunkt ausgeblendet. Der größte Vorteil konnte dabei aus der sofortigen Anzeige der Änderungen gezogen werden. Beispielsweise konnte direkt nach der Anpassung einer Konstante der geänderte Shader Code an das Spiel übergeben und die

Abbildung 5.1.: Momentaufnahme des Schild-Effekts

Auswirkungen beurteilt werden. Da dies im Bruchteil einer Sekunde ohne Neustart des Spiels möglich ist, konnten sehr schnell gute Ergebnisse erzielt werden. Da das Horde3D Development Environment auch sofort etwaige Probleme beim Kompilieren des Shaders anzeigte, konnten Fehler im Code schnell gefunden werden.

Problematisch bei der Entwicklung von Shadern ist das Debugging. Es gibt zwar Tools wie glslDevil[1] der Universität Stuttgart zum Debuggen von GLSL-Shadern, diese sind aber noch nicht sehr ausgereift. Es bleibt also nur die Möglichkeit zu klassischem „`printf`-Debugging“, angepasst auf Shader: Will man den Wert einer Variable wissen, so muss man die Variable als Ausgabe des Shaders – also die Farbe des Pixels – setzen und sich die Szene ansehen. Aus den Farbinformationen lassen sich dann Rückschlüsse auf den eigentlichen Wert der Variable ziehen. Diese Art des Debuggings wird vom Horde3D Development Environment schnell und unkompiliziert unterstützt und erleichtert es erheblich, fehlerhafte Shader zu debuggen.

Während der Entwicklung des Effekts zeigte sich, dass der Shader eine Textur als Eingabe benötigt. Dazu wurde das Material des Effekts angepasst und dort die Textur als Eingabe für den Shader festgelegt. Da beim Aktualisieren des geänderten Materials im Spiel automatisch auch die neue Textur geladen wurde, war auch hier kein Neustart des Spiels notwendig und die Änderungen konnten sofort beobachtet werden.

Der Effekt sollte schließlich noch durch eine Verzerrung des Raumschiff innerhalb des Schildes verbessert werden. Für eine solche Verzerrung benötigt der Shader zusätzlich die komplett gezeichnete Szene als Eingabe. Mit dem Horde3D Development Environment wurde die Pipeline-Konfiguration des Spiels geändert und ein *Render Target* mit dem Szenen-Inhalt für den Shader bereitgestellt. Dabei war es hilfreich, den Inhalt des *Render Targets* betrachten zu können, um Fehler in der Pipeline-Konfiguration und den Shadern zu beheben. Nach der Erweiterung des Shader Codes konnte der verbesserte Effekt wieder sofort im Spiel betrachtet

[1] `http://www.vis.uni-stuttgart.de/glsldevil`

werden.

Insgesamt erleichtert das Horde3D Development Environment die Entwicklung neuer Effekte hauptsächlich durch das sofortige Anzeigen der Änderungen sowie durch das sofortige Anzeigen von Problemen beim Kompilieren von Shadern, Einlesen von Materials, etc. Aber auch das Ändern von Texturen von Materials sowie der gesamten Pipeline-Konfiguration ist deutlich einfacher. Der Shader-Designer erwies sich gerade beim Einstellen der Kontext-Parameter wie Blend-Modi, Tiefentests, etc. als sehr hilfreich, weil man die entsprechenden Werte nicht auswendig wissen musste. Ein Großteil der Umsetzung des Shaders wurde im Designer erledigt; die zugrundeliegende XML-Datei wurde nur in wenigen Fällen direkt bearbeitet. Der XML-basierten Konfiguration der Pipeline und des Materials wäre ein ähnlicher Designer ebenfalls überlegen gewesen.

5.3. Meinungsumfrage

Es wurden einige Mitglieder der Horde3D-Community gebeten, das Tool zu testen und einen Fragebogen auszufüllen. Zwei Mitglieder des SheepMeUp-Entwicklungsteams, ein Entwickler von Alfred und einer der Entwickler von Horde3D nahmen an der Meinungsumfrage teil. Die Fragen und die Bewertungen der Teilnehmer werden in Abbildung 5.2 aufgelistet.

Zunächst wurde gefragt, wie wichtig den Teilnehmern die einzelnen Anforderungen sind, die an das Horde3D Development Environment gestellt wurden. Dabei fällt auf, dass der Schutz vor unerwünschtem *Reverse-Engineering* insgesamt das unwichtigste Feature ist, aber dennoch für die Teilnehmer von Bedeutung ist. Erwartungsgemäß wurde die Hauptaufgabe des Tools – das Bearbeiten von Ressourcen und das sofortiges Anzeigen der Änderungen – zusammen mit der Kompatibilität zu allen Horde3D-Anwendungen als am wichtigsten betrachtet.

Die Umsetzung der Ressourcenbearbeitung wurde übereinstimmend als sehr erfolgreich betrachtet, während es bei der Kompatibilität laut Meinung der Teilnehmer noch Nachbesserungsbedarf gibt. Darauf wird in Abschnitt 5.4 noch eingegangen.

Insgesamt ist der Gesamteindruck des Tools bei allen Teilnehmern sehr positiv ausgefallen. Dies spiegelt sich nicht nur in der Umsetzung der einzelnen Anforderungen wider, sondern auch an den Bewertungen der Oberfläche und des *Look and Feels* der Anwendung. Die beiden wichtigsten Fragen[2], ob das Tool die Entwicklung von Horde3D-Anwendungen erleichtere und ob sich die Teilnehmer vorstellen könnten, zukünftig das Horde3D Development Environment zur Entwicklung von Horde3D-Anwendungen zu verwenden, wurden einstimmig mit „ja“ beantwortet.

Im Hinblick auf Kapitel 2.4 und die teilweise überlappende Funktionalität des Horde3D Development Environments und des Horde3D Scene Editors wurden die Teilnehmer gefragt, ob sie ein ähnliches Tool kennen und gegebenenfalls auch bevorzugen würden. Die übereinstimmende Meinung war, dass der Szenen-Editor zwar teilweise ähnliche Features biete, aber doch einen grundsätzlich anderen Fokus habe, und man daher beide Tools problemlos parallel einsetzten könne.

Abschließend wurden die Teilnehmer nach Verbesserungsvorschlägen und Ideen für Erweiterungen gefragt. Es gab zwei Antworten, die teilweise in Abschnitt 5.4 noch einmal aufgegriffen werden:

[2] Ja/Nein-Fragen sowie Freitext-Antworten sind in Abbildung 5.2 nicht aufgelistet.

Frage	Antwort #1	Antwort #2	Antwort #3	Antwort #4	Ø
Wie wichtig sind für Sie die folgenden Ziele/Features des Horde3D Development Environments?					
Zusammenarbeit mit allen Horde3D-Anwendungen	2	1	1	1	1.25
Absicherung vor unerwünschtem Reverse-Engineering	2	4	2	3	2.75
Bearbeiten von Ressourcen und sofortiges Anzeigen der Änderungen	1	2	1	1	1.25
Anzeigen von Horde3D-Meldungen	2	2	1	1	1.50
Anzeigen von Render Targets	1	2	2	2	1.75
Profiling von Horde3D-Funktionsaufrufen	3	1	1	3	2.00
Anzeigen des Szenengraphs	2	1	1	1	1.25
Wie gut finden Sie die Umsetzung der folgenden Funktionalitäten?					
Zusammenarbeit mit allen Horde3D-Anwendungen	3	1	1	2	1.75
Absicherung vor unerwünschtem Reverse-Engineering	2	N/A	1	1	1.33
Horde3D-Meldungen anzeigen, sortieren und nach Wichtigkeit filtern	2	2	1	1	1.50
Anzeigen von Render Targets und automatische Aktualisierung der Anzeige	2	1	1	2	1.50
Profiling von Horde3D-Funktionsaufrufen und deren Auswertung	2	1	2	1	1.50
Anzeigen des Szenengraphs	2	1	1	2	1.50
Bearbeiten von Ressourcen und sofortiges Anzeigen der Änderungen	1	1	1	1	1.00
Visueller Designer zum Bearbeiten von Shader-Ressourcen	1	1	1	1	1.00
Wie beurteilen Sie folgenden Punkte der grafischen Benutzeroberfläche des Horde3D Development Environments?					
Wie beurteilen Sie die intuitive Verständlichkeit der Oberfläche?	2	1	1	2	1.50
Wie effizient können Sie mit dem Tool arbeiten?	2	1	1	1	1.25
Wie zufrieden sind Sie insgesamt mit dem „Look and Feel"?	1	1	1	1	1.00
Wie beurteilen Sie die folgenden Punkte?					
Wie beurteilen Sie die Qualität des Tools?	2	1	1	2	1.50
Was ist Ihr Gesamteindruck?	1	1	1	1	1.00

Abbildung 5.2.: Auswertung der Meinungsumfrage

„Ein für mich interessantes Feature wäre noch Overlays während der Laufzeit laden zu können und beliebig über das Horde-Fenster zu positioneren und zu strecken. Die Anordnung soll dann als Snapshot abgespeichert werden können. Dieser „Designer“ für Overlays fehlt mir noch in allen Horde3D Produkten.

Eine weitere Erweiterungsmöglichkeit sehe ich in einem Plugin-System, welches es erlaubt das Horde3D Development Environment um zusätzliche Editoren und Views zu erweitern.

Darüber hinaus wäre es interessant, Texturen prozedural erzeugen zu können und diese während Laufzeit einzusetzen.“

„Ich bin sehr zufrieden mit dem Programm. Für die Zukunft könnte es interessant sein, DLL-Injection-Methoden zu untersuchen um die Kompatibilität zu verbessern. Ansonsten könnte man den Texture Browser etwas ausbauen (Anzeige von DDS).“

5.4. Ausblick

Wie bereits mehrfach angedeutet wurde, gibt es eine Vielzahl an möglichen Erweiterungen und Verbesserungen für das Horde3D Development Environment. In diesem Abschnitt werden einige Punkte vorgestellt und kurz erläutert.

- **Plugin-System.** Ein Plugin-System wurde auch von einem der Umfrageteilnehmer genannt und wäre eine wichtige Erweiterung für die Shell, um zukünftige neue Features und die Kernfunktionalität sauber trennen zu können. Es gibt bereits einige Horde3D-Erweiterungen, wie zum Beispiel für Sound-Ausgabe[3], die mit einem Plugin-System ebenfalls einfach an das Tool angeschlossen werden könnten.

- **Verbesserung des DLL-*Replacement*-Mechanismus.** Der DLL-*Replacement*-Mechanismus sollte in zukünftigen Versionen verbessert werden. Die Kompatibilität mit allen Horde3D-Anwendungen ist eines der wichtigsten Ziele des Horde3D Development Environments. Derzeit funktioniert das DLL-*Replacement* nur bei Verwendung der Original-DLL von Horde3D. Der Code Generator ließe sich prinzipiell aber leicht so erweitern, dass der Code für die Einstiegspunkte beliebiger Horde3D-Extensions ebenfalls generiert wird. Wie ein Teilnehmer an der Meinungsumfrage bereits angedeutet hat, könnten andere *Hooking*-Methoden[4] im Allgemeinen eventuell überlegen sein.

- **„Offline“ Ressourcenverarbeitung.** Das Horde3D Development Environment unterstützt derzeit nur die Bearbeitung von Ressourcen, während oder nachdem eine Horde3D-Anwendung gelaufen ist. Es wäre von Vorteil, auch ohne laufende Anwendung – also quasi „offline“ – Funktionen wie den Shader Designer verwenden zu können.

- **Vernetzung der Daten.** Wie man bereits im Konzeptmodell erkennen konnte, sind die einzelnen Horde3D-Klassen eng miteinander verbunden. Horde3D bietet für fast alle dieser Beziehungen die Möglichkeit an, diese nachträglich auszulesen. Das Horde3D

[3] `http://www.horde3d.org/wiki/index.php5?title=Sound_Extension`
[4] siehe zum Beispiel `http://www.codeplex.com/easyhook/Thread/View.aspx?ThreadId=35209`

Development Environment könnte diese Informationen nutzen, um beispielsweise alle *Scene Nodes* anzuzeigen, die eine gewissen Ressource verwenden, oder umgekehrt. Auch könnten Fehler- und Debug-Meldungen der Engine mit den zugehörigen Ressourcen oder *Scene Nodes* verknüpft werden. Meldungen über eine fehlgeschlagene Shader-Kompilierung könnten per Doppelklick sofort zur fehlerhaften Zeile führen.

- **Ausbau und Ergänzung von Designern.** Wie die Entwicklung des Schild-Effekts zeigte, ist der Shader-Designer eine große Unterstützung beim Entwickeln eines Shaders. Die derzeitige Implementierung hat jedoch noch einige Limitierungen, die behoben werden sollten. So können keine neuen Shader *Sections*, Kontexte, *Sampler* oder *Uniforms* angelegt werden; dies muss in der zugrundeliegenden XML-Datei geschehen. Außerdem werden Kommentare in der XML-Datei vom Designer gelöscht. Ähnlich aufgebaute Designer wären ebenfalls für Pipelines, Materials und Partikel denkbar. Interessant wäre hier auch, die Ressourcen enger zu verknüpfen. Wählt man beispielsweise in einem Material eine Textur für einen *Sampler* aus, so könnte man dies an Stelle der händischen Eingabe des Pfades der Textur über eine *Drop-Down*-Auswahl umsetzen. Die Designer könnten auch überprüfen, ob die Pfade alle korrekt sind, ohne dass dafür extra die Horde3D-Anwendung gestartet werden muss.

- **Einfrieren der Szene.** Das Einfrieren der Szene ermöglicht es, zeitabhängige Effekte in einem festen Zustand bearbeiten zu können. Hier wäre es von Vorteil, durch Verlangsamen oder Beschleunigen der Zeit der Horde3D-Anwendung schneller oder präziser zu der Stelle des Effekts zu kommen, die gerade relevant ist. Im eingefrorenen Zustand würde außerdem eine *Free-Look* Kamera die Betrachtung des eingefrorenen Effekts von allen Richtungen möglich machen. Wie bereits in Kapitel 4.4.1 ausgeführt, ist das Anhalten der Zeit problematisch. Hier müsste nach weiteren Lösungsmöglichkeiten gesucht werden. Eventuell muss das komplette Anhalten der Threads doch implementiert werden, um Client-Server-Anwendungen einfrieren zu können. Aufgrund der Verwendung des *Strategy Patterns* wäre auch eine Auswahl der verwendeten Algorithmen durch den Benutzer pro Horde3D-Anwendung leicht umsetzbar.

- **Ausbau der Profiling-Funktionalität.** Die Profiling-Funktionalität wurde von den Umfrageteilnehmern als unwichtigstes Feature betrachtet, lässt aber dennoch Raum für einige interessante Erweiterungsmöglichkeiten. So werden bei einem Funktionsaufruf bereits der Rückgabewert und die Werte der Funktionsparameter protokolliert. Die Shell könnte diese Daten anzeigen, damit der Benutzer einen besseren Überblick über die Horde3D-Funktionsaufrufe erhält. Außerdem könnten dadurch Funktionsaufrufe mit den Ressourcen oder den *Scene Nodes* verknüpft werden, die als Parameter übergeben wurden. Auch wäre es interessant, Horde3D-Fehlermeldungen dem Funktionsaufruf zuzuordnen, der den Fehler erzeugt hat.

- **Integration oder Anbindung von Third-Party-Anwendungen.** Es gibt *Third-Party*-Anwendungen, deren Integration oder enge Verknüpfung mit dem Horde3D Development Environment die Produktitivät des Benutzers erhöhen könnten. Denkbar wäre beispielsweise eine Anbindung an glslDevil, um Shader debuggen zu können; eine Integration von SVN, um geänderte Ressourcen gleich in das *Repository* einchecken zu können; oder die Anbindung von Tools wie der Collada Converter von Horde3D oder FX Composer und RenderMonkey.

5.5. Konklusion

Die Anforderungen an das Horde3D Development Environment wurden ausgehend von den Erfahrungen bei der Entwicklung von SheepMeUp formuliert und in einem *Unified Process*-ähnlichen Vorgehen systematisch umgesetzt. Die detaillierte Analyse der Anforderungen und der Horde3D-Engine führten zu einem umfangreichen, aber flexiblen Design, das schließlich schnell und weitestgehend problemlos implementiert werden konnte. Dabei wurde eine Vielzahl an Problemen gelöst und generische Frameworks für die GUI-Anbindung und die Code Generierung entwickelt.

Durch die Umsetzung eines Schild-Effekts und einer Umfrage unter mehreren Horde3D-Community-Mitgliedern konnte auch bestätigt werden, dass die Anforderungen nicht nur ein Spezialfall der SheepMeUp-Entwicklung waren, sondern auch für andere Anwender und Anwendungen interessant sind. Die Verwendung des Horde3D Development Environments führt zu einem teilweise erheblichen Produktivitätsgewinn bei der Entwicklung von Special Effects für Horde3D-Anwendungen. Davon profitieren jedoch nicht nur die Entwickler, die Zeit und Geld sparen, sondern vor allem auch deren Kunden und Benutzer, die in kürzerer Zeit ein qualitativ besseres Produkt erhalten. Es zeigte sich aber auch das Potential, das in zukünftigen Versionen des Horde3D Development Environments steckt; die Liste der möglichen Erweiterungen ist umfangreich und viele der Punkte würden die Produktivität um einen weiteren Faktor erhöhen.

Beim Entwurf und der Konzeption des Horde3D Development Environments wurde oft an zukünftige Erweiterungen gedacht und bereits einige Vorkehrungen getroffen. Zusätzlich wurden Änderungen berücksichtigt, die an Horde3D geplant sind oder möglicherweise eines Tages kommen werden.

In der nächsten Version von Horde3D wird sich das Shader-System ein weiteres mal – wenn auch nicht so umfangreich wie beim Umstieg auf Beta 3 – ändern, um eine HLSL/CG-ähnlichere Syntax zu unterstützen. Aufgrund des modularen Designs des Tools wird diese Umstellung aber nur einige wenige lokale Änderungen an der Shell erfordern. Änderungen der Horde3D-API können mit Hilfe des Update-Mechanismus des Code Generators innerhalb von wenigen Sekunden übernommen werden.

Die nächste Horde3D-Version wird zusätzlich einige interessante Erweiterungen zum Abfragen von Ressourcen-Verknüpfungen und -Daten beinhalten, von denen einige geplante Erweiterungen des Horde3D Development Environments profitieren könnten. Wenn die immer wieder aufkommende Lizenzdiskussion[5] endgültig geklärt ist, besteht außerdem die Hoffnung auf einen neuen Extension-Mechanismus für Horde3D. Erweiterungen müssten dann nicht mehr statisch in die Horde3D DLL gelinkt werden, sondern könnten als DLLs veröffentlicht werden. Das löst zumindest teilweise das DLL-*Replacement*-Problem und vereinfacht die Entwicklung von Horde3D Development Environment-Plugins für Horde3D-Extensions.

Insgesamt hat das Horde3D Development Environment das Potential, integraler Bestandteil der Toolsuite eines jeden Horde3D-Anwendungsentwicklers zu werden und die Bekanntheit und Beliebtheit von Horde3D in der Open Source-Welt weiter zu erhöhen.

[5] `http://www.horde3d.org/forums/viewtopic.php?f=1&t=744&hilit=license`

A. Inhalt der beiliegenden CD-ROM

Auf der beiliegenden CD-ROM dieser Bachelorarbeit befinden sich einige der in der Arbeit referenzierten Dokumente und Anwendungen sowie der Source Code des Horde3D Development Environments.

- `Bachelorarbeit.pdf` ist diese Bachelorarbeit im .pdf-Format.
- `Designmodell.pdf` enthält das Designmodell des Horde3D Development Environments, das aufgrund seiner Größe in der ausgedruckten Arbeit nicht abgebildet werden konnte.
- In der Datei `Fragebögen.pdf` sind alle vier ausgefüllten Fragebögen (anonym) aufgelistet.
- `Konzeptmodell.pdf` enthält das Konzeptmodell des Horde3D Development Environments.
- Im Verzeichnis `Horde3D DevEnv` befindet sich der *Installer* für das Horde3D Development Environment. Die Installationsroutine wird durch das Ausführen der Datei `Horde3D DevEnv.msi` gestartet. Alle benötigten *Dependencies* (.NET 3.5 SP1 und Visual Studio *Redistributable* x86) sollten automatisch installiert werden. Nach der Installation kann das Horde3D Development Environment sofort gestartet werden. Es liegt das *Knight Sample* aus dem Horde3D SDK bei, das bereits vorkonfiguriert ist und gestartet werden kann.
- Im Verzeichnis `lwar` befindet sich ein *Debug Build* des Raumschiff-Spiels aus Kapitel 5.2. In dieser Version kann das Raumschiff nicht bewegt werden; durch Drücken der Tasten j, h, g, z können Treffer von rechts, unten, links und oben simuliert werden. Der Shader `shields.shader.xml`, der im Rahmen von Kapitel 5.2 entwickelt wurde, ist im Verzeichnis `lwar/Content/effects/shields` zu finden.
- Ein *Debug Build* des SheepMeUp-Ports auf Horde3D 1.0.0 Beta 3 befindet sich im Verzeichnis `SheepMeUp`. Wie in Kapitel 5.1 beschrieben, ist diese Version des Spiels zwar lauffähig, aber nicht spielbar. Gestartet werden kann das Spiel über die Datei `SheepMeUp.bat`.
- Das Verzeichnis `Source Code` enthält den kompletten Source Code des Horde3D Development Environments. Durch Öffnen der *Solution* `Bachelorarbeit.sln` kann das Horde3D Development Environment mit Visual Studio 2008 SP1 bearbeitet werden. Zum Kompilieren, Debuggen und Ausführen der Anwendung sind die Hinweise aus Kapitel 4.5 zu beachten.
- Im Verzeichnis `Websites` befinden sich Kopien der im Rahmen der Bachelorarbeit besuchten Websites. Das jeweilige Aufrufsdatum stimmt mit dem im Literaturverzeichnis angegebenen Datum überein.

B. Artefakte der Analyse-Phase

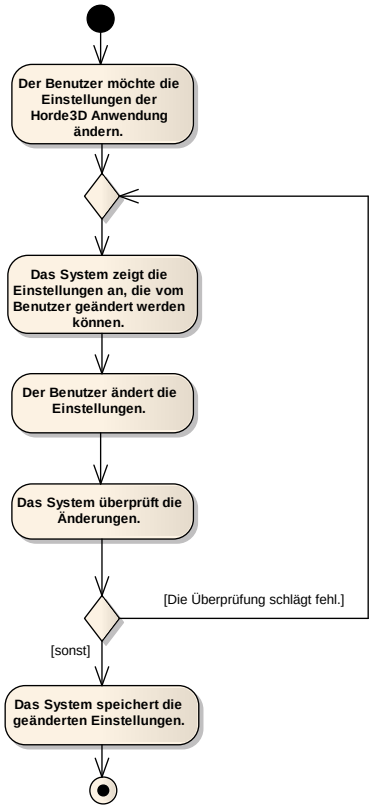

Abbildung B.1.: Aktivitätsdiagramm für den Anwendungsfall „Einstellungen ändern“

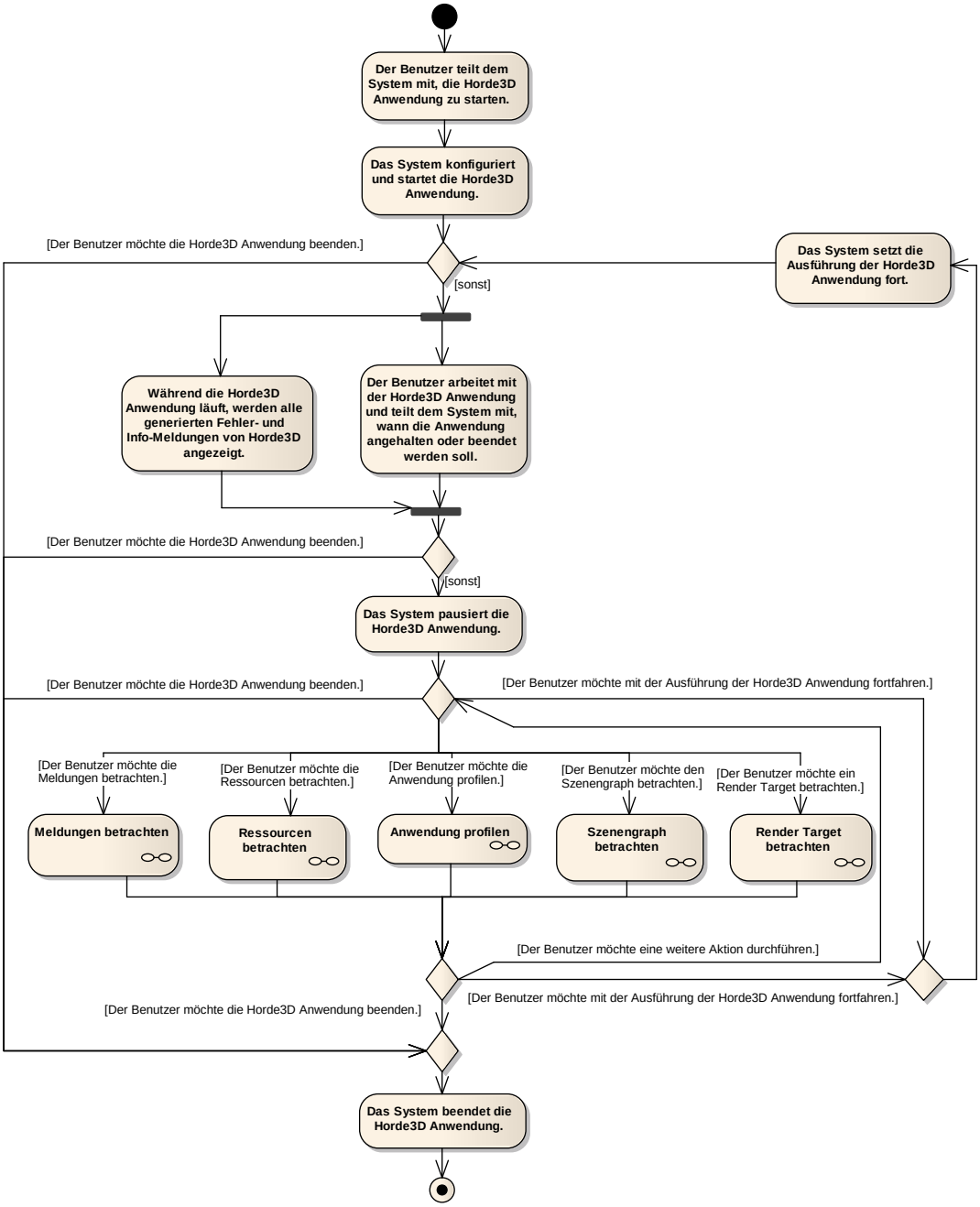

Abbildung B.2.: Aktivitätsdiagramm für den Anwendungsfall „Anwendung ausführen“

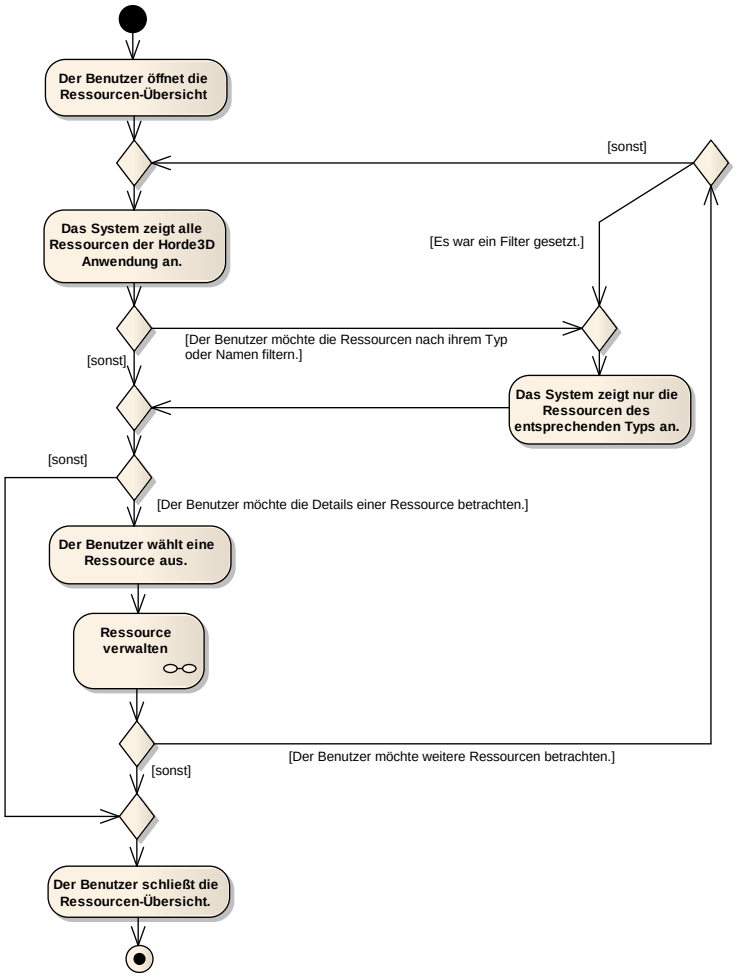

Abbildung B.3.: Aktivitätsdiagramm für den Anwendungsfall „Ressourcen betrachten“

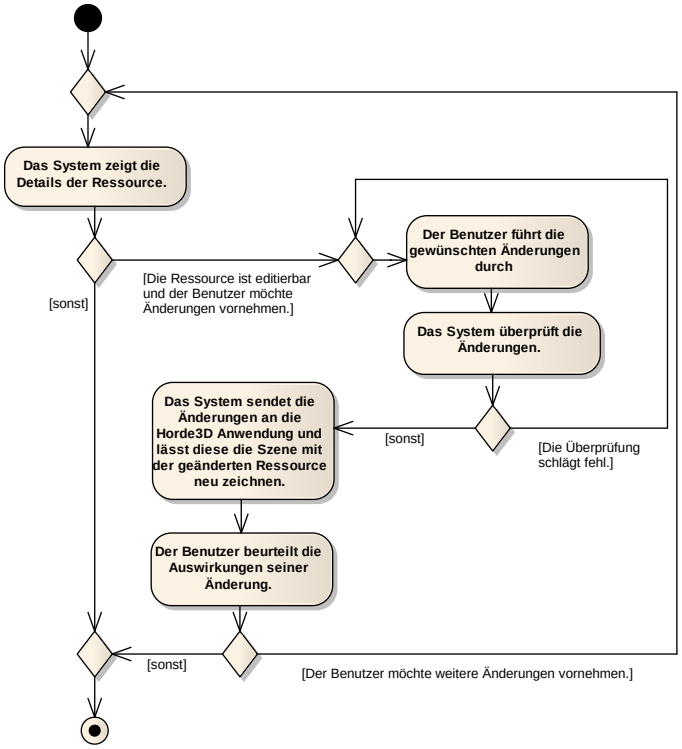

Abbildung B.4.: Aktivitätsdiagramm für den Anwendungsfall „Ressource verwalten“

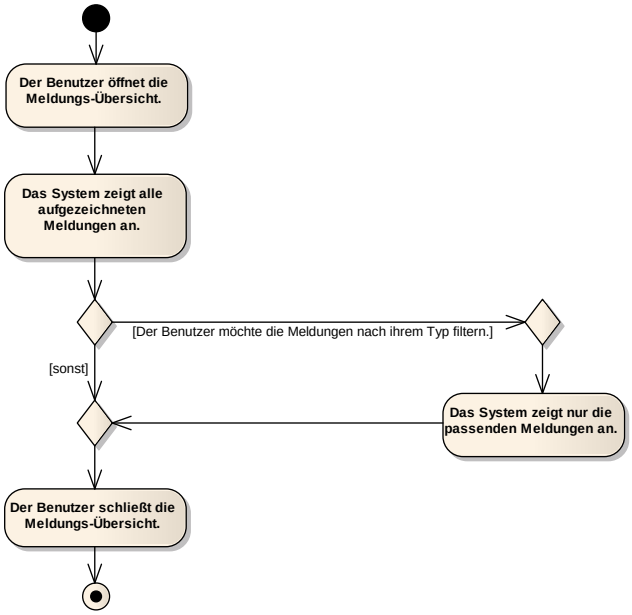

Abbildung B.5.: Aktivitätsdiagramm für den Anwendungsfall „Meldungen betrachten“

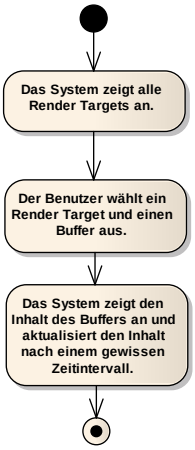

Abbildung B.6.: Aktivitätsdiagramm für den Anwendungsfall „Render Target betrachten“

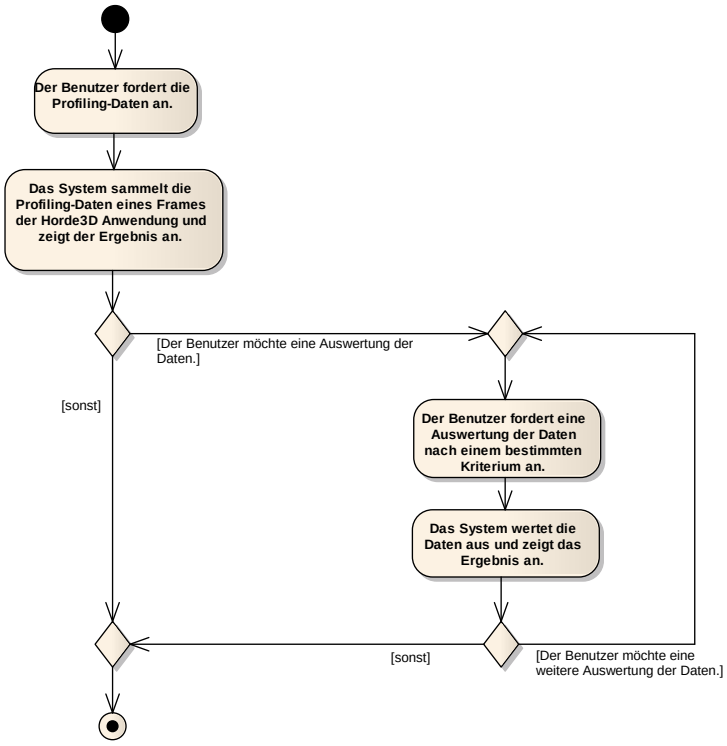

Abbildung B.7.: Aktivitätsdiagramm für den Anwendungsfall „Anwendung profilen“

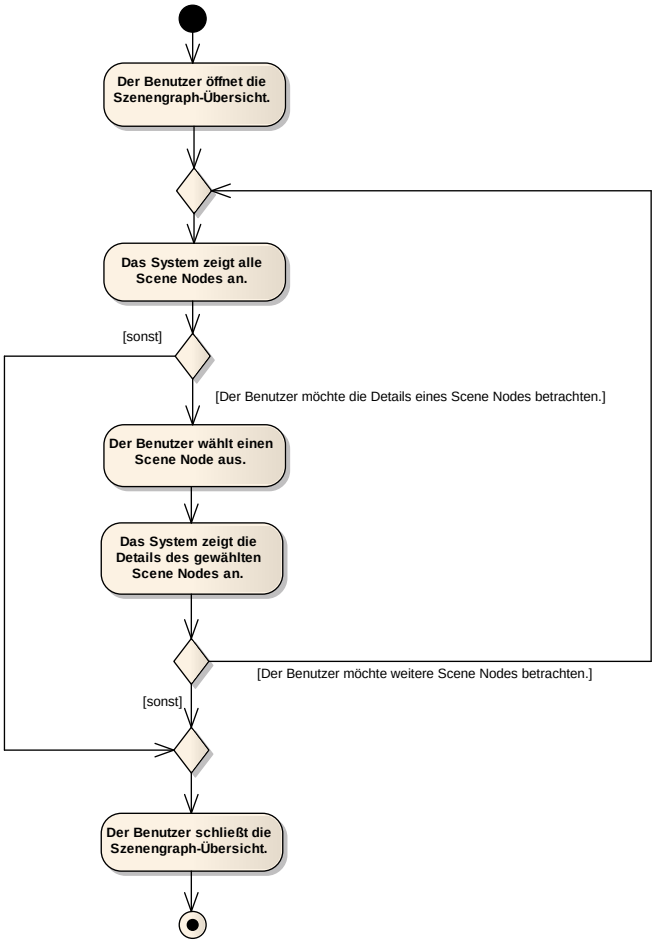

Abbildung B.8.: Aktivitätsdiagramm für den Anwendungsfall „Szenengraph betrachten“

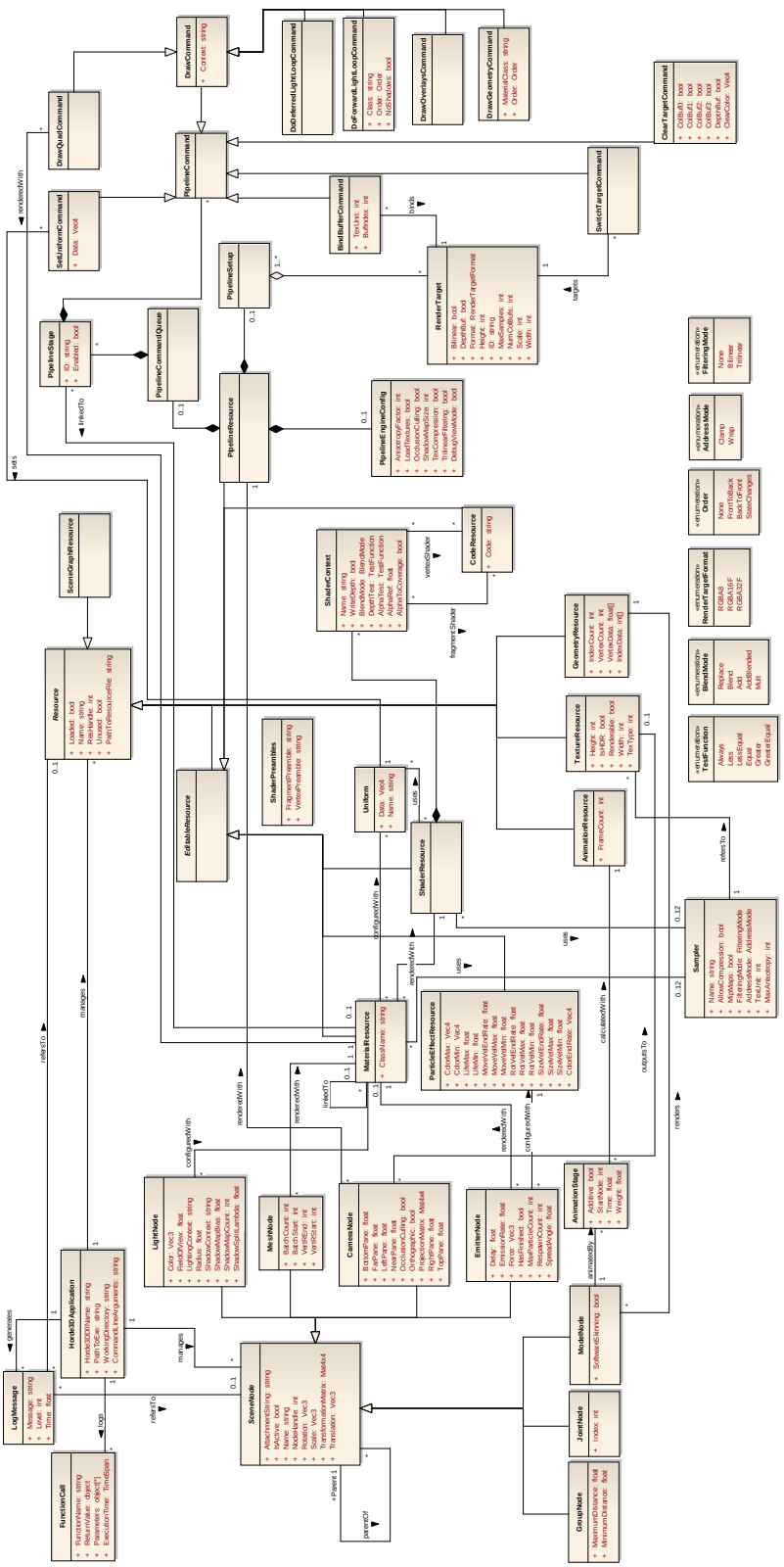

Abbildung B.9.: Das Konzeptmodell des Horde3D Development Environments

C. Artefakte der Design-Phase

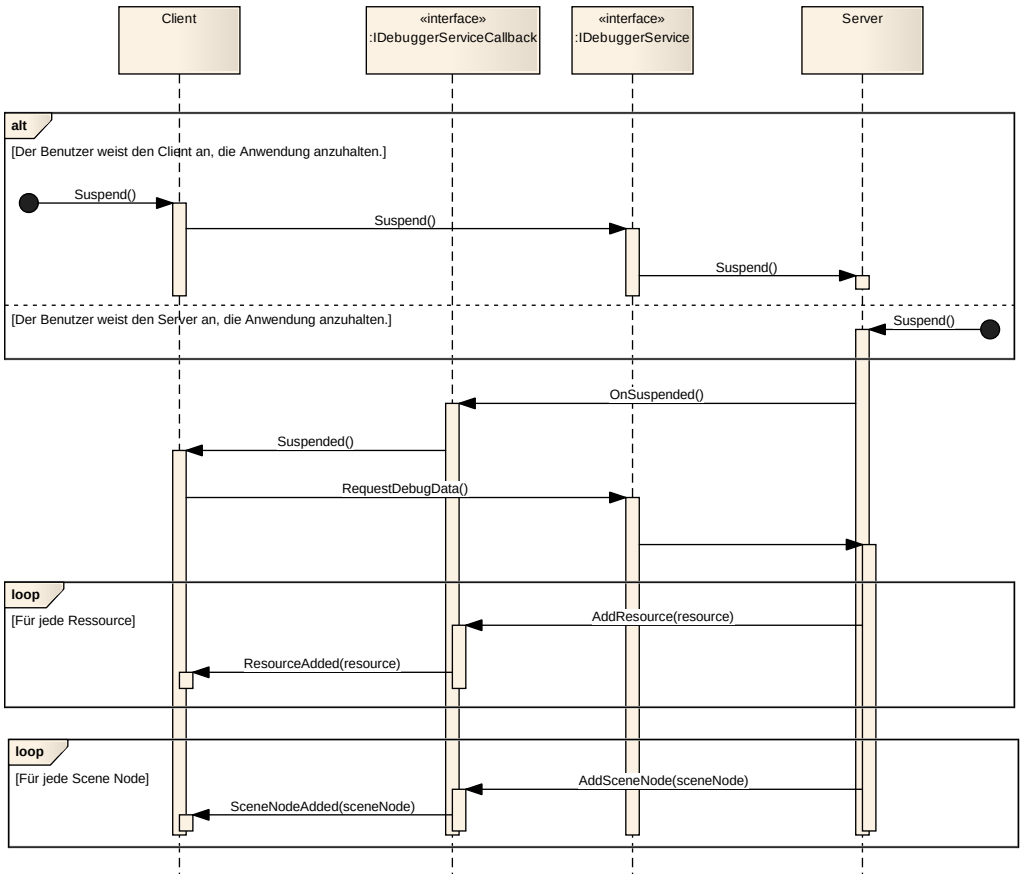

Abbildung C.1.: Sequenzdiagramm für die Client-Server-Interaktionen beim Anhalten der Anwendung

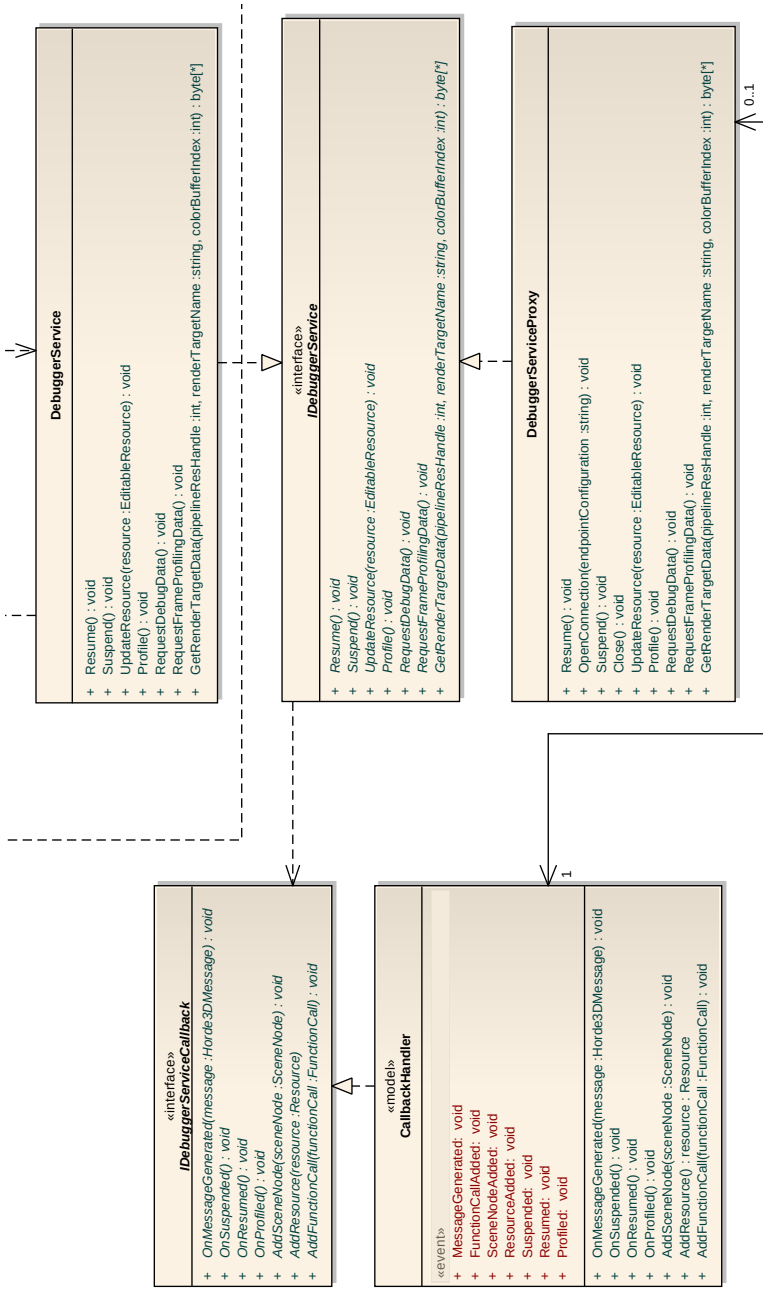

Abbildung C.2.: Ausschnitt aus dem Designmodell für die Client-Server-Schnittstelle

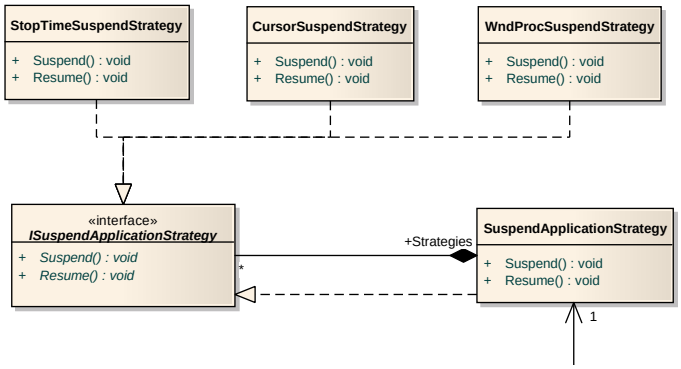

Abbildung C.3.: Ausschnitt aus dem Designklassen-Diagramm für die Anwendung des *Strategy Patterns* zum Anhalten der Anwendung

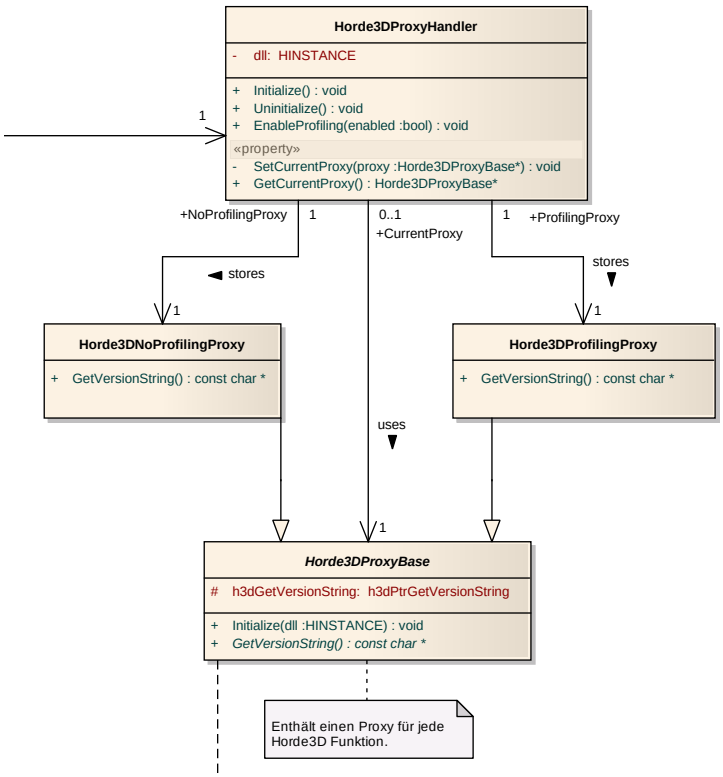

Abbildung C.4.: Ausschnitt aus dem Designklassen-Diagramm für die Horde3D-*Proxies*

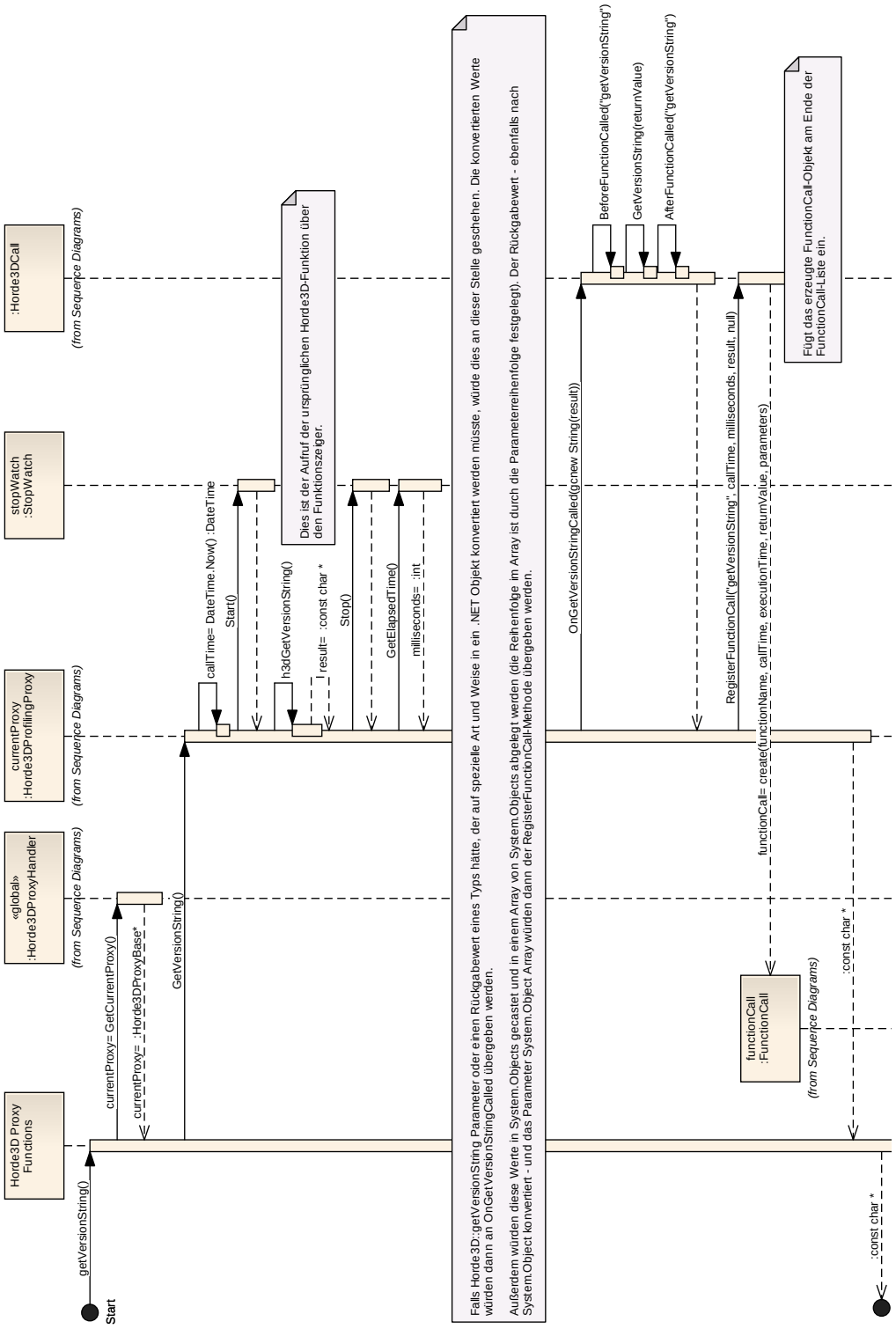

Abbildung C.5.: Sequenzdiagramm für den Aufruf von `Horde3D::getVersionString` bei aktiviertem Profiling

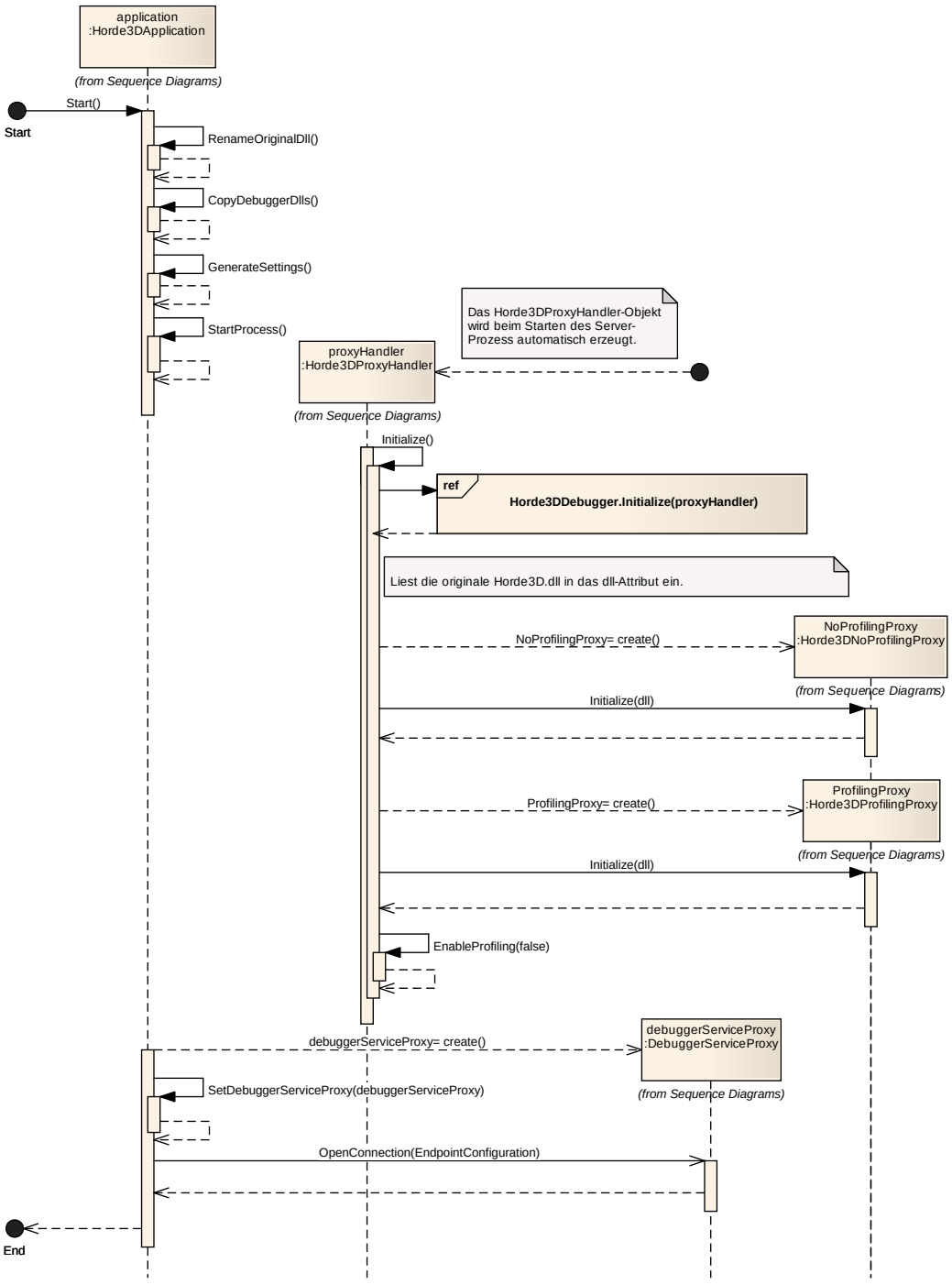

Abbildung C.6.: Sequenzdiagramm für `Horde3DApplication.Start`

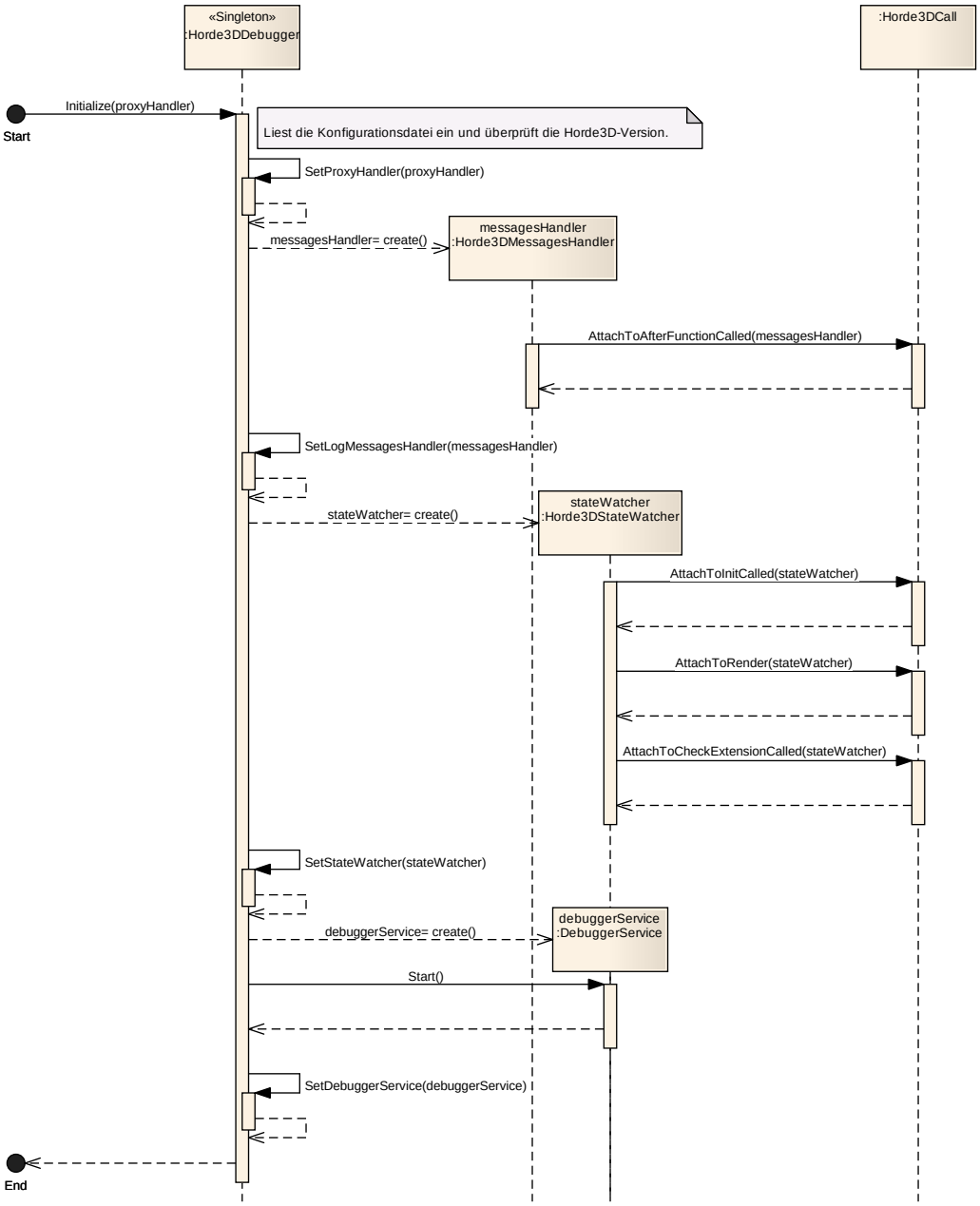

Abbildung C.7.: Sequenzdiagramm für `Horde3DDebugger.Initialize`

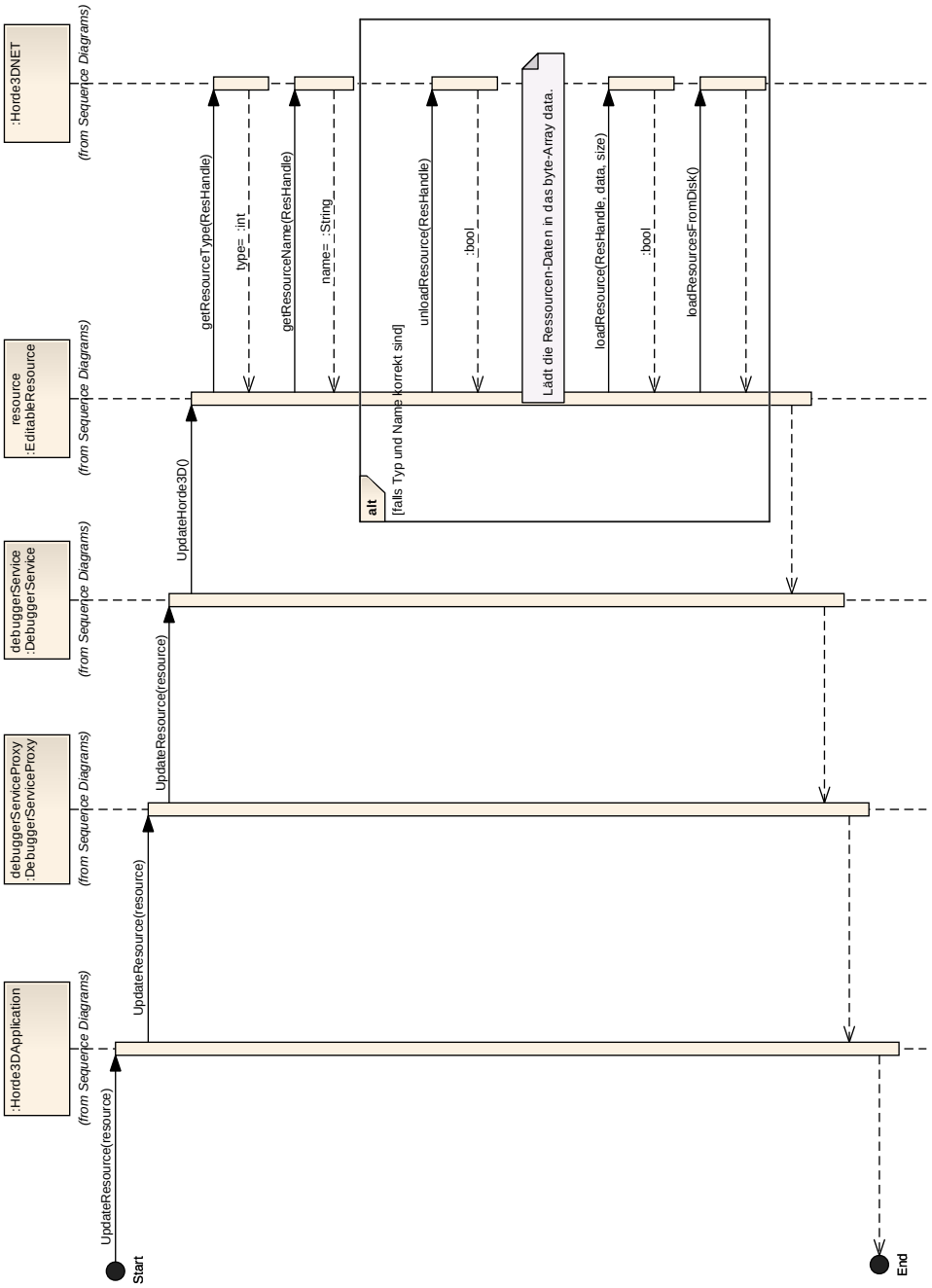

Abbildung C.8.: Sequenzdiagramm für `Horde3DApplication.UpdateResource`

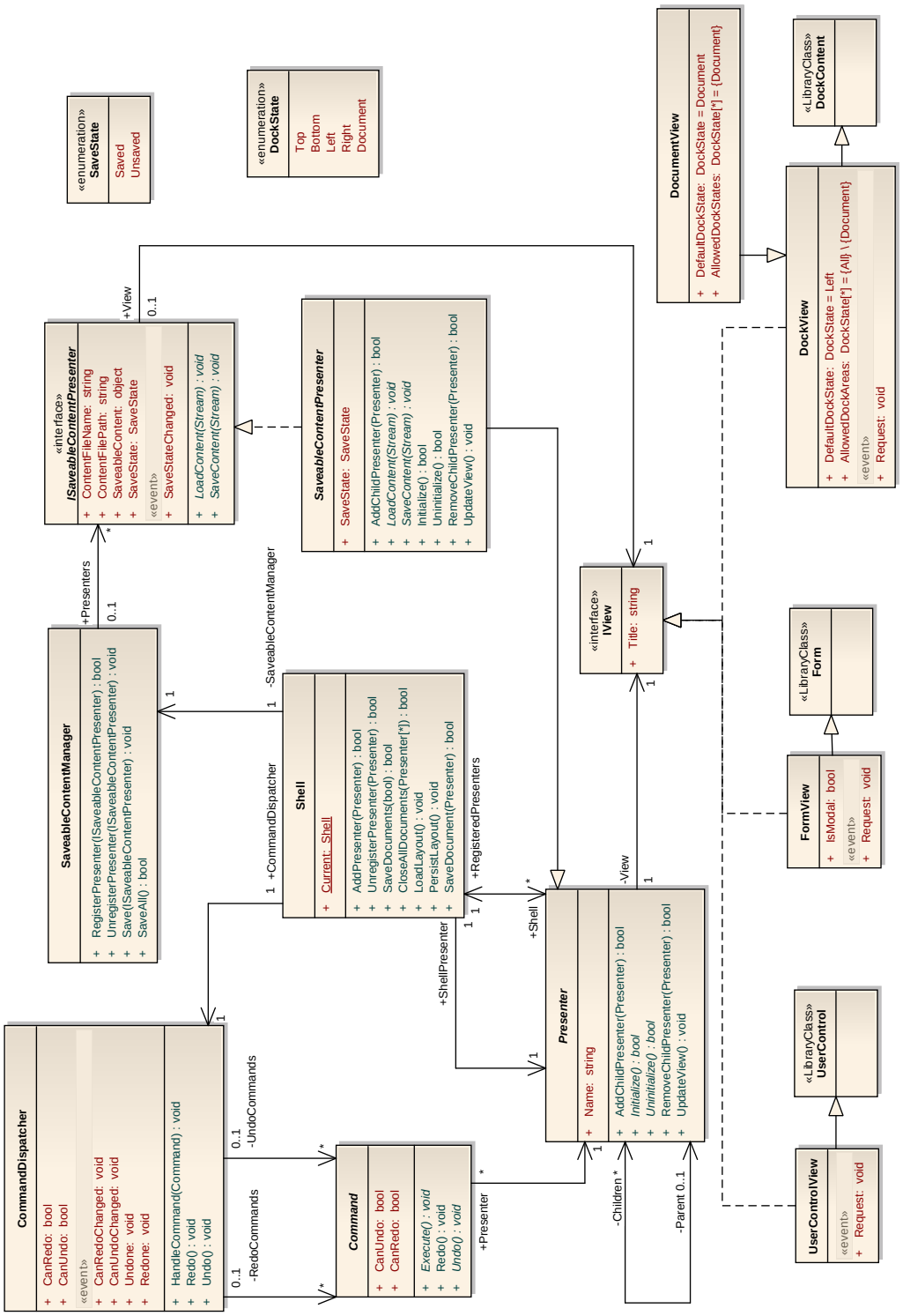

Abbildung C.9.: Designklassen-Diagramm für das GUI-Framework

D. Artefakte der Implementierungs-Phase

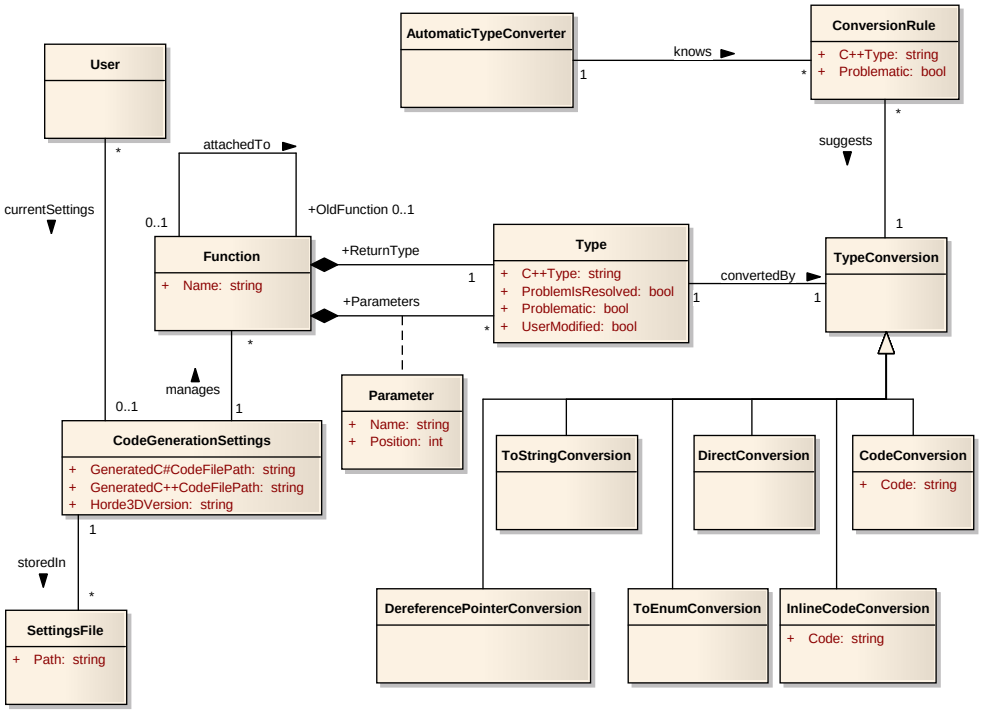

Abbildung D.1.: Das Konzeptmodell des Code Generators

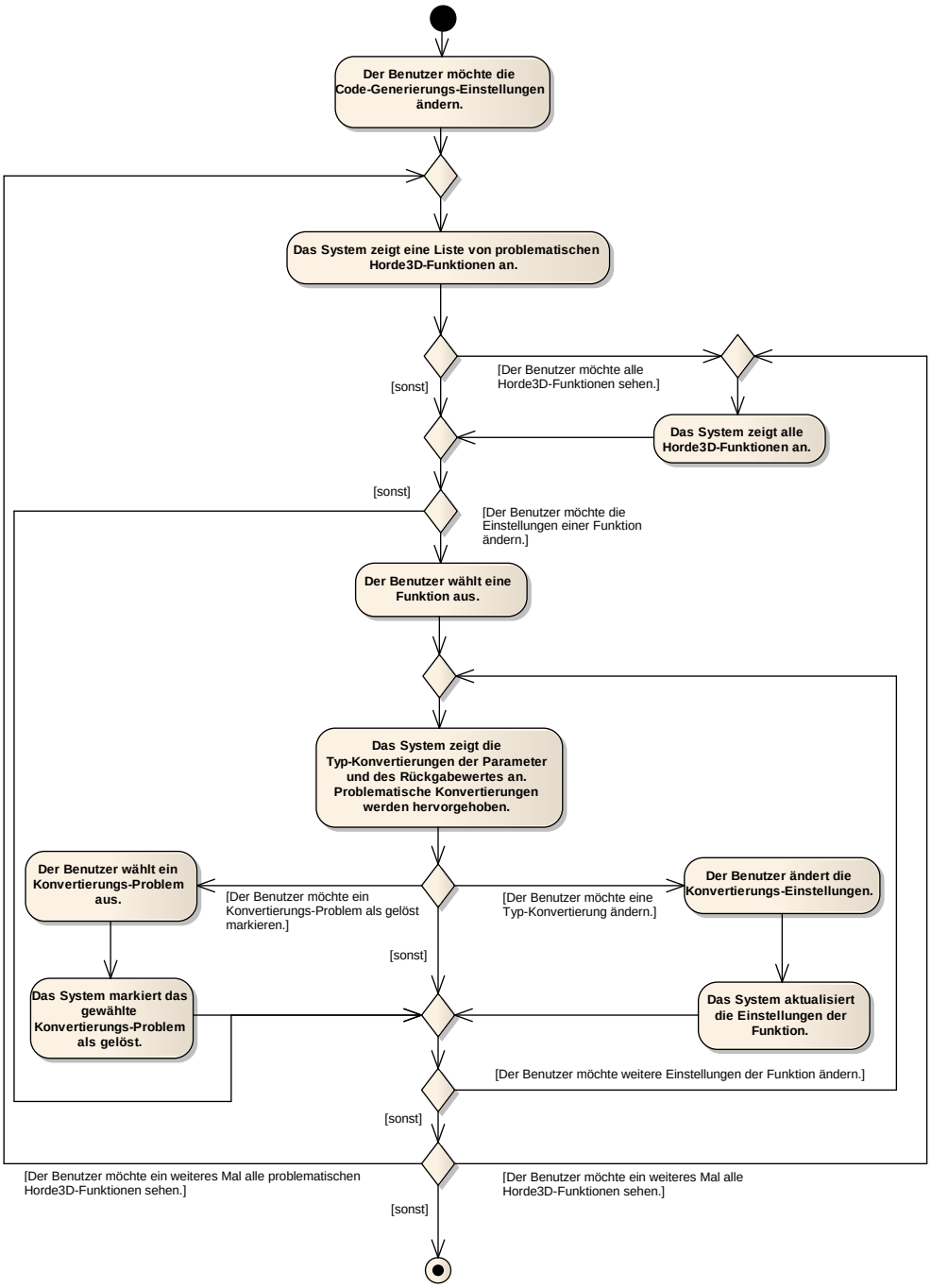

Abbildung D.2.: Aktivitätsdiagramm für den Anwendungsfall „Ändern der Einstellungen der Code-Generierung“

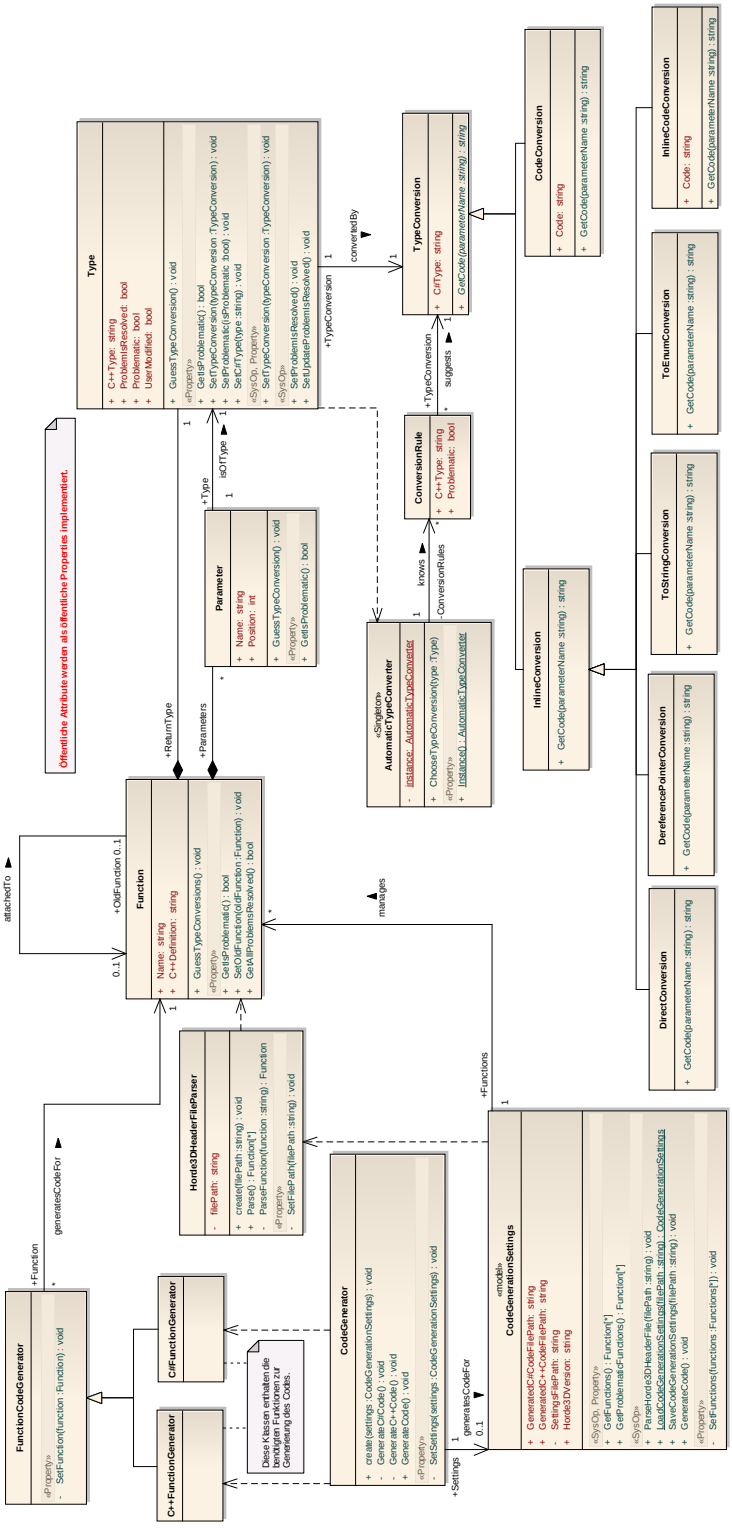

Abbildung D.3.: Das Designmodell des Code Generators

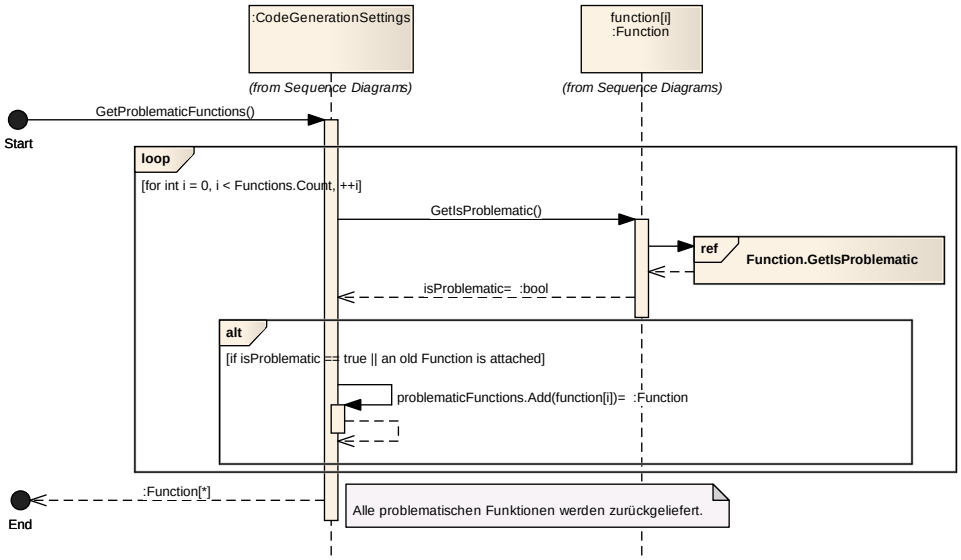

Abbildung D.4.: Sequenzdiagramm zum Auslesen aller problematischer Funktionen

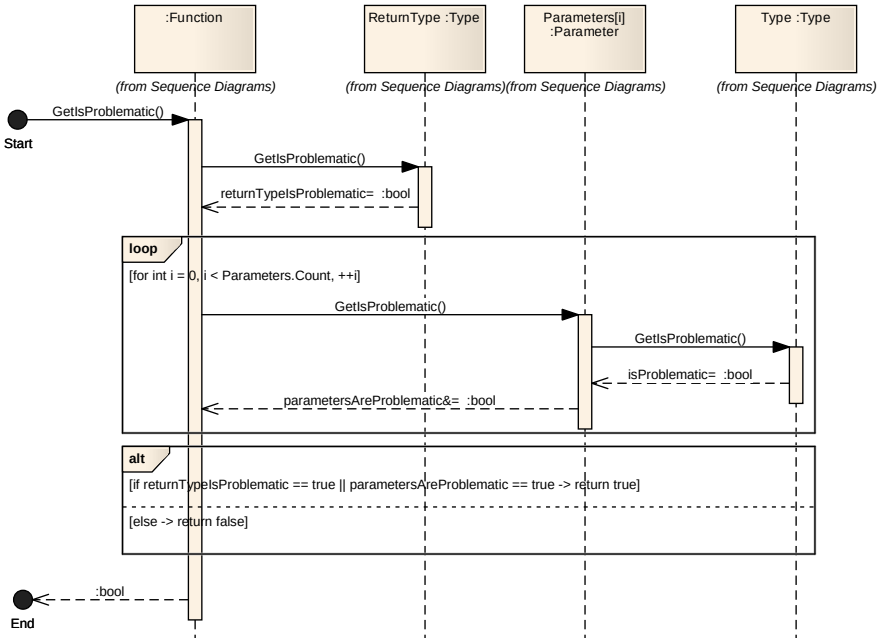

Abbildung D.5.: Sequenzdiagramm für das *Property* `Function.IsProblematic`

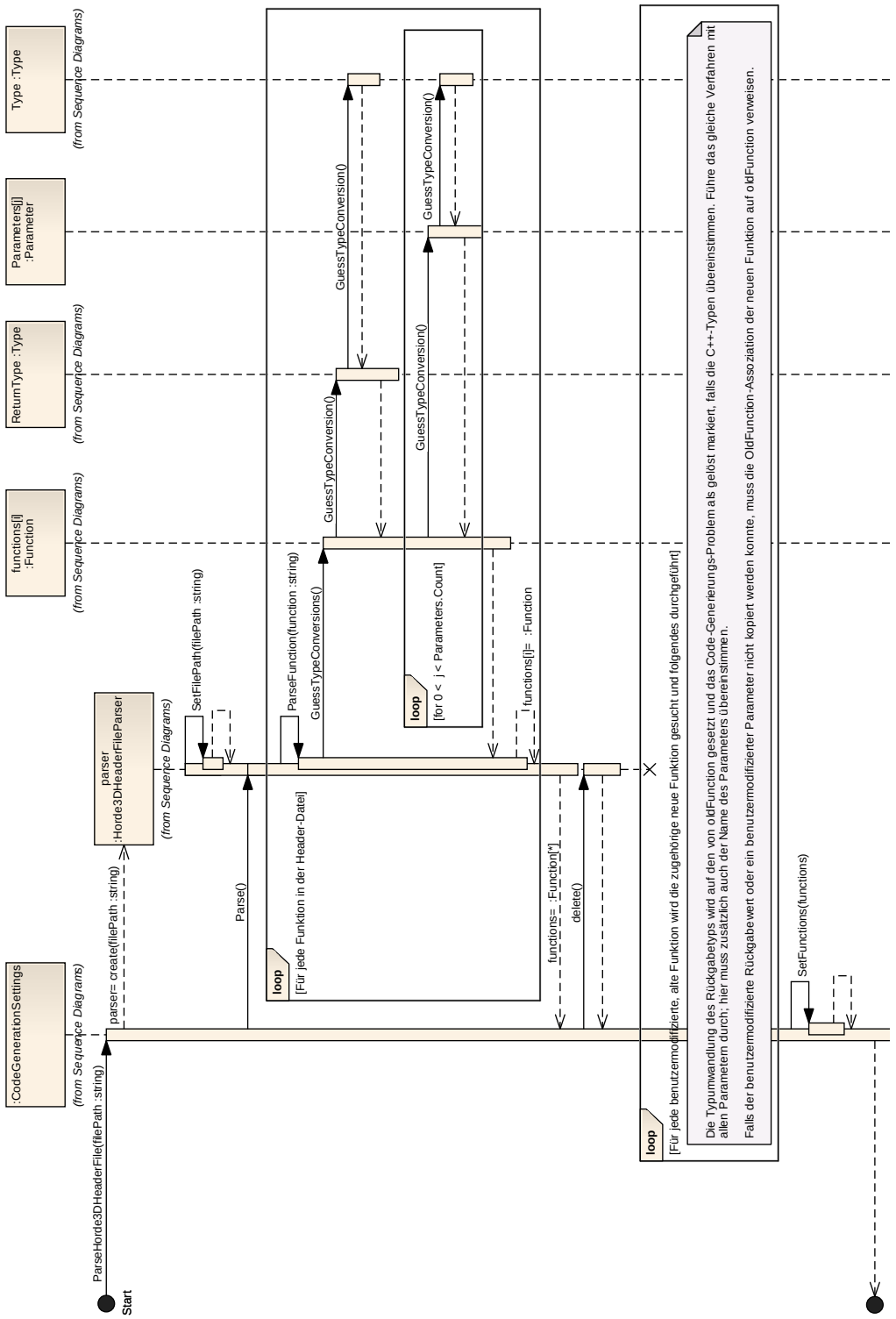

Abbildung D.6.: Sequenzdiagramm für das Parsen der *Header*-Datei

E. Abbildungsverzeichnis

F. Literaturverzeichnis

[1] By Yo Takatsuki. *Cost headache for game developers*, 2007. http://news.bbc.co.uk/1/hi/business/7151961.stm, abgerufen am 25.04.2009.

[2] Wikipedia. *Braid (video game)*, 2009. http://en.wikipedia.org/wiki/Braid_(video_game), abgerufen am 25.04.2009.

[3] Horde3D Development Community. *Horde3D Documentation*, 2009. enthalten im Horde3D 1.0.0 Beta 3 SDK.

[4] Jesse Schell. *The Art of Game Design: A Book of Lenses*. Morgen Kaufmann Publishers, 2008.

[5] Microsoft Corporation. *DirectX Software Development Kit (March 2009)*, 2009. http://www.microsoft.com/downloads/details.aspx?FamilyID=24a541d6-0486-4453-8641-1eee9e21b282.

[6] The Khronos Group Inc. *OpenGL 3.1 Specification*, 2009. http://www.opengl.org/registry/doc/glspec31.20090324.pdf.

[7] Dave Astle. *More OpenGL Game Programming*. Thomson Course Technology PTR, 2006.

[8] Horde3D Wiki. *Horde3D Scene Editor*, 2009. http://www.horde3d.org/wiki/index.php5?title=Horde3D_Scene_Editor, abgerufen am 29.04.2009.

[9] NVIDIA Corporation. *NVIDIA PerfHUD 6 User Guide*, 2008. http://developer.download.nvidia.com/tools/NVPerfKit/6.5/PerfHUD6-UserGuide.pdf, abgerufen am 01.05.2009.

[10] Andrei Alexandrescu. *Modern C++ Design - Generic Programming and Design Patterns Applied*. Addison-Wesley, 2006. 14. Auflage.

[11] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 2005. 32. Auflage.

[12] Wikipedia. *GRASP (Object Oriented Design)*, 2009. http://en.wikipedia.org/wiki/GRASP_(Object_Oriented_Design), abgerufen am 10.05.2009.

[13] Rich Newman. *Model-View-Presenter: Variations on the Basic Pattern (Introduction to CAB/SCSF Part 24)*, 2008. http://richnewman.wordpress.com/2008/02/26/model-view-presenter-variations-on-the-basic-pattern-introduction-to-cabscsf-part-24, abgerufen am 09.05.2009.

[14] Martin Fowler. *Supervising Controller*, 2006. `http://martinfowler.com/eaaDev/SupervisingPresenter.html`, abgerufen am 09.05.2009.

[15] Microsoft Corporation. *Field Usage Guidelines*, 2009. `http://msdn.microsoft.com/en-us/library/ta31s3bc(VS.71).aspx`, abgerufen am 12.05.2009.

[16] Christian Holm, Mike Krüger, and Bernhard Spuida. *Dissecting a C#Application - Inside SharpDevelop*. Apress, 2003.

[17] Wikipedia. *.NET Framework*, 2009. `http://en.wikipedia.org/wiki/.NET_Framework`, abgerufen am 15.05.2009.

[18] Wikipedia. *Windows Forms*, 2009. `http://en.wikipedia.org/wiki/Windows_Forms`, abgerufen am 15.05.2009.

[19] Juval Löwy. *Programming WCF Services*. O'Reilly Media Inc., 2007.

[20] Fabrice Marguerie, Steve Eichert, and Jim Wooley. *LINQ in Action*. Manning Publications, 2008.

[21] Bjarne Stroustrup. *Die C++ Programmiersprache*. Addison-Wesley, 2000. 4. Auflage.

[22] Justin McElroy. *Joystiq interview: Blow unravels Braid in postmortem*, 2008. `http://www.joystiq.com/2008/09/25/joystiq-interview-blow-unravels-braid-in-post-mortem`, abgerufen am 25.04.2009.

[23] Jonathan Blow. *Braid is the highest-rated XBLA game ever. (Also, sales data)*, 2008. `http://braid-game.com/news/?p=303`, abgerufen am 25.04.2009.

[24] Galen Hunt and Doug Brubacher. *Detours: Binary Interception of Win32 Functions*, 1999. `http://research.microsoft.com/apps/pubs/default.aspx?id=68568`.