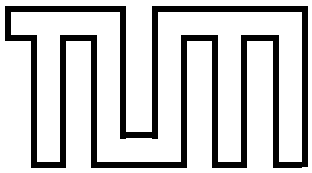
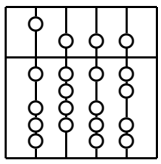


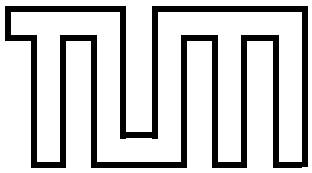
Technische Universität
München
Fakultät für Informatik



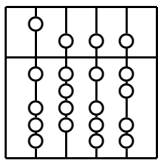
Systementwicklungsprojekt

Erstellen einer DWARF-basierten Darstellungskomponente für 3D Szenen auf dem iPAQ

Marco Feuerstein



Technische Universität
München
Fakultät für Informatik



Systementwicklungsprojekt

Erstellen einer DWARF-basierten Darstellungskomponente für 3D Szenen auf dem iPAQ

Marco Feuerstein

Aufgabensteller: Univ-Prof. Bernd Brügge, Ph.D.

Betreuer: Dipl.-Inform. Christian Sandor

Abgabedatum: 15. November 2002

Erklärung

Ich versichere, dass ich diese Ausarbeitung des Systementwicklungsprojektes selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. November 2002 Marco Feuerstein

Zusammenfassung

Diese Ausarbeitung des Systementwicklungsprojektes beschreibt Entwurf und Implementation einer DWARF-basierten 3-D Anzeigekomponente für den Taschencomputer HP (früher Compaq) iPAQ. Die Anzeigekomponente basiert auf dem Open Source VRML Browser FreeWRL, welcher über das External Authoring Interface (EAI) an einen Adapter angebunden ist, der mit der DWARF Middleware kommuniziert.
Bei der Implementierung kommt die Technologie von GCJ (GNU Compiler for Java) zum Einsatz, welche das Mischen von Java- und C/C++-Code erlaubt und das Erzeugen von performantem, nativem Maschinencode ermöglicht. Letzteres kommt insbesondere dem iPAQ zugute, dessen Prozessor leistungsschwächer als aktuelle Desktop-Prozessoren ist und somit mit möglichst wenig Rechenleistung zu belasten ist, um optimale Performanz zu erzielen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung - Der iPAQ als 3D tauglicher PDA

Die Computer- und Informationstechnologie richtete in den letzten Jahren ihr Augenmerk unter anderem auf folgende zukunftsweisende Forschungsrichtungen: Die Entwicklung immer kleinerer, mobiler Computer sowie die performante Darstellung dreidimensionaler (3D) Graphiken. Die rasante Entwicklung auf diesen Gebieten führte zu immer leistungsfähigeren, etwa handflächengroßen Taschencomputern. Diese Taschencomputer, auch *Personal Digital Assistants* (PDAs), *Palmtops* oder *Handhelds* genannt, spielen eine zunehmend wichtige Rolle in Feldern wie *Virtual Reality* (VR) und *Augmented Reality* (AR).
In einem VR System taucht der Benutzer gewissermaßen in eine auf ihn reagierende virtuelle Welt ein.[1] Diese virtuelle Welt zeichnet sich dadurch aus, real auszusehen, zu erklingen, sich gar real anzufühlen, und auf Benutzerinteraktionen in Echtzeit antworten zu können.[2]
Ein AR System hingegen erweitert die reale Welt um virtuelle (Computer-generierte) Objekte, welche im gleichen Raumgefüge wie die reale Welt zu koexistieren scheinen. Das AR System hat die Eigenschaften, dass es

- reale und virtuelle Objekte in einer realen Umgebung kombiniert
- interaktiv und in Echtzeit ausgeführt wird sowie
- reale und virtuelle Objekte aneinander ausrichtet.(nach [3])

Gerade in virtueller beziehungsweise erweiterter Realität ist nun ein extrem unhandlicher Desktop PC oft von minderem Nutzen dafür, Arbeiten unter höchstmöglicher Bewegungs- sowie Barrierefreiheit durchzuführen.
Einer der derzeit leistungsfähigsten und beliebtesten Taschencomputer ist der HP (früher Compaq) iPAQ, was ihn zur idealen Entwicklungsplattform für schnurlose und tragbare Anwendungen werden lässt. Dies spiegelt sich auch auf den zahlreichen Entwicklerseiten wider, die im Internet über den iPAQ zu finden sind.

1.1 Motivation

Die Idee dieses Systementwicklungsprojektes entstand während des Designs der Benutzerschnittstelle des *Distributed Wearable Augmented Reality Frameworks* (DWARF, [4]). Wie das Auflösen des Akronyms in seine Bestandteile schon vermuten lässt, handelt es sich bei DWARF um eine Architektur für verteilte, mobile AR Anwendungen auf Taschencomputern. Sie wurde am Lehrstuhl für Angewandte Softwaretechnik an der Technischen Universität München entwickelt. Auf der Mixed und Augmented Reality Konferenz ISMAR 2002 sollen die zahlreichen Möglichkeiten von DWARF durch eine anschauliche und aussagekräftige Demonstration in Form eines Spieles dem Publikum nähergebracht werden. Später soll DWARF dann als Open Source freigegeben werden.
Die Architekur von DWARF schließt intelligente Module ein, welche gewisse Fähigkeiten bereitstellen beziehungsweise von anderen Modulen beziehen. Eines dieser Module ist die mobile, tragbare Anzeigekomponente für dreidimensionale Szenen, welche in diesem Systementwicklungsprojekt entwickelt werden soll.

1.2 Struktur des Dokumentes

Dieses Dokument lehnt sich an die in [6] beschriebene Struktur, welche Softwaretechnikrelevanten Gesichtspunkten entsprechen soll. Die zu erstellende Anzeigekomponente konkretisiert sich mit Fortschreiten des Dokuments. Die Kapitel des Dokumentes sind im Einzelnen:

Kapitel 2: *Anforderungen an die Darstellungskomponente* beschreibt unter Darstellung von Szenario und Anwendungsfällen, welche Bedingungen die zu erarbeitende Darstellungskomponente erfüllen muss.

Kapitel 3: *Verwandte Arbeiten* präsentiert aktuelle Projekte und Entwicklungen, die mit dem Thema der Arbeit in engerer Beziehung stehen.

Kapitel 4: *System Design* erklärt das Konzept des ganzen Systems und desses Unterteilung in Untersysteme. Dabei wird auch dargelegt, wie das Systementwicklungsprojekt in das System einzuordnen ist.

Kapitel 5: *Implementation* gibt einen detaillierten Überblick darüber, wie die Darstellungskomponente und ihre Bestandteile implementiert sind.

Kapitel 2

Anforderungen an die Darstellungskomponente

Die 3D-Anzeigekomponente, die bei der Demonstration zum Einsatz kommt, muss gewisse Anforderungen erfüllen. Dazu betrachtet man auch das Szenario und die Anwendungsfälle der Demonstration, welche auch *SHEEP: The Shared Environment Entertainment Pasture*[5] heißt und im Folgenden kurz das SHEEP System genannt wird.

2.1 Szenario

Das Spielfeld des SHEEP Systems besteht aus einer Landschaft mit Hügeln, Feldern, Hecken, Flüssen und Brücken, in der sich Schafe und Wölfe aufhalten. Die Landschaft und ihre Tiere werden über einen an der Decke befestigten Projektor auf einen Tisch projiziert. An der Demonstration teilnehmende Spieler können anhand verschiedener Geräte auf mehrere Arten am Spiel teilhaben:

- Ein Interaktionsgerät für die Spieler repräsentiert der getrackte iPAQ, mit welchem man Schafe fangen, bearbeiten und wieder auf das Spielfeld setzen kann.
- Am Rande des Spielfeldes sind außerdem mehrere getrackte Laptops angebracht, welche benutzt werden, das Spielgeschehen dreidimensional betrachten zu können. Sie werden auch *See-through Laptops* genannt.
- Jeder Zuschauer kann auch sogenannter Zauberer werden, indem er einen Kopfhörer mit Mikrofon aufsetzt und einen getrackten Zauberstab zur Hand nimmt. Über Spracheingabe und mit dem Zauberstab kann er das Spielgeschehen manipulieren.
- Hat der Zuschauer einen Netzwerk-fähigen Laptop, kann er auch damit am Spiel teilhaben, indem er nach Installation der notwendigen Software die Möglichkeit hat, ein Schaf zu steuern.

- In der Spielewelt existiert auch ein sogenannter Gott, welcher ein Head Mounted Display (HMD) trägt und dessen Hände getrackt werden. Über im HMD dargestellte Objekte, Gestik oder Spracheingabe kann er Elemente der Landschaft hinzufügen, bearbeiten oder entfernen.

Abbildung 2.1: Die Interaktionsmöglichkeiten des SHEEP Systems

Das SHEEP System stützt sich auf DWARF und die mit diesem einhergehenden DWARF Services, welche über eine Peer-To-Peer Umgebung dynamisch miteinander kooperieren. Diese binden mobile Geräte, so auch die Laptops von Zuschauern, in das Spiel ein.

2.2 Anwendungsfälle

Folgende Anwendungsfälle ergeben sich für den iPAQ, welcher zur Anzeige und Manipulation der Landschaft und ihrer Tiere verwendet wird:

- Mit dem iPAQ kann ein Schaf vom Tisch entfernt werden, indem man sich mit dem iPAQ in die unmittelbare Nähe des Schafes begibt und dieses einfängt. Das gefangene

Schaf kann nun von allen Seiten betrachtet werden und wieder auf den Tisch gesetzt werden.

- Der iPAQ kann dreidimensionale Szenen schnell darstellen und jegliche Veränderungen in der Szene ohne Verzögerungen aktualisieren.
- Der Benutzer kann Schafe mit dem iPAQ anmalen, scheren und bewegen.

2.3 Resultierende Anforderungen

Die sich aus den Anwendungsfällen und dem Szenarion ergebenden Anforderungen unterteilen sich in funktionale, nichtfunktionale und Pseudo Anforderungen.

2.3.1 Funktionale Anforderungen

Funktionale Anforderungen beschreiben das Zusammenspiel zwischen dem Softwaresystem und dessen Umgebung, also Benutzern und anderen externen Systemen:

- Die Graphiken sollen in 3D angezeigt werden.
- Die Anzeigekomponente muss Veränderungen in der Szene mit ihrer Umgebung synchronisieren können.

2.3.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderugen sind für den Benutzer sichtbare Anforderungen, welche sich nicht auf das funktionale Verhalten des Systems beziehen:

- Schnelligkeit und Effizienz sollen das System auszeichnen, um dem Benutzer kurze Wartezeiten zu gewährleisten.
- Dem Benutzer muss eine angenehme und leichte Manipulation der Objekte der Szene gestattet werden, wozu eine entsprechende Benutzerschnittstelle erforderlich ist.

2.3.3 Pseudo Anforderungen

Vom Aufgabensteller oder Kunden vorgegebene Anforderungen nennt man Pseudo Anforderungen. Die für dieses Systementwicklungsprojekt unabdingbaren Anforderungen sind die Nutzung vorhandener Hardware und freier Software, die zu verwendende Augmented Reality Middleware sowie die Art der Dokumentation:

- **Hardware:** Der iPAQ Taschencomputer muss verwendet werden, welcher mit einem StrongARM-Prozessor ausgestattet ist.
- **Betriebssystem:** Als Betriebssystem muss Linux verwendet werden.

- **DWARF:** Die Anzeigekomponente muss an die DWARF Middleware angebunden werden.
- **Dokumentation:** Diese Ausarbeitung soll nach der "Object Oriented Analysis and Design" Methodik geschrieben werden, also Software Engineering Ansprüchen genügen, und Sourcecode ausreichend dokumentiert sein.

Kapitel 3

Verwandte Arbeiten

Im Folgenden werden nun mit der 3D Darstellung verwandte Arbeiten und Technologien genauer betrachtet, die zum Zeitpunkt der Erstellung des Systementwicklungsprojektes in Entwicklung sind oder schon abgeschlossen wurden.

3.1 VRML

Die *Virtual Reality Modeling Language* (VRML) ist eine Sprache zur Beschreibung von 3D Bildfolgen und wird in [10] spezifiziert. Für den Benutzer besteht die Möglichkeit der Interaktion mit den 3D Bildern. Um in VRML definierte Szenen darstellen zu können, benötigt man sogenannte VRML Viewer oder VRML Browser.
Da bei diesem Systementwicklungsprojekt zur Darstellung der 3D Szenen möglichst auf vorhandene Open-Source Programme zurückgegriffen werden soll, werden nun frei verfügbare VRML Browser genauer auf deren Einsatzfähigkeit und Tauglichkeit auf Taschencomputern im DWARF-System untersucht. Zuerst aber wird eine kurze Erklärung gegeben, wie der Benutzer mit einer in VRML definierten Szene interagieren kann.

3.1.1 EAI

Das *External Authoring Interface* (EAI) ist eine Schnittstelle des VRML Browsers zur Außenwelt. Durch diesen in der VRML 97 Spezifikation (siehe [11] und [12]) festgelegten Annex wird einer statischen VRML Szene Interaktivität und Dynamkik verliehen. Das EAI ist ein Application Programming Interface (API), welches eine allgemeingültige Schnittstelle definiert, durch die über genau festgelegte Befehle Java Applikationen oder Applets mit dem Browser kommunzieren sowie auf Objekte in der VRML Szene in Form von Knoten zugreifen und diese verändern können. Abbildung 3.1 veranschaulicht dies.

3.1.2 VRML Browser

Die durch VRML beschriebenen Szenen werden in einer Textdatei gespeichert. VRML Browser lesen und interpretieren nun diese Textdatei und stellen deren Inhalt in Echtzeit

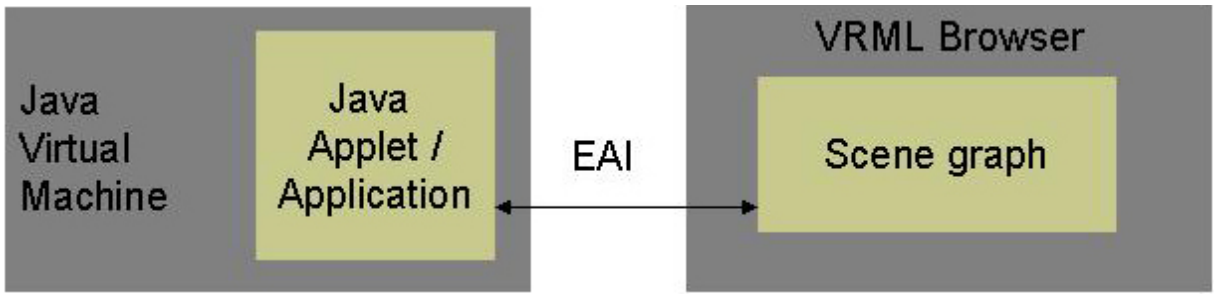


Abbildung 3.1: Die Funktionsweise des EAI

dar.
Einige der populärsten, für Linux erhältlichen Browser sind VRwave, OpenVRML und FreeWRL. Eine kurze Übersicht gibt Tabelle 3.1.

Browser	Betriebssystem	VRML Konformität	EAI Unterstützung	Performanz
VRwave	alle	sehr schlecht	ja	sehr schlecht
OpenVRML	alle	grundlegende Fkt.	nein	schlecht
FreeWRL	Unix/Linux, Mac OS/X	grundlegende Fkt.	ja	ok

Tabelle 3.1: Gegenüberstellung für Linux frei verfügbarer VRML Browser

VRwave

VRwave[18] ist ein VRML Browser, der am Institut für Informationsverarbeitung und computergestützte neue Medien der Technischen Universität Graz entwickelt wurde. Das VRwave Projekt wurde im Jahr 2000 abgeschlossen und bietet keinerlei technischen Support. Der Browser ist im Quellcode erhältlich und wurde weitgehend in Java programmiert. Leider zeigten erste Tests von einfachen VRML Beispielszenen, dass der Browser diese nicht darstellen konnte. Auf eine weitere Untersuchung musste deswegen verzichtet werden, auch wenn der Browser eine EAI Unterstützung geboten hätte.

OpenVRML

OpenVRML[19] ist eine frei erhältliche Laufzeitumgebung für VRML. Für jedes Betriebssystem existiert ein VRML Browser mit dem Namen "Lookat".

- **Betriebssystem:** Alle Betriebssysteme werden unterstützt.
- **Merkmale:** Das Projekt ist in Entwicklung und soll die VRML97 Spezifikation erfüllen.

- **Probleme:** Der Browser ist mit den VRML97 Spezifikationen noch nicht konform. Zahlreiche Fehler müssen noch beseitigt werden. Bei der Evaluation ergaben sich folgende Probleme:
  - Schlechte Performanz
  - Absturz beim Aufrufen von Rotationsbefehlen
  - Keine JavaScript-Unterstützung
  - Keine EAI-Unterstützung
  - Keine Sound-Unterstützung
  - Mangelnde Textunterstützung

FreeWRL

FreeWRL[20] ist ein weiterer Open-Source Browser, der am Communications Research Centre Canada entwickelt wird.

- **Betriebssystem:** Es werden Unix/Linux und Mac OS/X unterstützt.
- **Merkmale:** Das Projekt ist ebenfalls in aktiver Entwicklung und soll die VRML97 Spezifikation erfüllen.
- **Probleme:** Auch bei FreeWRL ergeben sich einige Probleme bei der Darstellung von VRML Szenen, wenngleich der Browser grundsätzlich leistungsfähiger ist als OpenVRML:
  - Begrenzte JavaScript-Unterstützung
  - Keine Kollisionserkennung
  - Falsche Transparenz
  - Begrenzte Textunterstützung

Unter allen evaluierten Browsern ist FreeWRL damit der beste VRML Browser.

3.2 Leistungssteigerung durch GCJ

Ab Version 3.1 verspricht die Compilersuite GNU Compiler Collection (GCC, [13]), welche Front-Ends für C, C++, Objective C, Fortran, Java und Ada enthält, besonders im Java-Bereich enorme Leistungssteigerungen. Verwendet man nämlich für die Kompilierung von Java Code den *GNU Compiler for Java* (GCJ, [14]), welcher erstmals 1999 veröffentlicht wurde, so kann damit nativer Maschinencode erzeugt werden, der schneller ist als interpretierter Java Code. Der so erzeugte Code überlässt zudem mehr Central Processing Unit (CPU) Takte anderen Tasks und verbraucht weniger Speicher als mit einer Java Virtual Machine (JVM) interpretierter Byte Code.[15] Dies verhilft einem PDA mit langsamem

Prozessor zu unverzichtbaren Leistungsgewinnen.
Von GCJ kompilierte Applikationen werden mit der GCJ Laufzeibibliothek "libgcj" gelinkt, welche die Kern-Klassenbibliotheken, einen Garbage Collector und einen Bytecode Interpreter bereitstellt. Derzeit enthält libgcj leider noch nicht alle Java-Klassen. So fehlt Swing vollständig, AWT und Java2D sind in Arbeit und zum Teil schon verfügbar.
Es ist außerdem möglich, GCJ als Cross-Compiler zu benutzen. Des Weiteren kann man mit GCJ Java Code innerhalb von C++ Code verwenden, indem man sich auf das sogenannte *Cygnus Native Interface* (CNI, [16]) stützt. Der Schlüsselgedanke von CNI ist, dass Java Objekte als C++ Objekte und alle Java Klassen als C++ Klassen gesehen werden.

Ein CNI Beispiel

Im Allgemeinen wird bei der Verwendung von Java Code innerhalb von C++ immer nach dem gleichen Schema vorgegangen. Zuerst muss CNI folgendermaßen eingebunden werden:
`#include <gcj/cni.h>`
An der Stelle im Quellcode, an der man nun Java verwenden will, wird eine Java Virtual Machine erzeugt:
`JvCreateJavaVM (NULL);`
Meist im direkten Anschluss daran folgt jetzt das Hinzufügen des aktuellen Threads zur Virtuellen Maschine:
`JvAttachCurrentThread(NULL, NULL);`
Als Implementierungsbeispiel seien an dieser Stelle Strings genannt. Um diese sowohl als Java-Strings als auch als C++-Strings verwenden zu können, müssen für einen String jeweils zwei Variablen definiert werden, zum Beispiel
`std::string nameBeispielString;` und
`java::lang::String* j_nameBeispielString;`.
Instanziiert werden diese nun mit

```
nameBeispielString = "meinBeispielString";
j_nameBeispielString = JvNewStringLatin1(nameBeispielString);
```

3.3 AR-PDA

Wie man schon vom Namen des Projektes "The Augmented Reality Personal Digital Assistant" (AR-PDA, [17]) schließen kann, nimmt es starken Bezug auf die Thematik, welche auch in diesem Systementwicklungsprojekt behandelt wird. Das AR-PDA Projekt wird von einem Konsortium sechser Partner aus Industrie und Wissenschaft von 2001 bis 2004 entwickelt und vom Bundesministerium für Bildung und Forschung (BMBF) gefördert. Der erste Prototyp wurde im März 2002 fertiggestellt.
Ziel des Projektes ist es, ein mobiles Endgerät zu entwickeln, das mittels AR-Technologie Kunden während ihrer alltäglichen Arbeiten wie Einkaufen, Benutzung von Haushaltsgeräten und anderen technischen Geräten oder bei Besichtigungsfahrten unterstützt. Die Forschungsschwerpunkte des Projektes sind das mobile Endgerät selbst, das Design einer

adäquaten Client-Server Architektur, Bildanalyse und 3D-Graphik sowie AR-Autorensystem und Usability.
Die rechenaufwendige Vor- und Nachbearbeitung wie zum Beispiel Rendering von Videobilddaten wird von AR-Servern übernommen und von dort an den mobilen Client geschickt. Dieser Ansatz ist somit gegensätzlich zu dem des Systementwicklungsprojektes, da bei letzterem alle graphischen Berechnungen auf dem mobilen Endgerät durchgeführt werden sollen.

Kapitel 4

System Design

Beim System Design werden die Systemanforderungen in ein Modell überführt, durch welches eine Zerlegung des SHEEP Systems in seine Subsysteme beschrieben wird.

4.1 Design Ziele

Die Design Ziele des Systems geben die Qualitäten wieder, welche die Benutzerschnittstelle inklusive ihrer dreidimensionalen Darstellungskomponente und deren Entwicklung haben muss.

- **Geschwindigkeit:** Da der iPAQ mit einem relativ langsamen Prozessor ohne Floating Point Unit (FPU) und einer Anzeige ohne 3D Beschleuniger ausgestattet ist, muss die Anzeigekomponente performant sein.
- **Speicherökonomie:** Die Speicherknappheit des iPAQs bedingt sorgsames Umgehen mit dessen Speicherressourcen.
- **Erweiterbarkeit und Modifizierbarkeit:** Das System wird auch nach der Demonstration weiterentwickelt werden und muss somit leicht erweitert beziehungsweise modifiziert werden können.
- **Integration des EAI:** Die Anbindung an den VRML Browser erfolgt über das EAI, weshalb Java Methoden und Klassen sinnvoll und performant in das SHEEP System eingebunden werden müssen.

4.2 Identifizierung von SHEEP Subsystemen

Das SHEEP System besteht größtenteils aus DWARF Services, mit welchen gewisse Funktionalitäten einhergehen. Das Konzept des Systems soll Abbildung 4.1 verdeutlichen. Im oberen Teil der Abbildung sind konzeptionelle Einheiten zu sehen, an den Blättern des

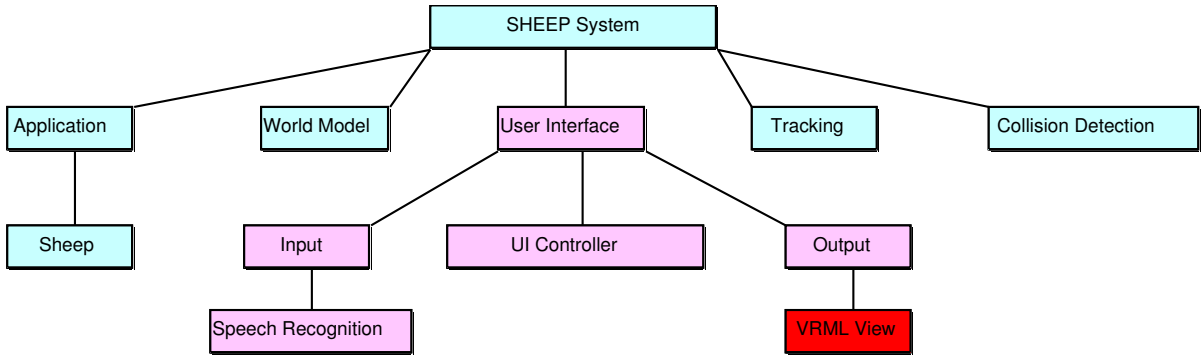


Abbildung 4.1: Zerlegung des SHEEP Systems

Baums DWARF Services. Das SHEEP System hat somit die Subsysteme Collision Detection, World Model, Application, Tracking und User Interface (UI). Für die drei letzteren, welche zu den wichtigsten Subsystemen des SHEEP Systems zählen, folgt nun eine detailliertere Beschreibung:

- **Application:** Die *Schafe* senden ununterbrochen ihre Positionsdaten in der Welt.
- **Tracking:** Die Trackingmodule liefern ununterbrochen die Positionsdaten der getrackten Objekte: Zauberstab, See-through Laptops, iPAQ, Hände des Gottes.
- **User Interface:** Das UI Subsystem kann seinserseits in ein Subsystem unterteilt werden, welches sich aus Input, Output und UI Controller zusammensetzt.
  - **Input:** Das Inputsubsystem erhält die Eingaben vom Spracherkennungsservice.
  - **Output:** Das Outputsubsystem enthält eine *VRML View*, welche die 3D Szenen dem Zuschauer präsentiert. Diese ist im Systementwicklungsprojekt zu implementieren.
  - **User Interface Controller:** Der Benutzerschnittstellencontroller koordiniert Input und Output.

Abbildung 4.2 zeigt nun die mit der VRML View in Korrelation stehenden Komponenten und das zugrundeliegende Model-View-Controller Designpattern.

4.3 Dynamische Modelle

Das interne Verhalten des Systems soll nun durch dynamische Modelle dargelegt werden, welche aufzeigen, wie die VRML View mit den Services anderer Komponenten korreliert. Die VRML View wird nämlich entweder vom UI Controller und Tracker beeinflusst oder vom VRML Browser. Eine Richtung wird in Abbildung 4.3 beschrieben, die andere Richtung in Abbildung 4.4.

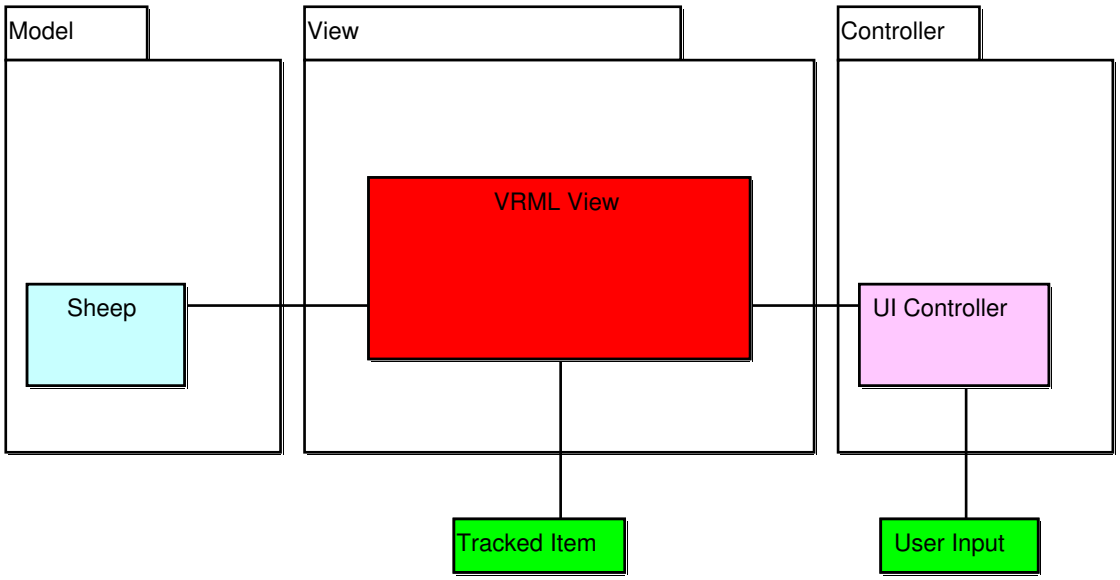


Abbildung 4.2: Mit der VRML View korrelierende Komponenten

Abbildung 4.3 zeigt zwei Benutzer oder Spieler, die mit dem System interagieren. Der erste Benutzer gibt einen Sprachbefehl aus, welcher vom User Input Service in einen entsprechenden Befehl an den UI Controller umgesetzt wird. Der UI Controller sendet daraufhin DWARF Methodenaufrufe an die VRML View. Methodenaufrufe können zum Beispiel Befehle zum Hinzufügen oder Entfernen von Objekten beinhalten. Der VRML View ist nun bekannt, wie mit diesen umzugehen ist, und sendet entsprechende Befehle in Form von EAI Aufrufen an den VRML Browser, welcher wiederum Änderungen an der dargestellten Szene durchführt.
Der zweite Benutzer hingegen wird getrackt und sendet somit ununterbrochen Positions- und Orientierungsänderungen als DWARF Events an die VRML View. Diese interpretiert die Events und sendet entsprechende EAI Aufrufe an den VRML Browser.

Abbildung 4.4 zeigt einen Benutzer, welcher Manipulationen an der im VRML Browser dargestellten Szene vornimmt. VRML View 1 erkennt diese, da sie über das EAI an den Browser angebunden ist. Sie sendet nun die Änderungen je nach Wichtigkeit als DWARF Events oder Methoden an den UI Controller. Vom UI Controller müssen die Manipulationen an alle anderen VRML Views propagiert werden.

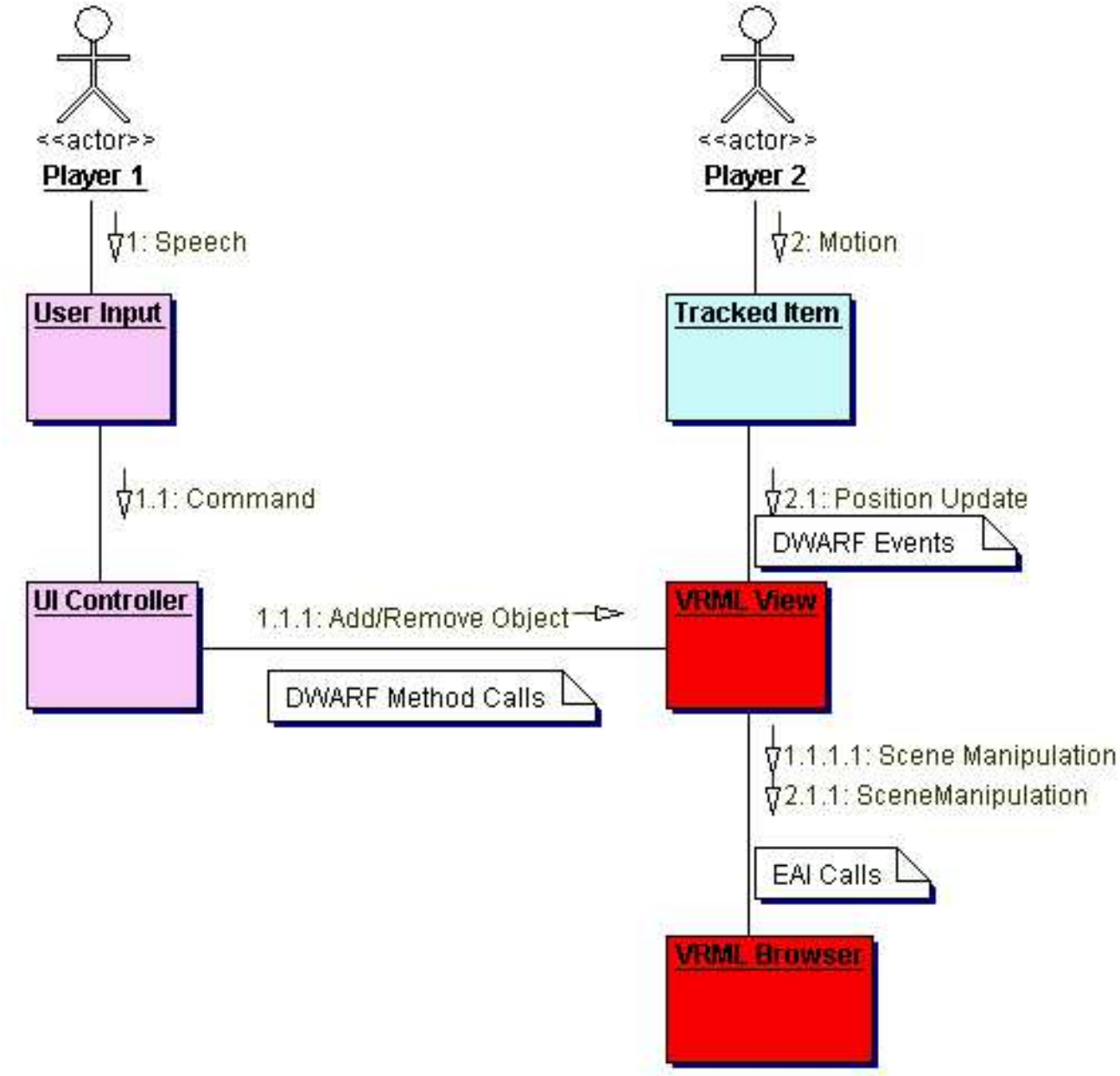


Abbildung 4.3: Dynamisches Modell der auf die VRML View einwirkenden Komponenten

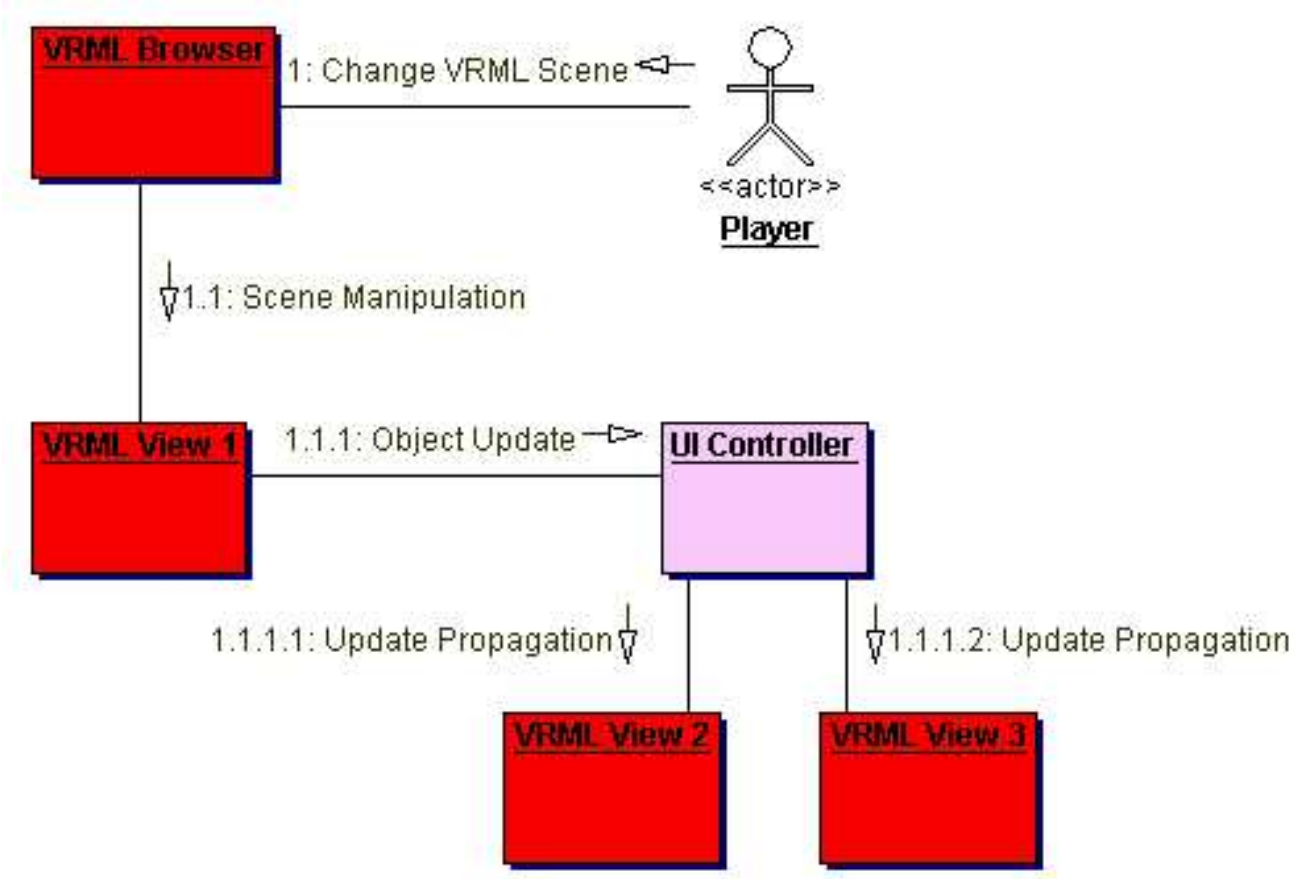


Abbildung 4.4: Dynamisches Modell der auf den UI Controller einwirkenden VRML View

Kapitel 5

Implementation

Das folgende Kapitel beschreibt nun im Detail, wie die VRML View und ihre Komponenten implementiert sind.

5.1 Überblick

Die VRML View besteht aus folgenden Komponenten: VRML Adapter, Eventsender und Manipulator. Abbildung 5.1 zeigt die Klassen und 5.2 wie sie mit anderen Komponenten zusammenhängen. Der VRML Adapter ist eng gekoppelt an Manipulator sowie Eventsender. Jeder Adapter besitzt einen Manipulator und einen Eventsender.
Jegliche Kommunikation, die über den DWARF Bus der DWARF Middleware läuft, kommt über DWARF Events beziehungsweise DWARF Methodenaufrufe zustande, die vom sogenannten Service Manager gehandhabt werden. Die Events können vom VRML Adapter abgegriffen beziehungsweise geschickt werden. Außerdem können bestimmte Methoden des VRML Adapters von der DWARF Middleware aufgerufen werden. Die Verarbeitung der Ereignisse und Methodenaufrufe übernimmt in der Regel der Manipulator, die Generierung neuer Ereignisse der Eventsender.
Die Funktionalität des VRML Adapter Subsystems ist nun folgende:

- Der *VRML Adapter* kann die Demonstrationswelt und deren Tiere sowie Landschaft vom DWARF Bus laden und auf den Bus schicken. Zudem gewährleistet er eine Synchronisation mit anderen DWARF Services, was heißt, dass die Veränderungen in der Welt beständig aktualisiert werden. Außerdem ist er dafür zuständig, der Welt neue Objekte (also Tiere oder andere Landschaftsbestandteile) hinzuzufügen oder diese wieder zu entfernen; der UI Controller ruft dazu Methoden des VRML Adapters auf.

- Der *Manipulator* existiert in zwei Versionen. Eine Version, genannt AddRemove_Manipulator, kann der VRML Szene Objekte hinzufügen und Objekte entfernen, die andere Version, genannt SixDOF_Manipulator, Positions- und Orientierungsänderungen in der Szene durchführen.

- Der *Eventsender* kann DWARF Events erzeugen und diese an den VRML Adapter weitergeben. Events werden immer dann erzeugt, wenn an einem Objekt der Welt Veränderungen vorgenommen werden, also zum Beispiel, wenn ein Schaf eine neue Farbe erhält.

Eine genauere Beschreibung folgt im Anschluss an diesen Abschnitt.

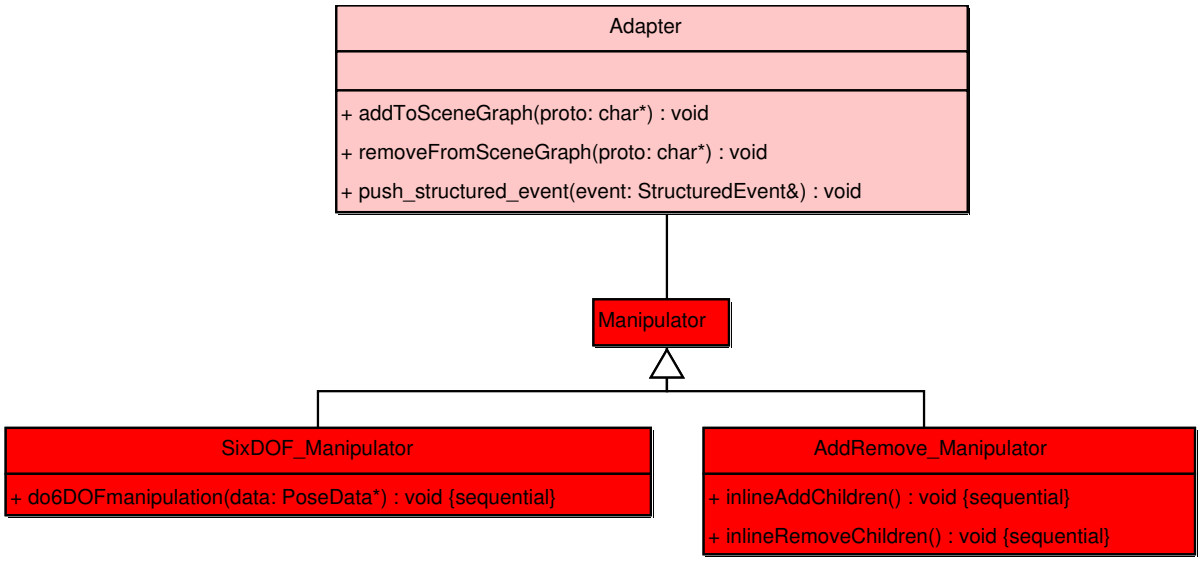


Abbildung 5.1: Die zur VRML View gehörenden Klassen

Eine Evaluation der VRML Browser führte dazu, für die Darstellung der 3D Graphiken FreeWRL zu verwenden. Für die Kommunikation mit dem VRML Browser wird dessen EAI über eine Socket Verbindung angesprochen. Das EAI von FreeWRL ist in Java geschrieben. Um die EAI Klassen nun sinnvoll und leistungsfähig an den Manipulator anbinden zu können, wird bei der Kompilierung jeglicher Klassen GCJ verwendet. Bei Klassen, welche sowohl Java als auch C++ Code beinhalten, also bei beiden Manipulatoren, wird für die Verwendung von Javavariablen und -objektinstanzen darauf geachtet, diese deutlich zu kennzeichnen. Dies geschieht dadurch, den Variablen und instanziierten Objekten ein ”j_” voranzustellen.

5.2 Klassendesign

Die implementierten Klassen VRML Adapter, SixDOF_Manipulator und AddRemove_Manipulator werden nun detaillierter beschrieben.

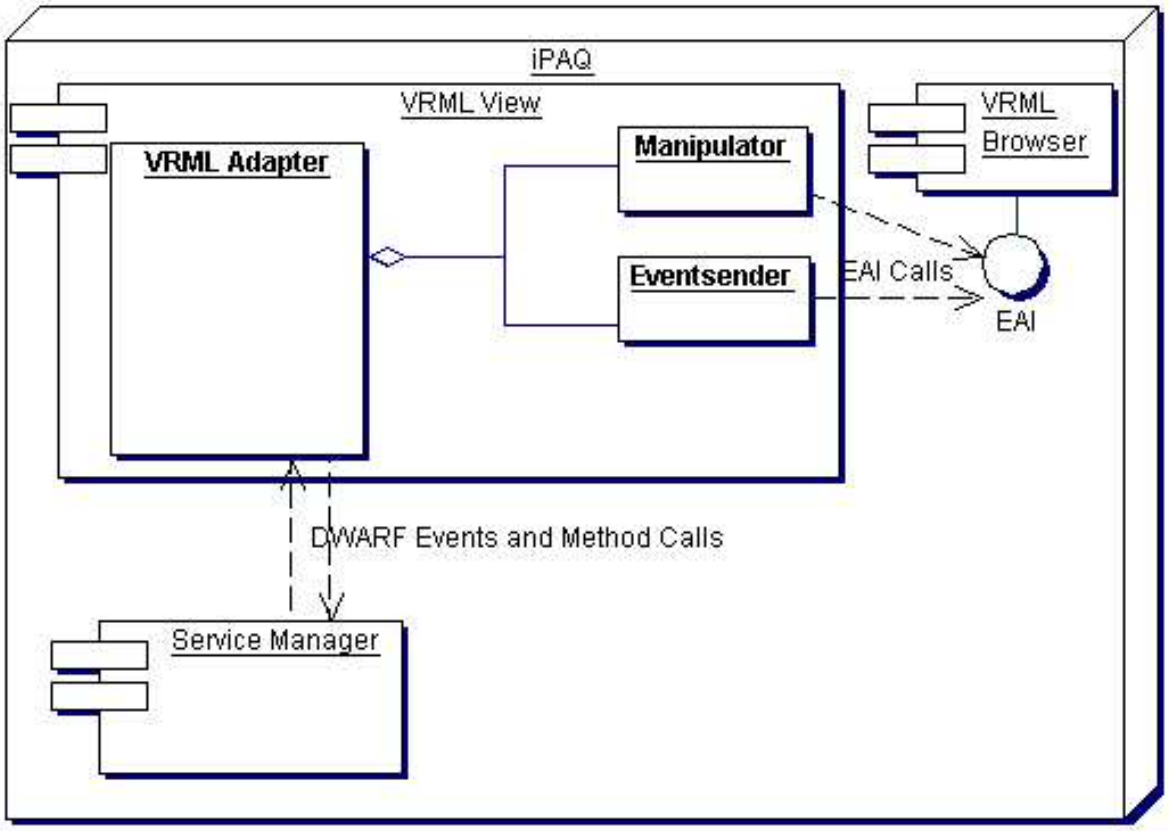


Abbildung 5.2: Das Zusammenspiel der VRML View mit Browser und DWARF

5.2.1 Die VRML Adapterklasse

Der VRML Adapter implementiert einige in Interface Definition Language (IDL) definierte Schnittstellen, deren Hierarchie Abbildung 5.3 verdeutlicht. Er implementiert dadurch die für einen DWARF Service typischen Methoden `newSession()`, `endSession()`, `startService()`, `stopService()`, `getStatus()` und für einen Push Consumer typischen Methoden `offer_change()`, `push_structured_event()` und `disconnect_structured_push_consumer()`. Zudem implementiert er die Methoden `addToSceneGraph()` und `removeFromSceneGraph()` der SceneGraphManipulator Schnittstelle. Für den VRML Adapter Service wichtig ist dabei die Methode `push_structured_event()`, die immer dann aufgerufen wird, wenn ein zum VRML Adapter Service passendes DWARF Ereignis auf dem Eventbus anliegt. Ein solches Ereignis wäre das Senden/Empfangen von Positionsdaten eines Objekts. Besitzt der VRML Adapter nun einen SixDOF_Manipulator, so werden die Positionsdaten an die Methode `do6DOFmanipulation()` des Manipulator weitergegeben und dort verarbeitet.
Besitzt der VRML Adapter aber einen AddRemove_Manipulator, so sind die wichtig-

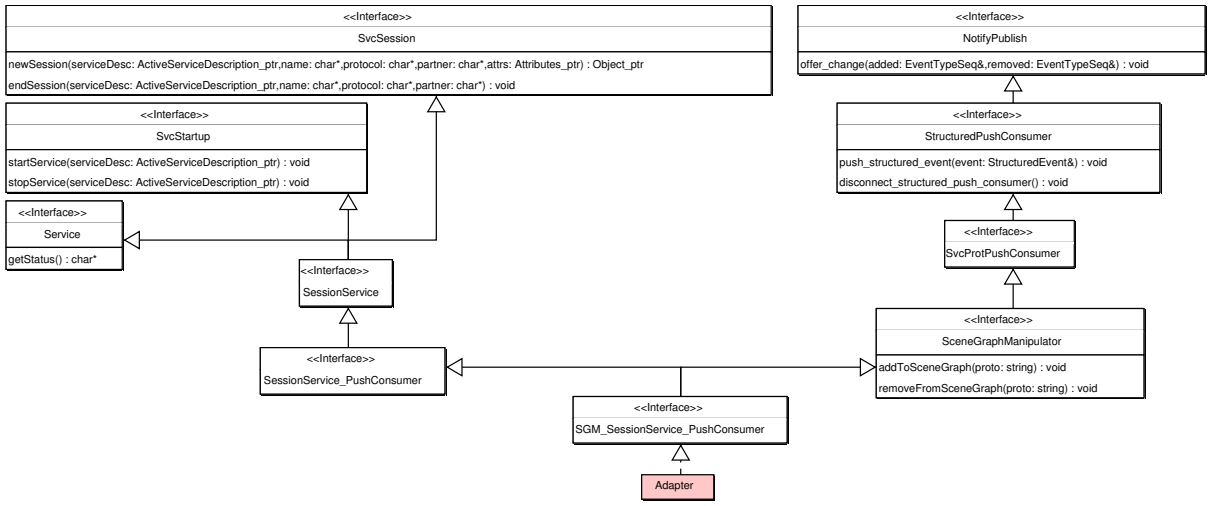


Abbildung 5.3: Die Schnittstellenimplementierung des VRML Adapters

sten Methoden `addToSceneGraph()` und `removeFromSceneGraph()`. Diese können vom UI Controller aufgerufen werden, um der aktuell geladenen VRML Szene ein Objekt hinzuzufügen oder ein Objekt aus der Szene zu entfernen. `addToSceneGraph()` ruft hierfür `inlineAddChildren()`, `removeFromSceneGraph()` dagegen `inlineRemoveChildren()` des Manipulators auf.

Dem VRML Adapter wird schon beim Aufruf übergeben, mit welcher Art von Manipulator er erzeugt werden soll. Erhält er als Argument `"type=sixdof"`, so wird ein SixDOF_Manipulator instanziiert, ansonsten ein AddRemove_Manipulator. Soll es sich um letzteren, also einen AddRemove_Manipulator handeln, so sollte dem VRML Adapter beim Aufruf noch
`"vrml=<name_der_vrml_datei>"` sowie
`"sound=<name_der_sound_datei>"` mit übergeben werden, um eine VRML Beschreibung des Objekts, das hinzuzufügen beziehungsweise zu entfernen ist, und eine Musikdatei anzugeben. Ein Beispielaufruf wäre somit:

```
./Adapter -Dvrml=./SheepWireIpaq.wrl -Dsound=./IpaqSheepAdded.wav
```

5.2.2 Der SixDOF_Manipulator

Der SixDOF_Manipulator enthält als wichtigste öffentliche Methode
`do6DOFmanipulation()`, welche vom VRML Adapter aufgerufen wird, um einem in der Demonstrationswelt existierenden Objekt neue Positions- und Orientierungsdaten zuzuweisen. Alle möglichen Objekttypen der VRML Szene der Welt sind fest verankert. Es gibt augenblicklich folgende Typen:

- Sheep

- Wolf
- TreeFir
- TreeMaple
- Bush
- Path

Will man nun einen neuen Objekttyp hinzufügen, müssen auch im Quellcode an genau kommentierter Stelle entsprechende Anpassungen durchgeführt werden.
In der vom VRML Browser geladenen Szene existiert immer eine gewisse Anzahl von Objekten eines Types, also zum Beispiel fünf Schafe, welche je eine eindeutige Identifikationsnummer (ID) haben. Diese Anzahl darf nicht überschritten werden oder muss in der VRML Datei angepasst werden.
Um nun die Positions- und Orientierungsdaten eines Objekts zu verändern, muss der Methode `do6DOFmanipulation()` die ID, Typ, Orientation und Position übermittelt werden, was über den Pointer `PoseData*` geschieht. Objekte werden aus der VRML Szene entfernt, wenn als Positions- und Orientierungsdaten nur noch 0 geliefert wird. Objekte werden der Szene hinzugefügt, sobald ein Objekttyp mit neuer ID, welche auch in der VRML Szene existiert, am DWARF Eventbus anliegt.

5.2.3 Der AddRemove_Manipulator

Der AddRemove_Manipulator wird zusammen mit einer VRML Szene geladen, welche einen leeren ROOT Knoten enthält, also de facto nichts. Diesem ROOT Knoten kann nun über die beiden wichtigsten öffentlichen Methoden des AddRemove_Manipulators, nämlich `inlineAddChildren()` sowie `inlineRemoveChildren()`, ein Kind hinzugefügt oder entfernt werden. Das Kind wird in der dem VRML Adapter übergebenen VRML Datei spezifiziert. Beim Hinzufügen wird außerdem eine Musik über den Soundplayer `bplay` abgespielt.

Kapitel 6

Schluss - Eine hinreichende Darstellungskomponente

Im Großen und Ganzen kann behauptet werden, dass der iPAQ für eine Darstellung von 3D Szenen hinreichend ist. Um aber größere Szenen darstellen zu können, fehlt diesem leider die nötige Rechenleistung. Die nächste Generation von Taschencomputern bringt aber vielleicht schon Abhilfe für dieses Problem, weshalb man gespannt sein darf, was zukünftige Entwicklungen auf diesem Forschungsgebiet bringen werden.
Nachstehend werden nun die Probleme und das Ergebnis dieses Systementwicklungsprojektes aufgezeigt, welche zur Definition neuer, zukünftiger Arbeiten beitragen sollen.

6.1 Ergebnis

Das Ergebnis dieser Arbeit ist das Entstehen eines auf VRML und DWARF basierten 3D Anzeigemoduls. Folgende Anforderungen werden von diesem erfüllt:

- Das intuitive User Interface des iPAQs, welcher über einen Touchscreen verfügt, ermöglicht dem Anwender eine einfache Interaktion.
- Die Anzeige ist in 3D und kann sich mit anderen DWARF Services insoweit synchronisieren, dass Szenenänderungen auf dem iPAQ aktualisiert werden.
- Die entstandene Software läuft auf dem iPAQ unter Linux und ist an DWARF angebunden.
- Dank der Verwendung von GCJ erhält man eine schnelle, speicherökonomische Anwendung.

6.2 Probleme

Sehr problematisch, zeitintensiv und nervenbelastend war die Einrichtung der Entwicklungsumgebung für den iPAQ. Das größte Problem daran ist, dass der Linux-Betriebssystemkernel für ARM Prozessoren noch nicht sehr ausgereift ist und zahlreiche Bugs enthält. Die Suche nach Ursachen und Abhilfe kostete oft tagelange Arbeit.

Auch der Entwicklungsstatus des verwendeten VRML Browsers FreeWRL ließ zahlreiche Probleme bei der Entwicklung aufkommen. Mit Hilfe eines der Hauptentwickler von FreeWRL, John Stewart, dem an dieser Stelle ein großer Dank ausgesprochen wird, konnten die meisten dieser allerdings gelöst werden.

Zudem traten immer wieder Probleme mit der DWARF Middleware auf. Im Glauben daran, dass DWARF funktioniere, suchte man deswegen oft fälschlicherweise bei sich den Fehler. Damit war zwar nicht einem selbst geholfen, aber glücklicherweise den DWARF Entwicklern.

6.3 Zukünftige Arbeiten

Wie in Kapitel 5 beschrieben wurde, existiert der Manipulator zur Zeit in zwei Versionen. Diese beiden Versionen sollten nach dem Refactoring Pattern zu einer Elterklasse vereinigt werden, von der dann die Hauptfunktionalität an Kinder mit speziellen Funktionalitäten vererbt wird. Dies ist vor allem dann sinnvoll, sobald die Funktionalität des Manipulators umfangreicher wird.

Außerdem sollte der VRML Adapter über einen Extensible Markup Language (XML) Configurator ausgestattet sein, welcher die Fähigkeit hat, eine XML Konfigurationsdatei zu analysieren. Damit werden bestimmte Eigenschaften festgelegt, zum Beispiel welche Manipulatorfunktionalität benötigt wird oder welche 3D Szene geladen werden soll.

Zudem muss der Eventsender implementiert werden, so dass auch die VRML View im Stande ist, Veränderungen an der VRML Szene an andere DWARF Services zu propagieren. Augenblicklich ist nur die umgekehrte Richtung möglich.

Vorstellbar wäre auch die Entwicklung eines eigens für DWARF und ARM Linux geschriebenen Browsers oder Viewers für 3D, da alle evaluierten VRML Browser Schwachstellen zeigten.

Anhang A

Einrichten der Entwicklungsumgebung für den iPAQ

Der mit einem StrongARM Prozessor ausgestattete HP iPAQ wird standardmäßig mit dem Betriebssystem Windows CE beziehungsweise Pocket PC ausgeliefert. Als Entwicklungsumgebung wird aber Linux und OpenGL vorausgesetzt. Die zur Einrichtung der Entwicklungsumgebung notwendigen Schritte werden nachstehend beschrieben.

A.1 Installation von Familiar Linux

Das meistverbreitetste Linux-Betriebssystem für Handhelds ist Familiar Linux. Es basiert gänzlich auf dem sogenannten Tiny-X Server, welcher eine kompakte, speichersparende Version des Standard X Servers ist und die neuesten `RENDER` Erweiterungen enthält. Es hat ein mit Debian Linux vergleichbares Paketverwaltungssystem, welches die Überführung von Debian Paketen in Familiar Pakete nahezu kompatibel macht.
Um Familiar Linux auf dem iPAQ zu installieren, müssen nacheinander der Bootloader (bootldr) installiert, das Root Dateisystem eingerichtet und der iPAQ internetfähig gemacht werden. Im Anschluss daran können weitere Software-Pakete installiert werden.

A.1.1 Installation des Bootloaders

Bevor man sich der Installation des Bootloaders widmet, sollte man eine Sicherungskopie des auf dem iPAQ aktuell installierten Microsoft-Betriebssystems erstellen, um später in der Lage zu sein, dieses wiederherzustellen. Bei der Installation von Familiar Linux gehen nämlich alle Benutzerdaten auf dem iPAQ verloren, weshalb an dieser Stelle eingehend zu einem Daten-Backup geraten wird.
Daraufhin kann über einen Windows Desktop-PC Verbindung zum iPAQ aufgenommen werden. Darüber kann der Bootloader bootldr eingerichtet werden.
Eine schrittweise Beschreibung des Bootloader-Installationsprozesses ist unter [23] zu finden.

A.1.2 Weitere Installationsschritte

Das weitere Vorgehen zur Installation von Familiar Linux erfordert ein serielles Kabel oder die sogenannte Cradle sowie einen Internetzugang über Ethernet, WLAN PPP oder USB. Zudem wird ein Terminal Emulator benötigt, der fähig ist, xmodem Uploads durchzuführen.
Alle Installationsdetails sind eingehend auf den Internetseiten des Familiar Projektes beschrieben, zu finden unter [24].

A.2 Installation von Intimate Linux

Nach dem erfolgreichen Setup von Familiar Linux kann man sich nun dem nächsten Schritt zuwenden: Der Installation von Intimate Linux. Sinnvoll ist die Installation aber nur, wenn man ein sogenanntes Microdrive oder eine PCMCIA Festplatte zur Verfügung hat, welche man zusammen mit dem sogenannten extension backpack für den iPAQ verwendet. Die für das Systementwicklungsprojekt verwendete Festplatte hat eine Kapazität von 5 GB.
Intimate Linux basiert auf Debian Linux, für welches Tausende von Paketen für ARM Prozessoren erhältlich sind, die damit auch alle für den iPAQ zugänglich und installierbar sind. Auf der Homepage des Intimate Projekts ([25]) sind alle notwendigen Informationen über die Installationsprozedur, das Projekt selbst und anderes zu finden.
Nach erfolgreicher Installation erscheint nach der Standardbootauswahl des Bootloaders eine weiteres Menü, welches nun zwischen dem Start von Familiar Linux oder Intimate Linux wählen lässt. Dabei ist es wichtig, vor dem Start von Intimate Linux die Festplatte aus dem Erweiterungs-Rucksack zu entfernen und neu einzusetzen, damit sie richtig erkannt wird. Dieses Fehlverhalten ist wahrscheinlich auf einen Kernel-Bug zurückzuführen und könnte eventuell mit dem Erscheinen und Update auf eine neue Kernelversion behoben werden.
Das ausgefeilte Paketverwaltungssystem von Debian beziehungsweise Intimate Linux setzt die Angabe der Quellen voraus, von denen man die Pakete bezieht. Zu finden sind diese unter /etc/apt/sources.list, welche man als root-Benutzer am Besten folgendermaßen editiert:

```
deb http://intimate.handhelds.org/debian unstable main
deb ftp://ftp.de.debian.org/debian woody main contrib non-free
```

Will man nun bestimmte Programme ausführen beziehungsweise versuchen, Programmcode zu kompilieren, erhält aber Meldungen über des Fehlen bestimmter Bibliotheken oder Header-Dateien, so kann man sich immer auf den Debian Internetseiten ([26]) darüber informieren, in welchem Paket die fehlenden Dateien zu finden sind. Die Pakete können dann mit den Debian-Tools `apt-get` oder `dselect` installiert werden.

A.3 Einrichten der VRML-/OpenGL-Umgebung

OpenGL ist ein plattformunabhängiger Standard für dreidimensionales Rendering und dreidimensionale Hardware-Beschleunigung. Auf diesen Standard setzt auch der VRML-Browser FreeWRL auf, weswegen man auch die frei erhältiche Mesa 3D-Implementierung der OpenGL-Bibliotheken installieren muss, welche auch schon als Sammlung von Debian Paketen erhältlich ist. Wichtig dabei ist, dass man nicht die XFree86-Version der Pakete installiert, da diese nicht mit dem Tiny-X Server zusammen funktionieren, sondern die Mesa-Standardpakete.
Folgende Pakete sind für die (beschleunigte) Grafikdarstellung zuständig:

- `xserver-tiny-h3600`
- `mesag3`
- `mesag-dev`

A.4 Installation von Java

Die Installation einer Laufzeitumgebung von Java gestaltet sich folgendermaßen. Zuerst entpackt man den von Blackdown erstellten Tarball j2re-1.3.1-RC1-linux-arm.tar.bz2 in ein Verzeichnis seiner Wahl und setzt den PATH und CLASSPATH entsprechend (zum Beispiel in /etc/profile). Danach muss noch der Tarball additional-ipaq-stuff.tar.gz in das Rootverzeichnis / entpackt werden. Die beiden Tarballs sind erhältlich unter [27].

A.5 SSH auf dem iPAQ

Sind alle notwendigen Installationsschritte erfolgt, empfiehlt es sich für alle Arbeiten auf dem iPAQ, diesen per SSH-Verbindung anzusprechen. Dazu muss auf dem iPAQ ein SSH-Server laufen, welcher aber automatisch gestartet wird, sobald man das ssh-Debianpaket installiert.

A.6 Einrichten der Compiler

Da das Kompilieren des Quellcodes auf dem iPAQ zu langsam wäre, ist es anzuraten, dies auf zwei verschiedene alternativen Arten zu tun. Zum Testen auf Desktop-Rechnern sollte nativ kompiliert werden, zum Testen auf dem iPAQ cross kompiliert werden. Letzteres heißt zum Beispiel, auf einem Desktop PC mit Intel Prozessor für den iPAQ mit ARM Prozessor zu kompilieren und das Ergebnis auf den iPAQ zu kopieren und dort auszuführen.

A.6.1 Einrichten einer GCC native compiler toolchain

Eine native Compilerumgebung lässt sich im Lehrstuhllabor, in dem Suse Linux installiert ist, am besten über die Installation von RPM Package Manager (RPM) Paketen bewerkstelligen, die im Internet erhältlich sind.

A.6.2 Einrichten einer GCC 3.1.1 cross compiler toolchain

Nachstehend findet sich eine stichpunktartige Auflistung der Schritte, welche durchzuführen sind, um eine Cross compiler toolchain zu erstellen, die auf einer Intel-basierten Plattform einen für ARM-basierte Plattformen ausführbaren Code erstellt.

Download der Quelldateien

Zuerst sind die neuesten Versionen folgender Quell-Balls herunterzuladen, welche zum Beispiel zu beziehen sind von [28].

- binutils
- gcc
- gdb
- glibc
- glibc-linuxthreads

Zudem sind die Kernel-Quellen herunterzuladen, welche erhältlich sind unter [29].

Nun muss man alle Quellen entpacken sowie die Build-Verzeichnisse und die Installationsverzeichnisse erstellen. Das Installationsverzeichniss wird im Folgenden <your prefix> genannt.
Dabei ist wichtig, dass das Quell- und Build-Verzeichnis unterschiedlich sein müssen.

Bin Utilities

Nun sind im Build-Verzeichnis für binutils folgende Befehle auszuführen, aber nicht in einem Unterverzeichnis des Quellbaums (source tree):

<path to your binutils-sources>/configure –prefix=<your prefix> –target=arm-linux
make
make install

Kernel Quellen

Man führt nun folgende Befehle aus:
cd <your prefix>
mkdir include
cd include
make menuconfig

Im Menü von menuconfig navigiert man jetzt zum Abschnitt über System- und Prozessortyp und wählt ein System aus, welches konsistent ist mit dem, das man erstellen will. Nach der Speicherung der Änderungen beendet man menuconfig.

Daraufhin führt man folgende Befehle aus:
make dep
cp -dR <path to your kernel-source>/include/asm-arm <your prefix>/include/asm
cp -dR <path to your kernel-source>/include/linux <your prefix>/include/linux

Vorbereitungen und Bugfixes für gcc

Man führt nun aus:
cd <path to your gcc-source>/gcc/config/arm

Mit einem Texteditor modifiziert man die Datei t-linux. Man fügt -Dinhibit_libc -D__gthr_posix_h zu der Zeile hinzu, die folgendermaßen beginnt:

TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC.

Man erhält somit folgende Zeile:

TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer -fPIC -Dinhibit_libc -D__gthr_posix_h

<your prefix>/bin ist nun der PATH-Variablen hinzuzufügen.

Daraufhin führt man aus:
cd <path to your gcc-source>/libjava/java/net
Die Zeile line 59 in natInetAddress.cc ist auszukommentieren:
/*extern "C" int gethostname (char *name, int namelen);*/

Kompilierung von gcc - erster Durchlauf

Im Build-Verzeichnis für gcc führt man nun folgende Befehle aus, wobei man sich nicht in einem Unterverzeichnis des Quellbaums befinden darf:

<path to your gcc-sources>/configure –prefix=<your prefix> –target=arm-linux –enable-languages=c –with-cpu=strongarm110 –disable-threads –disable-shared make make install

glibc

Wichtig ist, dass man für glibc genau Version 2.13 von autoconf benötigt.

Man muss nun glibc-linuxthreads in das glibc Quellverzeichnis entpacken.

<your prefix>/bin muss zur PATH-Variablen hinzugefügt werden.

Man wechselt nun in folgendes Verzeichnis:
cd <path to your glibc-source>/sysdeps/unix/sysv/linux/arm
Die Datei errlist.c ist zu editieren:
Die Zeilen 40 und 41 sind zu vertauschen, was zu folgendem Resultat führen sollte:

weak_alias (__old_sys_nerr, _old_sys_nerr)
strong_alias (__old_sys_nerr, _old_sys_nerr);

Man wechselt nun in folgendes Verzeichnis:
cd <path to your glibc-source>/sysdeps/unix/sysv/linux
Jetzt ist die Datei configure.in zu editieren:
Zum case "$machine" in Zeile 41 ist folgender switch hinzuzufügen:

```
arm*)
    arch\_minimum\_kernel=2.2.12
    libc\_cv\_gcc\_unwind\_find\_fde=yes
    ;;
```

Im Build-Verzeichnis für glibc führt man nun folgende Befehle aus, wobei man sich wiederum nicht in einem Unterverzeichnis des Quellbaumes befinden darf:

<path to your glibc-source>/configure arm-linux –prefix=<your prefix>/arm-linux –enable-add-ons=linuxthreads –build=i686-pc-linux-gnu
make
make install

Kompilierung von gcc - zweiter Durchlauf

Im Build-Verzeichnis für gcc führt man folgende Befehle aus, wobei man sich ein weiteres Mal nicht in einem Unterverzeichnis des Quellbaumes befinden darf:

<path to your gcc-sources>/configure –prefix=<your prefix> –target=arm-linux –enable-languages=c++,f77,java –with-cpu=strongarm110 –enable-threads –enable-shared

make
make install

gdb

Im Build-Verzeichnis für gdb führt man folgende Befehle aus, wobei man sich ein letztes Mal nicht in einem Unterverzeichnis des Quellbaumes befinden darf:

<path to your gdb-sources>/configure –prefix=<your prefix> –target=arm-linux
make
make install

A.7 Installation von FreeWRL

Der VRML Browser FreeWRL ist zu beziehen von [20]. Um ihn in der Version 0.33 zu installieren, kompiliert man diesen am besten nativ auf dem iPAQ, was in annehmbarer Zeit durchzuführen ist. Installationshinweise finden sich in der Datei INSTALL.html. Folgende Konfiguration ist für die Datei vrml.conf zu verwenden:

```
%VRML_CONFIG = (

# OpenGL (or Mesa) libraries, defines and includes (e.g. -I/usr/OpenGL...).
    OPENGL_LIBS => '-L/usr/lib -L/usr/X11R6/lib -lGL -lGLU -lXext -lX11',
    OPENGL_INC => '-I/usr/X11R6/include/',

# JPEG libraries & includes
    JPEG_LIBS => '-ljpeg',
    JPEG_INC => '',
    JPEG_DEFINE => '',

# PNG libraries & includes
    PNG_LIBS => '-lpng -lz',
    PNG_INC => '',
    PNG_DEFINE => '',

# Freetype things
    FREETYPE_LIBS => '-lfreetype',
    FREETYPE_INC => '-I/usr/include/freetype2/include -I/usr/include \
                     -I/usr/include/freetype2/freetype/config \
                     -I/usr/include/freetype2',
    FREETYPE_DEFINE => '',
```

```
# Javascript lib & include (relative to the JS/ directory)
    JS_LIBS => '-Ljs/Linux_All_DBG.OBJ -ljs',
    JS_INC => '-Ijs/Linux_All_DBG.OBJ -Ijs/',

);
```

Alle anderen Einstellungen in vrml.conf kann man bei den alten Werten belassen.
Um nun noch die Javascript-Libraries ohne Fehler kompilieren zu können, muss man in JS/js/prcpucfg.h noch Einträge für ARM Prozessoren machen:

- Aus `#elif __i386__` wird `#elif __i386__ || __arm__`
- Aus `#ifdef i386` wird `#ifdef i386 || arm`

Literaturverzeichnis

[1] Frederick P. Brooks, Jr.: *What's Real About Virtual Reality?*, IEEE Computer Graphics and Applications, 19, 6, Seite 16-27, 1999

[2] I.E. Sutherland: *The Ultimate Display*, invited lecture, IFIP Congress 65

[3] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, Blair MacIntyre: *Recent Advances in Augmented Reality*, IEEE Computer Graphics and Applications, 21, 6, Seite 34-47, 2001

[4] DWARF Projekt Homepage; http://www.augmentedreality.de/

[5] Christian Sandor, Asa MacWilliams, Martin Wagner, Martin Bauer, Gudrun Klinker: *SHEEP: The Shared Environment Entertainment Pasture*, IEEE and ACM International Symposium on Mixed and Augmented Reality ISMAR 2002, Darmstadt 2002

[6] Bernd Brügge, Allen H Dutoit: *Object-Oriented Software Engineering*, Prentice Hall, New Jersey 2000

[7] Helmut Kopka: *Latex: Eine Einführung*, Addison-Wesley, Bonn, München, Reading, Mass. (u.a.) 1991

[8] Martin Fowler, Kendall Scott: *UML konzentriert*, Addison-Wesley, München, 2000

[9] Ming-Ju Lee: *Authoring of 3D User Interfaces for Applications with DWARF*, Bachelor Thesis, Technische Universität München, München 2002

[10] Rikk Carey, Gavin Bell, Chris Marrin: *ISO/IEC 14772-1:1997, Virtual Reality Modeling Language, (VRML97)*, The VRML Consortium Incorporated 1997; http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[11] Chris Marrin: *Proposal for a VRML 2.0 Informative Annex, External Authoring Interface Reference*, Silicon Graphics, Inc. 1997; http://www.vrml.org/WorkingGroups/vrml-eai/ExternalInterface.html

[12] Andrew Phelps, J. Eric Mason: *External Authoring Interface Working Group*; http://www.web3d.org/WorkingGroups/vrml-eai/

[13] The GCC Homepage; http://gcc.gnu.org/

[14] The GCC team: *The GNU Compiler for the Java Programming Language*; http://gcc.gnu.org/java/

[15] Craig Delger, Mark G. Sobell: *gcj: Compile Java into Native Machine Code*, Technical Paper, Red Hat Inc., 2000; http://www.redhat.com/devnet/articles/gcj.pdf

[16] Cygnus Solutions: *The Cygnus Native Interface for C++/Java Integration*; http://gcc.gnu.org/java/papers/cni/t1.html

[17] J. Fründ, C. Geiger, M. Grafe, B. Kleinjohann: *The Augmented Reality Personal Digital Assistant*, Proceedings of the 2nd International Symposium on Mixed Reality ISMR2001, The Virtual Reality Society of Japan, 2001

[18] VRwave Homepage; http://www.iicm.edu/vrwave/

[19] OpenVRML Homepage; http://www.openvrml.org/

[20] FreeWRL Homepage; http://www.crc.ca/FreeWRL/

[21] handhelds.org Homepage; http://www.handhelds.org/

[22] Debian Homepage; http://www.debian.org/

[23] *iPaq H3100/H3600/H3700/H3800 Handhelds.org Bootloader Installation Instructions*; ftp://ftp.handhelds.org/pub/linux/dists/compaq/ipaq/stable/install.html

[24] Jamey Hicks, Alexander Guy: *Familiar v0.5.3 Installation Instructions*; http://familiar.handhelds.org/familiar/releases/v0.5.3/install/H3600/install.html

[25] The Intimate Project Homepage; http://intimate.handhelds.org/

[26] Debian - Search the contents of packages; http://www.debian.org/distrib/packages#search_contents

[27] Mirror von Blackdown Java Linux für ARM; ftp://ftp.gwdg.de/pub/languages/java/linux/JDK-1.3.1/arm/rc1/

[28] Mirror der GNU Applikationen; ftp://ftp.leo.org/pub/comp/os/unix/gnu

[29] Linux Kernel Quellen für Handhelds; ftp://ftp.handhelds.org/pub/linux/kernel/