

Future Office: A Networked Mixed Reality Framework

THOMAS SEIFRIED

BAKKALAUREATSARBEIT

Nr. 03-1-0238-160-B

eingereicht am
Fachhochschul-Bakkalaureatsstudiengang

MEDIEN-TECHNIK UND -DESIGN

in Hagenberg

im Jänner 2005

Diese Arbeit entstand im Rahmen des Gegenstands

Projektentwicklung

im

Wintersemester 2004/05

Betreuer:

Dr. Michael Haller

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 19. Januar 2005

Thomas Seifried

Inhaltsverzeichnis

Erklärung	iii
Danksagungen	vi
Kurzfassung	vii
Abstract	viii
1 Einleitung	1
1.1 Ausgangspunkt	1
1.2 Mixed Reality	1
1.3 Weitere ähnliche Projekte	3
1.3.1 Office of the future	3
1.3.2 Studierstube	4
2 Konzept	6
2.1 Dimensionen – 2D oder 3D?	6
2.2 Hyperdragging	6
2.3 Mausvektor	7
2.4 Privatsphäre	7
2.5 Dynamik	7
2.6 Zusammenarbeit	8
2.7 Easy to use	8
2.8 Hardware	8
2.9 Setup – Aufbau einer Anwendung im Future Office	9
3 Prototyp – der erste Test	10
3.1 Allgemeines	10
3.2 Ziele	10
3.3 Basis – Grundlegende Bibliotheken	10
3.4 Struktur – Der Aufbau des Prototyps	11
3.4.1 Rendering	11
3.4.2 Tracking	12
3.4.3 Netzwerk	12

3.4.4	Datenhaltung	13
3.5	Conclusio	14
3.5.1	Performance-Probleme am Server	15
3.5.2	Kein Tracking	15
3.5.3	Mangelhafte Erweiterungsfähigkeit	15
3.6	Rückmeldungen	15
4	Re-Design – Plugin System	17
4.1	Ausgangssituation	17
4.2	Zielsetzung	18
4.3	Konzeption	18
4.3.1	Was sind Objekte?	18
4.3.2	Client / Server	19
4.3.3	Warum Plugins?	19
4.4	Basis – Grundlegende Bibliotheken	19
4.5	Struktur – Der Aufbau des Systems	20
4.6	Future Office Base – Der Kern	21
4.6.1	MainManager – Steuerung	21
4.6.2	RessourceManager – Ressourcen-Verwaltung	21
4.6.3	Network – Netzwerkmodul	22
4.6.4	ApplicationContext und Szenegraph	24
4.7	Peripherie	26
4.7.1	Serialisierung	26
4.7.2	Event-System	28
4.8	Plugins – Die Schnittstellen zur Außenwelt	29
4.8.1	Application – Anwendungs-Plugin	30
4.8.2	InputDevice – Eingabegeräte-Plugin	31
4.8.3	Object – Objekt-Plugin	32
4.9	Status quo	33
5	Ausblick	35
A	HOWTOS	36
A.1	Application Plugins	36
A.1.1	Hello World!	37
B	Diagramme	40
	Literaturverzeichnis	41

Danksagungen

Danke an alle, die uns bei der Entwicklung des Future Office unterstützt und weitergeholfen haben. Im besonderen aber an Dipl.-Ing. Jürgen Zauner, der viele Stunden mit uns beim Debuggen verbracht hat und uns einige seiner Bibliotheken zur Verfügung gestellt hat, weiters Dipl.-Ing. Dr. Michael Haller, der die Idee für dieses Projekt hatte und uns bei der Entwicklung vorangetrieben hat, sowie Dipl.-Ing. Dr. Christoph Schaffer, der uns zahlreiche Hardware besorgt hat.

Kurzfassung

Diese Arbeit beschreibt das Design und die Entwicklung einer Umgebung für das intuitive Austauschen von Daten zwischen Computern. Dieser Austausch geschieht über einen Tisch, um der der Bildschirm der Laptops erweitert wird. Daten, repräsentiert durch virtuelle 3D Objekte, können über die Bildschirmgrenzen des Laptops hinaus, auf den Tisch gezogen werden. Es wird aber darauf geachtet, dass die Privatsphäre eingehalten wird. Da der Laptopbildschirm nur dem einzelnen Betrachter zugewendet ist, ist dieser der „private Bereich“. Der Tisch, der von jedem gesehen wird, gilt als „öffentlicher Bereich“ und kann von jedem bedient werden. Zur Präsentation dieses Konzeptes wurde das bekannte Kartenspiel UNO als Anwendung für diese Umgebung entwickelt. Diese Anwendung wurde dann als Plugin-Framework für vernetzte Mixed Reality Anwendungen weiterentwickelt.

Abstract

This work describes the design and development of an environment for intuitive exchange of information between computers. This happens on a table that works as an extension to the notebooks. Information is represented by virtual 3-dimensional objects. These objects can be moved across the borders of the display from the notebooks onto the table. This application also distinguishes between two spaces of privacy. Because the displays of the notebooks are toward the user, the notebook is called “private space”. Because everyone can look onto the table, it is called “public space”. Every user has access to this “public space”. To present these concepts we have developed an application like the famous UNO card-game. This application was enhanced to a plugin framework for networked Mixed Reality Applications.

Kapitel 1

Einleitung

1.1 Ausgangspunkt

Als Grundlage für Future Office dient ein Projekt, durchgeführt von Jun Rekimoto und Masanori Saitoh, welches sich mit dem Thema „Computer Augmented Environment“ beschäftigt.

Dieses Projekt ermöglicht den Benutzer den einfachen Austausch von digitale Informationen zwischen deren Notebooks, einem Tisch- oder Wand-Display oder anderen physischen Objekten, die als Display verwendet werden können. Durch ein optisches Objekterkennungssystem können die Benutzer ihre Notebooks mit den in der Umgebung fix installierten Rechnern sehr einfach verbinden. Weiters können die projizierten Flächen auf dem Tisch und der Wand als Erweiterung der Notebook Displays verwendet werden. Mittels einer Interaktionstechnik names „Hyperdragging“ könne die Benutzer Informationen von einem Computer zu einem anderen übertragen, nur durch das Wissen, wo sich dieser im Raum befindet. Zusätzlich ist es möglich digitale Daten auf physische Objekte wie Videokassetten abzulegen und somit diese mit dem physischen Objekten zu verknüpfen [6].

Als einzige Grundlage für das Future Office diene ein Video [7] sowie ein Artikel [6] über „Augmented Surfaces“. Unsere Motivation war es, aufgrund dieser Materialien ein eigenes Projekt zu entwickeln, das auf ähnliche Art und Weise funktioniert.

1.2 Mixed Reality

Das Future Office wird oft als „Augmented Reality“ Anwendung bezeichnet. Es ist aber nicht klar, was genau diese Bezeichnung zu bedeuten hat und ob sie für dieses Projekt zutreffend ist. Daher werden hier nun ein paar Worte zum „Reality–Virtuality Continuum“ von Milgram verloren.

Milgram beschreibt das Realität–Virtualitäts Kontinuum als einen fließenden Übergang von echter Umgebung (Real Environment) zu virtueller

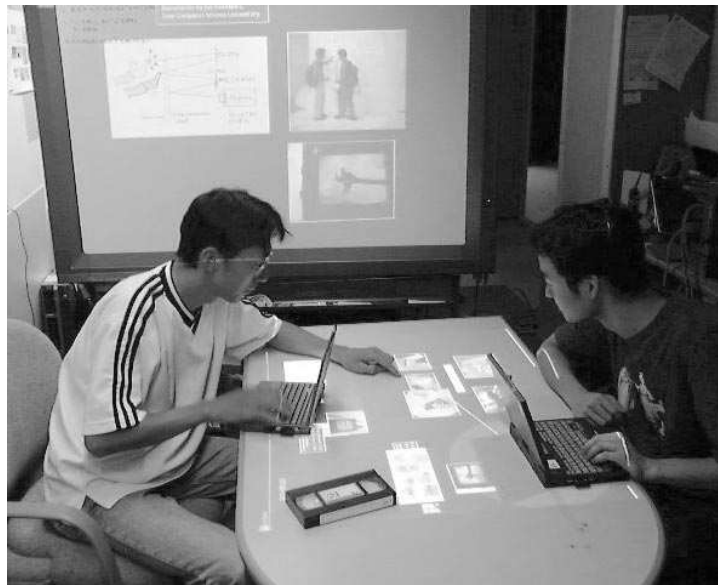


Abbildung 1.1: A spatial continues workspace [7].

Umgebung (Virtual Environment). Eine virtuelle Umgebung umschließt den Betrachter vollkommen mit einer syntetischen Welt und ersetzt somit die reale Umgebung. Als eine echte Umgebung wird eine Welt bezeichnet, in der der Betrachter nur echte Objekte sieht. Zu diesem zählt Milgram auch ein Videobild. Der große Bereich zwischen diesen beiden Welten wird als Mixed Reality (MR) bezeichnet. In diesem Bereich werden sowohl virtuelle Bilder wie auch reale Bilder vermischt. Laut Milgram werden bei Augmented Reality virtuelle Objekte in die reale Umgebung hinzugefügt. Wenn aber der Betrachter eine prinzipiell virtuelle Umgebung sieht in der reale Objekte, z.B. ein Videobild, eingebunden sind, dann bezeichnet Milgram dies als Augmented Virtuality [4].

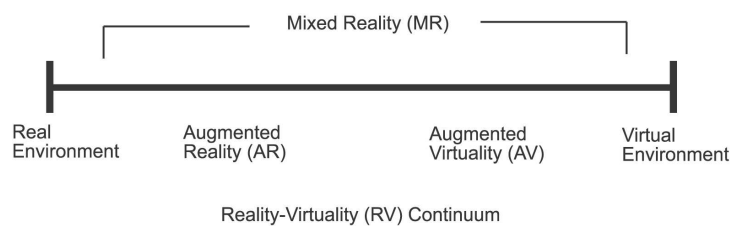


Abbildung 1.2: Vereinfachte Darstellung des RV Kontinuums [4].

Future Office ist in dieses Schema nicht einfach einzubetten. Die Projektionen und Bildschirmdarstellungen zeigen virtuelle Welten. Doch der Tisch, auf den projiziert wird, und die Notebooks sind reale Objekte. Somit sind sowohl virtuelle wie auch reale Objekte vorhanden, die aber erst

miteinander das System bilden.

1.3 Weitere ähnliche Projekte

Außer dem Projekt von Jun Rekimoto und Masanori Saitoh gibt es noch einige weitere Projekte, die sich mit „Computer Augmented Environments“ beschäftigen. Zwei davon sind das auf der TU Wien entwickelte Projekt „Studierstube“ und das auf der University of North Carolina entwickelte „Office of the future“. Wobei sich die „Studierstube“ mehr in Richtung Augmented Reality und das gemeinsame Arbeiten von mehreren Personen in einem Raum bewegt. Das „Office of the future“ beschäftigt dagegen mehr mit großflächigen Projektionen auf unter anderem nicht planaren Oberflächen.



Abbildung 1.3: Vision des „Office of the future“ [5].

1.3.1 Office of the future

1998 wurde auf der SIGGRAPH erstmals das Konzept des „Office of the future“ vorgestellt. Es wurde versucht verschiedene neue Lösungen für Teile eines Arbeitsplatz der Zukunft zu entwickeln. Dieser Arbeitsplatz der Zukunft sollte großflächige, hochauflösende und räumlich immersive Projektionsflächen besitzen, auf denen per Teleworking Arbeitskollegen, wenn möglich sogar dreidimensional, dargestellt werden. Somit sollte das Zusammenarbeiten über große Distanzen natürlicher werden, da der gesamte Körper des Gegenübers dargestellt wird. Gleichzeitig sollte es auch möglich sein an einem virtuellen Model zu arbeiten, das alle Betrachter sehen können

und jeder bearbeiten kann. Im Rahmen des „Office of the future“ wurden Techniken entwickelt um Projektionen auf unebene Oberflächen mit mehreren Projektoren und das Erzeugen von dreidimensionalen Objekten aus Bildern zu ermöglichen [5].

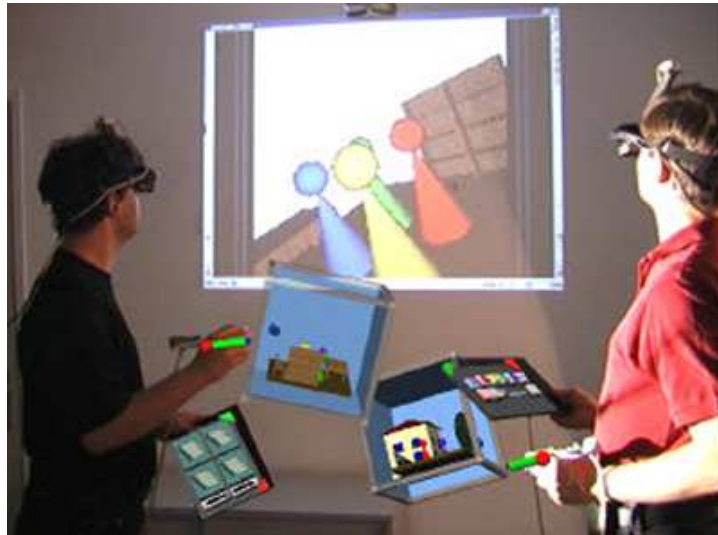


Abbildung 1.4: Studierstube: Eine Storyboard Anwendung mit gemischten Einsatz von einem Projektor und HMDs [8].

1.3.2 Studierstube

Das Projekt Studierstube war eines der ersten Augmented Reality Systeme, das das gleichzeitige Interagieren mehrerer Benutzer in einem virtuellen Raum ermöglichte. Über getrackte Head-Mounted-Displays (HMD) können die Benutzer die virtuellen Objekte von verschiedenen Sichtweisen betrachten, wobei alle Objekte durch stereoskopische Darstellung dreidimensional erscheinen. Im Rahmen des Studierstube Projekts wurde mit einigen verschiedene Darstellungsmethoden und Steuerungsmethoden experimentiert. Neben den HMDs wurde auch mit Projektoren experimentiert bei denen mit Shutterglases der stereoskopische Effekt erzeugt wurde. Bei den virtuellen Steuerungsgeräten wurden unter anderem getrackte Stifte und Schreibunterlagen verwendet, um nicht nur die virtuellen Steuerungselemente zu sehen, sondern um die Steuerungselemente auch fühlen zu können [8].

Im *nächsten Kapitel* wird das Konzept des Future Office erläutert. Darin geht es darum, wie der Aufbau mit Tisch, Projektor und Laptops aussieht und welche Ideen in das Projekt eingeflossen sind. Das *dritte Kapitel* wird sich danach mit den ersten Prototyp beschäftigen, der im ersten Halbjahr 2004 im Rahmen des PRO4 Unterrichts erstellt worden ist. Im *vierten Kapitel* wird das Re-Design des Projekts, das Plugin System, erläutert, das während der Entstehung dieser Arbeit noch in Entwicklung war. Die weiteren Zukunftspläne werden am Schluss noch im *fünfte Kapitel* behandelt.

Kapitel 2

Konzept

2.1 Dimensionen – 2D oder 3D?

Das Projekt von Rekimoto und Saitoh machte es möglich zweidimensionale Daten darzustellen. Diese Darstellungsart macht in Hinsicht, dass Objekte nur auf ebenen Flächen dargestellt werden, durchaus Sinn. Vor allem auf dem Tisch ist eine dreidimensionale Darstellung nicht zielführend, da der Tisch von allen Seiten betrachtet werden kann, aber die Darstellung nur von einer Seite korrekt ist. Trotzdem haben wir uns entschieden im Future Office mit dreidimensionalen Objekten zu arbeiten. Dafür gibt es zwei Gründe:

- Erweiterbarkeit: Durch einen dreidimensionalen Raum ist es möglich auch mit zusätzlichen Darstellungsgeräten wie HMDs die Szenerie dreidimensional zu betrachten. Eine dreidimensionale Darstellung der Objekte auf den Notebooks ist ebenfalls möglich.
- Einfachheit: Der Einsatz von 3D-Bibliotheken (in unseren Fall OpenGL) erleichtert die Transformation von Objekten.

2.2 Hyperdragging

Will ein Benutzer ein Objekt von seinem Notebook auf den Tisch legen, so klickt er auf das Objekt und zieht es bis zum Bildschirmrand. Überschreitet er diesen so wird das Objekt auf dem Tisch angezeigt, und der Benutzer kann es weiter auf dem Tisch verschieben. Dieses Verschieben eines Objektes über die Bildschirmgrenzen des Notebooks hinaus auf den Tisch, wird als Hyperdragging bezeichnet. Man kann den Tisch somit als eine Art Desktoperweiterung für das Notebook betrachten [6].

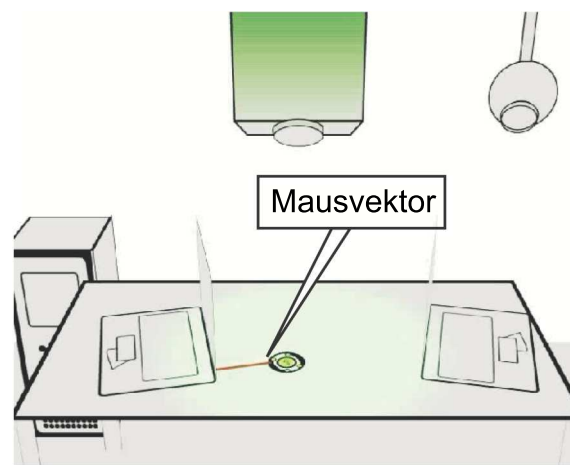


Abbildung 2.1: Der Mausvektor oder „Anchored Cursor“ am Tisch.

2.3 Mausvektor

Da auf dem Tisch mehrere Personen gleichzeitig agieren können, würde die Verwendung von normalen Mauszeigern verwirren, da man seinen eigenen Zeiger leicht „verliert“. Eine Möglichkeit wäre die Zeiger verschieden einzufärben oder zu texturieren. Es ist aber dann nicht immer erkennbar, welcher Zeiger zu welchem Notebook gehört. Durch den Mausvektor ist diese Zuordnung möglich. Der Mausvektor ist eine Linie, die vom Mauszeiger zum zugehörigen Notebook verläuft. Im Projekt von Reikimoto wird dieser Mausvektor als „Anchored Cursor“ bezeichnet [6].

2.4 Privatsphäre

Die Privatsphäre ist im Future Office ein wichtiges Thema. Alleine durch den Einsatz von Notebooks und Projektoren ist schon eine natürliche Trennung zwischen Privaten Bereich (Notebook) und Öffentlichen Bereich (Projektor) vorhanden. Diese Trennung zwischen den „Private Pool“ und den „Public Pool“ wird auch in der Datenhaltung beachtet. Es werden keine Daten auf einen anderen Computer übertragen, wenn dies nicht vom Benutzer ausdrücklich erlaubt worden ist (z.B. durch das Verschieben eines Objekts vom Notebook auf den Tisch).

2.5 Dynamik

Auf einen „echten Besprechungstisch“ ist im Normalfall reger „Betrieb“. Er wird von verschiedenen Personen benutzt und sie kommen und gehen normalerweise nicht gleichzeitig. Weiters benützt niemand fix installierte

Notebooks, sondern es hat jeder seinen eigenen mit. Dieses Verhalten ist auch im Future Office möglich. Die öffentlichen Bereiche (der Tisch) sind fest installiert und immer aktiv, es können aber jeder Zeit neue Benutzer mit ihren Notebooks hinzukommen oder wieder gehen. Kommt ein Benutzer mit seinem Notebook an den Tisch und verbindet ihn mit dem Netzwerk, dann wird das Notebook erkannt und automatisch in die Anwendung eingebunden. Natürlich kann man die Anwendung genauso einfach wieder verlassen.

2.6 Zusammenarbeit

Das Wichtigste des Future Office ist das gemeinsame Arbeiten. Die Benutzer können gleichzeitig auf dem Tisch agieren. Dies kann bei größeren Gruppen zwar sehr unübersichtlich werden, aber das ist auf einen normalen Besprechungstisch nicht anders. Durch den Einsatz von Computertechnik ist es aber möglich Sitzungen zu speichern, d. h. man kann die erzeugten Objekte abspeichern und an einem späteren Zeitpunkt wieder laden. Somit könnte man einmal gezeichnete Skizzen immer wieder verwenden, ohne sie in großen Papierstößen suchen zu müssen.

2.7 Easy to use

Ein Schlagwort das man bei fast jedem Softwareprojekt hört ist die Benutzerfreundlichkeit. Auch im Future Office spielt dies eine große Rolle. Aber nicht nur auf Seiten des Benutzers, sondern auch auf der des Entwicklers. Auf Benutzerseite wird versucht vieles zu vereinfachen. Beispielsweise wird die Netzwerkverbindung automatisch hergestellt, d. h. die Clientanwendung findet von selbst den Server im Netzwerk. Auf Seite des Entwicklers wird ein Framework zur Verfügung gestellt, das die Netzwerkkommunikation, das Tracking, Rendering für ihn übernimmt.

2.8 Hardware

Bei vielen Mixed Reality Projekten ist der Hardware Aufwand ein kein geringer. HMDs und viele Trackingsysteme sind im Allgemeinen teuer und wenig verbreitet. Im Future Office versuchen wir nur Hardware zu verwenden, die heute schon weit verbreitet sind. Das Future Office benötigt „nur“ einen Tisch, einen Projektor, einen Computer als Server, eine Webcam und Laptops. Natürlich kann man im Future Office auch HMDs einsetzen, aber es ist nicht notwendig.

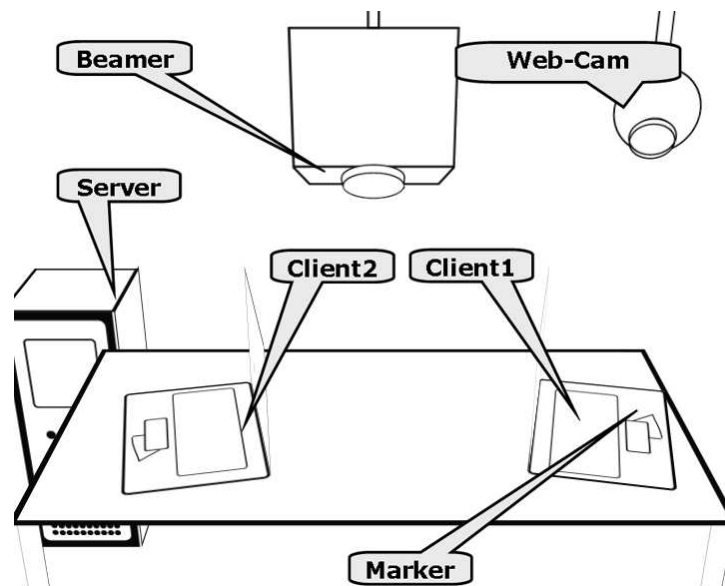


Abbildung 2.2: Setup des Future Office.

2.9 Setup – Aufbau einer Anwendung im Future Office

Das Future Office ermöglicht prinzipiell einen Aufbau mit mehreren Projektoren und verschiedensten Trackingsystemen. Das hier beschriebene Setup ist eine Möglichkeit des Aufbaus, welche bei dem Prototyp verwendet wurde.

Ein Projektor ist in der Höhe von 2,5 Meter über der Längskante des Tisches montiert. Der Tisch hat eine Höhe von 70 cm. Somit beträgt die Projektionsfläche 1,8 mal 1,3. Die Webcam (Tracking) wird neben dem Projektor installiert, so dass sie den gesamten Tisch erfasst. Der Projektor wird an den Server-Computer angeschlossen. Die Projektion auf den Tisch sowie das Tracking der Laptops übernimmt dieser Server. Bei diesem Aufbau werden nur zwei Laptops verwendet.

Kapitel 3

Prototyp – der erste Test

3.1 Allgemeines

Der Prototyp wurde im 4. Semester im Zuge von PRO4 erstellt. Dieser Prototyp ist auch unter den Namen Futo oder Footo bekannt.

3.2 Ziele

Das Ziel des Prototyps war es die Grundidee des Rekimoto/Saitoh Projekts nachzubauen und eine spezielle Anwendung – ein Spiel ähnlich UNO – zu realisieren. Dabei sollten folgende grundlegende Funktionen erfüllt werden:

- Zwei Benutzer können dreidimensional dargestellte Objekte (Spielkarten) von ihren Laptops auf den Tisch ablegen.
- Diese Objekte können auch wieder vom Tisch in den Laptop gezogen werden.
- Die Datenübertragung wird im Hintergrund über ein Netzwerk vollzogen.
- Eine einfache Spiellogik stellt die Grundvoraussetzungen (Rundenwechsel, Karten austeilen, usw.) für das UNO-Spiel her.

3.3 Basis – Grundlegende Bibliotheken

Der Future Office Prototyp verwendet folgende Bibliotheken:

- OpenGL und GLUT zur Visualisierung.
- ARToolKit als optisches Tracking System.
- PTypes als Netzwerkbibliothek.

- XML Bibliotheken aus Microsofts .net Umgebung für die Datenspeicherung.

3.4 Struktur – Der Aufbau des Prototyps

Der Prototyp gliedert sich in vier Schichten:

- Rendering
- Tracking
- Netzwerk
- Datenhaltung

Diese vier Teile wurden von je zwei Projektmitgliedern entwickelt und realisiert.

3.4.1 Rendering

Die Hauptaufgaben der Rendering-Schicht ist die Visualisierung der Karten, des Mausursors und des Menüs, sowie das Picking der Objekte (Cardpicking).

Zur Visualisierung: Um überhaupt anfangen zu können, muss zunächst eine OpenGL-Umgebung eingerichtet werden. Dies beinhaltet die Einbindung von GLUT, dass die Ausgabe von OpenGL in ein Fenster und das Abfragen von Maus- und Tastaturevents ermöglicht. Die Karten selbst werden als ein quadratisches Polygon mit einer beliebigen Textur dargestellt.

Zur Maus: Die Abfrage der Mauskoordinaten in einem System, dass auch dann Mausbewegungen verarbeitet möchte, wenn diese eigentlich außerhalb des Bildschirm wären, ist nicht trivial. Da Windows außerhalb des Bildschirms keine eigenen Mauskoordinaten liefert, und die Abfrage der relativen Mausbewegung nicht möglich ist, muss dies selbst gelöst werden. Die einfachste und auch in der Spielebranche weit verbreitete (Quake, Doom) Technik ist, die Mausposition nach jeder Mausbewegung wieder in die Mitte des Bildschirms zu setzen. Die Differenz zur Mitte ist somit die relative Mausbewegung. Die aktuelle Mausposition muss nun aber von der Anwendung selbst gespeichert werden.

Zum Punkt Cardpicking: Da es beim Prototyp möglich sein soll, die 3D-Objekte per Drag&Drop zu bewegen, ist eine Collision Detection notwendig, die optimalerweise sowie für die Karten und als auch dem Menü funktionieren sollte. Da alle bewegbaren Objekte in unserer UNO Anwendung kreisförmig sind, ist diese durch einer einfache Abstandsberechnung zwischen Mausposition und Objektmittelpunkt möglich. Eine Kollisionserkennung ist dann positiv, wenn dieser Abstand kleiner als der Objektradius ist. Ist nun die Collision Detection erfolgreich und wird der linke Mausbutton gedrückt,

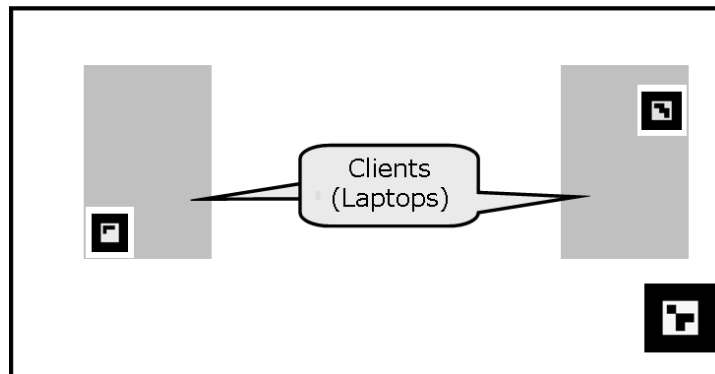


Abbildung 3.1: Marker auf Tisch und Laptops fürs Tracking.

kann das Objekt beliebig herumgezogen werden. Dies funktioniert auch über die Bildschirmgrenzen hinaus. Bewegt man eine Spielkarte aus einem Laptop auf dem Tisch, so folgt diese der Maus auch auf dem Tisch solange die linke Maustaste gedrückt ist. Diese Technik ist das im zweiten Kapitel beschriebene Hyperdragging.

Zum Menü ist noch zu sagen, dass die einzelnen Menü-Objekte ähnlich aufgebaut sind wie die Karten. Sie sind ebenfalls aus texturierten quadratischen Polygonen aufgebaut, wobei die Textur einen kreisförmigen Menüpunkt darstellt. Das gesamte Menü kann – wie die Karten – mit gedrückter Maustaste bewegt werden.

3.4.2 Tracking

Um das Spielsystem gänzlich dynamisch zu halten ist es notwendig auch die Position und Orientierung der Laptops am Tisch und des Tisches selbst automatisch zu erkennen. Die aktuelle Position und Orientierung von Laptop und Tisch ist vor allem für die korrekte Darstellung des Mauszeigers und Mausvektors außerhalb des Bildschirmbereichs, also auf dem Tisch, von großer Bedeutung.

Das Tracking der Laptops mit einer Kamera wird über die Bibliothek ARToolkit ermöglicht. ARToolkit erkennt durch eine WebCam bestimmte vordefinierte Marker in der „realen“ Welt und liefert die Transformationsmatrix zwischen dem real vorhandenen Marker und der Kamera zurück. Anhand dieser Transformationsmatrix kann man schließlich die Position und Orientierung des Markers und somit auch die des Objekts auf dem der Marker angebracht ist, ermitteln. Siehe Abbildung 3.1.

3.4.3 Netzwerk

Die Netzwerkschicht ist für die Kommunikation zwischen dem Server und den Clients zuständig. Diese Kommunikation basiert auf einem Eventsy-

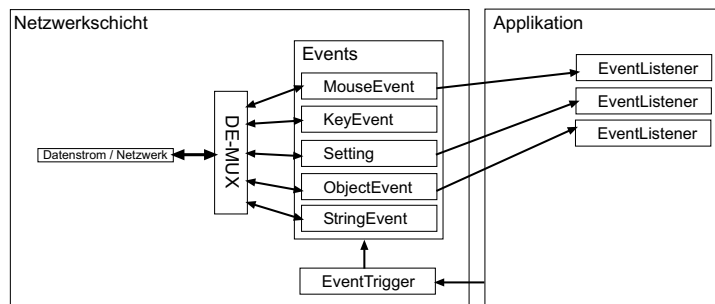


Abbildung 3.2: Netzwerk: Schematischer Aufbau der Eventverarbeitung.

stem, d.h. alle Daten und Informationen werden in Events verpackt übertragen.

Aufgaben der Netzwerkschicht:

- Auffinden des Servers im Netzwerk.
- Herstellung einer Verbindung.
- Halten der Verbindung.
- Übertragung von Events.
- Weiterleiten von Events.
- Eventmultiplexer.
- Eventdemultiplexer.
- Bereitstellen der geeigneten Schnittstellen.

Realisiert ist die Netzwerkschicht als Framework das nur einige wenige Schnittstellen bereitstellt. Diese Schnittstellen ermöglichen das Abfangen und Entgegennehmen von Events durch Event-Listener-Objekte sowie das Abschicken von Events durch Trigger-Methoden. Eine zentrale Rolle spielt hierbei der Multiplexer oder Demultiplexer, der die Events in eine sendbare Byteform bringt und auch wieder rückverwandelt.

Alle netzwerktypischen Eigenschaften werden aber von dem übrigen Modulen abgekapselt, somit ist die Netzwerkschicht sehr einfach veränderbar, bzw. ist es sehr einfach diese einzubinden.

3.4.4 Datenhaltung

Die Datenhaltung kümmert sich um folgende Themen:

- Welche Daten werden benötigt?

- Wo werden diese benötigt?
- Welche Daten müssen ausgetauscht werden?
- Wie werden diese ausgetauscht?
- Wie werden die Daten zur Verfügung gestellt?

Die Datenhaltung interagiert sehr stark mit der Netzwerkschicht, da diese alle Daten als Events verpackt benötigt. So muss jede Information zunächst von der Datenhaltung in ein Event verpackt und auf der anderen Seite wieder entpackt werden.

Da alle Daten als XML-Dateien vorhanden sind, muss die Datenhaltungsschicht auch einen XML-Parser beinhalten, der aus den XML-Dateien die jeweiligen C++ Objekte generiert. Diese Objekte stellen Informationen zu Verfügung wie: ID´s der auszutauschenden Objekte (Karten), Texturen dafür, anfängliche Position, Größe des Tisches, Auflösung des Beamers, Abmessungen der Clients, eingestellte Auflösung der Clients, Position der Marker, usw.

3.5 Conclusio

Während der Entwicklung entstanden einige Probleme (z. B. Relative Mausposition), wobei sich die meisten auch schnell lösen ließen. Alle Schichten, bis auf die Trackingschicht, wurden erfolgreich implementiert. Das UNO-Spiel funktioniert, abgesehen einiger kleinerer Spiellogikfehler, gut und erfüllt alle Ziele. Das Lesen der Daten aus den XML-Dateien und das Erzeugen der Objekte funktioniert genauso wie die Netzwerkkommunikation und das Rendering.

Beim Aufbau des Prototyps fiel eine Eigenschaft auf, die wir vorher nicht bedacht hatten. Es gibt einen signifikanten Auflösungsunterschied zwischen der projizierten Fläche auf dem Tisch und den Notebookbildschirmen. Durch diesen Unterschied werden die Objekte am Tisch um vieles größer dargestellt, als am Notebook. Auch die Mausgeschwindigkeit ist am Notebook um vieles langsamer als am Tisch. Zunächst erschien dies als ein Problem. Wir nahmen an, dass der „Sprung“ zwischen Tisch und Laptop für den Benutzer unangenehm ist, doch nach Tests stellten wir fest, dass dies nicht der Fall war. Im Gegenteil, durch die größere Darstellung der UNO-Karten auf dem Tisch, sind diese besser Erkennbar, als in einer verkleinerten Form. Auch die schnellere Mausbewegung ist durchaus von Vorteil, weil ansonsten eine viel größere Bewegung der physikalischen Maus notwendig wäre um von einer Seite des Tisches auf die anderen zu kommen.

Einige Probleme konnten aber nicht mehr gelöst werden:

- Performance-Probleme am Server.

- Kein Tracking.
- Mangelhafte Erweiterungsfähigkeit.

3.5.1 Performance-Probleme am Server

Der Server erreichte nur eine sehr geringe Framerate, teilweise setzte das Rendering sekundenlang aus. Das Aussetzen des Renderers entstand durch einen Fehler im Design des Prototyps. Da die einzelnen Netzwerkverbindungen als Threads ausgeführt wurden und der Renderer nicht und auch keine Threadsynchronisierung möglich war, bekam der Renderthread keine Rechenzeit, wenn Netzwerkevents eintrafen. So wurde der Renderer nur selten aufgerufen. Mit einem eigenen „Scheduler“ wurde zwar das Aussetzen verhindert, aber eine Erhöhung der Framerate hatte dies nicht zur Folge. Die geringe Framerate lag vermutlich auch an der schlechten Hardware (Onboard S3 Grafikkarte) des Servers.

3.5.2 Kein Tracking

Es gelang leider nicht das ARToolkit so zu kalibrieren, dass die gelieferten Transformationsmatrizen einen Sinn ergaben. Warum die Kalibrierung nicht funktionierte konnte nicht eruiert werden. Als Notlösung wurde die Laptop-Position und Orientierung statisch gesetzt, damit eine Demonstration des Prototyps trotzdem möglich war.

3.5.3 Mangelhafte Erweiterungsfähigkeit

Da es sich bei diesem ersten Projekt um einen Prototyp handelte und ein Großteil die Materie vorher unbekannt war, konnte auch kein genauer Aufbau geplant werden. Mit zunehmender Größe wurde der Prototyp auch immer unübersichtlicher, da immer wieder bereits fertige Teile verändert werden mussten, weil sie nach dem Integrieren in den Prototyp nicht mehr funktionierten. So entwickelte sich der Prototyp zu einer unübersichtlichen, nicht erweiterungsfähigen Anwendung.

3.6 Rückmeldungen

Der Prototyp des Future Office wurde erstmals auf der MTDGala im Oktober 2004 der Öffentlichkeit präsentiert. Im Rahmen dieser wurde das Projekt auch mit einem Award in der Kategorie Medientechnik ausgezeichnet. Bei der Präsentation des Future Office konnten erstmals Erfahrungen von projektfernen Personen mit der Handhabung der Anwendung gesammelt werden. Ungefähr 20 Personen testeten das UNO-Spiel. Die meisten waren sehr überrascht, dass man den Mauszeiger und auch Objekte auf den Tisch bewegen konnte. Weil man den Projektor nicht sofort erkennen konnte,

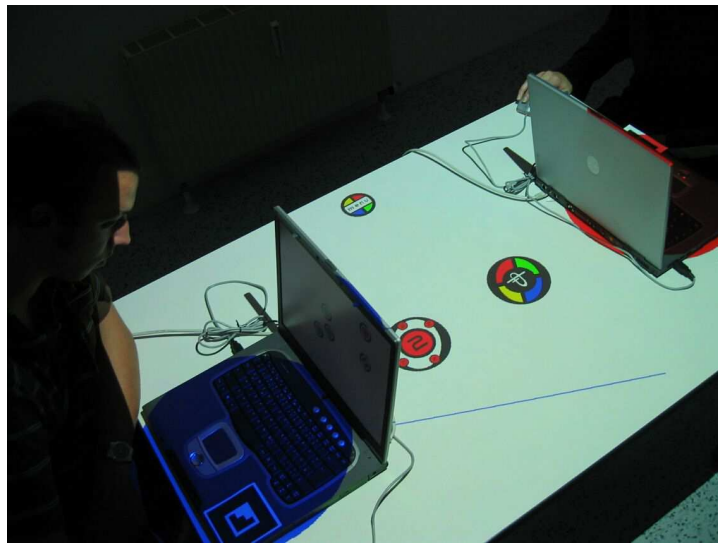


Abbildung 3.3: Das UNO-Spiel in Aktion.

fragten auch viele, woher die auf dem Tisch projizierte Darstellung kommt. Es war überraschend, dass alleine die Projektion auf den Tisch viele Besucher faszinierte. Oft genannte Fragen und Bemerkungen waren: „Das ist ja wie eine Desktoperweiterung auf einen Tisch“, „Woher kommt das Bild auf dem Tisch?“, „Wie funktioniert das überhaupt?“. Die Rückmeldungen waren größtenteils positiv, wenn auch viele das Spielprinzip von UNO nicht kannten.

Kapitel 4

Re-Design – Plugin System

4.1 Ausgangssituation

Das UNO-Spiel des Prototyps demonstrierte nur eine Möglichkeiten das Future Office zu verwenden. Es sollte auch möglich sein, andere Anwendungen auf Basis von Future Office zu entwickeln. Die Entwicklung neuer Anwendungen und das Integrieren dieser in die Future Office Umgebung sollte einfach von statten gehen. Somit sollte sich das Future Office Projekt zu einer Plattform für vernetzte Mixed Reality Anwendungen werden.

Weiters sollte das Future Office auch einfach Erweiterbar und Bedienbar sein, d. h. es sollte möglich sein ein beliebiges Setup aufzubauen oder neue Eingabegeräte zu integrieren. Aufgrund der Unübersichtlichkeit des Prototyps war dies in diesem nicht möglich. Somit musste das gesamte Projekt überarbeitet werden und neu entwickelt werden. In dieses Re-Design flossen die erworbenen Kenntnisse von der Entwicklung des Prototyps ein. Anders als beim Prototyp konnte somit auch eine längere Planungsphase durchgeführt werden.

An der Entwicklung des Plugin Systems sind beteiligt:

- Leitner Jakob
- Seifried Thomas

Neben diesem Re-Design wurden gleichzeitig zwei Projekte entwickelt die im direkten Zusammenhang mit dem Plugin System stehen. Eine Projektgruppe entwickelte als Anwendung für das Plugin-System ein Brainstorming Tool, das als erster Test für das Plugin Systems mit einem Anwendungsplugin gilt. Eine weitere Projektgruppe entwickelte als Eingabe-Plugin ein virtuelle Tastatur sowie eine virtuelle Maus. Diese Projekte werden in das

Plugin System integriert und sollen eine erste Test- und Präsentationsumgebung für das Future Office sein.

Da sich das Plugin System zum Zeitpunkt des Entstehens dieser Arbeit noch in Entwicklung befand, werden in diesem Kapitel manchmal Modulen beschrieben werden, die noch nicht implementiert oder getestet waren. Am Ende dieses Kapitels wird sich der Abschnitt 4.9 mit dem aktuellen Status befassen.

4.2 Zielsetzung

Das Hauptziel des Re-Designs war die Neustrukturierung des Future Office zu einem Plugin System, dass Anwendungen, Eingabegeräte und Objekte während der Laufzeit lädt und ausführt. Wichtig war, dass neue Anwendungen sehr einfach implementiert werden können, damit das Future Office auch ohne langes Einlernen verwendet werden kann. Zur weiteren Vereinfachung soll, anders als beim Prototyp, in der Plattform auch keine Unterscheidung zwischen Server und Client gemacht werden. Je nach Anwendung kann jeder Rechner Client, Server oder beides gleichzeitig sein.

Die Plattform soll auch so dynamisch wie möglich sein, d.h. je nach Anwendung sollen nur jene Tracking Systeme eingebaut werden können, die auch wirklich benötigt werden.

Das Future Office Plugin System soll eine Plattform für „Computer Augmented Environments“ sein. Es gibt schon viele einzelne MR Projekte die sich mit MR Eingabe oder Ausgabe beschäftigen. Ein Beispiel dafür ist ein virtuelles Keyboard oder ein Laserpointer-Tracking-System. Im Future Office sollten diese MR Projekte in ein System zusammengefasst werden können.

Somit ergeben sich folgende Ziele:

- Strukturierter Aufbau des Projekts.
- Erweiterbarkeit durch klare, einfache Schnittstellen.
- Ermöglichung von Plugins.
- Verbesserung der Performance.

4.3 Konzeption

4.3.1 Was sind Objekte?

Im Prototyp waren alle beweglichen Objekte flache Spielkarten. Im Plugin System ist ein Objekt ein Daten-Container der verschiedene Informationen

beinhaltet. Diese können Positionsinformationen, Forminformationen oder sonstige objektspezifische Informationen, aber auch Dateien sein. Aus diesen Informationen wird die Darstellung des Objekts bestimmt. Ein Objekt kann somit Geometriedateien enthalten die durch Verwendung der entsprechenden Bibliotheken als dreidimensionale Darstellungen des Objekts angezeigt werden. Ein Objekt kann aber auch eine Objektgruppe sein, mit der mehrere Objekte gruppiert werden können.

4.3.2 Client / Server

Eine MR Anwendung ist beim Prototyp in Client und Server aufgeteilt. Wobei sich ein Client nur zu einem einzigen Server verbinden kann. Diese Beschränkung ist im Pluginsystem nicht mehr vorhanden. Jede Anwendung kann zu beliebig vielen anderen Rechnern Kontakt aufnehmen. Jeder Rechner ist somit Client und Server gleichzeitig. Auf Anwendungsseite ist es aber oft sinnvoll, dass nur ein Server vorhanden ist. Aus diesem Grund kann sich ein Rechner auch explizit als Server im Netzwerk ausgeben. Alle anderen Rechner verbinden sich dann automatisch zu diesem.

4.3.3 Warum Plugins?

Warum werden Plugins eingesetzt? Plugins haben einige Vorteile. Zunächst ermöglichen Plugins das Hinzufügen von neuen Programmteilen ohne dass das Programm neu kompiliert werden muss. Das hat vor allem dann einen großen Vorteil, wenn der Quellcode des Programms nicht verfügbar ist. Somit macht sich das Programm veränderbar ohne den Quellcode freigeben zu müssen. Weiters können Plugins bei Laufzeit in das Programm hinzugefügt werden. Somit muss das Programm nicht extra beendet werden, um Plugins auszutauschen. Vor allem dieser Aspekt ist für das Future Office von großer Bedeutung, da die Anwendung als Plugin geladen wird und somit die Anwendung ohne Neustart des Systems gewechselt werden kann.

4.4 Basis – Grundlegende Bibliotheken

Einige Probleme des Prototyps entstanden durch Fehlentscheidungen bei der Auswahl der zu Grunde liegenden Bibliotheken. So war die Verwendung von Microsofts .net Umgebung im XML Bereich für die Übersichtlichkeit nicht sehr förderlich. Auch die Netzwerkbibliothek hatte einige Fehler, durch die es beim Beenden des Programms immer wieder zu Abstürzen kam. Während der Entwicklung des Prototyps fiel auf, dass viele Programmteile den Anforderungen von Spielen gleich kamen. Somit viel die Entscheidung, welche Basisbibliothek verwendet werden sollte, auf die ClanLib Bibliothek, welche aus dem Bereich der Spielentwicklung kommt. Diese ersetzt im Netzwerkbereich PTypes, im Visualisierungsbereich GLUT und in der Datenhaltung

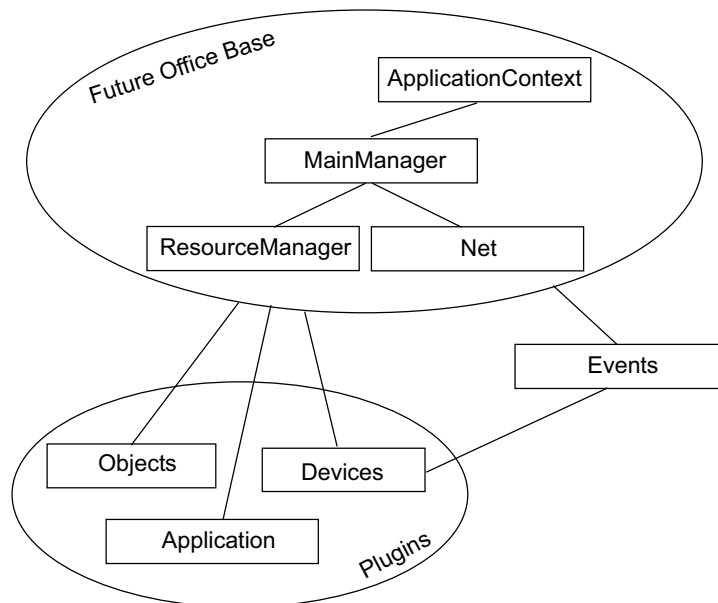


Abbildung 4.1: Schematischer Aufbau des Future Office Plugin Systems.

die .net XML Bibliotheken. Ein weiterer Entscheidungsgrund für die Clanlib ist die Multiplattformfähigkeit (Linux, Windows, Mac) und dass sie unter der LGPL Lizenz verbreitet wird. Somit muss der Quellcode des Future Office nicht veröffentlicht werden, wenn dieses verbreitet werden sollte. Somit werden folgende Bibliotheken als Basis verwendet:

- ClanLib als Anwendungsframework, Netzwerk- und XML-Bibliothek.
- OpenGL zum Rendern.
- ARToolKit als optisches Tracking System.
- libReference für Smart Pointer.

4.5 Struktur – Der Aufbau des Systems

Im Gegensatz zum Prototyp ist das Plugin System nicht in Schichten sondern in einer modularen Form aufgebaut. Den Kern des Systems bilden der `MainManager`, der `ResourceManager`, das Netzwerkmodul `Net` und der Applikations Kontext (`ApplicationContext`). Alle anwendungsspezifischen Module werden über Plugins eingebunden, die mit dem Kern interagieren. Der Großteil der Kommunikation wird über Events von statten. Diese sind in diesem Diagramm nicht im Kern, weil diese als Kommunikationsträger sowohl mit den Plugins (vor allem den Devices) und dem Kern interagieren.

Im Folgenden werden diese Module erklärt. Zu den jeweiligen Modulen sind auch die Klassendiagramme dieser hinzugefügt. Ein gesamtes Klassendiagramm ist im Anhang B abgebildet.

4.6 Future Office Base – Der Kern

Der Future Office Base ist die Sammlung aller notwendigen Module des Future Office, welche die gesamte Datenhaltung und Ressourcenverwaltung, die Steuerung und die Netzwerkkommunikation realisieren. Die Datenhaltung wird im `ApplicationContext` behandelt, die Ressourcenverwaltung im `ResourceManager`, die Steuerung im `MainManager` und die Netzwerkkommunikation natürlich im `Net` Objekt.

4.6.1 MainManager – Steuerung

Der `MainManager` ist das Herzstück des Future-Office-Systems. Von hier aus startet das Future Office und von hier aus wird das Framework gesteuert. Eine Hauptaufgabe des Steuerungsmoduls ist das „Booten“ des Frameworks. Er erstellt und lädt alle notwendigen Objekte und Plugins und startet das Anwendungsplugin (siehe Abbildung 4.3). Außerdem verfügt der `MainManager` alle notwendigen Zugriffsmethoden auf andere Kernmodule, wie den Applikationskontext und dem Netzwerkmodul. Eine weitere wichtige Aufgabe ist die Steuerung. Über Steuerungsmethoden wird der Anwendung ermöglicht direkt in den Ablauf des Systems einzugreifen. Die Anwendung kann zum Beispiel eine zeitgesteuerten Aufruf eigener Funktionen aktivieren oder deaktivieren. Die Steuerung des Netzwerkmoduls ist ebenfalls im `MainManger` möglich. Das deshalb notwendig, weil nur die Anwendung weiß, ob das Netzwerkmodul einen Server starten soll oder nicht. Zu letzt ist der `MainManger` auch zuständig für das saubere Beenden des Systems.

4.6.2 ResourceManager – Ressourcen-Verwaltung

Die Ressourcen-Verwaltung kümmert sich um alles was geladen und gespeichert werden muss/kann und um das Laden und Speichern an sich. Weiters verwaltet sie auch alle Ressourcen, wie beispielsweise Dateien (Bilder, Texturen, Modelle, usw.), und beachtet, dass gleiche Dateien nicht mehrfach in den Speicher geladen werden. Dies ist vor allem beim Texturen von großer Bedeutung, da diese oft mehrfach verwendet werden. Die Ressourcen-Verwaltung arbeitet mit dem XML-Parser von ClanLib. Alle Daten werden vom `ResourceManager` als XML-Dateien abgespeichert und wenn notwendig übertragen. Das ermöglicht zwar eine gute Flexilibiltät, aber auf Kosten von Performance. Da XML-Dateien vom Menschen sehr einfach les- und

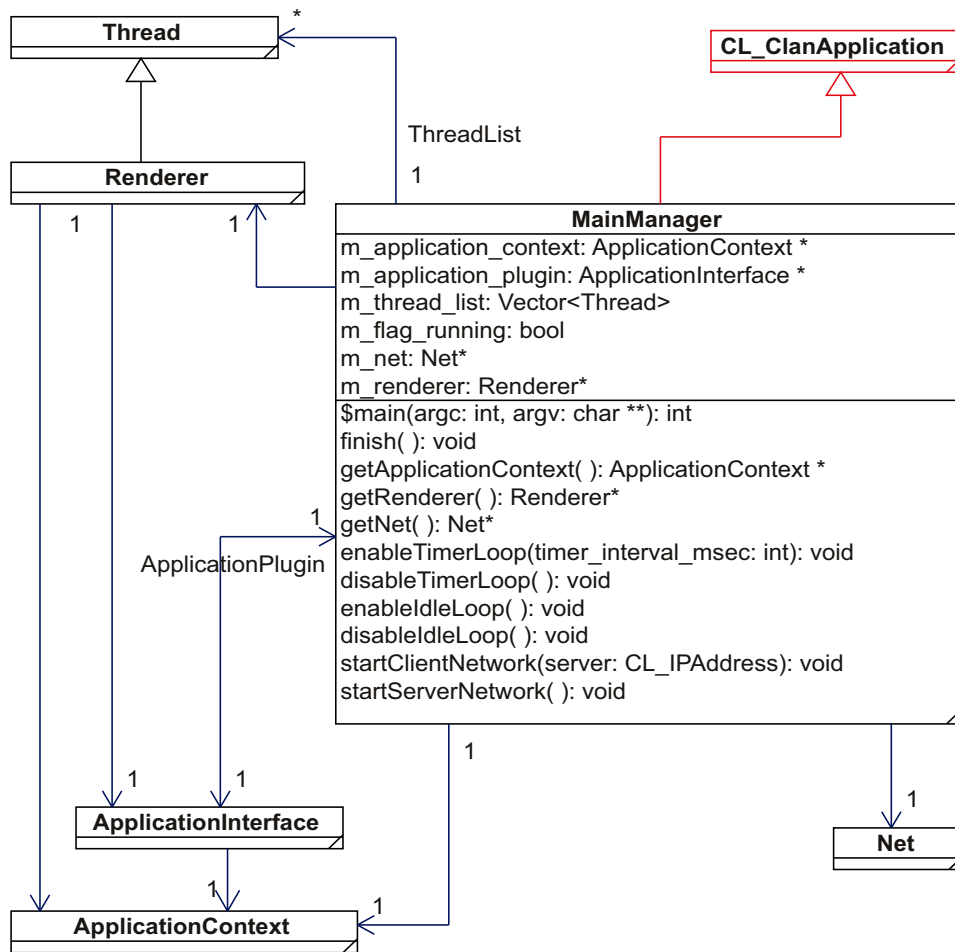


Abbildung 4.2: Klassendiagramm des Mainmanagers mit Umgebung.

bearbeitbar sind und leicht exportiert werden können, ist dieser Performanceverlust akzeptierbar.

4.6.3 Network – Netzwerkmodul

Die Kommunikation zwischen den einzelnen Computern geschieht über das Netzwerkmodul. Das Netzwerkmodul ist Server und Client zugleich. Jeder Computer, auf dem eine Future Office Anwendung läuft, kann für einen anderen ein Server sein. Je nach Anwendung macht es aber Sinn, dass nur ein Anwendungsserver im Netzwerk ist. Damit dieser gefunden werden kann, wird auf diesen ein eigener Service, der **EchoServer**, gestartet, der Clients antwortet, wenn diese einen Server suchen. Somit finden die Clients den Server selbst in Netzwerk von selbst. Das Netzwerkmodul ermöglicht natürlich das Verbinden zu einem Server, aber zusätzlich ist es auch möglich

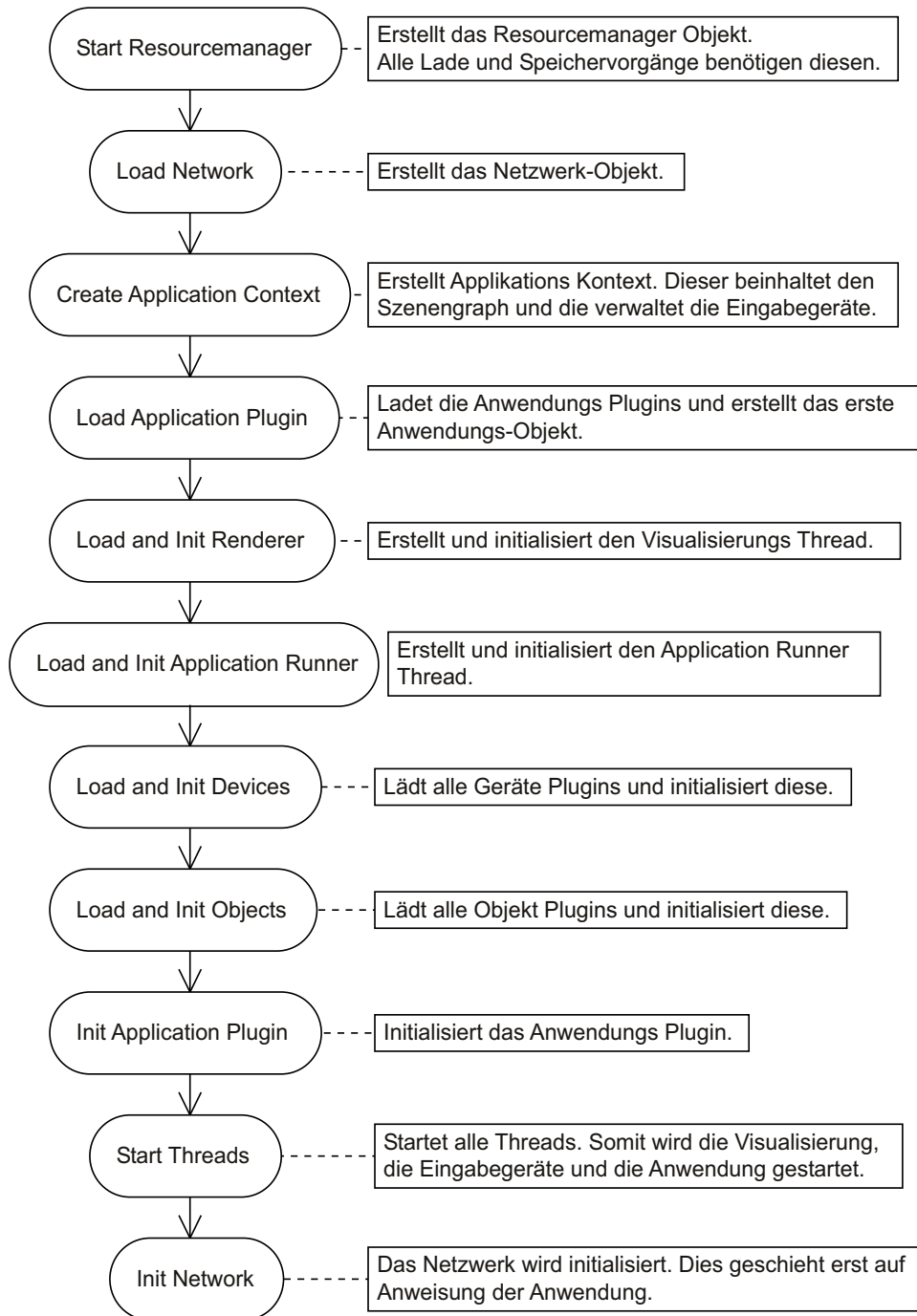


Abbildung 4.3: Bootvorgang des Mainmanager.

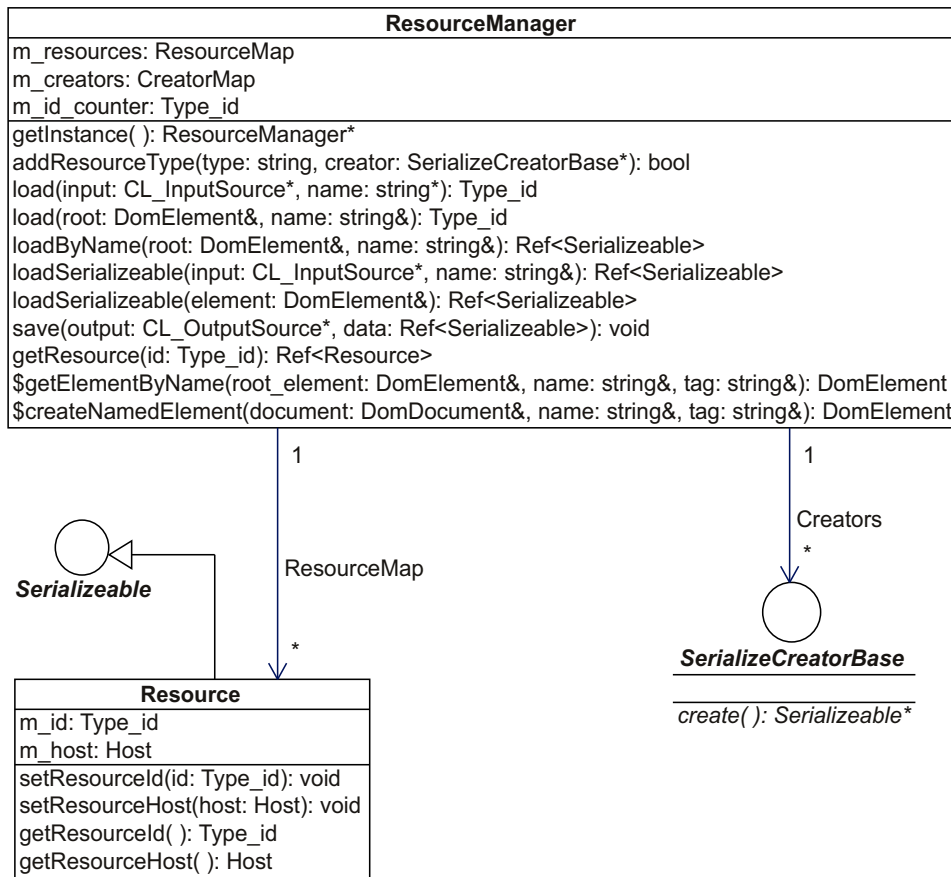


Abbildung 4.4: Klassendiagramm des RessourcenManagers und Umgebung.

zu beliebig vielen anderen Rechnern eine Verbindung herzustellen oder von beliebig vielen anderen Rechnern Verbindungen entgegen zu nehmen. So kann man eine direkte Verbindung zwischen zwei Clients erreichen und sogar ein Peer-to-Peer Netzwerk aufbauen.

Die Datenübertragung an sich funktioniert wie im Prototyp über Events. In diese werden alle zu übertragenden Daten verpackt und so übers Netzwerk übertragen. Auf der Gegenseite können diese Events mittels Eventhandler entgegen genommen und verarbeitet werden. Diese Eventhandler werden durch Signale aufgerufen, welche das Netzwerkmodul bei Empfang eines Events aufruft. Mehr dazu im Absatz 4.7.2.

4.6.4 ApplicationContext und Szenegraph

Der Applicationskontext ist der Container für alle visualisierbaren Objekte und ebenfalls für die Devices. Der Objektcontainer ist als Szenegraph ausgeführt. Es werden im Applikationskontext aber nur Objekte verwaltet,

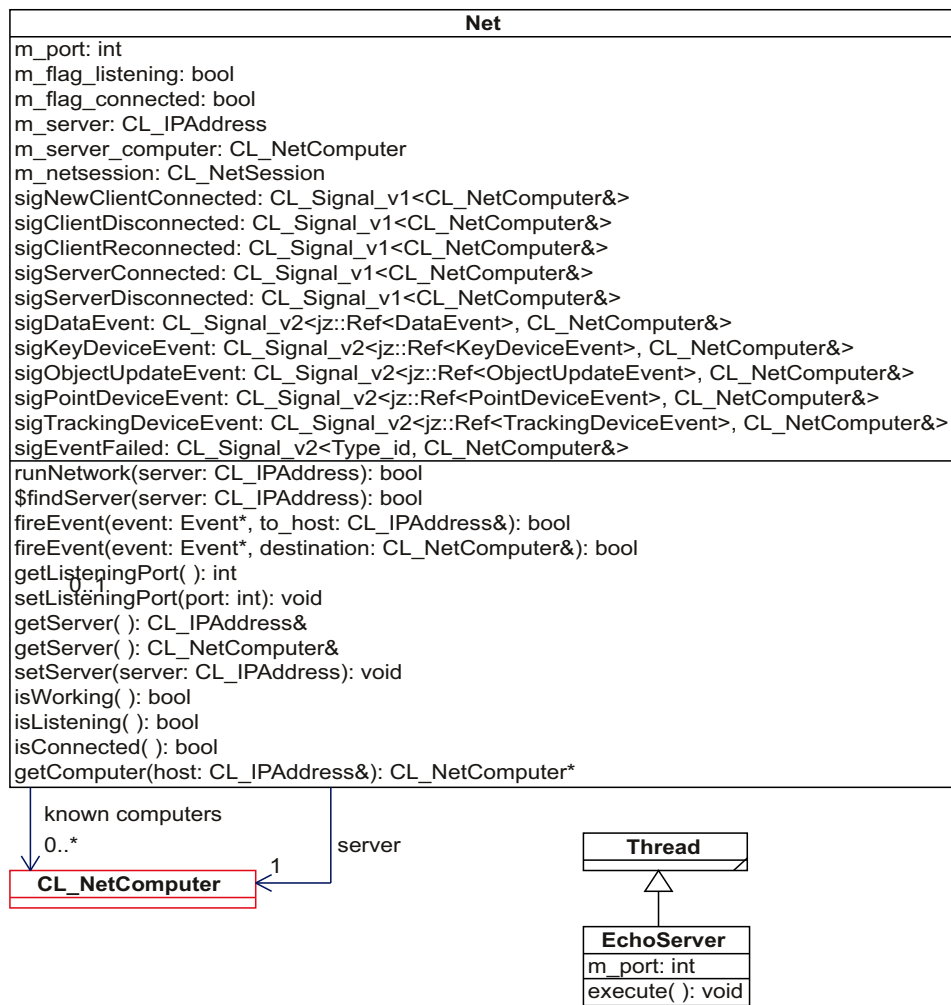


Abbildung 4.5: Klassendiagramm des Netzwerkmoduls.

nicht die dazugehörigen Ressourcen. Die Ressourcen wie Texturen, Geometriedaten, Animationsdaten werden vom **ResourceManager** verwaltet.

Ebenfalls im **ApplicationContext** werden die Eingabegeräte (InputDevices) verwaltet. Über Zugriffsfunktionen kann sowohl auf lokale Eingabegeräte, wie die Tastatur oder eine Maus, zugegriffen werden, als auch auf entfernte Eingabegeräte, die sich auf anderen Computern befinden.

Die Ausgabe auf den Bildschirm wird über den einfachen Szenengraphen bewerkstelligt. Ein Szenengraph ist eine Datenstruktur, in der alle für die Darstellung relevanten Objekte abgespeichert werden. Solche Objekte können sowohl Darstellungen, beispielsweise 3D-Objekte, Objektgruppierungen und auch Transformationen sein. Ist der Szenengraph als eine Baumstruktur organisiert, so wirken sich diese Transformationen auf alle

„Kindelemente“ der Transformation aus.

Der Szenengraph des Future Office ermöglicht den Aufbau einer Baumstruktur. Objekte können zu Objektgruppen zusammengefasst werden, wobei auch Objektgruppen zu anderen Objektgruppen hinzugefügt werden können. Die Transformationen werden nicht als eigene Objekte im Szenengraph eingebunden, sondern sind ein Teil der 3D-Objekte und Objektgruppierungen. Weiters können mit ihm Objekte auch versteckt und wieder angezeigt werden. Der Szenengraph ist aber nicht nur für die Ausgabe zuständig, sondern auch für die Verwaltung der Objekte. Alle Objekte die angezeigt werden können, sind auch im Szenengraph vorhanden. Natürlich ist es ein sehr einfacher Szenengraph und es werden keine Culling-Techniken unterstützt.

Der `ApplicationContext` kann als einziges Modul des Kerns auch Serialisiert, also abgespeichert und wieder geladen werden. Somit kann ein gesamter Status einer Anwendung gespeichert werden. Das Thema Serialisierung wird im nächsten Abschnitt 4.7.1 noch genauer erläutert.

4.7 Peripherie

Bisher wurde der Begriff Objekt für ein von Future Office visualisierbare Datenstruktur verwendet. In diesem Abschnitt wird der Begriff Objekt für ein Objekt aus der Programmierung, also eine Instanz einer Klasse, verwendet.

4.7.1 Serialisierung

Hinter dem Begriff Serialisierung steckt die Idee persistente Objekte zu erzeugen, d.h. Objekte, die beim Beenden eines Programms gespeichert und beim Starten des Programms wieder geladen werden können. Dies ist natürlich nicht nur beim Beenden und Starten sinnvoll, sondern auch immer dann, wenn Objekte gespeichert und wieder geladen werden sollen. Dieser Prozess des Speicherns und Ladens von Objekten wird als Serialisierung bezeichnet [2, S. 391]. Im Future Office wird die Serialisierung bei all jenen Objekten eingesetzt, die vor allem Daten beinhalten. Das sind natürlich alle Future Office Objekte, Matrizen, Texturen, Events und auch der Szenengraph. Diese Objekte müssen alle von der Klasse `Serializable` abgeleitet werden. `Serializable` beinhaltet alle Schnittstellen, die ein Serialisierbares Objekt implementieren muss, damit es abgespeichert werden kann. Anders als bei anderen Serialisierungssystemen werden im Future Office die Objekte aber nicht sofort in einen Byte- oder Character-Stream geschrieben, sondern zunächst in einen XML-DOM Baum. Erst danach wird, durch die Verwendung von XML-spezifischen Funktionen der ClanLib, der Byte- oder Zeichen-Code erzeugt. Dieser kann dann in einen beliebigen ClanLib Ausgabestream (`CL_OutputSource`) geschrieben werden. Das kann eine

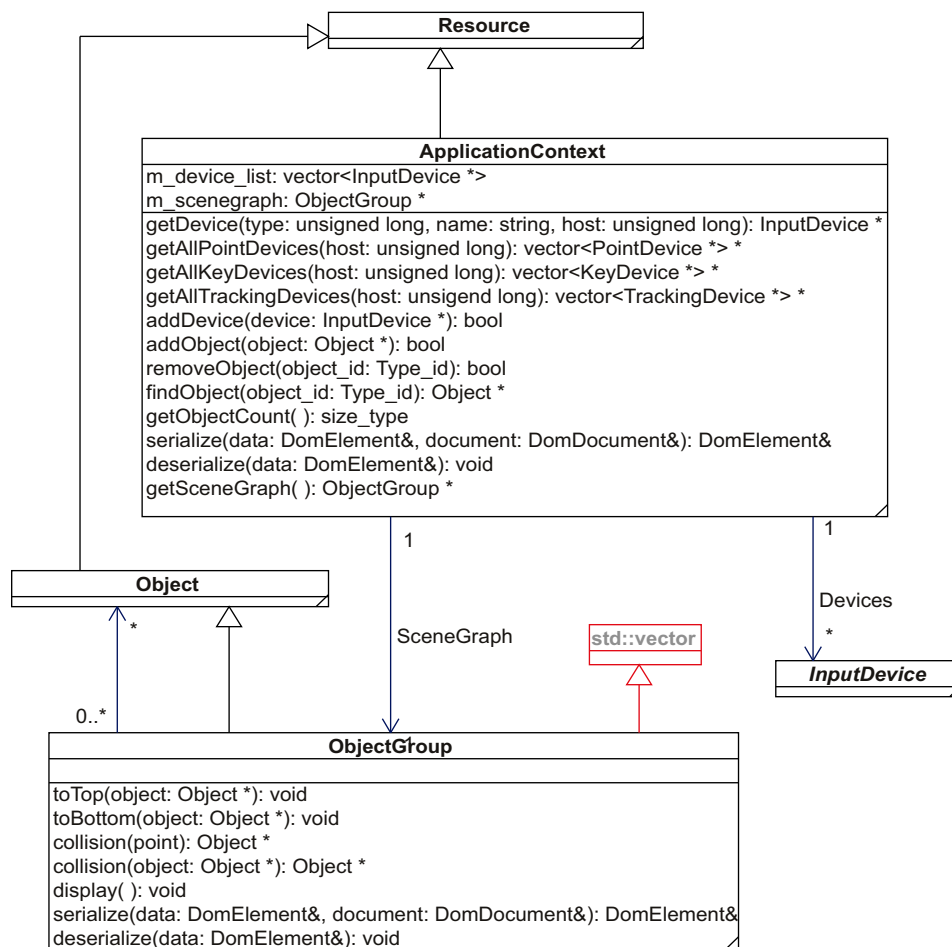


Abbildung 4.6: Klassendiagramm des Application-Kontexts.

normale Datei oder auch eine ZIP-komprimierte Datei sein, aber auch ein Netzwerkstream oder Packet.

Das Laden oder De-Serialisieren funktioniert genau umgekehrt, doch kommt es hier zu einem Problem. Wird ein Datei ausgelesen, so ist ohne zusätzliche Information nicht erkennbar, zu welcher Klasse die gelesenen Daten gehören, also welches Objekt erzeugt und deserialisiert werden soll. Diese Information wird über ein eigenes RTTI (Run Time Type Information) System zur Verfügung gestellt [3, Kap. 12]. Dieses fügt über ein Makro in jede serialisierbare Klasse die Funktion `getClassType` ein, die den Klassennamen zurückgibt. Weiters muss für jede serialisierbare Klasse ein Factory-Objekt (abgeleitet von `serializeCreatorBase`) erzeugt werden. Das Factory-Objekt hat dabei nur die Aufgabe ein Objekt der entsprechenden Klasse zu erzeugen. Damit der `ResourceManager` nun die richtigen Objekte erzeugen kann, wird jedem Klassennamen ein Factory-Objekt zu-

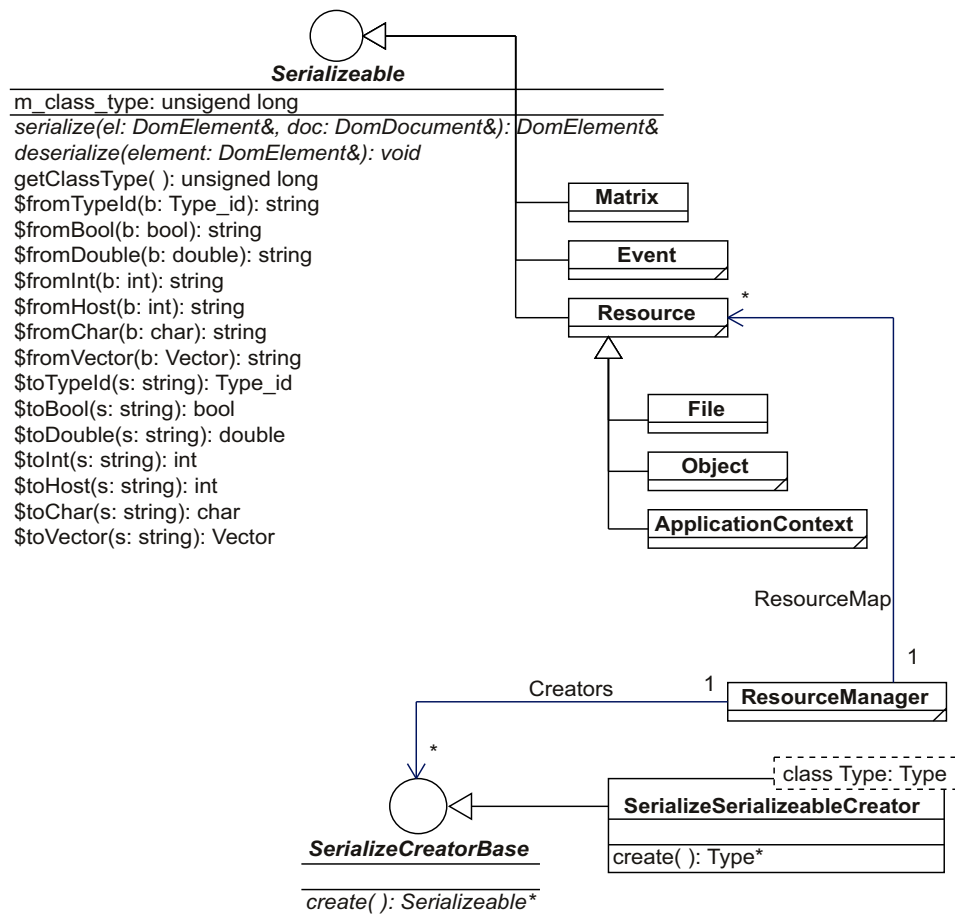


Abbildung 4.7: Klassendiagramm der Serialisierung.

geteilt. Somit können über die abgespeicherten Klassennamen die richtigen Objekte erzeugt und deserialisiert werden.

Das Laden und Speichern von Objekten übernimmt die Ressourcenverwaltung. Durch die `save-` und `load-`Methoden kann die Anwendung Objekte auf eine Datei abspeichern oder von einer Datei laden.

4.7.2 Event-System

Zur Kommunikation zwischen Objekten wird das Signal/Slot System von ClanLib verwendet. Oft soll eine Veränderung in einem Objekt einem anderen Objekt mitgeteilt werden. Zum Beispiel will die Anwendung wissen, wenn sich die Maus bewegt hat. Die Anwendung und die Maus sind beides Objekte. Diese Kommunikation kann auch über Callback-Funktionen erledigt werden, doch haben diese zwei große Nachteile. Zunächst sind sie nicht „type safe“, d. h. es ist nie sicher, ob die Callback-Funktion mit den

richtigen Argumenten aufgerufen wird. Zweites ist die Callback-Funktion stark mit der aufrufenden Funktion verbunden, da diese wissen muss welche Callback-Funktion aufgerufen werden muss.

Signals und Slots sind eine Alternative zum Callback Konzept. Ein Signal wird ausgelöst, wenn sich der Status eines Objekt verändert hat, und dieses Ereignis für andere Objekte interessant sein könnte. Das ist auch das Einzige, das das Objekt von seiner Umwelt wissen muss. Das Objekt weiß nicht wie viele Empfänger das Signal hat und welche Funktionen aufgerufen werden um dieses Signal zu verarbeiten. Um ein Signal abzufangen muss ein Slot mit diesem Signal verbunden werden. Ein Slot ist ein Empfänger von Signalen und ist aber auch eine normale Methode, ähnlich einer Callback-Funktion. Ein Slot weiß aber auch nicht, ob er zu einem Signal verbunden ist. Die Verbindung erledigt das Slot Objekt der ClanLib. Ein Slot Objekt verbindet einen Slot (Methode) mit einem entsprechenden Signal. Wird das Signal ausgelöst, so wird der Slot (Methode) vom Slot Objekt gestartet. Diese Verbindung wird durch das Löschen des Slot Objekts wieder getrennt [1].

Mit den Signalen können zusätzlich auch noch Parameter übertragen. Dies übertragenen Parameter sind im Future Office Netzwerk und bei den Eingabegeräten Events. Die Events beinhalten je nach Eingabegeräte verschiedene Informationen. Zum Beispiel sind bei einem Zeigergerät (`PointDevice`) wie der Maus, Informationen über die Mauskoordinaten und die gedrückten Maustasten in einem Event verpackt. Zusätzlich zu den Events der Eingabegeräte gibt es noch spezielle Netzwerkevents, die zur Kommunikation zwischen Anwendungen auf verschiedenen Rechnern benützt werden können oder um Future Office Objekte auf anderen Rechnern zu verändern.

4.8 Plugins – Die Schnittstellen zur Außenwelt

Damit das Future Office überhaupt verwendbar wird, benötigt man Schnittstellen über die man das Framework verwenden, steuern und erweitern kann. Diese Schnittstellen kann man über Plugins, die das Future Office Plugin System lädt, verwenden.

Es werden drei Arten von Plugins unterschieden:

- Anwendungs-Plugins: Die Anwendung an sich.
- Input-Device-Plugins: Zusätzliche Eingabegeräte.
- Object-Plugins: Zusätzliche Objekte oder Object-Viewer

Von all diesen Plugins können beliebig viele vorhanden sein, es kann aber immer nur ein Anwendungs-Plugin geladen sein. Bei den Input-Device-Plugins und Object-Plugins sind in der Regel alle geladen. Wie diese Plugins

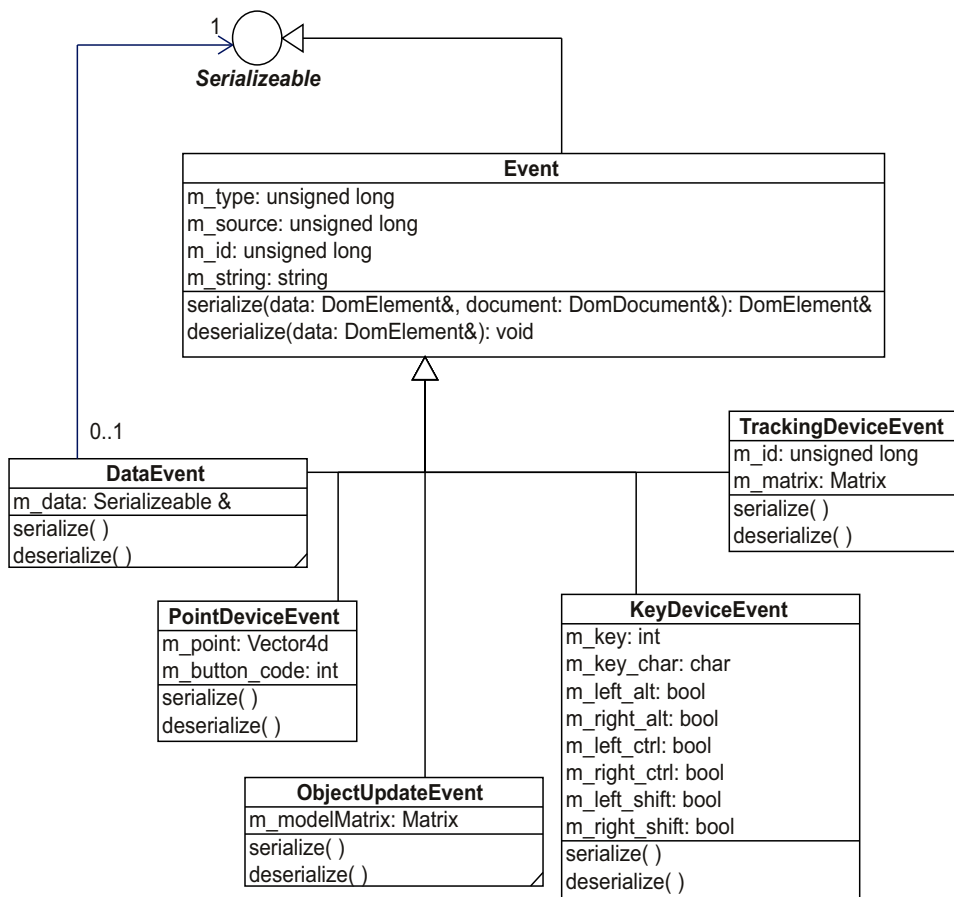


Abbildung 4.8: Klassendiagramm der Events.

nun funktionieren, und was ihre Aufgaben sind wird in den folgenden Seiten beschrieben.

4.8.1 Application – Anwendungs-Plugin

Das Anwendungsplugin steuert das Framework und ist Hauptverantwortlich für die Interaktion mit dem Benutzer. Die UNO-Spiellogik des Prototyps wäre hier angesiedelt. Dies ist auch ein Bereich der sehr leicht verständlich und handhabbar sein soll, weil hier neue Anwendungen hinzugefügt werden können. Damit das möglich ist, sind hier sehr strikte Schnittstellen definiert, mit denen man seine Anwendung in das Framework einbinden kann. Bei der Planung dieser Schnittstellen wurde versucht diese an GLUT anzulehnen, damit das Einlernen einfacher wird. Diese Schnittstellen sind Methoden die von der Anwendung überschrieben werden müssen und dann vom Framework ausgeführt werden. Der Anwendungsentwickler schreibt so nur kurze Methoden, die dann am entsprechenden Zeitpunkt aufgerufen

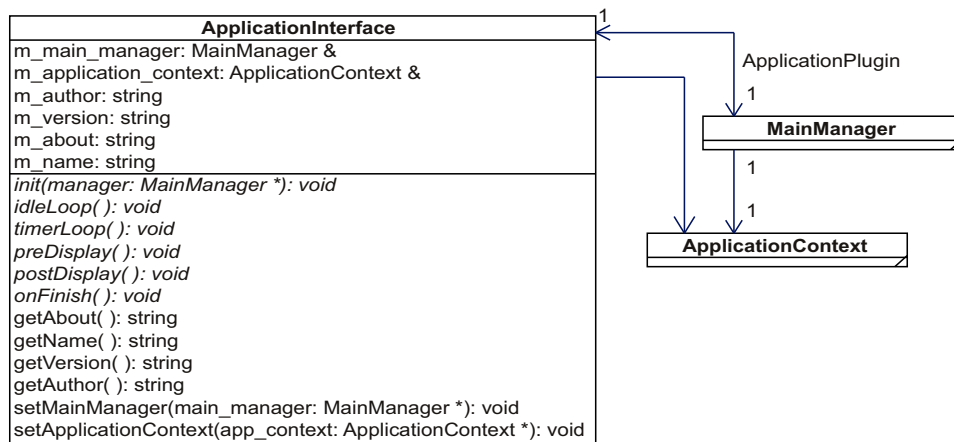


Abbildung 4.9: Klassendiagramm des Application Plugins.

werden. Die Anwendung ist in eine Klasse eingebettet die dann bei Laufzeit vom `MainManager` geladen und ausgeführt wird.

4.8.2 InputDevice – Eingabegeräte-Plugin

Mit Input-Device-Plugins wird es ermöglicht, das Future Office um eigenen Eingabegeräten zu erweitern. Standardmäßig verfügt das Framework Input-Device-Plugins für alle Standardeingabegeräte, also für Maus und Tastatur. Mit der Entwicklung eigener Plugins ist es auch möglich z. B. eine virtuelle Tastatur oder eine „Laser-Pointer-Maus“ einzubinden und diese dann in einer Anwendung zu verwenden. Die Input-Device-Plugins haben aber zum Unterschied zum Anwendungsplugin nur einen viel kleineren Spielraum. Es sind so auch nur weniger Schnittstellen zum Framework vorhanden und es gibt nur drei Typen von Input Devices, wobei jeder ein fixes Rückgabeformat hat. Diese sind:

- `PointDevice`: Zeigergeräte, z.B. eine Maus.
- `KeyDevice`: Tastaturen.
- `TrackingDevice`: Trackinggeräte die Matrizen erzeugen, z.B. ARToolkit.

Besondere Input-Devices sind die Remote-Input-Devices. Diese ermöglichen es Eingabegeräte von anderen Rechnern einzubinden oder ein lokales Eingabegerät für den Zugriff von einem anderen Rechner „freizugeben“. Für jeden Typ von Eingabegeräte gibt es ein eigenes Remote-Input-Device, dass für die Bearbeitung und das Weiterleiten der Events des jeweiligen Geräts entwickelt worden ist. Um die Events eines Eingabegeräts über das Netzwerk zu übertragen, muss beim Sender wie beim Empfänger ein Remote-Input-Device des jeweiligen Typs erzeugt werden.

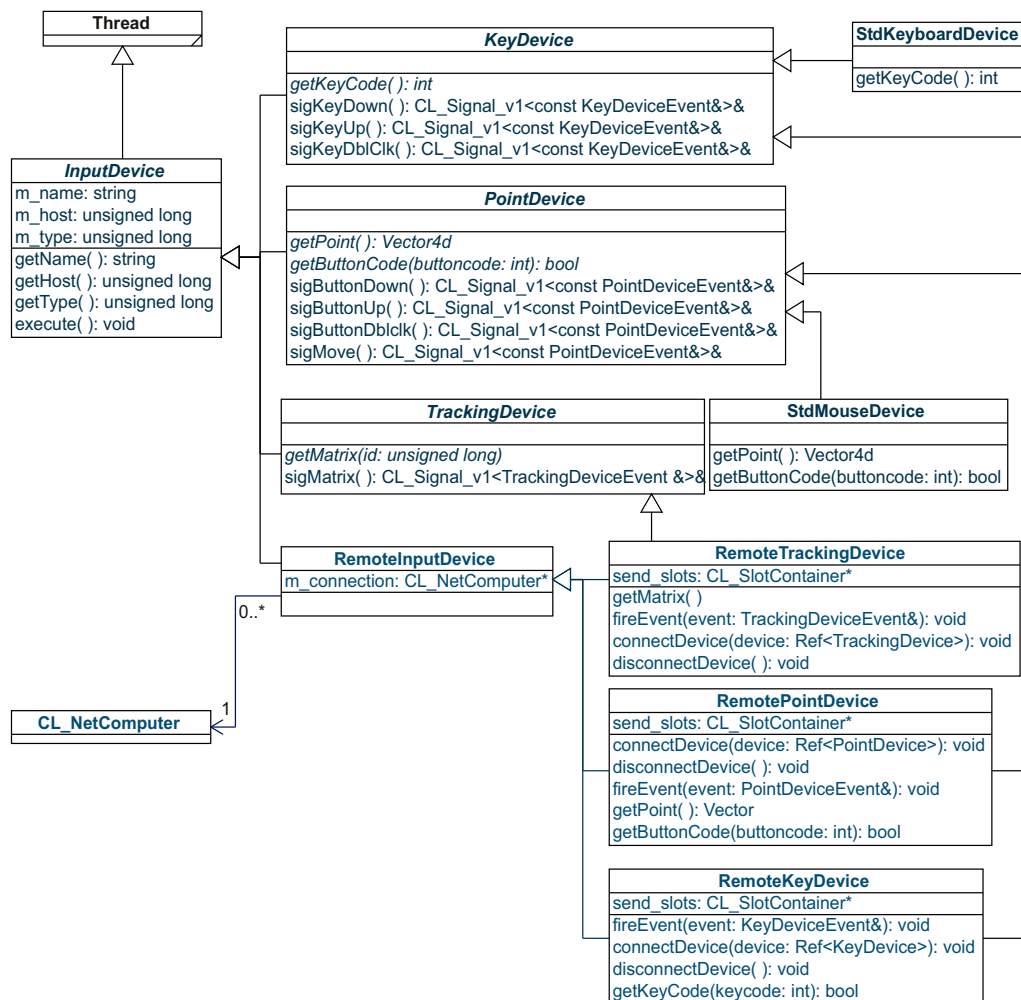


Abbildung 4.10: Klassendiagramm der Eingabegeräte.

4.8.3 Object – Objekt-Plugin

Die dritte und letzte Möglichkeit in das Framework einzugreifen sind die Object-Plugins oder auch Object-Viewer-Plugins. Mit diesen ist es möglich die Darstellung von Objekten zu verändern, bzw. überhaupt sinnvoll zu machen. Ein Beispiel für einen Object Viewer Plugin wäre ein Renderer für 3D Studio Max Dateien, oder auch PDF Dateien.

Der Viewer ist die einfachste Form eines Object Plugins, da er sich nur um die Darstellung und Kollisionserkennung kümmern muss. Möglich wäre es aber auch ein verbessertes Szenengraph Object zu entwickeln, das einige Culling Techniken beinhaltet. Object-Plugins haben dementsprechend viele Schnittstellen zum Framework, die aber nicht alle verwendet werden müssen. So müssen beim einen normalen Object-Viewer meistens nur drei bis fünf

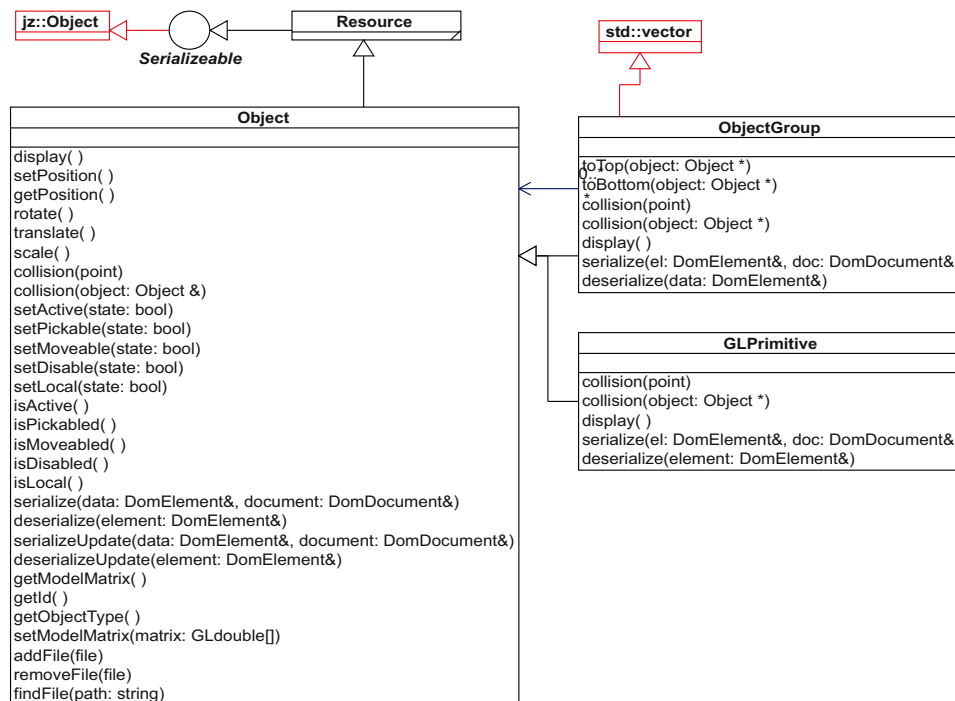


Abbildung 4.11: Klassendiagramm der Objekte.

Methoden implementiert werden.

4.9 Status quo

Zur Zeit der Erstellung dieses Dokuments (Mitte Jänner 2005) wurde zum erste Mal die, am Anfang des Kapitels erwähnte, Brainstorming Anwendung eingebaut. Dabei ergaben sich schon die ersten größeren Probleme. Das Brainstorming Tool verwendet als GUI die in ClanLib vorhandene. Aus einen uns zunächst nicht erkennbaren Grund konnte diese aber von dem Renderer Thread nicht angezeigt werden. Nach längeren Probieren stellte sich heraus, dass das ClanLib GUI unbekannte Probleme mit dem Multithreading hat. Diese Probleme konnten wir nicht beheben, daher entschieden wir uns den Renderer nicht mehr als Thread auszuführen, sondern in der Hauptschleife des `MainManagers` zu starten. Diese Veränderung brachte den gewünschten Erfolg, wenn gleich das keine gute Lösung ist.

Zum weiteren Entwicklungsstand: Die Entwicklung ist soweit fortgeschritten, dass alle Module bereits erstellt, aber noch nicht im vollen Umfang entwickelt worden sind. Es ist aber dennoch schon möglich das Future Office Plugin System zu verwenden, wenn auch in einem eingeschränkten Umfang.

Folgende Teile sind noch nicht oder nur Teilweise implementiert:

- Laden von Plugins.
- Collision Detection.
- Timerloop für die Anwendung.
- Zugriffsfunktionen auf entfernte Devices im Application Context.
- Remote-Devices für Tracking- und Key-Devices.

Auch das automatische Finden von Servern im Netzwerk funktioniert noch nicht korrekt. Es dürfte noch ein Fehler in der Suchfunktion des Clients vorhanden sein. Der **EchoServer** des Servers funktioniert einwandfrei und wenn man beide auf einen einzelnen Rechner ausgeführt, funktionieren beide einwandfrei.

Kapitel 5

Ausblick

Das Future Office wird 25. Jänner 2005 im Rahmen der PRO5 Projektpräsentationen das erste Mal einer größeren Zuschauermenge präsentiert werden. Bis zu diesem Zeitpunkt sollte eine präsentierbare Anwendung lauffähig sein. Eine Integration des Brainstorming Tools erscheint realistisch. Zusätzlich wird zurzeit auch an einem ARToolKit Tracking Device gearbeitet, das Möglicherweise auch schon zu diesem Termin fertig sein wird.

Am 10. Februar 2005 wird das Future Office in Christchurch, Neuseeland beim Consortium Meeting des HIT Lab NZ vorgestellt. Dort sollte die Brainstorming Anwendung, die virtuellen Tastatur und das ARToolkit Tracking am Future Office laufen. Nach diesem Termin wird das Future Office von Leitner Jakob und dem Autor, im Rahmen eines Auslandspraktikums bei HIT Lab NZ, zumindest vier Monate weiterentwickelt werden. In diesen vier Monaten sollten alle noch fehlende Teile des Future Office Plugin Systems entwickelt werden.

Anhang A

HOWTOS

A.1 Application Plugins

Um eine Future Office Anwendung zu schreiben, ist an-und-für-sich nur eine eigene Klasse von der Klasse `ApplicationInterface` abzuleiten und alle notwendigen Funktionen zu überschreiben. Aber fangen wir zunächst mit einer kleinen Einleitung an.

Das Application Plugin wird vom `MainManager` geladen und gestartet. Dieser erzeugt das Application Objekt und Initialisiert es indem er die `init`-Funktion der Anwendung ausführt. Das Ausführen der `init`-Funktion geschieht zu einem Zeitpunkt an dem schon alle anderen Objekte im Future Office System erstellt sind, also es kann schon darauf zugegriffen werden (siehe Abbildung 4.3). Ab diesen Zeitpunkt sollte die Anwendung vollständig initialisiert und lauffähig sein, denn nun kann jederzeit das Rendering starten und mit ihm die ersten Aufrufe von Loop-Funktionen.

Bei Beenden der Anwendung ruft der `MainManager` die Funktion `onFinish()` auf. Nach dieser wird erst der Destructor aufgerufen werden.

A.1.1 Hello World!

```
#ifndef DemoApplication_H
#define DemoApplication_H

#include "ApplicationInterface.h"

namespace ggl{

class DemoApplication : public ApplicationInterface{
private:
    CL_Slot slot1;
public:
    DemoApplication(){}
    virtual ~DemoApplication(){}
};
};
```

Zunächst werden alle notwendigen Header Dateien eingebunden. Die `ApplicationInterface.h` beinhaltet die Basisklasse `ApplicationInterface` für Anwendungen. Da sich alle Future Office Bezeichner im Namespace `ggl` befinden, geben wir unser Hello World auch in diese. Es wäre auch möglich allen Future Office Funktionen ein `ggl` voranzustellen, oder ein `using namespace ggl` zu verwenden. Wir definieren zunächst mal ein `CL_Slot` Objekt für eine spätere Verbindung mit einem Signal. Der Konstruktor und Destruktor ist leer.

```
virtual void init(MainManager *manager, jz::Ref<ApplicationContext> app_context){
    ApplicationInterface::init(manager, app_context);
    if(KeyDevice *k =
        (KeyDevice *) (app_context->getDevice(KEY_DEVICE, "stdkeyboard").get()) ){
        slot1 = k->sigKeyUp.connect(this, &DemoApplication::onKeyDown);
    }
}
```

Dies ist die erste Funktion die aufgerufen wird. Der Mainmanger übergibt der Anwendung somit einen Zeiger auf sich selbst und eine Referenz auf den Applikations Kontext. Als erstes müssen diese Parameter an die `init`-Funktion der Basisklasse übergeben werden. Diese Funktion speichert diese Zeiger für spätere Verwendung ab. Die nächsten beiden Zeilen sind etwas komplizierter. Im der ersten wird das lokale Standard-Key-Board-Device, dass immer den Name „stdkeyboard“ hat, vom Applikations Kontext geholt. Da `getDevice` eine Referenz auf NULL zurückgibt wenn das Device nicht gefunden worden ist, muss dies noch überprüft werden, bevor man darauf zugreifen kann. Mit der Methode `get()` erhält man den Zeiger einer Referenz.

In der zweiten Zeile wird die Funktion `DemoApplication::onKeyDown` mit dem Signal `sigKeyUp` des Keyboard Devices verbunden. Diese Ver-

bindung wird wieder getrennt, wenn das `CL_Slot`-Objekt `slot1` aus dem Speicher gelöscht wird.

```
virtual void idleLoop(){
    std::cout << "Hello idleLoop" << std::endl;
}

virtual void preDisplay(){
    std::cout << "I'm called before scenegraph rendering" << std::endl;
}

virtual void postDisplay(){
    std::cout << "I'm called after scenegraph rendering" << std::endl;
}

virtual void timerLoop(){
    std::cout << "I'm called in a timer interval" << std::endl;
}

virtual void onFinish(){
    std::cout << "It's over" << std::endl;
}
```

`idleLoop`, `preDisplay`, `postDisplay` und `timerLoop` sind die sogenannten „Loop-Methoden“. Diese werden im Normalfall in einer Schleife aufgerufen. `idleLoop` wird im Renderer Thread bei jedem Schleifendurchlauf am Anfang ausgeführt. Danach überprüft der Renderer ob die Szene neu gerendert werden muss oder nicht. Wenn dies der Fall ist dann wird zunächst `preDisplay` aufgerufen, dann wird der Szenengraph gerendert und am Schluss wird `postDisplay` aufgerufen.

Die Funktion `timerLoop` wird im Application Runner Thread ausgeführt. Dieser wird über den `MainManager` gesteuert und erlaubt einen Zeitinterwal gesteuerten Aufruf der Funktion.

Die letzte Funktion `onFinish` wird beim Beenden der Anwendung vom `MainManager` aufgerufen. Diese Funktion ist das Gegenstück zur `init` Funktion.

```
void onKeyDown(const KeyDeviceEvent &key){
    if(key.m_key == CL_KEY_ESCAPE){
        m_main_manager->finish();
    }
}

};
}

#endif // DemoApplication_H
```

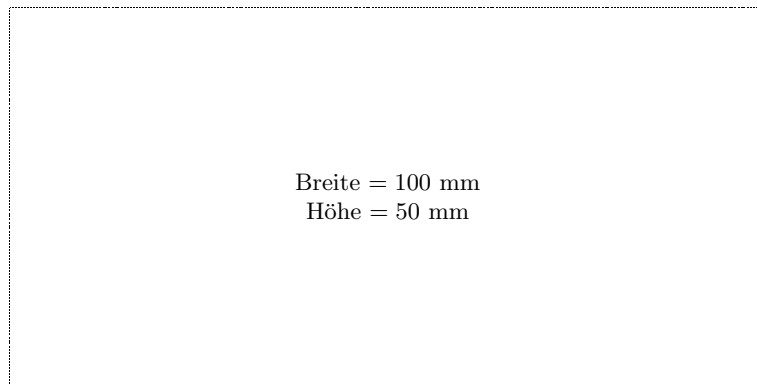
Am Ende noch ein so genannter Slot, oder einfacher, ein Eventhandler. Diese Funktion wird bei jedem Tastendruck der Tastatur ausgeführt. Mit dem Aufruf der Funktion `m_main_manager->finish()` wird die Anwendung beendet und somit auch die Funktion `onFinish` aufgerufen.

Literaturverzeichnis

- [1] GANGSTO, K.: *ClanLib - Overview - ClanLib Game SDK*. URL, www.clanlib.org/docs/Overview, 2004. Kopie auf CD-Rom.
- [2] KRUGLINSKI, D., S. WINGO und G. SHEPHERD: *Inside Visual C++ 6.0*. Microsoft Press, Redmond, Washington, USA, 1998.
- [3] LLOPIS, N.: *C++ for Game Programmers*. Charles River Media, Inc., Hingham, Massachusetts, USA, 2003.
- [4] MILGRAM, P. und F. KISHINO: *A Taxonomy of Mixed Reality Visual Displays*. IEICE Transactions on Information Systems, E77-D(12):1321–1329, December 1994.
- [5] RASKAR, R., G. WELCH, M. CUTTS, A. LAKE, L. STESIN und H. FUCHS: *The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays*. Computer Graphics, 32(Annual Conference Series):179–188, 1998.
- [6] REKIMOTO, J. und M. SAITOH: *Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments*. In: *CHI*, S. 378–385, 1999.
- [7] REKIMOTO, J. und M. SAITOH: *Augmented Surfaces: A Spatially Continuous Work Space for Hybrid Computing Environments*. URL, www.csl.sony.co.jp/person/rekimoto/movies/chi99.mpg, 1999. Kopie auf CD-Rom.
- [8] SCHMALSTIEG, D., A. FUHRMANN, G. HESINA, Z. SZALAVÁRI, L. M. ENCARNÇÃO, M. GERVAUTZ und W. PURGATHOFER: *The Studierstube Augmented Reality Project*. PRESENCE - Teleoperators and Virtual Environments, (1), 2002.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —