Entwurf eines Rahmensystems für mobile Augmented-Reality-Anwendungen

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)

von

Dipl.-Inform. Patrick Dähne

aus Hamburg

Referenten der Arbeit:	Prof. Dr. José L. Encarnação
	Prof. Dr. Dieter Schmalstieg
Tag der Einreichung:	28. November 2007
Tag der mündlichen Prüfung:	30. Januar 2008

Darmstadt 2008

D17

Inhaltsverzeichnis

1 Einleitung und Motivation

In den letzten Jahren hat es auf dem Gebiet der Graphischen Datenverarbeitung einen rasanten Fortschritt gegeben. Virtuelle Realität (VR) erfordert nicht mehr einen Hardwareaufwand, der nur von Großforschungseinrichtungen oder Industrieunternehmen finanziert werden kann. Anstelle dessen verfügt praktisch jeder PC, der „von der Stange“ gekauft wird, über eine Graphikleistung, die bis vor kurzem noch High-End-Graphikworkstations vorbehalten war. Der Einsatz von VR-Technik bei der Entwicklung von neuen Produkten ist in vielen Industriezweigen, wie z.B. der Automobilindustrie, inzwischen Standard. Das breite Publikum lernt VR in Form von Messedemonstratoren, in Vergnügungsparks und auf speziellen Veranstaltungen wie den Cybernarium Days [1] kennen. In Form der beliebten „First-Person-Shooter“-Spiele erobert VR sogar die heimischen Wohnzimmer.

Eine eng mit der Virtuellen Realität verwandte Technologie führt dagegen immer noch ein Schattendasein, nämlich Augmented Reality (AR). Im Gegensatz zur VR taucht der Anwender bei AR nicht in eine komplett synthetische Welt ein, vielmehr verbleibt er in der realen Welt, die jedoch mit virtuellen Objekten erweitert (augmentiert) wird. Dazu wird typischerweise ein Head-Mounted-Display (HMD) verwendet. Das HMD ist entweder durchsichtig, d.h. der Anwender sieht die reale Umgebung, in die über einen halbdurchlässigen Spiegel virtuelle Objekte eingefügt werden („optical see-through“), oder das HMD ist undurchsichtig, und der Anwender sieht das Bild einer am HMD befestigten Kamera, in das virtuelle Objekte eingefügt werden („video see-through“). Damit die virtuellen Objekte sich exakt an der richtigen Stelle im Blickfeld des Anwenders befinden, ist eine genaue Bestimmung von Position und Blickrichtung des Anwenders erforderlich.

1.1 Anwendungen von AR

Der folgende Text gibt eine Übersicht über die Vielzahl von Anwendungen, die über AR realisiert werden können, und nennt einige Projekte, die sich mit dem jeweiligen Anwendungsfeld beschäftigen. Es handelt sich dabei überwiegend um Projekte, an denen das Fraunhofer IGD beteiligt war – es wird also nur ein kleiner Ausschnitt aus der aktuellen Forschungsarbeit gezeigt.

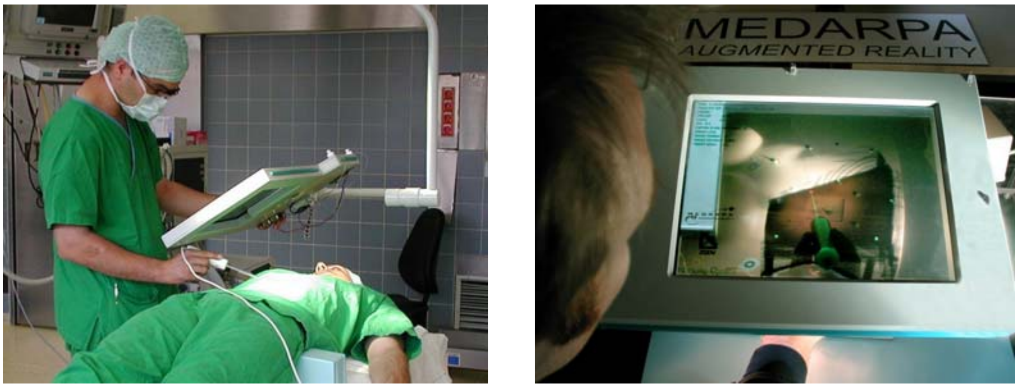


Abbildung 1: Das Medarpa-System erlaubt dem Mediziner einen Blick in das Innere des Körpers

- In der Medizin ermöglicht AR-Technik dem Mediziner einen „Röntgenblick“ in das Innere des menschlichen Körpers. Bislang werden operative Eingriffe im Voraus mit Hilfe von Röntgen-, Ultraschall- oder Tomographiebildern geplant. Die Schnitte werden dann jedoch mehr oder weniger „blind“ durchgeführt, d.h. es hängt vom Wissen und der Erfahrung des Chirurgen ab, wie exakt die geplanten Schnitte in die

Realität umgesetzt werden. In Zukunft wird man 3D-Bilder des Körperinneren, die aus den 2D-Aufnahmen berechnet werden, direkt während der Operation in das Blickfeld des Arztes einblenden können. Dies wird helfen, Fehler zu verhindern, und zu einem insgesamt viel schonenderen Operationsverlauf führen. Beispiele für den Einsatz von AR in der Medizin sind die Projekte Medarpa [2] und ARSyS-Tricorder [3].

Abbildung 2: Das Archeoguide-System zeigt dem Besucher die Rekonstruktionen alter Tempel, wo jetzt nur Ruinen stehen

- Der Tourist der Zukunft wird an historischen Stätten die Möglichkeit bekommen, virtuelle Rekonstruktionen längst zerstörter Bauwerke an Ort und Stelle zu betrachten. Jeder, der schon mal so wichtige archäologische Monumente wie Delphi oder das antike Olympia besucht hat, wird enttäuscht darüber gewesen sein, daß nur noch Trümmer an die ehemalige Bedeutung dieser Orte erinnern. Zwar versucht man, die Attraktivität dieser Orte durch den Wiederaufbau von einzelnen Tempelteilen zu verbessern, doch stößt diese Vorgehensweise auf den erbitterten Widerstand der Archäologen, die den gegenwärtigen Zustand dieser Stätten möglichst unverändert für zukünftige Generationen erhalten möchten. Einen möglichen Ausweg aus diesem Konflikt zwischen Vermarktung und Erforschung bietet der Einsatz von AR. Der Tourist der Zukunft wird wie bisher über die Trümmerfelder spazieren können. Er wird aber jederzeit eine AR-Ausrüstung, die er bei sich trägt, verwenden können, um einen Eindruck davon zu bekommen, wie es an dem jeweiligen Standort vor mehr als 2000 Jahren aussah.

  Doch nicht nur Bauwerke werden mit der neuen Technik virtuell wieder neu aufgebaut, sondern es ist auch möglich, die ehemaligen Bewohner dieser Bauwerke virtuell wieder zum Leben zu erwecken. Virtuelle Sportler veranstalten Wettkämpfe in den alten Stadien, virtuelle Priester führen Kulthandlungen in den alten Tempeln durch, alte Marktplätze füllen sich erneut mit Händlern, die ihre Waren anpreisen, und Käufern, die diese Waren kritisch prüfen. Kurz, die toten Trümmersteine werden erneut mit Leben gefüllt, und es ist ein direkter Einblick in längst vergangene Zeiten möglich.

  Beispiele für den Einsatz von AR im Themenfeld Tourismus sind die Projekte Archeoguide [4], Lifeplus [5] oder aktuell iTacitus.

- Genauso, wie es möglich ist, alte Bauwerke wieder zu rekonstruieren, kann natürlich auch die Wirkung geplanter Neubauten mittels der AR-Technologie an Ort und Stelle geprüft werden. Schon heute ist es Stand der Technik, daß Neubauten komplett per Computer entworfen werden. Die meisten der dazu verwendeten Softwareprodukte erlauben es, nicht nur zweidimensionale Pläne, sondern auch realitätsgetreue dreidimensionale Ansichten der Bauwerke zu erzeugen. Sogar eine virtuelle Besichtigung der 3D-Modelle ist möglich, lange bevor der erste Stein verbaut ist. Von hier aus ist es nur noch ein kurzer Schritt dahin, den Neubau direkt vor Ort in die vorhandene Bebauung einzublenden und zu prüfen, ob das Neue mit dem Alten harmoniert.

Abbildung 3: Einsatz von AR in Architektur und Stadtplanung

Genauso denkbar ist der Einsatz von AR in der Innenarchitektur. Wie wird die neue Couchgarnitur in der vorhandenen Wohnzimmereinrichtung wirken? Wie wird die neue Küche aussehen, und sind wirklich alle Küchengeräte in der gegenwärtigen Planung ergonomisch bedienbar? Antwort auf diese Fragen wird der Innenarchitekt der Zukunft mittels AR-Technologien bekommen.

Abbildung 4: Starmate unterstützt den Techniker bei Wartungsarbeiten

- Große Hoffnungen setzt die Industrie auf den Einsatz von AR in Service und Wartung. Der Wartungstechniker der Zukunft wird keine gedruckten Handbücher mehr benötigen. Defekte Maschinen repariert er, indem er seine AR-Ausrüstung aufsetzt, die ihm die notwendigen Arbeitsschritte und Informationen direkt in sein Blickfeld einblendet. Falls er Löcher in die Wand bohren muß, zeigt ihm sein AR-System, wo in der Wand Stromleitungen verlegt wurden. Wenn er einmal nicht mehr weiter weiß, kann er seinen Kollegen im Büro zur Hilfe rufen, der ihm virtuell über die Schulter blicken und Hinweise zur Behebung von Fehlern geben kann. Beispiele für die Anwendung von AR-Technologien in diesem Bereich sind die Projekte ARVIKA [6] und sein Nachfolgeprojekt ARTESAS, sowie die Projekte Starmate [7], ULTRA [8] und MARIO [9].

- Wie praktisch jede neue Technik kann leider auch AR für militärische Zwecke genutzt werden. So bekommt der Soldat der Zukunft im Projekt BARS [10] z.B. Stadtpläne, Gebäudepläne, Positionen von Scharfschützen oder befreundeten Einheiten sowie Informationen über das allgemeine Schlachtgeschehen in seinen Helm eingeblendet.

- Und zu guter Letzt sind natürlich eine Vielzahl von Anwendungen im Bereich Entertainment und Infotainment denkbar. Spielerisch bekommen Schüler z.B. beim

Projekt Geist [11] Wissen über das alte Heidelberg vermittelt, indem sie mit ihrer AR-Ausrüstung durch die Stadt laufen und mit den mittelalterlichen Bewohnern Heidelbergs sprechen. Oder warum nicht einmal wie beim Projekt ARQuake [12] virtuelle Monster unter ganzem Körpereinsatz in realen Räumen jagen, anstatt auf der heimischen Couch liegend am Fernsehapparat auf die Pirsch zu gehen?

Trotz dieser vielfältigen und vielversprechenden Einsatzmöglichkeiten ist Augmented Reality in der Öffentlichkeit bislang völlig unbekannt. Dabei hat AR viel mehr als Virtual Reality das Potential, das Leben vieler Menschen zu beinflussen und zu verbessern. Praktisch jeder hat schon heute Kontakt zu AR: Es ist die Rede von Kinofilmen wie Jurassic Park oder Star Wars, wo virtuelle Dinosaurier durch reale Filmszenen streifen oder reale Schauspieler vor komplett virtuellen Kulissen spielen. Auch das virtuelle Studio, das inzwischen schon Standard in vielen Fernsehsendungen ist, ist schließlich ein Beispiel für den Einsatz von AR. Doch diese Anwendungen sind rein passiv, der Anwender kann seine Position in der dargestellten Szene nicht verändern, er kann nicht interagieren.

Richtig in das Bewusstsein der Öffentlichkeit wird AR wohl erst dann eindringen, wenn es gelingt, brauchbare Anwendungen zu entwickeln, die auf mobiler Hardware laufen, interaktiv sind, und es den Anwendern erlauben, das Potential von AR selbst im täglichen Leben zu erleben.

1.2 Aktuelle Forschungsthemen im Bereich AR

Fragt man einen der an der Entwicklung der AR-Technologie beteiligten Experten, wann denn wohl mit den ersten praktischen Anwendungen zu rechnen ist, so erhält man meistens einen Zeitraum von 10 Jahren als Antwort, manchmal mehr, manchmal weniger. Fragt man nach den Gründen für diese lange Entwicklungszeit, so werden eine Reihe von ungelösten Problemen genannt:

- Das Problem der genauen Bestimmung von Position und Blickrichtung des Anwenders im Raum – das sogenannte „Tracking“. Offensichtlich sind diese Informationen von entscheidender Bedeutung, wenn man virtuelle Objekte in reale Szenen einblenden will. Umso erstaunlicher ist es, daß bis heute keine Technologie existiert, die diese Informationen mit einer befriedigenden Genauigkeit und Zuverlässigkeit liefern kann. Es existieren eine Vielzahl von Ansätzen (elektromagnetisches Tracking, Ultraschall, Beschleunigungssensoren, videobasiertes Tracking, GPS, Kompaß), doch kein Ansatz löst das Problem zufriedenstellend. Einige Trackingsysteme (z.B. Beschleunigungssensoren) haben eine Drift, d.h. die Genauigkeit nimmt im Laufe der Zeit ab. Oder sie sind empfindlich gegenüber metallischen Gegenständen in ihrer Umgebung (z.B. elektromagnetisches Tracking, Kompaß). Manche sind nur außerhalb von Gebäuden verfügbar (z.B. GPS), andere dagegen sind nur in geschlossenen Räumen zu betreiben, weil ihre Reichweite zu gering ist oder weil zu ihrem Betrieb größere Installationen notwendig sind (z.B. ektromagnetisches Tracking, Ultraschall). In praktisch allen existierenden AR-Systemen wird daher eine Mischung verschiedener Technologien verwendet (hybrides Tracking), typischerweise videobasiertes Tracking in Kombination mit einer für den jeweiligen Anwendungsfall geeigneten anderen Tracking-Technologie (z.B. Beschleunigungssensoren beim MATRIS-Projekt [13]), was die technische Komplexität und die Kosten dieser Systeme in die Höhe treibt.
- Das Problem der korrekten Beleuchtung und des korrekten Schattenwurfs der virtuellen Objekte, die in die reale Szene eingeblendet werden sollen [14]. Idealerweise sollen sich die virtuellen Bestandteile der dargestellten Szene nahtlos in die realen Bestandteile integrieren. Das bedeutet insbesondere, daß sie korrekt

beleuchtet werden, und daß sie auch korrekte Schatten werfen. Offensichtlich ist schon die Bestimmung der in der realen Welt vorhandenen Lichtquellen nicht trivial – woher sollte das AR-System diese Information auch bekommen? Aber auch bei komplett bekannten Beleuchtungsverhältnissen ist es schwierig, eine realistische Beleuchtung und einen realistischen Schattenwurf der virtuellen Objekte zu berechnen. Techniken wie Radiosity oder Raytracing sind für Online-Rendering bis heute einfach zu rechenaufwendig.

- Die Verdeckungsproblematik: Die virtuellen Objekte, die in die reale Szene eingeblendet werden, verdecken nicht nur reale Objekte, sondern sie können umgekehrt auch selbst von realen Objekten verdeckt werden. Doch wie kann man solche Verdeckungen feststellen? Dazu müßte das AR-System alle realen Objekte und ihre Position im Raum kennen, was in der Praxis unrealistisch ist. Zwar gibt es Ansätze, über paarweise angeordnete Kameras ein Stereobild der realen Szene aufzunehmen und aus diesem Bild eine Tiefenkarte zu erzeugen, doch ist der dafür notwendige Hardware- und Rechenaufwand für mobile Systeme einfach zu groß.
- Das unzureichende Hardware-Angebot für mobile AR-Systeme. Die für AR-Systeme benötigte Hardware unterscheidet sich grundlegend von der Hardware, mit der heutzutage üblicherweise mobile Rechner ausgestattet sind. AR-Systeme benötigen so exotische Geräte wie HMDs, GPS-Empfänger, Kompasse, Videokameras etc. Auf der anderen Seite wird Standard-Hardware wie Tastatur, Maus, Laptopdisplay und DVD-Laufwerk normalerweise nicht benötigt. Wünschenswert wären also speziell für AR-Anwendungen designte Geräte, die optimal auf ihren Einsatzzweck ausgerichtet sind. Leider gibt es solche Geräte nicht, weil der Markt für AR-Systeme momentan einfach zu klein ist (die bekannten Geräte von Xybernaut werden hier nicht berücksichtigt, weil sie in ihrer Leistungsfähigkeit dem aktuellen Stand der Technik um Jahre hinterherhinken). Es ergibt sich hier der bekannte Henne-Ei-Effekt – es gibt keine Hardware, weil es keine marktreife AR-Software gibt, und die Entwicklung markreifer AR-Software wird behindert durch den Umstand, daß es keine passende AR-Hardware gibt.

  Momentan behilft man sich bei der Entwicklung von AR-Systemen damit, daß man auf Standard-Hardwarekomponenten zurückgreift [15], die nicht optimal auf ihren Einsatzzweck zugeschnitten ist. Das führt dazu, daß AR-Systeme zu schwer, zu instabil und zu teuer werden. Darüber hinaus ist auch die Laufzeit der meisten Systeme aufgrund des hohen Stromverbrauchs und des stagnierenden Fortschritts bei Batterien und ähnlichen Stromquellen für den praktischen Gebrauch zu kurz.

Angesichts dieser schwerwiegenden Probleme stellt sich die Frage, mit welchen Mitteln man die Entwicklung von AR-Technologien beschleunigen kann. Offensichtlich ist die Prognose von 10 Jahren für praktische AR-Anwendungen eher unbefriedigend.

Aber vielleicht ist die Situation ja gar nicht so hoffnungslos, wie sie auf den ersten Blick aussieht? Vielleicht sind nur die Ansprüche zu hoch, die die Gemeinde der AR-Entwickler an sich selbst stellt. Wer sagt denn, daß virtuelle Objekte völlig nahtlos in virtuelle Szenen eingeblendet werden müssen? Daß sie völlig korrekt beleuchtet sein müssen, und daß sie 100-prozentig an der richtigen Stelle eingeblendet werden müssen?

Es ist offensichtlich, daß die Beleuchtungsproblematik und die Verdeckungsproblematik in absehbarer Zeit auf mobilen AR-Systemen nicht zufriedenstellend gelöst werden können. Bei der Bestimmung von Position und Orientierung gibt es dagegen Lösungen, die aber meist speziell auf die jeweilige Anwendung und den jeweiligen Einsatzzweck (Indoor oder Outdoor) abgestimmt sind. Nichtsdestotrotz ist hier der Einsatz in praktischen Anwendungen in realistische Nähe gerückt, wenn man nicht den Anspruch hat, perfekte Trackingergebnisse unter allen Umständen zu bekommen.

1.3 Fokus und Aufbau dieser Arbeit

So rückt neben den oben beschriebenen Problemen in den letzten Jahren eine andere Fragestellung immer stärker in den Vordergrund: Wie entwickelt man überhaupt AR-Anwendungen? Welche Softwarearchitektur sollten AR-Systeme haben?

Ein großes Problem bei AR-Anwendungen ist gegenwärtig, daß sich noch keine Standards etabliert haben, wie solche Anwendungen überhaupt auszusehen haben. Jedes AR-System, jede AR-Anwendung ist ein Einzelstück, das speziell auf seinen Einsatzzweck zugeschnitten ist. Dieser Zustand behindert offensichtlich die Entwicklung der AR-Technologie. In anderen Bereichen der Graphischen Datenverarbeitung haben sich längst Standards für die Entwicklung von Anwendungen etabliert. So verwenden 2D-Oberflächen für Desktop-Systeme inzwischen alle dieselben Interaktionsmetaphern (Anwendungen laufen in Fenstern, und sie werden über Schaltflächen bedient, die mit der Maus angeklickt werden können). Im Bereich der 3D-Anwendungen hat sich weitgehend das Konzept des Szenengraphen durchgesetzt.

Diese Arbeit will einen neuen Weg beschreiten. Der Fokus liegt nicht in der Lösung eines der bisher ungelösten Teilprobleme (Tracking, Beleuchtung, Verdeckung), in denen sich die Forschung festzufahren droht. Anstelle dessen soll versucht werden, aus dem gegenwärtigen Stand der Forschung heraus die Frage zu klären, wie man kurz- und mittelfristig praktisch einsetzbare AR-Anwendungen entwickeln kann. Die Hoffnung ist dabei, daß der praktische Einsatz (und die Vermarktung) von AR-Produkten die Entwicklung in den bisher problematischen Teilbereichen vorantreibt und beschleunigt. Daher soll in dieser Arbeit eine Plattform für AR-Anwendungen entworfen werden, die als flexible Basis für konkrete AR-Produkte dienen soll.

Wichtige Ziele wurden bei der Entwicklung des in dieser Arbeit vorgestellten AR-Systems waren dabei:

- Eine deutlich vereinfachte und beschleunigte Anwendungsentwicklung. AR-Anwendungen gehören zu den komplexesten Softwaresystemen. Es muß häufig eine Vielzahl von exotischen Hardwarekomponenten, wie z.B. GPS-Receiver, eingebunden werden. Viele AR-Projekte konzentrieren sich stark auf diese technischen Aspekte und vernachlässigen dabei das Design der eigentlichen Anwendung. Das in dieser Arbeit vorgestellte System soll dem Entwickler einen fertigen Satz von robusten und gut getesteten Softwarekomponenten bereitstellen, die alle Bereiche der AR-Anwendung abdecken, wie z.B. die Anbindung typischer AR-Geräte, Trackingalgorithmen, Rendering, Kommunikation mit anderen Komponenten über das Netzwerk, etc. Der Anwendungsentwickler kann sich schnell ein laufendes System zusammenstellen und sich dann auf die Entwicklung der eigentlichen Anwendung konzentrieren.

- Eine stärkere Wiederverwendung von bereits in früheren Projekten entwickelten Komponenten zu ermöglichen. Viele Projekte sind stark auf die Lösung eines speziellen Problems fokussiert. Am Schluß entsteht ein monolithisches System, das zwar für den gewünschten Zweck funktioniert, aber kaum wiederverwendbar ist und nur schwer als Basis für neue Projekte dienen kann. Das in dieser Arbeit vorgestellte System soll diesen Zustand verbessern. Das System stellt einen fertigen Satz von Softwarekomponenten zur Verfügung, die möglichst viele Bereiche der AR-Anwendungsentwicklung abdecken. Falls doch einmal eine Funktionalität fehlt, soll der Anwendungsentwickler die Möglichkeit haben, mit wenig Aufwand das Basissystem um eine neue Komponente zu erweitern, die die neue Funktionalität nicht nur ihm selbst, sondern auch allen anderen Anwendern des Systems zur Verfügung stellt.

- Eine möglichst weitgehende Plattformunabhängigkeit. Wie in Kapitel 2 gezeigt werden wird, laufen AR-Anwendungen auf sehr heterogenen Plattformen – es gibt auf der einen Seite High-End-Systeme mit Head Mounted Displays, die auf leistungsstarken Laptops laufen, und auf der anderen Seite Low-End-Systeme, die mit PDAs oder Mobiltelefonen arbeiten. Um diese Plattformunabhängigkeit zu erreichen, muß man darauf verzichten, die Anwendung auf herkömmliche Weise in plattformspezifischen Programmiersprachen wie C oder C++ zu programmieren. Das in dieser Arbeit vorgestellte System ist daher keine Software-Bibliothek, die AR-spezifische Funktionalität bereitstellt, die man in seinen Anwendungen verwenden kann. Anstelle dessen handelt es sich um eine Laufzeitumgebung, die die Anwendungslogik interpretiert und ausführt. Anwendungen können also auf allen Plattformen ausgeführt werden, auf die die Laufzeitumgebung portiert wurde. Ein besonderes Augenmerk muß dabei natürlich auf die Performanz gelegt werden – mobile AR-Plattformen haben nur beschränkte Ressourcen, und zeitkritische Aufgaben wie Videotracking oder Rendering können natürlich nicht in einer interpretierten Programmiersprache wie Javascript durchgeführt werden. Daher wird in dieser Arbeit ein komponentenbasierter Ansatz verwendet – die Laufzeitumgebung stellt eine Vielzahl von Softwaremodulen zur Verfügung, die klar umrissene Funktionalitäten zur Verfügung stellen. Die eigentliche Anwendung wird in einer Markup-Language geschrieben, die die Zusammenstellung und Verschaltung der einzelnen Module steuert. Nur die weniger zeitkritischen Teile der Anwendungslogik, also z.B. das User-Interface und die Interaktion mit dem Anwender, werden in einer interpretierten Scriptsprache umgesetzt.

- Das System sollte es auch Personen, die wenig Programmiererfahrung besitzen, ermöglichen, AR-Anwendungen zu entwickeln. Momentan sind die meisten AR-Projekte stark auf die Technik fokussiert, d.h. man versucht konkrete Probleme wie z.B. das Tracking zu lösen. Dieser Zustand beruht natürlich darauf, daß die gesamte AR-Technologie noch in den Kinderschuhen steckt. In Zukunft wird es aber immer, eine mehr Anwender- und Anwendungs-orientierte Sicht auf AR einzunehmen. D.h. man wird die Frage beantworten müssen, für welche Einsatzzwecke sich AR tatsächlich eignet, wie man AR ergonomisch und benutzerfreundlich macht, wie man AR in bestehende Arbeitsabläufe integriert, etc. Viele dieser Fragestellungen können jedoch nicht von Informatikern beantwortet werden, anstelle dessen ist es notwendig, auch Experten aus anderen Fachgebieten wie Gestalter, Ergonomen oder Psychologen hinzuzuziehen. Diese Experten werden natürlich häufig damit überfordert sein, AR-Anwendungen z.B. in C++ zu entwickeln. Aber viele von ihnen werden in der Lage sein, sich aus vorgefertigten Teilkomponenten über eine XML-basierte Markup-Language eine AR-Anwendung selbst zustammenzustellen und mit einer simplen, in Javascript programmierten Anwendungslogik zu versehen. Schließlich sind diese Konzepte einem vergleichsweise großem Personenkreis z.B. von HTML-Webseiten wohlbekannt.

Klar ist, daß es dabei beim gegenwärtigen Stand der Technik keine 100prozentig optimale Lösung wird geben können. Es müssen Kompromisse geschlossen werden zwischen dem, was theoretisch an Anwendungen denkbar ist, und dem, was mit den limitierten heutigen Möglichkeiten machbar ist. Das Ziel dieser Arbeit ist es nicht, eine Lösung für alle Probleme zu finden, sondern die Forschung im Bereich AR voranzutreiben, indem sie einen grundsätzlichen Rahmen für den Aufbau von AR-Systemen vorgibt.

Der weitere Aufbau dieser Arbeit sieht nun folgendermaßen aus:

- In Kapitel 2 werden zunächst Einsatzszenarien für AR-Anwendungen entworfen und daraus Anforderungen an ein AR-Rahmensystem abgeleitet. Es wird ein Überblick über existierende AR-Plattformen gegeben. Basierend auf dem Einsatzszenario und den Anforderungen werden die Grundzüge einer generischen Systemarchitektur einer AR-Plattform entwickelt.
- In den folgenden Kapiteln werden dann einzelne Komponenten der AR-Plattform im Detail beschrieben. In Kapitel 3 wird das Geräte-Management betrachtet. Kapitel 4 beschäftigt sich mit der Netzwerkkommunikation. Kapitel 5 behandelt das Rendering.
- Kapitel 6 beschäftigt sich mit einer Fragestellung, die bisher nur wenig beachtet wurde, nämlich der Entwicklung von mobilen AR-Anwendungen und dem Debugging.
- In Kapitel 7 werden einige Anwendungsbeispiele beschrieben. An konkreten Anwendungen wird der Einsatz des AR-Rahmensystems demonstriert.
- Zu guter Letzt werden in Kapitel 8 die Ergebnisse zusammengefaßt und ein Ausblick auf die zukünftige Entwicklung gegeben.

2 Entwurf einer Systemarchitektur

In diesem Kapitel wird die Architektur eines Rahmensystems für AR-Anwendungen skizziert. Dazu wird zunächst untersucht, wie die Hardware eines mobilen Systems jetzt und in der nahen Zukunft aussieht, und welche Anwendungsszenarien damit kurz- und mittelfristig realisierbar sind. Aus dieser Untersuchung der Hardwareplattformen und Anwendungsszenarien werden dann die Anforderungen an die Systemarchitektur abgeleitet. Nach einem Überblick über existierende Rahmensysteme für AR-Anwendungen und einer Diskussion ihrer spezifischen Vor- und Nachteile wird dann eine eigene Systemarchitektur vorgestellt. In den folgenden Kapiteln wird dann auf einzelne Teilaspekte der Systemarchitektur genauer eingegangen.

2.1 Hardwareplattformen

Wie im vorherigen Kapitel bereits angesprochen, gibt es gegenwärtig praktisch keine leistungsfähige Hardware, die speziell für den Einsatzzweck „Mobile Augmented Reality“ entwickelt wurde. Anstelle dessen wird Hardware verwendet, die aus Standard-Hard- und Software-Komponenten wie Intel-Prozessoren und Windows-Betriebssystemen besteht (sogenannte „Wintel“-Systeme), also typischerweise normalen Laptops. Solchen Laptops fehlt natürlich jegliche Form von Tracking-Hardware, wie z.B. GPS oder Kompass. Anstelle dessen sind sie mit Hardware ausgestattet, die in AR-Systemen nicht genutzt wird, wie z.B. Display, Tastatur, Maus und DVD-Laufwerk. Die Folge davon ist, daß existierende mobile AR-Systeme zu schwer sind, viel zu viel kosten und wegen des unnötig hohen Energieverbrauchs unter viel zu kurzen Laufzeiten leiden.

Betrachtet man z.B. die beim Archeoguide-Projekt verwendete Hardware [15], so stellt man fest, daß der praktische Einsatz nicht wirklich realistisch ist:

- Der Preis des Systems ist für einen wirtschaftlichen Einsatz viel zu groß. Das Gesamtsystem kostet fast 10.000 Euro. Die größten Einzelposten bilden dabei das HMD (über 4.000 Euro) und der Laptop (ca. 2.500 Euro). Dazu kommen noch die Kosten für Kamera, GPS-Empfänger und Kompaß.
- Das Gewicht des Gesamtsystems ist viel zu groß. Allein das Gewicht der Ausrüstung, die in einem Rucksack auf dem Rücken getragen wird, beträgt über 6 Kilogramm. Dazu kommt noch das Gewicht des HMD inklusive Kamera von fast einem Kilogramm. Es ist nicht realistisch, daß der Besucher einer historischen Stätte wärend seines Besuchs ein solches Gewicht mit sich herumtragen möchte.
- Die Batterielaufzeit ist zwar mit ca. zwei Stunden für einen einzelnen Besuch ausreichend. Allerdings muß man berücksichtigen, daß die AR-Systeme als Leihgeräte praktisch im Dauereinsatz stehen würden. Zwar könnte man die Batterien bei der Abgabe des Systems gegen neue austauschen, der dafür benötigte Pool an Batterien wäre allerdings zu teuer und zu unwirtschaftlich.

Bei der praktischen Erprobung des Systems mit Besuchern des antiken Olympia stellte sich neben diesen technischen und wirtschaftlichen Problemen auch noch heraus, daß der Einsatz von HMDs von den meisten Testpersonen abgelehnt wurde. Zum einen sind solche HMDs zu unbequem, zum anderen schirmen sie den Anwender auch zu stark von seiner Umwelt ab. Ein weiteres Problem ist die schlechte Darstellungsqualität der meisten HMDs bei starker Sonneneinstrahlung, wie sie bei einem mobilen Outdoor-System unvermeidlich ist.

Natürlich wird sich dieser Zustand mir der Fortschritt der Technik verbessern. In Zukunft wird es HMDs geben, die in normale Brillen integriert sind und auch unter schlechten

Beleuchtungsverhältnissen ein qualitativ hochwertiges Bild liefern werden. Und auch bei Größe, Gewicht und Laufzeit wird es Fortschritte geben.

Eine interessante Frage ist jedoch, ob sich tatsächlich die allgegenwärtige „Wintel“-Plattform als Plattform für AR-Anwendungen durchsetzen wird. Ein mobiles AR-System hat im Prinzip folgende Hardware-Anforderungen:

- Ein leistungsfähiger Rechner mit einer leistungsfähigen Graphikkarte, auf dem die Anwendung läuft.
- Ein graphisches Ausgabegerät. Ein solches Gerät ist wünschenswerterweise ein HMD, es kann aber auch ein ganz normales Display sein.
- Eine Videokamera, die für videobasiertes Tracking genutzt wird und (bei Video-See-Through-Systemen) die aktuelle Sicht des Anwenders liefert.
- Hardware zur groben Positionsbestimmung, z.B. GPS-Empfänger und Kompasse für Outdoor-Systeme.
- Hardware zur Kommunikation mit stationären Servern oder anderen mobilen Geräten, wie z.B. WLAN oder UMTS.

Tatsächlich gibt es inzwischen Geräte, in denen viele dieser Komponenten integriert sind. Diese Geräte sind Mobiltelefone, Handhelds & PDAs, Navigationsgeräte und mobile Spielekonsolen. Insofern ist es ein logischer Schritt, daß sich immer mehr Forschergruppen mit der Frage beschäftigen, wie man AR auf PDAs [16][17][18][19] oder sogar auf Mobiltelefonen [20][21] einsetzen kann.

Aktuelle Handhelds haben eine Prozessorleistung und einen Speicherausbau, der vor kurzem noch State-of-the-Art bei ausgewachsenen Laptop-Systemen war. Es tauchen, getrieben von dem Wunsch nach hochwertigen mobilen Spieleplattformen, die ersten 3D-Graphikbeschleunigerchips für Handhelds und Mobiltelefone auf. Und in einige Geräte sind sogar schon GPS-Empfänger eingebaut, die für den Einsatz in Navigationssystemen gedacht sind.

Abbildung 5: Die mobile Spielekonsole Gizmondo

Ein Beispiel für ein mobiles Gerät, das alle für AR-Anwendungen benötigten Hardware-Komponenten in sich vereint, war die Spielekonsole „Gizmondo“, die auf der CeBIT 2005 vorgestellt wurde – inzwischen ist der Hersteller allerdings insolvent, und die Konsole wird nicht mehr produziert. Das Gerät war mit einem 400MHz-ARM-9-Prozessor und dem nVidia GoForce 3D 4500 Graphikchip ausgerüstet, der Direct3D-Mobile und OpenGL-ES unterstützt. Als Betriebssystem kam Windows CE zum Einsatz. Die Konsole besaß ein 2,8

Zoll Farb-TFT mit einer Auflösung von 320×240 Pixeln, außerdem eine eingebaute Kamera, einen GPS-Empfänger und zur Kommunikation GPRS und Bluetooth.

Abbildung 6: Apple iPhone (links) und die Sony PlayStation Portable

Weitere Beispiele für potentielle AR-Plattformen sind SmartPhones wie z.B. das Apple iPhone oder mobile Spielekonsolen wie die Sony PlayStation Portable oder die Nintendo DS.

Abbildung 7: Samsung Q1 (links) und Sony Vaio UX

Eine andere interessante Entwicklung sind die sogenannten „Ultra Mobile PCs“ (UMPCs). Diese Geräte basieren auf normaler Laptop-Hardware (Intel Core Solo, Intel Graphics Media Accelerator 950 Graphikkarte, 1GB Hauptspeicher und Festplatte) und Standard-Betriebssystemen wie Windows XP oder Vista. Allerdings haben sie einen Formfaktor, der nahe an den von PDAs herankommt. Beispiele für solche Geräte sind z.B. der Samsung Q1 und der Sony Vaio UX (siehe Abbildung 7). Auch wenn diese Geräte eine sehr kurze Batterielaufzeit haben, sehr leistungsschwach sind und zum Teil schon mit der Ausführung des installierten Betriebssystems überfordert sind, so zeigen sie doch den aktuellen Trend hin zu einer immer stärkeren Miniaturisierung. So ist es sicherlich keine unrealistische Annahme, daß es schon in näherer Zukunft PDAs, mobile Spielekonsolen oder Mobiltelefone geben wird, die unter Standard-Betriebssystemen auf einer Standard-Hardware laufen werden.

Natürlich sind aktuelle Mobiltelefone, PDAs und Spielekonsolen nur mit Einschränkungen für AR-Anwendungen geeignet. Um interaktive Frameraten zu erreichen, muß man Abstriche beim Videotracking und bei der Rendering-Qualität machen. Darüber hinaus gibt es keine Möglichkeit, HMDs anzuschließen – die Darstellung der Szene auf den kleinen Displays ist zwar machbar, aber alles andere als optimal. Trotzdem zeigen diese Geräte, wie mobile AR-Systeme in Zukunft aussehen werden. Genauso, wie Organizer und Navigationssysteme gegenwärtig in Mobiltelefone integriert werden, wird in Zukunft wohl auch AR-Funktionalität integriert werden.

Dieses Szenario für zukünftige AR-Hardwareplattformen hat natürlich Konsequenzen für die Entwicklung von AR-Software. Während man bei der klassischen Plattform für mobile AR-Anwendungen, dem Laptop, eine homogene Umgebung vorfindet (die allgegenwärtigen „Wintel“-Systeme), gibt es bei Mobiltelefonen, PDAs und Spielekonsolen eine Vielzahl von verschiedenen Konfigurationen aus Prozessoren und Betriebssystemen. Eine Rahmensystem für mobile AR-Anwendungen sollte in der Lage sein, mit dieser heterogenen Umgebung zurechtzukommen – d.h. Plattformunabhängigkeit ist eine der wichtigsten Anforderungen.

2.2 Anwendungsszenarien

Wie sehen mögliche Anwendungsszenarien aus, die sich mit der kurz- und mittelfristig verfügbaren Hard- und Software realisieren lassen? Wie bereits im 1. Kapitel erwähnt, gibt es eine Reihe von Anforderungen, die gegenwärtig auf mobilen Low-End-Systemen wie Mobiltelefonen oder PDAs nur schwer oder nicht befriedigend zu erfüllen sind:

- Zuverlässiges und exaktes Bestimmen von Kopfposition und Blickrichtung des Anwenders (Tracking) über größere Distanzen und unter wechselnden Umweltbedingungen. Allerdings werden in diesem Bereich große Fortschritte gemacht, und zumindest auf UMPCs gibt es bereits vielversprechende Lösungen [22].
- Korrekte Beleuchtung und korrekter Schattenwurf der virtuellen Objekte, die in die reale Szene eingeblendet werden.
- Korrekte Verdeckung von virtuellen Objekten durch reale Objekte.

Das hier vorgestellte System soll nicht alle denkbaren AR-Einsatzszenarien abdecken, sondern nur einen Teilbereich, in dem eine kurz- bis mittelfristige Realisierung von praktischen Anwendungen realistisch erscheint. Daher wird (wie z.B. auch in aktuellen Projekten wie z.B. ULTRA [8]) bewußt darauf verzichtet, die drei oben genannten Anforderungen zu erfüllen, was selbstverständlich eine Reihe von Anwendungsszenarien ausschließt, wie z.B. medizinische Anwendungen, bei denen exaktes Tracking unverzichtbar ist. Viele andere Anwendungen sind aber nichtsdestotrotz möglich. Denkbare Einsatzszenarien sind z.B.:

- Der Tourist der Zukunft betritt eine historische Stätte. Sein AR-System verbindet sich automatisch über WLAN mit einem Content-Server, der Informationen über das Gelände liefert, insbesondere über die Position von historischen Bauwerken auf dem Gelände und 3D-Modelle der Rekonstruktionen. Das mobile Gerät bestimmt grob mittels integriertem GPS-Empfänger und Kompaß die Position und Blickrichtung des Besuchers auf dem Gelände. Eine integrierte Kamera nimmt ein Bild der Umgebung auf und zeigt es dem Display an. Ein markerloses, bildbasiertes Trackingsystem fügt ein Bild der rekonstruierten Gebäude in das Videobild ein.
- In vielen technischen Geräten wird ein virtuelles Handbuch verfügbar sein. Zur Inbetriebnahme z.B. eines neuen Videorekorders wird der Anwender sein AR-System verwenden. Über eine drahtlose Übertragungstechnik wie z.B. WLAN oder Bluetooth wird der Videorekorder das virtuelle Handbuch auf das mobile Gerät übertragen. Eine in das mobile Gerät integrierte Kamera nimmt ein Bild des Videorekorders auf und zeigt es auf dem Display an. Die relative Position des Geräts zum Videorekorder wird über ein markerbasiertes Videotrackingsystem bestimmt. Die notwendigen Marker sind dabei geschickt in das Design des Videorekorders integriert. In das Videobild vom Videorekorder werden dann Informationen zu den Bedienelementen o.ä. plaziert.
- Beim Betreten eines unbekannten Gebäudes (z.B. eines Museums, einer Messehalle oder eines Supermarktes) wird man in Zukunft über sein AR-System Informationen

über die Lage von bestimmten Punkten im Gebäude erhalten. Dazu verbindet sich das mobile Gerät automatisch über WLAN mit einem Server, der Informationen und Lagepläne des Gebäudes liefert. Über eine integrierte Kamera werden Bilder von der Umgebung des Anwenders geliefert. In diese Bilder werden z.B. Hinweispfeile eingeblendet, die den Anwender zum gewünschten Ort leiten. Die Position wird dabei über eine geeignete Technologie, wie z.B. Infrarotbaken, Videotracking, oder elektromagnetisches Tracking bestimmt.

- Der Servicetechniker der Zukunft wird ebenfalls virtuelle Handbücher bei der Wartung und Reparatur von Maschinen verwenden. Er kann jedoch auch Hilfe von Experten einholen, die sich nicht vor Ort aufhalten (sogenannte „Remote Experts“). Dazu nimmt er mit seinem mobilen Gerät über UMTS oder eine ähnliche drahtlose Kommunikationsverbindung Kontakt mit dem Experten auf, der in seinem Büro an einem Desktop-Rechner sitzt. Das mobile Gerät überträgt das aktuelle Kamerabild an den Experten. Dieser kann nun das jeweilige Problem auf seinem eigenen Rechner betrachten und analysieren und dem Techniker vor Ort Hinweise geben, entweder über eine Sprachverbindung oder dadurch, daß er direkt in das Blickfeld des Technikers Markierungen vornimmt oder andere Informationen einblendet.

Allen diesen Szenarien ist gemeinsam, daß die jeweilige Anwendung nicht a priori auf dem AR-System des Anwenders installiert ist. Anstelle dessen bekommt der Anwender auf dem Bildschirm des Geräts eine Liste der an seinem jeweiligen Standort verfügbaren Anwendungen. Wenn er eine der Anwendungen auswählt, wird sie automatisch von einem Server auf das mobile Gerät heruntergeladen und ausgeführt. Nach der Ausführung wird die Anwendung wieder vom mobilen Gerät entfernt. Die hier beschriebenen AR-Anwendungen ähneln also im Prinzip Web-Seiten, die von einem Web-Browser dargestellt werden. Analog zum Web-Browser wird also ein „AR-Browser“ benötigt, der die Laufzeitumgebung für die jeweilige AR-Anwendung zur Verfügung stellt.

2.3 Anforderungen an eine Software-Architektur

Die oben beschriebenen Hardware- und Anwendungsszenarien stellen eine Reihe von Anforderungen an eine Software-Architektur für mobile AR-Anwendungen:

- Die wichtigste Anforderung ist, daß Anwendungen, die auf diesem Framework basieren, möglichst weitgehend systemunabhängig sein müssen. Die Anwendung wird von Servern auf eine Vielzahl von unterschiedlichsten, heterogenen Geräten heruntergeladen. Die verwendeten Geräte können Laptops mit den Betriebssystemen Windows, Linux oder Mac OS X sein, PDAs mit den Beriebssystemen Windows CE oder PalmOS, oder Mobiltelefone mit einem Symbian-Betriebssystem. Da nicht von vornherein festgelegt ist, auf welchem Prozessor und auf welchem Betriebssystem die Anwendung laufen wird, muß sie systemunabhängig programmiert werden. Das bedeutet insbesondere, daß Anwendungen nicht in C bzw. C++ oder anderen Programmiersprachen geschrieben werden können, die in systemspezifischen Maschinencode compiliert werden. Anstelle dessen müssen Markup-Languages analog zu HTML für Webseiten oder VRML/X3D für VR-Welten verwendet werden, oder interpretierte Programmiersprachen wie JavaScript und Python, oder Programmiersprachen, die in systemunabhängigen Bytecode übersetzt werden, wie z.B. Java oder C#.
- Mobile Geräte, insbesondere PDAs und Mobiltelefone, besitzen nur vergleichsweise beschränkte Ressourcen. Ein Framework für AR-Anwendungen muß daher diese Ressourcen möglichst effizient verwalten und der Anwendung zur Verfügung stellen. Dies gilt insbesondere vor dem Hintergrund, daß wie eben erwähnt die Anwendungen

über Markup-Languages oder Scriptsprachen realisiert werden müssen, die häufig unter Verdacht stehen, besonders ineffizient zu sein und besonders verschwenderisch mit Ressourcen umzugehen. Aber auch für das Rendering und das Gerätemanagement hat dieser sparsame Umgang mit Ressourcen Konsequenzen. Die übliche Herangehensweise, die Anwendung um eine Renderingschleife herumzubauen, die ständig das Bild neu aufbaut und den Zustand aller angeschlossenen Geräte abfragt („Polling“), ist für mobile AR-Geräte nicht durchführbar. Anstelle dessen muß das System event-basiert sein, d.h. das Bild wird nur dann neu aufgebaut, wenn es sich tatsächlich geändert hat, und Geräte nur ausgelesen, wenn sich ihr Zustand tatsächlich geändert hat.

- Mobile AR-Anwendungen sind typischerweise verteilte Anwendungen, d.h. sie laufen nicht nur auf dem mobilen Gerät, sondern bestimmte Teilsysteme der Anwendung laufen auf stationären Servern, die über Netzwerke mit dem mobilen Gerät verbunden sind. Beispiele dafür sind stationäre Trackingsysteme, die die Position und Orientierung des Anwenders bestimmen und diese Information an das mobile Gerät liefern. Oder Datenbanken, die für die Ausführung der Anwendung notwendige Daten aufbereiten und bereitstellen, wie z.B. Lagepläne eines Gebäudes, Handbücher einer Maschine, etc. Es ist sogar denkbar, einen Großteil der Anwendung auf stationären Rechnern ablaufen zu lassen. Das mobile System dient in diesem Fall nur noch dazu, Videobilder von einer Kamera zu grabben, an einen Server zu schicken, die augmentierten Bilder wieder vom Server zu empfangen und darzustellen. Beispiele für solche Architekturen findet man beim Projekt AR-PDA [16] und bei [17]. Ein Rahmensystem für AR-Systeme muß diese verteilte Natur von mobilen AR-Anwendungen berücksichtigen und Methoden bereitstellen, die Kommunikation verschiedener Softwareteilsysteme im Netzwerk möglichst einfach und transparent für die Anwendung zu ermöglichen.

- Mobile AR-Anwendungen müssen sich möglichst schnell und kostengünstig entwickeln lassen. Das hat verschiedene Konsequenzen: Das AR-Rahmensystem muß übersichtlich und leicht verständlich sein. Es muß Werkzeuge geben, die die Entwicklung und das Debuggen von AR-Anwendungen unterstützen. In diesem Zusammenhang ist es günstig, wenn das Rahmensystem, wo möglich, auf existierende Standards aufsetzt, die gut dokumentiert sind, und für die es unter Umständen sogar schon Entwicklungswerkzeuge gibt. Ein Beispiel dafür ist das Datenformat der 3D-Objekte, die in die AR-Szene eingeblendet werden. Zur Modellierung solcher Objekte gibt es bereits ausgereifte 3D-Modellierungssoftware wie 3Dstudio Max oder Maya. Das AR-Rahmensystem sollte daher ein Datenformat unterstützen, daß von solchen Softwareprodukten direkt erzeugt werden kann.

Im folgenden Text werden nun existierende Rahmensysteme untersucht und überprüft, inwieweit sie die hier aufgestellten Anforderungen unterstützen. Danach wird auf der Basis der Anforderungen ein eigenes, neuartiges AR-Rahmensystem vorgestellt.

2.4 Überblick über existierende AR-Rahmensysteme

In den letzten Jahren ist die Frage, wie man AR-Anwendungen effizient entwickelt, zunehmend in den Fokus des wissenschaftlichen Interesses geraten. Der Grund dafür liegt in der Erkenntnis, daß es zwar eine Vielzahl von AR-Projekten mit vielversprechenden Ergebnissen gibt, aber die Ergebnisse dieser Projekte praktisch immer nur Einzelstücke sind, die auf einen speziellen Anwendungsfall zugeschnitten sind. Dieser Zustand behindert offensichtlich den Fortschritt und den praktischen Einsatz von AR-Systemen. Aus diesem Grunde legt man das Augenmerk verstärkt auf die Software-Architektur, und es sind eine

Reihe von AR-Rahmensystemen entwickelt worden. Sieben von ihnen werden hier untersucht, nämlich ARToolkit, Coterie, Studierstube, DWARF, Tinmith, ImageTclAR, MORGAN und DART. Außerdem werden auch zwei Frameworks für verteilte VR-Systeme kurz betrachtet, Avocado/AVANGO und DIVE.

2.4.1 ARToolkit

ARToolkit [24] ist nicht im eigentlichen Sinne ein AR-Rahmensystem. Es soll hier trotzdem kurz betrachtet werden, weil es Bestandteil einer großen Zahl von AR-Anwendungen ist und den Standard für markerbasiertes Tracking darstellt. ARToolkit wird am Human Interface Technology Laboratory (HITlab) Neuseeland entwickelt. Es handelt sich um eine C-Bibliothek, die auf alle wichtigen Betriebssystemplattformen portiert wurde, insbesondere auch auf PocketPC [26].

ARToolkit erlaubt die Bestimmung von Position und Orientierung von Markern relativ zu einer Kamera (videobasiertes Tracking unter Verwendung von Markern). Die Marker sind quadratische Flächen, die von einem schwarzen Rahmen begrenzt werden, der die Segmentierung der Marker im Bild erlaubt. Der Rahmen ist mit einem beliebigen Muster gefüllt, das mittels Pattern Matching die Identifikation verschiedener Marker ermöglicht. Es können theoretisch beliebig viele Marker gleichzeitig verwendet werden, allerdings sinkt die Erkennungsrate und die Systemperformanz mit zunehmender Anzahl von Markern. Objekte können mit mehreren Markern ausgestattet werden, was die Bestimmung von Position und Orientierung dieser Objekte erlaubt, auch wenn einzelne Marker im Videobild verdeckt sind.

Neben der Tracker-Funktionalität stellt ARToolkit auch einige primitive Funktionen bereit, die das Rendern mit OpenGL erleichtern. Es handelt sich jedoch nicht um ein echtes Renderingsystem.

Aktuelle Weiterentwicklungen von ARToolkit sind ARToolkitPlus und OSGART. ARToolkitPlus wird an der TU Graz entwickelt. Der Schwerpunkt liegt dabei auf der Portierung des Systems auf PDAs und Mobiltelefone. OSGART vom HITlab integriert ARToolkit in den Szenengraphen „OpenSceneGraph[1]“.

ARToolkit und ARToolkitPlus stellen somit trotz des mißverständlichen Namens nur eine Teilkomponente eines AR-Systems bereit, nämlich das markerbasierte Tracking. Sie sind kein vollständiges Rahmensystem zur Entwicklung von AR-Anwendungen. OSGART dagegen stellt auch Renderingfunktionalität zur Verfügung und kann damit als AR-Rahmensystem betrachtet werden, allerdings fehlen wichtige Teilkomponenten z.B. zur Kommunikation mit anderen Softwarekomponenten über das Netzwerk.

2.4.2 COTERIE

COTERIE [27] ist das älteste der hier betrachteten Systeme. Es wurde am Department of Computer Science der Columbia University, New York, entwickelt. COTERIE ist in Modula-3 geschrieben und stellt eine Reihe von Packages (d.h. Bibliotheken) für diese Programmiersprache bereit. Unterstützt werden die Betriebssysteme Windows sowie eine Reihe von Unix-Systemen.

Kernkonzept von COTERIE ist die anwendungstransparente Replikation von Datenobjekten („Shared Objects“) in Prozessen, die auf dem gleichen Rechner laufen oder auf verschiedenen Rechnern, die per Netzwerk miteinander verbunden sind („Distributed Shared Memory“). Um ein solches „Shared Object“ zu erzeugen, muß der Anwendungsprogrammierer ein Modula-3-

---

[1] http://www.openscenegraph.org/

Objekt anlegen, das keine öffentlichen Membervariablen besitzt und von der Klasse „SharedObj“ abgeleitet wird. Wenn eine der Membervariablen über eine Memberfunktion verändert wird, wird diese Änderung über das Netzwerk automatisch einem sogenannten „Sequencer“-Prozess mitgeteilt. Dieser Sequencer-Process teilt die Änderung dann allen anderen Prozessen mit, die das „Shared Object“ ebenfalls verwenden.

Um z.B. die Daten eines Tracking-Systems im System zu verteilen, erzeugt der Prozess, der den Tracker betreibt, ein Tracker-Shared-Object, in das er die aktuelle Position und Orientierung schreibt. Alle Prozesse, die an den Daten des Trackers interessiert sind, erzeugen ebenfalls ein Tracker-Shared-Object, das nun automatisch die Tracker-Daten vom Tracker-Prozess erhält.

Ein wichtiges Konzept von COTERIE sind die „Callback Objects“. Zu jedem Shared Object gibt es ein Callback Object. Wenn eine Membervariable des Shared Objects verändert wird, wird eine zugehörige Methode des Callback Objects aufgerufen. Anwendungen können diese Methode überschreiben und werden so über Änderungen der Membervariablen informiert.

Anwendungen, die auf COTERIE basieren, können nicht nur in Modula-3 programmiert werden, sondern auch in der Script-Sprache „Obliq“. Auf diese Weise ist es möglich, auf der einen Seite geschwindigkeitskritische Codeteile in einer effizienten, compilierten Sprache (Modula-3) zu programmieren, auf der anderen Seite aber auch schnell und systemunabhängig Anwendungen in einer interpretierten Skript-Sprache (Obliq) ablaufen zu lassen.

2.4.3 Studierstube[2]

Studierstube [28] wird an der Technischen Universität Graz entwickelt. Es ist eines der bekanntesten AR-Rahmensysteme. Es gibt mehrere Versionen des Systems. Der Fokus des ursprünglichen Systems liegt dabei auf neuen User-Interface-Konzepten für AR-Anwendungen und auf der Fragestellung, wie mehrere Anwender gleichzeitig in einer virtuellen Umgebung zusammenarbeiten können. Neuere Versionen konzentrieren sich auf mobiles AR auf Low-End-Geräten wie PDAs und Mobiltelefonen.

Studierstube ist in C++ geschrieben und basiert auf OpenInventor [65]. OpenInventor ist ein Rendering-Toolkit, das auf dem Konzept des Szenengraphen basiert. Es stellt eine Reihe von vorgefertigten Knoten zur Verfügung, aus denen die Anwendung den Szenengraphen zusammenbaut. Knoten besitzen Felder, in denen Daten gespeichert werden, und können untereinander kommunizieren. Der Szenengraph kann in einer Inventor-Datei abgelegt und zur Laufzeit wieder aus dieser Datei rekonstruiert werden.

AR-Anwendungen für Studierstube werden in Form von neuen Szenengraphen-Knoten entwickelt. Die Anwendungsklasse erbt von einer abstrakten Studierstube-Basisklasse. Alle Daten der Anwendung werden in den Feldern des Knotens abgelegt. Die graphische Repräsentation der Anwendung wird von Kind-Knoten erzeugt, die unterhalb des Anwendungsknotens in den Szenegraph eingehängt werden. Die Anwendungs-Knoten werden als Plugins auf der Festplatte abgelegt und von der Studierstube-Laufzeitumgebung zur Laufzeit nachgeladen. Dieses Konzept erlaubt es unter anderem auch, mehrere Anwendungen gleichzeitig laufen zu lassen (Stichwort Multitasking).

Studierstube ist netzwerktransparent und erlaubt es, das System auf mehreren Rechnern verteilt laufen zu lassen. Dazu wurde das Laufzeitsystem um eine Softwarekomponente erweitert, die es erlaubt, den OpenInventor-Szenengraphen auf verschiedenen Rechnern zu

---

[2] http://www.studierstube.org/

synchronisieren (Distributed OpenInventor [29]). Jede Änderung an einem Knoten-Feld des Master-Szenengraphen wird dabei über Multicast-Nachrichten den Slave-Szenengraphen mitgeteilt, die diese Änderungen übernehmen.

2.4.4 DWARF

Das „Distributed Wearable Augmented Reality Framework“ (DWARF) [30] wird seit 2000 an der Technischen Universität München entwickelt.

DWARF besteht aus einer Reihe von Softwaremodulen. Jedes Softwaremodul behandelt eine separate, klar umrissene Aufgabe. Es stellt eine Reihe von Diensten zur Verfügung, und es benötigt selbst eine Reihe von Diensten. Beim Start des Systems registriert sich jedes Modul bei einem Servicemanager. Dabei teilt es dem Servicemanager mit, welche Dienste es zur Verfügung stellt, welche es selbst für seine Arbeit benötigt, und auf welche weise es mit anderen Modulen kommunizieren kann (z.B. über CORBA oder Shared Memory). Der Service-Manager verbindet automatisch Softwaremodule miteinander, wenn ein Modul genau den Service anbietet, den ein anderes benötigt, und wenn es ein Kommunikationsverfahren gibt, das beide unterstützen. Dabei können Softwaremodule auch in verschiedenen Prozessen und sogar auf verschiedenen Rechnern ablaufen – DWARF ist also netzwerktransparent.

Folgende Softwaremodule werden von DWARF angeboten:

- Ein Tracker-Modul. Dieses Tracker-Modul besteht aus verschiedenen Gerätetreibern für Tracking-Hardware, wie z.B. GPS und elektronischer Kompaß. Darüber hinaus gibt es einen „Tracking Manager“, der aus den vorliegenden Tracking-Daten verschiedener Tracking-Geräte die aktuelle Position und Orientierung berechnet (Hybrides Tracking).
- Eine Datenbank mit Informationen über reale Objekte und virtuelle Objekte (das „World Model“. Diese Datenbank enthält z.B. die Lage von Räumen und die Position von realen und virtuellen Objekten in diesen Räumen, oder die 3D-Geometrie von virtuellen Objekten.
- Die „Taskflow Engine“, die den Abblauf der Anwendung festlegt. Es handelt sich um eine Statusmaschine, bei der der Übergang von einem Zustand zum anderen durch externe Ereignisse ausgelöst wird, die von externen Sensoren (z.B. Trackern) registriert werden. Die Zustände und die Übergänge der Statusmaschine werden in einer „taskflow description language“ (TDL) definiert, einer XML-Datei, die beim Systemstart eingelesen wird.
- „Context-aware service access“ (CAP). Dieses Modul erlaubt es dem Anwender, kontext-abhängige Dienste in Anspruch zu nehmen, z.B. um ein Dokument an dem Ort auszudrucken, an dem man sich gerade befindet.
- Die „User interface engine“. Dieses Modul stellt ein multimodales User-Inface zur Verfügung. Das Interface wird in einer Erweiterung der „User Interface Markup Language“ (UIML) [32], der „Cooperative UIML“ (CUIML) [34], beschrieben. UIML erlaubt es, User-Interfaces unabhängig von den verwendeten Ein- und Ausgabegeräten zu beschreiben.

Eine DWARF-Anwendung ist selber eine Software-Komponente. Ihre Hauptaufgabe besteht darin, das System zu starten und ein Bindeglied zwischen den verschiedenen Softwaremodulen darzustellen.

DWARF ist von allen hier vorgestellten AR-Rahmensystemen sicherlich das fortschrittlichste und ehrgeizigste. Das Systems besitzt allerdings keinen integrierten Szenengraphen – es können verschiedene Renderingsysteme angebunden werden, die jedoch jeweils komplett in einer einzelnen Softwarekomponente gekapselt werden.

2.4.5 Tinmith

Das Tinmith-System [35] wird an der University of South Australia entwickelt. Es ist speziell als Grundlage für mobile AR-Anwendungen konzipiert worden, die auf leistungsschwacher Hardware laufen. Besonderes Augenmerk wurde daher auf Effizienz gelegt. Tinmith ist in C++ programmiert und läuft auf Unix-Plattformen.

Tinmith basiert auf einem Datenflußkonzept. Die Anwendung besteht aus C++-Klassen, die von einer speziellen Basisklasse abgeleitet werden. Die Instanzen dieser Klassen können Daten empfangen, verarbeiten und an andere C++-Objekte weiterleiten. Das Weiterleiten von Daten geschieht über Callbacks. Jedes Objekt, das Daten von einem anderen Objekt empfangen möchte, registriert eine Callback-Funktion bei diesem Objekt, die aufgerufen wird, sobald sich das Objekt ändert. Da Objekte selber andere Objekte beinhalten können, können Callbacks auf verschiedenen Ebenen der Objekt-Hierarchie eingesetzt werden – Callbacks werden aufgerufen, wenn sich ein Objekt ändert, oder wenn sich eines der Objekte ändert, aus denen es zusammengesetzt ist.

Alle Objekte werden in einem speziellen Repository gespeichert. Dieses Repository lehnt sich an das Konzept von Dateisystemen an. Jedes Objekt hat einen „Pfad“, unter dem es im Repository erreicht werden kann. Auch hier wird wieder der hierarchische Aufbau von Objekten berücksichtigt – so könnte man z.B. ein Tracker-Objekt unter dem Pfad „Tracker“ erreichen, die von dem Tracker gelieferten Positionsdaten unter dem Pfad „Tracker/Position“, und die Orientierungsdaten unter dem Pfad „Tracker/Orientation“. Wie bei Dateisystemen ist es möglich, Hard- und Softlinks zu erzeugen, d.h. Objekte unter mehreren Pfaden gleichzeitig im Repository abzulegen. Das Repository erlaubt es, daß verschiedene Objekte im System sich nicht „kennen“ müssen, um miteinander zu kommunizieren, d.h. daß die Verbindungen untereinander nicht zur Compile-Zeit festgelegt werden müssen, sondern zur Laufzeit dynamisch erzeugt werden können.

Jedes Objekt ist reflektiv, d.h. es besitzt Meta-Informationen über seinen Typ und über die in ihm enthaltenen Objekte. Diese Reflektivität wird von der Basisklasse bereitgestellt, von der alle anderen Klassen abgeleitet werden. Objekte besitzen darüber hinaus die Möglichkeit, ihren Zustand (d.h. die in ihnen enthaltenen Daten) in einer XML-Datei abzulegen bzw. aus einer XML-Datei wieder herzustellen. Dies erlaubt es, eine Anwendung, d.h. die Struktur des Datenflußgraphen, in einer XML-Datei abzulegen und zu einem späteren Zeitpunkt auf der selben oder einer anderen Maschine wieder zu laden.

Tinmith unterstützt Netzwerktransparenz. Das Repository-Konzept erlaubt es nicht nur, Objekte innerhalb eines Prozesses miteinander zu verbinden, sondern auch Objekte auf unterschiedlichen Maschinen. Eine spezielle Netzwerk-Komponente registriert dann einen Callback auf das Objekt, das Daten liefern soll. Wenn sich das Objekt ändert, speichert die Netzwerk-Komponente den Zustand des Objekts in einer XML-Datei, transferiert diese Datei über das Netzwerk, und rekonstruiert das Objekt auf der Empfängerseite.

2.4.6 ImageTclAR

ImageTclAR [36] ist ein AR-Rahmensystem, das an der Michigan State University entwickelt wird. Es basiert auf der Script-Sprache „Tool Command Language“ (Tcl) [37]. Tcl ist eine interpretierte Sprache, die innerhalb einer Laufzeitumgebung ausgeführt wird. Diese Laufzeitumgebung kann um eigene Module erweitert werden, die in einer effizienten, compilierten Sprache wie C oder C++ geschrieben werden. Auf diese Weise kann den Scripten zusätzliche Funktionalität zur Verfügung gestellt werden.

ImageTclAR stellt genau so eine Spracherweiterung dar. Es erweitert Tcl um Funktionalitäten, die für die Entwicklung von AR-Anwendungen benötigt werden. Dabei greift es selbst auf zwei andere Spracherweiterungen zurück. Zum einen auf „Tk“, einer Standardkomponente von Tcl, die es erlaubt, graphische Oberflächen zu erzeugen. Zum anderen auf der „ImageTcl“-Erweiterung [38], die Funktionalität zur Verfügung stellt, um Multimedia-Daten zu verarbeiten, also z.B. Videobilder und Audiosamples zu grabben, zu übertragen und darzustellen bzw. abzuspielen. ImageTclAR erweitert nun Tcl um spezielle AR-Komponenten, u.a. Trackermodule zum Bestimmen von Position und Orientierung, ein Joystickmodul zum Anbinden von Joysticks, ein VRML-Import-Modul zum Laden von VRML-3D-Objekten, ein Kalibrierungsmodul zum Kalibrieren des Systems, sowie einem Darstellungsmodul, mit dem die AR-Szene auf Anzeigegeräten wie z.B. einem HMD dargestellt werden kann.

ImageTclAR stellt keine Lösungen zur Verfügung, um netzwerktransparent auf Geräte und Dienste auf anderen Rechnern zuzugreifen.

2.4.7 MORGAN

Das MORGAN-Framework [39][40] wird am Fraunhofer-Institut für Angewandte Informationstechnik entwickelt. MORGAN ist ein komponentenorientiertes Framework für dynamische Mehrbenutzer-AR- und VR-Anwendungen. Es verwaltet mehrere Szenengraphen, die für Mehrbenutzeranwendungen auf mehrere Rechner repliziert werden können und vom System automatisch synchronisiert werden. Der interne Szenengraph enthält nur die Daten, die zum Rendering notwendig sind. Beliebig viele externe Szenengraphen enthalten anwendungsspezifische Daten. Darüber hinaus enthält das Framework Module für den netzwerktransparenten Zugriff auf Geräte wie elektromagnetische und videobasierte Trackingsysteme. Die Kommunikation der verschiedenen Softwarekomponenten im Netzwerk wird über CORBA abgewickelt. Der Zugriff auf das Framework erfolgt über eine C++-API.

Der Fokus vom Morgan liegt auf der einfachen Erstellung von Mehrbenutzer-AR-Systemen.

2.4.8 DART

The Designer s Augmented Reality Toolkit (DART) [41] wird am Georgia Institute of Technology entwickelt. DART verwendet einen komplett anderen Ansatz als alle anderen hier betrachteten AR-Rahmensysteme. Der Fokus liegt nicht auf der Lösung technischer Probleme, sondern auf der Frage, wie man AR als Medium auch Personen zugänglich machen kann, die keine Programmierer sind, wie z.B. Designern. Es erweitert das Authoring-Tool Macromedia Director um Komponenten, die es erlauben, AR-Shockwave-Anwendungen zu erzeugen. Man erstellt AR-Anwendungen also wie ganz normale Shockwave-Anwendungen in einer robusten, ausgereiften Entwicklungsumgebung und kann sie dann überall dort abspielen, wo ein Shockwave-Player zur Verfügung steht.

2.4.9 Avocado/AVANGO

Avocado [82][83] ist ein verteiltes VR-Framework, das am Fraunhofer Institut für Intelligente Analyse- und Informationssysteme entwickelt wird. Es basiert auf dem Szenengraphen Performer [42], den es um Sensoren, Animationen und Interaktivität erweitert. Dazu werden von den ursprünglichen Performer C++-Klassen neue Klassen abgeleitet, die als „Fieldcontainer“ fungieren, d.h. den Zustand eines Objektes in Feldern speichern. Felder gleicher oder verschiedener Objekte können miteinander verbunden werden, wenn sie zueinander kompatibel sind. Auf diese Weise wird der ursprünglich statische Performer-

Szenengraph um einen Datenflußgraph erweitert, der es erlaubt, dynamisches Verhalten in die Szene einzubauen.

Avocado erlaubt es, Teile des Graphen transparent für die Anwendung auf verschiedenen Rechnern im Netzwerk zu replizieren. Änderungen an einer Kopie werden automatisch auch an allen anderen Kopien durchgeführt. Auf diese Weise kann man verteilte VR-Anwendungen schreiben oder Geräte einbinden, die auf anderen Rechnern im Netzwerk laufen.

Die komplette Avocado-API kann über die Script-Sprache „Scheme“ [43] angesprochen werden. Es werden üblicherweise nur Performanz-kritische Teile in C++ programmiert, indem von vorhandenen Avocado-Klassen abgeleitet wird. Die Anwendung selber besteht aus Scheme-Scripts.

2.4.10 DIVE

Distributed Interactive Virtual Environment (DIVE) [44][45] ist eine Plattform für verteilte Multi-User VR-Welten, die am Swedish Institute of Computer Science entwickelt wird. Eine DIVE-Welt ist eine hierarchische Datenbank von sogenannten „Entities“. Neben graphischen Objekten und Verhaltensbeschreibungen können diese Entities auch Anwendungsdaten enthalten. Netzwerktransparenz wird dadurch erreicht, daß Teile dieser Datenbank transparent für die Anwendung auf verschiedenen Rechnern repliziert werden können. Änderungen an einer Kopie der Datenbank werden automatisch auch an allen anderen Kopien durchgeführt. Auf diese Weise können sich mehrere Anwender über das Internet in einer VR-Welt aufhalten, diese manipulieren und miteinander interagieren. DIVE unterstützt auch Video- und Audio-Kommunikation zwischen den Anwendern.

2.4.11 Bewertung der AR-Rahmensysteme

Bei den vorgestellten AR-Rahmensystemen findet man eine Reihe von interessanten Konzepten. Die meisten der hier vorgestellten Systeme basieren auf einer Form von Datenflußgraph, d.h. sie bestehen aus mehr oder weniger selbständigen Objekten, die miteinander kommunizieren. Der Vorteil des Datenflußgraphen ist, daß er zum einen sehr elegant eine Netwerktransparenz des Systems ermöglicht, indem man Kommunikationskanäle zwischen Objekten auf verschiedenen Rechnern erlaubt. Zum anderen stellen Datenflußgraphen eine event-getriebene Anwendungsarchitektur dar, die nur auf Zustandsänderungen reagiert, anstatt ständig Ressourcen in einer Renderingschleife oder beim Abfragen von Geräten („Polling“) zu verschwenden.

Darüber hinaus zeigt ein Teil der vorgestellten Systeme, insbesondere ImageTclAR, wie man den scheinbar unauflöslichen Widerspruch zwischen Plattformunabhängigkeit und Effizienz überwinden kann. Plattformunabhängigkeit bedeutet fast zwangsläufig, daß man auf Script-Sprachen zurückgreifen muß. Solche Script-Sprachen können niemals die Effizienz von kompilierten Softwaremodulen erreichen. Das Rahmensystem muß daher so konzipiert sein, daß es für alle zeitkritischen Aufgaben, wie z.B. Rendering, Behandlung von Videoströmen, und Video-Tracking, Softwaremodule gibt, die in einer effizienten Programmiersprache wie C oder C++ entwickelt werden. Die eigentliche, per Script programmierte Anwendung stellt dann nur noch das Bindeglied zwischen diesen fertigen Modulen dar.

2.5 Konzeption einer neuartigen Systemarchitektur

In diesem Kapitel wird nun die Systemarchitektur eines neuartigen Rahmensystems für mobile AR-Anwendungen vorgestellt. Ein wichtiges Ziel ist, daß sie weitestgehend unabhängig von der Hard- und Softwareplattform ist, auf der sie später laufen soll.

Das in dieser Arbeit vorgestellte System läuft gegenwärtig auf normalen Laptops, Tablet-PCs oder UMPCs unter den Betriebssystemen Windows, Linux und Mac OS X. Es wurde nicht auf Mobiltelefonen oder PDAs getestet. Das bedeutet aber nicht, daß die hier vorgestellten Konzepte auf solchen Geräten keine Gültigkeit haben. Wenn man die gegenwärtige technische Entwicklung betrachtet, stellt man fest, das PDAs unter dem Betriebssystem Windows Mobile vom Markt verschwinden. Anstelle dessen tauchen UMPCs auf, die einen ähnlichen Formfaktor haben wie PDAs, aber normale Desktop-Betriebssysteme verwenden. Auch das iPhone verwendet eine auf ARM portierte Version von Mac OS X. Andere Smartphones basieren auf Linux. Die begründete Vermutung ist daher, daß die Unterschiede zwischen „normalen“ Rechnern und Mobiltelefonen immer kleiner werden und wir schon in naher Zukunft Mobiltelefone sehen werden, auf denen ein normales Windows XP oder Windows Vista laufen wird.

Wie oben beschrieben kann der Entwickler einer Anwendung nicht vorhersehen, ob sie später auf einem Laptop, einem PDA oder einem Mobiltelephon laufen wird. Die Konsequenz daraus ist, daß man Anwendungen nicht in einer Programmiersprache wie C oder C++ entwickeln kann, die in systemspezifischen Maschinencode compiliert wird. Anstelle dessen müssen Markup-Languages wie HTML für Webseiten und VRML/X3D für 3D-Welten, interpretierte Scriptsprachen wie JavaScript und Python, oder Programmiersprachen wie Java oder C#, die in systemunabhängigen Bytecode compiliert werden, verwendet werden.

Auf den ersten Blick steht man hier vor einem unlösbaren Dilemma: Auf der einen Seite muß das Framework besonders effizient mit den beschränkten Ressourcen mobiler Geräte umgehen. Auf der anderen Seite sollen Scriptsprachen verwendet werden, die in dem Ruf stehen, besonders verschwenderisch mit Ressourcen umzugehen. Das Beispiel VRML [46] und sein Nachfolger X3D [47] zeigen, wie man diesen Widerspruch auflösen kann.

VRML ist das Standard-Datenformat für die Präsentation von 3D-Welten im Internet. Es stehen eine Reihe von VRML-Browsern frei zum Download zur Verfügung, die als Plugins in Webbrowser wie Internet Explorer und Firefox integriert werden können. Klickt der Anwender im Webbrowser auf einen Link, der auf eine VRML-Welt verweist, so wird der VRML-Browser gestartet, der dem Anwender die VRML-Welt präsentiert.

Der folgende Abschnitt zeigt beispielhaft den Source-Code einer VRML-Szene sowie einen Screenshot dieser Szene in einem VRML-Browser:

Eine einfache VRML-Szene

```
#VRML V2.0 utf8

Viewpoint {
  position 0 0 5
  orientation 0 0 1 0
}

DEF transform Transform {
  children [
    Shape {
      appearance Appearance {
        material Material { diffuseColor 1 0 0 }
      }
      geometry Teapot {}
    }
  ]
}
```

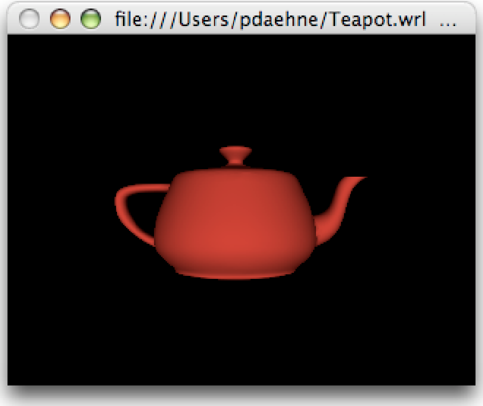


Abbildung 8: Screenshot der obigen VRML-Szene

Wie man sieht, beschreibt der VRML-Code einen typischen Szenengraphen, in dem die inneren Knoten aus Gruppierungs- und Transformationsknoten bestehen und in die Blätter aus den eigentlichen Geometrie-Knoten sowie der Kamera.

Der Unterschied von VRML zu anderen 3D-Dateiformaten wie COLLADA besteht jedoch darin, daß es nicht nur zur Beschreibung des Szenengraphen sowie der in ihm enthaltenen geometrischen Objekte dient. Vielmehr kann man mit VRML auch das dynamische Verhalten dieser Objekte beschreiben – eine wichtige Fähigkeit, die VRML von seinem Vorgänger Open Inventor [65] geerbt hat.

Jeder Knoten im VRML-Szenengraph ist eine kleine Statusmaschine. Er hat einen bestimmten Zustand, eine Reihe von Eingängen, auf denen er Daten empfangen kann, und eine Reihe von Ausgängen, auf denen er Daten verschicken kann. Wenn Daten über einen Eingang empfangen werden, geht der Knoten in einen neuen Zustand über und verschickt entsprechende Daten über seine Ausgänge. Ausgänge und Eingänge der Knoten können beliebig miteinander verbunden werden.

Der folgende Abschnitt zeigt z.B., wie man den obigen Szenengraphen erweitern muß, damit das geometrische Objekt rotiert. Dazu wird der Graph um einen TimeSensor-Knoten und einen Interpolationsknoten erweitert. Der TimeSensor-Knoten steuert den Interpolationsknoten, und die Ergebnisse des Interpolationsknotens werden an den Transformationsknoten über dem Geometrieknoten weitergeleitet.

Erweiterung des Szenengraphen um dynamische Elemente

```
DEF ts TimeSensor {
  loop TRUE
  cycleInterval 5
}

DEF oi OrientationInterpolator {
  key [ 0 0.25 0.5 0.75 1 ]
  keyValue [ 0 1 0 0, 0 1 0 1.5708, 0 1 0 3.1416, 0 1 0 4.71239, 0 1 0 0]
}

ROUTE ts.fraction_changed TO oi.set_fraction
ROUTE oi.value_changed TO transform.set_rotation
```

De facto wird damit neben dem Szenengraph ein zweiter Graph etabliert, nämlich ein Datenflußgraph. Dieser Datenflußgraph erlaubt es, komplette Anwendungen aus einem vorgefertigten Satz von kleinen Softwaremodulen zusammenzusetzen, von denen jedes eine kleine, eng begrenzte Aufgabe übernimmt und mit beliebigen anderen Softwaremodulen kommunizieren kann. Ergänzt wird dieses Konzept durch den speziellen Script-Knoten. Dieser Knoten erlaubt es dem Anwendungsentwickler, neue Softwaremodule zu entwickeln, die beliebige Ein- und Ausgänge haben und deren Verhalten in Programmiersprachen wie Java und JavaScript/ECMAScript [48] programmiert werden kann.

VRML hat mit zwei Vorurteilen zu kämpfen: Zum einen sei es langsam, verglichen mit der direkten Programmierung einer 3D-Anwendung unter Verwendung von APIs wie OpenGL oder Direct3D. Zum anderen sei es nur für 3D-Anwendungen im Internet geeignet, die auf Desktop-Computern ablaufen.

Beides stimmt so nicht. Der Vorwurf der Langsamkeit stützt sich meist auf zwei Punke: Zum einen auf dem Mißverständnis, VRML sei eine Scriptsprache und müsse bei jedem Bildaufbau erneut interpretiert werden. Tatsächlich ist VRML eine Markup Language, d.h. der VRML-Code wird nur beim Einlesen der VRML-Datei interpretiert. Das Ergebnis dieses Interpretationsvorgangs ist eine Datenstruktur im Speicher des Rechners, die den Szenengraph und den Datenflußgraph repräsentiert. Zum Rendern der 3D-Welt traversiert die in einer effizienten Programmiersprache wie C oder C++ entwickelte VRML-Laufzeitumgebung nur noch diese internen Datenstrukturen. Die Effizienz von in VRML programmierten Anwendungen ist also vergleichbar mit der von in C oder C++ programmierten Anwendungen, die direkt auf OpenGL oder Direct3D aufsetzen.

Zum anderen stützt sich der Vorwurf der Langsamkeit auf die Tatsache, daß die Traversierung von schlecht konstruierten Szenengraphen sehr langsam sein kann. Zugegebenermaßen sind die meisten der verfügbaren VRML-Browser nicht besonders effizient programmiert. Unter Umständen kann eine VRML-Laufzeitumgebung aber sogar bessere Ergebnisse liefern als direkt programmierte Anwendungen. Die VRML-Laufzeitumgebung besitzt die Freiheit, die interne Darstellung der Graphen zu optimieren. Der in dieser Arbeit verwendete VRML-Renderer basiert auf dem freien Szenengraphen OpenSG [81], der unter anderem automatisch Optimierungen durchführt:

- View Frustum Culling, bei dem Objekte, die komplett außerhalb des aktuellen View-Frustums liegen, nicht gerendert werden.
- Occlusion Culling, bei dem Objekte, die komplett von anderen Objekten verdeckt werden, nicht gerendert werden.
- Small Feature Culling, bei dem Objekte, deren Größe eine bestimmte Schwelle unterschreitet, nicht gerendert werden.
- State-Sorting, bei dem Objekte nach ihren Materialien sortiert werden.

- Striping, bei dem benachbarte geometrische Primitive wie Dreiecke zu zusammenhängenden Dreiecks-Streifen zusammengefaßt werden.
- Progressive Meshes, bei denen automatisch vereinfachte Varianten von Geometrien berechnet und gerendert werden, wenn diese weit vom Betrachter entfernt sind.

Die Traversierung von Szenengraphen ist also nicht zwangsläufig langsam, sondern erfordert nur eine gewisse Aufmerksamkeit beim Entwurf des Szenengraphen. Natürlich können die oben genannten Optimierungen auch bei direkter OpenGL- bzw. Direct3D-Programmierung angewendet werden, der Vorteil von Szenengraphen ist jedoch, daß diese Optimierungen automatisch durchgeführt werden, ohne daß sich der Anwendungsentwickler explizit darum kümmern muß. Ein anderer Vorteil besteht darin, daß jede neue Optimierung, die in den Szenengraphen eingebaut wird, automatisch allen Anwendungen zur Verfügung steht, die auf dem Szenengraphen basieren.

Auch das zweite Vorurteil, VRML sei nur für Internetanwendungen geeignet, die auf Desktop-Rechnern ablaufen, stimmt so nicht. Richtig ist, daß der VRML-Standard hauptsächlich auf dieses Anwendungsgebiet abzielt und die frei verfügbaren VRML-Browser Desktop-Anwendungen sind. VRML-Anwendungen können aber auch als richtige VR-Anwendungen in Mehrseiten-Projektionssystemen und bei stereographischer Darstellung genutzt werden, wenn die Laufzeitumgebung dies unterstützt [23][49].

Allerdings hat VRML zugegebenermaßen Schwächen bei der Unterstützung von VR- und AR-Anwendungen. Der modulare Aufbau von VRML macht es aber leicht, eigene Knotentypen zu spezifizieren, um fehlende Funktionalität nachzurüsten.

Zusammenfassend kann man feststellen, daß VRML einen Weg aus dem Dilemma der hohen Effizienz bei gleichzeitiger Systemunabhängigkeit zeigt. Es ist daher wenig verwunderlich, daß VRML einen Kernbestandteil des AR-Frameworks bildet, der in dieser Arbeit vorgestellt wird. Tatsächlich übernimmt das selbstentwickelte VRML-Laufzeitsystem „Avalon“ [33] das gesamte Rendering und die für die AR-Anwendungen notwendige Anwendungslogik in diesem Framework. Mehr dazu im 5. Kapitel.

Eine AR-Anwendung besteht jedoch nicht nur aus Rendering und Anwendungslogik. Ein Großteil der Entwicklungsarbeit besteht im Einbinden einer Vielzahl von Geräten und der Verarbeitung der von ihnen gelieferten Daten. Diese Aufgabe übernimmt ein Gerätemanagement-System, das vom VRML-Datenflußgraphen inspiriert ist.

Das Gerätemanagement besteht aus einer Vielzahl von kleinen Software-Modulen, die wie die VRML-Knoten Statusmaschinen sind. Sie besitzten Eingänge und Ausgänge, die miteinander verbunden werden können. Auf diese Weise bilden diese Module einen Datenflußgraphen, der nicht nur einfachen, flexiblen und effizienten Zugriff auf Geräte bietet, sondern auch bereits einen Großteil der Vorverarbeitung der von diesen Geräten gelieferten Daten durchführt.

Wie sehen die Softwaremodule des Gerätemanagements aus? Zum einen gibt es natürlich Module, die die Rolle von Gerätetreibern übernehmen, d.h. sie liefern die Positionsdaten von Trackern, behandeln Eingabegeräte wie Joysticks, grabben Videobilder von Framegrabberkarten oder Webcams, nehmen Audioströme über Mikrophone auf oder spielen sie über Lautsprecher wieder ab. Ein deutlich größerer Teil der Softwaremodule spielt jedoch die Rolle von Filtern, d.h. sie übernehmen einen Großteil der Verarbeitung der von den Gerätetreibern gelieferten Daten. So gibt es z.B. Module, die von Trackern gelieferte Positionsdaten in andere Koordinatensysteme transformieren, oder es gibt komplette Videotracking-Module, die aus Videobildern die Position und Orientierung des Anwenders oder von Objekten relativ zum Anwender berechnen.

Dieses Konzept des Datenflußgraphen ist entscheidend für die Leistungsfähigkeit des AR-Rahmensystems. Zeitkritische Aufgaben wie videobasiertes Tracking werden nicht von der Anwendung durchgeführt, sondern von speziell auf die jeweilige Hardware- und Betriebssystemplattform zugeschnittenen Softwaremodulen. Die Anwendung bedient sich nur noch aus diesem „Pool“ von Softwaremodulen und kombiniert sie in einer für die jeweilige Anwendung nützliche Weise.

Die Details des Gerätemanagements werden im 3. Kapitel beschrieben.

Wie sieht nun konkret eine Anwendung aus, die auf dem hier beschriebenen Rahmensystem basiert? Sie besteht im Prinzip nur aus einer Reihe von Text-Dateien:

- Zunächst einmal natürlich VRML/X3D-Dateien. Diese Dateien enthalten alle graphischen Bestandteile der Anwendung, also den Szenengraphen und alle 3D-Objekte.
- Darüber hinaus Konfigurationsdateien des Gerätemanagement-Systems. Es handelt sich um XML-Dateien, die den Aufbau des Datenflußgraphen beschreiben.
- Und schließlich natürlich noch Code-Dateien, die Anwendungscode in Form von systemunabhängigem JavaScript-Quellcode oder Java-Bytecode enthalten. Dieser Code wird über VRML-Script-Knoten in das Gesamtsystem eingebunden und behandelt alles, was nicht bereits über die vorgefertigten Softwaremodule des Renderingsystems und des Gerätemanagement-Systems abgedeckt werden kann.

Alle diese Dateien werden automatisch auf das mobile Gerät heruntergeladen, wenn der Anwender eine Anwendung startet, und in der AR-Laufzeitumgebung (sozusagen dem „AR-Browser“) ausgeführt. Dazu muß der Anwender wissen, welche Anwendungen an seinem gegenwärtigen Standpunkt zur Verfügung stehen. Es wäre wenig benutzerfreundlich, wenn der Anwender zum Start einer Anwendung eine kryptische URL eintippen müßte, anstelle dessen sollte er eine Liste der verfügbaren Anwendungen angezeigt bekommen. Die dafür notwendigen Techniken werden im 4. Kapitel beschrieben.

Nach dem Download wird die Anwendung automatisch gestartet und nimmt Verbindung zu im Netz vorhandenen Diensten auf. Diese Dienste können Datenbanken sein, die Anwendungsdaten zur Verfügung stellen, stationäre Trackingsysteme oder sogar Teilkomponenten der Anwendung, die auf einem stationären Rechner ablaufen. Das Rahmensystem muß dabei in der Lage sein, diese Dienste zu lokalisieren und möglichst transparent für die Anwendung mit diesen Diensten über das Netzwerk kommunizieren. Auch dies wird im 4. Kapitel genauer beschrieben.

Eine bislang in der Forschung wenig berücksichtigte Fragestellung ist, wie AR-Anwendungen eigentlich entwickelt werden, und wie sie getestet und wie von Fehlern bereinigt werden können (Debugging). Das hier vorgestellte Rahmensystem löst diese Fragestellung, indem die Hauptaufgabe bei der Anwendungsentwicklung im Kombinieren von Softwaremodulen und im Aufbau von Datenflußgraphen besteht. Die eigentliche Programmieraufgabe, das Entwickeln von Scripts, die Aufgaben übernehmen, die nicht durch vorgefertigte Softwaremodule abgedeckt werden, ist auf ein Minimum reduziert.

Zur Anwendungentwicklung und zum Debugging besitzt das Rahmensystem ein graphisches Userinterface (GUI), das es erlaubt, während der Laufzeit des Systems die verschiedenen Graphen beliebig zu modifizieren, d.h. Module hinzuzufügen oder zu entfernen, Parameter zu ändern, oder Verbindungen zwischen Modulen herzustellen oder zu trennen. Es handelt sich aber nicht um eine klassische GUI aus Fenstern und Bedienelementen, die nur auf dem Gerät verwendet werden kann, auf dem die AR-Anwendung läuft. Anstelle dessen wurde ein kleiner Web-Server in den Framework integriert, der diese GUI in Form von HTML-Seiten zur

Verfügung stellt. Es ist also möglich, von jedem Rechner aus, der eine Netzwerkverbindung zum mobilen Gerät besitzt, die Anwendung zu überwachen und zu modifizieren, indem einfach ein normaler Web-Browser verwendet wird. Die Details dieses Konzeptes werden im 6. Kapitel beschrieben.

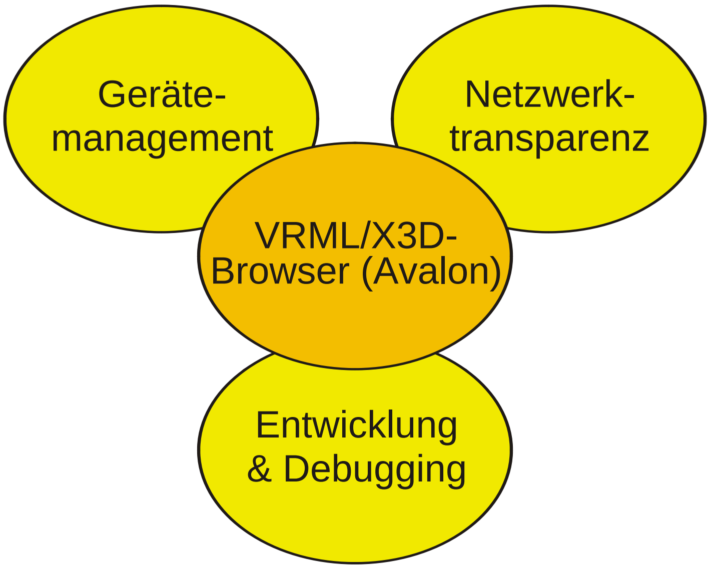


Abbildung 9: Bestandteile des Systems

Abbildung 9 zeigt die Systemarchitektur nochmal im Überblick. Das System basiert auf dem existierenden VR-System „Avalon“ [33], das bereits erfolgreich für VR-Anwendungen eingesetzt wird und VRML/X3D zur Beschreibung und Entwicklung der Anwendungen verwendet. Dieses System wird in dieser Arbeit für den Einsatz in mobilen AR-Systemen um ein neuartiges Gerätemanagement-System erweitert. Da mobile AR-Systeme besonders auf die Kommunikation mit anderen Systemen über Netzwerke angewiesen sind, wird um Systemkomponenten erweitert, die eine netzwerktransparente Anwendungsentwicklung ermöglichen. Darüber hinaus wird Avalon um eine Schnittstelle zum Entwickeln und Debuggen von AR-Anwendungen erweitert, die speziell auf die Anforderungen von mobilen Systemen ausgerichtet ist.

3 Gerätemanagement

In diesem Kapitel wird die Teilkomponente „Gerätemanagement“ beschrieben. Es werden zuerst die speziellen Anforderungen an ein Gerätemanagement-System identifiziert, das in einem mobilen AR-System eingesetzt werden soll. Danach wird ein Überblick über existierende Gerätemanagement-Systeme gegeben. Anschließend wird der Entwurf eines eigenen Gerätemanagement-Systems vorgestellt.

3.1 Anforderungen

Die Aufgabe des Gerätemanagement-Systems ist es, eine Schnittstelle zwischen der Anwendung und der Hardware, auf der die Anwendung läuft, bereitzustellen. Dabei lassen sich eine Reihe von Anforderungen an ein solches System identifizieren. Zum Teil sind diese Anforderungen Ergebnis der Architektur des in dieser Arbeit vorgestellen AR-Rahmensystem, andere sind das Ergebnis praktischer Erfahrung bei der Arbeit an VR- und AR-Anwendungen:

- Das Gerätemanagement muß beliebige Geräte behandeln können. Diese Anforderung erscheint zunächst trivial. Tatsächlich zeigt aber ein Blick auf existierende Gerätemanagement-Systeme (siehe weiter unten), daß dies nicht von allen Systemen erfüllt wird.

  Die klassische Herangehensweise bei der Einbindung von Eingabegeräten in graphische Systeme besteht darin, die Hardware auf eine Menge von Eingabeklassen abzubilden. Diese Eingabeklassen sind typischerweise:

  - „Locator“: Eingabe einer Position im Raum, z.B. durch ein Trackingsystem
  - „Pick“: Identifikation eines Bildelementes, z.B. durch Anklicken eines Objektes mit der Maus
  - „String“: Eingabe eines Textstrings, z.B. über ein Spracherkennungssystem oder eine Tastatur
  - „Valuator“: Eingabe eines Fließkommawertes, z.B. durch Drehregler, Schieberegler o.Ä.
  - „Choice“: Auswahl eines Menüeintrages

  Man erkennt leicht, daß sich eine Vielzahl von Geräten nicht ohne weiteres in dieses Klassifizierungssystem einordnen lassen. Wohin soll man z.B. einen Framegrabber einordnen, der Videobilder liefert? Ein Mikrophon, das Geräusche aufnimmt? Oder ein typisches VR-Eingabegerät wie den Datenhandschuh?

  Natürlich kann man die Menge der Eingabeklassen beliebig erweitern, aber das grundlegende Problem löst man dadurch nicht. Ein modernes Gerätemanagement-System sollte flexibel genug sein, um alle Typen von Eingabegeräten behandeln zu können. Jede Festlegung auf eine beschränkte Menge von Eingabeklassen wird früher oder später an seine Grenzen stoßen, denn es wird immer Geräte geben, die sich nicht in ein solches Schema pressen lassen. Es muß also ein Konzept gefunden werden, das nicht auf Eingabeklassen basiert.

- Das Gerätemanagement sollte nicht nur Eingabegeräte behandeln, sondern auch Ausgabegeräte: Eigentlich eine triviale Anforderung – es gibt natürlich nicht nur Geräte, die Daten an die VR-/AR-Anwendung liefern, sondern auch Geräte, die Daten von der Anwendung empfangen. Trotzdem behandeln viele gegenwärtig für VR-/AR-Anwendungen verfügbaren Gerätemanagement-Systeme Ausgabegeräte entweder gar nicht oder nur stiefmütterlich. Typische im VR-/AR-Bereich verwendete Ausgabegeräte sind z.B.:

  - Geräte, die ein haptisches Feedback liefern, wie z.B. der SensAble Technologies PHANTOM[3] oder Force-Feedback-Joysticks und -Gamepads. Gerade im Bereich AR, wo das Sichtfeld des Anwenders durch HMDs eingeschränkt ist, kann haptisches Feedback einen wichtigen Rückkanal vom System zum Anwender darstellen.
  - Audioausgabegeräte. Wie bei haptischen Ausgabegeräten gilt auch hier, daß akustisches Feedback einen wichtigen Rückkanal vom System zum Anwender darstellt.
  - Zusätzliche optische Ausgabegeräte neben dem HMD, wie z.B. Lämpchen, LEDs oder kleine LC-Displays.

- Unterstützung für eine weitgehende Vorverarbeitung von Daten innerhalb des Gerätemanagements. Häufig werden nicht die „rohen“ Daten eines Gerätes benötigt, sondern es ist eine weitere Verarbeitung der gelieferten Datenströme notwendig. Beispiele sind die Umwandlung von Positionsdaten, die von Trackingsystemen geliefert werden, in andere Koordinatensysteme, oder die Umwandlung von Farbbildern, die von einer Farbkamera geliefert werden, in Graustufenbilder. Solche typischen grundlegenden Verarbeitungsschritte sollten direkt vom Gerätemanagement durchgeführt werden können. Einige existierende Gerätemanagement-Systeme unterstützen dies z.B. durch „virtuelle“ Geräte, die Daten von anderen Geräten transformieren.

- Das Gerätemanagement sollte ein Umkonfigurieren während des laufenden Betriebs ermöglichen. D.h., es sollte möglich sein, Geräte bei laufender Anwendung hinzuzufügen, zu entfernen oder auszutauschen. Der Start einer komplexen VR-/AR-Anwendung ist häufig ein zeitfressender Vorgang. Kommunikationsverbindungen müssen hergestellt, diverse Geräte initialisiert und komplexe 3D-Szenen über das Netzwerk geladen werden. Vor diesem Hintergrund ist es äußerst ineffizient, wenn man bei jeder fehlgeschlagenen Initialisierung eines einzelnen Gerätes die gesamte Anwendung erneut starten muß. Anstelle dessen sollte es möglich sein, Fehlersituationen im laufenden Betrieb über das User-Interface des Gerätemanagement-Systems zu beheben.

- Das Gerätemanagement sollte netzwerktransparent sein, d.h. es sollte aus Sicht der Anwendung ohne Bedeutung sein, ob ein Gerät lokal an den Rechner angeschlossen ist, oder ob es an einen anderen Rechner im Netzwerk angeschlossen ist. Dieser Fall tritt bei VR-/AR-Systemen häufig auf. Gerade leistungsschwache mobile AR-Systeme sind auf eine unterstützende Infrastruktur angewiesen, die rechenaufwendige Arbeiten übernimmt und die Ergebnisse über das Netzwerk bereitstellt. Z.B. ist es nicht unüblich, daß die Positionsbestimmung eines mobilen AR-Systems nicht vom mobilen Gerät selbst durchgeführt wird, sondern von einem stationär installierten Trackingsystem, das die Trackingergebnisse dem mobilen Gerät übermittelt.

- Das Device-Management sollte effizient sein. Device-Management-Systeme kapseln den Zugriff auf Geräte, um eine einheitliche Schnittstelle zur Anwendung bereitzustellen. Dies erzeugt unvermeidlicherweise einen Overhead, verglichen mit dem direkten Zugriff auf ein Gerät. Gerade bei leistungsschwachen mobilen Geräten

---

[3] http://www.sensable.com/

ist es von entscheidender Bedeutung, diesen Overhead möglichst gering zu halten. Dies gilt insbesondere für Geräte, die umfangreiche Datenströme erzeugen, wie z.B. die bei AR-Anwendungen häufig eingesetzten Videokameras. Betrachtet man existierende Gerätemanagement-Systeme, so fällt auf, daß häufig nur Geräte mit vergleichsweise geringer Bandbreite unterstützt werden, wie z.B. Trackingsysteme. Videodatenströme müssen von der Anwendung selbst gehandhabt werden. Dieser Zustand ist nicht befriedigend, insbesondere vor dem Hintergrund, daß Videokameras ein Bestandteil praktischer jeder AR-Anwendung sind. Ein Gerätemanagement-System muß auch solche Geräte mit hoher Bandbreite unterstützen, und zwar mit einer dem direkten Gerätezugriff vergleichbaren Effizienz.

- Das Geräte-Management sollte Event-basiert sein. Viele gebräuchliche Gerätemanagement-Systeme stammen aus dem VR-Bereich, d.h. sie sind nicht für mobile, sondern für stationäre Systeme gedacht. Stationäre VR-Systeme arbeiten meist mit Rendering-Schleifen, d.h. das Bild wird ständig neu berechnet. Gerätemanagement-Systeme werden üblicherweise innerhalb dieser Renderingschleife aufgerufen, um die Latenz möglichst gering zu halten. Diese Herangehensweise ist bei mobilen Systemen nicht möglich, weil auf diesen Systemen nur eingeschränkte Ressourcen zur Verfügung stehen und insbesondere die Batterielaufzeit ein kritischer Punkt ist. Mobile Systeme müssen Event-basiert sein, d.h. es wird nur dann etwas gerendert, wenn es tatsächlich notwendig ist. Selbstverständlich folgt daraus, daß auch das Gerätemanagement Event-basiert sein muß.
- Das Gerätemanagement sollte einfach zu verwenden sein. Das Gerätemanagement stellt eine Art „Dienstleistung“ für den Anwendungsentwickler bereit. Er soll sich auf das Entwickeln seiner Anwendung konzentrieren können, ohne sich in die Niederungen der Hardwareprogrammierung begeben zu müssen. Von diesem Blickwinkel aus betrachtet ist das Gerätemanagement nur ein untergeordneter Bestandteil der Anwendung, der möglichst geräuschlos und unauffällig seine Dienste verrichten soll. Auf der anderen Seite ist das Gerätemanagement jedoch eine zentrale Komponente einer jeden Anwendung, da es der Anwendung den Zugriff auf Ein- und Ausgabegeräte und damit die Kommunikation mit seiner Umgebung ermöglicht. Das bedeutet aber auch, daß für alle verwendeten Geräte Gerätetreiber entwickelt und in das Gerätemanagement-System eingebunden werden müssen. Zwar können (und sollen) die Gerätetreiber zwischen verschiedenen Anwendungen ausgetauscht werden, trotzdem ist in der Praxis für neue Anwendungen ein gewisser Entwicklungsaufwand beim Gerätemanagement unvermeidlich. Dieser Entwicklungsaufwand sollte so minimal wie möglich sein.
- Und zu guter Letzt sollte das Gerätemanagement unabhängig von der Betriebsystem-Plattform sein, auf der die Anwendung bzw. die von der Anwendung verwendeten Geräte laufen.

Im folgenden Kapitel werden nun existierende Gerätemanagement-Systeme vorgestellt und vor dem Hintergrund der hier aufgestellten Anforderungen bewertet.

3.2 Übersicht und Vergleich existierender Gerätemanagement-Systeme

Gerätemanagement-Systeme sind ein wichtiger Bestandteil einer jeden VR-/AR-Anwendung. Ohne ein solches System ist es unmöglich, mit den virtuellen Welten zu interagieren. Daher ist im Laufe der Zeit eine Vielzahl von solchen Systemen entstanden. Alle diese Systeme hier vorzustellen ist unmöglich, daher werden hier nur solche Systeme betrachtet, die entweder

- frei verfügbar und weit verbreitet sind, oder
- besonders interessante Konzepte benutzen, die für den in dieser Arbeit beschriebenen Entwurf eines Gerätemanagement-Systems von besonderer Bedeutung waren.

3.2.1 VR Juggler[4]

VR Juggler [50] ist ein Framework für VR-Anwendungen, das an der Iowa State University entwickelt wird. Es ist in C++ geschrieben und läuft unter Windows und diversen Unix-Systemen. Neben anderen Komponenten enthält es auch ein Gerätemanagement-System, das den Zugriff auf Eingabegeräte erlaubt.

VR Juggler bildet Geräte auf logische Geräteklassen ab, unter anderem „Position“ für Tracker, die Position und Orientierung liefern, „Digital“ für Buttons von Joysticks oder ähnlichen Geräten, „Analog“ für analoge Joystick-Achsen, sowie „Glove“ für Datenhandschuhe. Für jede logische Geräteklasse gibt es eine entsprechende C++-Klasse. Der Anwendungsentwickler erzeugt die benötigten Instanzen der Geräteklassen und übergibt einen Identifier-String an eine „init“-Methode. Die Indentifier-Strings werden von der VR-Juggler-Konfiguration auf die vorhandenen Geräte abgebildet. Dabei unterstützt VR-Juggler Netzwerktransparenz, d.h. Geräte können an beliebige Rechner im Netzwerk angeschlossen werden. Geräte können im laufenden Betrieb ausgetauscht und neu gestartet werden. Neue Datenwerte erhält die Anwendung, indem sie eine „getData“-Methode aufrufen, die den zum jeweiligen Gerätetyp gehörigen Datentyp zurückliefert.

Das Gerätemanagement von VR Juggler ist ein typisches Beispiel für ein klassisches VR-Gerätemanagement. Es erlaubt eine klare Trennung zwischen Anwendungslogik und den verwendeten Geräten. Welche Geräte eine Anwendung konkret verwendet, wird über Konfigurationsdateien festgelegt. Dabei kann diese Konfiguration zur Laufzeit geändert werden.

Leider basiert VR Juggler auf dem Konzept der Geräteklassen, was wie oben beschrieben in der Praxis problematisch ist. Es gibt keinerlei Verarbeitung der Daten innerhalb des Systems, was das Austauschen von Geräten erschwert und es erforderlich macht, typische, immer wiederkehrende Verarbeitungsaufgaben wie Koordinatentransformationen innerhalb der Anwendung zu programmieren. Es werden nur klassische VR-Geräte unterstützt, aber keine typischen AR-Geräte wie Kameras. Problematisch ist auch, das VR Juggler auf Polling basiert, d.h. neue Datenwerte durch einen Methodenaufruf von der Anwendung geholt werden muß. Das ist zwar für VR-Systeme wie VR-Juggler in Ordnung, die auf einer Renderingschleife basieren, für mobile AR-Systeme ist das jedoch Ressourcenverschwendung.

3.2.2 OpenTracker[5]

OpenTracker [55] ist ein Device-Management-System, das an der TU Graz entwickelt wird. Es ist in C++ geschrieben und auf allen wichtigen Plattformen verfügbar.

Wie der Name schon andeutet, war OpenTracker ursprünglich nur auf das Verwalten von Trackern ausgerichtet. Das bedeutet, das nur Informationen über die Position und Orientierung von Gegenständen im Raum sowie der Status von möglicherweise vom Trackingsystem bereitgestellten Buttons übertragen wurde. Inzwischen wurde OpenTracker

---

[4] http://www.vrjuggler.org/

[5] http://www.studierstube.org/opentracker/

komplett überarbeitet und verwendet ein generisches Event-Konzept, das praktisch jede Form von Gerät unterstützt [56].

OpenTracker verwendet ein Datenflußkonzept, d.h. die von Geräten gelieferten Datenwerte durchlaufen einen Graphen. Gerätetreiber, die neue Datenwerte produzieren, sind Knoten in dem Graphen. Die Datenwerte durchlaufen dann diverse Filter-Knoten im Graphen, die die Datenwerte transformieren. Schließlich werden die Datenwerte an die Anwendung übergeben. Der genaue Aufbau des Graphen wird in einer XML-Konfigurationsdatei festgelegt, die beim Start des Systems eingelesen wird.

Eine Besonderheit des Systems ist es, daß verschiedene Datenströme gebündelt werden können – auf diese Weise können z.B. Positions- und Orientierungswerte von Videotrackingsystemen zusammen mit dem Bild, aus denen diese Informationen berechnet wurden, an die Anwendung weitergereicht werden.

OpenTracker erlaubt nicht nur den Zugriff auf Geräte, die an denselben Rechner angeschlossen sind, auf dem auch die Anwendung läuft, sondern auch auf Geräte, die an beliebige Rechner im Netzwerk angeschlossen sind. Es gibt spezielle Knoten, die in den Graphen eingefügt werden können und es erlauben, Daten von anderen Rechnern im Netzwerk zu empfangen.

OpenTracker erlaubt es, die Systemkonfiguration zur Laufzeit zu ändern [57][58].

OpenTracker verwendet eine Reihe von sehr interessanten Konzepten. Es basiert auf einem Datenflußgraphen, der eine sehr weitgehende Vorverarbeitung der von den Geräten gelieferten Datenwerte erlaubt. Die Struktur dieses Datenflußgraphen wird nicht im Code der Anwendung festgelegt, sondern in einer XML-Datei abgelegt und beim Start der Anwendung aus dieser Datei heraus aufgebaut. Der Datenflußgraph erlaubt es sehr elegant, Netzwerktransparenz in das System einzubauen, indem es spezielle Knoten zur Übertragung von Daten gibt.

3.2.3 VRPN[6]

Virtual-Reality Peripheral Network (VRPN) [59] ist ein Device-Management-System, das an der University of North Carolina entwickelt wird. Es ist in C++ geschrieben und läuft unter Windows und diversen Unix-Systemen.

VRPN bildet Geräte auf logische Geräteklassen ab. Für Eingabegeräte gibt es die Klassen „Tracker“ für Geräte, die Position und Orientierung liefern, „Button“ für digitale Eingabegeräte wie die Buttons von Joysticks und ähnlichen Geräten, „Analog“ für analoge Eingabegeräte wie Joystick-Achsen, sowie „Dial“ für Rotationswerte. Für Ausgabegeräte gibt es die Geräteklassen „Analog“ zur Ausgabe von analogen Werten, „Poser“ für die Ausgabe von Position und Orientierung, „Sound“ für Soundausgabegeräte, sowie „ForceDevice“ für Force-Feedback-Geräte.

Geräte können gleichzeitig Mitglieder von mehreren Geräteklassen sein, d.h. sie können über mehrere Interfaces angesprochen werden. So kann z.B. ein Trackingsystem, das Position und Orientierung liefert, aber auch einen Button besitzt, sowohl über das „Tracker“ als auch über das „Button“-Interface angesprochen werden. Joysticks werden sowohl über das „Analog“-Interface (für die Stellung der Joystick-Achsen) als auch über das „Button“-Interface (für die Feuerknöpfe) angesprochen.

---

[6] http://www.cs.unc.edu/Research/vrpn/

Darüber hinaus gibt es auch sogenannte „Layered Devices“, die nicht direkt Hardware repräsentieren, sondern auf anderen Geräten aufbauen und die von diesen gelieferten Daten transformieren. Ein Beispiel ist das „AnalogFly“-Gerät, das es erlaubt, Positionsdaten über Joysticks zu erzeugen, indem die analogen Ausschläge der Joystick-Achsen aufintegriert werden.

VRPN ist komplett netzwerktransparent, d.h. es ist für die Anwendung unerheblich, ob ein Gerät direkt mit dem Rechner verbunden ist, auf dem die Anwendung läuft, oder ob es mit einem Rechner im Netzwerk verbunden ist.

Um auf ein Gerät zugreifen zu können, muß die Anwendung eine neue Instanz der C++-Klasse erzeugen, die das Interface der gewünschten logischen Geräteklasse bereitstellt. Dem Konstruktor wird dabei ein String mitgegeben, der aus zwei Teilen besteht: Zum einem aus dem Namen des gewünschten Geräts, und zum anderen den Namen des Rechners, an den das Gerät angeschlossen ist. Auf diesem Rechner läuft ein VRPN-Server, der die an den Rechner angeschlossenen Geräte betreibt und ihre Datenwerte an die Clients verschickt. Dabei wird in der Server-Konfigurationsdatei festgelegt, unter welchem Namen ein Gerät von Clients angesprochen wird.

Nachdem die Anwendung das Geräteobjekt erzeugt hat, kann sie Callback-Routinen bei diesem Objekt registrieren, die aufgerufen werden, wenn ein neuer Wert vom jeweiligen Gerät zur Verfügung steht. Da VRPN keine eigenen Threads erzeugt, muß die Anwendung regelmäßig eine Methode aufrufen, die den aktuellen Zustand des Geräts abfragt und, falls erforderlich, die Callbacks mit den neuesten Werten aufruft.

VRPN erlaubt es nicht, während der Laufzeit Änderungen an der Systemkonfiguration vorzunehmen.

Zusammenfassend muß man feststellen, daß VRPN nicht alle oben aufgestellten Anforderungen an ein Gerätemanagementsystem erfüllt. Es ist zwar netzwerktransparent und plattformunabhängig. Eine Einschränkung ist allerdings, daß in der Anwendung festgelegt werden muß, auf welchem Rechner im Netzwerk ein Gerät zu finden ist. Dies ist umso unverständlicher, als das die Abbildung eines von der Anwendung angeforderten Geräts auf die vorhandene Hardware ansonsten über Konfigurationsdateien erfolgt. VRPN basiert auf dem Konzept der Geräteklassen, das wie oben dargelegt das System viel zu stark einschränkt. Zwar war den Entwicklern des Systems die Problematik der Geräteklassen wohl bewußt, da sie die Möglichkeit eingebaut haben, ein Gerät unter mehreren Geräteklassen anzusprechen. Dieses Konzept wirkt zwar auf den ersten Blick elegant, dient aber letztendlich nur dazu, die gröbsten Probleme einer grundsätzlich falschen Systemarchitektur zu beseitigen. Interessant ist das Konzept der „Layered Devices“, das es erlaubt, eine Vorverarbeitung von gelieferten Datenwerten innerhalb des Gerätemanagementsystems durchzuführen. Unschön ist die stiefmütterliche Behandlung von Ausgabegeräten und die Tatsache, daß das System nicht während der Laufzeit umkonfiguriert werden kann. Die Notwendigkeit, zum Auslesen aktueller Daten von den Geräten regelmäßig eine Methode aufzurufen stellt eine Form von „Polling“ dar, die auf den heutigen, Event-basierten Systemen nicht mehr zeitgemäß ist.

3.2.4 DIVERSE[7]

DIVERSE [60] ist ein Toolkit für verteilte VR-Anwendungen, das an der Virginia Tech entwickelt wird. Sein Fokus liegt auf der Anbindung von Simulationssystemen an immersive VR-Systeme. Es handelt sich um in C++ geschriebene Bibliotheken, die momentan nur unter Linux laufen.

---

[7] http://diverse-vr.org/

Das Grundkonzept von DIVERSE besteht aus dem sogenannten „Remote Shared Memory“. Softwarekomponenten, die auf verschiedenen Rechnern im Netzwerk laufen können, fordern vom System diese Speicherblöcke an. Identifiziert werden die Speicherblöcke über einen Namen. Das System stellt sicher, daß jeder Schreibzugriff einer Softwarekomponente auf Remote Shared Memory auf allen Rechnern im Netzwerk repliziert wird, auf denen das System läuft. Auf diese Weise können Softwarekomponenten auf verschiedenen Rechnern Daten austauschen, indem sie in Remote Shared Memory schreiben bzw. aus ihnen lesen. Dabei wird zugunsten einer höheren Verarbeitungsgeschwindigkeit darauf verzichtet, sicherzustellen, daß der Inhalt aller Speicherblöcke im Netz zu jeder Zeit kohärent ist.

Dieser Mechanismus wird auch genutzt, um den Zugriff auf Geräte im Netz zu ermöglichen. So gibt es sogenannte DTK-Services, die z.B. Positionswerte von Trackern auslesen und in einen Remote Shared Memory mit dem Namen „Head“ schreiben. Softwarekomponenten, die an der Kopfposition interessiert sind, fordern diesen Speicherblock vom System an und lesen die aktuelle Kopfposition wieder aus.

DIVERSE erfüllt zwar eine Reihe von Anforderungen, wie z.B. Netzwerktransparenz oder die Möglichkeit, das System zur Laufzeit zu modifizieren. Es verwendet auch nicht das Konzept der Geräteklassen – allerdings werden diese über die auf DIVERSE aufsetzende Renderingschicht DgiPf [61] wieder eingeführt. Auf der anderen Seite setzt auch dieses System wieder eine Renderingschleife voraus. Darüber hinaus ist es bislang nicht Plattform-unabhängig.

3.2.5 DEVAL/MORGAN

Die „Device Abstraction Layer“ (DEVAL) ist das Gerätemanagement-System des AR-Frameworks MORGAN [39][40], das bereits im 2. Kapitel vorgestellt wurde. DEVAL ist voll in das MORGAN CORBA-Komponentensystem integriert. Das System definiert eine Hierarchie von Geräte-Interfaces. An der Spitze der Hierarchie steht das generische „Device“-Interface, davon sind die Interfaces „Input Device“ für Eingabegeräte und „Output Device“ für Ausgabegeräte abgeleitet. Am Ende der Hierarchie befinden sich Interfaces für konkrete Gerätetypen, wie z.B. „3 DOF PositionTracker“ oder „Button“. DEVAL verwendet also das Konzept der Geräteklassen. Allerdings können konkrete Gerätetreiber von beliebig vielen Interfaces erben, um so mehrere logische Geräteklassen gleichzeitig abzudecken.

Da Gerätetreiber CORBA-Komponenten sind, ist der Zugriff auf Geräte netzwerktransparent. Das System kann zur Laufzeit umkonfiguriert werden. Softwarekomponenten können über ein Publish-Subscribe-Pattern Daten von Geräten empfangen, d.h. sie melden sich beim Gerätetreiber an und werden dann von ihm benachrichtigt, wenn neue Daten vorliegen – DEVAL ist also Event-basiert.

DEVAL kennt sogenannte „Adapter“ – Komponenten, die Daten von Geräten filtern, transformieren oder kombinieren und über ein anderes Interface anbieten können. Auf diese Weise wird de facto ein Datenflußgraph aufgebaut.

3.2.6 IDEAL

IDEAL [62][63][64] ist ein Gerätemanagement-System, das am Fraunhofer IGD entwickelt wurde. Es ist in C++ geschrieben und auf allen wichtigen Plattformen verfügbar.

IDEAL bildet Geräte auf logische Geräteklassen ab. Diese Geräteklassen sind „Button“ für digitale Eingabegeräte wie Tasten und Knöpfe, „Value“ für analoge Geräte wie Joystick-Achsen, „Choice“ für Geräte, die diskrete Werte liefern, „Location“ für 3D-Positionen, „Orientation“ für 3D-Orientierungen, „Space“ für eine Kombination aus Position und Orientierung, „Hand“ für Datenhandschuhe, sowie „Speech“ für Spracheingabe. IDEAL unterstützt nur Eingabegeräte. Zwar kann man über IDEAL als Ausgabegeräte auch Force-Feedback-Geräte und Sprachausgabe verwenden, diese sind aber nicht Bestandteil des eigentlichen Gerätekonzepts.

IDEAL ist komplett netzwerktransparent, d.h. es ist für die Anwendung nicht erkennbar, ob ein Gerät an denselben Rechner angeschlossen ist, auf dem auch die Anwendung läuft, oder ob das Gerät an einen beliebigen Rechner im Netzwerk angeschlossen ist.

Anwendungen greifen auf Geräte zu, indem sie Instanzen der C++-Klassen erzeugen, die die logischen Geräteklassen repräsentieren. Dabei wird dem Konstruktor ein String übergeben. Dieser String stellt einen Schlüssel dar, mit dessen Hilfe das Device-Management-System über eine Konfigurationsdatei herausfinden kann, welches Gerät an welchem Rechner konkret angesprochen werden soll.

Nachdem die Anwendung das Geräteobjekt erzeugt hat, kann sie Callback-Routinen bei diesem Objekt registrieren, die aufgerufen werden, wenn ein neuer Wert vom jeweiligen Gerät zur Verfügung steht. Da IDEAL keine eigenen Threads erzeugt, muß die Anwendung regelmäßig eine Methode aufrufen, die den aktuellen Zustand des Geräts abfragt und, falls erforderlich, die Callbacks mit den neuesten Werten aufruft.

IDEAL erlaubt es, während der Laufzeit Änderungen an der Systemkonfiguration vorzunehmen. Es gibt ein integriertes User-Interface, das es erlaubt, Geräte zu konfigurieren und zu kalibrieren.

Zusammenfassend kann man sagen, daß IDEAL zwar eine Reihe von interessanten Eigenschaften besitzt, insbesondere die Netzwerktransparenz, die Plattformunabhängigkeit, sowie die Fähigkeit, während der Laufzeit Veränderungen an der Konfiguration vorzunehmen. Leider basiert aber auch IDEAL auf dem Konzept der Geräteklassen, das in der Praxis problematisch ist. Darüber hinaus werden Ausgabegeräte de facto nicht unterstützt. Problematisch ist erneut auch bei diesem Gerätemanagement-System die Tatsache, daß regelmäßig eine Funktion des Gerätemanagementsystems aufgerufen werden muß, um neue Gerätedaten zu erhalten. Diese Form des „Polling“ ist in heutigen, Event-basierten Systemen nicht mehr zeitgemäß.

3.2.7 Microsoft DirectX

Microsoft DirectX fällt hier ein wenig aus der Reihe, da es im eigentlichen Sinne kein Gerätemanagement-System ist. DirectX ist eine API der Microsoft-Betriebssystemfamilie Windows, die einen direkten Zugriff auf die Hardware des Systems erlaubt. DirectX wird häufig fälschlicherweise für eine 3D-Rendering-API gehalten und mit OpenGL verglichen. DirectX besteht aber aus einer ganzen Reihe von Teilkomponenten, von denen das 3D-Rendering nur eine ist (Direct3D). Zwei der Komponenten, nämlich DirectInput und DirectShow, sollen hier näher betrachtet werden, weil sie den Zugriff auf bestimmte Geräteklassen erlauben und dabei Konzepte verwenden, die auch im Kontext eines VR-/AR-Gerätemanagement-Systems interessant sind.

DirectInput erlaubt den Zugriff auf sogenannte „Human Interface Devices“, also Geräte wie Tastaturen, Mäuse und Joysticks. Die Anwendung kann abfragen, welche Geräte im System vorhanden sind, und den Zustand dieser Geräte auf unterschiedliche Weise abfragen: Zum einen durch Polling, zum anderen Event-basiert. Interessanterweise kennt DirectInput keine Geräteklassen. Für DirectInput besteht ein Gerät nur aus n Boolean-Werten (den Tasten oder Knöpfen) und m scalaren Werten (den Achsen). So besteht z.B. eine Wheel-Maus aus drei Boolean-Werten (linke und rechte Maustaste sowie das Wheel) sowie drei Achsen (die Bewegung der Maus in X- und Y-Richtung sowie die Drehung des Wheels). Eine Standard-PC-Tastatur (deutsches Layout) besteht aus 105 Boolean-Werten (den 105 Tasten). Ein

Joystick besteht aus zwei Achsen (X- und Y-Richtung des Knüppels) und einem Boolean-Wert (dem Feuerknopf).

DirectShow stellt eine Schnittstelle zum Verwalten von Audio- und Video-Datenströmen dar. Hauptsächlich dafür gedacht, Videos abzuspielen, besitzt es aber auch Funktionen, um Anwendungen den Zugriff auf Framegrabber, Webcams und Soundkarten zu ermöglichen. Dabei wird konsequent auf das Konzept des Datenflußgraphen gesetzt.

DirectShow stellt dem Anwendungsentwickler eine Reihe von Knoten zur Verfügung. Jeder Knoten erfüllt eine eng umrissene Aufgabe. So gibt es Knoten, die Video- oder Audiodaten von Framegrabber- oder Soundkarten digitalisieren, Multiplexer- und Demultiplexer-Knoten, die Video- und Audiokanäle zu gemultiplexten MPEG-Datenströmen zusammenfassen oder aus solchen Datenströmen extrahieren, Kompressions- und Dekompressionsknoten, die Video- und Audiodaten komprimieren und dekomprimieren, sowie Knoten, die Video- und Audiodaten auf dem Bildschirm darstellen bzw. über eine Soundkarte abspielen. Darüber hinaus gibt es noch eine Vielzahl von weiteren Knotentypen.

Knoten kommunizieren untereinander über Outslots und Inslots. Jeder Slot hat einen Namen sowie einen Typ, der angibt, welche Daten über den Slot verschickt oder empfangen werden. Outslots und Inslots von gleichem Typ können miteinander verbunden werden, um Daten miteinander auszutauschen.

Die Menge der verfügbaren Knoten kann erweitert werden. Dies geschieht z.B., wenn man Treiber für Multimedia-Geräte installiert. Jedes Hardware-Gerät wird im DirectShow-Datenflußgraphen durch einen eigenen Knoten repräsentiert, um es in die Anwendung einbinden zu können. Darüber hinaus kann man neue Knoten durch Plugins hinzufügen, um DirectShow z.B. um neue Kompressionsverfahren zu erweitern. Und zu guter Letzt kann jede Anwendung noch eigene, anwendungsspezifische Knoten verwenden. Dies ist z.B. erforderlich, um die vom Datenflußgraphen produzierten Datenströme innerhalb der Anwendung zu verwenden.

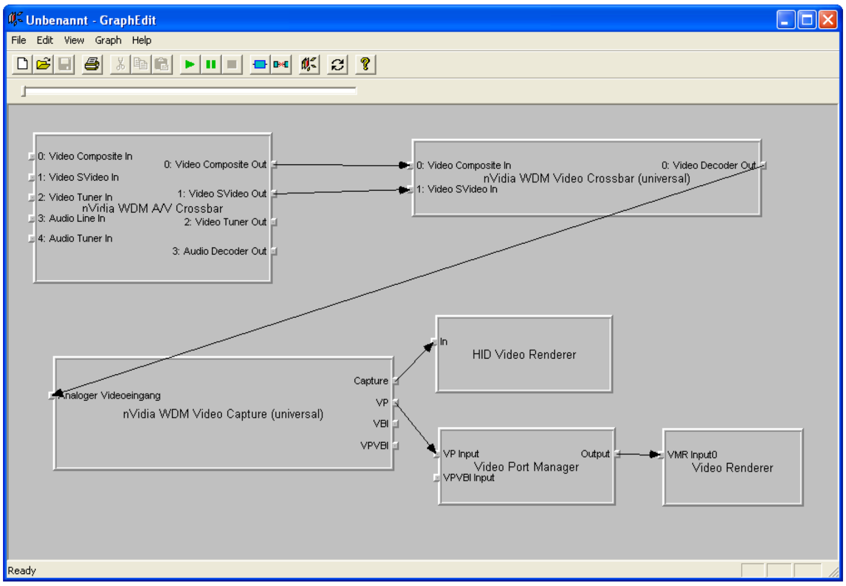


Abbildung 10: Screenshot der GraphEdit-Anwendung mit einem DirectShow-Filtergraphen

Der DirectShow-Datenflußgraph wird durch entsprechende Anweisungen im Code der Anwendung erzeugt. Leider ist es nicht möglich, die Struktur des Graphen in Form einer Datei abzulegen und beim Programmstart aus dieser Datei zu rekonstruieren. Interessanterweise gibt es von Microsoft einen graphischen Editor („GraphEdit“), der es erlaubt, DirectShow-Graphen interaktiv am Bildschirm zu erzeugen (siehe Abbildung 10).

Man kann aus einer Liste der aktuell im System verfügbaren Knoten den Graphen zusammenstellen und die Outslots mit den Inslots verbinden. Dann kann man den Graphen starten und das Ergebnis überprüfen. Darüber hinaus kann man auch DirectShow-Datenflußgraphen von gerade auf dem System laufenden Anwendungen darstellen lassen, wenn diese die dazu notwendigen Schnittstellen unterstützen. In diesem Fall ist es sogar mit gewissen Einschränkungen möglich, den Graphen im laufenden Programm zu modifizieren. So stellt das GraphEdit-Programm in Abbildung 10 z.B. den Graphen eines Programms dar, das Videobilder vom Videoeingang einer nVidia-Graphikkarte digitalisiert. Wie man sieht, können solche Graphen selbst bei einer scheinbar so einfachen Aufgabe sehr komplex werden. Die mit dem GraphEdit-Programm erzeugten Graphen können aber nicht gespeichert und in eigene Anwendungen integriert werden. Das Programm dient nur dazu, mit Graphen herumzuexperimentieren oder Graphen von laufenden Anwendungen zu debuggen.

DirectInput und DirectShow sind beide nicht netzwerktransparent, d.h. man kann nur auf Geräte zugreifen, die direkt an das System angeschlossen sind, auf dem die Anwendung läuft. Darüber hinaus sind beide DirectX-Komponenten selbstverständlich nur für Windows-Betriebssysteme verfügbar.

Zusammenfassend kann man sagen, daß DirectInput und DirectShow nur Teilbereiche eines Gerätemanagement-Systems abdecken. Sie sind nicht netzwerktransparent und nicht systemunabhängig, können nicht durch Dateien, sondern nur im Code der Anwendung konfiguriert werden, und unterstützen nur eine sehr beschränkte Auswahl an Gerätetypen (Eingabegeräte wie Joysticks, Mäuse, und Tastaturen sowie Multimediageräte wie Web-Cams, Framegrabber und Soundkarten). Nichtsdestotrotz findet man hier einige interessante Konzepte: Es gibt keine Geräteklassen und keine Trennung zwischen Ein- und Ausgabegeräten mehr. Jedes Gerät ist eine Einheit (ein Knoten), der beliebig viele ein- und ausgehende Datenströme besitzt. Im Falle von DirectShow erlaubt das Konzept des Datenflußgraphen eine sehr weitgehende Vorverarbeitung der von den Geräten gelieferten Daten.

3.2.8 Bewertung der Gerätemanagement-Systeme

Der Überblick über existierende Gerätemanagement-Systeme zeigt, daß jedes System seine spezifischen Stärken und Schwächen hat. Es ist überraschend, daß sich in einem so wichtigen Bereich der graphischen Datenverarbeitung, in dem eine solche Vielzahl von Lösungen existiert, noch kein Standardvorgehen durchgesetzt hat. Insbesondere das Konzept der Geräteklassen erweist sich als schädlich – obwohl seine Einschränkungen offensichtlich und vielen Entwicklern offenbar auch bewußt sind, bildet dieses Konzept die Grundlage für viele im VR-/AR-Bereich existierenden Gerätemanagement-Systeme.

Die hier vorgestellten Gerätemanagement-Systeme sind nicht etwa schlecht entworfen – ganz im Gegenteil. Sie decken nur häufig nicht alle Anforderungen ab, die an ein solches System im Kontext eines AR-Rahmensystems gestellt werden müssen, sondern nur Teilbereiche. Dabei fallen vor allem folgende Schwachpunkte auf:

- Wie bereits oben erwähnt, basieren viele im VR-/AR-Bereich existierenden Gerätemanagement-Systeme auf dem Konzept der Geräteklassen. Das Problem dabei ist, daß sich viele Geräte nicht ohne weiteres eindeutig auf eine Geräteklasse abbilden lassen.

- Ausgabegeräte werden meist stiefmütterlich behandelt. Dafür gibt es nicht wirklich eine Begründung – Ausgabegeräte sind genauso Bestandteil von VR- und AR-Systemen wie Eingabegeräte. Häufig hat man den Eindruck, das Ausgabegeräte nicht Bestandteil der Architektur eines Gerätemanagement-Systems sind, sondern nur „irgendwie“ nachträglich zum System hinzugefügt wurden.

- Häufig erlauben die Gerätemanagement-Systeme keine Vorverarbeitung der Gerätedaten, sondern liefern nur die „rohen“ Daten, wie sie vom Gerät geliefert werden, bei der Anwendung ab. Das ist umso erstaunlicher, als daß auf diese Weise die Anwendung häufig spezifisch an das verwendete Gerät angepaßt werden muß und ein späterer Wechsel auf ein anderes Gerät nicht ohne Änderungen an der Anwendung selbst möglich ist. Dabei ist es doch eigentlich eine Kernaufgabe des Gerätemanagements, alle gerätespezifischen Details vor der Anwendung zu verbergen.
- Auffällig ist auch, daß die meisten Systeme sich auf Geräte beschränken, die nur eine vergleichsweise geringe Datenrate bzw. Bandbreite haben, also z.B. Eingabegeräte wie Joysticks oder Trackingsysteme. Dagegen werden Framegrabberkarten, Web-Cams und Soundkarten von den klassischen Gerätemanagementsystemen selten unterstützt. Selbstverständlich stellen solche Geräte viel größere Anforderungen an ein Gerätemanagementsystem – im Gegensatz zum Betrieb von Joysticks und Trackingsystemen muß die Architektur des Systems hier schon sehr sorgfältig geplant werden, damit auch große Datenmengen ohne großen Performanzverlust durch das System geschleust werden können.
- Außerdem fällt auf, das viele der klassischen Gerätemanagementsysteme, die aus dem VR-Bereich stammen, die Geräte „pollen“, d.h. es muß von der Anwendung regelmäßig eine Methode des Systems aufgerufen werden, um neue Datenwerte zu erhalten. Ein solches Verfahren ist aber auf modernen, Event-basierten Systemen nicht mehr zeitgemäß. Es erklärt sich aus der Tatsache, daß viele VR-Systeme auf Ressourcen wie Rechenzeit keine Rücksicht nehmen müssen, weil sie rein stationär eingesetzt werden. Solche Systeme arbeiten mit einer Rendering-Schleife, d.h. immer dann, wenn ein Bild fertig gezeichnet wurde, wird mit dem Zeichnen des nächsten Bildes begonnen – völlig unabhängig davon, ob sich in der Zwischenzeit überhaupt irgendetwas an der Szene geändert hat. In eine solche Rendering-Schleife fügt sich das Abfragen von Geräten natürlich nahtlos ein. Bei einem mobilen AR-System kann man jedoch nicht so verschwenderisch mit Rechenzeit umgehen – hier gibt es keine Rendering-Schleife und damit kein „Polling“, sondern Geräte müssen sich selbst (z.B. über Events) melden, wenn sie neue Daten haben oder sich ihr Zustand ändert.

Nichtsdestotrotz gibt eine Reihe von interessanten Ideen, die von den hier vorgestellten Gerätemanagement-Systemen umgesetzt werden. Im folgenden Abschnitt wird nun ein eigenes Gerätemanagement-System vorgestellt, das nicht die obigen Schwachpunkte besitzt. Dabei wird nicht alles anders gemacht – anstelle dessen wird das Beste der existierenden Gerätemanagement-Systeme in einem neuen System zusammengefaßt und, wo erforderlich, durch neue Konzepte erweitert.

3.3 Konzept eines neuartigen Gerätemanagement-Systems

Wie sieht nun ein neuartiges Gerätemanagement-System aus, das nicht die oben genannten Schwachpunkte besitzt? Für einen Teil der Schwachpunkte findet man bereits unter den existierenden Systemen Lösungen:

- DirectInput und DirectShow zeigen eine Alternative zum Konzept der Geräteklassen. Ein Gerät ist in diesen Systemen eine Einheit, die beliebig viele Datenströme erzeugen und empfangen kann. Damit kann man problemlos jedes Gerät darstellen – ein Joystick mit zwei analogen Achsen und drei Buttons erzeugt z.B. zwei Datenströme aus Fließkommazahlen und drei Datenströme aus Boolean-Werten. Darüber hinaus schließt dieses Konzept nicht nur Eingabegeräte, sondern völlig gleichberechtigt auch Ausgabegeräte ein.

- OpenTracker und DirectShow zeigen interessante Lösungen, um innerhalb des Gerätemanagement-System Daten zu verarbeiten. Beide verwenden dazu das Konzept des Datenflußgraphen. Diese Graphen bestehen aus Knoten, die über Kanten untereinander verbunden sind und miteinander kommunizieren. Jeder Knoten hat eine eng umrissene Teilaufgabe in dem System – einige sind Treiber für Eingabegeräte und erzeugen Datenströme, andere transformieren Datenströme, und wieder andere sind Treiber für Ausgabegeräte, die Datenströme konsumieren. Diese Datenflußgraphen können beliebig komplex sein und auf diese Weise eine weitgehende selbständige Verarbeitung von Daten innerhalb des Gerätemanagement-Systems vornehmen.

Das hier vorgestellte neuartige Gerätemanagement-System greift diese Lösungen auf. Es basiert ebenfalls auf einem Datenflußgraphen. Das Gerätemanagement-System stellt dem Anwendungsentwickler eine Menge von vorgefertigten Knoten zur Verfügung – Treiber für Ein- und Ausgabegeräte sowie Knoten, die Daten transformieren. Knoten besitzen sogenannte „Outslots“, über die sie Datenwerte verschicken können, und „Inslots“, über die sie Datenwerte empfangen können.

Der Typ der Datenwerte ist völlig beliebig – es kann sich um die Grunddatentypen der jeweils verwendeten Programmiersprache handeln (z.B. Float- oder Integer-Werte), oder um zusammengesetzte Datentypen wie z.B. Vektoren und Matrizen. Das Gerätemanagement stellt für die am häufigsten verwendeten zusammengesetzten Datentypen bereits fertige Implementierungen zur Verfügung.

Für jedes Gerät muß ein Knoten mit einer spezifischen Kombination von Out- und Inslots implementiert werden. Da den Kombinationsvarianten keine Grenzen gesetzt sind, gibt es im Gegensatz zu dem Konzept der Geräteklassen kein Problem, auch die exotischste Hardware innerhalb des Gerätemanagement-Systems abzubilden. Die Anzahl der Out- und Inslots kann auch variieren, so z.B. beim „Joystick“-Knoten, der Joysticks und Gamepads betreibt. Je nachdem, mit wievielen Achsen und Knöpfen das jeweils verwendete Gerät ausgestattet ist, verändert sich auch die Anzahl der Outslots, die die jeweilige Instanz des Joystick-Knotens im Datenflußgraphen besitzt.

Outslots und Inslots werden über Routes miteinander verbunden. Slots sind wie Routes immer Bestandteil eines Namespaces und besitzen innerhalb eines solchen Namen. Dieser Name wird von Routes benutzt, um die Slots zu identifizieren, die von ihnen verbunden werden. Selbstverständlich können nur Outslots und Inslots miteinander verbunden werden, die den gleichen Datentypen verwenden.

Diese einzelnen Bestandteile des Datenflußgraphen werden in den folgenden Abschnitten im Detail beschrieben. Dabei werden die Bestandteile in ein Lowlevel-Interface und ein Highlevel-Interface aufgeteilt. Der Hintergrund dieser Aufteilung ist ein Problem beim Einsatz von Datenflußgraphen: Nämlich die Frage, wie die Anwendung letztendlich an die Daten im Graphen herankommt bzw. selber Daten in den Graphen einschleusen kann. Streng genommen können ja nur Knoten im Datenflußgraphen miteinander kommunizieren. Das hat zur Folge, daß normalerweise die Anwendung einen speziellen Knoten bereitstellen muß, um Daten vom Graphen zu empfangen oder in den Graphen zu senden. Wer schon mal selber einen solchen Knoten z.B. für DirectShow programmiert hat, weiß, was für ein Overhead damit verbunden ist. Dies widerspricht der Anforderung, daß das Gerätemanagement einfach zu verwenden sein soll. Das ist umso schlimmer, als daß AR-Systeme ja häufig nicht „aus einer Hand“ stammen, sondern von mehreren Projektpartnern entwickelt werden, die jeweils ihre eigenen Softwarekomponenten in das System integrieren. Diese Integration wird dann problemlos und erfolgreich sein, wenn die Projektpartner nur geringe Änderungen an ihren Komponenten vornehmen müssen. Der Aufwand, Komponenten in eigene Knoten zu verpacken, wird dagegen meist zu hoch sein.

Aus diesen Gründen können Outslots und Inslots auch unabhängig von Knoten existieren. Sie sind zusammen mit Namespaces und Routes Bestandteil des Lowlevel-Interfaces. Um Daten aus dem Datenflußgraphen zu empfangen bzw. Daten in den Graphen zu senden, muß die Anwendung keine anwendungsspezifischen Knoten mehr enthalten – anstelle dessen erzeugt der Entwickler einfach Outslots und Inslots, die über Routes mit den Outslots und Inslots des Graphen verbunden werden.

Das Highlevel-Interface besteht dagegen aus den Knoten des Datenflußgraphen. Hier findet man die diversen Komponenten, aus denen der Anwendungsentwickler den Graphen zusammensetzt. Dieses „Zusammensetzen" kann zwar auch über den Anwendungscode geschehen, aber dies ist nicht der optimale Weg, weil so jede Änderung der Gerätekonfiguration eine Änderung der Anwendung erfordert. Anstelle dessen ist vorgesehen, daß der Entwickler den Datenflußgraphen über das in Kapitel 6 beschriebene integrierte User-Interface aufbaut. Die Struktur des Graphen, d.h. die verwendeten Knoten sowie die Verbindungen zwischen ihnen, werden dann in einer Datei gespeichert. Beim Start der Anwendung wird diese Datei wieder eingelesen und die Struktur des Graphen von der Laufzeitumgebung rekonstruiert. Dieses Speichern und Rekonstruieren des Graphen ist ebenfalls Bestandteil des Highlevel-Interfaces.

3.3.1 Lowlevel-Interface

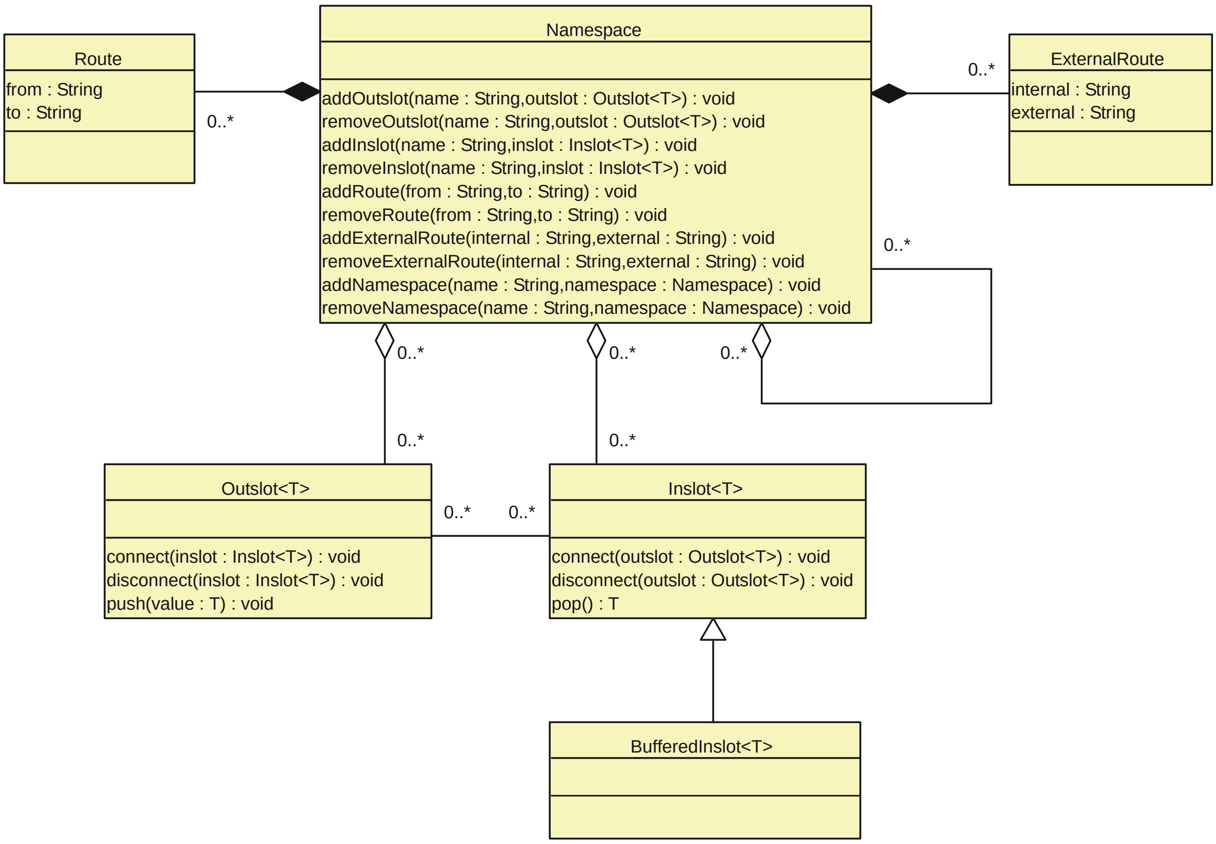


Abbildung 11: UML-Diagramm des Lowlevel-Interface

Das Lowlevel-Interface stellt die Basis des Gerätemanagement-Systems dar. Prinzipiell kann man es als die Kommunikationsschicht des Systems betrachten. Es besteht aus den Inslots und Outslots, Namespaces und Routes. Geräte kommen auf dieser Ebene des Systems nicht vor – hier geht es nur um die Übertragung von Datenströmen zwischen verschiedenen Softwarekomponenten. Das Lowlevel-Interface stellt auch die Schnittstelle zur Anwendung dar sowie zu anderen Teilkomponenten des Systems, die zwar über das Gerätemanagement mit der Anwendung kommunizieren, aber selbst keine Knoten und damit kein Teil des Datenflußgraphen sind.

3.3.1.1 Inslots und Outslots

Die Grundlage des Gerätemanagement-Systems bilden die sogenannten Inslots und Outslots. Sie stellen die Basis dar, auf der verschiedene Softwarekomponenten miteinander kommunizieren können. Outslots werden verwendet, um Daten an andere Softwarekomponenten zu verschicken, und Inslots werden verwendet, um Daten von anderen Softwarekomponenten zu empfangen.

Inslots und Outslots sind typisiert, d.h. es muß beim Erzeugen eines Slots angegeben werden, welcher Datentyp über ihn versendet oder empfangen werden kann. Die Typisierung geschieht unter C++ über Template-Parameter, und unter Java durch die Übergabe eines Klassen-Objekts, das Meta-Informationen über den Datentypen enthält. Prinzipiell können hier beliebige C++- oder Java-Datentypen verwendet werden, also insbesondere auch anwendungsspezifische, selbst definierte Datentypen. Um aber die Interoperabilität verschiedener Softwarekomponenten zu gewährleisten, wurde eine Reihe von Standarddatentypen spezifiziert. Diese Datentypen orientieren sich eng an den im VRML-Standard spezifizierten Datentypen. Wo möglich, wurde auf Datentypen zurückgegriffen, die von den jeweiligen Programmiersprachen direkt unterstützt werden. Wo das nicht möglich war (C++ und Java besitzen z.B. keine Datentypen für Vektoren, Rotationen und Matrizen), wurden im Gerätemanagement-System Datentypen spezifiziert. Diese Datentypen sind nicht dazu gedacht, innerhalb der jeweiligen Anwendung verwendet zu werden – so enthält die Vektor-Klasse keinerlei Funktionalität, um Vektorarithmetik durchzuführen. Sie dienen nur als Container, um die Daten auf standardisierte Weise zwischen verschiedenen Softwarekomponenten auszutauschen.

Outslots und Inslots können zur Laufzeit beliebig miteinander verbunden werden, wenn sie den gleichen Datentyp verwenden. Sie können zur Laufzeit auch wieder voneinander getrennt werden. Dieses Verbinden und Trennen kann zwar manuell über Methoden der Outslot- und Inslot-Klassen geschehen, in der Praxis werden Outslots und Inslots jedoch automatisch über Routes miteinander verbunden. Wie das genau geschieht, wird im folgenden Abschnitt beschrieben.

Wichtig ist, daß Outslots und Inslots auch (völlig transparent für die Anwendung) von anderen Anwendungen, die im Netzwerk laufen, importiert werden können. D.h. es können auch Verbindungen über Gerätegrenzen hinweg erzeugt werden, ohne daß die Anwendung dafür speziell modifiziert werden müßte. Mehr dazu im 4. Kapitel.

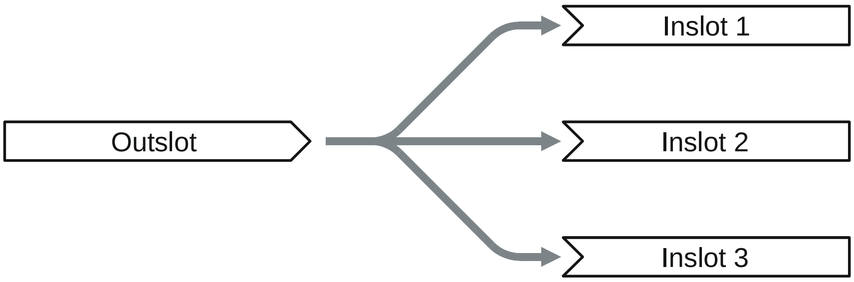


Abbildung 12: Outslots können mit beliebig vielen Inslots verbunden werden. Wird ein Datenwert in den Outslot geschrieben, erhalten alle Inslots eine Kopie dieses Datenwerts

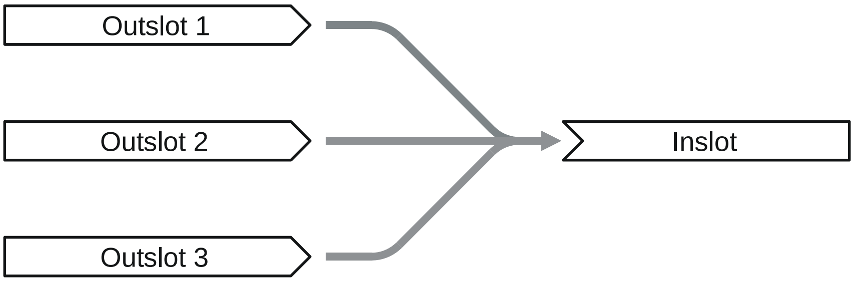


Abbildung 13: Inslots können mit beliebig vielen Outslots verbunden werden. Der Inslot erhält alle Datenwerte, die in die Outslots geschrieben werden – es ist jedoch nicht möglich festzustellen, von welchem Outslot ein empfangener Datenwert stammt

Die Verbindungen sind n-zu-n-Verbindungen, d.h. ein Outslot kann mit beliebig vielen Inslots (oder auch gar keinem) verbunden sein, und ein Inslot kann mit beliebig vielen Outslots (oder auch gar keinem) verbunden sein. Solche n-zu-n-Verbindungen sind in der Praxis durchaus nicht ungewöhnlich:

- Es könnte in einem Museum nützlich sein, Nachrichten an alle gerade im Betrieb befindlichen mobilen AR-Geräte zu senden, um z.B. die Besucher darauf hinzuweisen, daß das Museum in Kürze schließt o.ä. Zu diesem Zweck könnte es einen Outslot geben, der den Datentyp „SFString“ verwendet, um Nachrichten zu verschicken. Die Anwendungssoftware auf den mobilen Geräten verwendet dagegen einen Inslot vom Datentyp „SFString“, um die Nachrichten zu empfangen und auf dem Bildschirm des Besuchers darzustellen. Wir haben also die Situation, daß ein Outslot mit n Inslots verbunden ist.
- Es könnte für die Betreiber eines Museums von Interesse sein, jederzeit herausfinden zu können, wo im Gebäude sich die mobilen AR-Geräte befinden. Dazu könnten die Anwendungen auf den mobilen Geräten einen Outslot verwenden, der einen speziellen Datentyp verschickt, der eine Geräte-ID mit einer Positionsangabe kombiniert. Auf einem zentralen Server gibt es dagegen einen Inslot, der alle Positionsangaben empfängt und in einer Datenbank ablegt. Wir haben also die Situation, daß ein Inslot mit n Outslots verbunden ist.
- Zu guter Letzt könnte es auch gewünscht sein, daß die Anwender der mobilen AR-Geräte untereinander „chatten“ können, d.h. Nachrichten untereinander austauschen können. Dazu besitzt jedes mobile Gerät einen SFString-Outslot, um Text-Nachrichten an alle anderen Geräte zu verschicken, und einen SFString-Inslot, um Text-Nachrichten von allen anderen Geräten zu empfangen. Wir haben also die Situation, daß n Outslots mit n Inslots verbunden sind.

Outslots werden verwendet, um Daten an andere Softwarekomponenten zu verschicken. Dazu besitzen sie eine „push“-Methode, mit der Anwendungen jeweils ein neues Datenelement an den Outslot übergeben. Der Outslot reicht das Datenelement an alle Inslots weiter, die zu diesem Zeitpunkt mit ihm verbunden sind. Dabei kann das Datenelement von der Anwendung mit einem Zeitstempel versehen werden, oder es erhält automatisch vom Outslot einen Zeitstempel (die aktuelle Systemzeit).

Das Datenelement wird bei der Übertragung in einen Inslot kopiert, d.h. jeder mit einem Outslot verbundene Inslot erhält seine eigene Kopie. Das ist für Basisdatentypen die effizienteste Form der Übertragung. Bei größeren Datenmengen, wie z.B. einem kompletten Videobild, wäre dieser Kopiervorgang extrem ineffizient. Aus diesem Grund stellt das Gerätemanagement für solche Datentypen einen thread-sicheren Smart-Pointer, d.h. es wird nur noch der Zeiger auf ein Datenelement kopiert. Ein Reference-Counter stellt sicher, das der von dem Datenelement belegte Speicher frei gegeben wird, wenn die letzte Referenz auf das Element gelöscht wird. Genaueres dazu im 5. Kapitel.

Das jeweils letzte versendete Datenelement wird automatisch im Outslot zwischengespeichert und kann von der Anwendung abgefragt werden. Dies ist aus zwei Gründen nützlich:

1. Es dient Optimierungszwecken. Viele Geräte liefern, wenn sich ein Datenelement ändert, nicht nur das geänderte Datenelement, sondern alle verfügbaren Datenelemente. Wenn z.B. bei einem Gamepad mit 10 Buttons ein Button gedrückt wird, wird eine Bitmaske mit dem Zustand aller 10 Buttons geliefert. Es wäre jetzt nicht sehr ökonomisch, jedesmal den Zustand aller 10 Buttons über die jeweiligen Outslots zu versenden. Anstelle dessen kann die Anwendung den gegenwärtigen Zustand des Buttons mit dem zuletzt über den zugehörigen Outslot versendeten Wert vergleichen und nur geänderte Werte versenden. Diese Optimierung ist insbesondere dann wichtig, wenn die Daten über Netzwerke verschickt werden, deren Übertragungsbandbreite beschränkt ist.
2. Das zwischengespeicherte Datenelement wird automatisch vom Outslot an jeden Inslot verschickt, der sich neu mit dem Outslot verbindet. Dies ist notwendig, um den jeweils aktuellen Zustand einer neu gestarteten Softwarekomponente mitzuteilen, insbesondere wenn zu Optimierungszwecken wie oben beschrieben nur Zustandsänderungen vom Gerätemanagement-System übertragen werden.

Anwendungen können über eine Methode des Outslots abfragen, ob der Outslot mit mindestens einem Inslot verbunden ist. Auch dies dient Optimierungszwecken, weil auf diese Weise u.U. aufwendige Berechnungen von Datenelementen vermieden werden können, für die sich im Moment ohnehin keine andere Softwarekomponente interessiert. Anwendungen können aber nicht abfragen, mit wie vielen Inslots ein Outslot verbunden ist, und welche Inslots dies konkret sind.

Inslots werden verwendet, um Daten von anderen Softwarekomponenten zu empfangen. Dazu besitzen sie eine „pop“-Methode, um jeweils ein neues Datenelement vom Inslot zu erhalten. Diese Methode blockiert, bis ein neues Datenelement verfügbar ist. Falls das nicht erwünscht ist (beim sogenannten „Polling“), kann auch vor dem Aufruf der Methode abgefragt werden, ob ein neues Datenelement verfügbar ist.

Anwendungen können über eine Methode des Inslots abfragen, ob der Inslot mit mindestens einem Outslot verbunden ist. Wie bei den Outslots kann dies Optimierungszwecken dienen – z.B. könnte eine Softwarekomponente, die eingehende Sampledaten auf einer Soundkarte abspielt, die Soundhardware nur belegen, wenn tatsächlich Outslots vorhanden sind, die Sampledaten liefern. Analog zu den Outslots können Anwendungen aber nicht abfragen, mit wie vielen Outslots ein Inslot verbunden ist, und welche Outslots dies konkret sind.

Standardmäßig werden Datenelemente im Inslot nicht gepuffert, d.h. es gehen Datenwerte verloren, wenn die mit dem Inslot verbundenen Outslots schneller Daten liefern, als die Anwendung verarbeiten kann. In vielen Fällen ist dies auch sinnvoll – bei Positionsdaten von Trackern ist z.B. immer nur die aktuellste Position von Interesse und daher der Verlust von älteren Positionsdaten unproblematisch und aus Gründen der Optimierung sogar erwünscht. Bei anderen Daten ist eine Pufferung dagegen unverzichtbar. Wenn z.B. bei einem Gamepad mit jedem Knopfdruck der gerade aktive Eintrag eines Menüs auf den nächsten Menüpunkt gesetzt wird, ist es wichtig, daß kein Knopfdruck verloren wird, andernfalls wird die Bedienung des Systems aus der Sicht des Anwenders „hakelig“. Daher gibt es einen „BufferedInslot“, der vom ungepufferten Inslot abgeleitet ist. Beim Erzeugen eines BufferedInslot kann man festlegen, wieviele Datenelemente von diesem Inslot gepuffert werden sollen. Allerdings gilt auch hier: Wenn der Puffer komplett gefüllt wird, gehen

Datenwerte verloren – der jeweils älteste Datenwert wird mit einem neuen Datenwert überschrieben.

Leider kann das Gerätemanagement-System nicht selbst feststellen, welche Daten gepuffert werden müssen, und welche nicht. Dies ergibt sich insbesondere nicht aus dem Datentyp, der vom Inslot empfangen werden soll.

Ebensowenig kann das Device-Management-System von sich aus feststellen, wie groß der Puffer sein muß, damit keine Daten verloren gehen. Das hängt stark von der Anwendung ab. Man kann insbesondere auch den Puffer nicht so einrichten, daß seine Größe automatisch wächst, wenn er voll ist. In diesem Fall bestünde die Gefahr, daß der gesamte Speicher des Systems aufgebraucht und das System insgesamt instabil werden würde, wenn eine Softwarekomponente aus irgendeinem Grund keine Daten von seinen Inslots mehr lesen würde.

Wenn ein Inslot mit mehreren Outslots verbunden ist, erhält er Datenwerte von allen diesen Outslots. Es gibt keine Möglichkeit herauszufinden, von welchem konkreten Outslot ein Datenwert stammt. Um sicherzustellen, daß nicht mehrere Outslots gleichzeitig in einen Inslot schreiben können, wird der Zugriff auf den Inslot über ein Mutex abgesichert. Bei gepufferten Inslots werden die Datenwerte in der Reihenfolge, in der sie eintreffen, im Puffer abgelegt, unabhängig davon, von welchem Outslot sie stammen.

3.3.1.2 Namespaces und Routes

Wie im vorherigen Kapitel beschrieben, können Outslots und Inslots manuell miteinander verbunden werden. Das mag im Einzelfall nützlich sein, im Allgemeinen ist diese Vorgehensweise aber nicht praktikabel, weil Anwendungen ja gerade nicht wissen sollen, mit welchen Slots sie sich verbinden müssen. Das Verbinden von Slots muß über einen anderen Mechanismus laufen, und dieser Mechanismus besteht aus den Namespaces und Routes, die in diesem Abschnitt beschrieben werden.

Namespaces sind Objekte, die beliebig von Anwendungen erzeugt werden können. In diese Namespaces können beliebig Outslots und Inslots eingefügt werden, wobei sie einen Namen erhalten müssen. Dieser Name ist ein beliebiger Textstring, der nicht eindeutig sein muß – es ist durchaus erlaubt, daß mehrere Outslots und Inslots den gleichen Namen haben. Aus dem Namen sollte aber hervorgehen, welchem Zweck der Slot dient.

Outslots und Inslots können zu beliebig vielen verschiedenen Namespaces hinzugefügt werden und dabei unterschiedliche Namen erhalten. Sie können aber auch beliebig oft zu ein und demselben Namespace hinzugefügt werden und dabei unterschiedliche Namen erhalten.

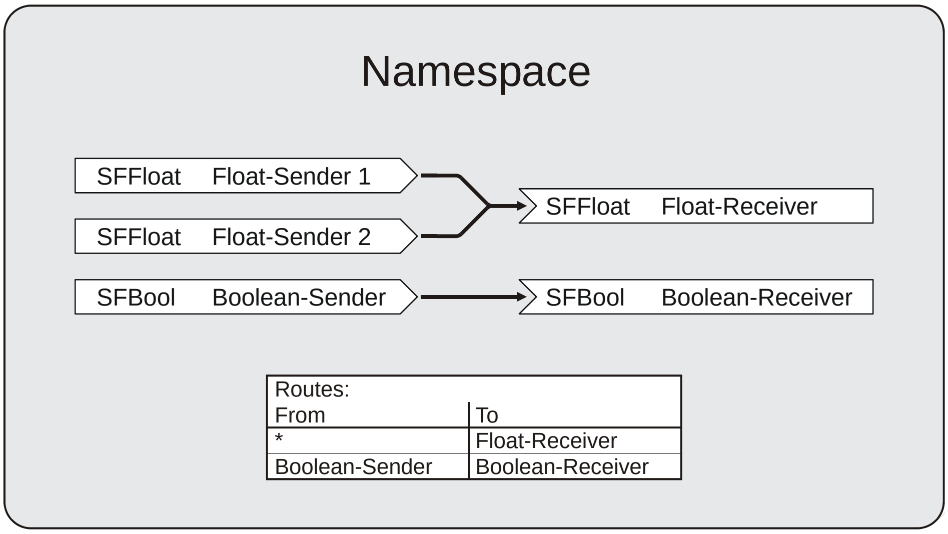


Abbildung 14: Routes verbinden Outslots mit Inslots

Neben den Slots enthalten Namespaces auch sogenannte „Routes“. Routes dienen dazu, Outslots und Inslots miteinander zu verbinden. Sie bestehen aus einem Paar von Textstrings, das Outslot-Namen auf Inslot-Namen abbildet. Outslots und Inslots werden automatisch miteinander verbunden,

1. wenn sie den gleichen Datentyp haben,
2. wenn sie beide Element des gleichen Namespaces sind,
3. und wenn in diesem Namespace eine Route existiert, die den Namen des Outslots in diesem Namespace auf den Namen des Inslots in diesem Namespace abbildet.

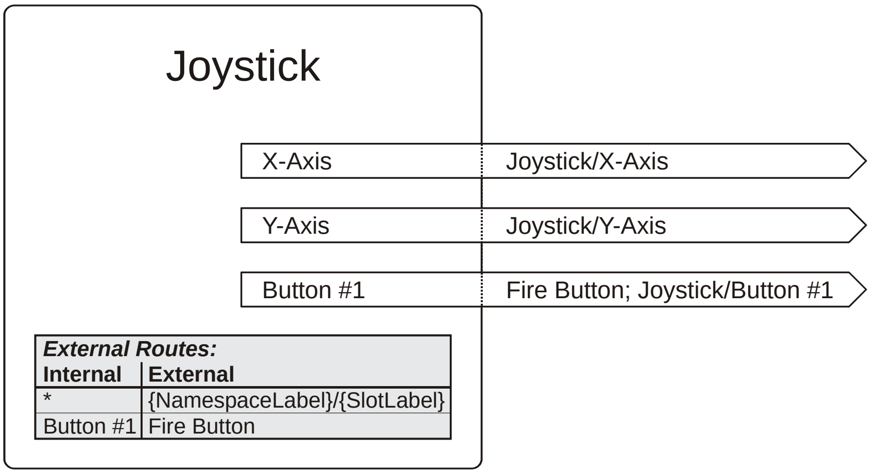
Nehmen wir z.B. mal an, die Anwendung sei ein Fahrsimulator. In diesem Fall benötigt die Anwendung Informationen über die Stellung von Lenkrad, Gaspedal und Bremspedal. Sie würde also drei SFFloat-Inslots erzeugen und mit den Namen „Wheel“, „Gas“ und „Brake“ einem Namespace hinzufügen. An den Rechner ist ein Lenkrad inklusive Pedale angeschlossen. Ein Joystick-Treiber fragt die aktuellen Stellungen des Lenkrads und der Pedale ab und liefert sie über drei SFFloat-OutSlots mit den Namen „Joystick/X-Axis“, „Joystick/Y-Axis“ und „Joystick/Z-Axis“ (genaueres zu Gerätetreibern etc. folgt im Abschnitt über das Highlevel-Interface). Die Anwendung müßte nun folgende Routes zum Namespace hinzufügen, um die Outslots mit den Inslots zu verbinden:

1. Von „Joystick/X-Axis“ nach „Wheel“
2. Von „Joystick/Y-Axis“ nach „Gas“
3. Von „Joystick/Z-Axis“ nach „Break“

Routes müssen nicht komplett den Namen der Slots spezifizieren, anstelle dessen können auch die Wildcards „?“ und „*“ in ihrer üblichen Bedeutung verwendet werden („?“ steht für genau ein beliebiges Zeichen, und „*“ steht für beliebig viele beliebige Zeichen).

Bei Anwendungen, die aus einer Vielzahl von Teilkomponenten bestehen, die über Slots miteinander kommunizieren, kann leicht die Übersicht über die in einem Namespace verfügbaren Slots und Routes verloren gehen. Daher gibt es die Möglichkeit, die Konfiguration eines Namespaces durch Subnamespaces zu strukturieren. Man kann zu jedem Namespace beliebig viele Subnamespaces hinzufügen. Um diese Subnamespaces zu identifizieren, erhalten sie wie Slots einen Namen, der aber im Gegensatz zu den Slotnamen eindeutig sein muß. Jeder dieser Subnamespaces kann einen beliebigen Teil seiner Outslots und Inslots an den Parent-Namespace exportieren. Dazu gibt es einen speziellen Typ von

Routes, die sogenannten „External-Routes“. External-Routes bestehen aus zwei Text-Strings, nämlich dem „Internal-Namen“ und dem „External-Namen“.



External Routes:

Internal	**External**
*	{NamespaceLabel}/{SlotLabel}
Button #1	Fire Button

Abbildung 15: External Routes exportieren Slots in den übergeordneten Namespace

Jeder Inslot und jeder Outslot, deren Name im Subnamespace mit dem Internal-Namen der Route übereinstimmt, wird mit dem External-Namen automatisch in den Parent-Namespace eingefügt (exportiert). Der Internal-Name darf dabei wieder die Wildcards „?“ und „*“ enthalten. Der External-Name darf dagegen keine Wildcards enthalten, schließlich muß er eindeutig sein. Dafür kann der External-Name zwei Variablen enthalten, nämlich „{NamespaceLabel}“ und „{SlotLabel}“. „{NamespaceLabel}“ wird dabei durch den Namen des Namespaces ersetzt, aus dem der Slot exportiert wird. „{SlotLabel}“ wird durch den ursprünglichen Namen des Slots im exportierenden Namespace ersetzt. Dies erlaubt es z.B. sehr elegant, alle in einem Subnamespace enthaltenen Slots mit nur einer einzigen External-Route in den übergeordneten Namespace zu exportieren und dabei die Namen aller Slots mit dem Namen des Namespace als Prefix zu versehen. Dazu muß der Internal-Name „*“ lauten, also alle Slots erfassen, und der External-Name „{NamespaceLabel}/{SlotLabel}“. Diese Technik wird intensiv von den Knoten des Datenflußgraphen genutzt, die Teil des Highlevel-Interface sind.

Subnamespaces sind ein sehr mächtiges Konstrukt, das es erlaubt, komplexe Konfigurationen von Outslots, Inslots und Routes in einer Art von „Makro“ zusammenzufassen, ähnlich wie es das PROTO-Konzept von VRML bzw. X3D erlaubt, neue Knotentypen aus Teilszenen zu erzeugen. Dieses Makro ist eine „Black Box“, d.h. der übergeordnete Namespace kann nicht sehen, wie der interne Aufbau der Subnamespaces ist, sondern er sieht nur die exportierten Outslots und Inslots. Es entsteht ein hierarchisches System von Teil-Datengraphen, das es erlaubt, komplexe Graphen übersichtlich zu strukturieren.

Das Konzept der Namespaces und Routes erlaubt es Anwendungen bereits, Outslots und Inslots miteinander zu verbinden, ohne die in einem Namespace vorhandenen Slots direkt zu kennen. Beliebig viele unterschiedliche Softwarekomponenten können ihre jeweiligen Slots einem Namespace hinzufügen. Das Verbinden der Slots übernimmt das Gerätemanagement-System. Die Anwendung steuert diesen Prozeß nur indirekt, indem sie Outslot-Namen über Routes auf Inslot-Namen abbildet. Trotzdem gibt es noch immer ein Problem: Wie können sich verschiedene Software-Komponenten darüber verständigen, welchem konkreten Namespace-Objekt sie ihre Slots hinzufügen? Im Idealfall wissen die beteiligten Komponenten nichts voneinander, d.h. das Gerätemanagement-System muß hier eine Lösung bereitstellen. Diese Lösung ist der sogenannte „Root-Namespace“. Es wird vom Gerätemanagement-System bereits ein Default-Namespace-Objekt zur Verfügung gestellt.

Dieses Namespace-Objekt können alle Softwarekomponenten nutzen, um eigene Slots oder Subnamespaces hinzuzufügen.

3.3.2 Highlevel-Interface

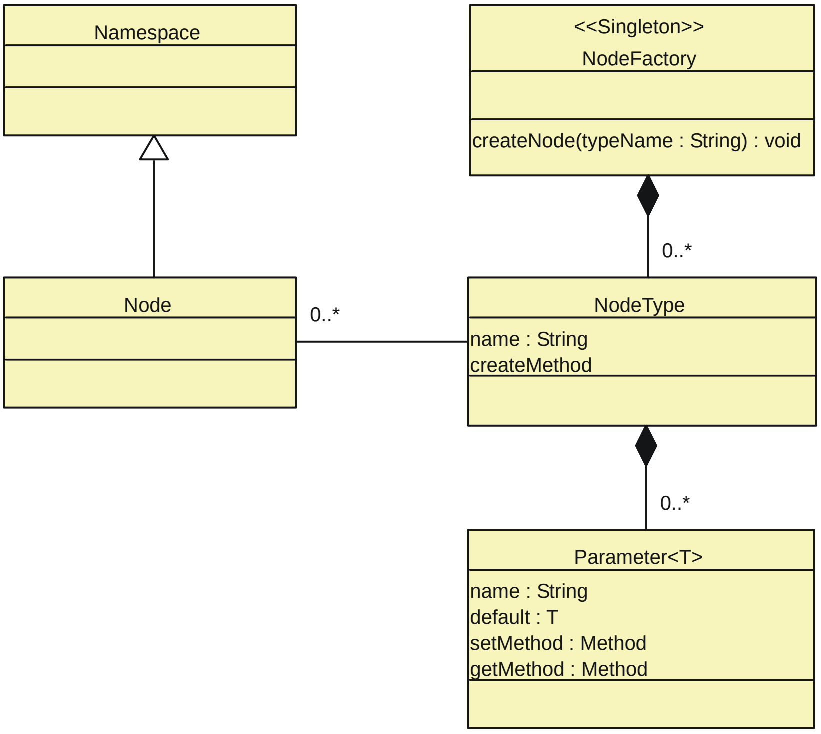


Abbildung 16: UML-Diagramm des Highlevel-Interface

Das im vorhergehenden Abschnitt beschriebene Lowlevel-Interface stellt bereits ein mächtiges Werkzeug dar, um Daten zwischen verschiedenen Softwarekomponenten auszutauschen. Entscheidend ist dabei, das die Softwarekomponenten nichts voneinander wissen müssen, um miteinander zu kommunizieren. Prinzipiell hat die bisher beschriebene Softwarearchitektur aber nichts mit Geräten zu tun. Sie ist nur die stabile Grundlage für das Highlevel-Interface, das in diesem Abschnitt behandelt wird.

3.3.2.1 Nodes

Nodes sind die Knoten des Datenflußgraphen. Es sind kleine Softwarekomponenten, die eine klar umrissene Teilaufgabe übernehmen. Diese Teilaufgaben sind z.B.:

- Gerätetreiber: Nodes, die dazu dienen, Daten von Eingabegeräten dem System bereitzustellen, oder Daten vom System an Ausgabegeräte zu transferieren. So gibt es Nodes, die Positionsdaten von Trackern oder GPS-Empfängern bereitstellen, die Stellung von Joystick- oder Spacemouse-Achsen abfragen, Sound-Daten von Soundkarten grabben oder auf Soundkarten abspielen, oder Video-Frames von Web-Cams oder Framegrabbern zu digitalisieren.
- Filter: Nodes, die dazu dienen, Daten zu filtern bzw. zu transformieren. So gibt es Nodes, die WGS84-Koordinaten, wie sie von GPS-Empfängern geliefert werden, in UTM-Koordinaten umwandeln, oder Farbbilder in Schwarz-Weiß-Bilder umwandeln, etc. Letztendlich sind auch Videotracker Filterknoten, die Bilder einer Videokamera in Positionswerte umsetzen.
- Zu guter Letzt gibt es noch Nodes, die überhaupt keine Daten erzeugen, transformieren oder verbrauchen. Dies klingt zunächst widersinnig, kommt aber

dennoch vor. Zu dieser Gruppe von Nodes gehören z.B. der Network-Node, der dafür verantwortlich ist, die Netzwerktransparenz herzustellen (siehe Kapitel 4), der Web-Node, der es erlaubt, das Gerätemanagement-System über einen normalen Web-Browser zu konfigurieren (siehe Kapitel 6), oder der Inline-Node, der es erlaubt, Datenflußgraphen aus Dateien nachzuladen und in den aktuellen Datenflußgraphen zu integrieren.

Nodes sind von Namespaces abgeleitet. Das bedeutet, daß sie genauso wie Namespaces verwendet werden können, d.h. man kann sie in die Namespace-Hierarchie einbauen sowie Outslots, Inslots und Routes hinzufügen. Allerdings macht es normalerweise wenig Sinn, Slots und Routes einem Node hinzuzufügen. Nodes sind Softwaremodule, die Outslots und Inslots in übergeordnete Namespaces exportieren und über diese Slots Daten an die Anwendung oder andere Nodes senden, oder Daten von der Anwendung oder anderen Nodes empfangen. Das Ziel ist es dabei, der Anwendung einen Pool von kleinen Modulen bereitzustellen, die klar umrissene Aufgaben haben. Die Anwendung kann diese Module zu einem Graphen zusammenstellen, der Daten von Geräten empfängt und in ein für die Anwendung brauchbares Format transformiert, oder Daten von der Anwendung in ein für die Geräte brauchbares Format transformiert und an die Geräte überträgt. Dabei können Teilbäume des Graphen auf verschiedenen Rechnern im Netzwerk laufen.

Eine wichtige Entscheidung, die der Entwickler eins Knotens fällen muß, ist die Frage, ob ein Knoten einen eigenen Thread erhalten sollte oder nicht. Knoten, die Datenwerte von Eingabegeräten lesen und dem System zur Verfügung stellen, erhalten normalerweise einen eigenen Thread, da sie an der Schnittstelle, an die das Gerät angeschlossen ist, auf neue Datenwerte warten müssen. Diese Möglichkeit, jeden Knoten mit einem eigenen Thread zu versehen, stellt einen großen Unterschied zu vielen existierenden Gerätemanagement-Systemen dar. Das hier beschriebene System ist kein Single-Thread-System, wo regelmäßig von einer Anwendung eine Methode aufgerufen werden muß, um aktuelle Datenwerte zu erhalten (das sogenannte „Polling“). Anstelle dessen kann jeder Knoten selbst aktiv sein und die Anwendung über Events (also über das Versenden von neuen Datenwerten über Outslots) auf Veränderungen hinweisen. Dies ist insbesondere auf leistungsschwachen mobilen Systemen wichtig, wo man die knappe Ressource Rechenzeit nicht durch unnützes Polling verschwenden will. Damit beim Einsatz mehrerer Threads die Datenkonsistenz gewahrt bleibt, ist die Datenübertragung über Outslots und Inslots thread-safe angelegt, d.h. Slots sind über Mutexe gegen den gleichzeitigen Zugriff zweier Threads geschützt.

Andere Knoten wie z.B. Filterknoten müssen keinen eigenen Thread enthalten. Diese Knoten können sich über Callbacks darüber informieren lassen, daß ein neuer Datenwert zur Verarbeitung eingetroffen ist, d.h. es wird immer dann, wenn ein Datenwert in einen InSlot des Knoten geschrieben wird, eine Methode des Knotens aufgerufen. Die Verarbeitung der Daten im Knoten wird dabei de facto von dem Thread übernommen, der den neuen Datenwert produziert hat, also üblicherweise dem Thread eines Gerätetreiber-Knotens. Allerdings können Filterknoten auch einen eigenen Thread enthalten. Dies empfiehlt sich immer dann, wenn die Verarbeitung der gelieferten Daten sehr zeitaufwendig ist, wie z.B. bei einem Videotracking-Knoten. Ohne einen eigenen Thread wäre der Thread des Videograbber-Knotens, der die digitalisierten Videobilder liefert, bis zum Ende des Tracking-Prozesses blockiert. Dadurch, daß der Tracking-Knoten einen eigenen Thread erhält, kann der Videograbber-Knoten dagegen schon während der Verarbeitung ein neues Bild von der Kamera entgegennehmen, d.h. die Parallelität der Verarbeitung steigt.

Nodes können von der Anwendung direkt erzeugt werden, da sie normale C++- bzw. Java-Objekte sind. Dann können Methoden dieser Objekte aufgerufen werden, um verschiedene Parameter zu setzten. Danach werden die Nodes in die Namespace-Hierarchie eingefügt und ihre Slots über Routes mit den Slots anderer Nodes oder den Slots der Anwendung verbunden.

Allerdings ist dieser Weg unflexibel, weil die Anwendung zum Erzeugen der Node-Instanzen die Node-Klasse zur Compile-Zeit kennen muß. Genau daß ist jedoch nicht erwünscht – schließlich soll die Anwendung ja zur Laufzeit umkonfiguriert werden und über Plugin-Mechanismen mit neuen Geräte-Treibern ausgerüstet werden können. Es muß also ein Weg gefunden werden, Nodes zu erzeugen, ohne die konkrete Nodeklasse zu kennen. Dies geschieht über einen System von Typ-Objekten, die Meta-Informationen über die verschiedenen Node-Klassen besitzen, sowie eines Factory-Objekts – ein klassisches Design Pattern, das z.B. schon vom 3D-Toolkit Inventor [65] verwendet wurde.

Zu jeder Node-Klasse gibt es ein Typ-Objekt, das eine Reihe von Meta-Informationen speichert:

- Einen eindeutigen Identifier. Dieser Identifier ist ein beliebiger Text-String, der verwendet wird, um die Node-Klasse aus der Anwendung heraus anzusprechen.
- Die Adresse einer Funktion, die neue Instanzen der jeweiligen Nodeklasse erzeugen kann.
- Informationen über die Parameter, die ein Node benötigt. Diese Information besteht aus den Parameter-Namen, den Parameter-Datentypen, und den Default-Werten, der verwendet werden, wenn die Anwendung den jeweiligen Parameter nicht spezifziert. Darüber hinaus werden die Klassenmethoden zum Lesen und Schreiben der jeweiligen Parameter gespeichert.
- Diversen anderen Informationen, die zwar vom Gerätemanagement-System nicht verwendet werden, aber Dokumentationszwecken dienen, wie z.B. Informationen über den Einsatzzweck eines Node, über die Personen, die den Node programmiert haben, sowie Beschreibungen der verfügbaren Parameter.

Die Typ-Objekte werden als statische Objekte angelegt, d.h. sie werden automatisch beim Programmstart oder beim Laden eines Plugins instanziiert und tragen sich automatisch in eine Liste aller verfügbaren Typ-Objekte ein.

Anwendungen können jetzt mittels eines Node-Factory-Objekts Instanzen von Nodes erzeugen, indem sie den Node-Identifier verwenden. Das Factory-Object sucht in der Liste der Typ-Objekte das Typ-Objekt mit diesem Identifier, und verwendet die im Typ-Objekt vorhandene Methode, um eine Instanz des gewünschten Nodes zu erzeugen. Dieses Node-Objekt wird dann von der Factory an die Anwendung zurückgeliefert.

Auch Parameter können von der Anwendung gelesen oder geschrieben werden, ohne daß die Anwendung das Interface der Node-Klasse kennen muß. Dazu gibt es in der abstrakten Node-Klasse, die der Vorfahre aller konkreten Node-Klassen ist, Methoden zum Lesen und Schreiben von Parametern. Die Methode zum Lesen von Parametern bekommt von der Anwendung einen String mit dem Namen des Parameters. Die Node-Klasse verwendet das zugehörige Typ-Objekt, um die zum Auslesen des Parameters benötigte Methode zu bestimmen und aufzurufen. Der von der Methode zurückgelieferte Parameter-Wert wird automatisch in einen String umgewandelt und an die Anwendung zurückgeliefert. Die Methode zum Schreiben von Parametern funktioniert analog – sie erhält von der Anwendung ebenfalls den Namen des Parameters und einen String mit dem Parameterwert. Die Node-Klasse verwendet das zugehörige Typ-Objekt, um die zum Schreiben des Parameters benötigte Methode zu bestimmen. Sie wandelt den Parameter-String automatisch in den benötigten Parameter-Datentyp um und ruft die Methode auf, um den Parameter zu setzen.

Das in diesem Abschnitt beschriebene System von Meta-Typinformationen und dem Node-Factory-Objekt eignet sich bereits dazu, die von der Anwendung verwendete Gerätekonfiguration über ein User-Interface während der Laufzeit zu modifizieren oder in Konfigurationsdateien zwischenzuspeichern. Weil das Speichern einer Gerätekonfiguration in einer Datei und das Wiederherstellen einer Gerätekonfiguration aus einer Datei ein häufig verwendete Vorgehensweise ist, um Anwendungen unabhängig von der im jeweiligen Fall verwendeten Gerätehardware zu machen, wird im folgenden Kapitel ein Standard-Format für eine solche Konfigurationsdatei beschrieben.

3.3.3 Konfigurationsdateien

Wie in den vorangehenden Abschnitten beschrieben, können Anwendungen die gesamte jeweils benötigte Konfiguration des Gerätemanagement-Systems manuell über das Erzeugen der notwendigen Objekte und durch Methodenaufrufen auf diesen Objekten erzeugen. In der Praxis ist diese Vorgehensweise jedoch zu unflexibel – jede Änderung in der Konfiguration würde Änderungen im Source-Code der Anwendung und ein anschließendes Neucompilieren erfordern. Daher gibt es auch die Möglichkeit, die komplette Konfiguration oder auch nur Teilbereiche davon in Konfigurationsdateien abzuspeichern. Diese Konfigurationsdateien können auf zwei Weisen verwendet werden:

1. Zum einen besitzt die Namespace-Klasse eine Methode, um die gesamte in diesem Namespace gewünschte Konfiguration aus der Konfigurationsdatei aufzubauen. Eine zuvor bereits im Namespace vorhandene Konfiguration wird dabei komplett überschrieben.
2. Zum anderen kann man die Konfigurationsdatei per Inline-Node nachladen. Der Inline-Node erhält beim Erzeugen als Parameter die URL der Konfigurationsdatei und baut die in dieser Datei enthaltene Konfiguration in sich selbst auf (Nodes sind ja von Namespaces abgeleitet). Alle Outslots und Inslots, die auf diese Weise im Node erzeugt werden, werden in den Parent-Namespace exportiert, der den Inline-Node enthält.

Für das Format der Konfigurationsdatei fiel die Wahl auf XML [66]. Diese Entscheidung wurde von den folgenden Faktoren beeinflußt:

- XML stellt den wichtigsten Standard für die Speicherung von Daten aller Art dar.
- Es ist eine Vielzahl von freien, stabilen und leistungsfähigen XML-Parsern für alle Programmiersprachen und Plattformen vorhanden. In Java ist ein XML-Parser sogar standardmäßig integriert.
- Es gibt eine Vielzahl von Editoren, mit denen man XML-Dateien erzeugen und editieren kann.
- Es gibt standardisierte Mechanismen und Verfahren, um ein XML-Dokument in ein anderes zu transformieren (XSLT [67]). Diese Mechanismen könnten in Zukunft auch für das hier beschriebene AR-Rahmensystem interessant werden, indem Teile der Anwendung auf einem noch höheren Abstraktionslevel über ein XML-Dokument beschrieben werden, und dieses XML-Dokument dann in das vom Gerätemanagement benötigte XML-Format bzw. in die X3D-XML-Darstellung des Szenengraphen transformiert wird. Ein Beispiel für die Anwendung eines solchen Verfahrens im Kontext des Studierstube-Systems ist die Augmented Presentation and Interaction Language (APRIL) [68].

Bekanntlich ist XML ein textbasiertes Format, das aus sogenannten „Tags“ und ihren Attributen besteht. Tags können hierarchisch ineinander verschachtelt werden. Für das Gerätemanagement wurden folgende Tags definiert:

HID:

Das „HID“-Tag ist das Wurzelelement des XML-Dokuments („HID“ ist der interne Codename des Systems). Es enthält zwei Attribute, „versionMajor“ und „versionMinor“, die die Version des XML-Formats enthalten, in dem die Konfiguration gespeichert ist. Änderungen, die die Kompatibilität zu älteren Versionen nicht zerstören, werden durch Hochzählen der „versionMinor“-Zahl angezeigt, und neue Formate, die zu alten nicht kompatibel sind, durch Hochzählen der „versionMajor“-Zahl. Das „HID“-Tag kann „Namespace“-, „Node“-, „Route“- und „ExternalRoute“-Tags enthalten.

Namespace:

Das „Namespace“-Tag erzeugt einen neuen Namespace im Datenflußgraphen. Es enthält ein Attribute, „label“, das den Namen des Namespace im übergeordneten Namespace festlegt. Wie das „HID“-Tag kann das „Namespace“-Tag „Namespace“-, „Node“-, „Route“- und „ExternalRoute“-Tags enthalten.

Node:

Das „Node“-Tag erzeugt einen neuen Knoten im Datenflußgraphen. Es enthält zwei Attribute, „label“ und „type“. „label“ legt wie beim „Namespace“-Tag den Namen des Knoten im übergeordneten Namespace fest. „type“ gibt an, welcher Knotentyp erzeugt werden soll. Wie das „Namespace“-Tag kann das „Node“-Tag „Namespace“-, „Node“-, „Route“- und „ExternalRoute“-Tags enthalten. Darüber hinaus kann es auch noch „Parameter“-Tags enthalten.

Route:

Das „Route“-Tag erzeugt eine neue Route im Datenflußgraphen, die Outslots und Inslots miteinander verbindet. Es enthält zwei Attribute, „from“ und „to“. „from“ legt den Namen des Outslot im aktuellen Namespace fest, „to“ den Namen des Inslot, der mit dem Outslot verbunden werden soll. Wie bereits erwähnt, kann sowohl „from“ als auch „to“ die Wildcards „*“ und „?“ in ihrer üblichen Bedeutung enthalten. Das „Route“-Tag kann keine anderen Tags enthalten.

ExternalRoute:

Das „ExternalRoute“-Tag erzeugt eine neue ExternalRoute im Datenflußgraphen, die Outslots oder Inslots vom aktuellen Namespace in den übergeordneten Namespace exportiert. Es enthält zwei Attribute, „internal“ und „external“. „internal“ legt den Namen des Outslots oder Inslots fest, der exportiert werden soll. Wie bereits erwähnt kann „internal“ die Wildcards „*“ und „?“ in ihrer üblichen Bedeutung enthalten. „external“ legt den Namen fest, den der exportierte Slot im übergeordneten Namespace erhalten soll. „external“ darf keine Wildcards enthalten, weil der Name selbstverständlich eindeutig sein muß. Es darf jedoch die Variablen „{NamespaceLabel}“ und „{SlotLabel}“ enthalten. „{NamespaceLabel}“ wird durch den Namen des Namespace ersetzt, aus dem der Slot exportiert wird. „{SlotLabel}“ wird durch den ursprünglichen Namen des Slots im exportierenden Namespace ersetzt. Das „ExternalRoute“-Tag kann keine anderen Tags enthalten.

Parameter:

Das „Parameter“-Tag definiert einen Parameter eines Knoten. Es enthält zwei Attribute, „name“ und „value“. „name“ legt fest, welcher Parameter durch das Tag definiert wird. „value“ legt den Wert des Parameters fest. Das „Parameter“-Tag kann keine anderen Tags enthalten.

Diese sechs Tags reichen aus, um die komplette Struktur eines Datenflußgraphen in einer Datei zu speichern und wieder zu rekonstruieren. Konkrete Beispiele für solche Dateien werden im 7. Kapitel vorgestellt.

3.3.4 Performanz-Betrachtungen

Ein wichtiges Qualitätsmerkmal von Gerätemanagement-Systemen ist ihre Performanz, d.h. die Frage,

- wieviel Overhead erzeugt wird, d.h. wieviel Rechenzeit und wieviele Ressourcen vom Gerätemanagement verbraucht werden,
- und wieviel Latenz (Verzögerung) das Gerätemanagement beim Zugriff auf die Daten erzeugt.

Diese Frage soll hier nun für das hier vorgestellte Gerätemanagement beantwortet werden. Es ist klar, daß ein gewisser Overhead und das Erzeugen von Latenz unvermeidbar ist. Das Gerätemanagement stellt eine zusätzliche Schicht zwischen der Hardware und der Anwendung dar, die selbstverständlich ihren Preis hat. Dafür gewinnt man allerdings eine größere Flexibilität und eine größere Effizienz beim Erstellen der Anwendung.

An dieser Stelle werden nun nicht die üblichen Meßreihen abgedruckt, die z.B. die Latenz des Systems bei verschiedenen Anwendungsszenarien angeben. Solche Meßreihen zeigen nur die Leistungsfähigkeit des verwendeten Rechnersystems, haben aber keinerlei Aussagekraft bezüglich der Performanz des Gerätemanagements. Anstelle dessen wird dargestellt, an welchen Stellen Overhead und Latenz entsteht.

Das optimale Szenario mit minimalem Overhead und minimaler Latenz stellt das direkte „Polling“ eines Geräts in einer Renderingschleife dar, um neue Werte zu erhalten. Dies stellt also die Meßlatte dar, mit der wir das Gerätemanagement zu vergleichen haben.

Wie bereits dargestellt, ist „Polling“ bei mobilen AR-Systemen nicht das optimale Verfahren, da es unnötig Ressourcen verbraucht. Anstelle dessen ist das hier beschriebene Gerätemanagement Event-basiert und besteht aus einer Reihe von Threads, die sich die Arbeit teilen. Für jedes Gerät, das Werte liefert, gibt es einen eigenen Knoten mit einem eigenen Thread. Dieser Thread ist normalerweise blockiert und wartet auf neue Datenwerte vom Gerät. Sind solche Datenwerte verfügbar, wird der Thread vom Betriebssystem aufgeweckt. Dieser Thread-Wechsel ist also die erste Verzögerungsquelle. Der Knoten-Thread liest dann die Werte aus und verschickt sie über einen Outslot an alle mit diesem verbundenen Inslots. Dabei werden die folgenden Schritte ausgeführt:

1. Das Mutex, welches den thread-sicheren Zugriff auf den Outslot sicherstellt, wird gelockt.
2. Es wird eine Liste mit allen gegenwärtig verbundenen Inslots abgearbeitet.
3. Bei jedem Inslot wird zunächst ein Mutex, das den thread-sicheren Zugriff auf den Inslot sicherstellt, gelockt.
4. Der neue Datenwert wird in jeden Inslot kopiert. Falls es sich um große Datenwerte handelt, wird nur ein Zeiger auf den Datenwert kopiert. In diesem Fall wird ein Smart-

Pointer verwendet, dessen Reference-Counter hochgezählt werden muß. Dieser Reference-Counter ist ebenfalls über ein Mutex gesichert, das gelockt werden muß.

5. Die Mutexe der Inslots sowie das Mutex des Outslots werden wieder geunlockt.

Nach diesen Schritten ist der neue Wert im Inslot der Anwendung eingetroffen. Die weitere Verzögerung hängt nun davon ab, auf welche Weise die Anwendung neue Datenwerte aus Inslots ausliest:

- Duch Polling. In diesem Fall wird das Mutex des Inslots gelockt, der Datenwert ausgelesen, und das Mutex wieder geunlockt. Allerdings ist Polling wie oben beschrieben für mobile AR-Systeme nicht durchführbar.
- Durch Callbacks: In diesem Fall wird eine Methode der Anwendung aufgerufen, wenn ein neuer Datenwert in den InSlot geschrieben wird. Das Mutex muß beim Auslesen des Datenwertes nicht gelockt werden, das es bereits gelockt ist. Dies ist die effizienteste Form für die Anwendung, neue Datenwerte zu erhalten.
- Durch Blockieren der Anwendung am Inslot, bis neue Werte eintreffen. Wenn das passiert, wird der Anwendungsthread aufgeweckt, lockt das Mutex des Inslots, liest den Datenwert aus, und unlockt das Mutex. Diese Form des Zugriffs ist für die Anwendung die ineffizienteste, um neue Daten zu erhalten.

Im allerschlimmsten Fall (Verwendung von Smart-Pointern, die Anwendung blockiert am Inslot) müssen also zwei Threads geweckt und vier Mutexe gelockt werden, was selbstverständlich eine Verzögerung bedeutet. Man sollte diese Verzögerung aber auch nicht überbewerten: Zum einen sind moderne Systeme für Multithreading-Anwendungen optimiert, d.h. das Aufwecken von Threads und das Locken von Mutexen ist nicht so teuer, wie es zunächst den Eindruck hat. Außerdem kann man ja davon ausgehen, daß ein Event-basiertes System (im Gegensatz zu Systemen mit einer Renderingschleife) nicht mit 100% Prozessorlast läuft – eine Reaktion auf eintreffende Datenwerte kann also sehr schnell erfolgen. Darüber hinaus sollte man auch nicht aus den Augen verlieren, wo in einer modernen AR-Anwendung die meiste Verzögerung auftritt: Nämlich beim Tracking, insbesondere der Bestimmung von Position und Orientierung des Anwenders aus Videobildern, sowie beim Rendern. Im Vergleich zur Rechenzeit, die für diese beiden Tätigkeiten verloren geht, ist der Overhead des Gerätemanagement-Systems verschwindend gering. Es macht also normalerweise mehr Sinn, Videotracking und Rendering zu optimieren, als beim Overhead des Gerätemanagements anzusetzen. Insbesondere, weil man für diesen Overhead ja auch etwas gewinnt, nämlich eine größere Flexibilität, eine einfachere Anwendungsentwicklung und eine größere Parallelität beim Verarbeiten von Daten.

3.4 Zusammenfassung

In diesem Kapitel wurde ein neuartiges Gerätemanagement-System vorgestellt. Es wurden die Anforderungen an ein Gerätemanagement beim Einsatz in einem mobilen AR-System aufgestellt und ein Überblick über andere Systeme gegeben. Das hier vorgestellte Gerätemanagement-System unterscheidet sich in den folgenden Punkten von anderen Lösungen:

1. Es verwendet nicht das Konzept der Geräteklassen. Dieses Konzept hat sich in der Praxis als zu unzureichend erwiesen, um alle existierenden Geräte innerhalb des Gerätemanagements abzubilden. Anstelle dessen werden Geräte als Knoten dargestellt, die eine beliebige Anzahl von Datenströmen an die Anwendung senden oder von der Anwendung empfangen können.

2. Ausgabegeräte stehen völlig gleichberechtigt neben den Eingabegeräten. Praktisch alle existierenden Systeme behandeln Ausgabegeräte entweder gar nicht oder nur stiefmütterlich. Das hier verwendete Konzept der Datenströme, die von Knoten (d.h. Geräten) nicht nur versendet, sondern auch empfangen werden können, bindet dagegen Ausgabegeräte gleichwertig in das System ein.
3. Das hier vorgestellte Gerätemanagement basiert auf einem modernen Event-Konzept und nicht auf „Polling“, wie es von vielen aus dem VR-Bereich stammenden Systemen verwendet wird. Polling ist problemlos durchführbar, wenn man eine Renderingschleife besitzt, wie sie bei stationären VR-Installationen üblich ist. Auf einem mobilen AR-System kann man dagegen nicht so verschwenderisch mit der knappen Ressource Rechenzeit umgehen.
4. Das hier vorgestellte Gerätemanagement erlaubt eine weitgehende Verarbeitung von Daten innerhalb des Systems. Viele Systeme liefern meist nur die „rohen“ Gerätedaten, die üblicherweise erst in ein für die Anwendung brauchbares Format umgewandelt werden müssen. Das Konzept des Datenflußgraphen dagegen erlaubt es elegant, diese Form der Verarbeitung innerhalb des Gerätemanagement-Systems durchzuführen, indem die Gerätetreiber-Knoten mit speziellen Filterknoten verbunden werden. Dies entlastet die Anwendung, sorgt für eine stärkere Kapselung von Geräteabhängigkeiten, und erleichtert die Arbeit des Anwendungsentwicklers.

Es wurde in diesem Kapitel auch dargelegt, welchen Overhead das Gerätemanagement-System erzeugt. Dieser Overhead ist aber klein im Vergleich zu den wirklichen Ressourcenfressern Videotracking und Rendering auf einem mobilen AR-System. Außerdem erhält man für diesen Overhead auch den Vorteil einer größeren Flexibilität und einer leichteren Anwendungsentwicklung.

Im folgenden Kapitel wird nun unter anderem darauf eingegangen, wie das Gerätemanagement netzwerktransparent gemacht wird. Im 5. Kapitel wird gezeigt, wie das Gerätemanagement-System in das Rendering-System integriert wird. Das User-Interface des Gerätemanagement-Systems wird im 6. Kapitel beschrieben. Beispiele für Datenflußgraphen und ihre Darstellung als XML-Dateien sind unter den Anwendungsbeispielen im 7. Kapitel zu finden.

4 Netzwerkkommunikation

Die Fähigkeit, über ein Netzwerk mit anderen Geräten zu kommunizieren und Informationen mit ihnen auszutauschen, ist insbesondere für mobile Geräte eine Notwendigkeit und eine besondere Herausforderung. Mobile Geräte besitzen meist nur eingeschränkte Ressourcen, sei es Speicherplatz oder Rechenleistung. Sie sind daher auf eine unterstützende Infrastruktur angewiesen. Außerdem ist es heute fast schon eine Selbstverständlichkeit, auch unterwegs „Online" zu sein, Informationen aus dem Internet abzurufen oder mit anderen Menschen zu kommunizieren.

Die besondere Herausforderung bei mobilen Geräten besteht genau darin, daß sie ständig ihren Standort verändern können. Das bedeutet, daß sie nicht wie Desktop-Rechner statisch von einem Systemadministrator konfiguriert werden können. Anstelle dessen müssen sie in der Lage sein, selbständig die am gegenwärtigen Standpunkt verfügbaren Kommunikationsnetze aufzuspüren und sich in sie einzuklinken sowie die im Netzwerk vorhandenen Dienste ausfindig zu machen.

Da mobile Systeme ihren Standort verändern, kann es ständig passieren, daß die Verbindung zu einem Netzwerk verloren geht. WLAN z.B. hat eine Reichweite von maximal 300 Metern. In der Praxis liegt die erzielbare Reichweite deutlich darunter, in geschlossenen Räumen innerhalb von Gebäuden, die in massiver Stahlbetonbauweise ausgeführt sind, kann die Verbindung schon beim Wechsel in einen anderen Raum abbrechen. Das System muß in diesem Fall in der Lage sein, die Verbindung mit dem Netzwerk und mit Diensten innerhalb des Netzwerkes wiederherzustellen, sobald das System wieder ein Signal empfängt – oder es muß in der Lage sein, Verbindung mit einem anderen Netzwerk aufzunehmen (das sogenannte „Roaming").

Ein weiteres Problem besteht darin, daß die Bandbreite drahtloser Kommunikationsnetze deutlich geringer ist als die drahtgebundener Netze. Bluetooth 1.2 hat nur eine Bandbreite von 1 Mbit/s. Das weitverbreitete WLAN nach dem IEEE 802.11b-Standard hat eine Bandbreite von 11 Mbit/s, der neuere Standard IEEE 802.11g bietet 54 Mbit/s. Dies sind nur die theoretischen Maximalwerte unter optimalen Bedingungen, in der Praxis liegen die erzielbaren Bandbreiten deutlich darunter. Darüber hinaus teilen sich alle Anwender die verfügbare Bandbreite. Bei drahtgebundenem Ethernet sind dagegen 100 Mbit/s Standard, und Gigabit-Ethernet erlaubt sogar bis zu 1 Gbit/s. Es ist daher bei mobilen Systemen wichtig, die Menge der über das Netzwerk zu übertragenden Daten zu minimieren.

In diesem Kapitel werden Techniken zum Einsatz von mobilen AR-Systemen in einem Netzwerk behandelt. Es wird dabei davon ausgegangen, daß es sich um ein WLAN auf der Basis von TCP/IP handelt – dies ist gegenwärtig die Standardtechnik zur drahtlosen Datenübertragung. Im Einzelnen werden die folgenden Punkte behandelt:

- Wie kann der Anwender erkennen, welche Netze am aktuellen Standort vorhanden sind, und wie kann er sich mit ihnen verbinden? Dies ist ein Problem, das nicht nur AR-Systeme, sondern generell alle mobilen Computersysteme betrifft. Daher bieten alle aktuellen Betriebssysteme hierfür bereits Lösungen an. Ein kurzer Überblick darüber wird in Kapitel 4.1 gegeben.
- Wie kann der Anwender nach einem Verbindungsaufbau mit einem Netzwerk feststellen, welche Dienstleistungen in diesem Netzwerk verfügbar sind? Und wie kann das mobile AR-System die im Netz verfügbaren Ressourcen wie z.B. Daten oder Geräte lokalisieren, die es zum Ausführen der vom Benutzer gewünschten Anwendung benötigt? Auch dieses Problem ist nicht allein auf AR-Systeme beschränkt. Dennoch ist es gegenwärtig Stand der Technik, daß der Anwender

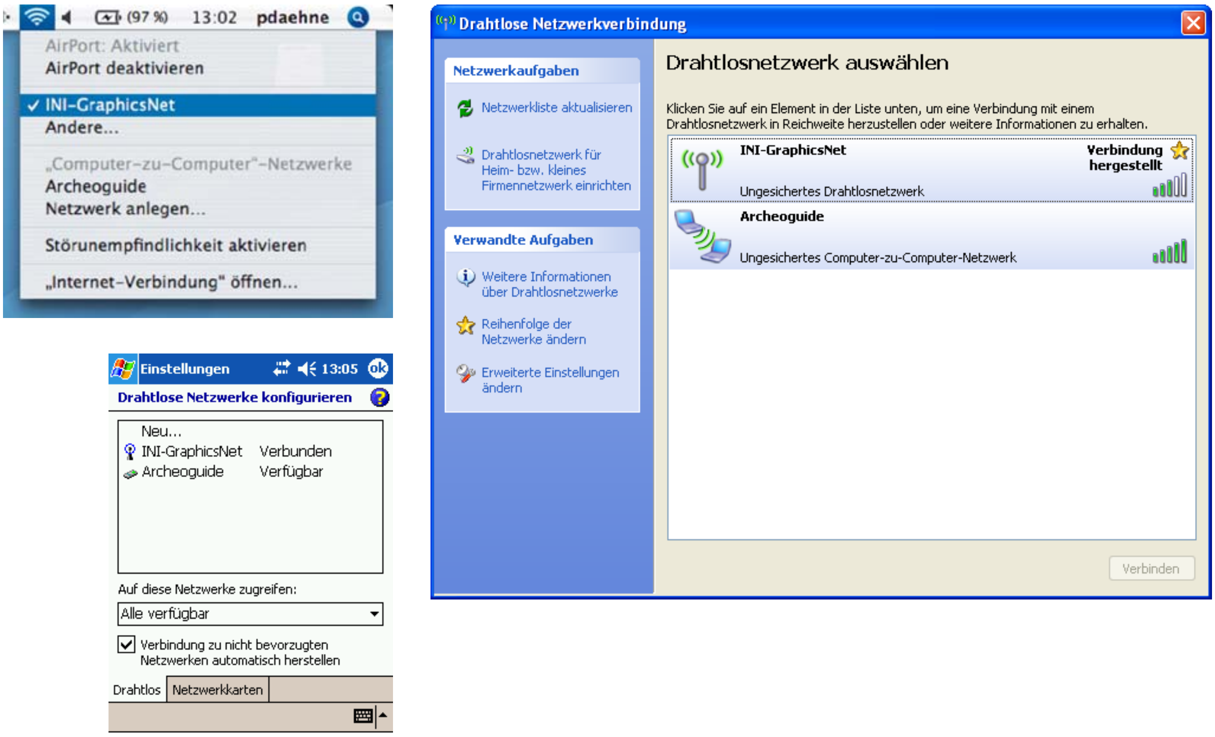
Ressourcen nicht lokalisieren kann, sondern ihre Adresse (also z.B. ihre URL) kennen muß. Eine Reihe von neuen Techniken soll diesen Mißstand beseitigen. Sie werden in Kapitel 4.2 vorgestellt und auf ihre Brauchbarkeit für das in dieser Arbeit konzipierte AR-System geprüft.

- Und wie kann letztendlich die Kommunikation zwischen dem AR-System und den im Netzwerk verfügbaren Ressourcen softwaretechnisch realisiert werden? Dabei gilt es verschiedene Kommunikationsformen zu berücksichtigen:
    - Download von Daten, die zur Anwendungsausführung benötigt werden, wie z.B. 3D-Modelle, Bilder oder Videos.
    - Kommunikation mit Geräten im Netz, wie z.B. stationären Trackingsystemen.
    - Kommunikation mit anderen AR-Systemen, z.B. zur Umsetzung von CSCW-Anwendungen.

  Kapitel 4.3 beschreibt, wie Netzwerkkommunikation in das in dieser Arbeit vorgestellte AR-Rahmensystem integriert ist.

4.1 Verbindungsaufbau mit einem Netzwerk

Der erste Schritt, um über ein Netzwerk zu kommunizieren, ist der Verbindungsaufbau mit dem Netzwerk. Dabei gilt es folgende Parameter definieren:



☑ Verbindung zu nicht bevorzugten Netzwerken automatisch herstellen

Klicken Sie auf ein Element in der Liste unten, um eine Verbindung mit einem Drahtlosnetzwerk in Reichweite herzustellen oder weitere Informationen zu erhalten.

Abbildung 17: Moderne Betriebssysteme erlauben es dem Anwender, die am aktuellen Standort verfügbaren drahtlosen Netzwerke aufzulisten (links oben: Mac OS X, links unten: Windows CE, rechts: Windows XP)

- Welches der am aktuellen Standort verfügbaren Netzwerke soll verwendet werden? Es ist ohne weiteres möglich, daß sich mehrere WLANs überlappen. Zur eindeutigen Identifizierung der WLAN-Netzwerke wird einfach jedem Netzwerk ein eindeutiger Name zugewiesen, der sogenannte „Service Set Identifier“ (SSID). Um sich mit einem Netzwerk zu verbinden, muß der Anwender diesen SSID angeben. Während dies früher manuell erfolgen mußte, erlauben es inzwischen alle wichtigen

Betriebssysteme, die an einem Ort vorhandenen Netzwerke aufzulisten, sodaß der Anwender durch einfaches Anklicken mit der Maus das gewünschte Netzwerk auswählen kann (siehe Abbildung 17).

- Welches Verschlüsselungsverfahren wird verwendet, und wie identifiziere ich mich gegenüber dem Netzwerk? Da die Daten drahtlos über das Netzwerk übertragen werden, kann praktisch jeder, der über einen passenden Empfänger verfügt, die versendeten Daten mitlesen. Um dies zu verhindern, müssen die Daten verschlüsselt werden. Dafür gibt es eine Reihe von verschiedenen Verfahren. Standard für WLANs sind die Verfahren „WEP“ sowie das modernere „WPA“, die die Eingabe eines Schlüssels erfordern, um mit dem Netzwerk zu kommunizieren. Eine andere Möglichkeit ist die Verwendung eines „Virtual Private Networks“ (VPN). Es liegt in der Natur der Sache, daß sich der Vorgang der Identifizierung nicht beliebig automatisieren läßt. In vielen Anwendungsfällen ist eine Identifizierung auch gar nicht notwendig, wie z.B. bei Netzwerken, die ausschließlich kostenlos lokale Informationen zur Verfügung stellen. Oder der Tourist, der eine historische Stätte besucht, erhält zusammen mit seiner Eintrittskarte ein Passwort oder eine Schlüsselkarte, die es ihm erlaubt, sein AR-System in das lokale Netz einzuklinken.
- Schließlich müssen auch noch die typischen Parameter für das IP-Protokoll definiert werden, wie die eindeutige IP-Adresse des mobilen Geräts, die Netzwerk-Maske und das Gateway. Während dies früher manuell von der Systemverwaltung spezifiziert werden mußte, haben sich hier inzwischen automatisierte Verfahren durchgesetzt. Beim „Dynamic Host Configuration Protocol“ (DHCP) [69] erhalten die mobilen Geräte diese Informationen von einem speziellen DHCP-Server, der direkt nach dem Verbindungsaufbau mit dem Netzwerk kontaktiert wird. Für Netzwerke ohne DHCP-Server gibt es das Autokonfigurations-Verfahren Auto-IP [70][71], das es erlaubt, in Netzwerken ohne jede zentrale Verwaltungsinfrastruktur eindeutige IP-Adressen für die am Netzwerk beteiligten Geräte zu erzeugen.

Zusammenfassend kann man feststellen, daß es für die Verbindungsaufnahme mit einem drahtlosen Netzwerk inzwischen ausgereifte Techniken gibt, die diesen Vorgang weitestgehend automatisieren und auch für den Computerlaien beherrschbar machen. Dies hängt natürlich damit zusammen, daß die hier beschriebenen Probleme nicht spezifisch für AR-Systeme sind – jedes mobile Gerät, das mit einem Netzwerk verbunden werden soll, muß wie oben beschrieben konfiguriert werden.

4.2 Lokalisierung von Diensten im Netz

Nachdem der Anwender sein AR-System erfolgreich mit dem lokalen Netzwerk verbunden hat, steht er vor dem Problem, die in diesem Netzwerk verfügbaren Dienste und Anwendungen zu lokalisieren. Im Gegensatz zum Verbindungsaufbau mit dem Netzwerk, den moderne Betriebssysteme inzwischen sehr anwenderfreundlich und weitestgehend automatisiert ermöglichen, wird die automatische Lokalisierung von Diensten im Netz bislang noch kaum unterstützt – Dienste müssen immer noch manuell vom Anwender lokalisiert werden. Das bedeutet: Um Anwendungen oder Daten von einem Server (z.B. Web- oder FTP-Server oder Datenbanken) herunterzuladen, muß der Anwender die Adresse (bzw. die URL) des Servers kennen. Genauso ist es mit im Netzwerk verfügbaren Geräten: Auch hier muß der Anwender die Adresse kennen, unter der das Gerät verfügbar ist.

Es ist offensichtlich, daß dieser Zustand bei mobilen Anwendungen nicht akzeptabel ist. Der Anwender eines mobilen AR-Systems kennt ja nicht die Struktur des jeweiligen Netzwerks, in das er sich einklinkt. Er kann daher auch nicht die Adressen von Diensten eingeben, die er in Anspruch nehmen möchte.

Anstelle dessen wird ein Mechanismus benötigt, der dem Anwender alle aktuell verfügbaren Dienste auflistet. Aus dieser Liste kann der Anwender dann den gewünschten Dienst auswählen. Darüber hinaus müssen Anwendungen in die Lage versetzt werden, selbständig von ihnen benötigte Geräte im Netz zu finden.

Eine Lösung für dieses Problem ist ein „Whiteboard", auf dem sich alle verfügbaren Dienste eintragen, d.h. ein spezieller Server, auf dem Informationen über Dienste gesammelt und für andere Softwarekomponenten zur Verfügung gestellt werden. Ein Beispiel dafür sind die CORBA Object Request Broker (ORB), die die Kommunikation zwischen verschiedenen CORBA-Komponenten ermöglichen. Aber diese Lösung vereinfacht das Problem nur – der Anwender muß nicht mehr die Adressen von allen Diensten kennen, aber er muß immer noch die Adresse des Whiteboards kennen.

Auch wenn die automatische Lokalisierung von Diensten im Netz von aktuellen Betriebssystemen kaum unterstützt wird, stehen für diese Aufgabe bereits eine Reihe von Lösungen zur Verfügung. Diese Lösungen funktionieren alle nach einem ähnlichen Prinzip:

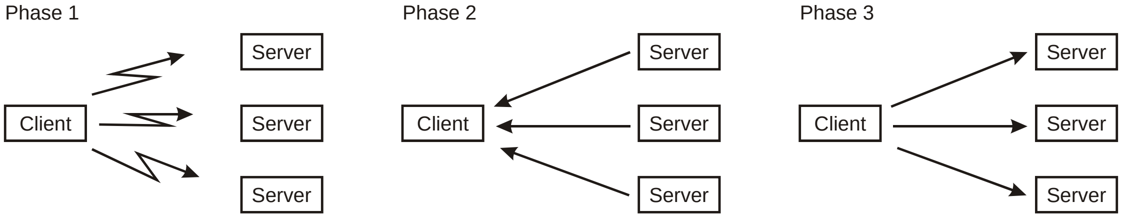


Abbildung 18: Service Discovery

- Wenn ein Client gestartet wird, der einen Dienst (Server) benötigt, so wird ein sogenanntes „Service Discovery" durchgeführt, das aus drei Phasen besteht (siehe Abbildung 18). In der ersten Phase sendet der Client eine Anfrage an eine Multicast-Gruppe, in welcher der Typ des gesuchten Dienstes spezifiziert wird. Alle vorhandenen Server empfangen Anfragen, die an diese Multicast-Gruppe geschickt werden. Falls ein Server den vom Client gesuchten Dienst anbietet, sendet er eine Antwort mit seiner Adresse an den Client (Phase 2). Der Client kann sich dann in der dritten Phase mit dem Server verbinden.

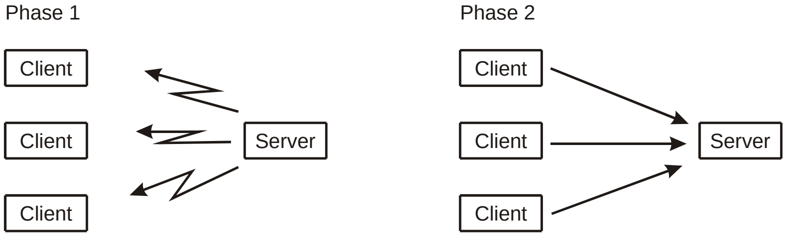


Abbildung 19: Service Announcement

- Wenn umgekehrt ein neuer Server gestartet wird, der einen Dienst anbietet, wird das sogenannte „Service Announcement" oder „Service Advertisement" durchgeführt, das aus zwei Phasen besteht (siehe Abbildung 19). In der ersten Phase sendet der Server eine Nachricht an die Multicast-Gruppe, die den Typ des Dienstes und seine Adresse enthält. Alle Clients empfangen Nachrichten, die an diese Multicast-Gruppe geschickt werden. Wenn der angebotene Dienst dem vom Client gesuchten Dienst entspricht, kann sich der Client in der zweiten Phase mit dem Server verbinden.

Im Folgenden wird eine Übersicht und Vergleich existierender Lösungen zu Lokalisierung von Diensten im Netz gegeben. Für eine Bewertung sind die folgenden Punkte von Bedeutung:

- Die Lokalisierung sollte möglich sein, ohne daß ein spezieller zentraler Server benötigt wird, der Informationen über die verfügbaren Dienste sammelt. Zentrale Server können einen Flaschenhals darstellen. Beim Ausfall eines solchen Servers wird das gesamte System lahmgelegt. Darüber hinaus sind auch AR-Einsatzszenarien denkbar, bei denen es keine zentrale Verwaltungsinfrastruktur gibt, z.B. wenn sich mehrere Besitzer von AR-Systemen spontan zusammentun, um ein AR-Spiel miteinander zu spielen. Andererseits ist die Lokalisierung von Diensten über einen lokalen Server häufig effizienter, zuverlässiger und ressourcenschonender. Daher wäre eine optionale Unterstützung von zentralen Servern wünschenswert.
- Das Verfahren zur Lokalisierung von Diensten sollte nicht festlegen, wie die Kommunikation zwischen den Clients mit den Diensten ablaufen soll. Es gibt eine Vielzahl von Kommunikationsprotokollen, die für ihren spezifischen Einsatzzweck und für spezifische Daten optimiert sind, wie z.B. HTTP zur Übertragung von Hypertext-Dokumenten oder RTSP und RTP zur Übertragung von Video- und Audio-Datenströmen. Einschränkungen bei der Auswahlt der Kommunikationsprotokolle durch das Lokalisierungsverfahren sind daher nicht akzeptabel.
- Es sollte eine Implementierung frei im Sourcecode für alle wichtigen Betriebssysteme und Programmiersprachen zur Verfügung stehen. Das in dieser Arbeit vorgestellte AR-Rahmensystem ermöglicht es, AR-Anwendungen auf allen Betriebssystemplattformen auszuführen, für die das Rahmensystem zur Verfügung steht. Diese Plattformunabhängigkeit sollte nicht durch das Lokalisierungsverfahren eingeschränkt werden.

In im folgenden Text werden nun drei Lokalisierungsverfahren, nämlich „Universal Plug and Play“, „Multicast Domain Name Server“ sowie das „Service Location Protocol“, genauer beschrieben und festgestellt, inwieweit sie die oben aufgestellten Anforderungen erfüllen.

4.2.1 Universal Plug and Play (UPnP)

Universal Plug and Play ist ein Verfahren zur Lokalisierung von Diensten im Netz, das ursprünglich von Microsoft entwickelt wurde. Heute liegt die Weiterentwicklung in den Händen des UPnP Forums[8]. UPnP ist in die aktuellen Microsoft-Betriebssysteme integriert.

UPnP basiert auf dem Hypertext Transfer Protocol (HTTP), das auch bei der Kommunikation zwischen Web-Servern und -Browsern eingesetzt wird. HTTP diente ursprünglich nur zum Transfer von Dateien über TCP-Verbindungen. Für den Einsatz bei UPnP wurden eine Reihe von neuen Protokollen spezifiziert, die es erlauben, HTTP auch zum Nachrichtenaustausch über Multicast-Verbindungen zu verwenden. Diese Protokolle sind im einzelnen:

- HTTP Multicast over UDP (HTTPMU) [72]
- Simple Service Discovery Protocol (SSDP) [73]
- General Event Notification Architecture (GENA) [74]

Wenn ein Rechner einen Dienst benötigt, sendet er eine SSDP-Anfrage an eine Multicast-Gruppe, in der er den gewünschten Dienst spezifiziert. Alle Rechner, die Dienste anbieten, empfangen Anfragen, die an diese Multicast-Gruppe geschickt werden. Falls ein Rechner den gewünschten Dienst anbietet, sendet er eine SSDP-Antwort, die die URL eines XML-Dokumentes enthält, das weitere Informationen über den Dienst liefert.

---

[8] http://www.upnp.org/

Neben diesem Frage-Antwort-Prozess (dem sogenannten „Discovery“) können Rechner auch die von ihnen angebotenen Dienste bekanntgeben (das sogenannte „Advertisement“), ohne daß es vorher eine Anfrage gab. Dazu wird eine SSDP-Nachricht an die Multicast-Gruppe geschickt. Dies passiert u.a., wenn ein neuer Dienst gestartet wird, oder wenn ein Dienst beendet wird. Auf diese Weise wird sichergestellt, daß alle Rechner im Netz alle verfügbaren Dienste kennen, ohne ständig neue Anfragen verschicken zu müssen.

Nachdem ein Rechner einen Dienst gefunden hat, den er in Anspruch nehmen möchte, verwendet er die erhaltene URL, um ein XML-Dokument herunterzuladen, das weitere Informationen über den Dienst enthält. Insbesondere steht in diesem Dokument, welche SOAP-Funktionsaufrufe der Dienst unterstützt, und welche Parameter diese Dienste benötigen.

Die eigentliche Kommunikation mit dem Dienst erfolgt über SOAP-Funktionsaufrufe. Die Tatsache, daß nicht nur das Protokoll zum Auffinden von Diensten von UPnP spezifiziert wird, sondern auch das Protokoll zwischen Dienst und Client-Rechner, stellt die größte Schwachstelle von UPnP dar.

Freie Implementierungen von UPnP sind für die wichtigsten Betriebssystemplattformen verfügbar.

4.2.2 Multicast Domain Name Server (MDNS) / Bonjour

Multicast DNS [75][76] ist ein Verfahren zur Lokalisierung von Diensten im Netz, das von der Zeroconf Working Group[9] entwickelt wurde. Ziel dieser Arbeitsgruppe ist es, Protokolle zu entwickeln, die eine automatische Konfiguration aller für die Kommunikation in IP-basierten Netzen benötigter Parameter durchführen. MDNS wird insbesondere von Apple propagiert, das diese Technologie unter dem Markennamen „Bonjour“ in Mac OS X integriert hat.

MDNS stellt eine Erweiterung des altbekannten DNS-Protokolls dar. Bekanntlich besitzen alle Rechner in IP-basierten Netzwerken eine eindeutige IP-Adresse. Um mit einem Rechner zu kommunizieren, muß dessen IP-Adresse bekannt sein. Da sich numerische Adressen schlecht merken lassen, verwendet man üblicherweise symbolische Rechnernamen, wie z.B. „www.igd.fraunhofer.de“. Diese Rechnernamen werden in eine IP-Adresse umgewandelt, indem man eine Anfrage an einen sogenannten DNS-Server stellt. DNS-Server haben Zugriff auf Datenbanken, in denen zu jedem symbolischen Rechnernamen die zugehörige IP-Adresse gespeichert ist. Dieser ganze Prozess der Umwandlung eines symbolischen Rechnernamens in eine IP-Adresse wird vom Betriebssystem durchgeführt und ist für den Anwender völlig transparent.

Interessanterweise können DNS-Server aber nicht nur IP-Adressen zur Verfügung stellen, sondern auch eine Vielzahl von weiteren Informationen in ihrer Datenbank speichern, die von anderen Rechnern über das DNS-Protokoll abgefragt werden können. Insbesondere können sie auch Informationen über im Netzwerk vorhandene Dienste bereitstellen. Dazu gibt es in der Datenbank sogenannte SRV-Records, die den Typ des Dienstes speichern sowie die Adresse und die Port-Nummer, unter denen der Dienst verfügbar ist.

Das herkömmliche DNS-Protokoll erfordert zwingend einen DNS-Server im Netzwerk. Das Multicast-DNS-Protokoll stellt eine Erweiterung des DNS-Protokolls dar, die es erlaubt, auf den DNS-Server zu verzichten. Die Idee dabei ist, die DNS-Anfrage nicht an einen DNS-Server zu schicken, sondern an eine Multicast-Gruppe. Jeder Rechner, der Dienste bereitstellt,

---

[9] http://www.zeroconf.org/

empfängt Anfragen, die an diese Multicast-Gruppe geschickt werden, und beantwortet sie, wenn er dazu in der Lage ist.

Wenn also z.B. ein Rechner auf der Suche nach allen Web-Servern im Netzwerk ist, dann schickt er eine entsprechende DNS-Anfrage an die Multicast-Gruppe. Alle Rechner im Netz empfangen diese Anfrage. Rechner, auf denen kein Web-Server läuft, ignorieren die Anfrage einfach. Rechner dagegen, auf denen ein Web-Server läuft, schicken eine DNS-Antwort mit der eigenen Adresse und der Port-Nummer, unter der der Web-Server erreichbar ist, an die Multicast-Gruppe. Der Rechner, der die Anfrage ursprünglich gestellt hat, empfängt die Antworten und erhält so die Adressen und Port-Nummern aller im Netz erreichbaren Web-Server.

Neben diesem Frage-Antwort-Prozess (dem sogenannten „Discovery“) können Rechner auch DNS-Antworten mit Informationen über die von ihnen angebotenen Dienste verschicken (das sogenannte „Advertisement“), ohne daß es vorher eine Anfrage gab. Dies passiert u.a., wenn ein neuer Dienst gestartet wird, oder wenn ein Dienst beendet wird. Auf diese Weise wird sichergestellt, daß alle Rechner im Netz alle verfügbaren Dienste kennen, ohne ständig neue Anfragen verschicken zu müssen.

Herkömmliches DNS (mit einem DNS-Server) und Multicast-DNS lassen sich problemlos nebeneinander betreiben. So können Anfragen zuerst an einen DNS-Server geschickt werden und dann, wenn sie von diesem nicht beantwortet werden können, an das gesamte Netzwerk.

Multicast-DNS schreibt nicht vor, welche Protokolle die Dienste verwenden müssen, die über MDNS bereitgestellt werden. Ein Rechner, der einen Dienst sucht, spezifiziert diesen Dienst über einen Identifier. Die Antwort enthält die Adresse und die Port-Nummer, unter der der Dienst erreichbar ist. Wie die Kommunikation zwischen dem Dienst und dem Client-Rechner abläuft, wird von MDNS nicht festgelegt.

Eine freie Implementierung des MDNS-Protokolls in C wird von Apple im Source-Code für alle wichtigen Betriebsystemplattformen zur Verfügung gestellt. Darüber hinaus sind auch andere Implementierungen in C und in Java frei verfügbar (z.B. Avahi[10]).

4.2.3 Service Location Protocol (SLP)

Service Location Protocol [77] ist ein Verfahren zur Lokalisierung von Diensten im Netz, das von Sun entwickelt wurde.

Anders als UPnP oder MDNS basiert SLP nicht auf existierenden Standards. Die prinzipielle Vorgehensweise ähnelt aber den beiden anderen Verfahren. Eine Anwendung, die einen Service benötigt, sendet eine SLP-Nachricht an eine Multicast-Gruppe, in der sie den gewünschten Service spezifiziert. Alle Rechner, die einen Dienst anbieten, sind Mitglied dieser Multicast-Gruppe. Wenn ein Rechner den in der SLP-Nachricht angegebenen Dienst anbietet, antwortet er dem Client mit einer SLP-Nachricht, die die URL des Service enthält. Der Aufbau dieser speziellen Service-URLs wird in einem weiteren Dokument [78] spezifiziert. Der Client verwendet die in der Service-URL enthaltenen Informationen, um sich mit dem Server zu verbinden.

Anders als bei UPnP oder MDNS führen neu gestartete Services kein „Service Announcement“ durch, d.h. sie senden keine Multicast-Nachricht an alle Clients, mit der sie ihre Existenz bekannt geben. Das bedeutet, das Clients immer aktiv nach Diensten suchen müssen – der Einsatz von SLP ist also eher auf statische Netzwerke ausgerichtet, bei denen sich das Angeboten an Diensten eher selten ändert.

---

[10] http://avahi.org/

Zusätzlich zur direkten Kommunikation zwischen Clients und Services können auch noch zentrale Server eingesetzt werden, sogenannte „Directory Agents“ (DA). Clients und Services erhalten die Adresse eines Directory Agents entweder über Konfigurationsdateien, einem DHCP-Server oder durch den normalen, oben beschriebenen „Service Discovery“-Prozeß. Sobald Services einen Directory Agent gefunden haben, müssen sie sich bei ihm registrieren. Sobald Clients einen Directory Agent gefunden haben, müssen sie Services ausschließlich bei diesem Directory Agent anfragen, d.h es werden keine Multicast-Anfragen mehr verschickt.

Neben dem SLP-Verfahren, den verwendeten Nachrichten und den Service-URLs wird auch noch eine SLP-API für die Programmiersprachen C und Java sowie das Format von Konfigurationsdateien spezifiziert [79]. Dies soll es Anwendungsentwicklern ermöglichen, ohne Modifikation ihrer Anwendungen die aktuelle SLP-Implementierung wechseln zu können.

Es sind freie Implementierungen von SLP für alle wichtigen Plattformen verfügbar, z.B. OpenSLP[11]. Im Kontext AR wird SLP von DWARF [30] verwendet.

4.2.4 Bewertung der Lokalisierungsverfahren

Die folgende Tabelle zeigt noch einmal zusammenfassend, inwieweit die drei untersuchten Lokalisierungsverfahren die an sie gestellten Anforderungen erfüllen:

	UPnP	**MDNS**	**SLP**
Zentraler Server	optional	optional	optional
Beschränkung der Kommunikationsprotokolle	ja	nein	nein
Plattformunabhängigkeit	ja	ja	ja

Wie man sieht, erfüllt nur UPnP die Anforderungen nicht, weil es als Kommunikationsprotokoll zwischen den Diensten zwingend SOAP festlegt. MDNS und SLP erfüllen dagegen die Anforderungen und sind beide ähnlich gut geeignet. Für die Implementierung des in dieser Arbeit beschriebenen AR-Rahmensystems wurde MDNS ausgewählt, weil es im Gegensatz zu SLP dynamische Netzwerke unterstützt, in denen sich die angebotenen Dienste häufiger ändern. Darüber hinaus ist es bereits in ein Betriebssystem (Mac OS X) integriert und damit schon erprobter Bestandteil kommerziell genutzter Software.

4.3 Kommunikation im Netz

Nachdem sich der Anwender mit einem lokalen drahtlosen Netzwerk verbunden, die verfügbaren Dienste lokalisiert und eine AR-Anwendung gestartet hat, ist das System in der Lage, mit anderen Software- und Hardware-Komponenten im Netz zu kommunizieren. In diesem Kapitel wird nun beschrieben, auf welche Weise das AR-Rahmensystem diese Kommunikation durchführt. Dabei gilt es verschiedene Kommunikationsformen zu betrachten:

- Daten-Download von Servern im Netz. Auf dem mobilen Endgerät ist ja zunächst einmal nur das AR-Rahmensystem installiert. Die eigentliche Anwendung sowie die zur Anwendungsausführung notwendigen Daten wie 3D-Objekte, Bilder, Texte sowie

---

[11] http://www.openslp.org/

Video- und Audio-Datenströme werden vom lokalen Netzwerk zur Verfügung gestellt. Das AR-Rahmensystem muß daher in der Lage sein, Anwendungen und Daten während der Laufzeit vom Netzwerk zu laden.

- Kommunikation mit Geräten im Netzwerk. Mobile Endgeräte besitzen nur eingeschränkte Ressourcen. Sie sind daher darauf angewiesen, durch eine leistungsfähige lokale Infrastruktur unterstützt zu werden, die möglichst viele rechenaufwendige Aufgaben übernehmen kann. Ein typisches Beispiel ist das Tracking der Geräte – in einigen AR-Szenarien [16][17] bestimmen die mobilen Geräte ihre aktuelle Position nicht selbst, sondern bekommen sie von einer stationären Tracking-Hardware übermittelt. Das in Kapitel 3 beschriebene Gerätemanagement-System muß also nicht nur in der Lage sein, dem System den Zugriff auf lokal angeschlossene Geräte zu ermöglichen, sondern auch auf im Netzwerk vorhandene Hardware.
- Kommunikation mit anderen Anwendern. In vielen Fällen sind AR-Anwendungen ein Gemeinschaftserlebnis (z.B. bei Infotainment-Anwendungen) oder sie unterstützen die Arbeit in einem Team (z.B. beim Remote-Expert-Szenario). Daher ist es selbstverständlich, daß die Anwender auch untereinander über die mobilen Geräte kommunizieren wollen oder müssen. Das reicht von einfachen Chat- bzw. Telefonie-Anwendungen bis hin zu komplexen Computer-Supported-Cooperative-Work-(CSCW-)Szenarien. Auch diese Form der Kommunikation muß das AR-System unterstützen.

4.3.1 Anwendungs- und Daten-Download

Der Download der Anwendung und der von der Anwendung benötigten Daten wie z.B. Texte, Bilder, 3D-Objekte sowie Video- und Audio-Datenströme aus dem Netzwerk wird vom VRML-/X3D-Standard, der die Basis des AR-Systems darstellt, bereits weitestgehend unterstützt. VRML war ja ursprünglich für die Erstellung von 3D-Anwendungen im Internet konzipiert. Alle Knoten im Szenengraphen, die externe Daten benötigen, spezifizieren die Position dieser Daten im Netz über URLs. Der eigentliche Download der Daten geschieht über Standard-Internet-Protokolle wie HTTP und FTP.

Datentyp	**X3D-Knoten**	**Beispiel**
3D-Objekte, Subgraphen	Inline	Inline { url "http://www.igd.fhg.de/object.wrl" }
Anwendungslogik	Script	Script { url "http://www.igd.fhg.de/script.js" }
Bilder	ImageTexture	ImageTexture { url "http://www.igd.fhg.de/image.jpg" }
Audio-Daten	AudioClip	AudioClip { url "http://www.igd.fhg.de/sound.wav" }
Videos	MovieTexture	MovieTexture { url "http://www.igd.fhg.de/movie.mpg" }

Die Knoten erlauben einen Datendownload nicht nur zum Zeitpunkt der Instanziierung, d.h. beim Anwendungsstart. Da die URL-Felder sogenannte „Exposed fields“ sind, können sie zur Laufzeit geändert werden – auf diese Weise können während der Anwendungsausführung weitere Daten nachgeladen werden.

Falls diese Form des Datendownloads zu eingeschränkt ist, hat der Anwendungsentwickler die Möglichkeit, Daten auch über den Anwendungscode nachzuladen, der über Script-Knoten in den Szenengraphen integriert ist. Das System unterstützt gegenwärtig Java- und Javascript-Code. Java bietet vielfältige Möglichkeiten, mit dem Netzwerk zu kommunizieren, insbesondere erlaubt es auch, über die JDBC-Schnittstelle Daten aus Datenbanken auszulesen. Javascript dagegen bietet dagegen in der vom X3D-Standard spezifizierten Form keine Möglichkeit, auf Ressourcen im Netzwerk zuzugreifen. Gleiches gilt z.B. für Javascript-Code, der in HTML-Seiten eingebunden ist. Seit kurzem gibt es jedoch eine Javascript-Erweiterung, die von allen Web-Browsern unterstützt wird, nämlich AJAX (Asynchronous JavaScript and XML). AJAX besteht im Kern aus einem neuen Javascript-Objekt „XMLHttpRequest“ [80], das es Javascript-Code erlaubt, Daten von Servern herunterzuladen. Trotz seines Namens ist dieses Objekt nicht auf XML-Daten und HTTP als Protokoll beschränkt, vielmehr kann es jeden Datentyp über jedes vom Web-Browser unterstützte Protokoll herunterladen. XML-Daten werden allerdings besonders unterstützt, weil sie der Javascript-Anwendung bereits fertig geparsed als DOM-Baum zur Verfügung gestellt werden.

Die AJAX-Erweiterung ist natürlich nicht nur für HTML-Seiten interessant, sie kann genau so auch für X3D-Anwendungen verwendet werden. Daher wurde für das in dieser Arbeit beschriebene AR-System die Standard-X3D-Javascript-Implementierung um das XMLHttpRequest-Objekt erweitert, sodaß auch von Javascript aus ein Zugriff auf Daten im Netzwerk möglich ist.

4.3.2 Kommunikation mit Geräten

Bei der Kommunikation des AR-Systems mit Geräten im Netzwerk besteht die Aufgabe darin, das im 3. Kapitel beschriebene Gerätemanagement-System netzwerktransparent zu machen, d.h. es sollte aus der Sicht der Anwendung keinen Unterschied machen, ob ein Gerät direkt an das AR-System angeschlossen ist oder über das Netzwerk angesprochen werden muß.

Prinzipiell gibt es zwei Ansätze, wie man Anwendungen, die auf Graphen basieren, netzwerktransparent gestalten kann:

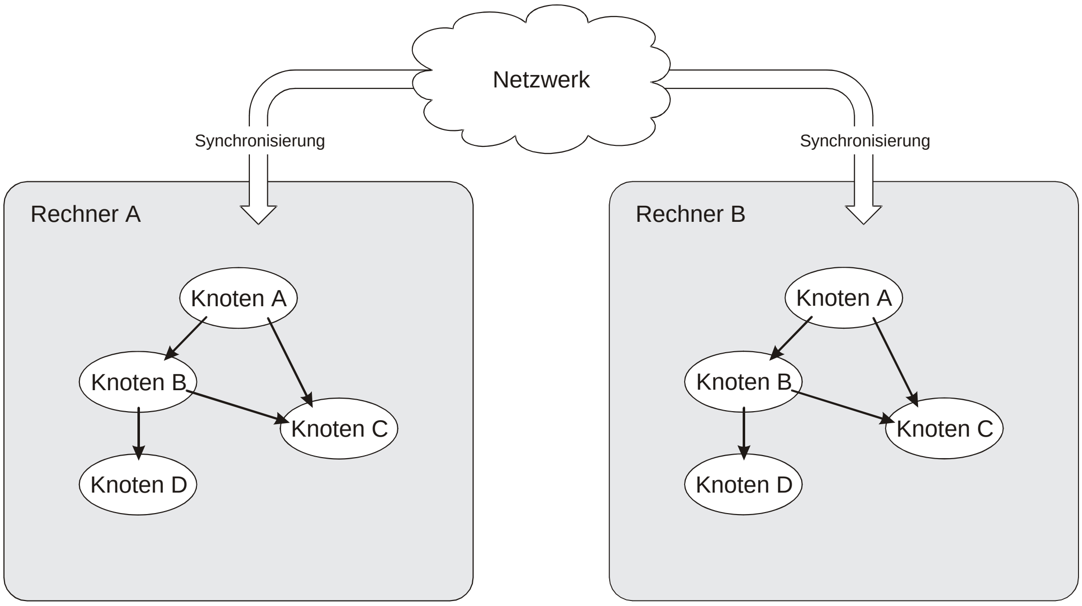


Abbildung 20: Netzwerktransparenz durch Erzeugen identischer Graphen auf allen Rechnern, die miteinander synchronisiert werden

Zum einen kann man auf allen beteiligten Rechnern im Netzwerk identische Kopien des Graphen erzeugen (siehe Abbildung 20). Änderungen in einem Graphen, d.h. Hinzufügen oder Entfernen von Knoten bzw. Zustandsänderungen der Knoten, werden auch in allen anderen Graphen durchgeführt, d.h. die Graphen werden über das Netzwerk synchronisiert. Diese Form der Netzwerktransparenz ist bequem für den Anwendungsentwickler, weil es für ihn überhaupt keine Rolle spielt, daß die Anwendung auf verschiedenen Rechnern im Netzwerk abläuft. Er arbeitet nur auf seiner lokalen Kopie des Graphen, und um die Synchronisation mit den anderen Kopien kümmert sich komplett das Laufzeitsystem. Allerdings hat diese Lösung auch schwerwiegende Nachteile. Eine komplette Kopie des kompletten Graphen auf allen beteiligten Systemen anzulegen ist häufig überflüssig – ein Trackingsystem, das einfach nur die Position der mobilen Systeme bereitstellt, benötigt keine komplette Kopie der Anwendungsdaten. Um die Menge der zu synchronisierenden Daten zu verringern, gibt es in vielen Systemen, die auf dieser Form der Netzwerktransparenz basieren, die Möglichkeit, nur Teilgraphen zu replizieren. Darüber hinaus ist unklar, was passiert, wenn unterschiedliche Änderungen gleichzeitig an der gleichen Stelle in zwei oder mehr Kopien des Graphen durchgeführt werden – welche Kopie hat nun den gültigen Zustand des Graphen? Letztendlich ist diese Form der Netzwerktransparenz dann sinnvoll, wenn die Anwendung eine weitgehende Spiegelung von Anwendungsdaten auf den beteiligten Rechnersystemen erfordert und klar geregelt ist, wie mit Konflikten umgegangen wird. So findet man diese Form der Netzwerktransparenz z.B. beim AR-Rahmensystem Studierstube [28], dessen Fokus insbesondere auf CSCW-Anwendungen liegt, oder bei Szenengraphen-Systemen wie z.B. OpenSG [81], die die komplette Szene auf mehreren Rechnern im Netz rendern müssen (Clustering).

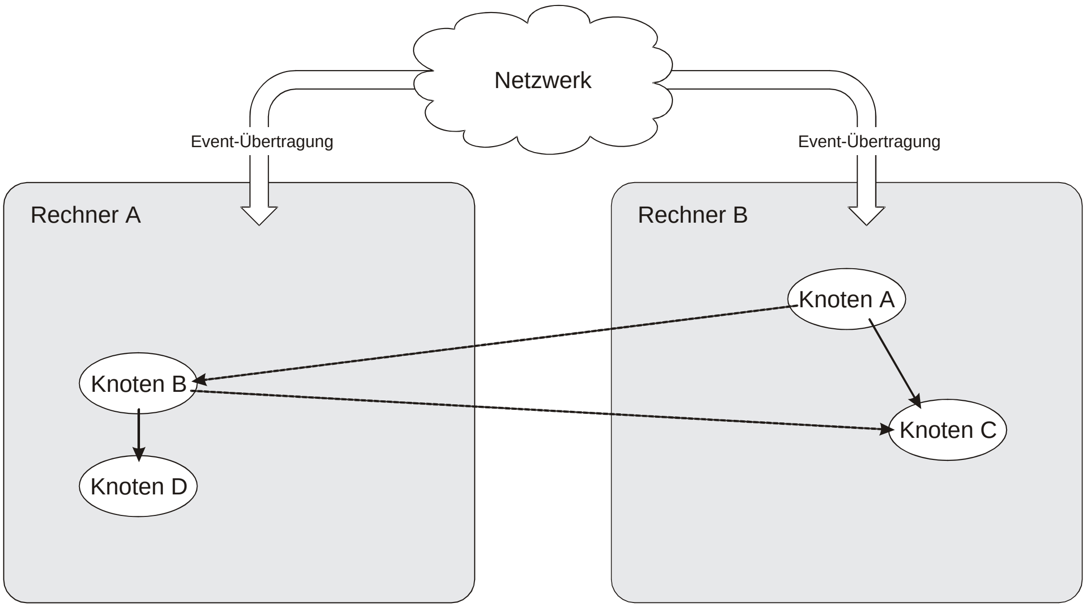


Abbildung 21: Erzeugen von Netzwerktransparenz durch Kanten zwischen Knoten auf verschiedenen Rechnern

Ein anderer Weg, Netzwerktransparenz zu erzeugen, besteht darin, Kanten zwischen Knoten auf verschiedenen Rechnern zu ermöglichen (siehe Abbildung 21). Anstatt identische Kopien des gesamten Graphen auf allen Rechnern zu erzeugen, werden einzelne Teile des Graphen auf verschiedene Rechner im Netz verteilt. Die einzelnen Teilkomponenten können über Kanten miteinander kommunizieren, genau so wie Knoten, die sich auf ein und demselben Rechner befinden. Eines der frühesten VR-Frameworks, das diese Form der Netzwerktransparenz beherrscht, ist Avocado bzw. AVANGO [82][83]. Im Kontext von Augmented Reality verwenden u.a. OpenTracker [55], DWARF [30] und MORGAN [39] diese Lösung. Netzwerktransparenz über Kanten zwischen Knoten auf verschiedenen Rechnern erfordert mehr Aufwand vom Anwendungsentwickler – er muß eine sinnvolle Verteilung der Teilgraphen auf die Rechner festlegen. Dafür ist diese Lösung deutlich effizienter – jede Teilkomponente des Systems bekommt nur diejenigen Daten übertragen, die sie tatsächlich zur Durchführung ihrer jeweiligen Aufgabe benötigt. Für den Einsatz in einem Gerätemanagement-System ist diese Eigenschaft besonders günstig, weil hier die Teilkomponenten des Graphen, d.h. die einzelnen Geräte, klar umrissene, isolierte Aufgaben haben, für die nicht das gesamte Anwendungswissen erforderlich ist. Daher wird dieses Verfahren verwendet, um das in Kapitel 3 beschriebene Gerätemanagement-System netzwerktransparent zu gestalten.

Netzwerktransparenz ist kein direkter Bestandteil des Gerätemanagement-Systems. Anstelle dessen wird sie durch einen speziellen „Network“-Knoten erzeugt. Dieser Knoten wird genauso wie andere Knoten erzeugt und zu einem Namespace hinzugefügt. Jeder Network-Knoten verbindet sich mit Hilfe des in Kapitel 4.2 beschriebenen MDNS-Verfahrens [75] automatisch mit anderen Network-Knoten im lokalen Netzwerk. Das bedeutet, daß der Anwendungsentwickler nicht spezifizieren muß, wo in einem Netzwerk ein bestimmter Teilgraph vorhanden ist. Abbildung 22 zeigt zwei Rechner R_1 und R_2, auf denen jeweils ein Teilgraph mit einem Network-Knoten (N_1 bzw. N_2) vorhanden ist. Die Network-Knoten sind über das Netzwerk miteinander verbunden.

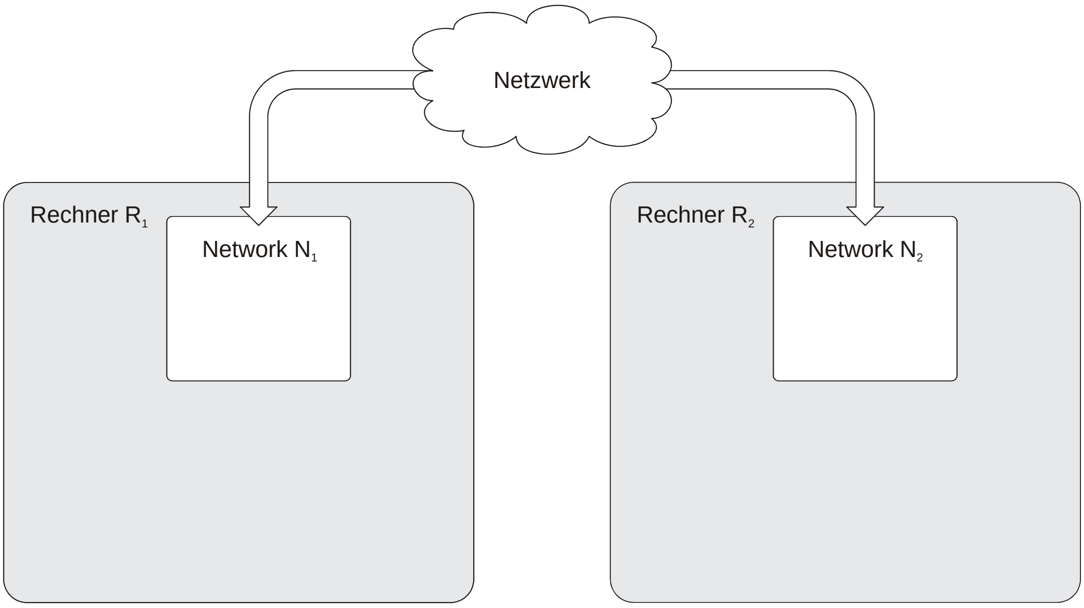


Abbildung 22: Zwei „Network“-Knoten auf verschiedenen Rechnern, die über das Netzwerk miteinander verbunden sind

Abbildung 23 zeigt, was passiert, wenn ein Knoten K_1 mit einem Outslot O_1 ① dem gleichen Namespace hinzugefügt wird, in dem sich auch der Network-Knoten N_1 auf Rechner R_1 befindet. Der Network-Knoten N_1 sendet eine „addOutslot“-Nachricht ② an alle anderen Network-Knoten, mit denen er verbunden ist. Diese Nachricht enthält den Namen „O_1“ des Outslots sowie den Typ „T“ der Daten, die über diesen Outslot verschickt werden können. Die anderen Network-Knoten fügen sich selbst einen Outslot mit diesem Namen und Datentyp hinzu ③.

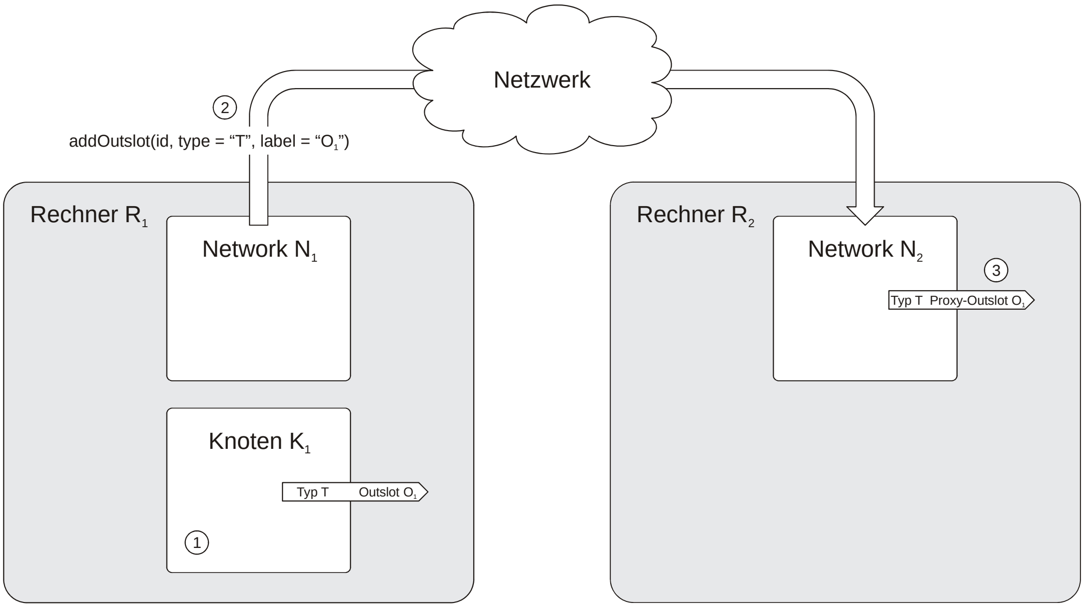


Abbildung 23: Ein Knoten K_1 mit einem Outslot O_1 wird dem Graphen auf Rechner R_1 hinzugefügt

Diese Outslots auf den anderen Rechnern stellen nun „Proxies“ dar, d.h. sie übernehmen eine Stellvertreterrolle für den ursprünglichen Outslot O_1 auf Rechner R_1. Dies erlaubt es, wie gewohnt Routes zwischen Slots auf verschiedenen Rechnern zu erzeugen, indem Routes zwischen den lokalen Slots und den Proxy-Slots erzeugt werden. Dies wird z.B. in Abbildung 24 dargestellt. Dem Graphen auf Rechner R_2 wird ein Knoten K_2 mit einem Inslot I_2 hinzugefügt ① und dieser Inslot mit dem Proxy-Outslot O_1 verbunden. Der Network-Knoten N_2 auf Rechner R_2 sendet eine „startOutslot“-Nachricht ② an Rechner R_1. Der Network-Knoten N_1 auf Rechner R_1 zählt daraufhin einen Reference-Counter für den Outslot O_1 hoch. Wenn dieser Reference-Counter Eins wird, also die erste Verbindung aufgebaut wird, fügt der Network-Knoten N_1 sich selbst einen Inslot I_P hinzu und verbindet ihn mit dem ursprünglichen Outslot O_1 ③. Dieser Inslot I_P bildet jetzt einen Proxy für den Inslot I_2 auf Rechner R_2.

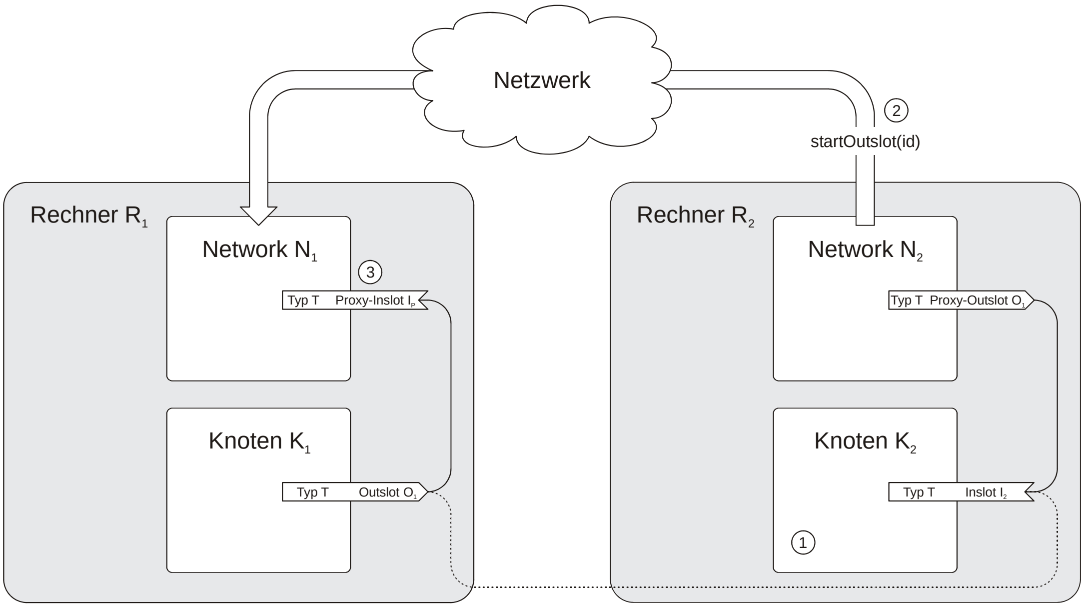


Abbildung 24: Ein Inslot auf Rechner R_2 verbindet sich mit dem Proxy-Outslot

Alle Daten, die nun über den Outslot O_1 verschickt werden ①, werden vom Network-Knoten N_1 auf Rechner R_1 über seinen Proxy-Inslot I_P empfangen ② (siehe Abbildung 25). Diese

Daten werden nun kodiert, d.h. in eine Form gebracht, die unabhängig von Rechnerhardware und Betriebssystem ist („Serialisierung“) und, im Falle von großvolumigen Daten wie z.B. Videobildern oder Audiosamples, komprimiert. Dies wird von speziellen Codecs (Codierern/Decodierern) übernommen, die das System bereitstellt. Die resultierenden Daten werden über eine „newData“-Nachricht ③ an den Network-Knoten N_2 auf Rechner R_2 übertragen. Dieser dekomprimiert bzw. dekodiert die Daten („Deserialisierung“) und verschickt sie über seinen Proxy-Outslot O_1 ④. Letzendlich werden die Daten vom Inslot I_2 empfangen, der mit dem Proxy-Outslot O_2 verbunden ist ⑤. Über die physikalischen Verbindungen über die Proxy-Slots wurde also eine logische Verbindung zwischen dem Outslot O_1 auf Rechner R_1 und dem Inslot I_2 auf Rechner R_2 hergestellt (in der Abbildung gestrichelt dargestellt). Wichtig ist, daß dieser Vorgang für den Anwendungsentwickler vollkommen transparent abläuft – er sieht auf Rechner R_2 die Slots des Graphen auf Rechner R_1 und kann wie gewohnt diese Slots über Routes mit seinen lokalen Slots verbinden.

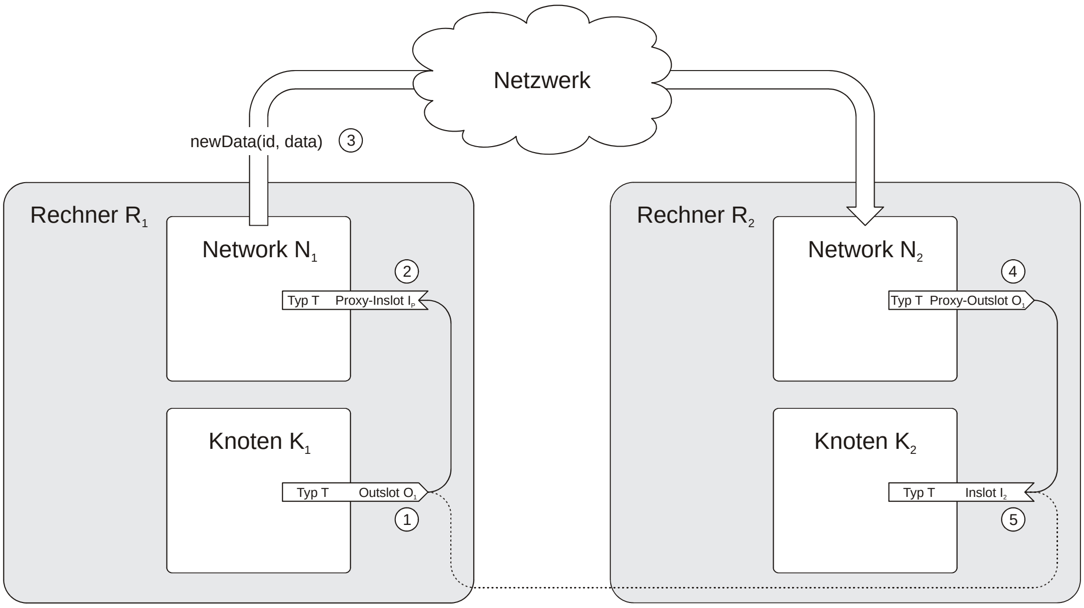


Abbildung 25: Daten werden vom Outslot auf Rechner R_1 zum Inslot auf Rechner R_2 übertragen

Abbildung 26 zeigt nun umgekehrt, was passiert, wenn die letzte Route vom Proxy-Outslot O_1 auf Rechner R_2 zu einem Inslot entfernt wurde ①. In diesem Fall sendet der Network-Knoten N_2 auf Rechner R_2 eine „stopOutslot“-Nachricht ② an den Network-Knoten N_1 auf Rechner R_1. Dieser zählt den Reference-Counter für den Proxy-Inslot I_P runter. Wenn der Reference-Counter Null wird, wenn also kein anderer Rechner im Netzwerk mehr Interesse am Outslot O_1 hat, entfernt der Network-Knoten den Proxy-Inslot I_P wieder.

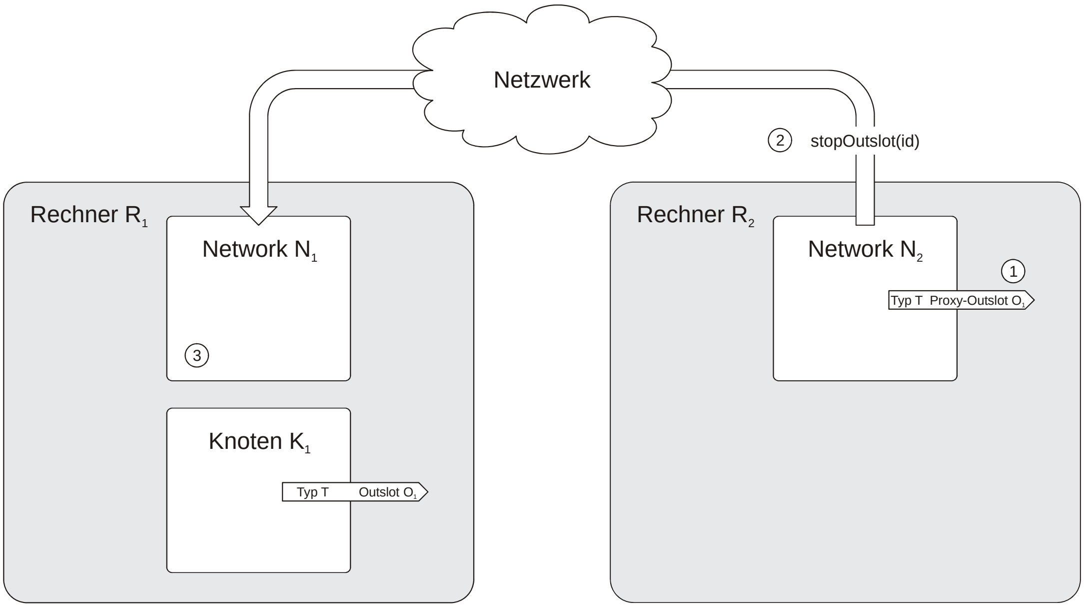


Abbildung 26: Die letzte Route vom Proxy-Outslot O_2 auf Rechner R_2 zu einem Inslot wurde entfernt

Abbildung 27 zeigt schließlich, was passiert, wenn der Knoten K_1 mit dem Outslot O_1 wieder vom Rechner R_1 entfernt wird. Der Network-Knoten N_1 auf Rechner R_1 ① sendet eine „removeOutslot“-Nachricht ② an alle anderen Network-Knoten, mit denen er verbunden ist. Diese Knoten entfernen den Proxy-Outslot O_1 ③. Fall der Proxy-Knoten noch mit Inslots verbunden war, wird die Verbindung einfach getrennt, genau wie es auch im Falle einer lokalen Verbindung passieren würde.

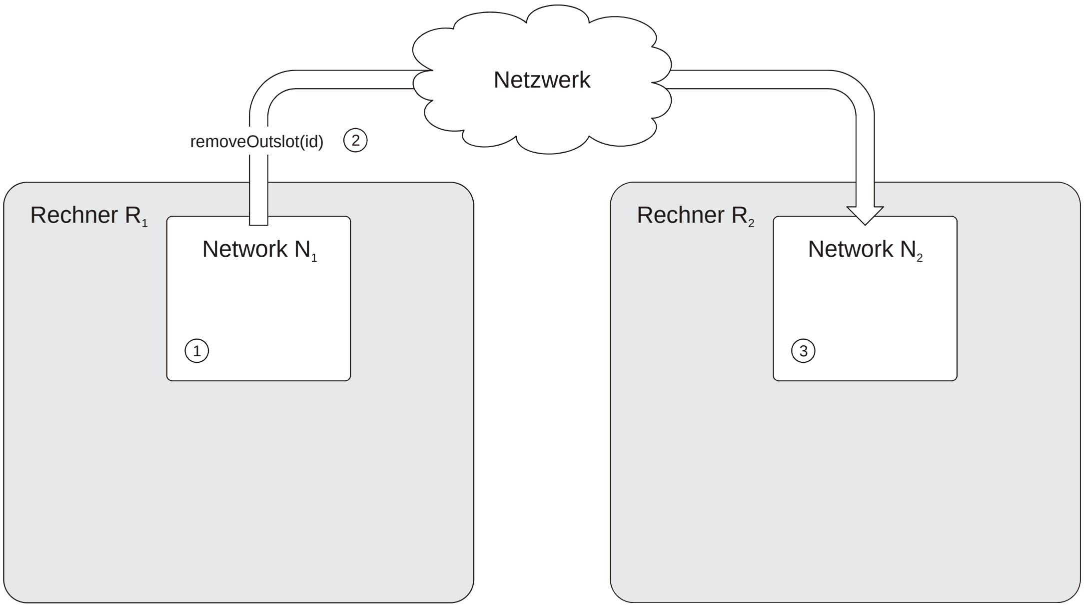


Abbildung 27: Der Knoten K_1 mit dem Outslot O_1 wird von Rechner R_1 entfernt

Der für die Outslots beschriebene Mechanismus läuft so auch analog für Inslots ab. Ausgangspunkt ist wieder die in Abbildung 22 dargestellte Situation von über das Netzwerk miteinander verbundenen Network-Knoten, die Bestandteil von Datenflußgraphen auf verschiedenen Rechnern sind.

Wird nun ein Knoten K_1 ① mit einem Inslot I_1 dem Graphen auf Rechner R_1 hinzugefügt (siehe Abbildung 28), so sendet der Network-Knoten N_1 eine „addInslot“-Nachricht ② an alle anderen Network-Knoten, mit denen er verbunden ist, in unserem Fall also an den Network-

Knoten N_2 auf Rechner R_2. Die „addInslot“-Nachricht enthält neben dem Namen I_1 des Inslots auch den Typ T der Daten, die über diesen Inslot empfangen werden können. Der Network-Knoten N_2 fügt sich selbst daraufhin einen Inslot zu ③, der den gleichen Namen und Datentyp hat. Dieser Inslot fungiert nun als Proxy-Inslot für den Original-Inslot auf Rechner R_1.

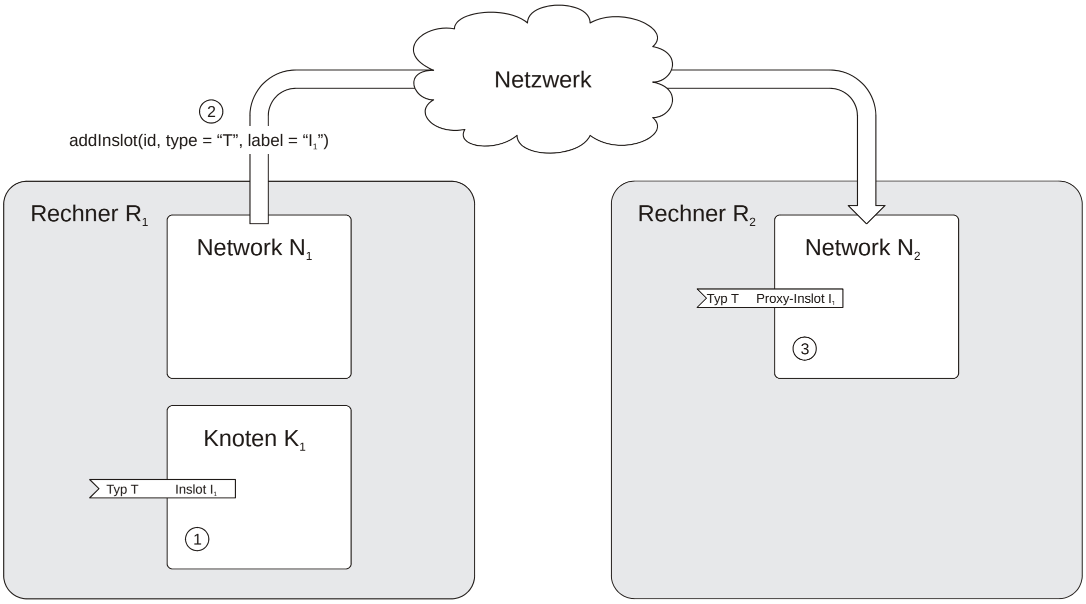


Abbildung 28: Ein Knoten mit einem Inslot wird dem Graphen auf Rechner R1 hinzugefügt

Outslots können nun mit dem Inslot I_1 auf Rechner R_1 verbunden werden, indem man sie mit dem jeweiligen lokalen Proxy-Inslot verbindet. Abbildung 29 zeigt einen Knoten K_2 mit einem Outslot O_2, der dem Graphen auf Rechner R_2 hinzugefügt wurde ①. Der Outslot O_2 wurde mit dem Proxy-Inslot I_1 verbunden. Der Network-Knoten N_2 schickt daraufhin eine „startOutslot“-Nachricht ② an den Network-Knoten N_1. Dieser zählt einen Reference-Counter hoch. Wenn der Counter 1 wird, fügt der Network-Knoten N_1 sich selbst einen Proxy-Outslot O_P hinzu und verbindet ihn über eine Route mit dem Inslot I_1 ③. Auf diese Weise wird über die physikalischen Verbindungen mit den Proxy-Slots die gewünschte logische Verbindung zwischen Outslot O_2 auf Rechner R_2 und Inslot I_1 auf Rechner R_1 hergestellt (in der Abbildung gepunktet dargestellt).

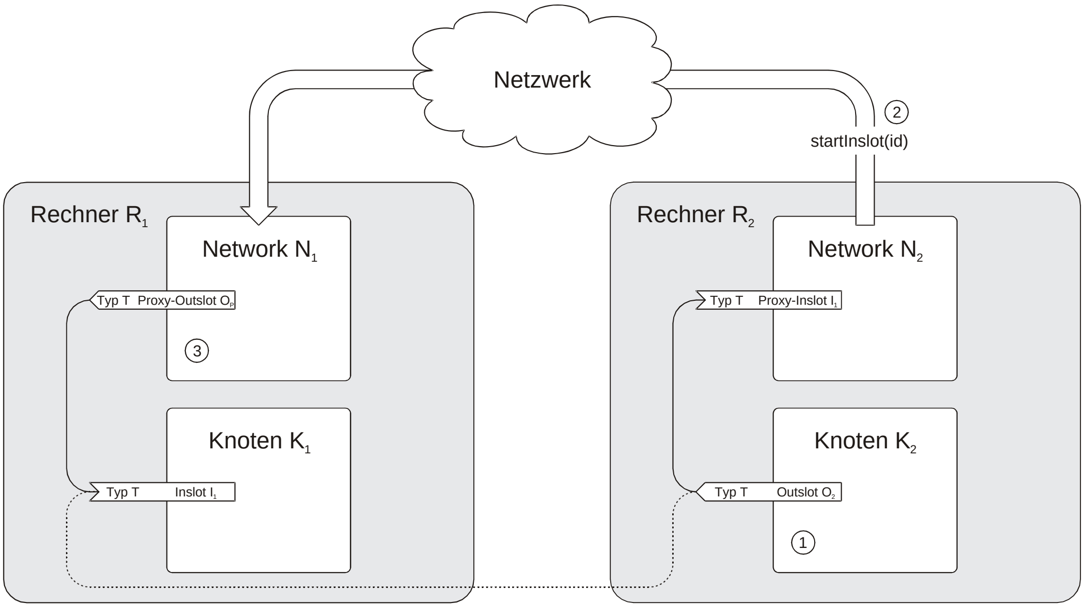


Abbildung 29: Ein Outslot auf Rechner R_2 verbindet sich mit dem Proxy-Inslot

Werden nun Daten über den Outslot O_2 verschickt ①, so empfängt sie der Network-Knoten N_2 über den Proxy-Inslot I_1 ② (siehe Abbildung 30). Die Daten werden vom Network-Knoten N_2 in eine Hardware- und Betriebssystem-unabhängige Form gebracht, abhängig vom Typ der Daten optional komprimiert und über eine „newData“-Nachricht ③ an den Network-Knoten N_1 geschickt. Der Network-Knoten N_1 dekomprimiert und dekodiert die Daten wieder und schickt sie über den Proxy-Outslot O_P an den Inslot I_1.

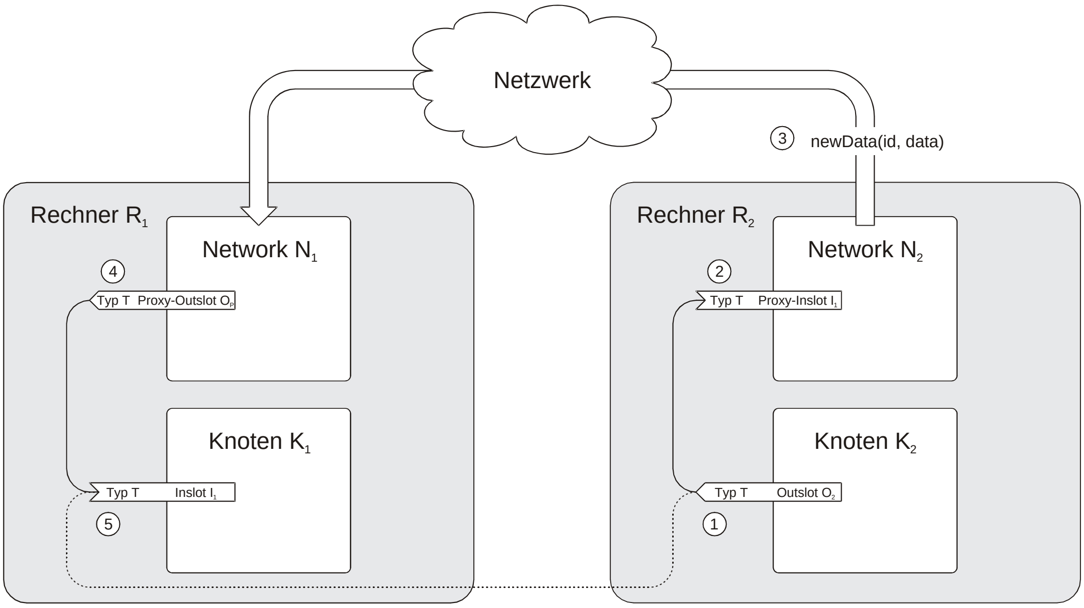


Abbildung 30: Daten werden vom Outslot O_2 auf Rechner R_2 zum Inslot I_1 auf Rechner R_1 übertragen

Wenn die letzte Verbindung eines Outslots mit dem Proxy-Inslot I_1 auf Rechner R_2 wieder getrennt wird ①, sendet der Network-Knoten N_2 eine „stopInslot“-Nachricht ② an den Network-Knoten N_1 auf Rechner R_1 (siehe Abbildung 31). Dieser zählt den Reference-Counter runter. Wenn der Counter 0 erreicht, gibt es keinen Outslot im Netzwerk mehr, der mit dem Inslot I_1 verbunden ist. In diesem Fall entfernt der Network-Knoten N_1 den Proxy-Outslot O_P wieder ③.

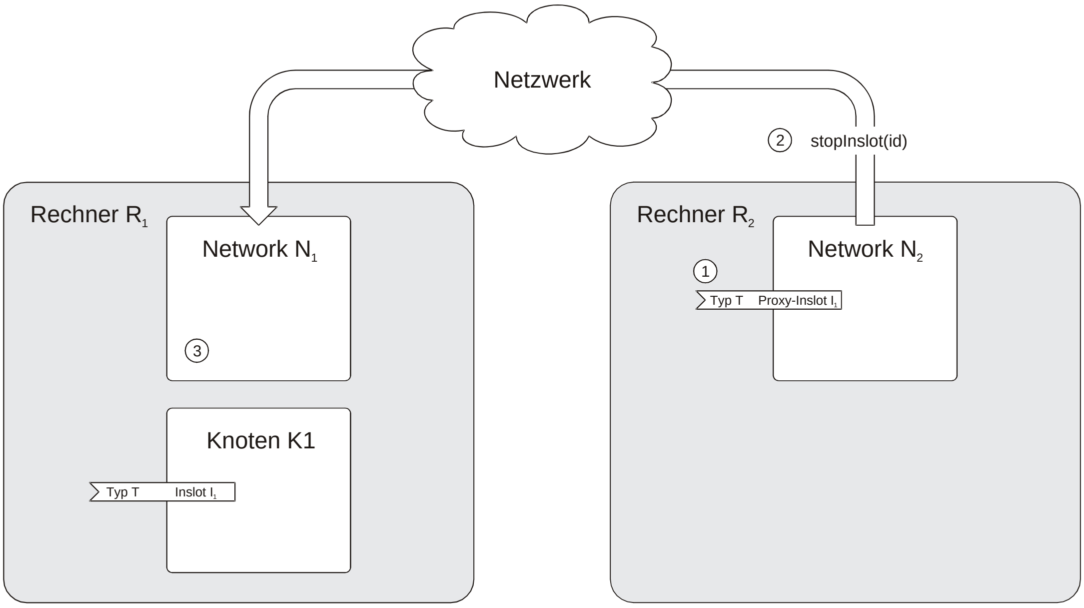


Abbildung 31: Die letzte Route von einem Outslot zum Proxy-Inslot I_1 auf Rechner R_2 wurde entfernt

Falls der Knoten K_1 mit dem Inslot I_1 wieder von Rechner R_1 entfernt wird ①, sendet der Network-Knoten N_1 eine „removeInslot"-Nachricht ② an alle Network-Knoten, mit denen er verbunden ist (siehe Abbildung 32). Diese Network-Knoten entfernen den Proxy-Inslot I_1 wieder ③. Falls der Proxy-Inslot zu dieser Zeit noch mit Outslots verbunden sein sollte, wird diese Verbindung einfach getrennt, genau so, wie es bei lokalen Verbindungen passieren würde.

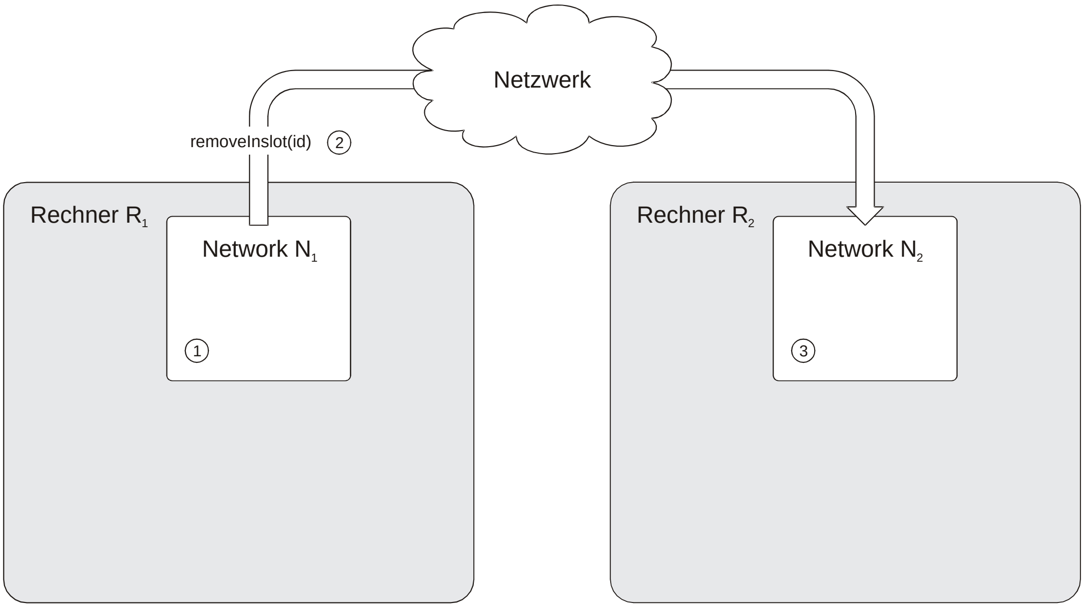


Abbildung 32: Der Knoten K_1 mit dem Inslot I_1 wird wieder von Rechner R_1 entfernt

Um die Datenwerte über das Netzwerk zu übertragen, müssen sie in einen Bytestrom umgewandelt, über das Netzwerk verschickt, und wieder aus dem Bytestrom rekonstruiert werden („Serialisierung" und „Deserialisierung"). Wie oben bereits erwähnt, muß es dazu für jeden Datentypen, der verschickt werden soll, einen passenden „Codec" geben. Für die Standard-Datentypen des Systems sind bereits Codecs implementiert, für anwendungsspezifische Datentypen können neue Codecs implementiert und im System registriert werden.

Codecs bestehen aus Paaren von „Encodern“ und „Decodern“. Für jeden Proxy-Inslot erzeugt der Network-Knoten einen Encoder, der die vom Inslot empfangenen Datenwerte in einen Bytestrom umwandelt. Analog dazu erzeugt der Network-Knoten für jeden Proxy-Outslot einen Decoder, der den Bytestrom wieder in Datenwerte umwandelt und über den Outslot verschickt.

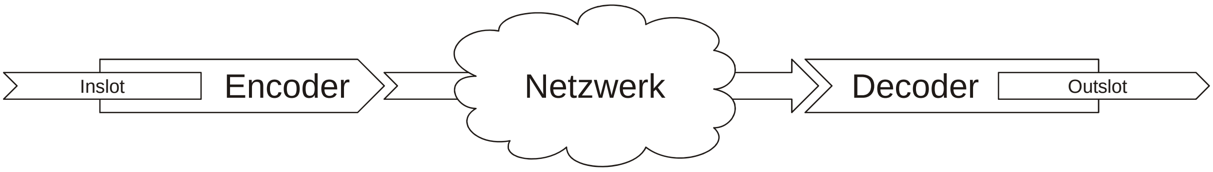


Abbildung 33: Serialisierung und Deserialisierung von Datenwerten über Encoder und Decoder

Standardmäßig werden die Datenwerte nicht komprimiert und zusammen mit der „newData“-Nachricht über eine TCP-Netzwerkverbindung verschickt. Diese Vorgehensweise reicht für kleinvolumige Datenwerte wie z.B. Positions- und Orientierungswerte von Trackern vollkommen aus. Für großvolumige Daten wie z.B. Videoströme ist diese Form der Übertragung nicht geeignet. Dafür bietet die verwendete Codec-Architektur zwei Lösungen:

- Datenwerte können komprimiert werden – da es eine 1-zu-1-Zuordnung gibt zwischen Inslot und Encoder bzw. Outslot und Decoder, können Encoder und Decoder Statusinformationen speichern. So kann der Encoder z.B. neue Datenwerte mit vorherigen vergleichen und nur Änderungen übertragen. Der Decoder kann dann die Änderungen in die vorher empfangenen Datenwerte einfügen, um die neuen Datenwerte zu rekonstruieren. In der Praxis werden auf diese Weise Video- und Audio-Ströme mit Hilfe von frei verfügbaren Video- und Audio-Codecs vor der Übertragung vom Encoder komprimiert, übertragen und vom Decoder wieder dekomprimiert. Dies hilft, die Menge der übertragenen Daten drastisch zu verringern.
- Die Datenübertragung per „newData“-Event hat den Nachteil, daß die Daten über TCP verschickt werden. TCP ist eine Punkt-zu-Punkt-Verbindung, d.h. bei n Empfängern müssen die Daten n-mal verschickt werden, was bei großvolumigen Daten natürlich inakzeptabel ist. Aus diesem Grund können Encoder und Decoder optional auch andere Übertragungsprotokolle verwenden, z.B. Multicast, um die Daten direkt miteinander auszutauschen. In diesem Fall werden die Daten nur einmal für alle Empfänger gleichzeitig über das Netzwerk übertragen.

Im Endergebnis erlaubt es das hier beschriebene Verfahren mit den Network-Knoten und den Proxy-Slots, völlig transparent für den Anwendungsentwickler Verbindungen zwischen Teilgraphen aufzubauen, die sich auf verschiedenen Rechnern befinden. Dabei werden keine Knoten dupliziert, sondern nur die Schnittstellen der Knoten (d.h. Outslots & Inslots), über die sie mit ihrer Umgebung kommunizieren. Netzwerktransparenz ist dabei kein eigentlicher Bestandteil des Gerätemanagementsystems, sondern wird über einen speziellen Knotentyp hergestellt, dem „Network“-Knoten. Diese Trennung erlaubt es problemlos, andere Formen von Netzwerktransparenz für andere Anwendungsfälle zu implementieren und parallel zueinander einzusetzen. Aufgrund der Tatsache, daß die Network-Knoten nur diejenigen Slots exportieren, die sich mit ihnen im gleichen Namespace befinden, kann der Anwendungsentwickler sehr genau steuern, welche Teile des Datenflußgraphen er im Netzwerk veröffentlichen will und welche Teile verborgen bleiben.

4.3.3 Kommunikation mit anderen Anwendern

Bei mobilen AR-Systemen gibt es sehr viel stärker als bei stationären Systemen die Anforderung, daß der Anwender mit anderen Anwendern kommunizieren kann – Stichwort

Computer Supported Cooperative Work (CSCW). Beispiele sind Edutainment-Anwendungen oder AR-Spiele, die erst durch das Gemeinschaftserlebnis in der Gruppe ihren Reiz erhalten, oder das klassische AR-Szenarios des Wartungstechnikers, der Unterstützung durch einen Remote-Experten erhält.

Interessanterweise wird dieser Aspekt des AR-Systems bereits zum Teil durch das soeben beschriebene netzwerktransparente Gerätemanagement-System abgedeckt. Es ist völlig irrelevant, ob über dieses Verfahren Geräte, Softwarekomponenten oder eben Anwender miteinander kommunizieren.

Wie wird z.B. die Kommunikation der Wartungstechnikers und des Experten beim Remote-Experten-Szenario realisiert? Zunächst einmal benötigt der Experte das aktuelle Bild, das der Wartungstechniker sieht, sowie die Trackingergebnisse. Dazu verbindet er einfach den Datenflußgraphen seines eigenen Systems über das oben beschriebene Verfahren mit dem auf dem Rechner des Wartungstechnikers. Falls auch ein Sprachkanal zwischen Experte und Techniker aufgebaut werden soll, werden einfach auf beiden Seiten Knoten zum Aufnehmen und Abspielen von Audiosignalen erzeugt und über Kreuz miteinander verbunden. Genaueres dazu im Anwendungsbeispiel „MARIO“ im 7. Kapitel.

Natürlich stellt das Gerätemanagement-System selber kein komplettes CSCW-System dar. CSCW ist ein sehr komplexes Thema mit einer Vielzahl von noch ungelösten Problemen und Forschungsthemen. Die netzwerktransparente Kommunikation zwischen AR-Systemen stellt nur die Grundlage für solche Anwendungen bereit – der Fokus des in dieser Arbeit vorgestellten Systems liegt aber nicht auf der speziellen Problematik von CSCW-Anwendungen.

4.4 Zusammenfassung

In diesem Kapitel wurde untersucht, durch welche Technologien und Methoden das in dieser Arbeit vorgestellte Rahmensystem für mobile AR-Systeme mit anderen Geräten, Softwarekomponenten und Anwendern im Netzwerk kommunizieren kann. Dabei wurden im Einzelnen die folgenden Fragestellungen betrachtet:

- Wie kann sich der Anwender an einem ihm fremden Ort in ein bestehendes Netzwerk einklinken, und wie kann er die in diesem Netzwerk angebotenen Dienste lokalisieren? Es wurde festgestellt, daß die Verbindungsaufnahme mit Netzwerken inzwischen in allen verbreiteten Betriebssystem zufriedenstellend gelöst ist. Für das Auffinden von Diensten hat sich dagegen noch kein Standardverfahren durchgesetzt. Es wurde ein Vergleich der drei existierenden Verfahren UPnP, MDNS und SLP durchgeführt und MDNS für die Implementierung des Rahmensystems ausgewählt, weil es am besten für dynamische Netzwerke geeignet ist, die verwendeten Kommunikationsprotokolle nicht einschränkt und für alle wichtigen Betriebssystemplattformen in freien Implementierungen vorliegt.
- Wie kann die AR-Laufzeitumgebung die für die Anwendung benötigten Daten von Servern herunterladen? Dafür stellt der VRML/X3D-Standard schon Mechanismen zur Verfügung, die es erlauben, 3D-Objekte, Texturen, Videos, Audio-Dateien und Anwendungsskripte über Standard-Kommunikationsprotokolle wie HTTP und FTP von Servern herunterzuladen. Falls weitergehende Mechanismen benötigt werden, kann auf die umfangreiche Java-Klassenbibliothek zurückgegriffen werden. Für den Einsatz von Javascript wurde das von HTML-Seiten bekannte Ajax-Verfahren mit dem neuen Javascript-Objekt „XMLHttpRequest“ übernommen und für den Einsatz in VRML/X3D portiert.

- Wie kann die AR-Laufzeitumgebung für die Anwendung transparent auf Geräte im Netzwerk zugreifen? Es wurden zwei mögliche Verfahren betrachtet, um Graphen im Netzwerk zu verteilen. Das eine Verfahren besteht darin, die Graphen auf allen beteiligten Rechnern im Netz komplett zu replizieren und dann synchron zu halten. Dies hat insbesondere für Mehrbenutzer- und CSCW-Anwendungen seine Vorteile, ist aber für den Einsatz in Gerätemanagement-Systemen extrem uneffizient. Daher wurde ein anderes Verfahren ausgewählt, bei dem es möglich ist, Verbindungen (Kanten/Routes) zwischen Knoten aufzubauen, die sich in verschiedenen Teilgraphen auf verschiedenen Rechnern im Netz befinden.
- Wie kann der Anwender mit anderen Anwendern kommunizieren? Das Gerätemanagement-System stellt dadurch, daß es nicht nur Geräte unterstützt, sondern auch ganz generell die netzwerktransparente Kommunikation zwischen verschiedenen Softwarekomponenten, eine Grundlage für CSCW-Anwendungen dar.

Im folgenden Kapitel wird nun betrachtet, wie das AR-Rahmensystem in den bestehenden VRML/X3D-Browser „Avalon“ integriert wird, und welche Erweiterungen des VRML/X3D-Standards dazu notwendig sind.

5 Rendering

Als Basis für das Rendering wird, wie im 2. Kapitel beschrieben, VRML [46] bzw. dessen Nachfolger X3D [47] verwendet, ein Standard, der es erlaubt, plattformunabhängige 3D-Anwendungen für das Internet zu entwickeln. Der Hauptfokus von VRML liegt bis heute auf der Entwicklung von Web-Anwendungen, d.h. es ist darauf ausgerichtet, VR-Anwendungen über Browser-Plugins in Webbrowsern darzustellen. Nichtsdestotrotz gibt es schon seit dem Beginn der VRML-Spezifizierung Bestrebungen, diesen Standard auch für immersive VR- und AR-Anwendungen einzusetzten. Folgende Kernfragen sind dabei zu klären:

- Wie bindet man die für immersive Anwendungen typischerweise verwendeten Ein- und Ausgabegeräte in den VRML-Standard ein? Aufgrund seiner Ausrichtung auf Web-Anwendungen unterstützt VRML von Haus aus nur Standard-Hardware, wie man sie an einem normalen Arbeitsplatzrechner findet, also 2D-Darstellung, Soundausgabe sowie Tastatur und Maus als Eingabegeräte. Zwar lassen sich die gegenwärtig verfügbaren Interaktionsparadigmen für immersive 3D-Anwendungen „aufbohren“ [49], aber die grundsätzliche Beschränktheit von VRML bezüglich der Anbindung von externen Geräten bleibt bestehen und behindert bis heute den Einsatz von VRML in immersiven VR- und AR-Anwendungen. Der folgende Abschnitt 5.1 schlägt Erweiterungen des VRML/X3D-Standards vor, die diese Einschränkung beseitigen.
- Wie implementiert man ein effizientes Rendering von Videoströmen? Bei AR-Systemen, die auf dem Video-see-through-Ansatz basieren, nimmt eine Videokamera (bzw. bei Stereosystemen sogar zwei) die Umgebung in Blickrichtung des Anwenders auf. Das Videobild wird dann in den Hintergrund der virtuellen AR-Szene gerendert, um die Illusion eines transparenten Displays/HMDs zu erzeugen. Typischerweise müssen dabei die Videobilder auch noch entzerrt werden, um Verzeichnungen der Kamera bei der Aufnahme auszugleichen. Der folgende Abschnitt 5.2 beschäftigt sich mit der Frage, wie diese Live-Videoströme effizient von einem VRML-Renderingsystem dargestellt werden können.

5.1 *Einbindung des Device Managements*

In diesem Abschnitt wird eine Erweiterung des VRML-/X3D-Standards vorgeschlagen, die es erlaubt, Ein- und Ausgabegeräte in VRML-Anwendungen einzubinden. Zunächst werden die bereits existierenden Schnittstellen und alternative Erweiterungsvorschläge betrachtet und ihre Stärken und Schwächen diskutiert. Auf dieser Grundlage werden sogenannte Low-Level-Sensoren vorgestellt, die eine Anbindung externer Geräte erlauben, ohne die Designschwächen anderer Vorschläge zu übernehmen.

5.1.1 Existierende Schnittstellen zu externen Geräten

VRML bzw X3D bieten in der aktuellen Spezifikation nur sehr eingeschränkte Möglichkeiten, externe Geräte anzusprechen. Das hängt primär mit der sehr speziellen Ausrichtung von VRML auf 3D-Web-Applikationen zusammen. Typischerweise werden VRML-Applikationen über ein Plugin im Internet-Browser dargestellt. Der Anwender sitzt an einem normalen Desktop-PC, klickt auf einen Link in seinem Browserfenster und bekommt eine 3D-Welt auf dem Bildschirm dargestellt, in der er mit Maus und Tastatur interagieren kann. Dabei sind die Interaktionsmöglichkeiten momentan stark an Walk-Through-Szenarien ausgerichtet, d.h. man kann sich durch die Welt bewegen und bestimmte Objekte anklicken, um sie zu bewegen oder Aktionen auszulösen.

Folgende Knoten erlauben gegenwärtig im VRML-Standard die Anbindung von Geräten:

- Der „NavigationInfo“-Knoten erlaubt es, die Navigation in der 3D-Welt zu steuern. Der Standard spezifiziert drei Navigationstypen: „Walk“, „Fly“ und „Examine“. „Walk“ erlaubt es, durch die 3D-Welt zu gehen bzw. zu fahren. Dabei folgt man dem Gelände, wobei die negative Y-Koordinatenachse nach unten zeigt. „Fly“ funktioniert ähnlich wie „Walk“, nur ist man hier nicht an das Gelände gebunden, sondern kann sich auch in der Y-Achse frei bewegen. „Examine“ erlaubt es, ein Objekt zu betrachten, d.h. man bewegt sich um ein Objekt herum.
- Die Pointing-Sensoren („CylinderSensor“, „PlaneSensor“, „SphereSensor“ und „TouchSensor“) erlauben es, Objekte in der virtuellen Welt anzuklicken und zu bewegen oder Aktionen auszulösen.

Der VRML-Standard legt nicht fest, wie genau (d.h. mit welchen Geräten) man die Navigation steuert bzw. wie man Objekte anklickt. Üblicherweise verwenden VRML-Browser die Cursortasten oder die Maus, um sich durch die Welt zu bewegen, und die Maus, um Objekte anzuklicken. Anstelle dieser traditionellen Eingabegeräte könnte man auch andere Geräte wie zum Beispiel Tracker verwenden, um die Applikation über die Standard-VRML-Knoten zu steuern. Nichtsdestotrotz erlaubt der VRML-Standard nur eine sehr beschränkte Einbindung von Eingabegeräten, und Ausgabegeräte (abgesehen von Bildschirm und Soundkarte) werden überhaupt nicht unterstützt.

Im neueren X3D-Standard hat sich die Situation nicht grundlegend verbessert:

- Die Keyboard-Sensoren („KeySensor“ und „StringSensor“) erlauben die Abfrage von Tastendrücken auf der Tastatur bzw. die Eingabe von ganzen Textstrings.
- Die „Distributed Interactive Simulation“ (DIS) Komponente erlaubt es, Nachrichten auszutauschen, die im DIS-Standard [84] spezifiziert sind. Der DIS-Standard spezifiziert das Zusammenspiel von verschiedenen, im Netzwerk verteilten Komponenten in Simulationssystemen und konzentriert sich insbesondere auf militärische Simulationen. Die in der DIS-Komponente spezifizierten X3D-Knoten erlauben es, die Position und Orientierung von Objekten im Raum zu kontrollieren, sowie Parameter, Audio-Sample-Daten und beliebige andere Datenströme mit anderen Softwarekomponenten im Netzwerk auszutauschen. Auch wenn der Zweck der DIS-Knoten darin besteht, das Rendering-System in Simulationsnetzwerke einzubinden, wären sie doch prinzipiell dazu geeignet, ein Gerätemanagementsystem anzubinden.

Zusammenfassend kann man feststellen, daß die Einbindung von Geräten in VRML bzw. X3D bislang nur sehr unzulänglich ist. Die vorhandenen Möglichkeiten erlauben praktisch nur einfache Walk-Through-Anwendungen unter Verwendung von Maus, Tastatur, Joystick oder Spacemouse. Natürlich erlaubt es der VRML-Script-Knoten, beliebigen Java-Code in den Szenengraphen einzubinden und damit alle Möglichkeiten zu nutzen, die Java bietet, um auf Geräte zuzugreifen. Ein modernes Renderingsystem sollte jedoch nicht auf solche Hilfskonstrukte angewiesen sein, sondern eigenständige Schnittstellen zu externen Geräten anbieten.

5.1.2 Andere Ansätze, Geräte in VRML/X3D einzubinden

Die schlechte Anbindung von Ein- und Ausgabegeräten ist schon lange ein offensichtlicher Hemmschuh, der die Entwicklung von VRML-Anwendungen insbesondere im professionellen Umfeld behindert. Daher gibt es bereits Vorschläge, wie man dieses Manko beseitigen kann. Eine gute Übersicht in die Problematik findet man in [85]. Die folgenden Abschnitte geben eine Übersicht über diese Erweiterungsvorschläge und diskutieren ihre Vor- und Nachteile.

5.1.2.1 Blaxxun / Bitmanagement Contact

Der von Blaxxun bzw. Bitmanagement vertriebene VRML-Browser „Contact“ besitzt einen sogenannten „DeviceSensor“-Knoten [85]. Dieser Knoten hat folgendes Interface:

Interface des im „Contact“-Browser implementierten DeviceSensor-Knotens

```
DeviceSensor {
  SFString [in,out] device
  SFString [in,out] eventType
  SFNode   [in,out] event
}
```

Das „device“-Feld spezifziert den zu verwendenden Gerätetreiber, z.B. „JOYSTICK“, „MOUSE“ oder „SPACEMOUSE“. Falls von einem Gerätetyp mehr als eine Instanz vorhanden ist, kann über eine angehängte Nummer festgelegt werden, welche Instanz verwendet werden soll, z.B. „JOYSTICK 2“. Das „eventType“-Feld erlaubt es, Geräte-spezifische Parameter an den Gerätetreiber zu übergeben. Das „event“-Feld enthält einen Geräte-spezifischen Knoten-Prototypen, der Out- bzw. Inslots für alle vom Gerät bereitgestellten Sensoren enthält. Für einen Joystick könnte dieser Prototyp z.B. folgendermaßen aussehen:

Prototypen-Deklaration für einen Joystick-Knoten

```
PROTO Joystick [
  eventOut SFVec2f stick
  eventOut SFBool button1
  eventOut SFBool button2
] {}
```

Im VRML-Code wird dann einfach eine Instanz des DeviceSensor-Knotens erzeugt, der eine Instanz des Joystick-Prototypen in seinem event-Feld besitzt. Die Slots des Joystick-Knotens werden dann über Routes mit den Slots anderer VRML-Knoten verbunden:

Beispiel für den Einsatz des DeviceSensor-Knotens

```
…
DEF controller Script { … }

DEF DS DeviceSensor {
  device "JOYSTICK"
  event DEF JS JoyStick {}
}
ROUTE JS.button1 TO controller.set_button1
…
```

5.1.2.2 Xj3D[12]

Xj3D ist ein frei verfügbarer, Open-Source X3D-Browser. Er dient als Referenzimplementierung des X3D-Standards. Zur Anbindung von Geräten besitzt er acht Browser-spezifische Knoten. Ein Knoten mit dem Namen „DeviceManager“ dient dazu, Informationen über die an den Rechner angeschlossenen Geräte zu liefern. Die anderen sieben Knoten mit den Namen „GamepadSensor“, „GenericHIDSensor“, „JoystickSensor“, „MidiSensor“, „MouseSensor“, „TrackerSensor“ und „WheelSensor“ dienen dazu, die entsprechenden Geräte in die X3D-Anwendung einzubinden. Jeder dieser Knoten hat ein SFString-Feld mit dem Namen „name“, mit dem man das zu verwendende Gerät spezifizieren kann, z.B. „Gamepad-0“ für das erste an das System angeschlossene Gamepad. Ein ähnlicher Ansatz wird auch in [87] vorgeschlagen.

---

[12] http://www.xj3d.org/

5.1.2.3 Bewertung der bisher vorgeschlagenen Erweiterungen

Leider muß man feststellen, daß sowohl die im Contact-Browser als auch die im Xj3D-Browser implementierte Geräteanbindung entscheidende Schwächen hat.

Zunächst einmal fällt einem bei der Xj3D-Lösung auf, daß es ungeschickt ist, Knoten für spezifische Geräte zu entwerfen. Mit jedem neuen Gerätetyp, den man an das System anschließen will, muß man einen neuen Knoten entwerfen. Egal, wieviele Knoten man entwirft, man wird immer ein Gerät finden, das nicht in die vorhandenen Gerätetypen paßt. De facto explodiert die Anzahl der Knoten, ohne daß man das Problem wirklich löst.

Betrachtet man die Contact- und die Xj3D-Lösungen, so fällt einem außerdem auf, daß sie beide den gleichen, grundlegenden Designfehler haben: Es wird in den VRML-Knoten direkt spezifiziert, auf welche konkrete Hardware zugegriffen werden soll. Aber wie soll der Entwickler einer VRML-Anwendung wissen, welche Hardware beim zukünftigen Anwender vorhanden ist?

Das typische Einsatzszenario einer VRML-Anwendung ist, daß der Anwender in seinem Webbrowser auf einen Link klickt, woraufhin die Anwendung vom Webserver heruntergeladen und in einem VRML-Player-Plugin dargestellt wird. Die Hardware-Umgebung, die auf der Zielplattform der Anwendung vorhanden ist, ist also völlig unbekannt. Es kann sich um einen einfachen Arbeitsplatzrechner handeln, der mit einem Joystick zur Steuerung der VR-Anwendung ausgestattet ist. Es kann sich aber auch um eine CAVE handeln, in der über spezielle VR-Hardware wie z.B. eine Spacemouse die Position in der virtuellen Welt gesteuert wird.

Aus der Tatsache, daß dem Anwendungsentwickler die Hardwareausstattung der Zielplattform völlig unbekannt ist, folgt zwingend, daß er auch nicht in der Lage ist zu spezifizieren, welchen Typ von Gerät (z.B. Spacemouse oder Joystick) die Anwendung verwenden soll. Geschweige denn, welche Geräteinstanz verwendet werden soll, falls von einem Gerätetyp mehrere Instanzen an den Rechner angeschlossen sind.

Aufgrund dieser schwerwiegenden Schwächen wird im folgenden Abschnitt ein eigener Vorschlag zur Erweiterung des X3D-Standards gemacht. Dieser Vorschlag basiert auf den folgenden Designkriterien:

- Zur Anbindungen von Geräten sollte der X3D-Standard um möglichst wenige Knoten erweitert werden. Trotzdem sollten diese Knoten grundsätzlich ausreichend sein, um alle jetzt und zukünftig verfügbaren Geräte zu integrieren.
- Der Anwendungsentwickler ist nicht in der Lage vorherzusehen, welche Geräte auf der Zielplattform vorhanden sind. Daher kann er weder vorgeben, welche Gerätetypen verwendet werden sollen, noch, welche konkrete Instanz eines Gerätetyps verwendet werden soll. Es muß also eine Lösung gefunden werden, wie das „Mapping“ von am Rechner vorhandenen Geräten auf die von der Anwendung benötigten Geräte vorgenommen werden kann.

5.1.3 Erweiterung des VRML/X3D-Standards um Low-Level-Sensoren

Als größte Schwäche der existierenden VRML-Knoten erweist sich ihr hoher Abstraktionsgrad. Die Datenströme von externen Geräten werden vom System bereits mit einer Bedeutung belegt, also z.B. „Positionssteuerung/Navigation im Raum“ oder „Steuerung eines virtuellen Zeigers, mit dem Aktionen ausgelöst werden können“. Diese „High-Level-Sensoren“ decken sehr elegant typische Interaktionsparadigmen ab, die von VR-Anwendungen häufig verwendet werden. Natürlich könnte man jetzt versuchen, weitere High-Level-Sensoren zu entwerfen, um ein größeres Feld von externen Ein-/Ausgabegeräten in

VRML zu unterstützen. Letztendlich wäre dieses Vorhaben jedoch zum Scheitern verurteilt. Es ist schlicht und einfach unmöglich, Knoten für jedes existierendes oder zukünftiges Gerät zu entwerfen. Dies würde den Standard gewaltig aufblähen, und man würde immer noch Geräte finden, die nicht optimal unterstützt werden.

Erfolgversprechender ist daher der Ansatz, „Low-Level-Sensoren“ zu entwerfen, die Datenströme von und zu Geräten nicht mit einer vordefinierten Bedeutung belegen. Das Design ist dabei angelehnt an die von Araki vorgeschlagenen „Netnodes“ [88]. VRML/X3D definiert eine Menge von Basisdatentypen (SFBool, SFInt, SFFloat, SFVec3f etc.). VRML-Knoten können über Out- und Inslots miteinander kommunizieren und typisierte Datenströme miteinander austauschen, die aus diesen Basisdatentypen bestehen. Ein naheliegender Ansatz für den Entwurf von Low-Level-Sensoren ist es, für genau jeden VRML-Basisdatentypen einen korrespondierenden Sensorknoten bereitzustellen, der es erlaubt, Datenströme des jeweiligen Basisdatentyps über einen Inslot von anderen Knoten zu empfangen und an Ausgabegeräte weiterzuleiten, bzw. Datenströme von Eingabegeräten zu empfangen und über einen Out-Slot an andere Knoten zu senden. Das Interface eines Low-Level-Knotens sieht also folgendermaßen aus (die Interfacebeschreibung verwendet dabei das in der X3D-Spezifikation verwendete Format, „x“ steht dabei für einen der VRML-Basisdatentypen):

Interface der VRML-Low-Level-Sensoren

```
xSensor {
  SFString []       label       ""
  SFString []       description ""
  SFBool   []       out         FALSE
  x        [in,out] value
}
```

Das Interface eines Sensor-Knotens, der SFVec3f-Datenströme verarbeiten kann, sieht also so aus (ein solcher Knoten könnte z.B. die Kopfposition des Anwenders von einem Trackingsystem empfangen):

Interface des SFVec3f-Low-Level-Sensors

```
SFVec3fSensor {
  SFString []       label       ""
  SFString []       description ""
  SFBool   []       out         FALSE
  SFVec3f  [in,out] value
}
```

Die Sensor-Knoten enthalten vier Felder: Ein SFString-Feld „label“, das einen Namen für den Sensor-Knoten enthält, ein SFString-Feld „description“, das den Zweck des Sensors beschreibt, ein SFBool-Feld „out“, das die Richtung des Sensors festlegt (Ein- oder Ausgabe), sowie ein Exposed-Feld „value“, dessen Typ variiert und über das Datenwerte empfangen oder verschickt werden. Mehr zu diesen Feldern später.

Auf den ersten Blick wirkt diese Lösung vergleichsweise unelegant. Schließlich müßte für jeden der 38 im X3D-Standard definierten Basisdatentypen ein eigener Sensor-Knoten zur Verfügung gestellt, der Standard also um 38 Knoten erweitert werden. Tatsache ist jedoch, daß nur diese Lösung sicherstellt, daß alle denkbaren Ein- und Ausgabegeräte vom X3D-Standard unterstützt werden. X3D-Knoten kommunizieren über typisierte Datenströme, die aus 38 Basisdatentypen bestehen, mit ihrer Umwelt. Ein Set von 38 Sensor-Knoten, für jeden Datentyp einer, ermöglicht es also, die komplette Bandbreite von Kommunikationsformen an das Gerätemanagement-System weiterzuleiten. 38 neue Knoten sind zwar eine ganze Menge, aber dafür hat man das Problem der Anbindung externer Geräte zuverlässig gelöst und benötigt auch in Zukunft, wenn neue Gerätetypen auftauchen, keine neuen Sensor-Knoten. Der X3D-Standard wird also stabilisiert.

Ein Nachteil der hier präsentierten Lösung ist, daß Datenstöme, die zusammengehören, innerhalb der X3D-Szene über getrennte Sensor-Knoten empfangen werden. Ein Beispiel dafür sind Videoströme, bei denen die einzelnen Bilder zusammen mit den jeweils zugehörigen Videotrackingergebnissen in der Szene eintreffen müssen. Bei der aktuellen Implementierung des in dieser Arbeit vorgestellten AR-Systems wird das nicht durch das System garantiert – anstelle dessen besitzen Videotracking-Knoten im Gerätemanagement-System zusätzlich zu den Outslots für Position und Orientierung einen Outslot, auf dem sie am Ende des Trackingprozesses das verwendete Videobild zusätzlich zum Trackingergebnis verschicken, um den zeitlichen Abstand dieser Events und damit auch die Wahrscheinlichkeit von Race Conditions zu minimieren (siehe auch die Anwendungsbeispiele in Kapitel 7). In der Praxis führt das zu befriedigenden Ergebnissen. Eine andere denkbare Lösung wäre, das der Anwender über spezielle Gruppierungs-Knoten Sensoren zusammenfassen kann, die immer gleichzeitig feuern müssen, o.Ä.

5.1.4 Mapping der Low-Level-Sensoren auf konkrete Geräte / Datenströme

Die im vorhergehenden Abschnitt beschriebenen Low-Level-Sensor-Knoten erlauben es, beliebige Datenströme in die VRML-Applikation zu integrieren. Es wurde allerdings noch nicht spezifiziert, wie das Mapping zwischen den Low-Level-Sensoren auf der einen Seite und den Datenströmen des Gerätemanagement-Systems auf der anderen Seite funktioniert.

Wie in Abschnitt 5.1.2 gezeigt wurde, haben die bisher gemachten Vorschläge zur Einbindung von Geräten in X3D-Anwendungen alle den gleichen grundlegenden Designfehler: Sie fordern vom Anwendungsentwickler, ein konkretes Gerät auf der Zielplattform zu spezifizieren – eine Aufgabe, die er offensichtlich nicht lösen kann, weil er normalerweise die Zielplattform nicht kennt.

Wie kann man es nun besser machen? Zur Beantwortung dieser Frage ist es hilfreich, einen Blick auf das erfolgreichste Genre von VR-Anwendungen zu werfen, nämlich 3D-Computerspielen, wie z.B. den beliebten First-Person-Shootern.

Computerspiele standen schon immer vor dem Problem, wie die komplexen Bewegungsabläufe z.B. einer Spielfigur in einem First-Person-Shooter auf die beim Anwender vorhandene Hardware abgebildet werden. Viele Spieler verwenden zur Steuerung Tastatur oder Maus. Andere besitzen einen oder sogar mehrere Joysticks. Dabei müssen die Entwickler von Computerspielen beachten, daß die Anwender ihrer Spiele nicht unbedingt nur „Computerfreaks“ sind – sie müssen also einen relativ einfachen und intuitiven Weg bereitstellen, auf dem der Spieler das Spiel konfigurieren kann.

```
ENTER TO CHANGE, BACKSPACE TO CLEAR

ATTACK            ▶CTRL OR MOUSE1
CHANGE WEAPON      /
JUMP / SWIM UP     ENTER OR SPACE
WALK FORWARD       UPARROW OR MOUSE2
BACKPEDAL          DOWNARROW
TURN LEFT          LEFTARROW
TURN RIGHT         RIGHTARROW
RUN                SHIFT
STEP LEFT          ,
STEP RIGHT         .
SIDESTEP           OPTION
LOOK UP            A OR PGDN
LOOK DOWN          Z OR DEL
CENTER VIEW        END
MOUSE LOOK         \ OR MOUSE3
KEYBOARD LOOK      INS
SWIM UP            D
SWIM DOWN          C
```

Abbildung 34: Dialog zur Gerätekonfiguration in einem Computerspiel (Quake)

Üblicherweise besitzen Computerspiele einen speziellen Konfigurationsdialog (siehe Abbildung 34), der es erlaubt, Aktionen der Spielfigur auf die am Rechner vorhandene Hardware abzubilden. In diesem Dialog werden auf der einen Seite die Aktionen der Spielfigur aufgelistet („Vorwärts/rückwärts gehen“, „Links/rechts drehen“, „Schießen“ etc.). Neben den Aktionen wird angezeigt, durch welchen Sensor der vorhandenen Eingabehardware die jeweilige Aktion gesteuert wird („X-Achse von Joysticks 1“, „Button 1 von Joystick 2“, „Eingabetaste“ etc.). Der Anwender kann diese Zuordnung ändern, indem er eine Aktion markiert und entweder den gewünschten Sensor betätigt oder aus einer Liste einen passenden Sensor auswählt. Dabei ist zu beachten, daß sowohl Aktionen als auch Sensoren typisiert sind, d.h. man kann eine Aktion wie „Gehen“ nur auf eine analoge Achse abbilden, oder eine Aktion wie „Schießen“ nur auf einen Button.

Wichtig sind dabei folgende Erkenntnisse:

- Die Anwendung (also das Computerspiel) spezifiziert nicht, welcher konkrete Sensor an welchem Eingabegerät als Quelle für Datenwerte verwendet werden soll. Vielmehr spezifiziert die Anwendung, in welcher Weise die Datenwerte verwendet werden sollen, also z.B. um die Aktionen „Gehen“, „Drehen“ oder „Schießen“ zu kontrollieren. Die Abbildung von konkreten Sensoren auf die abstrakten Aktionen kann der Anwender (der Spieler) in einem Konfigurationsdialog festlegen.

- Darüber hinaus spezifiziert die Anwendung den Typ der Datenwerte und die Richtung des Datenflusses (Ein- oder Ausgabegerät). Eine Aktion wie „Gehen“ erfordert einen Eingabesensor, der skalare Werte liefert. Eine Aktion wie „Schießen“ erfordert dagegen einen digitalen Eingabesensor. Auf diese Weise wird sichergestellt, daß der Anwender im Konfigurationsdialog nur Sensoren auf Aktionen abbildet, die zueinander kompatibel sind.

Übertragen auf die VRML-Low-Level-Sensoren bedeutet das:

- Die Low-Level-Sensoren besitzen ein Text-Feld „label“. Anders als bei allen bisher vorgeschlagenen Lösungen zum Anbinden von Geräten an VRML-Anwendungen wird aber in diesem Feld nicht das konkrete Gerät bzw. der konkrete Sensor spezifiziert. Vielmehr beschreibt der Entwickler der Anwendung hier den Verwendungszweck der Datenwerte, also z.B. „Gehen“, „Drehen“ oder „Schießen“.

- Neben dem Verwendungszweck beschreiben die Low-Level-Sensoren auch den Typ der Datenwerte und die Richtung des Datenflusses. Der Typ der Datenwerte wird durch den verwendeten Knoten-Typ festgelegt („SFBoolSensoren“ verarbeiten „SFBool“-Werte, „SFFloatSensoren“ verarbeiten „SFFloat“-Werte). Die Richtung wird durch ein Boolean-Feld „out“ festgelegt. Wenn „out“ den Wert „TRUE“ hat, dient der Sensor dazu, Datenwerte von der Anwendung an ein Ausgabegerät zu schicken. Wenn „out“ dagegen den Wert „FALSE“ hat, dient der Sensor dazu, Datenwerte von einem Eingabegerät zu empfangen.

Startet der Anwender nun zum ersten Mal eine VRML-Anwendung, so wird er wie bei Computerspielen über einen Dialog dazu aufgefordert, die in der VRML-Anwendung vorhandenen Aktionen („Gehen“, „Schiessen“ etc.) auf konkrete, mit dem Rechner verbundene Geräte abzubilden („Joystick 1/Y-Achse“, „Joystick 1/Button #1“,...). Dieses Mapping kann dann in einem Cache gespeichert werden, damit es der Anwender nicht jedesmal neu spezifizieren muß, wenn er eine Anwendung erneut startet. Die URL der VRML-Anwendung dient dabei als Schlüssel, um eine gespeicherte Konfiguration im Cache zu finden.

5.1.5 Anwendungsbeispiel

Wie werden die Low-Level-Sensoren nun konkret vom Anwendungsentwickler eingesetzt? Betrachten wird dazu als Beispiel einen in VRML geschriebenen First-Person-Shooter. Der Spieler kann in der 3D-Welt vor und zurück gehen, er kann sich links oder rechts drehen, und er kann schießen, um seine Gegner im Spiel zu vernichten. Dazu definiert der Entwickler drei Low-Level-Sensoren im VRML-Code, nämlich zwei, die skalare Datenwerte empfangen und zur Steuerung der Bewegung in der 3D-Welt dienen, und einen, der digitale Datenwerte empfängt, die das Schießen auslösen. Diese Datenwerte werden an einen Script-Knoten weitergeleitet, die Aktionen des Spielers in der 3D-Welt kontrolliert. Der relevante Ausschnitt des VRML-Codes sieht folgendermaßen aus:

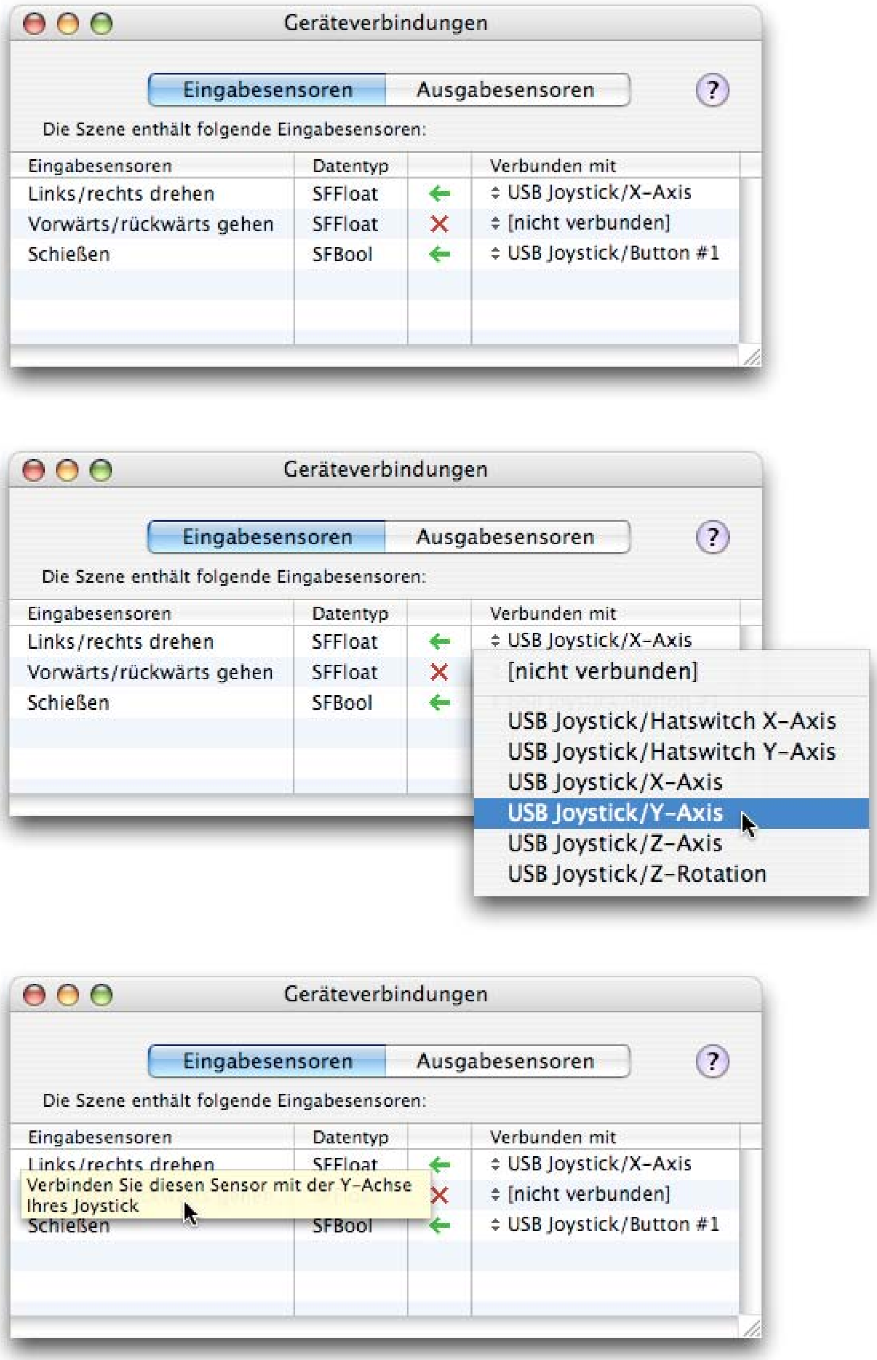
Ausschnitt aus dem VRML-Code eines First-Person-Shooters

```
…
DEF controller Script
{
  eventIn SFFloat rotate
  eventIn SFFloat walk
  eventIn SFBool shoot
  url ["controller.js"]
}

DEF rotateSensor SFFloatSensor
{
  label "Links/rechts drehen"
  description "Verbinden Sie diesen Sensor mit der X-Achse Ihres Joystick"
  out FALSE
}
ROUTE rotateSensor.value_changed TO controller.rotate

DEF walkSensor SFFloatSensor
{
  label "Vorwärts/rückwärts gehen"
  description "Verbinden Sie diesen Sensor mit der Y-Achse Ihres Joystick"
  out FALSE
}
ROUTE walkSensor.value_changed TO controller.walk

DEF shootSensor SFBoolSensor
{
  label "Schießen"
  description "Verbinden Sie diesen Sensor mit dem Feuerknopf Ihres
Joystick"
  out FALSE
}
ROUTE shootSensor.value_changed TO controller.shoot
…
```



Die Szene enthält folgende Eingabesensoren:

Eingabesensoren	Datentyp		Verbunden mit
Links/rechts drehen	SFFloat	←	USB Joystick/X-Axis
Vorwärts/rückwärts gehen	SFFloat	✗	[nicht verbunden]
Schießen	SFBool	←	USB Joystick/Button #1

Verbinden Sie diesen Sensor mit der Y-Achse Ihres Joystick

Abbildung 35: GUI zum Mappen von X3D-Sensoren auf konkrete Geräte

Der Spieler startet den First-Person-Shooter, indem er auf einen Link in seinem Webbrowser klickt. Daraufhin wird die VRML-Welt heruntergeladen und mittels eines VRML-Browsers dargestellt. Der VRML-Browser stellt beim Parsen des VRML-Codes fest, daß diese VRML-Welt drei Eingabe-Datenströme benötigt, zwei davon für SFFloat-Werte, und einer für SFBool-Werte. Daraufhin öffnet der Browser ein Dialogfenster. Abbildung 35 zeigt, wie der Dialog aussehen könnte, mit dem der Anwender das Mapping festlegt. Es gibt zwei Tabellen für die Eingabe- und Ausgabesensoren. In der ersten Spalte steht der Name des Sensors, wie er im „label“-Feld spezifiziert wurde, und in der zweiten Spalte der X3D-Datentyp, den er verarbeiten kann. Bewegt man den Mauszeiger über einen Namen in der ersten Spalte, so erhält man zusatzliche, ausführliche Informationen, die aus dem optionalen „description“-Feld des Sensors stammen. Über ein Icon in der dritten Spalte kann man erkennen, ob Sensor verbunden ist oder nicht. In der vierten Spalte steht schließlich, womit der Sensor verbunden ist. Durch Anklicken einzelner Einträge in dieser Spalte kann man ein Popup-Menü öffnen, das eine Auswahl der Gerätesensoren enthält, die man mit dem in der Szene vorhandenen Sensor verbinden kann. Dabei kann (wie bei Computerspielen) eine einfache Heuristik verwendet werden, die in den meisten Fällen automatisch eine passende Abbildung erzeugt. Und zwar sortiert man die Sensoren der an den Rechner angeschlossenen Geräte in sinnvoller Weise (also z.B. bei SFFloat-Eingabedatenströmen zuerst die Achsen des ersten Joysticks im System, dann die Achsen des zweiten Joysticks im System, etc.). Dann bildet man die Low-Level-Sensoren in der Reihenfolge, in der sie im VRML-Code stehen, auf die passenden Geräte-Sensoren ab. In unserem Beispiel würde das folgende, bereits völlig korrekte Standard-Abbildung erzeugen:

- Der erste SFFloatSensor für die Drehung in der 3D-Welt wird auf die erste Achse (die X-Achse) des ersten Joysticks im System abgebildet.
- Der zweite SFFloatSensor für das Gehen in der 3D-Welt wird auf die zweite Achse (die Y-Achse) des ersten Joysticks im System abgebildet.
- Der SFBoolSensor für das Schießen wird auf den ersten Button (den Feuer-Button) des ersten Joysticks im System abgebildet.

Nachdem der Spieler das gewünschte Mapping von Eingabehardware auf Spielaktionen festgelegt hat, kann er das Dialogfenster schließen und das Spiel beginnen. Der VRML-Browser speichert das festgelegte Mapping in einer Datenbank. Dabei wird die URL der VRML-Welt als Schlüssel verwendet, unter dem das Mapping abgelegt wird. Wenn der Spieler die VRML-Welt später erneut herunterlädt, kann der VRML-Browser feststellen, daß unter der URL bereits ein Mapping in der Datenbank abgelegt ist, und das frühere Mapping erneut verwenden.

5.2 Effizientes Rendern von Videoströmen

Eine der größten Herausforderung bei der Entwicklung von mobilen AR-Systemen ist die effiziente Handhabung von Videoströmen. Die Mehrzahl der gegenwärtig verfügbaren AR-Anwendungen verwendet die sogenannte „Video-see-through“-Technik, d.h. es werden keine transparenten Displays verwendet, sondern eine Kamera (oder bei Stereo-fähigen Systemen zwei) nimmt das Bild in Blickrichtung des Anwenders auf. Dieses Videobild wird in den Hintergrund der virtuellen AR-Szene gelegt, um die Illusion eines transparenten Displays zu erzeugen. Für den Einsatz der „Video-see-through“-Technik sprechen folgende Gründe:

- Die Bildqualität transparenter Displays ist beim gegenwärtigen Stand der Technik einfach zu schlecht. Während die virtuellen Bestandteile der Szene möglichst undurchsichtig sein müssen, sollte der Blick auf die reale Umgebung möglichst unbeeinträchtigt bleiben. In der Praxis lassen die meisten transparenten HMDs nur einen Bruchteil des Umgebungslichtes zum Auge des Betrachters durch, wirken also wie Sonnenbrillen. Darüber hinaus sind die virtuellen Bestandteile der Szene mehr oder weniger transparent, was zu einer „geisterhaften“ Darstellung führt. Eine Ausnahme bilden hier Laser-Retinal-Systeme, bei denen das Bild direkt auf die Netzhaut des Auges projeziert wird.
- Es ist sehr schwierig, „Optical-see-through“-Systeme zu kalibrieren. Bei AR-Systemen ist es entscheidend, daß die virtuellen Bestandteile möglichst korrekt in das Blickfeld des Anwenders eingeblendet werden. Bei „Video-see-through“-Systemen muß dazu nur die genaue Position des Displays bzw. genauer gesagt die genaue Position der Kamera bestimmt werden. Bei „Optical-see-through“-Systemen muß dagegen neben der Position des Displays im Raum auch noch die Position relativ zu

- Viele mobile AR-Systeme benutzen zur Positionsbestimmung ein optisches Trackingverfahren, weil andere gegenwärtig verfügbare Verfahren entweder zu ungenau oder nur für stationäre Systeme geeignet sind. Das bedeutet, daß diese AR-Systeme ohnedies schon mit Kameras ausgerüstet sind. Bei einem geschickten Entwurf des optischen Trackingsystems kann man daher das Videobild, das eigentlich zur Positionsbestimmung dient, als Nebenprodukt in den Hintergrund der virtuellen Szene legen, um ohne weiteren Aufwand ein „Video-see-through“-System zu erhalten.

Leider hat der Einsatz der „Video-see-through“-Technik in Kombination mit optischen Trackingverfahren den Nachteil, daß gewaltige Datenmengen vom (eher leistungsschwachen) mobilen System verarbeitet werden müssen. Eine kleine Beispielrechnung verdeutlicht das:

Eine hochwertige, handelsübliche Web-Cam liefert ein Bild mit einer Auflösung von 640×480 Pixeln. Jeder Pixel besteht aus drei Bytes, einem für jeden der drei Farbkanäle (Rot, Grün und Blau). Pro Sekunde werden 60 Bilder geliefert. Die pro Sekunde anfallende Datenmenge beträgt also 640×480×3×60 = 55.296.000 Bytes = 52,73 Megabytes. Bei einem Stereo-System mit zwei Kameras verdoppelt sich die Datenmenge auf 105,47 MB.

In der Praxis gibt man sich mit deutlich geringeren Auflösungen und Frame-Raten zufrieden. Aber auch bei einer Auflösung von 320×240 bei 30 Bildern pro Sekunde fallen noch 6,59 MB an, die von der Kamera in den Hauptspeicher transportiert, dort vom optischen Tracking verarbeitet, und zuletzt vom Hauptspeicher auf die Graphikkarte kopiert werden müssen. Es ist offensichtlich, daß dieser gesamte Ablauf perfekt optimiert werden muß, damit vor dem Hintergrund der stark eingeschränkten Ressourcen mobiler AR-Systeme noch zufriedenstellende Frame-Raten und Batterielaufzeiten erreicht werden.

Viele klassische AR-Systeme stellen keine zufriedenstellende Lösung für die Behandlung der Videodaten bereit. Typisch für diese Systeme ist der ARToolkit-Ansatz [24]. Bei diesem Ansatz werden die Videodaten nur innerhalb des optischen Trackingsystems verarbeitet. Das Trackingsystem besitzt seine eigenen Softwaremodule, um Videobilder von Web-Cams oder Framegrabberkarten zu erhalten. Diese Bilder werden dann analysiert, um die aktuelle Position und Blickrichtung des Anwenders zu bestimmen. Zuletzt kopiert das Trackingsystem das Videobild in den Videospeicher der Graphikkarte und gibt dann der Anwendung die Gelegenheit, die virtuelle Szene und das User-Interface über das Videobild zu rendern.

Dieser monolithische Ansatz erlaubt es natürlich, sehr effizient die Videodaten zu verarbeiten – die Daten bleiben ja schließlich die ganze Zeit innerhalb eines Softwaremoduls. Die Nachteile dieses Verfahrens sind aber offensichtlich:

- Der Ansatz ist extrem unflexibel. Man ist auf die Videoquellen eingeschränkt, die vom Trackingsystem unterstützt werden. Üblicherweise sind das nur die gebräuchlichen Web-Cams und Framegrabber-Karten, die die Standard-Systemschnittstellen (DirectShow, QuickTime bzw. Video4Linux) verwenden. Andere, exotischere Verfahren können nicht oder nur unter großem Aufwand umgesetzt werden, wie z.B. das beim AR-PDA-Projekt [16] verwendete Verfahren, bei dem das Kamerabild vom AR-System über ein Netzwerk an einen leistungsfähigen Server übertragen wird, der das eigentliche optische Tracking durchführt. Darüber hinaus schränkt einen das Trackingsystem bei der Auswahl der Rendering-API ein – wenn das Trackingsystem das Videobild z.B. per OpenGL an die Graphikkarte überträgt, ist die Anwendung ebenfalls gezwungen, OpenGL zum Rendering der virtuellen Szene zu verwenden.

- Es ist konzeptionell eher schwierig, außerhalb des Trackingmoduls an die Videobilder heranzukommen. Das führt dazu, daß typische AR-Einsatzszenarios nur mit einem gewissen Programmieraufwand umgesetzt werden können, wie z.B. das „Remote-Expert-Szenario“, bei dem das Videobild über das Netzwerk an einen Experten übertragen wird, der an seinem Arbeitsplatzrechner sitzt und dem Wartungstechniker vor Ort direkt Anweisungen in dessen Blickfeld einblenden kann. Darüber hinaus wird auch die Entwicklung und das Debugging von AR-Anwendungen erschwert, weil man das Videobild ausschließlich auf dem Anwendungsrechner sehen kann und nicht auch auf anderen, an der Anwendungsentwicklung beteiligten Rechnern (siehe auch Kapitel 6).
- Letztendlich stellt der monolithische Ansatz die Systemarchitektur auf den Kopf. Eine periphere Teilkomponente des Systems, nämlich das optische Tracking, stellt plötzlich die Kernkomponente des Systems dar, der sich alle anderen Komponenten unterzuordnen haben. Die eigentliche Kernkomponente, nämlich die Anwendung selbst, ist plötzlich ein Untermodul des optischen Trackings, das nur noch aufgerufen wird, um die virtuelle Szene zu rendern.

Bei modernen AR-Systemen wird inzwischen das reine Grabben von Videobildern getrennt von den optischen Trackingverfahren implementiert. So basieren aktuelle Versionen von Studierstube [28] z.B. auf der an der TU Graz entwickelten Erweiterung von ARToolkit, „ARToolkitPlus“ [25], die nur noch die reine Tracking-Funktionalität bereitstellt. Für das Videograbbing wird dagegen die Bibliothek „OpenVideo[13]“ verwendet.

Bei dem hier vorgestellten System wird eine Lösung zur Handhabung der Videodatenströme verwendet, die auf der einen Seite effizient, auf der anderen Seite aber flexibel ist und keine unnatürlichen Änderungen an der Architektur der AR-Anwendungen erfordert. Videodatenströme werden vom zugrundeliegenden Gerätemanagement-System genauso gehandhabt wie alle anderen Datenströme (siehe Kapitel 3). Sie werden, wie im vorhergehenden Abschnitt beschrieben, über Low-Level-Sensoren in das Event-System des VRML/X3D-Renderers integriert. Innerhalb des Renderers sind die Videobilder ganz normale Texturen, die wie alle anderen Texturen beliebig weiterverarbeitet und auf beliebige geometrische Objekte plaziert werden können.

Der folgende VRML-Code demonstriert, wie Videobilder einer Kamera in den Hintergrund einer AR-Szene gerendert werden können:

---

[13] http://studierstube.icg.tu-graz.ac.at/openvideo/

VRML-Code zum Rendern des Kamerabildes

```
Shape
{
  appearance Appearance
  {
    texture DEF videoTexture PixelTexture {}
  }
  geometry Rectangle2D
  {
    size 1.0 0.75
  }
}

DEF videoSensor SFImageSensor
{
  label "Video Frames"
}
ROUTE videoSensor.value_changed TO videoTexture.set_image
```

Zunächst einmal wird eine Geometrie definiert, in diesem Fall ein Rechteck mit dem Seitenverhältnis 4:3 im Koordinatenursprung. Außerdem wird ein Low-Level-Sensor angelegt, der die Videobilder in Form des VRML-Datentyps „SFImage“ vom Device-Mangement-System erhält. Diese Bilder werden dann über eine Route an den Textur-Knoten weitergeleitet, der das Bilder als Textur auf das Rechteck mappt.

Wie man sieht, erlaubt es diese Lösung, sehr elegant und mit wenigen Zeilen Code ein „Video-see-through“-System zu erzeugen. Das Verfahren ist sehr flexibel – es ist nicht festgelegt, von woher die Videobilder stammen, sie könnten direkt von einer Kamera stammen, die am Rechner angeschlossen ist, sie könnten aber auch von einer Kamera stammen, die an einen anderem Rechner im Netzwerk angeschlossen ist (Stichwort „Remote-Expert-Szenario“). Letztendlich könnten die Bilder sogar aus einer Aufzeichnung auf der Festplatte stammen, was zu Testzwecken während der Entwicklung sehr hilfreich ist. Darüber hinaus greift diese Lösung nicht in die Architektur der Anwendung ein – die Anwendung ist immer noch die zentrale Systemkomponente.

Abbildung 36: Live-Videos werden als normale Texturen behandelt und können daher auf beliebige geometrische Objekte abgebildet werden

In den folgenden Abschnitten wird jetzt erklärt, wie die hier skizzierte Lösung praktisch implementiert wird. Dabei wird zunächst betrachtet, wie die Videobilder innerhalb des Gerätemanagement-Systems und innerhalb des Rendering-Systems transportiert werden. Danach wird beschrieben, wie typische rechenaufwendige Verarbeitungsschritte wie Entzerrung oder Spiegelung des Videobildes ressourcenschonend von der Graphikhardware durchgeführt werden können. Zuletzt wird darauf eingegangen, wie die Videobilder möglichst effizient in der Texturspeicher der Graphikkarte transferriert werden können.

5.2.1 Transport der Videobilder innerhalb des Systems

Wie bereits im 3. Kapitel erläutert wurde, kommunizieren verschiedene Softwarekomponenten innerhalb des Gerätemanagementsystems über Slots miteinander. Ein Datenwert, der in einen Outslot geschrieben wird, wird vom System in alle Inslots kopiert, die mit dem Outslot über Routes verbunden sind.

Dieser Kopiervorgang ist natürlich nur für einfache Daten effizient. Für großvolumige Daten wie z.B. Videobilder ist eine Kopie der Daten indiskutabel. Deshalb werden vom System Smart-Pointer bereitgestellt, die auf die Daten verweisen. Diese Smart-Pointer verweisen auf ein Reference-Counting-Objekt, das den eigentlichen Zeiger auf das Datenobjekt enthält und außerdem einen Zähler, der die Anzahl der Referenzen auf das Datenobjekt zählt.

Anstatt nun die Datenobjekte zu kopieren, werden Smart-Pointer kopiert. Eine solche Kopie kann sehr effizient erzeugt werden – es wird einfach der Zeiger auf das Reference-Counting-Objekt kopiert und der Reference-Counter inkrementiert. Da der Datenflußgraph beliebig viele Threads unterstützt, muß der Zugriff auf den Reference-Counter natürlich auch noch durch ein Mutex synchronisiert werden. Nichtsdestotrotz ist dieser Vorgang deutlich effizienter als eine Kopie eines gesamten Videobildes.

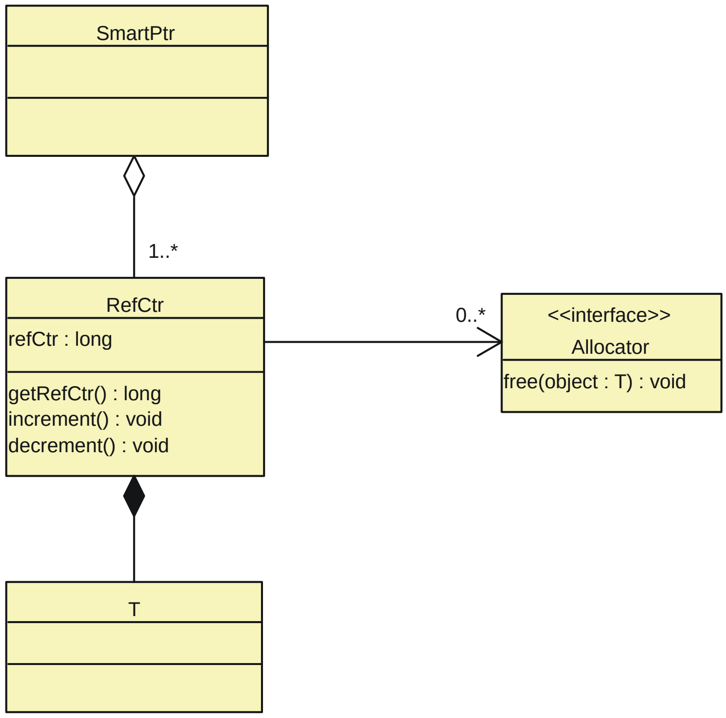


Abbildung 37: Die Standard-Smart-Pointer-Implementierung, erweitert um das Allocator-Konzept

Typische Smart-Pointer-Implementierungen, wie man sie z.B. in der C++-Bibliothek „Boost[14]“ findet, reservieren den Speicherplatz für das Datenobjekt, wenn das erste Mal auf das Objekt zugegriffen wird. Dieser Speicherplatz wird wieder freigegeben, wenn der Reference-Counter 0 wird, also keine Referenz mehr auf das Datenobjekt vorhanden ist. Diese Form der Speicherverwaltung ist aus mehreren Gründen für den Einsatz in einem Datenflußgraphen suboptimal:

- Belegen und Freigeben von Speicherblöcken sind vergleichsweise aufwendige Vorgänge. Das ist insbesondere vor dem Hintergrund problematisch, daß diese Operationen in einem Datenflußgraphen sehr häufig vorkommen – bei einer Framerate von 60 Hz muß z.B. 60 mal in der Sekunde ein Speicherblock für das aktuelle Bild reserviert und später wieder freigegeben werden. Umso ärgerlicher ist dabei, daß die verwendeten Speicherblöcke alle identisch sind (solange man natürlich nicht die Parameter der Kamera ändert).

- Es besteht die Gefahr, daß der Speicher durch das permanente Belegen und Freigeben von Speicherblöcken fragmentiert wird.

- Bei manchen externen Geräten kann das Smart-Pointer-Konzept in dieser Form nicht umgesetzt werden. So ist es z.B. bei einigen Framegrabber-APIs so, daß diese es nicht erlauben, von der Anwendung bereitgestellten Speicher mit Bilddaten zu füllen. Anstelle dessen werden die Hardware-Puffer der Framegrabber-Karte in den Adressraum der Anwendung eingeblendet, und die Anwendung erhält von der API nur die Adresse eines der Hardware-Puffer, der die aktuellen Bild-Daten enthält. Nach der Verarbeitung der Daten muß die Anwendung der API mitteilen, daß der betreffende

---

[14] http://www.boost.org/

Puffer wieder von der Framegrabber-Karte mit neuen Daten überschrieben werden kann.

Um dem speziellen Einsatzfall von Smart-Pointern in einem Datenflußgraphen Rechnung zu tragen, wurde das Smart-Pointer-Konzept entsprechend angepaßt. Das Reference-Counting-Objekt besitzt zusätzlich noch einen Zeiger auf ein „Allocator“-Objekt. Wenn der Reference-Counter auf 0 fällt, wird das Datenobjekt nicht gelöscht, sondern als Parameter der „free“-Methode des Allocator-Objekts übergeben. Falls kein Allocator-Objekt vorhanden ist, der Zeiger also 0 ist, wird das Datenobjekt wie beim herkömmlichen Smart-Pointer freigegeben. Ein ähnliches Verfahren wird z.B. auch in der DirectShow-API von Microsoft angewandt.

Knoten im Datenflußgraphen, die per Smart-Pointer verwaltete Datenobjekte erzeugen, können nun das Allocator-Interface implementieren und dadurch die Speicherverwaltung für die Datenobjekte kontrollieren. Sie können den Speicher wie gehabt freigeben und damit das klassische Smart-Pointer-Verhalten umsetzen. Oder sie können den Speicher nicht löschen, sondern in eine Freispeicherliste eintragen. Wenn ein neues Datenobjekt erzeugt wird, schauen die Knoten erst in der Speicherliste nach, ob eventuell ein altes Datenobjekt recycelt werden kann, bevor sie neuen Speicher vom System anfordern. Oder sie können z.B. der API einer Framegrabber-Karte mitteilen, daß ein Hardware-Puffer wieder frei zur Aufnahme neuer Daten ist.

5.2.2 Verlagerung rechenaufwendiger Verarbeitungsschritte auf die Graphikhardware

Das Renderingsystem erhält vom Gerätemanagement-System die Videobilder genau so, wie sie von der Videokamera geliefert werden. Üblicherweise können diese Bilder aber nicht direkt zum Rendering verwendet werden, sondern erfordern erst einige Verarbeitungsschritte:

- Häufig müssen die Videobilder horizontal oder vertikal gespiegelt werden. Der Koordinatenursprung von Videobildern ist üblicherweise die linke obere Ecke. Bei Texturen liegt der Koordinatenursprung dagegen in der linken unteren Ecke. Das Ergebnis ist, daß die Videobilder auf dem Kopf stehen und daher vertikal gespiegelt werden müssen. Aber auch das horizontale Spiegeln des Bildes kann notwendig sein, wenn einfache Web-Cams als Kameras eingesetzt werden. Solche Kameras sind eigentlich für Video-Chat-Systeme gedacht, bei denen sie das Gesicht des Anwenders vor dem Rechner aufnehmen. Manche Kameras liefern dabei ein spiegelverkehrtes Bild, um dem Anwender das Gefühl zu vermitteln, sich selbst in einem virtuellen Spiegel zu betrachten. Dies ist beim Einsatz in einem AR-System natürlich kontraproduktiv und muß durch ein horizontales Spiegeln des Videobildes korrigiert werden.
- Insbesondere beim Einsatz von preiswerten Web-Cams sind die gelieferten Videobilder stark verzerrt. Typischerweise beobachtet man eine tonnenförmige Verzeichnung des Videobildes, d.h. gerade Linien sind in der Mitte des Videobildes nach außen verbogen [89]. Da die virtuellen Objekte der AR-Szene natürlich ohne eine solche Verzeichnung, die von der Kameraoptik herrührt, gerendert werden, muß das Videobild entzerrt werden, damit reale und virtuelle Szene nahtlos zusammenpassen.

Leider sind solche Verarbeitungsoperationen nur sehr ineffizient auf dem Rechner durchzuführen. Schon eine eigentlich einfache Operation wie das Spiegeln erfordert es, jeden Pixel des Videobildes einmal anzufassen. Bei einer Auflösung 320×240 sind das bereits 76.800 Pixel pro Bild. Letztendlich handelt es sich um einen kompletten Kopiervorgang, der im Hauptspeicher des Rechners durchgeführt werden muß. Bei der Entzerrung des

Videobildes sieht es noch schlimmer aus, hier muß eine komplexe mathematische Operation auf den Bilddaten durchgeführt werden.

Glücklicherweise erlaubt es die Tatsache, daß die Videobilder innerhalb des Renderingsystems als normale Texturen gehandhabt werden, die oben genannten Operationen resourcenschonend auf der Graphikhardware durchzuführen.

Betrachten wir zunächst das Spiegeln. Das Videobild wird als Textur auf ein geometrisches Objekt gemappt. Dieses Objekt besteht aus Polygonen, die durch ihre Eckpunkte (Vertices) definiert werden. Die Eckpunkte enthalten neben ihrer Position im Raum auch noch die Information, welcher Teil der Textur auf sie abgebildet wird (die Texturkoordinate). Die linke untere Ecke der Textur hat die Koordinate (0, 0), die rechte obere Ecke die Koordinate (1, 1). Im einfachsten Fall besteht das geometrische Objekt aus einem einzigen, rechteckigen Polygon. Um die komplette Textur 1:1 ohne Spiegelung auf das Rechteck abzubilden, muß das Rechteck in seiner linken unteren Ecke die Texturkoordinate (0, 0) und in seiner rechten oberen Ecke die Texturkoordinate (1, 1) besitzen. Dies ist z.B. beim X3D-Knoten „Rectangle2D“ der Fall, der ein zweidimensionales Rechteck im Koodinatenursprung erzeugt der Fall.

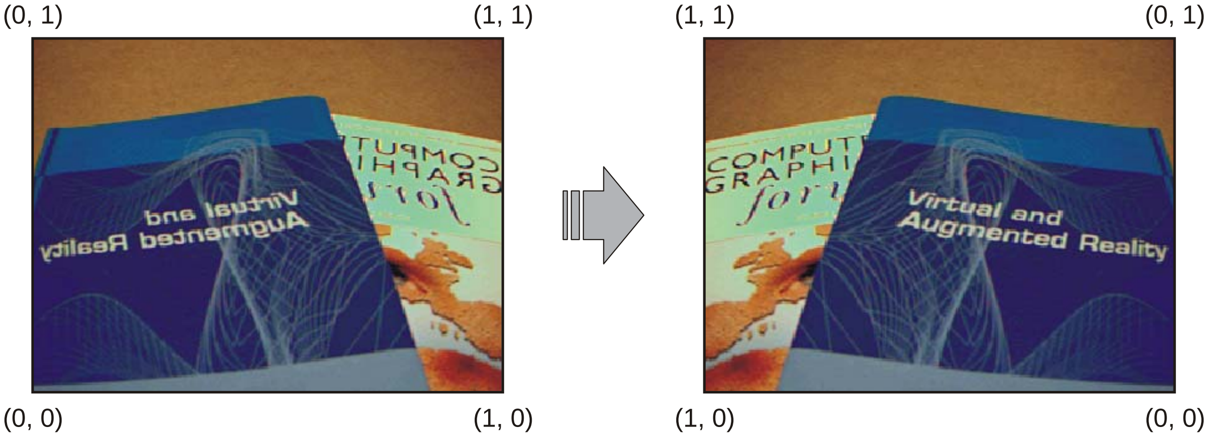


Abbildung 38: Eine handelsübliche Webcam (ADS Pyro 1394 Webcam) liefert ein spiegelverkehrtes Bild. Einfaches Vertauschen der Texturkoordinaten korrigiert diesen Fehler

Das Spiegeln einer Textur ist nun sehr einfach – man muß nur die Texturkoordinaten anpassen:

- Beim horizontalen Spiegeln muß die linke untere Ecke des Rechtecks die Texturkoordinaten (1, 0) und die rechte obere Ecke die Texturkoordinaten (0, 1) erhalten.
- Beim vertikalen Spiegeln muß die linke unter Ecke des Rechtecks die Texturkoordinaten (0, 1) und die rechte obere Ecke die Texturkoodinaten (1, 0) erhalten.

Dabei stellt sich jedoch das Problem, daß die Texturkoordinaten von Standard-Geometrie-Knoten wie „Rectangle2D“ vom X3D-Standard fest definiert sind. Natürlich könnte man einen flexibleren Geometrieknoten wie z.B. „IndexedFaceSet“ verwenden, der es erlaubt, Texturkoodinaten frei zu vergeben. Es geht jedoch noch einfacher, nämlich über eine Texturtransformation.

Texturtransformationen erlauben es, Texturkoodinaten eines geometrischen Objektes beliebig zu transformieren. Dazu skaliert man je nach gewünschter Spiegelung die x- oder y-Komponenten der Texturkoordinaten mit –1 und addiert 1. Für den Einsatz von Texturtransformationen definiert VRML/X3D einen speziellen Knoten, „TextureTransform“.

Die folgenden Codebeispiele demonstrieren, wie man mit Hilfe des „TextureTransform“-Knotens Videobilder horizontal oder vertikal spiegelt:

Horizontales Spiegeln des Videobildes

```
Shape
{
  appearance Appearance
  {
    texture DEF videoTexture PixelTexture {}
    textureTransform TextureTransform
    {
      scale –1 1
      translation 1 0
    }
  }
  geometry Rectangle2D { size 1.0  0.75 }
}
```

Vertikales Spiegeln funktioniert analog:

Vertikales Spiegeln des Videobildes

```
Shape
{
  appearance Appearance
  {
    texture DEF videoTexture PixelTexture {}
    textureTransform TextureTransform
    {
      scale 1 -1
      translation 0 1
    }
  }
  geometry Rectangle2D { size 1.0  0.75 }
}
```

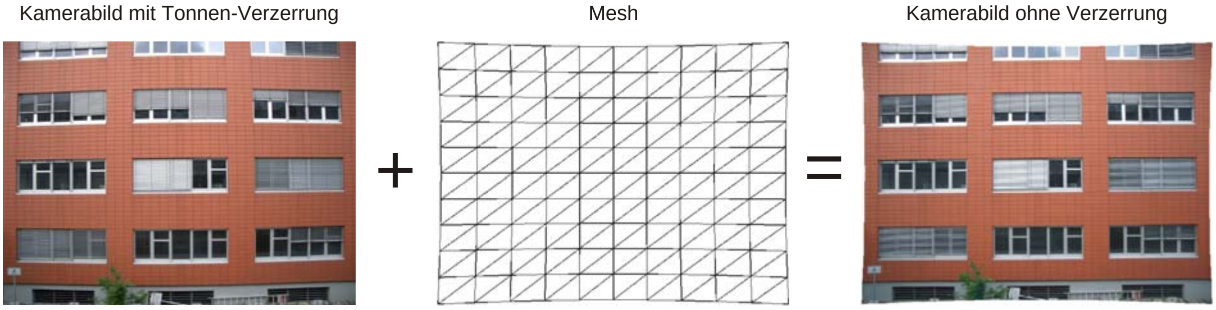


Abbildung 39: Entzerrung von Kamerabildern über entsprechend verzerrte Meshes

Genauso einfach wie das Spiegeln der Videobilder kann man auch andere, komplexere Operationen auf den Bildern effizient in Hardware durchführen. Als Beispiel wird hier die Beseitigung von Verzerrungen betrachtet, die von der Kameraoptik herrühren. Typischerweise sind Videobilder tonnenförmig verzerrt. Um diese Art von Verzerrungen zu beseitigen, muß man sich nur in Erinnerung rufen, daß die Bilder ganz normale Texturen sind, die man auf beliebige geometrische Objekte mappen kann. Bisher wurden die Videobilder auf einfache Rechtecke abgebildet. Man kann sie aber auch auf jedes beliebige andere Polygonnetz abbilden. Diese Netze können beliebig verzerrt werden und damit auch die Videobilder beliebig verzerren. Um also eine tonnenförmige Verzerrung des Videobildes zu beseitigen, mappt man es auf ein spezielles Polygonnetz, das genau so verzerrt wurde, daß es die ursprüngliche Verzerrung wieder aufhebt (siehe Abbildung 39) – ein gänges Verfahren, um mit Hilfe der Graphikhardware Verzerrungen des Videobildes auszugleichen, siehe z.B.

[90]. Natürlich stellt diese Form nur eine Annäherung an die optimale Lösung dar, aber sie ist effizient und für den hier gewünschten Zweck vollkommen ausreichend.

5.2.3 Effizienter Transfer der Videobilder in den Texturspeicher der Graphikkarte

Der letzte Schritt bei der Verarbeitung von Videodatenströmen ist der Transfer der Daten in den Texturspeicher der Graphikkarte. Videobilder werden innerhalb des Systems als normale Texturen aufgefaßt, daher müssen sie auch entsprechend behandelt werden. Damit möglichst wenig Performanz verloren geht, sollten die Daten möglichst unverändert ohne weitere Modifikationen durch die CPU an die Graphik-Hardware übergeben werden. Insbesondere sollten die Bilder nicht aufwendig skaliert werden müssen.

Skalierung von Texturen ist u.a. erforderlich, um sogenannte „Mipmaps“ zu verwenden. Mipmaps sind eine Technik, um die Rendering-Geschwindigkeit zu erhöhen und die Darstellungsqualität zu verbessern. Dabei definiert man nicht nur die eigentliche Textur, sondern auch in der Größe herunterskalierte Versionen der Textur. Dabei wird die Textur jeweils auf ein Viertel der vorhergehenden Größe skaliert, bis man eine Textur erhält, die nur noch 1×1 Pixel groß ist. Hat man z.B. eine Textur der Größe 16×8, so erzeugt man auch herunterskalierte Varianten in den Größen 8×4, 4×2, 2×1 und 1×1. Es ist offensichtlich, daß der Prozeß der Mipmap-Erzeugung rechenintensiv und zeitaufwändig ist. Er sollte nur für statische Texturen durchgeführt werden, um die Performanz und Darstellungsqualität zu erhöhen, aber nicht für dynamische Texturen, die sich ständig ändern. Doch wie kann der X3D-Browser wissen, ob eine Textur statisch oder dynamisch ist? In der neuesten Version des X3D-Standards wird dazu ein neuer Knoten spezifiziert, der „TextureProperties“-Knoten. Er enthält ein SFString-Feld „minificationFilter“, mit dem man die in OpenGL [91] verfügbaren Minification-Filter „nearest“, „linear“, „nearest_mipmap_nearest“, „nearest_mipmap_linear“, „linear_mipmap_nearest“ sowie „linear_mipmap_linear“ setzen kann. Darüber hinaus gibt es ein SFBool-Feld „generateMipMaps”, mit dem man die Erzeugung von Mipmaps explizit an- oder ausschalten kann.

Eine Skalierung der Textur ist ebenfalls notwendig, wenn die Breite und Höhe der Textur in Pixeln keine Zweierpotenz ist. X3D erlaubt Texturen mit beliebigen Ausmaßen. OpenGL dagegen erlaubt nur Texturen, deren Breite und Höhe Zweierpotenzen sind. Typische Videoformate wie 320×240 oder 640×480 sind jedoch keine Zweierpotenzen.

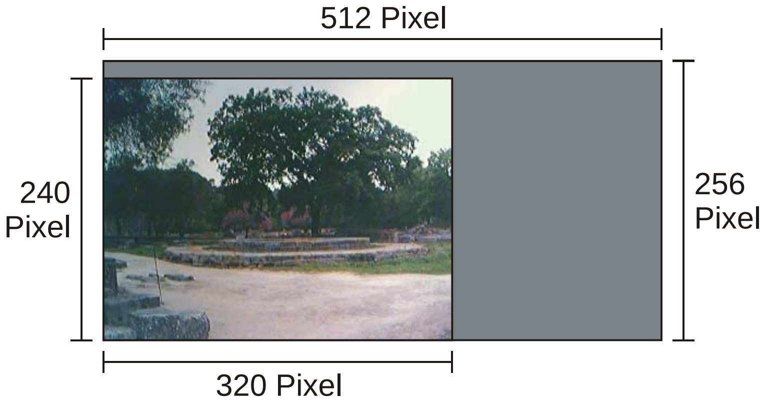


Abbildung 40: Ein Videobild der Größe 320×240 Pixel wird in die linke untere Ecke einer Textur der Größe 512×256 Pixel eingefügt

Prinzipiell stellt das kein Problem dar. Man definiert einfach eine Textur, deren Breite und Höhe die nächstgrößeren Zweierpotenzen sind. Für ein Videobild der Größe 320×240 erzeugt man also eine Textur der Größe 512×256, oder für ein Videobild der Größe 640×480 eine Textur der Größe 1024×512. Dann kopiert man mittels des OpenGL-Befehls

„glTexSubImage2D“ das Videobild in die linke untere Ecke der Textur und skaliert über eine Texturtransformation die Texturkoordinaten (bei einem Videobild der Größe 320×240 skaliert man also die u-Koordinate mit 320 ÷ 512 = 0,625 und die v-Koordinate mit 240 ÷ 256 = 0,9375).

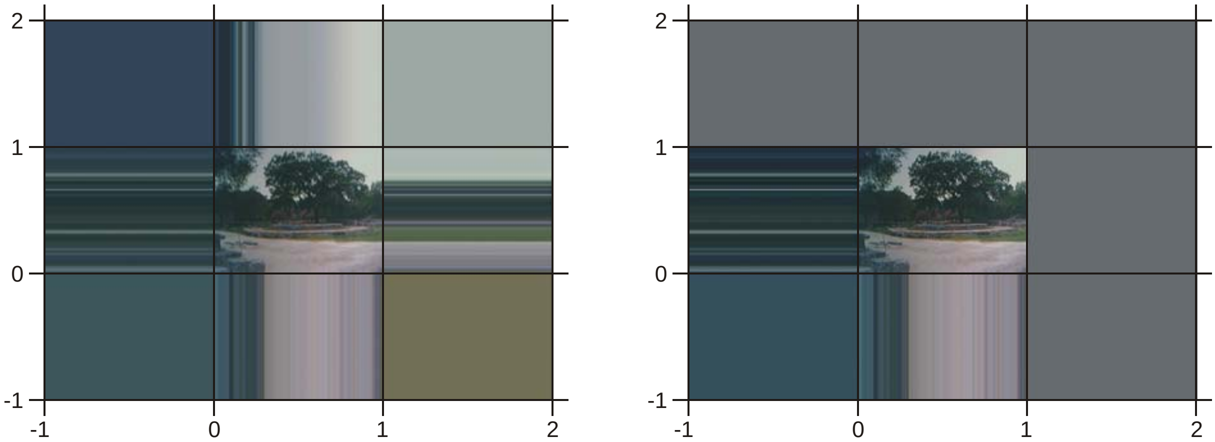


Abbildung 41: Korrekte (links) und fehlerhafte Darstellung der Textur, wenn die Texturkoordinaten nicht zwischen 0 und 1 liegen (Wrapmode=Clamp)

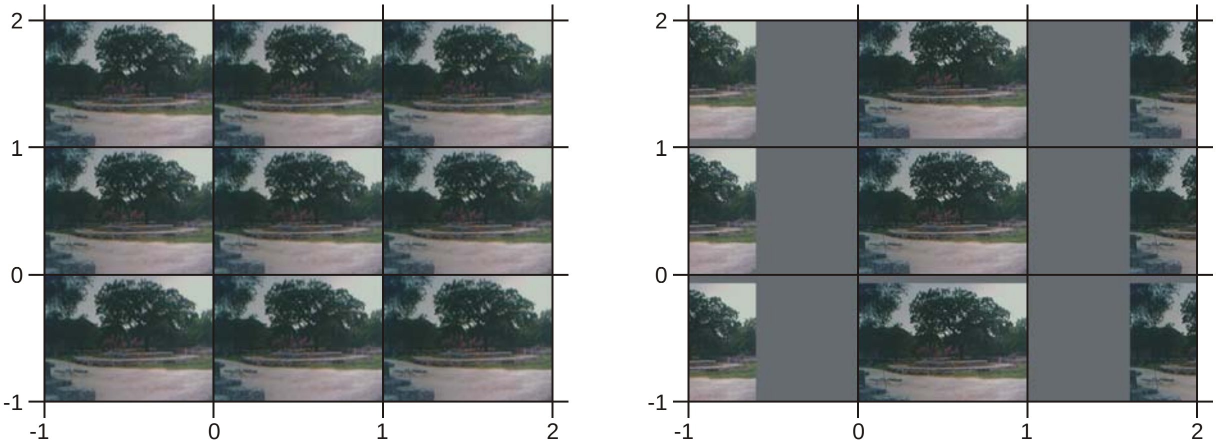


Abbildung 42: Korrekte (links) und fehlerhafte Darstellung der Textur, wenn die Texturkoordinaten nicht zwischen 0 und 1 liegen (Wrapmode=Repeat)

Die Probleme beginnen bei dieser Lösung aber, wenn die Texturkoordinaten nicht zwischen 0 und 1 liegen. In diesem Fall werden die „Lücken“ in der Textur sichtbar – die Textur verhält sich bei dieser Lösung also nicht standardkonform.

Es gibt allerdings zwei OpenGL-Erweiterungen, die die Definition von Texturen erlauben, deren Breite und Höhe keine Zweierpotenzen sind. Es handelt sich um die Extensions „GL_EXT_texture_rectangle“ und „GL_ARB_texture_non_power_of_two“.

„GL_EXT_texture_rectangle“ erfordert die Angabe von Texturkoordinaten in Pixeln, nicht in normalisierten uv-Koordinaten zwischen 0 und 1. Das stellt kein Problem dar – man skaliert einfach die X3D-Texturkoordinaten mit der Breite und Höhe der Textur in Pixeln. Problematisch ist dagegen die zweite Einschränkung – der Wrap-Mode „repeat“ wird von dieser Extension nicht unterstützt.

„GL_ARB_texture_non_power_of_two“ dagegen stellt die perfekte Lösung des Problems dar. Diese Extension erlaubt die Definition von Texturen, deren Breite und Höhe keine Zweierpotenzen sind, ohne irgendwelche neuen Einschränkungen. Leider wird diese Extension nur von neueren Graphikkarten unterstützt und ist mit einem Geschwindigkeitsverlust verbunden.

Die einzige praktikable standardkonforme Lösung besteht also darin, die Texturen auf die nächsthöhere Zweierpotenz zu skalieren. Dies ist zwar bei statischen Texturen machbar, aber bei dynamischen Texturen ist der Rechenaufwand viel zu groß. Auf der anderen Seite gibt es eine effiziente Lösung, die funktioniert, wenn die Texturkoordinaten zwischen 0 und 1 liegen. Dies ist bei Texturen, die der Darstellung von Videos dienen, üblicherweise der Fall. Die Lösung besteht darin, Texturknoten um ein weiteres SFBool-Feld „autoScale" zu erweitern. Wenn dieses Feld den Wert „TRUE" hat, verhält sich der Texturknoten standardkonform, d.h. Texturen werden automatisch auf die nächstgrößere Zweierpotenz skaliert. Wenn das Feld dagegen den Wert „FALSE" hat, werden die Texuren nicht skaliert, sondern in die linke untere Ecke einer Textur kopiert, deren Breite und Höhe die nächstgroßeren Zweierpotenzen sind. In diesem Fall können nur Texturkoordinaten zwischen 0 und 1 verwendet werden, dafür ist der Transfer der Bilddaten in den Texturspeicher der Graphikkarte deutlich schneller.

Das folgende Beispiel zeigt, wie man eine Textur in der X3D-Anwendung spezifiziert, bei der jede Form von Skalierung unterbunden wird und daher maximale Performanz beim Rendering von Videoströmen erlaubt:

Beispiel einer Textur, die jede Form von Skalierung unterbindet

```
…
DEF videoTexture PixelTexture
{
  textureProperites TextureProperties
  {
    generateMipMaps FALSE
  }
  autoScale FALSE
}
…
```

Natürlich ist dieses manuelle „Frisieren" des Texturknotens nicht optimal. Aber de facto gibt es für einen X3D-Browser keine zuverlässige und schnelle Methode, diese Parameter automatisch zu bestimmen. Außerdem sind solche Flags im X3D-Standard nichts neues, so gibt es Ähnliches bereits im „IndexedFaceSet"-Knoten in Form des „convex"-Feldes, mit dem man dem Browser einen Hinweis geben kann, daß alle Polygone der im Knoten spezifizierten Geometrie konvex sind und daher nicht trianguliert werden müssen.

5.3 Zusammenfassung

In diesem Kapitel wurde die Einbindung des AR-Frameworks in das existierende, auf dem X3D-Standard basierende VR-Renderingsystem Avalon beschrieben. Es wurden Erweiterungen des X3D-Standards vorgeschlagen und das effiziente Rendering von Videodatenströmen beschrieben.

Ein Problem bei der Entwicklung von AR-Anwendungen auf der Basis von X3D ist die eingeschränkte Fähigkeit von X3D-Anwendungen, auf typische AR-Geräte wie Tracker und Videokameras zuzugreifen. Zwar gibt es in mehreren X3D-Browser-Implementierungen Ansätze, diesen Schwachpunkt zu beseitigen. Diese Ansätze haben jedoch ihre Schwächen. Daher wurde in diesem Kapitel ein eigener Erweiterungsvorschlag gemacht, der sich in zwei wichtigen Punkten von anderen Vorschlägen unterscheidet:

1. Es werden keine Geräte-spezifischen Knoten wie z.B. „JoystickSensor" etc. verwendet, sondern es werden Knoten für jeden Datentypen spezifiziert, den X3D intern verarbeiten kann. Diese Knoten werden nicht auf Geräte abgebildet, sondern auf die einzelnen Sensoren, die die Geräte bereitstellen. Auf diese Weise ist sichergestellt, daß nicht mit jedem neuen Gerätetyp der X3D-Standard um einen neuen, entsprechenden Knoten erweitert werden muß.

2. Der Entwickler einer X3D-Anwendung spezifiziert nicht das Mapping der X3D-Knoten auf ein konkretes Gerät – dazu ist er selbstverständlich nicht in der Lage, da er die Zielplattform, auf der die Anwendung läuft, überhaupt nicht kennt. Anstelle dessen definiert er, wozu die von den Knoten gelieferten oder empfangenen Daten verwendet werden sollen. Das eigentliche Mapping von den Knoten auf die konkreten Geräte übernimmt der Anwender über ein Dialogfenster der Laufzeitumgebung – der Anwender ist der einzige, der zu diesem Mapping in der Lage ist.

Das Rendering von Videodatenströmen wird im Gegensatz zu anderen AR-Frameworks komplett vom Renderingsystem übernommen. Das bedeutet, daß die Videobilder normale Texturen sind und entsprechend flexibel verarbeitet werden können. Rechenaufwendige Verarbeitungsschritte wie die Beseitigung von Verzerrungen können direkt von der Graphikhardware übernommen werden. Damit das funktioniert, muß die interne Verarbeitung der Videodaten so effizient wie möglich implementiert sein – insbesondere sollte unnötiges Kopieren und Skalieren der Daten vermieden werden. Dazu wurde in diesem Kapitel eine Smart-Pointer-Konzepts beschrieben, daß es erlaubt, auch Hardwarepuffer von Framegrabberkarten zu verwalten bzw. genutzte Puffer zu recyceln und so eine Fragmentierung des Speichers zu vermeiden. Der X3D-Knoten „TextureProperties“ sowie eine Erweiterung der X3D-Texturknoten um ein neues „autoScale“-Feld erlauben es dem Anwendungsentwickler, die Erzeugung von Mipmaps und das Skalieren der Videobilder auf Zweierpotenzen zu unterbinden.

Im folgenden Kapitel wird nun dargestellt, welche Hilfsmittel es gibt, mit denen der Entwickler Anwendungen für dieses AR-Framework erzeugen kann.

6 Entwicklung & Debugging

Eine interessante und bislang in der wissenschaftlichen Arbeit kaum berücksichtigte Frage ist, wie man Anwendungen für mobile AR-Systeme entwickeln und insbesondere debuggen kann. Eine Lösung für dieses Problem wird in diesem Kapitel vorgestellt. Dazu wird zunächst die besondere Problematik bei der Entwicklung von mobilen AR-Anwendungen dargestellt. Danach werden aus den praktischen Erfahrungen bei der Entwicklung von AR-Anwendungen heraus die Anforderungen an eine Entwicklungsumgebung aufgestellt. Anschließend wird ein neuartiges Konzept vorgestellt, das diese Anforderungen erfüllt und es erlaubt, mobile AR-Anwendungen auf der Basis des in dieser Arbeit vorgestellten Rahmensystems zu entwickeln und zu debuggen. Dabei sollen auch die Grenzen des Konzepts nicht verschwiegen werden. Am Ende dieses Kapitels wird noch auf einige interessante „Nebeneffekte“ dieses Konzepts hingewiesen.

6.1 Problemstellung

Für die Entwicklung von herkömmlichen, Desktop-basierten Anwendungen werden üblicherweise integrierte Entwicklungsumgebungen verwendet, die es erlauben, die Anwendung aus der Entwicklungsumgebung heraus zu starten und über einen integrierten Debugger zu überwachen, d.h. die Werte von Variablen auszulesen und zu verändern sowie sogenannte Breakpoints zu setzen, an denen die Programmausführung gestoppt wird. Die Entwicklung einer Anwendung erfolgt üblicherweise in Zyklen, in denen Features der Anwendung entwickelt, getestet und von Fehlern bereinigt werden.

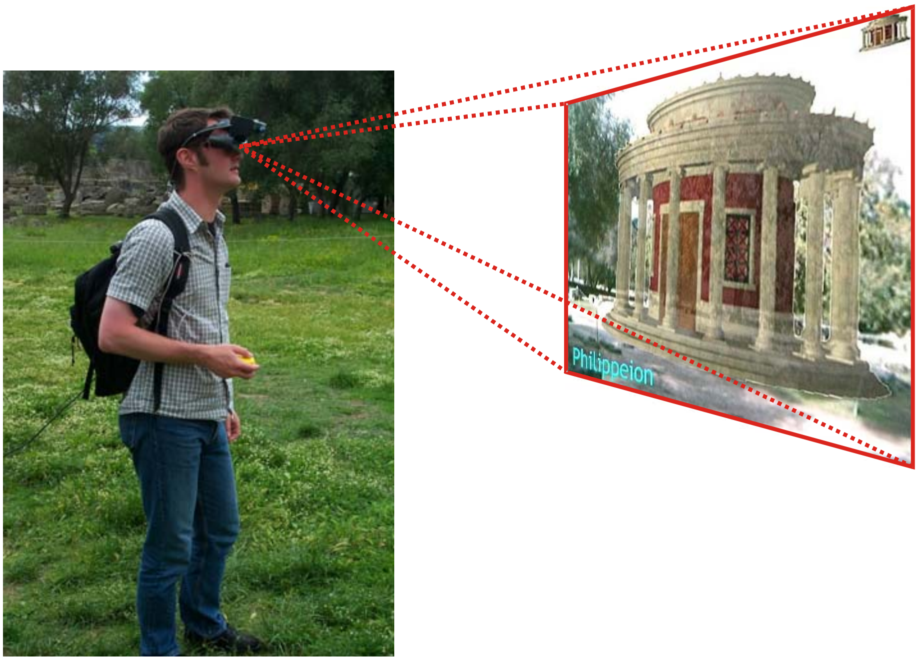


Abbildung 43: Der Entwickler beim Testen einer AR-Anwendung

Entscheidend bei diesem Entwicklungsprozeß ist, daß die Anwendungsentwicklung direkt auf dem Zielsystem stattfindet. Die Entwicklungsumgebung läuft genau wie die Anwendung auf einem normalen Desktop-Rechner. Es stellt daher kein Problem dar, während des Entwicklungsprozesses zwischen Anwendung und Entwicklungsumgebung hin- und herzuschalten.

Es wird schnell klar, daß dieser Entwicklungsprozeß nicht ohne weiteres auf mobile AR-Systeme übertragbar ist. Bei mobilen AR-Systemen kann aus verschiedenen Gründen das Entwicklungssystem nicht identisch mit dem Zielsystem der Anwendung sein. Abbildung 43 zeigt den Entwickler eines AR-Systems beim Testen der Anwendung. Zunächst einmal fällt auf, daß er keinerlei Eingabegeräte besitzt, mit denen er klassische 2D-Userinterfaces (Fenster, Buttons, Menüs etc.) bedienen könnte. Er ist daher nicht in der Lage, eine herkömmliche Entwicklungsumgebung zu benutzen, die auf einem 2D-Userinterface aufsetzt. Darüber hinaus sieht man auch, daß die zu testende AR-Anwendung nicht in einem Fenster auf einem normalen Bildschirm abläuft, sondern im Vollbildmodus auf dem HMD. Selbst wenn der Entwickler in der Lage wäre, eine herkömmliche Entwicklungsumgebung zu bedienen, würde er eventuelle Debug-Ausgaben nicht sehen können, weil sie von der Vollbilddarstellung der Anwendung verdeckt würden. Verschlimmert wird die Situation dadurch, daß die Darstellungsqualität gängiger HMDs, besonders beim Einsatz im Freien, extrem schlecht ist. Typischerweise haben sie nur eine Auflösung von 800×600 Pixeln oder sogar noch weniger. Die Lesbarkeit von klassischen 2D-Overflächen auf solchen HMDs ist extrem gering.

Zusammenfassend kann man feststellen, daß die Entwicklung von AR-Systemen aus drei Gründen nicht auf dem mobilen System selbst stattfinden kann:

1. Mobile AR-Systeme besitzen üblicherweise weder Tastatur noch Maus. Das bedeutet, daß man auf diesen Systemen weder die üblichen 2D-Desktopoberflächen bedienen noch überhaupt irgendwelchen Code programmieren kann.
2. Mobile AR-Systeme laufen typischerweise im Vollbildmodus, d.h. es ist auch nicht ohne weiteres möglich, irgendwelche Debug-Ausgaben eines Debuggers auf dem Bildschirm darzustellen.
3. Die Lesbarkeit von textuellen Informationen auf HMDs ist so schlecht, daß klassische 2D-Desktopoberflächen allein aus diesem Grund schon nicht zu bedienen sind.

Die Anwendungsentwicklung für mobile AR-Systeme entscheidet sich also in einem entscheidenden Punkt von der Anwendungsentwicklung für klassische Desktop-Systeme: Die Entwicklungsarbeit kann nicht auf der Zielplattform stattfinden.

Prinzipiell bieten klassische Entwicklungsumgebungen auch für diesen Fall Lösungen an: Das sogenannte „Remote Debugging“, bei dem der Entwicklungsrechner über Netzwerk oder serieller Schnittstelle mit dem Rechner verbunden ist, auf dem die Anwendung läuft. Der Entwickler könnte die Anwendungsentwicklung auf einem Laptop durchführen. Dieser Laptop ist z.B. über WLAN mit dem mobilen AR-Gerät verbunden. Wenn der Entwickler die AR-Anwendung testet, läßt er sich von einem Gehilfen begleiten, der den Laptop mitnimmt und bei eventuellen Fehlfunktionen ein klassisches Debugging der Anwendung durchführt.

Leider ist dieses Vorgehen in der Praxis schwer durchführbar:

- Normale Laptops sind nicht dazu gedacht, im Freien verwendet zu werden. In der Konsequenz ist es leider so, daß bei sehr heller Umgebung, insbesondere bei Sonnenschein, normale Laptop-Displays nicht zu lesen sind. Für die Arbeit im Freien benötigt man spezielle Geräte, deren Displays nicht hintergrundbeleuchtet sind, sondern über eine Reflektionsschicht verfügen, die das Umgebungslicht reflektiert. Solche Displays sind leider selten, sehr teuer, und funktionieren meist nicht in geschlossenen Räumen.
- Darüber hinaus ist das klassische Debugging bei dem in dieser Arbeit vorgestellten Rahmensystem nicht sonderlich sinnvoll. Bei diesem Rahmensystem wird die Anwendung ja nicht in C++ oder einer ähnlichen Programmiersprache programmiert. Anstelle dessen stellt der Programmierer seine Anwendung aus vorgefertigten Komponenten zusammen. Die eigentliche Anwendung besteht aus diversen Textdateien, die in einer Markup-Language die Struktur vom Datenflußgraphen des

Gerätemanagements und vom VRML-Szenengraphen des Rendering-Systems enthalten. Diese Dateien werden von einer Laufzeitumgebung eingelesen, die die in den Dateien beschriebene Struktur der Anwendung rekonstruiert und die Anwendung selbst ablaufen läßt. Mit einem klassischen Debuggingsystem würde man logischerweise nur das Laufzeitsystem selber debuggen (bei dem natürlich davon ausgegangen wird, daß es fehlerfrei funktioniert). Was der Entwickler aber eigentlich möchte ist, seine eigene Anwendung, die auf einem viel höheren Abstraktionslevel läuft, zu debuggen.

Das von klassischen Entwicklungsumgebungen angebotene „Remote Debugging" stellt also auch keine Lösung für die Probleme beim Entwickeln von mobilen AR-Anwendungen dar.

6.2 Anforderungen an eine Entwicklungsumgebung

Vor dem Hintergrund dieser Probleme stellt sich die Frage, wie AR-Anwendungen bislang in der Praxis entwickelt werden.

Die Anwort ist, daß sie auf Desktop-Systemen entwickelt werden. Immer dann, wenn die Anwendung getestet werden muß, kopiert der Entwickler sie auf das mobile Gerät, packt die verwendete Hardware z.B. in einen Rucksack, schnallt diesen auf seinen Rücken, setzt das HMD auf, überprüft die Verkabelung und verläßt das Büro, um die Anwendung in der Praxis zu testen. Während des Testvorgangs kann er eventuelle Fehlfunktionen nur registrieren, aber ihnen nicht auf den Grund gehen, weil es keine Möglichkeit gibt, während der Praxistests die Anwendung zu debuggen. Danach geht der Entwickler wieder in sein Büro und versucht, irgendwie die Ursachen der Fehlfunktionen zu finden und zu beseitigen.

Es ist offensichtlich, daß diese Vorgehensweise extrem ineffektiv ist. Anders als Desktop-Rechner, die im Allgemeinen aus erprobten Standardkomponenten bestehen, die zuverlässig funktionieren, sind die meisten mobilen AR-Geräte Eigenkonstruktionen. Die verwendete Hardware (GPS-Empfänger, Kompasse, Trackingsysteme) ist exotisch und funktioniert häufig nur unzuverlässig. Das gesamte System ist nicht integriert, sondern besteht aus einer Vielzahl von Einzelteilen, die über unzählige Kabel miteinander verbunden sind. Viele dieser Einzelteile benötigen eine eigene Stromversorgung, so daß der Anwender noch eine Vielzahl von Batterien mit sich herumträgt. Wie man sich vorstellen kann, sind Hardwarefehlfunktionen bei einem solchen System an der Tagesordnung – sei es, weil ein Gerät nicht korrekt initialisiert wurde, ein Stecker beim Herumlaufen aus der Buchse gerutscht ist, irgendwelche Wackelkontakte auftreten, oder eine der diversen Batterien leer ist.

Der Entwickler selber ist beim Testen meist nicht in der Lage herauszufinden, warum sein System nicht funktioniert – er kann nur feststellen, daß eine Fehlfunktion auftritt, aber nicht, warum sie auftritt und ob die Ursache dafür in seiner Software zu finden ist oder in einer Fehlfunktion der Hardware. Ein gutes Beispiel dafür ist der GPS-Empfänger. Nehmen wir an, der Entwickler stellt während eines Testlaufs fest, daß seine Software nicht die korrekte Position verwendet. Dafür kann es eine ganze Reihe von Ursachen geben:

- Die Software kann falsch programmiert sein, d.h. das System erhält zwar die korrekten Positionsdaten, aber sie werden von der Anwendung falsch ausgewertet. Dies ist letztendlich der einzige Fehler, der in der Verantwortung des Entwicklers liegt.

- Der GPS-Empfänger wurde falsch initialisiert. GPS-Empfänger sind eine relativ exotische Hardware und funktionieren daher häufig nicht so zuverläßig wie Standard-PC-Komponenten. Selbst wenn das Initialisierungsprotokoll von der Anwendung korrekt durchgeführt wird, kann es passieren, daß der Empfänger nicht wie gewünscht funktioniert.

- Der GPS-Empfänger funktioniert zwar korrekt, kann aber derzeit die Position nicht bestimmen. GPS-Empfänger benötigen immer Kontakt mit einer ausreichenden Anzahl von Satelliten. Dieser Kontakt kann sehr schnell verloren gehen, z.B. dadurch, daß man sich hinter Gebäude oder sogar einfach nur unter das dichte Blätterwerk einer Baumkrone begibt. Im laufenden Betrieb liefern GPS-Empfänger in einem solchen Fall eine angenäherte Position, aber bei der Initialisierung des Empfängers beim Start der Anwendung funktioniert das nicht. Praktische Erfahrungen haben gezeigt, daß es bis zu 15 Minuten dauern kann, bis GPS-Empfänger die ersten Positionsdaten liefern, wenn sich der Standort seit dem letzten Betrieb des Empfängers stark verändert hat.
- Unter Umständen ist einfach ein Stecker aus seiner Buchse gerutscht, oder es gibt irgendwo einen Wackelkontakt. Als problematisch erweisen sich hier insbesondere die weit verbreiteten USB-Anschlüsse. Auf der einen Seite erleichtern sie das Leben des Entwicklers, weil er über Hubs beliebig viele USB-Geräte an den Rechner anschließen kann und damit die Ausrüstung nicht mehr von der Zahl der am Rechner vorhandenen Schnittstellen limitiert wird. Auf der anderen Seite kann man USB-Stecker aber nicht mehr wie die alten RS232-Stecker an der Buchse festschrauben. Bei mobilen Geräten, wo ständig der gesamte Hardwareaufbau in Bewegung ist, stellt das ein großes Problem dar, welches sich nur durch den großzügigen Einsatz von Klebeband bewältigen läßt.
- Und zu guter Letzt kann auch einfach die Batterie leer sein, die den GPS-Empfänger mit Strom versorgt. Die Versorgung aller Geräte eines mobilen AR-Systems mit Strom stellt eine der größten Herausforderungen dar. Leider beziehen viele Geräte den benötigten Strom nicht über die Verbindung mit dem Rechner, sondern benötigen eigene, externe Stromversorgungen. Selbstverständlich hat hier jedes Gerät seine eigenen Anforderungen an Stromspannung und Stromstärke, sodaß der Anwender üblicherweise eine ganze Reihe von unterschiedlichen Batterien mit sich herumtragen muß, die alle einen unterschiedlichen Ladezustand und unterschiedliche Kapazität haben und daher zu den verschiedensten Zeitpunkten neu geladen werden müssen.

Wenn der Entwickler während des Tests seines AR-Systems also feststellt, daß das System seine Position nicht korrekt berücksichtigt, stellen sich ihm folgende Fragen:

- Liefert der GPS-Empfänger irgendwelche Positionsdaten? Wenn ja, liegt die Ursache für die Fehlfunktion in der Anwendung. Um diese Frage zu beantworten, müßte der Entwickler die Möglichkeit haben, sich vom Gerätemanagement die zuletzt vom GPS-Empfänger erhaltenen Nachrichten anzeigen zu lassen.
- Wenn der GPS-Empfänger keine Positionsdaten liefert, wurde er überhaupt korrekt initialisiert? Hier müßte der Entwickler vom Gerätemanagement einen Statusbericht erhalten, aus dem er erkennen könnte, ob die Initialisierung geklappt hat. Wenn nicht, müßte er die Möglichkeit haben, den Treiber (d.h. den Knoten im Gerätemanagement) des Empfängers im laufenden Betrieb neu zu starten.
- Wenn die Initialisierung nicht klappen will, liegt ein Hardware-Problem vor, d.h. die Batterien, die den GPS-Empfänger mit Strom versorgen, sind leer, oder es ist ein Stecker aus seiner Buchse gerutscht. In diesem Fall bleibt dem Entwickler nichts anderes übrig, als wieder in sein Büro zurückzukehren und den Ladezustand der Batterien bzw. die Verkabelung zu überprüfen.
- Wenn die Initialisierung geklappt hat, aber trotzdem keine Positionsdaten geliefert werden, so hat der GPS-Empfänger unter Umständen keinen Kontakt zu ausreichend vielen GPS-Satelliten. Wenn das der Fall ist, muß der Entwickler einfach eine Weile

warten bzw. sich einen besseren Standort suchen. Auch hier müßte der Entwickler die Möglichkeit haben, einen Statusbericht vom Gerätemanagement anzufordern.

Aus diesen Praxiserfahrungen lassen sich nun eine Reihe von Anforderungen an die Entwicklungsumgebung ableiten:

1. Die Entwicklungsumgebung kann nicht auf dem Rechner laufen, auf dem die AR-Anwendung läuft. Anstelle dessen muß es möglich sein, sie auf irgendeinem Rechner laufen zu lassen, der eine Netzwerkverbindung zum mobilen AR-Gerät aufbauen kann.
2. Die Entwicklungsumgebung sollte auf einer möglichst großen Palette von Geräten laufen. Wie oben bereits erwähnt, sind Laptops für den Betrieb im Freien meistens ungeeignet, weil ihr Display in einer hellen Umgebung nicht oder nur schwer lesbar ist. Andere Geräte wie Mobiltelephone und PDAs sind dagegen für einen Einsatz im Freien konzipiert und besitzen reflektive Displays, die auch unter Sonnenlicht lesbar bleiben. Es wäre wünschenswert, wenn die Entwicklungsumgebung auch auf solchen Geräten verfügbar wäre.
3. Eine wichtige Aufgabe der Entwicklungsumgebung ist selbstverständlich, dem Programmierer die Werkzeuge zum Entwickeln von AR-Anwendungen zur Verfügung zu stellen. Im Falle des in dieser Arbeit vorgestellten Rahmensystems bedeutet das, daß die Entwicklungsergebung den Aufbau des Datenflußgraphen des Gerätemanagements und den Aufbau des VRML-Szenengraphen unterstützen muß. Damit ist nicht gemeint, daß die Entwicklungsumgebung die Modellierung von 3D-Objekten erlauben soll – dafür gibt es perfekt spezialisierte Softwarepakete wie 3DStudio Max oder Maya. Anstelle dessen sollte die Entwicklungsumgebung das Zusammenstellen von Logikkomponenten und fertig modellierten 3D-Objekten zu einer AR-Anwendung anbieten, d.h. das Erzeugen und Modifizieren von Knoten und der Routes, die diese Knoten miteinander verbinden.
4. Eine weitere wichtige Aufgabe der Entwicklungsumgebung ist das Debugging, d.h. die Fehlersuche bei laufender AR-Anwendung. Anders als bei klassischer Programmentwicklung heißt das bei dem hier vorgestellten Rahmensystem nicht, daß man schrittweise durch einen Code gehen kann und dabei Variablen beobachtet. Es bedeutet vielmehr, daß man die Graphen (Datenflußgraph des Gerätemanagements und VRML-Szenengraph) im laufenden Betrieb betrachten und modifizieren kann. Es sollte möglich sein, Knoten hinzuzufügen, zu entfernen oder zu modifizieren. Darüber hinaus sollte es möglich sein, Routes zwischen Knoten zu erzeugen und zu entfernen. Der Entwickler muß die Möglichkeit haben, Informationen zum aktuellen Zustand von Knoten abzufragen, die zuletzt über Slots verschickten oder empfangenen Datenwerte anzeigen zu lassen, und unter Umständen selbst Datenwerte über die Entwicklungsumgebung zu erzeugen und in das System einzuschleusen.

Bei fast allen existierenden AR-Frameworks und Gerätemanagement-Systemen gibt es inzwischen integrierte User-Interfaces, die es erlauben, diese Systeme während des laufenden Betriebs zu konfigurieren und zu überwachen. Zum Teil erlauben diese User-Interfaces auch das Entwerfen und Debuggen von Anwendungen über das Netzwerk (Punkt 1). So besitzt z.B. das Tinmith-System [35] einen integrierten Network File System (NFS)-Server, der es erlaubt, die Baumstruktur des Systems als Dateisystem auf anderen Rechnern zu mounten und dann mit den normalen Werkzeugen zum Verwalten von Dateien zu bearbeiten. Die „DWARF Interactive Visualization Environment“ [92] erlaubt es, über die in DWARF integrierten CORBA-Mechanismen den aktuellen Zustand des Systems zu visualisieren und zu modifizieren. VR Juggler kann über das in Java implementierte „VjControl“-Programm [52] zur Laufzeit über eine Netzwerkverbindung kontrolliert werden. Der Nachteil dieser

Lösungen ist, daß sie entweder nicht wirklich anwenderfreundlich sind (wie der in Tinmith integrierte NFS-Server) oder die Installation einer speziellen Softwarekomponente auf dem Entwicklungs- bzw. Debugging-Rechner erfordern, die natürlich auf die jeweilige Plattform portiert werden muß.

6.3 Konzept einer Entwicklungsumgebung

Wie sieht nun eine Entwicklungsumgebung aus, die alle oben angesprochenen Anforderungen erfüllt? Die beiden wichtigsten Punkte sind, daß die Entwicklungsumgebung auf einem anderen Rechner läuft als die Anwendung, und das die Entwicklungsumgebung auch auf Mobiltelephonen und PDAs funktionieren sollte. Optimal wäre eine Lösung, die keine zusätzliche Software auf den Entwicklungsmaschinen erfordert, sondern auf allen Plattformen mit „Bordmitteln“ funktioniert.

Tatsächlich findet man eine Lösung, wenn man sich Hardwaregeräte anschaut, deren Konfiguration komplex ist, die keine Bedienungselemente besitzen, aber an ein Netzwerk angeschlossen sind. Solche Geräte sind z.B. Router, Internet-Kameras oder Netzwerkdrucker. Bei diesen Geräten ist es inzwischen Standard, daß sie einen eingebauten Web-Server besitzen, über den die Geräte konfiguriert werden können. Dazu startet man einfach auf irgendeinem Rechner einen normalen Web-Browser, tippt als URL den Namen oder die IP-Adresse des Geräts ein, und erhält dann vom Gerät erzeugte Web-Seiten, über die verschiedene Parameter eingestellt werden können.

Interessanterweise ist dieses Konzept nicht nur auf Geräte beschränkt. Selbstverständlich kann man auch in Softwarepakete einen kleinen Web-Server einbauen, der es erlaubt, die Software über einen Web-Browser zu bedienen und fernzusteuern. Genau darauf baut die Entwicklungsumgebung des in dieser Arbeit beschriebenen AR-Rahmensystems auf. Das Bestechende dabei ist, daß dieser Ansatz es erlaubt, das AR-Rahmensystem über alle Arten von Geräten zu steuern, die über einen Web-Browser und eine Netzwerkverbindung zum AR-System verfügen, insbesondere also auch Mobiltelephonen und PDAs. Dabei muß auf diesen Systemen noch nicht einmal eine spezielle Software installiert werden – alles, was zur Entwicklung und zum Debugging von AR-Anwendungen erforderlich ist, bringen solche Systeme schon von Haus aus mit!

Betrachten wir zur Verdeutlichung dieses Konzepts einmal genauer, was passiert, wenn man in die Adressleiste eines Web-Browser (z.B. Firefox) eine URL eingibt, z.B. „http://www.zgdv.de/avalon/index.html“.

Zunächst einmal zerlegt der Web-Browser die URL in ihre Einzelteile – in diesem Fall in das sogenannte „Schema“ („http“), den Rechnernamen („www.zgdv.de“) und den Pfad des gewünschten Dokuments („/avalon/index.html“).

Das „Schema“ gibt an, welches Protokoll der Rechner zur Kommunikation mit dem Web-Server verwenden soll. Normalerweise ist das das HyperText Transfer Protocol (HTTP) [93].

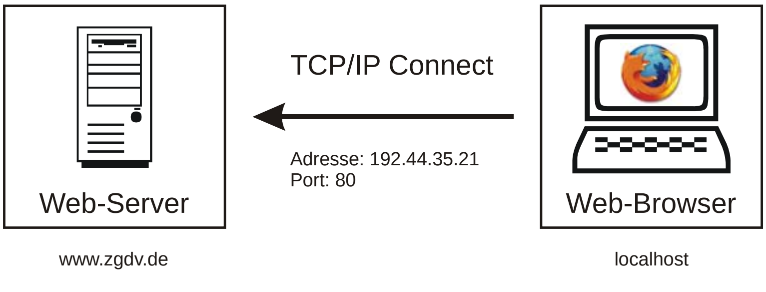


Abbildung 44: 1. Schritt - der Web-Browser baut eine TCP/IP-Verbindung zum Web-Server auf

Im ersten Schritt stellt der Web-Browser eine TCP/IP-Verbindung zum Web-Server her, z.B. einem „Apache“ (siehe Abbildung 44). Dazu wandelt er zunächst den symbolischen Rechnernamen („www.zgdv.de“) mit der Hilfe eines Nameservers in eine IP-Adresse (192.44.35.21) um. Außerdem benötigt er die Port-Nummer, unter der der Web-Server auf dem Rechner zu erreichen ist. Diese Port-Nummer kann Bestandteil der URL sein (nämlich hinter dem Rechnernamen, mit einem Doppelpunkt von diesem getrennt), oder es wird der Default-Port „80“ verwendet, wenn wie in unserem Fall keine Port-Nummer in der URL angegeben wird.

Abbildung 45: 2. Schritt - Der Web-Browser sendet einen HTTP-GET-Befehl an den Web-Server

Nachdem der Verbindungsaufbau gelungen ist, beginnen Web-Browser und Web-Server über die TCP/IP-Verbindung miteinander zu kommunizieren. Das dabei verwendete Protokoll ist, wie oben bereits erwähnt, HTTP. HTTP besteht aus einfachen Text-Nachrichten. Zunächst sendet der Web-Browser einen „GET“-Befehl an den Web-Server, in dem er den Pfad (in unserem Beispiel „/avalon/index.html“) des gewünschten Dokuments angibt (siehe Abbildung 45). Die komplette HTTP-Nachricht sieht beim Firefox-Browser z.B. folgendermaßen aus:

HTTP Request

```
GET /avalon/index.html HTTP/1.1
Accept: text/xml,application/xml,application/xhtml+xml,
  text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Accept-Encoding: gzip,deflate
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Connection: keep-alive
Host: www.zgdv.de:80
Keep-Alive: 300
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O;
  de-DE; rv:1.7.5) Gecko/20041108 Firefox/1.0
```

Die erste Zeile der Nachricht besteht aus dem HTTP-Kommando („GET“), dem Pfad des gewünschten Dokuments („/avalon/index.html“), und der verwendeten HTTP-Version („HTTP/1.1“). Der Rest der Nachricht ist der sogenannte Message-Header, in dem der Browser u.a. spezifiziert, welche Dokumenttypen in welchen Sprachen er akzeptiert. Hinter dem Message-Header folgt der Message-Body, der im Falle des GET-Kommandos aber leer ist.

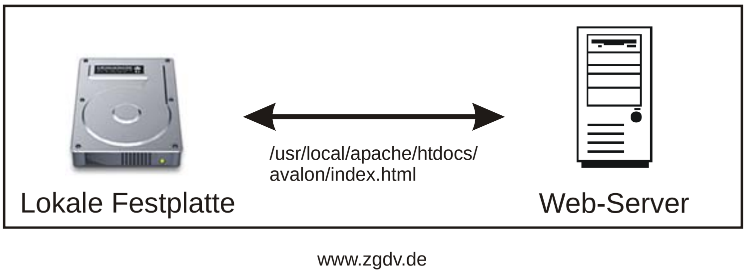


Abbildung 46: 3. Schritt - der Web-Server sucht das gewünschte Dokument auf der lokalen Festplatte

Als Reaktion auf das GET-Kommando versucht der Web-Server, das gewünschte Dokument „/avalon/index.html“ auf der lokalen Festplatte zu finden. Dazu wird der Pfad des Dokumenten-Wurzelverzeichnisses (z.B. „/usr/local/apache/htdocs“) vor den Dokumentenpfad gesetzt und die Datei unter dem resultierenden Pfad („/usr/local/apache/htdocs/avalon/index.html“) gesucht (siehe Abbildung 46).

Abbildung 47: 4. Schritt: Der Web-Server sendet eine HTTP-Reply an den Web-Browser

Im letzten Schritt sendet der Web-Server eine HTTP-Reply-Nachricht an den Web-Browser (siehe Abbildung 47). Falls das Dokument nicht gefunden wurde, wird eine Fehlermeldung zurückgeschickt, ansonsten das gewünschte Dokument. Die Reply-Nachricht sieht in unserem Beispiel folgendermaßen aus:

HTTP Reply

```
HTTP/1.1 200 OK
Date: Fri, 11 Mar 2005 16:39:28 GMT
Server: Apache/1.3.28 (Unix) mod_gzip/1.3.26.1a mod_ssl/2.8.15
  OpenSSL/0.9.7b
Last-Modified: Fri, 02 Apr 2004 12:36:12 GMT
ETag: "16deb-5aa-406d5e3c"
Accept-Ranges: bytes
Content-Length: 1450
Content-Type: text/html

...
```

In der ersten Zeile der Reply-Nachricht befindet sich eine Statuszeile mit der verwendeten HTTP-Version („HTTP/1.1“) sowie einem Statuscode („200 OK“), der das Ergebnis des HTTP-GET-Kommandos in einer von der Browser-Software interpretierbaren Weise zurückliefert. Der Statuscode 200 bedeutet in diesem Fall, daß das Dokument gefunden und das Kommando erfolgreich ausgeführt werden konnte.

Direkt nach der Statuszeile folgt wie beim GET-Kommando der Message-Header. Er enthält unter anderem Informationen über die Größe des Dokuments in Bytes („Content-Length“) sowie über den MIME-Type des Dokuments („Content-Type“).

Direkt im Anschluß an den Header folgt der Message-Body mit dem Dokument selbst (oben nicht mit abgedruckt).

Nach Abschluß des Dokument-Downloads kann der Web-Browser weitere Kommandos an den Web-Server verschicken oder die TCP/IP-Verbindung abbrechen.

Betrachtet man diesen Ablauf etwas genauer, so fällt einem ein wichtiger Punkt auf: Der dritte Schritt kann auch anders implementiert werden. Nirgendwo ist festgelegt, daß der Web-Server ein bereits existierendes Dokument von der lokalen Festplatte zurückliefen muß. Anstelle dessen kann das Dokument bei jedem Abruf auch „on-the-fly“ generiert werden, entweder vom Web-Server selber oder von anderen Software-Komponenten. Genau dies wird auch bereits von vielen Web-Sites so gehandhabt, z.B. durch den Einsatz von CGI-Skripten, PHP, Active Server Pages (ASP) oder JSP. Letztendlich kann man den Download eines

Dokuments durch einen Web-Browser auch so betrachten, daß der Web-Browser über das HTTP-GET-Kommando einen beliebigen Befehlsstring an den Web-Server sendet, den dieser auswertet und das Ergebnis in der HTTP-Reply-Nachricht zurückschickt. Es spricht also nichts dagegen, daß man einen Web-Server in ein Software-Paket einbaut und auf diese Weise die Software mit Hilfe eines einfachen Web-Browsers über das Netzwerk fernsteuert.

Ein Einwand an dieser Stelle könnte sein, daß ein Web-Server wie z.B. der Apache viel zu umfangreich ist, um in ein Softwarepaket eingebaut zu werden – insbesondere im Falle eines AR-Rahmensystems für mobile Systeme wie Mobiltelephone oder PDAs, wo Ressourcen wie z.B. Speicher knapp sind. Auf der anderen Seite war bereits in einem der ältesten mobilen AR-Systeme, MARS [94], ein Web-Server Bestandteil der Systemarchitektur. Tatsächlich sind normale, full-featured Web-Server wie der Apache für diesen Anwendungszweck viel zu mächtig. Es wird ja kein Web-Server benötigt, der tausende von gleichzeitigen Zugriffen effizient und unter allen Umständen fehlerfrei ausführt. Hier geht es nur darum, eine Schnittstelle zwischen AR-Rahmensystem und Web-Browser zu schaffen, die es einem Entwickler erlaubt, über ein zweites Gerät die auf dem mobilen AR-Gerät laufende AR-Anwendung zu debuggen. Zu diesem Zweck genügt schon ein kleiner, einfacher Web-Server, der nur wenige Ressourcen verbraucht.

Einen solchen Web-Server zu programmieren ist einfach. Er muß einfach nur die an einem Port eingehenden TCP/IP-Verbindungen entgegennehmen. Vom eingehenden HTTP-Request braucht er im einfachsten Falle nur die erste Zeile parsen, die aus den drei Teilen HTTP-Kommando („GET“), dem Pfad des gewünschten Dokuments sowie der verwendeten HTTP-Version („HTTP/1.1“) besteht. Von diesen drei Teilen ist nur der mittlere (der Pfad) interessant, der sich mit Leichtigkeit aus der ersten Zeile extrahieren läßt. Daraufhin führt der integrierte Web-Server die im Pfad kodierte Operation aus und liefert das Ergebnis in der HTTP-Reply-Nachricht an den Web-Browser zurück. Alle diese einfachen Operationen lassen sich mit geringem Aufwand implementieren. Wer selbst diesen Aufwand scheut, kann auch auf einfache, frei verfügbare Web-Server-Implementierungen zurückgreifen, wie z.B. WebFS[15].

Wie kann man nun über diese Schnittstelle ein User-Interface realisieren, das es erlaubt, eine AR-Anwendung zu entwickeln und zu debuggen? Einfache, von der Laufzeitumgebung generierte Textdokumente können zwar einen Einblick in den Status des laufenden Systems geben, aber wie kann man auf diese Weise neue Knoten erzeugen oder Verbindungen zwischen den Knoten?

Um diese Frage zu beantworten, muß man wissen, daß die vom Web-Browser dargestellten Textdokumente in einer speziellen Layout-Sprache codiert sind, nämlich der „HyperText Markup Language“ (HTML) [95]. Diese spezielle Programmiersprache für Hypertext-Dokumente kann nicht nur das Layout des Dokuments festlegen, sondern unterstützt auch das Erzeugen von Eingabemasken, in die man Daten eingeben kann. Diese Daten können dann vom Web-Browser an den Web-Server zurückgeschickt werden. Die Eingabemasken können praktisch alle Elemente enthalten, die man von modernen 2D-Oberflächen kennt: Buttons, Texteingabefelder, Options- und Checkboxen, Listen, Drop-Down-Listen etc.

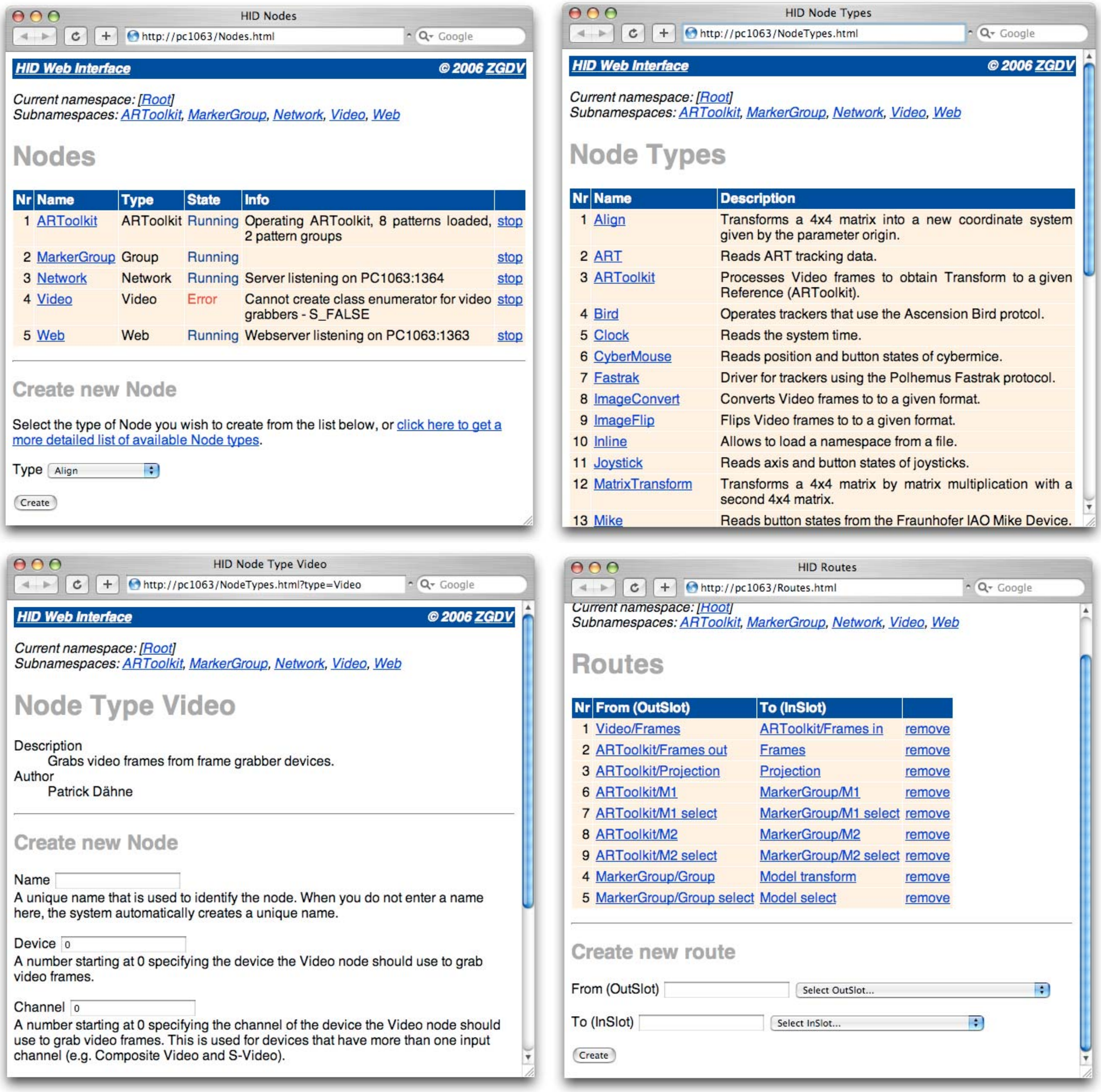
In Form der HTML-Eingabemasken stehen alle Grundbausteine zur Verfügung, um eine vollwertige 2D-Oberfläche zur Bedienung des AR-Rahmensystems im Web-Browser aufzubauen. Das AR-Rahmensystem stellt über den integrierten Web-Server eine Startseite bereit. Von dieser Startseite aus kann der Entwickler über Hyperlinks andere Seiten erreichen, die Informationen über den Datenflußgraphen des Gerätemanagements oder den VRML-

---

[15] http://linux.bytesex.org/misc/webfs.html

Szenengraphen des Rendering-Systems enthalten. Dabei kommt dem Web-Interface die Tatsache entgegen, das alle Elemente dieser Graphen ein reflektives Interface zur Verfügung stellen, d.h. man kann von einem Knoten z.B. erfahren, von welchem Typ er ist, welche Out- und Inslots er hat etc. Die so erzeugten Textdokumente enthalten aber nicht nur textuelle Informationen, sondern auch Eingabemasken, über die man das System modifizieren kann. Diese Eingabemasken bestehen aus den üblichen 2D-Userinterface-Elementen. Wenn der Entwickler einen speziellen Button (vergleichbar mit dem „OK“-Button in Dialogfenstern) in einer Eingabemaske betätigt, werden alle in der Eingabemaske eingegebenen Informationen in einer URL kodiert vom Web-Browser an den Web-Server übertragen. Der Web-Server dekodiert die URL, führt die gewünschten Modifikationen durch, und liefert eine neue, aktualisierte HTML-Seite mit dem modifizierten Systemzustand an den Web-Browser zurück.

Neben einfachen Modifikationen erlaubt es HTML auch, Daten vom Web-Server herunterzuladen oder auf den Server hochzuladen. Dies wird dazu genutzt, die Konfiguration des Systems in Konfigurationsdateien zu speichern oder aus Konfigurationsdateien zu laden. Zu diesem Zweck gibt es wie von herkömmlichen 2D-Oberflächen gewohnt spezielle Buttons zum Laden und Speichern. Klickt man auf einen dieser Buttons, so öffnet der Browser einen Datei-Dialog, in dem man den gewünschten Dateinamen beim Speichern eingeben bzw. die gewünschte Datei beim Laden auswählen kann. Die Übertragung der Dateien vom mobilen AR-System auf den Entwicklungsrechner und umgekehrt erfolgt dabei völlig transparent für den Anwender.



Current namespace: [Root]
Subnamespaces: ARToolkit, MarkerGroup, Network, Video, Web

Nodes

Nr	Name	Type	State	Info	
1	ARToolkit	ARToolkit	Running	Operating ARToolkit, 8 patterns loaded, 2 pattern groups	stop
2	MarkerGroup	Group	Running		stop
3	Network	Network	Running	Server listening on PC1063:1364	stop
4	Video	Video	Error	Cannot create class enumerator for video grabbers - S_FALSE	stop
5	Web	Web	Running	Webserver listening on PC1063:1363	stop

Create new Node

Select the type of Node you wish to create from the list below, or click here to get a more detailed list of available Node types.

Type: Align

Current namespace: [Root]
Subnamespaces: ARToolkit, MarkerGroup, Network, Video, Web

Node Types

Nr	Name	Description
1	Align	Transforms a 4x4 matrix into a new coordinate system given by the parameter origin.
2	ART	Reads ART tracking data.
3	ARToolkit	Processes Video frames to obtain Transform to a given Reference (ARToolkit).
4	Bird	Operates trackers that use the Ascension Bird protcol.
5	Clock	Reads the system time.
6	CyberMouse	Reads position and button states of cybermice.
7	Fastrak	Driver for trackers using the Polhemus Fastrak protocol.
8	ImageConvert	Converts Video frames to to a given format.
9	ImageFlip	Flips Video frames to to a given format.
10	Inline	Allows to load a namespace from a file.
11	Joystick	Reads axis and button states of joysticks.
12	MatrixTransform	Transforms a 4x4 matrix by matrix multiplication with a second 4x4 matrix.
13	Mike	Reads button states from the Fraunhofer IAO Mike Device.

Current namespace: [Root]
Subnamespaces: ARToolkit, MarkerGroup, Network, Video, Web

Node Type Video

Description
Grabs video frames from frame grabber devices.
Author
Patrick Dähne

Create new Node

Name:
A unique name that is used to identify the node. When you do not enter a name here, the system automatically creates a unique name.

Device: 0
A number starting at 0 specifying the device the Video node should use to grab video frames.

Channel: 0
A number starting at 0 specifying the channel of the device the Video node should use to grab video frames. This is used for devices that have more than one input channel (e.g. Composite Video and S-Video).

Current namespace: [Root]
Subnamespaces: ARToolkit, MarkerGroup, Network, Video, Web

Routes

Nr	From (OutSlot)	To (InSlot)	
1	Video/Frames	ARToolkit/Frames in	remove
2	ARToolkit/Frames out	Frames	remove
3	ARToolkit/Projection	Projection	remove
6	ARToolkit/M1	MarkerGroup/M1	remove
7	ARToolkit/M1 select	MarkerGroup/M1 select	remove
8	ARToolkit/M2	MarkerGroup/M2	remove
9	ARToolkit/M2 select	MarkerGroup/M2 select	remove
4	MarkerGroup/Group	Model transform	remove
5	MarkerGroup/Group select	Model select	remove

Create new route

From (OutSlot): Select OutSlot...
To (InSlot): Select InSlot...

Abbildung 48: Screenshots des Webinterface

Abbildung 48 zeigt einige Screenshots des Web-Interface. Links oben sieht man alle gerade gestarteten Knoten. Darüber hinaus werden Status-Informationen angezeigt – z.B. konnte in diesem Fall der Knoten zum Grabben von Videobildern nicht gestartet werden. Auf dieser Webseite können Knoten neu hinzugefügt, gelöscht oder neu gestartet werden. Wenn man einen neuen Knoten erzeugen möchte, klickt man auf den Namen, und die Seite rechts oben wird dargestellt. Der Anwender wählt aus der Liste der verfügbaren Knoten den gewünschten aus. Auf der folgenden Seite (links unten) kann er dann Parameter spezifizieren. Schließlich können Outslots und Inslots des neuen Knotens über Routes mit anderen Knoten verbunden werden (rechts unten).

6.4 Grenzen des Konzepts

Auch wenn es das hier beschriebene Konzept des Web-Interfaces auf elegante Weise erlaubt, AR-Anwendungen zu erzeugen und zu debuggen, soll doch nicht verschwiegen werden, daß das Konzept seine Grenzen hat.

Die Darstellung der Graphen (Datenflußgraph des Gerätemanagements und VRML-Szenengraph) über HTML-Textdokumente ist sicherlich nicht optimal. Graphen sollten besser (wie der Name schon andeutet) in graphischer Form dargestellt werden. Leider hat sich bisher kein Standard zur Darstellung von Vektorgraphiken im Internet etablieren können. SVG stellt eine sehr vielversprechende Lösung für Vektorgraphiken dar, wird aber gegenwärtig nur vom Firefox-Browser direkt verarbeitet, von anderen Web-Browsern nur über Plugins. Diese Plugins sind aber noch sehr eingeschränkt und stehen gerade für Mobiltelephone und PDAs häufig überhaupt nicht zur Verfügung. Trotzdem wäre es für die Zukunft wünschenswert, das Web-Interface von HTML-Text-Seiten auf SVG-Vektorgraphiken umzustellen, um so die Graphen direkt und nicht in Form von Texttabellen darstellen zu können.

Schwerer als solche „kosmetischen“ Einschränkungen wiegt jedoch die Tatsache, daß die HTTP-Kommunikation zwischen Web-Browser und Web-Server eine Einbahnstraße darstellt. Web-Browser sind in dieser Kommunikation aktive Elemente, d.h. sie initiieren den Verbindungsaufbau, während Web-Server passiv sind und nur Verbindungen entgegennehmen. Dies hat insbesondere zur Folge, daß der Web-Server nicht von sich aus in der Lage ist, einem Web-Browser Nachrichten zukommen zu lassen oder ihn auf irgendetwas hinzuweisen.

Dies bedeutet in unserem Falle, daß es nicht möglich ist, sich über das Web-Interface Zustandsveränderungen eines Knoten im Graphen anzeigen zu lassen oder neue, über Slots verschickte oder empfangene Daten. Anstelle dessen kann man sich nur einen „Schnappschuß“ des momentanen Zustandes anzeigen lassen. Die im Browser dargestellte HTML-Seite wird nicht automatisch aktualisiert, wenn sich die dargestellten Informationen ändern, sondern der Entwickler muß aktiv das Anfertigen eines neuen „Schnappschusses“ auslösen, indem er den „Reload“-Button des Web-Browsers drückt.

Trotz dieser Einschränkungen stellt das Web-Interface in der Praxis jedoch ein wichtiges Werkzeug zum Erstellen und Debuggen von AR-Anwendungen dar. Wie der folgende Abschnitt zeigen wird, eröffnet es sogar noch weitere, ursprünglich unvorhergesehene Einsatzmöglichkeiten.

6.5 Weitere Einsatzmöglichkeiten

Das in diesem Kapitel beschriebene Web-Interface sollte primär dazu dienen, AR-Anwendungen zu entwickeln und zu debuggen. Nachdem es fertig implementiert und in das VR-/AR-System „Avalon“ integriert war, stellte sich jedoch schnell heraus, daß es noch weitere, nützliche Einsatzmöglichkeiten gibt.

Eine wichtige Anforderung an VR-Systeme ist, daß sie sich möglichst einfach in andere Softwarekomponenten integrieren lassen. VR ist kein Selbstzweck, sondern dient dazu, komplexe Sachverhalte in einer möglichst natürlichen, leicht verständlichen Weise darzustellen. Aus diesem Grund sind VR-Systeme häufig nur ein Bestandteile größerer Softwaresysteme, z.B. um die Ergebnisse von Simulationssystemen darzustellen. Daher ist es wichtig, daß VR-Systeme leistungsfähige Schnittstellen nach außen zur Verfügung stellen.

Diese Schnittstellen sind üblicherweise im Falle von Programmiersprachen wie C und C++ als Bibliotheken ausgelegt, die man zur Anwendung hinzufügt, oder im Falle von Java als Java-Packages. Dabei muß für jede gewünschte Programmiersprache eine eigene Lösung, ein spezielles „Language binding“ erzeugt werden. Dies ist sehr aufwendig, weshalb üblicherweise nur wenige Programmiersprachen unterstützt werden.

Das Web-Interface stellt hier eine völlig neuartige Schnittstelle zur Verfügung. Wie oben beschrieben, erlaubt es zum Zwecke der Anwendungsentwicklung und des Debugging, die AR-Anwendung beliebig zu modifizieren (Knoten und Routes hinzuzufügen und zu entfernen etc.) und sogar Nachrichten von außen in die Anwendung einzuschleusen. Genau dies kann man natürlich auch dazu nutzen, um die Anwendung über externe Softwarekomponenten zu steuern. Dazu wird kein spezielles „Language binding“ benötigt, sondern die externen Softwarekomponenten müssen nur dazu in der Lage sein, Daten von URLs herunterzuladen.

Dazu sind aber praktisch alle momentan gebräuchlichen Programmiersprachen (C/C++, Java, C#, Visual Basic, Python, Perl etc.) in der Lage. Aber nicht nur Programmiersprachen können damit die Anwendung über das Web-Interface zu kontrollieren, sondern praktisch alle Softwarekomponenten, die URLs herunterladen können. So können z.B. einfache Unix-Shell-Scripte AR-Anwendungen mit Hilfe des „wget“-Kommandos steuern, oder sogar HTML-Webseiten und Flash-Filme.

Eine Auswahl möglicher Kommandos sieht z.B. so aus („pc1013“ ist der Name des Rechners und „8080“ die Portnummer, unter der das Web-Interface erreichbar ist):

```
http://pc1013:8080/loadURL?url=scene.wrl
```

Ersetzt die gegenwärtig geladene Szene durch eine neue Szene „scene.wrl“.

```
http://pc1013:8080/createVRMLFromURL?url=object.wrl&parent=root
```

Lädt einen Teilgraphen von der URL „object.wrl“ und hängt ihn unter den Gruppenknoten mit dem Namen „root“

```
http://pc1013:8080/getFieldValue?node=material&field=diffuseColor
```

Liefert den aktuellen Wert des Felds „diffuseColor“ im Knoten mit dem Namen „material“.

```
http://pc1013:8080/setFieldValue?node=material&field=diffuseColor&value=1+0+0
```

Setzt den Wert des SFColor-Feldes „diffuseColor“ im Knoten mit dem Namen „material“ auf den Wert „1 0 0“ (entspricht der Farbe Rot).

Selbstverständlich gelten auch hier die oben beschriebenen Einschränkungen. Das Web-Interface stellt eine Einbahnstraße dar, d.h. es ist zwar möglich, von externen Softwarekomponenten aus mit der VR-/AR-Anwendung zu kommunizieren, aber die VR-/AR-Anwendung kann nicht von sich aus Verbindung zu externen Komponenten aufnehmen, um diesen irgendwelche Ereignisse in der VR-Szene mitzuteilen.

Abbildung 49: Die Walk-through-Anwendung „Dom von Siena“ (links) und das zugehörige User-Interface (rechts)

In vielen Fällen ist eine solche Form der Kommunikation jedoch völlig ausreichend, z.B. bei vielen VR-Walk-through-Anwendungen, bei denen man sich zwar in einer VR-Welt bewegen kann, aber diese Welt relativ statisch ist und man mit ihr nur eingeschränkt interagieren kann. Ein Beispiel ist die VR-Anwendung „Dom von Siena“ [93] (siehe Abbildung 49). Diese

Anwendung erlaubt es, sich in einem 3D-Modell des Domes von Siena zu bewegen. Dabei wird man vom virtuellen Reiseführer „Luigi“ begleitet, der einem Informationen zu den verschiedenen Teilen des Domes liefert. Gesteuert wird diese Anwendung über ein einfaches Flash-Applet, das in einem Web-Browser dargestellt wird, z.B. wie in der Abbildung oben auf einem PDA. Das Flash-Applet stellt einen Grundriss des Domes dar. Indem man mit dem Stift des PDAs auf verschiedene Punkte auf diesem Grundriss deutet, kann man sich in der VR-Welt zu diesen Punkten begeben. Außerdem kann man Luigi anweisen, zu jedem Punkt etwas aus den Themengebieten „Kultur“, „Geschichte“ und „Architektur“ zu erzählen.

Es sei hier nochmals darauf hingewiesen, daß auf dem PDA keinerlei spezielle Software installiert ist. Alles, was vorhanden sein muß, ist ein Web-Browser und ein Plugin zum Darstellen von Flash-Applets. Es ist also möglich, daß ein Anwender seinen eigenen PDA verwendet, mit diesem Verbindung zum VR-System aufnimmt, den Flash-Film herunterlädt und die Anwendung steuert.

Den hier betriebenen „Mißbrauch“ des Web-Interfaces kann man sogar noch weiter treiben. SOAP [97] ist eine neue Technik, die in letzter Zeit viel Interesse auf sich gezogen hat. SOAP, auch unter dem Stichwort „Web-Objects“ bekannt, ist die Grundlage für .NET-Strategie von Microsoft oder JavaONE von Sun. Im Prinzip handelt es sich um eine modernisierte Form der altbekannten „Remote Procedure Calls“ (RPC). Anstelle von proprietären Schnittstellen zum Aufruf von Funktionen auf fremden Rechnern im Netzwerk wird dabei konsequent auf aktuelle Standards gesetzt. Der Funktionsaufruf mit allen Parametern wird in eine XML-Nachricht umgesetzt und zum ausführenden Rechner geschickt. Das Funktionsergebnis wird ebenfalls in eine XML-Nachricht verpackt und vom ausführenden Rechner zurückgeschickt. Als Übertragungsprotokoll kann dabei u.a. HTTP verwendet werden. Auf dem ausführenden Rechner muß dabei nur ein Web-Server laufen, der SOAP unterstützt.

Es ist offensichtlich, daß das Web-Interface eine perfekte Basis darstellt, um eine Schnittstelle auf der Basis von SOAP zur Verfügung zu stellen. Der Web-Server ist ja schon vorhanden, er muß nur um Code erweitert werden, der SOAP-XML-Nachrichten parsen, den Funktionsaufruf durchführen und das Funktionsergebnis in eine SOAP-XML-Nachricht verpacken kann. Bislang wurde dieses Interface aber nur prototypisch als „Proof-of-Concept“ realisiert.

6.6 Zusammenfassung

In diesem Kapitel wurde ein neues Konzept vorgestellt, um AR-Anwendungen zu entwickeln und zu Debuggen. Bisher werden AR-Anwendungen häufig wie herkömmliche Desktop-Anwendungen entwickelt, was wie oben dargelegt bei mobilen AR-Anwendungen problematisch ist. Zwar haben viele existierende Gerätemanagement-Systeme und AR-Frameworks integrierte User-Interfaces, die es erlauben, die Anwendung während der Laufzeit umzukonfigurieren und zu debuggen. Einige dieser Systeme ermöglichen es sogar, über das Netzwerk von einem anderen Rechner aus auf die Anwendung zuzugreifen. Dazu muß jedoch eine Softwarekomponente auf dem Debugging-Rechner installiert werden, was eine Portierung auf die jeweilige Plattform erfordert.

Das hier vorgestellte neuartige Konzept eines in das Laufzeitsystem integrierten Web-Servers erlaubt es, die Anwendung von jedem beliebigen Rechner aus zu debuggen und zu steuern, auf dem ein Web-Browser installiert ist und der (z.B. über WLAN) eine Netzwerkverbindung zum AR-System aufbauen kann. Das Konzept hat seine Nachteile, insbesondere Aufgrund der Tatsache, daß die Kommunikation zwischen Web-Browser und AR-System eine Einbahnstraße darstellt. Auf der anderen Seite erhält man aber auch eine völlig neuartige Schnittstelle vom AR-System zu externen Softwarekomponenten, die es erlaubt, das AR-

System von jeder Softwarekomponente aus zu steuern, das in der Lage ist, Daten von einer URL abzurufen.

Im folgenden Kapitel wird nun anhand von zwei Anwendungsbeispielen gezeigt, wie man mit dem in dieser Arbeit vorgestellten Rahmensystem konkret AR-Anwendungen erstellt.

7 Anwendungsbeispiele

Die in den vorangehenden Kapiteln beschriebenen Konzepte und Lösungen wurden implementiert und sind Bestandteil der am Fraunhofer IGD und am Zentrum für Graphische Datenverarbeitung (ZGDV) entwickelten VR-/AR-Software „Avalon“. Sie bilden die Grundlage für eine Reihe von AR-Projekten, die am IGD und am ZGDV durchgeführt wurden und werden. Dabei wurden die theoretischen Konzepte und Lösungen in der Praxis erprobt und verbessert. In diesem Kapitel werden nun zwei von diesen Projekten näher betrachtet. Es handelt sich um das Outdoor-AR-Projekt „Archeoguide“ und das Indoor-AR-Projekt „MARIO“.

7.1 Archeoguide

Abbildung 50: Die Ruine des Hera-Tempels (links) und der rekonstruierte Hera-Tempel, wie ihn der Anwender des Archeoguide-Systems sieht

Archeoguide [4] ist ein typisches Outdoor-AR-Projekt. Die Idee des Projektes ist es, dem Besucher einer historischen Stätte mittels einer AR-Ausrüstung zu demonstrieren, wie es früher auf dem Gelände aussah.

Das Archeoguide-System wurde beispielhaft für das antike Olympia in Griechenland implementiert, dem Ort, an dem die olympischen Spiele im Altertum stattfanden. Es handelt sich um eine der wichtigsten archäologischen Ausgrabungsstätten in Griechenland, und gleichzeitig ist es die mit Abstand wichtigste touristische Sehenswürdigkeit auf der Halbinsel Peleponnes, einem wirtschaftlich stark unterentwickelten Gebiet, für das der Tourismus eine wichtige Einnahmequelle ist.

Der (touristische) Besucher des Ausgrabungsgeländes wird wahrscheinlich enttäuscht sein. Keines der alten Bauwerke ist mehr vorhanden. Das Gelände ist eine mit Trümmern bedeckte Parklandschaft. Sich hier die prachtvollen alten Tempel vorzustellen, gefüllt mit pulsierendem Leben, fällt schwer. Aus diesem Grund drängen Tourismusmanager darauf, die alten Gebäude zu rekonstruieren, um auf diese Weise die Attraktivität des Geländes zu steigern. Dies stößt selbstverständlich auf den erbitterten Widerstand der Archäologen, die den gegenwärtigen Status Quo am liebsten unverändert für die Forschung erhalten wollen. Momentan behilft man sich mit einem Kompromiß – es werden einzelne Säulen der Tempel (z.B. Zeus- und Hera-Tempel) wieder aufgerichtet, damit der Besucher zumindest einen Eindruck von einstmaligen Größer der Bauwerke erhält. In einem angrenzenden Museum werden Bilder und Modelle der rekonstruierten Tempel ausgestellt.

Abbildung 51: Besucher mit der Archeoguide-Ausrüstung

Abbildung 52: Darstellung von Avataren

Das Archeoguide-System stellt eine Lösung für den Konflikt zwischen der touristischen Vermarktung einer archäologischen Stätte und dem Wunsch der Archäologen nach Konservierung und Erhaltung der Ausgrabungsstätten dar. Die Idee des Projektes war es, daß der Besucher der Stätte beim Betreten eine AR-Ausrüstung (siehe Abbildung 51) erhält, genauso, wie er bereits heute einen Audio-Guide mieten kann. Er wandert dann wie bisher über das Trümmerfeld, kann aber, wann immer er es wünscht, die AR-Ausrüstung verwenden, die ihm Rekonstruktionen der alten Tempel in sein aktuelles Blickfeld einblendet (siehe Abbildung 50). Doch nicht nur „tote“ Gebäude können so dargestellt werden, sondern auch das damalige Leben, das sich in und um ihnen herum abspielte. Animierte Avatare stellen Priester dar, die rituelle Handlungen in den Tempeln vornehmen, und im alten Stadion von Olympia kann man Athleten bei der sportlichen Betätigung beobachten (siehe Abbildung 52). All das erlaubt es dem Besucher, einen realistischen Eindruck davon zu bekommen, was sich früher auf dem jetzt toten und leeren Trümmerfeld abgespielt hat. Im Gegensatz zur Darstellung von Modellen in einem Museum befindet er sich „mitten drin“, er sieht die reale Umgebung, hört die Geräusche und riecht die Gerüche der realen Umgebung und erhält so ein viel intensiveres Bild.

Die technische Umsetzung dieser Vision war zwar eine Herausforderung, aber machbar. Als Knackpunkt erwies sich jedoch die Wirtschaftlichkeit eines solchen Systems. Die Kosten eines einzelnen AR-Systems sind viel zu hoch – selbst bei der einfachsten Umsetzung mittels PDA würden die Anschaffungskosten pro Gerät ca. 700 Euro beträgen. Wenn man davon ausgeht, daß sich ständig mehere hundert Besucher auf dem Gelände aufhalten, wird schnell klar, daß hier allein die Grundausstattung mit Geräten eine größere Investition erfordern würde. Neben den fixen Anschaffungskosten fallen jedoch noch laufende Kosten an. Die AR-Systeme müssen gewartet werden. Leider sind Webpads und PDAs empfindliche Geräte, und es ist damit zu rechnen, daß unter den rauen Einsatzbedingungen eine Vielzahl von Beschädigungen und technischen Mängeln bis hin zum Totalverlust einzelner Geräte auftreten. Darüber hinaus besitzen diese Geräte nur eine kurze Batterielaufzeit, weshalb sich eine nicht unerhebliche Anzahl dieser Geräte ungenutzt in Ladestationen befinden wird. Zusammengenommen führen alle diese Punkte dazu, daß das ursprüngliche Archeoguide-Konzept nicht kostendeckend umgesetzt werden kann.

Die einzige realistische Lösung für dieses Problem besteht darin, daß man auf den Verleih von Hardware komplett verzichtet. Tatsache ist, daß schon heute eine Vielzahl von Besuchern eigene Geräte bei sich tragen, die in der Lage wären, die Archeoguide-Software auszuführen. Bei diesen Geräten handelt es sich um moderne Mobiltelephone, insbesondere Smartphones, die in der Lage sind, Java-Applikationen auszuführen, ein (relativ) hochauflösendes Farbdisplay und eingebaute Kameras besitzen, und manchmal sogar schon über einen GPS-Empfänger verfügen. Anstatt sich eine AR-Ausrüstung auszuleihen, könnte der Besucher gegen Gebühr die Archeoguide-Software auf sein eigenens Mobiltelephon laden. Der Aufwand beim Betreiber des archäologischen Geländes würde sich auf einen Server-Rechner und den Beratungs-Support beschränken.

In den folgenden Kapiteln werden nun das technische Archeoguide-Konzept und seine Realisierung genauer beschrieben.

7.1.1 Systemarchitektur

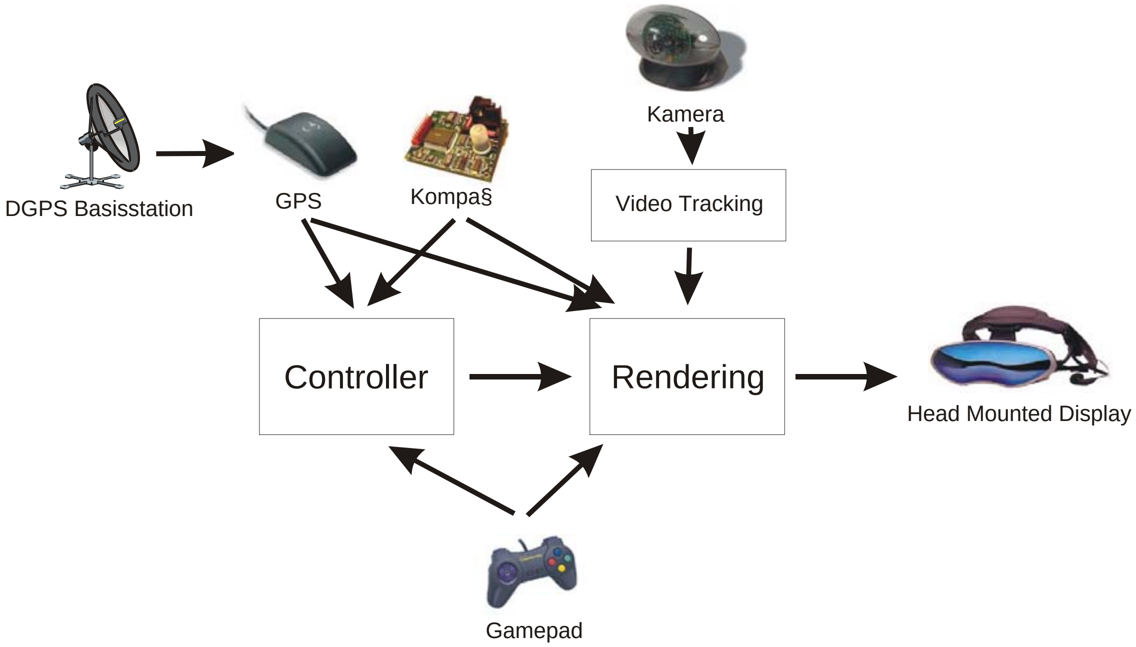


Abbildung 53: Architektur des Archeoguide-Systems

Eine wichtige Einschränkung beim Entwurf des Archeoguide-Systems war, daß das historische Gelände selbst in keiner Weise modifiziert werden durfte. Das bedeutete insbesondere, daß keine optischen Marker installiert werden konnten, die die Positionsbestimmung per Videotracking hätten unterstützen können. Aufgrund dieser Einschränkung wurde der ursprüngliche Ansatz, an jedem Punkt des Geländes eine AR-Darstellung der Umgebung anzubieten, verworfen. Anstelle dessen gab es eine Reihe von „Aussichtspunkten“, an denen eine AR-Ansicht der Umgebung geboten wurde (Abbildung 54). Außerhalb dieser Aussichtspunkte war keine AR-Darstellung möglich – hier verhielt sich das System ähnlich wie ein herkömmlicher Audioguide.

Abbildung 54: Aussichtspunkte auf dem Olympia-Gelände

Am Rande des Geländes wird ein Content-Server aufgestellt, auf dem alle Informationen (Audio-Erklärungen, Bilder, Videos, 3D-Objekte etc.), die die mobilen Geräte dem Anwender präsentieren können, abgelegt sind. Die mobilen Geräte sind über ein WLAN mit desem Server verbunden. Dieses WLAN dient auch anderen Zwecken, z.B. der Kommunikation der Geräte untereinander oder zur Übertragung von Korrekturdaten des Differential GPS (siehe unten).

Abbildung 53 zeigt die grobe Architektur des Archeoguide-Systems. Die mobilen Geräte bestimmen Ihre Position grob mittels GPS (für die Positionsbestimmung) und elektronischem Kompaß (für die Bestimmung der Blickrichtung). Aus diesen Daten leitet eine Controller-Komponente ab, ob und wenn ja an welchem Aussichtspunkt der Besucher sich gerade befindet. Wenn sich der Besucher an einem Aussichtspunkt befindet, wird eine AR-Darstellung der rekonstruierten Gebäude in das Blickfeld des Anwenders eingeblendet. Zu diesem Zweck wird mittels Videotracking die genaue Position und Orientierung des Besuchers bestimmt.

Wie oben bereits erwähnt, war es nicht möglich, irgendwelche optischen Marker zur Unterstützung des Videotracking auf dem Gelände zu installieren. Es wurde daher auf ein markerloses Tracking zurückgegriffen. Dieses Tracking basiert auf Referenzbildern. Es werden bei der Installation des Archeoguide-Systems auf dem Gelände Referenzbilder von allen Aussichtspunkten aufgenommen und in einer Datenbank abgelegt. Dabei kann (und soll) es mehrere Referenzbilder pro Aussichtspunkt geben, die von leicht unterschiedlichen Positionen aus aufgenommen werden. Für jedes Referenzbild wird ein 2D-Overlay-Bild der rekonstruierten Tempel gerendert, manuell in das Referenzbild plaziert, und ebenfalls in der Datenbank abgelegt. Während des Tracking-Prozesses wählt das Tracking-System das Referenzbild aus der Datenbank, das die größte Übereinstimmung mit dem aktuellen Videobild besitzt, und berechnet die Transformation zwischen dem Referenzbild und dem aktuellen Videobild. Diese Transformation wird benutzt, um das zum Referenzbild gehörende 2D-Overlay-Bild in das aktuelle Videobild einzufügen. Es handelt sich beim Archeoguide-System also um ein reines 2D-AR-System.

Abbildung 55: Webpad- und PDA-Variante des Archeoguide-Systems

Neben dieser High-End-Lösung gibt es auch noch mehrere „abgespeckte“ Varianten, die mit Webpads und PDAs realisiert werden (Abbildung 55). Die Webpad-Lösung besitzt keine Kamera, aber GPS und Kompaß. Die Bestimmung des aktuellen Aussichtspunktes geschieht wie bei der High-End-Lösung. Wenn der Besucher sich an einem Aussichtspunkt befindet, wird ihm allerdings keine echte AR-Darstellung geboten, sondern es wird ein vorgefertigtes Panoramabild auf dem Display des Webpads dargestellt, in das die rekonstruierten Gebäude eingeblendet werden. Die Blickrichtung im Panoramabild stimmt mit der aktuellen Blickrichtung überein, die mittels des elektronischen Kompasses bestimmt wird.

Die PDA-Lösung besteht nur aus einfachen HTML-Seiten, die dem Besucher Informationen über das Gelände vermitteln. Falls das Gerät mit GPS ausgerüstet ist, wird automatisch die zur aktuellen Position passende Seite dargestellt, ansonsten muß der Besucher die aktuelle Position manuell angeben.

7.1.2 Umsetzung

Wie wurde das Archeoguide-System nun auf Grundlage des in dieser Arbeit beschriebenen AR-Frameworks umgesetzt? Abbildung 56 zeigt den Datenflußgraphen des Archeoguide-Systems.

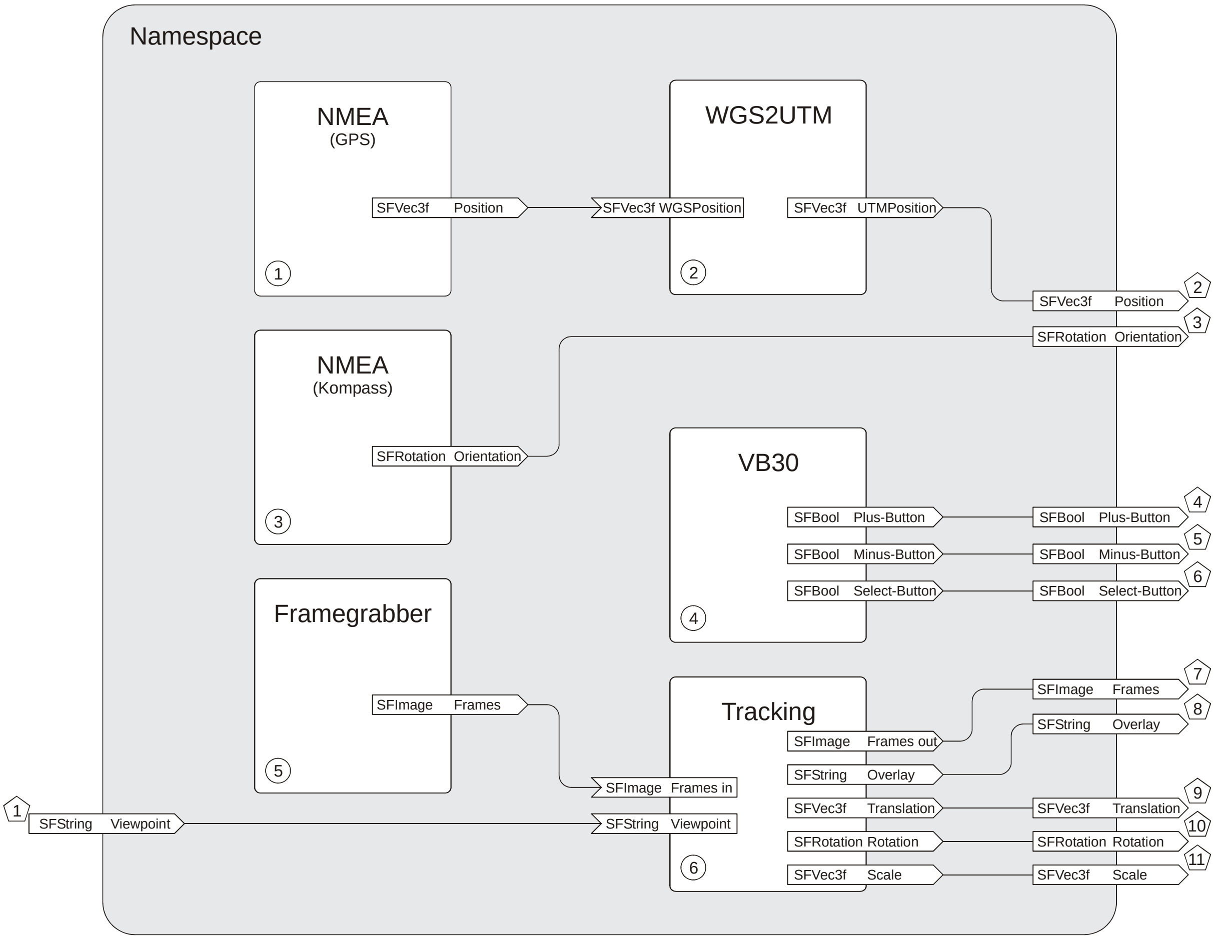


Abbildung 56: Der Datenflußgraph des Archeoguide-Systems

GPS und Kompaß sind über serielle Schnittstellen angeschlossen und kommunizieren über das NMEA-Protokoll mit dem Rechner. Entsprechend findet man im Datenflußgraphen zwei NMEA-Knoten (mit 1 und 3 nummeriert) für diese Geräte mit jeweils einem Outslot, nämlich der Position beim GPS-Empfänger und der Blickrichtung beim Kompaß. Die Blickrichtung kann direkt im System verwendet werden und wird daher über den Outslot Nummer 3 aus dem Graphen exportiert. Die GPS-Position dagegen wird dagegen im WGS84-Format geliefert, einem Polarkoordinatensystem, das die Position in Länge, Breite und Höhe über dem Erdboden angibt. Dieses Format kann vom Renderingsystem nicht direkt verarbeitet werden, da dieses ein kartesisches Koordinatensystem verwendet. Daher werden die WGS-Koordinaten von einem speziellen Knoten (Nummer 2) in UTM-Koordinaten transformiert. UTM ist ein kartesisches Koordinatensystem, bei dem die Erdoberfläche wie beim Schälen einer Orange in Streifen aufgeteilt wird, die auf eine Ebene projeziert werden. Auf diese Weise kann man die Position auf einem dieser gekrümmten Oberflächenstreifen angenähert mit kartesischen Koordinaten angeben. Die UTM-Position wird über den Outslot Nummer 2 aus dem Graphen exportiert.

Das komplette Userinterface wird über drei Buttons („Plus“, „Minus“ und „Select“) auf dem nVision VB-30 Head Mounted Display gesteuert. Die Buttons sind ebenfalls über eine serielle Schnittstelle angeschlossen, die von einem speziellen „VB30“-Knoten (Nummer 4) ausgelesen wird. Der Status der drei Buttons wird über drei SFBool-Outslots (Nummer 4 bis 6) exportiert.

Die Videobilder der Webcam werden von einem „Framegrabber“-Knoten (Nummer 5) in den Datenflußgraphen eingespeist. Die Videoframes werden über eine Route an den den „Tracking“-Knoten (Nummer 6) übertragen, der das gesamte Videotracking übernimmt. Der

Tracking-Knoten besitzt neben dem Inslot für die Videoframes noch einen zweiten Inslot, über den die Controller-Komponente ihm den Identifier des aktuellen Aussichtspunktes mitteilt. Dieser Identifier wird über den exportierten Inslot Nummer 1 in den Graphen eingespeist. Der Tracking-Knoten holt sich aus einer Datenbank die zum jeweiligen Aussichtspunkt gehörigen Referenzbilder, bestimmt das Referenzbild mit der größten Übereinstimmung mit dem aktuellen Videobild, berechnet die Transformationen (Translation, Rotation und Skalierung), die notwendig sind, um das Referenzbild mit dem Videobild zur Deckung zu bringen, und stellt diese Transformationen über drei Outslots (Nummer 9-11) dem Archeoguide-System zur Verfügung. Darüber hinaus besitzt der Tracking-Knoten noch einen „Overlay“-Outslot (Nummer 8), über den die URL des zum jeweiligen Referenzbild gehörigen Overlay-Bildes verschickt wird.

Neben den diesen Outslots besitzt der Tracker-Knoten noch einen weiteren, der das Videobild zur Verfügung stellt (über den Outslot Nummer 7). Dieser Outslot scheint zunächst überflüssig zu sein, denn man könnte ja anstelle dessen auch direkt den Outslot des Framegrabber-Knotens verwenden. Man muß jedoch berücksichtigen, daß das Videotracking ein sehr rechenintensiver Vorgang ist, der Zeit benötigt. Würde man das Videobild des Framegrabber-Knotens verwenden, so würde das Videobild deutlich früher vom Rendering dargestellt werden als die zugehörigen, vom Tracking für dieses Videobild berechneten Transformationen – Videobild und Overlay-Bild wären nicht mehr synchron zueinander. Über den Videoframe-Outslot des Tracking-Knotens wird dagegen das Videobild gleichzeitig mit den aus ihm berechneten Transformationen verschickt. Der zusätzliche Outslot stellt also ein einfaches Mittel dar, um die Videobilder mit den Transformationen zu synchonisieren.

Abbildung 56 stellt den Datenflußgraphen des Archeoguide-Systems abstrakt dar. In der Praxis wird seine Struktur in Form einer XML-Datei gespeichert und beim Start des Systems wiederhergestellt. Diese XML-Datei sieht folgendermaßen aus:

XML-Darstellung des Datenflußgraphen

```xml
<?xml version="1.0"?>
<HID versionMajor="1" versionMinor="0">
  <Node type="NMEA" label="GPS">
    <Parameter name="Device" value="0"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Node type="WGS2UTM" label="WGS2UTM">
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Route from="GPS/Position" to="WGS2UTM/WGSPosition"/>
  <ExternalRoute internal="WGS2UTM/UTMPosition" external="Position"/>
  <Node type="NMEA" label="Compass">
    <Parameter name="Device" value="1"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <ExternalRoute internal="Compass/Orientation" external="Orientation"/>
  <Node type="VB30" label="VB30">
    <Parameter name="Device" value="2"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <ExternalRoute internal="VB30/Plus-Button" external="Plus-Button"/>
  <ExternalRoute internal="VB30/Minus-Button" external="Minus-Button"/>
  <ExternalRoute internal="VB30/Select-Button" external="Select-Button"/>
  <Node type="Framegrabber" label="Framegrabber">
    <Parameter name="Device" value="0"/>
    <Parameter name="Width" value="320"/>
    <Parameter name="Height" value="240"/>
    <Parameter name="Format" value="BGR24"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Node type="Tracking" label="Tracking">
    <Parameter name="URL" value="http://pc-server/tracking.db"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Route from="Framegrabber/Frames" to="Tracking/Frames in"/>
  <ExternalRoute internal="Tracking/Viewpoint" external="Viewpoint"/>
  <ExternalRoute internal="Tracking/Frames out" external="Frames"/>
  <ExternalRoute internal="Tracking/Overlay" external="Overlay"/>
  <ExternalRoute internal="Tracking/Translation" external="Translation"/>
  <ExternalRoute internal="Tracking/Rotation" external="Rotation"/>
  <ExternalRoute internal="Tracking/Scale" external="Scale"/>
</HID>
```

Man findet in der XML-Datei alle Elemente des Datenflußgraphen in Abbildung 56 wieder. Das XML-Tag „Node“ erzeugt die Knoten des Graphen. Es hat zwei Attribute, „type“ und „label“, die den Typ des Knoten (NMEA, WGS2UTM, Framegrabber etc.) festlegen und seinen eindeutigen Namen im Graphen (z.B. „GPS“ für den NMEA-Knoten, der den GPS-Empfänger ausliest, oder „Compass“ für den NMEA-Knoten, der den Kompaß ausliest). Innerhalb der Node-Tags werden die Parameter des jeweiligen Knotens festgelegt, z.B. die Nummer der seriellen Schnittstelle beim NMEA-Knoten und beim VB30-Knoten (über den „Device“-Parameter), die Auflösung und das Format des Videobildes beim Framegrabber-Knoten, oder die URL der Datenbank für das Videotracking. Darüber hinaus werden alle Slots der Knoten über ExternalRoutes dem Graphen zur Verfügung gestellt.

Neben den Node-Tags findet man noch „Route“- und „ExternalRoute“-Tags. Route-Tags verbinden Outslots und Inslot im Datenflußgraphen. So findet man eine Route, die den „Position“-Outslot des „GPS“-Knotens mit dem „WGSPosition“-Inslot des „WGS2UTM“-Knotens verbindet, sowie eine Route, die den „Frames“-Outslot des „Framegrabber“-Knotens mit dem „Frames In“-Inslot des „Tracking“-Knotens verbindet. Die Outslots und Inslots, die für die Anwendung aus dem Datenflußgraphen exportiert werden, werden über ExternalRoute-Tags spezifiziert. So kann man sehen, daß u.a. der „Viewpoint“-Inslot des „Tracking“-Knotens unter dem Namen „Viewpoint“ für die Anwendung zur Verfügung steht, und der „Frames out“-Outslot des „Tracking“-Knotens unter dem Namen „Frames“.

Die eigentliche Archeoguide-Anwendung liegt in der Form von VRML-Code vor. Selbstverständlich kann hier nicht der gesamte Code abgedruckt werden, sondern nur eine stark vereinfachte Version, die zur besseren Lesbarkeit in mehrere Abschnitte eingeteilt wird:

VRML-Code (1. Abschnitt)

```
Shape {
  appearance Appearance {
    texture DEF videoImage PixelTexture {}
  }
  geometry IndexedFaceSet {
    ... # Eine Ebene, auf die das Videobild projeziert wird
  }
}

DEF videoImageSensor SFImageSensor {
  label "Frames"
}

ROUTE videoImageSensor.value_changed TO videoImage.set_image
```

Der erste Abschnitt des VRML-Codes dient der Darstellung des aktuellen Videobildes. Zuerst wird eine Geometrie erzeugt, auf das eine Textur gemappt wird. Die verwendete Geometrie ist im einfachsten Fall eine rechteckige Ebene, sie kann aber auch komplizierter sein, um z.B. eine Verzerrung des Kamerabildes auszugleichen. Nach der Geometrie wird ein SFImageSensor mit dem Label „Frames“ erzeugt. Dieser Sensor fügt die Bilder in die VRML-Szene ein, die den Datenflußgraphen über den „Frames“-Outslot verlassen. Eine Route verbindet den „value_changed“-Outslot des Sensors mit dem „set_image“-Inslot der Textur, die auf die Geometrie gemappt wird. Das Ergebnis dieses Code-Abschnitts ist also eine Geometrie im VRML-Szenengraphen, auf die das aktuelle Videobild gemappt wird.

VRML-Code (2. Abschnitt)

```
DEF overlayTransform Transform {
  children [
    Shape {
      appearance DEF overlayAppearance Appearance {}
      geometry IndexedFaceSet {
        ... # Eine Ebene, auf die das Overlaybild projeziert wird
      }
    }
  ]
}

DEF translationSensor SFVec3fSensor {
  label "Translation"
}

ROUTE translationSensor.value_changed TO overlayTransform.set_translation

DEF rotationSensor SFRotationSensor {
  label "Rotation"
}

ROUTE rotationSensor.value_changed TO overlayTransform.set_rotation

DEF scaleSensor SFVec3fSensor {
  label "Scale"
}

ROUTE scaleSensor.value_changed TO overlayTransform.set_scale

DEF overlayScript Script {
  eventIn SFString set_overlayURL
  eventOut SFNode overlayNode_changed
  url [
    ... # URL des Script-Codes
  ]
}

ROUTE overlayScript.overlayNode_changed TO overlayAppearance.set_texture

DEF overlaySensor SFStringSensor {
  label "Overlay"
}

ROUTE overlaySensor.value_changed TO overlayScript.set_overlayURL
```

Der zweite Abschnitt des VRML-Codes dient der Darstellung des Overlay-Bildes. Zunächst wird wieder eine Geometrie erzeugt, auf die das Overlay-Bild gemappt wird. Die Geometrie ist ein einfaches Rechteck, das so vor das Videobild aus dem ersten Abschnitt des VRML-Codes plaziert wird, das beide Bilder zur Deckung kommen.

Die Overlay-Geometrie ist Kind eines Transform-Knotens („overlayTransform“). Dieser Transform-Knoten dient dazu, die Overlay-Geometrie entsprechend der Ergebnisse des Tracking-Prozesses zu transformieren, sodaß sich das Overlay-Bild immer an der korrekten Position über dem aktuellen Videobild befindet. Zu diesem Zweck gibt es drei Sensorknoten („translationSensor“, „rotationSensor“ und „scaleSensor“), die die aktuelle Translation, Rotation und Skalierung des Overlaybildes vom Datenflußgraphen erhalten. Diese drei Parameter werden über Routes an die entsprechenden Felder des Transformknotens weitergeleitet.

Am Ende des zweiten Abschnitts des VRML-Codes befinden sich noch ein Script-Knoten und ein weiterer Sensor-Knoten. Wie weiter oben beschrieben, existiert zu jedem Referenzbild, das vom Videotracking zur Bestimmung der aktuellen Position und Orientierung benutzt wird, ein zugehöriges Overlaybild. Die URL dieses Overlaybildes wird vom Tracking-Knoten im Datenflußgraphen über einen speziellen Outslot dem System zur Verfügung gestellt. Der „overlaySensor“-Knoten im VRML-Code oben empfängt diese URL und leitet sie über eine Route an einen Script-Knoten („overlayScript“) weiter.

Das Skript erscheint auf den ersten Blick überflüssig. Man könnte einen ImageTexture-VRML-Knoten in die Szene einfügen und die URL direkt an dessen URL-Feld weiterleiten. Dies wäre zwar theoretisch möglich, wurde aber aus zwei Gründen nicht durchgeführt:

1. Es wäre extrem ineffizient gewesen. Es kann zu jedem Aussichtspunkt beliebig viele Referenzbilder und damit beliebig viele Overlaybilder geben. Während sich der Anwender an einem Aussichtspunkt befindet, sich bewegt, verschiedene Positionen einnimmt und seine Blickrichtung wechselt, wird ständig das Referenzbild und damit das Overlaybild gewechselt. Würde man die URL direkt in einen ImageTexture-Knoten routen, würde dieser Knoten jedesmal das Overlaybild wieder herunterladen und dekodieren, was die Darstellung spürbar verzögern würde.
2. Wie bereits oben beschrieben, bestehen die Overlays nicht nur aus Bildern, sondern auch aus kurzen Filmsequenzen. So kann man von den Aussichtspunkten am Rande des alten Olympia-Stadions aus auch Sportler bei der Ausübung historischer Sportarten beobachten (Abbildung 52). Für die Darstellung von Filmen benötigt man jedoch keinen ImageTexture-, sondern einen MovieTexture-VRML-Knoten.

Aus diesen beiden Gründen muß der Umweg über einen kleinen Script-Knoten erfolgen. Er erhält die URL des aktuellen Overlays vom Datenflußgraphen. Zunächst schaut er in einem Cache nach, ob für diese URL bereits ein VRML-Knoten zur Darstellung erzeugt wurde. Wenn ja, wird dieser Knoten über den „overlayNode_changed“-Outslot des Script-Knotens und eine Route an das „texture“-Feld des „overlayAppearance“-Knotens geschickt und auf diese Weise in den VRML-Szenengraphen eingefügt. Wenn nein, muß ein neuer VRML-Knoten erzeugt werden. Zunächst wird aus der Dateiendung der URL bestimmt, ob es sich um ein Standbild oder um einen Film handelt. Entsprechend wird ein ImageTexture- oder MovieTexture-VRML-Knoten erzeugt. Dieser Knoten wird in den Cache eingefügt und wie oben beschrieben in die VRML-Szene eingefügt.

Der Umweg über einen Script-Knoten erlaubt es also, sehr effizient zwischen verschiedenen Overlays hin- und herzuschalten und Standbilder sowie Filme mit ein- und demselben VRML-Code zu behandeln. Während sich der Anwender an einem Aussichtspunkt befindet und sich dort bewegt, wird der Cache der Texturknoten nach und nach mit den für die Darstellung der Overlays benötigten VRML-Knoten gefüllt. Selbstverständlich muß dieser Cache irgendwann auch wieder geleert werden. Dies geschieht, wenn der Anwender den Aussichtspunkt wieder verläßt. Der dazu notwendige VRML-Code wird aus Gründen der Übersichtlichkeit hier nicht abgedruckt. Ausgelöst wird dieses Löschen des Caches von der Controller-Komponente, die sich im dritten Abschnitt des VRML-Codes befindet.

VRML-Code (3. Abschnitt)

```
DEF controller Script {
  eventIn SFVec3f set_position
  eventIn SFRotation set_rotation
  eventOut SFString viewpoint_changed
  eventIn SFBool set_plusButton
  eventIn SFBool set_minusButton
  eventIn SFBool set_selectButton
  ... # Weitere Slots
  url [
    ... # URL des Script-Codes
  ]
}

DEF positionSensor SFVec3fSensor {
  label "Position"
}

ROUTE positionSensor.value_changed TO controller.set_position

DEF orientationSensor SFRotationSensor {
  label "Orientation"
}

ROUTE orientationSensor.value_changed TO controller.set_orientation

DEF viewpointSensor SFStringSensor {
  label "Viewpoint"
}

ROUTE controller.viewpoint_changed TO viewpointSensor.set_value

DEF plusButtonSensor SFBoolSensor {
  label "Plus-Button"
}

ROUTE plusButtonSensor.value_changed TO controller.set_plusButton

DEF minusButtonSensor SFBoolSensor {
  label "Minus-Button"
}

ROUTE minusButtonSensor.value_changed TO controller.set_minusButton

DEF selectButtonSensor SFBoolSensor {
  label "Select-Button"
}

ROUTE selectButtonSensor.value_changed TO controller.set_selectButton
```

Der dritte Abschnitt enthält den Code für die Controller-Komponente, die das Archeoguide-System steuert. Implementiert ist die Controller-Komponente in Java. Sie wird über einen VRML-Script-Knoten in die VRML-Szene eingefügt. Zwei Sensor-Knoten („positionSensor“ und „orientationSensor“) empfangen die aktuelle Position und Orientierung des Anwenders vom Datenflußgraphen und leiten sie an den Controller. Der Controller berechnet aus diesen Informationen den aktuellen Aussichtspunkt, an dem sich der Anwender befindet, und schickt den Identifier-String des Aussichtspunktes über einen weiteren Sensor („viewpointSensor“) wieder zurück in den Datenflußgraphen. Innerhalb des Datenflußgraphen wird diese Information wie oben beschrieben vom Videotracking-Knoten benötigt, um die zum Aussichtspunkt gehörigen Referenz- und Overlaybilder aus der Datenbank zu holen.

Abbildung 57: Transparenzregelung des Overlay-Bildes

Schließlich befinden sich im dritten Abschnitt noch drei Sensor-Knoten („plusButtonSensor“, „minusButtonSensor“ und „selectButtonSensor“, die den Zustand der Knöpfe vom nVision VB-30 Head Mounted Display aus dem Datenflußgraphen empfangen und an das Controller-Script weiterleiten. Über diese drei Buttons wird das User-Interface des Systems gesteuert, d.h. man konnte z.B. die Transparenz des Overlay-Bildes verändern (Abbildung 57), die Lautstärke der Audioausgabe regeln, Landkarten des Geländes einblenden etc. Das gesamte User-Interface ist Teil des VRML-Szenegraphen und wird über Inslots und Outslots vom Controller-Script gesteuert. Der zugehörige Code wird hier aus Gründen der Übersichtlichkeit nicht abgedruckt.

7.1.3 Modifikationen des Archeoguide-Systems

Im vorhergehenden Abschnitt wurde die Standard-Konfiguration des Archeoguide-Systems beschrieben, bestehend aus nVision VB-30, GPS und Kompaß. Ausgehend von dieser Standard-Konfiguration gibt es mehrere abgeleitete Konfigurationen, die sich leicht von der Standardkonfiguration unterscheiden. Die dafür notwendigen Modifikationen werden in diesem Abschnitt beschrieben, weil sie verdeutlichen, wie man mit geringem Aufwand das System an diese neuen Konfigurationen anpassen konnte.

Die erste Modifikation war, daß anstelle des nVision VB-30 Head Mounted Displays ein anderes Produkt eingesetzt wurde, nämlich ein Sony Glasstron HMD. Dies machte zwar keine Änderungen beim Rendering notwendig, das neue HMD wurde einfach anstelle des alten an den Videoausgang des Rechners angeschlossen. Allerdings besitzt das Glasstron HMD im Gegensatz zum VB-30 keine Buttons, über die das User-Interface bedient werden konnte. Anstelle dessen wurde ein einfaches Gamepad an das System angeschlossen. Dieses Gamepad wird nicht wie die Buttons des VB-30 über eine serielle Schnittstelle angeschlossen, sondern über USB. Der Zugriff auf das Gamepad erfolgt über die Microsoft DirectInput-Schnittstelle.

Um das Archeoguide-System an diese Änderung anzupassen, war eine kleine Modifikation des Datenflußgraphen notwendig. Der „VB30“-Knoten wurde entfernt, anstelle dessen wurde ein „Joystick“-Knoten in den Graphen eingefügt. Die ersten drei Outslots des Joystickknotens, die die Stellungen der Gamepad-Buttons liefern („Joystick/Button #1“, „Joystick/Button #2“ und „Joystick/Button #3“) wurden unter dem gleichen Namen wie die VB-30-Buttons („Plus-Button“, „Minus-Button“ und „Select-Button“) aus dem Datenflußgraphen exportiert. Die neuen Bestandteile der XML-Datei, die den Datenflußgraphen beschreibt, sehen also folgendermaßen aus:

Neue Bestandteile der XML-Darstellung des Datenflußgraphen für die Benutzung eines Gamepads anstelle der Buttons des VB-30 HMD

```
...

<Node type="Joystick" label="Joystick">
  <Parameter name="Device" value="0"/>
  <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
</Node>
<ExternalRoute internal="Joystick/Button #1" external="Plus-Button"/>
<ExternalRoute internal="Joystick/Button #2" external="Minus-Button"/>
<ExternalRoute internal="Joystick/Button #3" external="Select-Button"/>

...
```

Genau genommen ist es gar nicht notwendig, den VB30-Knoten aus dem Datenflußgraphen zu entfernen. VB30-Knoten und Joystick-Knoten könnten auch problemlos nebeneinander existieren. Wenn das VB-30-HMD angeschlossen ist, liefert der VB30-Knoten die Button-Stellungen, während der Joystick-Knoten kein Gamepad findet und seine Arbeit daher nicht aufnimmt. Ist dagegen ein Gamepad angeschlossen, so ist es genau andersherum: Der VB30-Knoten findet kein VB-30-HMD und nimmt seine Arbeit nicht auf, während der Joystick-Knoten die Button-Stellungen des Gamepads liefert. Man erinnere sich, daß es das Konzept des Datenflußgraphen erlaubt, daß sich beliebing viele Outslots mit beliebig vielen Inslots verbinden können. In diesem Fall sind die Button-Inslots des VRML-Szenengraphen mit jeweils zwei Outslots (vom VB30-Knoten und vom Joystick-Knoten) verbunden, von denen aber je nach Hardwarekonfiguration nur jeweils einer Werte liefert. Es ist also ohne weiteres möglich, bei geschicktem Aufbau des Datenflußgraphen mehrere Hardwarekonfigurationen mit ein- und demselben Graphen zu behandeln.

Man beachte darüber hinaus auch, daß sich die notwendigen Modifikationen auf den Datenflußgraphen beschränken. Die VRML-Szene und die in ihr enthaltenen Skripte werden nicht verändert. Das bedeutet, daß die eigentliche Anwendung nicht modifiziert werden muß, wenn sich die Hardwarekonfiguration ändert – ein wichtiger Vorteil von Systemen, die auf einem Datenflußkonzept beruhen, das Änderungen an der Hardware weitgehend vor der Anwendung verbergen kann.

Eine weitere Modifikation des ursprünglichen Archeoguide-Systems bestand im Einsatz von Differential GPS (DGPS). DGPS ist eine Technik, die es erlaubt, die Genauigkeit der vom GPS-Empfänger gelieferten Positionsdaten drastisch zu vergrößern. Dazu setzt man eine Basisstation ein, die sich an einem genau bekannten Standpunkt befindet und ständig ihre Position per GPS bestimmt. Da der korrekte Standpunkt bekannt ist, kann die Basisstation die Abweichung der GPS-Position vom tatsächlichen Standpunkt und somit den aktuellen Fehler des GPS-Systems bestimmen. Da dieser Fehler für alle GPS-Empfänger, die sich in der Nähe der Basisstation befinden, identisch ist, kann die Basisstation Korrekturdaten an diese GPS-Empfänger übermitteln, die es diesen erlauben, den Fehler ihrer eigenen Positionsbestimmung zu verringern. DGPS-Systeme sind also nicht mehr nur von den GPS-Satelliten abhängig, sondern auch von zusätzlicher Hardware am Boden (der Basisstation). In der näheren Umgebung dieser Basisstation können mobile Geräte dadurch ihre Position mit drastisch erhöhter Genauigkeit bestimmen.

Es stellt sich jedoch das Problem, wie man die Korrekturdaten der Basisstation an die GPS-Empfänger übermittelt. Das Format der Korrekturdaten selbst ist im RTCM-Standard festgelegt, d.h. die von der Basisstation gelieferten Korrekturdaten können von allen GPS-Empfängern, die RTCM unterstützen, direkt verarbeitet werden. Auch für die Übertragung der RTCM-Daten gibt es ein Standardverfahren, bei dem die Basisstation ein Funksignal ausstrahlt, das von GPS-Empfängern empfangen werden kann. Leider besitzen nur wenige hochpreisige High-End-GPS-Empfänger eine solche Empfangseinheit für RTCM-Daten. Da das Archeoguide-System aber ohnedies schon über ein WLAN mit einem zentralen Server verbunden ist, wurde entschieden, die Korrekturdaten der Basisstation über dieses WLAN zu übertragen und damit die Kosten der mobilen AR-Systeme gering zu halten.

Der Umsetzung dieses Konzeptes kommt zugute, daß der Datenflußgraph völlig netzwerktransparent ist. Es wurden daher zwei neue Knotentypen implementiert, nämlich der „RTCMReader“ und der „RTCMWriter“. Der RTCM-Reader liest die RTCM-Korrekturdaten von einer an die serielle Schnittstelle des Archeoguide-Servers angeschlossenen DGPS-Basisstation und verschickt sie über einen „Data“-Outslot. Der RTCM-Writer empfängt RTCM-Korrekturdaten über einen „Data“-Inslot und überträgt sie an einen an die serielle Schnittstelle angeschlossenen GPS-Empfänger (diese serielle Schnittstelle ist bei allen bekannten GPS-Empfängern nicht identisch mit der Schnittstelle, die die NMEA-Daten liefert – es tritt daher kein Zugriffskonflikt mit dem NMEA-Knoten auf). Dazu wurde ein spezieller Datentyp („RTCMData“) implementiert, der Blöcke von RTCM-Daten aufnehmen kann. Wie im 3. Kapitel beschrieben, unterstützt der Datenflußgraph beliebige Datentypen und nicht nur Geräteklassen. An diesem Beispiel kann man sehen, wie wichtig dieses Grundkonzept in der Praxis ist – auf welche herkömmliche Geräteklasse hätte man die RTCM-Datenblöcke abbilden können? Darüber hinaus wurden wie im 4. Kapitel beschrieben ein Encoder und ein Decoder geschrieben, die die RTCM-Datenblöcke in eine netzwerk-transparente Darstellung umwandelten und aus einer solchen wieder rekonstruierten.

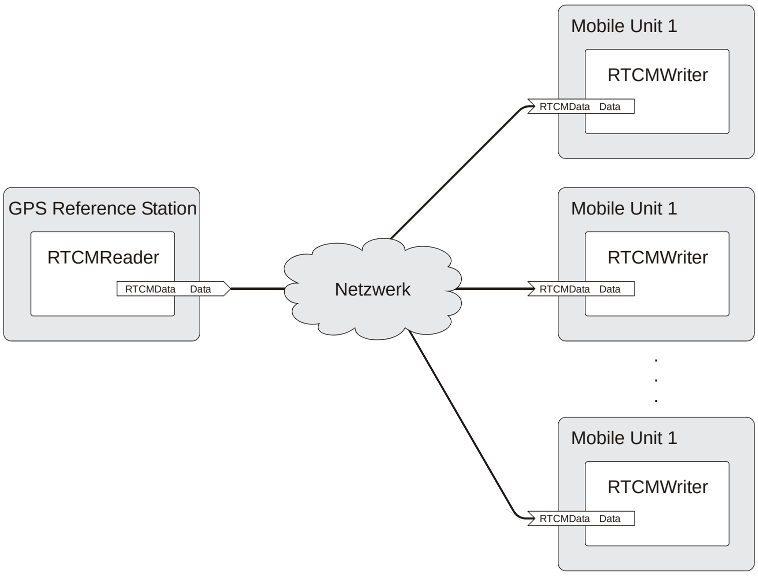


Abbildung 58: Übertragung von RTCM-Daten über das WLAN

Die Übertragung der RTCM-Daten ist jetzt mit Leichtigkeit zu bewerkstelligen. Auf dem Server wird ein Datenflußgraph, bestehend aus RTCMReader-Knoten und Network-Knoten (der die Netzwerktransparenz herstellt), aufgebaut. Auf den mobilen AR-Geräten wird der vorhandene Datenflußgraph um einen RTCMWriter-Knoten und ebenfalls um einen Network-Knoten erweitert. Darüber hinaus wird auf den mobilen Geräten eine Route angelegt, die die Outslots des RTCMReader-Knotens auf dem Server über das Netzwerk hinweg mit den Inslots der RTCMWriter-Knoten verbindet. Abbildung 58 stellt dies schematisch dar. Auch an dieser Stelle sei darauf hingewiesen, daß ein Outslot mit beliebig vielen Inslots verbunden sein kann. Daten, die in einen solchen Outslot geschrieben werden, werden an alle angeschlossenen Inslots übertragen.

Die XML-Beschreibung des Datenflußgraphen auf der Server-Seite sieht folgendermaßen aus:

XML-Beschreibung des Datenflußgraphen der DGPS-Basisstation

```
<?xml version="1.0"?>
<HID versionMajor="1" versionMinor="0">
  <Node type="RTCMReader" label="RTCMReader">
    <Parameter name="Device" value="0"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Node type="Network" label="Network">
    <Parameter name="Prefix" value="DGPSBase/{SlotLabel}"/>
    <ExternalRoute internal="*" external="{SlotLabel}"/>
  </Node>
</HID>
```

Auf der Seite der mobilen AR-Geräte wird folgendes zur XML-Beschreibung des Datenflußgraphen hinzugefügt:

Neue Bestandteile der XML-Darstellung des Datenflußgraphen auf den mobilen für den Einsatz von DGPS

```
...

<Node type="RTCMWriter" label="RTCMWriter">
  <Parameter name="Device" value="3"/>
  <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
</Node>
<Route from="DGPSBase/RTCMReader/Data" to="RTCMWriter/Data"/>
<Node type="Network" label="Network">
  <ExternalRoute internal="*" external="{SlotLabel}"/>
</Node>

...
```

Der auf der Serverseite vorhandene Outslot „RTCMReader/Data“ wird über die auf Server- und Clientseite vorhandenen Networkknoten unter dem Namen „DGPSBase/RTCMReader/Data“ auf allen mobilen AR-Geräten in den Datenflußgraphen eingefügt. Eine Route verbindet den Outslot mit dem jeweiligen „RTCMWriter/Data“-Inslot des mobilen AR-Geräts. Auf diese Weise werden die RTCM-Korrekturdaten an die jeweiligen GPS-Empfänger der Geräte übertragen.

Auch bei dieser Erweiterung des Systems kann man wiederum zwei Punkte feststellen:

1. Es ist keine Modifikation der eigentlichen Anwendungslogik im VRML-Code und den zugehörigen Scripten notwendig. Die Änderungen finden ausschließlich im Datenflußgraphen statt.
2. Die vorgenommenen Änderungen am Datenflußgraphen verhindern nicht die Funktionsfähigkeit des Systems, wenn am Einsatzort des Archeoguide-Systems keine DGPS-Basisstation vorhanden ist, oder wenn der von den mobilen AR-Geräten eingesetzte GPS-Empfänger keine RTCM-Daten verarbeiten kann. Im ersten Fall gibt es einfach keinen Outslot, der die RTCM-Daten liefert. Die Inslots der RTCM-Writer auf den mobilen Einheiten empfangen keine RTCM-Daten, und es werden keine RTCM-Daten über die serielle Schnittstelle an den GPS-Empfänger übertragen. Das verhindert nicht die korrekte Funktion des Gesamtsystems. Im zweiten Fall werden die RTCM-Writer seine Arbeit nicht aufnehmen, und die auf den Inslots eintreffenden RTCM-Daten werden ignoriert.

7.2 MARIO

Abbildung 59: Screenshot der MARIO-Anwendung (links), Anwender (rechts)

Das zweite Anwendungsbeispiel ist das „MARIO“-System [9]. MARIO ist ein Indoor-System, das einen klassischen Anwendungsfall für AR abdeckt, nämlich Wartung von komplexen technischen Systemen und Maschinen. Abbildung 59 zeigt auf der linken Seite einen Screenshot der MARIO-Anwendung und auf der rechten Seite einen Anwender des Systems. Die Idee hinter dem System ist es, Wartungstechniker in Zukunft mit leichten, tragbaren AR-Systemen auszurüsten, auf denen Handbücher sowie alle zur Reparatur einer Maschine benötigten Arbeitsschritte gespeichert sind. Über ein Head Mounted Display (HMD) bekommt der Wartungstechniker passgenau in sein aktuelles Blickfeld eingeblendet, welche Schrauben er lösen, welche Schalter er umlegen muß etc.

Ähnlich wie das Archeoguide-System ist MARIO ein „Video-see-through“-System, d.h. es ist eine Videokamera am HMD befestigt. Diese Videokamera liefert zum einen aktuelle Bilder vom Blickfeld des Anwenders, die hinter die künstlichen, vom AR-System in die Szene eingeblendeten Objekte projeziert werden. Zum anderen liefert sie Bilder für ein Video-Tracking-System, das die Position und Blickrichtung des Anwenders in Bezug auf die Maschine bestimmt. Im Gegensatz zum Archeoguide-System wird bei MARIO ein markerbasiertes Tracking verwendet, d.h. auf der Maschine sind Marker befestigt, deren Position dem System genau bekannt ist. Für das Tracking wurde das bekannte „AR-Toolkit“ verwendet, das im AR-Bereich den Standard für markerbasiertes Tracking darstellt und frei im Sourcecode verfügbar ist. AR-Toolkit wurde dabei als Knoten in das Gerätemanagement-System eingebaut.

7.2.1 Umsetzung

Wie beim Archeoguide-System betrachten wir zunächst die schematische Darstellung des Datenflußgraphen (Abbildung 60).

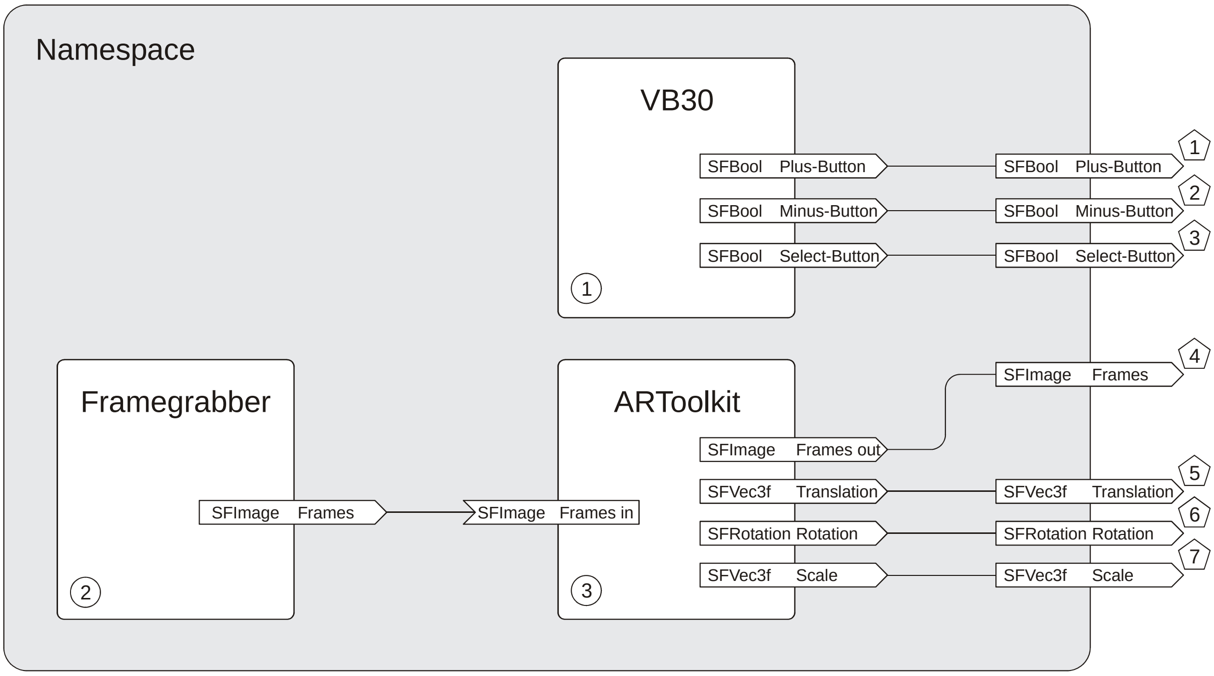


Abbildung 60: Datenflußgraph des MARIO-Systems

Der Datenflußgraph weist eine gewisse Ähnlichkeit mit dem des Archeoguide-Systems auf. Die Knoten zum Betreiben von GPS-Empfänger und Kompaß fehlen natürlich. Wie beim Archeoguide-System wird das nVision VB-30 HMD verwendet, dessen drei Buttons wiederum zu Steuerung es User-Interfaces verwendet werden. Ein Framegrabber-Knoten liefert erneut die Videobilder. Neu ist der „ARToolkit“-Knoten. Er erhält über eine Route die Videobilder des Framegrabber-Knotens, extrahiert aus den Bildern die Marker und berechnet aus der Position der Marker in den Videobildern die Position und Orientierung des Anwenders relativ zu der Maschine, an der die Marker befestigt sind. Diese Informationen werden verwendet, um die künstlichen 3D-Objekte (Augmentierungen) passgenau in das Blickfeld des Anwenders einzublenden. Wie beim VideoTracking-Knoten des Archeoguide-Systems wird dabei ein einfacher Trick angewandt, um sicherzustellen, daß die 3D-Objekte und das Videobild immer synchron zueinander sind: Der ARToolkit-Knoten besitzt einen eigenen Outslot für die Videobilder. Wann immer er aus einem Videobild die aktuelle Transformation bestimmt hat, schreibt er das Bild gleichzeitig mit der Translation, Rotation und Skalierung in diesen Video-Outslot. Die Anwendung verwendet diesen Video-Outslot anstelle des ursprünglichen Video-Outslots des Framegrabber-Knotens und stellt so die Synchronität von Videobild und Augmentierungen sicher.

Die XML-Darstellung dieses Datenflußgraphen sieht folgendermaßen aus:

XML-Darstellung des MARIO-Datenflußgraphen

```xml
<?xml version="1.0"?>
<HID versionMajor="1" versionMinor="0">
  <Node type="VB30" label="VB30">
    <Parameter name="Device" value="0"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <ExternalRoute internal="VB30/Plus-Button" external="Plus-Button"/>
  <ExternalRoute internal="VB30/Minus-Button" external="Minus-Button"/>
  <ExternalRoute internal="VB30/Select-Button" external="Select-Button"/>
  <Node type="Framegrabber" label="Framegrabber">
    <Parameter name="Device" value="0"/>
    <Parameter name="Width" value="320"/>
    <Parameter name="Height" value="240"/>
    <Parameter name="Format" value="BGR24"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Node type="ARToolkit" label="ARToolkit">
    <Parameter name="URL" value="http://localhost/tracking.conf"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Route from="Framegrabber/Frames" to="ARToolkit/Frames in"/>
  <ExternalRoute internal="ARToolkit/Frames out" external="Frames"/>
  <ExternalRoute internal="ARToolkit/Translation" external="Translation"/>
  <ExternalRoute internal="ARToolkit/Rotation" external="Rotation"/>
  <ExternalRoute internal="ARToolkit/Scale" external="Scale"/>
</HID>
```

Zunächst wird der VB30-Knoten erzeugt. Er erhält als Parameter die Nummer der seriellen Schnittstelle, an die die Buttons des VB-30 HMDs angeschlossen sind. Die drei Outslots des VB30-Knotens, die die Stellung der drei Buttons liefern, werden dann über ExternalRoutes aus dem Datenflußgraphen exportiert und damit der Anwendung zur Verfügung gestellt.

Danach wird der Framegrabber-Knoten erzeugt, der die Videobilder liefert. Auch dieser Knoten hat eine Reihe von Parametern, die festlegen, welche Kamera verwendet wird, wie groß die Videobilder sein und welches Format sie haben sollen.

Zuletzt wird der ARToolkit-Knoten erzeugt. Er hat als Parameter die URL einer Konfigurationsdatei. In dieser Konfiguration werden die verwendeten Marker und ihre genaue Position auf der Maschine festgelegt. Über eine Route wird der „Frames in“-Inslot des ARToolkit-Knotens mit dem „Frames“-Outslot des Framegrabber-Knotens verbunden. Der „Frames out“-Outslot, der das letzte vom Videotracking verwendete Videobild liefert, sowie die drei Outslots, die die vom Tracking berechnete Translation, Rotation und Skalierung liefern, werden über ExternalRoutes aus dem Datenflußgraphen exportiert und so der Anwendung zur Verfügung gestellt.

Der VRML-Code der MARIO-Anwendung wird aus Gründen der Übersichtlichkeit erneut in drei Abschnitte aufgeteilt.

VRML-Code (1. Abschnitt)

```
Shape {
  appearance Appearance {
    texture DEF videoImage PixelTexture {}
  }
  geometry IndexedFaceSet {
    ... # Eine Ebene, auf die das Videobild projeziert wird
  }
}

DEF videoImageSensor SFImageSensor {
  label "Frames"
}

ROUTE videoImageSensor.value_changed TO videoImage.set_image
```

Der erste Abschnitt des VRML-Codes dient erneut dazu, das Videobild hinter die VRML-Szene zu projezieren. Dazu wird zunächst eine Geometrie erzeugt, auf die das aktuelle Videobild als Textur gemappt wird. Diese Geometrie ist im einfachsten Falle einfach eine rechteckige Ebene, kann aber auch kompliziertere Formen annehmen, um eventuelle Verzerrungen des Kamerabildes auszugleichen.

Hinter dem Geometrieknoten befindet sich ein Sensorknoten, der das aktuelle Videobild vom Datenflußgraphen empfängt und der VRML-Szene zur Verfügung stellt. Eine Route verbindet den „value_changed“-Outslot des Sensors mit dem „set_image“-Inslot des Texturknotens der Geometrie.

VRML-Code (2. Abschnitt)

```
DEF scene Transform {
  children [
    ... # 3D-Objekte der AR-Szene
  ]
}

DEF translationSensor SFVec3fSensor {
  label "Translation"
}

ROUTE translationSensor.value_changed TO scene.set_translation

DEF rotationSensor SFRotationSensor {
  label "Rotation"
}

ROUTE rotationSensor.value_changed TO scene.set_rotation

DEF scaleSensor SFVec3fSensor {
  label "Scale"
}

ROUTE scaleSensor.value_changed TO scene.set_scale
```

Der zweite Abschnitt des VRML-Codes beinhaltet die Augmentierungen, d.h. die künstlichen 3D-Objekte, die passgenau in das Blickfeld des Anwenders eingeblendet werden. Zunächst wird ein Transform-Knoten („scene“) erzeugt, unterhalb dessen die Augmentierungen in den Szenengraphen eingehängt werden. Drei Sensorknoten empfangen die aktuelle Translation, Rotation und Skalierung vom ARToolkit-Knoten im Datenflußgraphen und übertragen sie über Routes an den Transform-Knoten.

VRML-Code (3. Abschnitt)

```
DEF controller Script {
  eventIn SFBool set_plusButton
  eventIn SFBool set_minusButton
  eventIn SFBool set_selectButton
  ... # Weitere Slots
  url [
    ... # URL des Script-Codes
  ]
}

DEF plusButtonSensor SFBoolSensor {
  label "Plus-Button"
}

ROUTE plusButtonSensor.value_changed TO controller.set_plusButton

DEF minusButtonSensor SFBoolSensor {
  label "Minus-Button"
}

ROUTE minusButtonSensor.value_changed TO controller.set_minusButton

DEF selectButtonSensor SFBoolSensor {
  label "Select-Button"
}

ROUTE selectButtonSensor.value_changed TO controller.set_selectButton
```

Der dritte Abschnitt des VRML-Codes enthält wiederum die Controller-Komponente in Form eines Script-Knotens. Drei Sensor-Knoten empfangen die Stellung von Plus-, Minus- und Select-Button des VB-30 HMDs vom Datenflußgraph und leiten sie über Routes an das Controller-Script weiter.

Das Controller-Script steuert das gesamte User-Interface des MARIO-Systems. Das User-Interface ist ebenfalls in VRML programmiert und Bestandteil des VRML-Szenengraphen. Der für das User-Interface notwendige Code ist hier aus Gründen der Übersichtlichkeit nicht abgedruckt.

7.2.2 Erweiterung um einen Remote-Expert-Modus

Das grundlegende MARIO-Konzept wurde im weiteren Verlauf des Projekts um einen sogenannten Remote-Expert-Modus erweitert. Die Idee beim Remote-Expert-Modus ist es, daß es vielleicht Probleme gibt, die der Wartungstechniker vor Ort mit Hilfe der im MARIO-System hinterlegten Arbeitsanweisungen nicht lösen kann. In diesem Fall könnte er sich von einem Experten unterstützen lassen, der nicht vor Ort ist, sondern sich an einem Bildschirmarbeitsplatz z.B. in seinem Büro befindet. Der Wartungstechniker nimmt von seinem mobilen AR-Gerät über eine Netzwerkverbindung Kontakt zum Arbeitsplatzrechner des Experten auf. Über einen Audiokanal können der Wartungstechniker und der Experte das Problem miteinander besprechen. Das Videobild, die aktuelle Position und Blickrichtung des Technikers sowie die Bestandteile der augmentierten Szene werden vom mobilen AR-Gerät auf den Rechner des Experten übertragen, damit dieser auf seinem Arbeitplatz genau das gleiche sehen kann, was der Techniker in seinem HMD sieht. Der Experte besitzt eine Art „Mauszeiger“, mit dem er auf Teile der augmentierten Szene deuten kann. Dieser Mauszeiger wird auch in das Blickfeld des Technikers vor Ort eingeblendet.

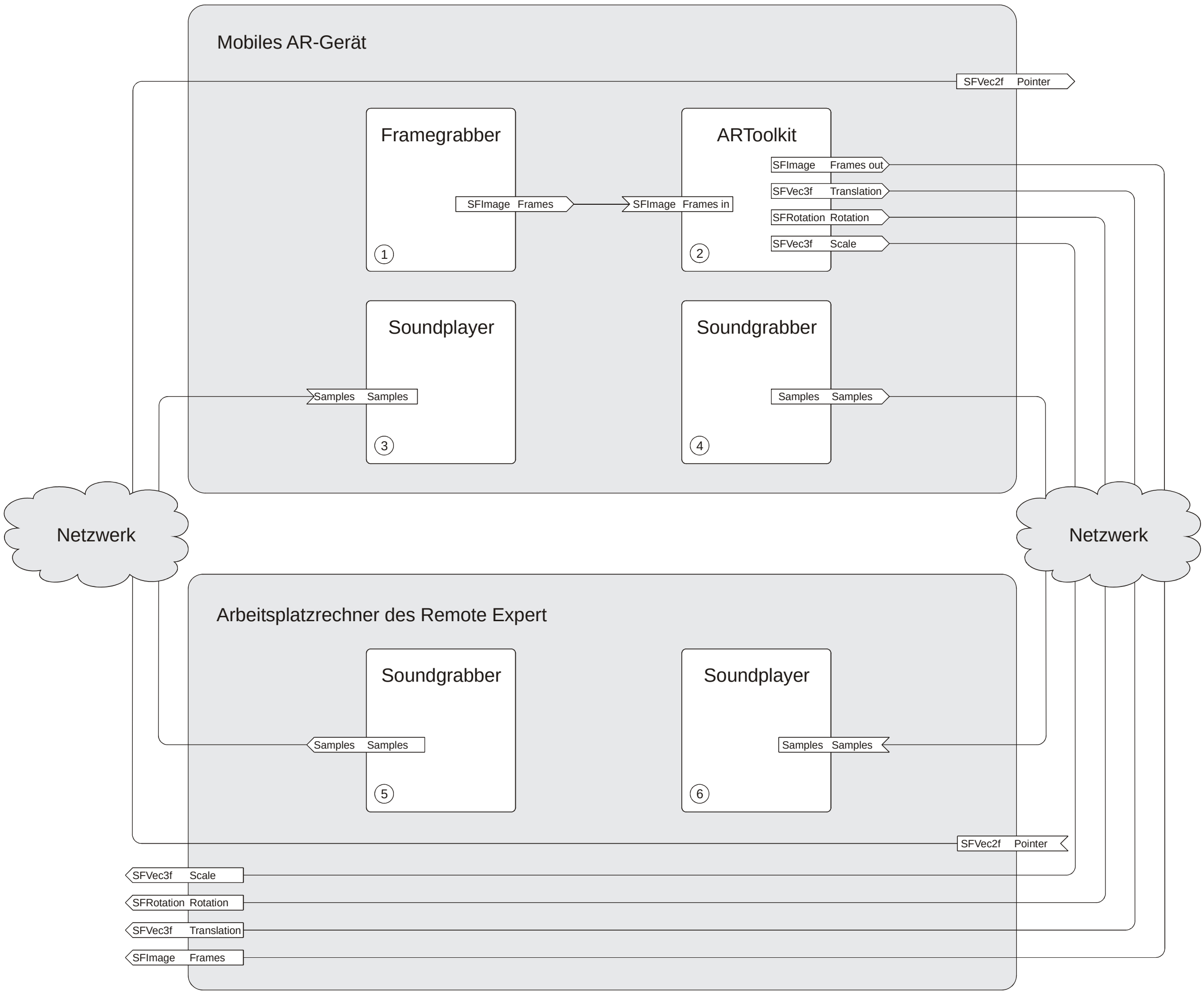


Abbildung 61: Datenflußgraph des Remote-Expert-Szenarios

Abbildung 61 zeigt den Datenflußgraphen des Remote-Expert-Szenarios. Er besteht aus zwei Teilgraphen, einem auf dem mobilen AR-Gerät (oben in der Abbildung), und einem auf dem Arbeitsplatzrechner des Experten (unten in der Abbildung).

Auf dem mobilen Gerät wird der Datenflußgraph um zwei neue Knoten erweitert, einem Soundplayer-Knoten (Nummer 3) und einem Soundgrabber-Knoten (Nummer 4). Der Soundplayer-Knoten empfängt Audio-Samples über einen Inslot und spielt sie über die Soundkarte des mobilen Rechners ab. Der Soundgrabber digitalisiert Geräusche, die über ein an der Soundkarte des mobilen Rechners angeschlossenes Mikrophon empfangen werden, und verschickt sie als Audio-Samples über einen Outslot. Entsprechende Gegenstücke gibt es im Datenflußgraphen des Experten-Rechners (Knoten Nummer 5 & 6). Der Soundgrabber-Knoten auf dem mobilen Gerät ist über das Netzwerk mit dem Soundplayer-Knoten des Experten-Rechners verbunden, und umgekehrt ist der Soundgrabber-Knoten des Experten-Rechners über das Netzwerk mit dem Soundplayer-Knoten des mobilen Geräts verbunden. Als Ergebnis erhält man eine bidirektionale Audio-Verbindung, die es dem Wartungstechniker und dem Experten ermöglicht, sich miteinander zu unterhalten.

Vom restlichen Datenflußgraphen des mobilen Geräts sind in Abbildung 61 aus Gründen der Übersichtlichkeit nur der Framegrabber-Knoten (Nummer 1) und der ARToolkit-Knoten (Nummer 2) abgebildet. Sämtliche Outslots des ARToolkit-Knotens (d.h. das aktuelle Videobild und die zugehörige Transformation) werden über das Netzwerk an den Expertenrechner exportiert und stehen dort zur Verfügung, damit auf dem Expertenrechner exakt das gleiche Bild dargestellt werden kann wie im HMD des mobilen Geräts.

Während der Experte das zu lösende Problem mit dem Wartungstechniker bespricht, sieht er also auf dem Bildschirm seines Arbeitsplatzrechners genau das gleiche wie der Techniker vor Ort. Er kann nun seine Maus verwenden, um auf einzelne Elemente des Bildes zu zeigen. Die Mausposition wird vom System über den „Pointer“-Outslot dem Datenflußgraphen zur Verfügung gestellt, über das Netzwerk an das mobile Gerät übertragen, und über den „Pointer“-Inslot vom Rendering-System des mobilen Geräts empfangen und zur Darstellung des Mauszeigers auf dem HMD genutzt.

Die XML-Darstellung des Datenflußgraphen auf dem Experten-Rechner sieht folgendemaßen aus:

XML-Darstellung des Datenflußgraphen auf dem Experten-Rechner

```
<?xml version="1.0"?>
<HID versionMajor="1" versionMinor="0">
  <Node type="Soundgrabber" label="Soundgrabber">
    <Parameter name="Device" value="0"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Route from="Soundgrabber/Samples"
         to="MobileSystem/Soundplayer/Samples"/>
  <Node type="Soundplayer" label="Soundplayer">
    <Parameter name="Device" value="0"/>
    <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
  </Node>
  <Route from="MobileSystem/Soundgrabber/Samples"
         to="Soundplayer/Samples"/>
  <Node type="Network" label="Network">
    <Parameter name="Prefix" value="ExpertSystem/{SlotLabel}"/>
    <ExternalRoute internal="*" external="{SlotLabel}"/>
  </Node>
  <Route from="MobileSystem/ARToolkit/Frames out" to="Frames"/>
  <Route from="MobileSystem/ARToolkit/Translation" to="Translation"/>
  <Route from="MobileSystem/ARToolkit/Rotation" to="Rotation"/>
  <Route from="MobileSystem/ARToolkit/Scale" to="Scale"/>
  <Route from="MobileSystem/Pointer" to="Pointer"/>
</HID>
```

Der Datenflußgraph auf dem Experten-Rechner enthält den „Soundgrabber“- und den „Soundplayer“-Knoten, wie in Abbildung 61 dargestellt. Darüber hinaus enthält er noch den Netzwerkknoten, der in Abbildung 61 nicht dargestellt ist. Der Netzwerkknoten stellt die Netzwerktransparenz her. Zum einen exportiert er alle Out- und Inslots vom Experten-Rechner auf das mobile AR-System. Dabei erhalten die Namen aller exportierten Slots auf dem mobilen System das Prefix „ExpertSystem“ (das wird durch den Parameter „Prefix“ des Netzwerkknotens festgelegt). Zum anderen importiert der Netzwerkknoten alle Out- und Inslots vom mobilen AR-System auf den Experten-Rechner. Dabei erhalten die Namen aller importierten Slots auf dem Experten-Rechner das Prefix „MobileSystem“.

Zwei Routes verbinden jeweils den Soundgrabber-Knoten auf dem Experten-System mit dem Soundplayer-Knoten auf dem mobilen System und umgekehrt den Soundgrabber-Knoten auf dem mobilen System mit dem Soundplayer-Knoten auf dem Experten-Rechner. Damit ist die bidirektionale Audio-Kommunikationsverbindung hergestellt.

Vier weitere Routes verbinden die vier Outslots des ARToolkit-Knotens auf dem mobilen System mit den InSlots des Rendering-Systems auf dem Experten-Rechner. Damit ist auch die optische Verbindung hergestellt, d.h. der Remote-Expert sieht auf seinem Bildschirm das aktuelle Videobild und die Augmentierungen, wie sie auch der Techniker vor Ort in seinem HMD sieht.

Am Schluß wird noch der „Pointer-Outslot“ der Anwendung auf dem Experten-Rechner mit dem „Pointer“-Inslot der Anwendung auf dem mobilen AR-System verbunden. Damit erhält der Remote-Expert die Möglichkeit, auf einzelne Elemente des Bildes zu zeigen. Die Position des Zeigers wird dabei an das mobile System übertragen und auch dort dargestellt.

Die XML-Darstellung des Datenflußgraphen auf dem mobilen System sieht ganz ähnlich aus. Dabei werden hier nur die zusätzlichen Bestandteile zum existierenden Graphen dargestellt:

Ergänzungen der XML-Darstellung des Datenflußgraphen auf dem Experten-Rechner

```
…

<Node type="Soundgrabber" label="Soundgrabber">
  <Parameter name="Device" value="0"/>
  <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
</Node>
<Node type="Soundplayer" label="Soundplayer">
  <Parameter name="Device" value="0"/>
  <ExternalRoute internal="*" external="{NamespaceLabel}/{SlotLabel}"/>
</Node>
<Node type="Network" label="Network">
  <Parameter name="Prefix" value="MobileSystem/{SlotLabel}"/>
  <ExternalRoute internal="*" external="{SlotLabel}"/>
</Node>

…
```

Wie auf dem Experten-Rechner werden wieder drei Knoten erzeugt, nämlich ein „Soundgrabber“- und ein „Soundplayer“-Knoten für die Audio-Kommunikationsverbindung sowie ein „Network“-Knoten, der die Netzwerktransparenz herstellt. Der Parameter „Prefix“ des „Network“-Knotens sorgt dafür, daß die Namen aller auf den Experten-Rechner exportierten Slots das Prefix „MobileSystem/“ erhalten.

Auffällig ist, daß in diesem Graphen keine Routes vorhanden sind. Das ist nicht notwendig, da alle benötigten Routes bereits auf dem Experten-Rechner angelegt werden. Selbstverständlich hätte man sie anstelle dessen auch hier anlegen können – das Ergebnis wäre dasselbe geblieben.

Der VRML-Code auf dem Experten-Rechner ist in weiten Teilen identisch mit dem Code auf dem mobilen System. Zunächst wird wieder das vom mobilen System empfangene Videobild im Hintergrund angezeigt:

VRML-Code auf dem Experten-Rechner (1. Abschnitt)

```
Shape {
  appearance Appearance {
    texture DEF videoImage PixelTexture {}
  }
  geometry IndexedFaceSet {
    ... # Eine Ebene, auf die das Videobild projeziert wird
  }
}

DEF videoImageSensor SFImageSensor {
  label "Frames"
}

ROUTE videoImageSensor.value_changed TO videoImage.set_image
```

Zunächst wird wieder eine Geometrie erzeugt, auf die das Videobild projeziert wird. Diese Geometrie ist im einfachsten Fall eine Ebene, kann aber auch komplexere Formen haben, um eine eventuelle Verzerrung des Videobildes auszugleichen.

Darauf folgt ein „SFImageSensor“-Knoten, der die Videobilder vom Datenflußgraphen empfängt und in den VRML-Szenengraphen einspeist.

Eine Route sorgt dafür, daß die Videobilder an die Textur weitergeleitet werden, die auf die Geometrie gemappt wird.

Der zweite Abschnitt des VRML-Codes enthält die Bestandteile des Szenengraphen, die die Augmentierungen enthalten. Auch dieser Abschnitt ist identisch mit dem Code auf dem mobilen System:

VRML-Code auf dem Experten-Rechner (2. Abschnitt)

```
DEF scene Transform {
  children [
    ... # 3D-Objekte der AR-Szene
  ]
}

DEF translationSensor SFVec3fSensor {
  label "Translation"
}

ROUTE translationSensor.value_changed TO scene.set_translation

DEF rotationSensor SFRotationSensor {
  label "Rotation"
}

ROUTE rotationSensor.value_changed TO scene.set_rotation

DEF scaleSensor SFVec3fSensor {
  label "Scale"
}

ROUTE scaleSensor.value_changed TO scene.set_scale
```

Zunächst wird ein „Transform“-Knoten erzeugt, der als Kind-Knoten die 3D-Objekte der AR-Szene enthält. Daraufhin folgen drei Sensor-Knoten, die die Translation, Rotation und Skalierung der 3D-Objekte vom Datenflußgraphen erhalten. Routes leiten diese Werte an den Transform-Knoten weiter.

Der einzige Abschnitt, der sich vom Szenengraphen auf dem mobilen Gerät unterscheidet, ist der dritte und letzte Abschnitt. Er enthält den VRML-Code, der die Position des Mauszeigers des Experten bestimmt und an das mobile Gerät verschickt:

VRML-Code auf dem Experten-Rechner (3. Abschnitt)

```
Group {
  children [
    Shape {
      appearance Appearance {
        material Material { transparency 1 } # Komplett transparent
      }
      geometry IndexedFaceSet {
        … # Code zum Erzeugen einer Ebene
      }
    }

    DEF touchSensor TouchSensor {}
  ]
}

DEF pointerSensor SFVec3fSensor {
  label "Pointer"
  out TRUE
}

ROUTE touchSensor.hitPoint_changed TO pointerSensor.set_value
```

Zunächst einmal wird eine komplett transparente (also unsichtbare) Ebene erzeugt, die sich über den ganzen Bildschirm erstreckt und sich vor allen anderen geometrischen Objekten befindet. Diese Ebene dient nur dazu, zusammen mit dem im gleichen „Group“-Knoten befindlichen „TouchSensor“-Knoten die Position des Mauszeigers zu bestimmen.

„TouchSensor“-Knoten liefern Informationen darüber, ob der Anwender eine Geometrie des Szenengraphen mit dem Mauszeiger berührt. Dabei werden nur Geometrien berücksichtigt, die sich im Szenengraphen auf der gleichen Ebene wie der „TouchSensor“-Knoten befinden, also „Geschwister“ bzw. deren „Nachkommen“ sind. U.a. liefert der „TouchSensor“-Knoten den Schnittpunkt eines Strahls, der von der aktuellen Mausposition aus in die Szene geschossen wird, mit der Geometrie zurück. In unserem Fall ist das der Schnittpunkt mit der transparenten Ebene.

Außerdem befindet sich noch ein „SFVec3fSensor“-Knoten in der Szene. Im Gegensatz zu den bisher verwendeten Sensor-Knoten dient dieser Knoten nicht dazu, Datenwerte vom Datenflußgraphen zu empfangen, sondern Datenwerte in den Datenflußgraphen zu senden. In diesem Fall wird die aktuelle Position des Mauszeigers verschickt. Dazu verbindet eine Route den „hitPoint_changed“-Outslot des „TouchSensor“-Knotens mit dem „set_Value“-Inslot des „SFVec3fSensor“-Knotens.

Der VRML-Code auf dem mobilen System muß nur um die Bestandteile erweitern werden, die der Darstellung des Zeigers dienen, mit dem der Remote-Expert auf einzelne Bestandteile des Bildes zeigen kann:

Ergänzungen am VRML-Code des mobilen Systems

```
DEF pointerTransform Transform {
  children [
    Shape { … } # Die Zeiger-Geometrie
  ]
}

DEF pointerSensor SFVec3fSensor {
  label "Pointer"
}

ROUTE pointerSensor.value_changed TO pointerTransform.set_translation
```

Zunächst wird wieder ein „Transform“-Knoten erzeugt, der als Kind die Geometrie des Zeigers enthält. Danach wird ein „SFVec3fSensor“-Knoten erzeugt, der die Position des Zeigers vom Datenflußgraphen erhält. Eine Route verbindet den „value_changed“-Outslot des Sensors mit dem „set_translation“-Inslot des Transformationsknotens.

Es sei nochmals darauf hingewiesen, daß diese Darstellung vereinfacht worden ist. Selbstverständlich ist das MARIO-System deutlich komplexer. So muß ja z.B. auch irgendwie sichergestellt werden, das mobiles System und Experten-Rechner die gleichen Augmentierungen darstellen, also die gleichen 3D-Objekte vor dem Videobild darstellen. Dazu gibt es noch weitere Bestandteile des Szenengraphen und des Datenflußgraphen des Gerätemanagements. Da es hier aber nur darum ging, die grundlegenden Konzepte darzustellen, wurden diese eher uninteressanten Bestandteile hier weggelassen.

Auch hier kann man wie bei den Erweiterungen des Archeoguide-Systems feststellen, daß relativ wenige Änderungen an der AR-Anwendung selbst notwendig sind. Die meisten Änderungen finden innerhalb des Datenflußgraphen statt. Die Anwendung wurde beim Experten-Modus nur insoweit geändert, als daß in den Szenengraphen eine Zeigergeometrie sowie der Code zum Empfang der Zeigerposition eingebunden wurden.

Interessanterweise funktioniert die Anwendung mit diesen Änderungen auch dann problemlos, wenn kein Experten-Rechner mit dem mobilen System verbunden ist. In diesem Fall werden einfach nur das Videobild und die Tracking-Ergebnisse nicht vom mobilen System abgerufen, und das mobile System erhält keine Mauszeigerposition. Es bleibt aber voll funktionsfähig – dieselbe Anwendung kann also sowohl im Standalone- als auch im Experten-Modus betrieben werden.

7.3 Zusammenfassung

Die hier vorgestellten Anwendungsbeispiele („Archeoguide“ und „MARIO“) demonstrieren, wie man mit dem in dieser Arbeit vorgestellten AR-Rahmensystem in der Praxis AR-Anwendungen entwickeln kann. Dabei handelt es sich nicht um speziell konstruierte Szenarien, sondern um typische AR-Systeme. Selbstverständlich konnten beide Systeme hier nur in stark vereinfachter Form dargestellt werden, insbesondere die Controller-Komponenten, die große Teile der eigentlichen Anwendungslogik enthalten, sind hier aus Platzgründen nicht besprochen worden. Dennoch kann man folgende Punkte feststellen:

- Das System erlaubt es, AR-Anwendungen zu entwickeln, die unabhängig von der verwendeten Betriebsystem- und Hardwareplattform sind. Die Anwendung besteht aus drei Komponenten, dem Datenflußgraphen des Gerätemanagements, dem VRML-Szenengraphen, sowie JavaScript- oder Java-Code. Diese Komponenten werden von einem Laufzeitsystem ausgeführt bzw. dargestellt. Die Anwendung läuft also auf jeder Plattform, auf der das Laufzeitsystem verfügbar ist.

- Die Abhängigkeiten von der verwendeten Hardwareplattform sind komplett im Datenflußgraphen des Gerätemanagements gekapselt. Ändert man die Hardwareausrüstung des Systems, so sind Anpassungen nur an diesem Datenflußgraphen notwendig.
- Das AR-Rahmensystem stellt eine große Auswahl an Funktionalität zu Verfügung. Die Erweiterung des MARIO-Systems um einen Remote-Expert-Modus läßt sich komplett mit den vom Rahmensystem verfügbaren Mitteln umsetzen, und das in kürzester Zeit (Stichwort „Rapid Prototyping“). Eine handvoll Änderungen am Datenflußgraphen und am VRML-Szenengraphen, durchgeführt in einer Stunde, reichte aus. Man überlege, wieviel Zeit es dagegen gekostet hätte, die gleiche Funktionalität in einer Programmiersprache wie C++ zu implementieren (und dann insbesondere auch noch zu debuggen).

Zusammenfassend kann man feststellen, daß sich das hier vorgestellte AR-Rahmensystem in der Praxis bewährt hat. Es entlastet den Anwendungsentwickler von der Aufgabe, sich um so grundlegende Dinge wie Rendering, Netzwerkkommunikation und Gerätezugriff zu kümmern. Anstelle dessen kann er sich auf das Design der Anwendung selbst konzentrieren.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde ein neuartiges Rahmensystem für Augmented-Reality-Anwendungen vorgestellt. Es handelt sich nicht um eine in C++ oder einer anderen Programmiersprache geschriebene Code-Bibliothek ist, deren Funktionalität der Anwendungsentwickler in seiner jeweiligen AR-Anwendung verwenden kann. Stattdessen handelt es sich um eine Laufzeitumgebung, die AR-Anwendungen ausführt. Diese Anwendungen bestehen aus XML-Dateien, die über eine Markup-Language die Struktur des Szenengraphen, die Gerätekonfiguration sowie die Logik der Anwendung beschreiben. Funktionalität, die nicht über die vorhandenen Elemente der Markup-Language abgedeckt wird, kann über Scriptsprachen wie JavaScript oder Java programmiert werden.

Der Fokus der Forschung im Bereich AR lag in der Vergangenheit insbesondere auf grundlegenden Problemen wie der genauen Bestimmung von Position und Blickrichtung des Anwenders sowie der nahtlosen Integration virtueller Objekte in die reale Umgebung. Wie im 1. Kapitel dargestellt wurde, gab und gibt es eine Vielzahl von Forschungsprojekten, in denen die prinzipielle Machbarkeit und der Nutzen von AR in den verschiedensten Anwendungsgebieten bestätigt wurde. Trotzdem gibt es bis heute außerhalb der Forschung kein AR-System im praktischen Einsatz. Wichtige Gründe dafür sind die hohen Kosten für spezielle AR-Hardware wie HMDs und die aufwendige Softwareentwicklung. Praktisch alle der in Forschungsprojekten entwickelten AR-Prototypen sind Einzelstücke, zugeschnitten auf genau ein spezielles Einsatzszenario, und können nur unter großem Aufwand an andere Szenarien angepaßt werden. Die Ergebnisse eines Projektes können meist nicht ohne weiteres als Grundlage für Nachfolgeprojekte dienen. Aus diesem Grund rückt aktuell die Architektur von AR-Systemen und die Anwendungsentwicklung immer mehr in das Zentrum der Forschungsarbeiten.

Im 2. Kapitel wurden Anforderungen an die Softwarearchitektur eines mobilen AR-Systems aufgestellt. Sie sollte effizient und flexibel sein. Es sollte eine klare Trennung zwischen der eigentlichen Anwendungslogik und der verwendeten Gerätekonfiguration geben, d.h. es sollte ohne größeren Aufwand möglich sein, z.B. die verwendete Trackingtechnologie auszutauschen. Ein wichtiger Punkt ist darüber hinaus die Unabhängigkeit von der verwendeten Systemplattform. Ein AR-System muß nicht mehr unbedingt auf einem Windows-Intel-System basieren – der technische Fortschritt der letzten Jahre hat inzwischen dazu geführt, daß es eine Vielzahl von anderen möglichen Plattformen gibt, darunter Mobiltelefone und mobile Spielekonsolen. Das bedeutet, daß ein modernes AR-System ein sehr heterogenes Feld aus Hardware-Software-Konfigurationen abdecken muß.

Anschließend wurden im 2. Kapitel existierende Lösungen daraufhin untersucht, inwieweit sie die aufgestellten Anforderungen erfüllen. Systeme wie AR-Toolkit, Studierstube, DWARF oder Tinmith stellen dem Anwendungsentwickler leistungsfähige Bausteine zur Verfügung, aus denen er seine AR-Anwendungs zusammensetzen kann.

Ausgehend von der Anforderungsanalyse wurde dann eine eigene Systemarchitektur vorgestellt. Dabei wurde nicht „das Rad neu erfunden“, sondern auf einem existierenden Standard (VRML bzw. sein Nachfolger X3D) aufgebaut. X3D ist ein etablierter ISO-Standard für die Entwicklung von VR-Anwendungen. Ein wichtiges Merkmal von X3D-Anwendungen ist, daß sie plattformunabhängig sind – sie sind kein prozessorspezifischer Binärcode, sondern bestehen aus einer Beschreibung des Szenengraphen und der Anwendungslogik in einer speziellen Markup-Language. Die Besonderheit dabei ist, daß solche Anwendungen anders als Bytecode-Lösungen wie Java oder C# nicht mit Performanzproblemen zu kämpfen haben, sondern ähnlich effizient sind wie in C oder C++ entwickelte Anwendungen. Denn die Markup-Language wird nicht interpretiert, sondern sie beschreibt nur die Verschaltung von vorgefertigten Softwarekomponenten, die in C oder C++ implementiert sind. Nur wenn eine benötigte Funktionalität nicht vom X3D-Framework zur Verfügung gestellt wird, kommen interpretierte Programmiersprachen wie JavaScript oder Java zum Einsatz.

X3D ist bis heute hauptsächlich auf VR-Anwendungen im Internet ausgerichtet, d.h. auf klassische Walk-through-Szenarien in Desktop-Umgebungen mit Tastatur und Maus. Für mobile AR-Szenarien mit exotischer Hardware wie Trackingsystemen und Life-Videoströmen sowie für die Kommunikation mit anderen Komponenten in einem Netzwerk bietet X3D dagegen keine bzw. nur eingeschränkte Unterstützung. Daher mußte die existierende X3D-Renderingkomponente um ein Gerätemanagement-System erweitert werden, und es mußte eine Lösung gefunden werden, wie man Systeme aufbauen kann, die im Netzwerk verteilt sind.

Eine bislang in der Forschung kaum behandelte Frage ist, wie man mobile AR-Anwendungen entwickelt und debuggt. Da mobile Systeme meist keine klassischen Eingabegeräte wie Tastatur und Maus besitzen und AR-Anwendungen üblicherweise im Vollbildmodus ablaufen, kann man nicht ohne weiteres Standard-Entwicklungsumgebungen verwenden, wie sie für die normale Softwareentwicklung eingesetzt werden. Auch dafür mußte eine Lösung gefunden und in die existierende Renderingkomponente integriert werden.

Im 3. Kapitel wurde zunächst das Gerätemanagement behandelt. Es wurden auch hier Anforderungen aufgestellt und bestehende Lösungen wie VRPN oder OpenTracker untersucht. Dabei wurde festgestellt, daß diese Systeme häufig auf VR-Anwendungen ausgerichtet sind und sich nur eingeschränkt für AR-Anwendungen eignen. Problematisch ist dabei insbesondere, daß viele Lösungen per „Polling“ funktionieren, also der Zustand von Geräten permanent abgefragt werden muß. Dies ist für stationäre VR-Anwendungen praktikabel, die auf maximale Bildwiederholrate bei minimaler Latenz optimiert sind und über eine Renderingschleife verfügen, die permanent abgearbeitet wird. Bei mobilen AR-Anwendungen dagegen gibt es eine solche Renderingschleife nicht – hier muß man sparsam mit den begrenzten Ressourcen umgehen und bemüht sich daher, nur dann die Szene zu rendern, wenn man tatsächlich neue Videobilder oder Trackingergebnisse erhält. Weitere Einschränkungen vieler Gerätemanagement-Systeme ist die fehlende Unterstützung für Geräte mit hoher Bandbreite wie Videokameras und Videotrackingsysteme, die für AR-Systeme typisch sind, sowie häufig eine fast vollstände Vernachlässigung von Ausgabegeräten.

Auf Basis der aufgestellten Anforderungen wurde ein neuartiges Gerätemanagement-System entworfen und implementiert. Dieses System zeichnet sich dadurch aus, daß es Event-basiert ist, Videokameras und Videotrackingsysteme unterstützt, Ausgabegeräte gleichberechtigt neben Eingabegeräten in das System integriert, und eine weitgehende Verarbeitung der gelieferten Daten erlaubt. Das grundlegende Konzept ist der Datenflußgraph. Es gibt Knoten, die Daten produzieren oder konsumieren (die Treiber für Ein- oder Ausgabegeräte), und Knoten, die Daten verarbeiten und modifizieren (z.B. Transformation von Positionsdaten, Magnetfeldentzerrung bei elektromagnetischen Trackingsystemen, Videotracking-Algorithmen etc.). Diese Knoten tauschen über die Kanten des Graphen (bestehend aus Outslots, Inslots und Routes) Datenströme miteinander aus. Der Anwendungsentwickler wählt aus diesem Knoten-Pool die benötigten Komponenten aus und verschaltet sie miteinander.

Ein mobiles AR-System ist, stärker noch als stationäre Systeme, auf die Kommunikation mit anderen Systemkomponenten im Netzwerk angewiesen. Dafür wurden im 4. Kapitel Lösungen vorgestellt. Ein wichtiger Punkt ist, daß man dem Anwender keine langwierige Konfiguration seines AR-Systems zumuten kann. Aktuelle Techniken wie DHCP und Service-Discovery-Protokolle wie UPnP, SLP oder MDNS erlauben es, daß sich mobile

Geräte automatisch in vorhandene drahtlose Netzwerke einklinken und die vorhandenen Dienste abrufen können. Da X3D für den Einsatz im Internet ausgerichtet ist, ist der Download von Geometriedaten, Texturen sowie Audio- und Videoströmen von Servern bereits standardmäßig verfügbar. Andere Kommunikationsaufgaben, wie der Kontakt mit Geräten im Netzwerk oder mit anderen Anwendern (Stichwort CSCW), werden dagegen nicht unterstützt. Zum Teil ist das über die Java-Klassenbibliothek möglich, indem man Java-Code über X3D-Script-Knoten in die Szene integriert. Allerdings stellt Java im Gegensatz zu JavaScript für unerfahrene Anwendungsentwickler eine große Hürde dar. Aus diesem Grund wurde in dieser Arbeit vorgeschlagen, das von HTML-Webseiten bekannte XMLHttpRequest-Objekt in X3D zu integrieren. Das XMLHttpRequest-Objekt ist die Grundlage für moderne Web 2.0-Anwendungen (Stichwort Ajax), indem es erlaubt, aus JavaScript heraus Daten mit Servern im Netzwerk auszutauschen. Genau diese Funktionalität kann man analog auch in X3D-Script-Knoten verwenden.

Die Kommunikation mit anderen Komponenten im Netzwerk über Software-Bibliotheken und Sprachfeature der in X3D integrierten Scriptsprachen ist zwar sehr mächtig, aber auch sehr komplex, fehleranfällig und erfordert Know-How des Anwendungsentwicklers. Es gibt allerdings auch noch eine sehr viel einfachere Lösung: Sowohl der X3D-Szenengraph als auch der Datenflußgraph des Gerätemanagements bestehen aus Knoten, die über Kanten Daten miteinander austauschen. Eine naheliegende Lösung besteht darin, Kanten auch zwischen Knoten zu erlauben, die sich auf verschiedenen Rechnern im Netzwerk befinden. Genau dies wurde für das im 3. Kapitel beschriebene Gerätemanagement-System implementiert. Es gibt einen speziellen „Network“-Knoten, der es erlaubt, Verbindungen mit anderen Knoten im Netzwerk herzustellen. Auf diese Weise wird es völlig transparent, wo sich ein Gerät im Netzwerk befindet – man muß nur den Namen des gewünschten Knoten und seiner Slots kennen und über Routes mit seinen eigenen, lokalen Knoten verbinden, das System kümmert sich dann automatisch darum, die konkrete Position der Knoten im Netzwerk zu bestimmen und, falls erforderlich, die Datenströme über das Netzwerk zu transferieren.

Im 5. Kapitel wurde dann untersucht, wie man das Gerätemanagement-System in den bestehenden Rahmen des X3D-Standards integrieren kann. X3D ist momentan stark auf VR-Anwendungen fokussiert, die auf normalen Desktop-Rechnern mit Tastatur und Maus ablaufen. Die Einbindung von Geräten ist daher bislang nur in einer sehr abstrakten, eingeschränkten Form möglich, die es erlaubt, klassische Walk-Through-Szenarien abzudecken: Man kann die Kameraposition steuern, man kann virtuelle Objekte anklicken und bewegen, und man Tastendrücke abfragen. Dieser hohe Abstraktionsgrad über High-Level-Sensoren ist zwar elegant und effizient, aber für VR- und insbesondere AR-Anwendungen, die nicht dem Walk-Through-Szenario folgen, viel zu eingeschränkt. Daher gibt es bereits seit einiger Zeit Bestrebungen, Low-Level-Sensoren in den X3D-Standard zu integrieren, die einen direkten Zugriff auf ein- und ausgehende Datenströme erlauben. Die bisher in verschiedenen X3D-Browsern (Blaxxun/Bitmanagement Contact, Xj3D) implementierten Lösungen wurden im 5. Kapitel untersucht. Sie haben alle den entscheidenden Konstruktionsfehler, daß der Anwendungsentwickler ein konkretes Gerät spezifizieren muß. Dazu ist er aber offensichtlich nicht in der Lage – woher soll der Anwendungsentwickler wissen, welche Hardware auf dem System verfügbar ist, auf dem die Anwendung später laufen wird? Aus diesem Grund wurde im 5. Kapitel ein neuer Ansatz vorgestellt, der darauf basiert, daß für jeden der in X3D verfügbaren Datentypen ein spezieller Sensor-Knoten implementiert wird, der diesen Datentypen empfangen oder versenden kann. Der Anwendungsentwickler baut dann je nachdem, welche Datenströme er für seine Anwendung benötigt, entsprechende Sensor-Knoten in seine X3D-Szene ein. In den Sensor-Knoten spezifiziert er über ein Label-Feld, wofür diese Daten benötigt werden (z.B.

Vorwärts- oder Rückwärtsgehen, Links- oder Rechtsdrehen). Dies ist der entscheidende Unterschied zu existierenden Lösungen, wo der Anwendungsentwickler spezifizieren muß, woher (von welchem Gerät) die Daten kommen – wozu er nicht in der Lage ist – und nicht, wozu sie verwendet werden sollen. Das eigentliche Mapping der vorhandenen Geräte auf die in der Szene benötigten Datenströme muß der Anwender über das X3D-Browser-Userinterface vornehmen – nur er ist dazu in der Lage, weil er die konkret verfügbare Hardware kennt. Das mag zwar aufwendig erscheinen, ist jedoch genau der gleiche Weg, über den z.B. in Computerspielen der Spieler festlegt, auf welche Weise (mit welchen Geräten) er seine Spielfigur o.ä. steuern möchte. Außerdem muß das Mapping nur einmal beim ersten Start der Anwendung festgelegt werden – der X3D-Browser kann es sich schließlich merken und bei einem erneuten Start der Anwendung wiederverwenden.

Ein andere Frage, die im 5. Kapitel behandelt wurde, ist, wie man innerhalb einer X3D-Szene Videodatenströme effizient behandeln kann. Videodaten besitzen eine vergleichsweise große Bandbreite, daher muß ihre Verarbeitung besonders sorgfältig geplant werden. Wie bereits im 3. Kapitel angesprochen wurde, kapitulieren viele Gerätemanagement-Systeme vor dieser Aufgabe und bieten keine Unterstützung für Videokameras. Anstelle dessen überläßt man es den Videotrackingsystemen, Videobilder von der Kamera zu grabben, zu verarbeiten und (im Fall von Video-See-Through-Systemen) auf dem Bildschirm zu rendern. Die eigentliche 3D-Szene wird dann in einem folgenden Schritt über das Videobild gerendert. Der Nachteil dieser Lösung ist, daß damit das Videotracking in das Zentrum der Systemarchitektur rückt, wo es nicht hingehört – in das Zentrum gehört die eigentliche Anwendung und nicht eine periphere Softwarekomponente, die die Position und Blickrichtung des Anwenders bestimmt. Das hier vorgestellte Gerätemanagement-System unterstützt dagegen ohne Einschränkungen Video-Datenströme. Das Videotracking ist nur ein weiterer Knoten im Datenflußgraph der Gerätemanagement-Systems, der die Videodaten empfängt und weiterverarbeitet. Die Videodaten werden (wie alle anderen Daten auch) über SFImage-Sensor-Knoten in der X3D-Szene empfangen und über Routes an einen PixelTexture-Knoten weitergeleitet, mit dem die Videobilder ganz normal als Textur auf Objekte in der 3D-Szene gemappt werden können. Voraussetzung dafür ist der konsequente Einsatz von Smart-Pointern, damit nicht die gesamten Videodaten, sondern nur Zeiger auf diese Daten bei der Übertragung zwischen verschiedenen Knoten kopiert werden müssen. Ein ausgeklügeltes System sorgt dafür, daß nicht für jedes Videobild erneut Speicher vom Betriebssystem angefordert werden muß, sondern es werden bereits benutzte Speicherblöcke recycled und damit eine Fragmentierung des Speichers verhindert. Der große Vorteil dieses Ansatzes ist, daß die Videobilder normale Texturen sind und damit viele aufwendige Bildbarbeitungsschritte effizient von der Graphik-Hardware erledigt werden können. Als Beispiel dafür wurde im 5. Kapitel die Korrektur der typischen Kissen- oder Tonnenverzerrung von Videobildern genannt.

Ein in der aktuellen Forschung kaum behandeltes Thema ist die Frage, wie man AR-Anwendungen entwickeln und insbesondere debuggen kann. Auf der einen Seite hat man exotische und leider häufig nicht sehr zuverlässige Hard- und Softwarekomponenten wie GPS, Kompasse und Videotrackingsysteme. Auf der anderen Seite ist es praktisch unmöglich, herkömmliche Entwicklungswerkzeuge einzusetzen, da AR-Anwendungen meist im Vollbildmodus laufen und klassische Eingabegeräte wie Tastatur und Maus fehlen. Im 6. Kapitel wurde daher vorgeschlagen, einen kleinen Webserver in das System zu integrieren. Dieser Webserver erlaubt es, von jedem Gerät im Netzwerk mit einem normalen Web-Browser eine Verbindung zum laufenden AR-System aufzubauen, den Zustand der diversen Hard- und Softwarekomponenten abzufragen, und das System sogar im Betrieb zu modifizieren und einzelne Teilkomponenten auszutauschen oder neu zu starten. Die modifizierten Geräte-Datenflußgraphen und der Szenengraph können über das Web-Interface abgespeichert und später wieder geladen werden. Natürlich hat das Web-Interface seine

Einschränkungen – da ein Webserver nie von sich aus Kontakt zu einem Web-Browser aufnehmen kann, ist es nicht möglich, z.B. Breakpoints auf Variablen zu setzen o.ä. Man kann nur einen aktuellen „Schnappschuß" des Systemzustands abrufen. Dieser Nachteil wird aber dadurch aufgewogen, daß man das Web-Interface von jedem Gerät im Netzwerk aus nutzen kann, auf dem ein Web-Browser vorhanden ist – es muß keine weitere Software installiert werden.

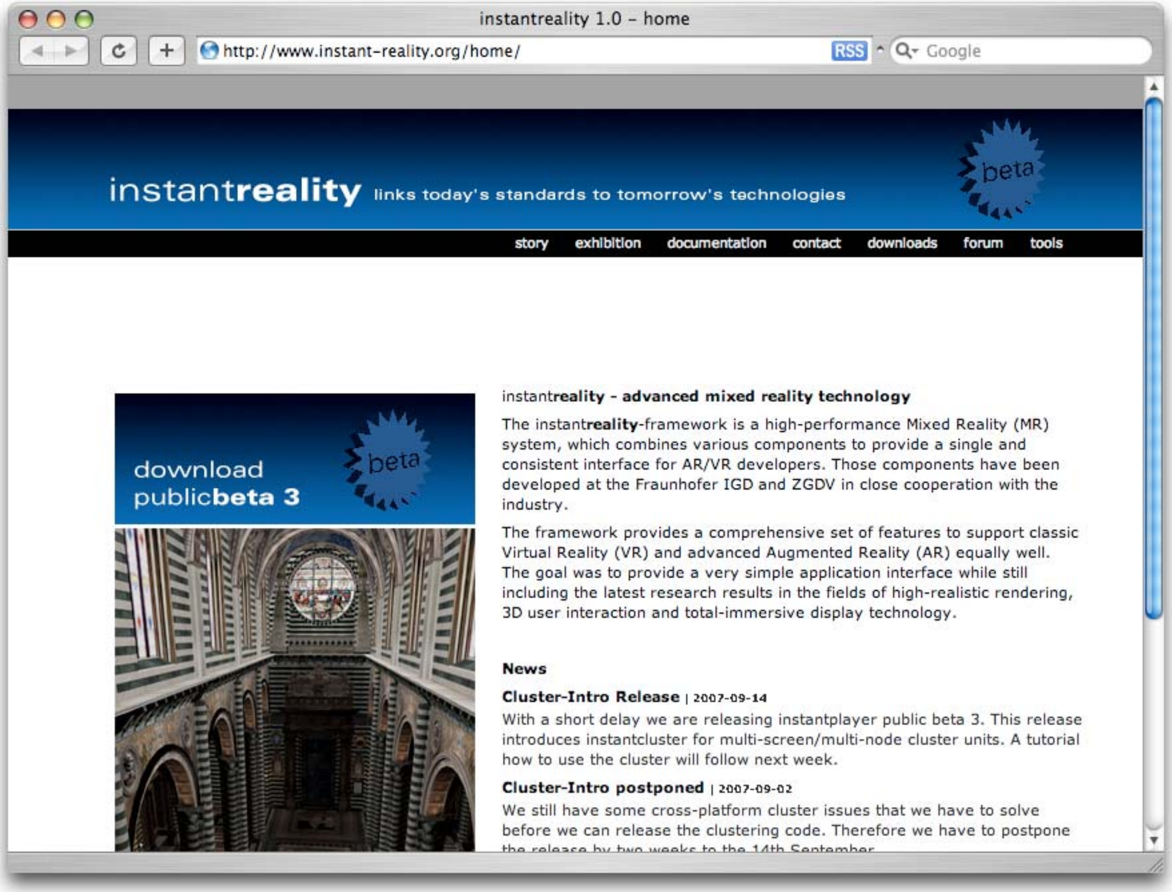
In der Praxis zeigte sich schnell, daß das Web-Interface nicht nur eines der wichtigsten Werkzeuge bei der Entwicklung von VR-/AR-Anwendungen wurde, sondern auch zu anderen, ursprünglich nicht vorgesehenen Zwecken genutzt werden konnte. Denn die Möglichkeit, den Szenengraphen über spezielle URLs zu modifizieren, bildet eine Schnittstelle, über die andere Softwarekomponenten mit der VR-/AR-Anwendung kommunizieren können, solange sie den Download von einer URL beherrschen. Natürlich ist das nur eine One-Way-Kommunikation – externe Softwarekomponenten können Nachrichten an das VR-/AR-System schicken, aber nicht umgekehrt. Für viele Anwendungsfälle, wo einfach nur Aktionen in der virtuellen Szene getriggert werden müssen, ist das jedoch vollkommen ausreichend.

Im 7. Kapitel wurde schließlich am Beispiel zweier AR-Projekte gezeigt, wie das in dieser Arbeit vorgestellte System in der Praxis eingesetzt wird. Es handelt sich um das Outdoor-AR-Projekt Archeoguide und das Indoor-AR-Projekt MARIO. Beide demonstrieren beispielhaft, wie man mit dem AR-Framework voll funktionsfähige AR-Anwendungen erstellen kann. Es handelt sich bei beiden Projekten nicht um Nischensysteme, die sich aufgrund ihrer speziellen Ausrichtung besonders für das AR-Rahmensystem eignen, sondern um typische Vertreter von AR-Projekten. Die eigentliche AR-Anwendung ist kein monolithisches Objekt, sondern besteht aus mehreren XML-Dateien, die den Szenengraphen, die Anwendungslogik und die Gerätekonfiguration enthalten. Das Ergebnis ist eine Anwendung, die weitestgehend unabhängig von der verwendeten Hard- und Software-Plattform ist und leicht an veränderte Szenarien angepaßt werden kann.

Das System wurde nicht nur in diesen beiden Projekten verwendet, sondern darüber hinaus in einer Vielzahl von anderen AR-Projekten. Es wird auch in den nächsten Jahren eine wichtige Grundlage für AR-Projekte am Fraunhofer IGD darstellen. Momentan gibt es Bestrebungen, das System einem größeren Kreis von AR-Entwicklern zugänglich zu machen. Unter dem Namen „Instant Reality Framework" kann man eine Demoversion des Systems seit April 2007 im Internet herunterladen[16]. Die momentan verfügbare Demoversion enthält fast alle Bestandteile des hier vorgestellten Systems, es gibt allerdings bislang noch kaum Dokumentation und nur ein sehr eingeschränktes graphisches Userinterface, sodaß eine praktische Arbeit mit dem System nur mit Unterstützung des Entwicklerteams möglich ist. Gerade laufende Arbeiten zielen darauf ab, das System zu vervollständigen, Fehler zu beseitigen, graphische Userinterfaces zu entwickeln und Dokumentation sowie Tutorials zu schreiben. Das Endziel ist es, ein mächtiges und flexibles Werkzeug zur Entwicklung von VR- und AR-Anwendungen bereitzustellen, das nicht nur von den Entwicklern selber als Grundlage für ihre eigenen Projekte genutzt werden kann, sondern es jeder interessierten Person ermöglicht, ohne große Vorkenntnisse und ohne großen Entwicklungsaufwand VR- und AR-Anwendungen zu schreiben.

---

[16] http://www.instantreality.org/



instantreality - advanced mixed reality technology

The instantreality-framework is a high-performance Mixed Reality (MR) system, which combines various components to provide a single and consistent interface for AR/VR developers. Those components have been developed at the Fraunhofer IGD and ZGDV in close cooperation with the industry.

The framework provides a comprehensive set of features to support classic Virtual Reality (VR) and advanced Augmented Reality (AR) equally well. The goal was to provide a very simple application interface while still including the latest research results in the fields of high-realistic rendering, 3D user interaction and total-immersive display technology.

News

Cluster-Intro Release | 2007-09-14

With a short delay we are releasing instantplayer public beta 3. This release introduces instantcluster for multi-screen/multi-node cluster units. A tutorial how to use the cluster will follow next week.

Cluster-Intro postponed | 2007-09-02

We still have some cross-platform cluster issues that we have to solve before we can release the clustering code. Therefore we have to postpone the release by two weeks to the 14th September

Abbildung 62: Webseite des Instant Reality Frameworks im Internet

Neben diesen Bestrebungen, das System einem breiteren Nutzerkreis verfügbar zu machen, gibt es auch Bemühungen, die in dieser Arbeit vorgeschlagenen Erweiterungen des X3D-Standards in den Standardisierungsprozeß einzubringen. Seit Mitte 2007 ist das Fraunhofer IGD Mitglied im Web3D Konsortium[17], in dessen Hand die Weiterentwicklung des X3D-Standards liegt, und hat damit Stimme und Einfluß in den entsprechenden Gremien. Ein wichtiges Ziel der laufenden Standardisierungsbemühungen ist es, X3D aus der Nische der Webbrowser-Plugins herauszuholen und für fortgeschrittene immersive VR- und AR-Anwendungen zu öffnen. Eine wichtige Voraussetzung dafür sind u.a. bessere Schnittstellen zur Anbindung externer Geräte und anderer Softwarekomponenten – also genau die Themen, die in dieser Arbeit behandelt werden. Insbesondere die im 5. Kapitel vorgestellten Low-Level-Sensorknoten haben gute Chancen, in den Standard integriert zu werden. Dabei ist die Veröffentlichung des Instant Reality Frameworks eine wichtige Voraussetzung – es stellt eine Referenzimplementierung dar, die die prinzipielle Funktionsweise dieser Knoten demonstriert. Interessierte Personen können sich das Framework herunterladen und in der Praxis nachvollziehen, wie man mit diesen Knoten Geräte einbindet oder simple CSCW-Anwendungen schreibt, bei denen mehrere laufende Instanzen der Anwendung über das Netzwerk miteinander kommunizieren.

Das in dieser Arbeit vorgestellte Rahmensystem erleichtert die Entwicklung von AR-Anwendungen erheblich. Der Anwendungsentwickler kann auf einen Pool von vorgefertigten Komponenten zurückgreifen. Werden in Projekten fehlende oder neuartige Komponenten zum Rahmensystem hinzugefügt, stehen sie sofort auch allen anderen Anwendern der

---

[17] http://www.web3d.org/

Laufzeitumgebung für deren Projekte zur Verfügung – Projektergebnisse gehen also nicht mehr verloren. AR-Projekte beginnen nicht mehr bei Null, sondern bauen auf der bisher erreichten Entwicklungsstufe auf. Das bedeutet, daß zukünftige Projekte nicht mehr Zeit mit der banalen Implementierung von Grundfunktionalitäten verschwenden, wie z.B. der Anbindung von Kameras oder GPS-Empfängern. Anstelle dessen kann das Augenmerk stärker auf die eigentlich wichtigen Forschungsthemen gerichtet werden, wie z.B. robustes markerloses Tracking, realistisches Rendering, Verdeckungsproblematik und die Interaktion zwischen Anwender und AR-Anwendung (siehe auch 1. Kapitel).

Ein weiterer Vorteil des Systems besteht darin, daß es keine Programmierkenntnisse in C++ oder ähnlichen klassischen Programmiersprachen voraussetzt. Die Anwendung besteht aus simplen XML-Dateien und in Java oder JavaScript geschriebenen Codeschnipseln, wie man sie z.B. auch von Web2.0-Anwendungen kennt. Das bedeutet, daß es nur eine sehr niedrige Schwelle gibt, um eigene Anwendungen zu erzeugen. Jeder an der Technologie Interessierte kann sich eine einfache Beispielanwendung nehmen und mit einem Texteditor oder über das Webinterface an seine Zwecke anpassen. Mit wenigen Zeilen Code kann er Bilder von einer Kamera abrufen, mit modernsten markerlosen Trackingverfahren seine Position und Blickrichtung bestimmen und das Kamerabild mitsamt den virtuellen Objekten rendern. Die Anwendungsentwicklung ist also nicht mehr nur auf speziell ausgebildete Informatiker begrenzt, sondern steht auch Personen mit einem anderen Hintergrund offen, wie z.B. Designern/Gestaltern, die völlig neue Ansätze für die Schnittstelle Anwender/AR-System liefern können – ein wichtiger Aspekt für die Weiterentwicklung von AR-Anwendungen in Richtung markfähiger Produkte. So ist es kein Zufall, daß das System seit einiger Zeit an der Hochschule für Gestaltung in Offenbach und der Fakultät für Gestaltung der FH Würzburg für die Entwicklung von Mixed-Reality-Anwendungen genutzt wird – es stellt momentan für diesen Personenkreis den einfachsten Einstieg in die Zukunftstechnologie AR dar.

Der größte Vorteil des Systems, die Anwendungsentwicklung mit einem simplen Text-Editor, stellt aber zugleich auch seinen größten Nachteil dar. Das direkte Bearbeiten von Textdateien ist selbstverständlich keine optimale Lösung, wünschenswert wären spezielle Anwendungseditoren. Die Tatsache, daß die mit diesem System erzeugten Anwendungen hauptsächlich aus Graphen bestehen, deren Knoten über Kanten miteinander kommunizieren, macht einen graphischen Editor naheliegend, mit dem man Knoten erzeugen und miteinander verbinden kann. Entsprechende Entwicklungsarbeiten an einer Software mit dem Namen „Instant Composer“ laufen momentan. Allerdings ist auch der Abstraktionslevel eines solchen graphischen Editors immer noch relativ gering. Wie die Forschungsarbeiten im EU-Projekt ULTRA [8] und an dem ebenfalls am Fraunhofer IGD entwickelten ARVIKA-AR-Browser [98] gezeigt haben, kann man z.B. für AR-Anwendungen im Bereich der Wartung von Maschinen eine relativ kleine Menge von typischen, immer wiederkehrenden Arbeitsschritten identifizieren, die in solchen Anwendungen visualisiert werden müssen. Die Idee ist daher, spezielle Authoringwerkzeuge für die verschiedenen AR-Anwendungsszenarien zu entwickeln, die Schablonen bereitstellen, mit denen man schnell und ohne größeres Vorwissen Anwendungen zusammenstellen kann. Dieses sogenannte „Template Based Authoring“ [99] steckt aber noch in den Kinderschuhen.

Das in dieser Arbeit vorgestellte Rahmensystem für mobile AR-Anwendungen stellt einen wichtigen Teilschritt auf dem Weg hin zu praktisch verwendbaren AR-Anwendungen dar. Eine Menge von Teilproblemen bleibt allerdings noch zu lösen. Nichtsdestotrotz bildet das System eine stabile Grundlage für die weitere Entwicklungs- und Forschungsarbeit.

9 Literaturverzeichnis

[1] Behr, J.; Eschler, P.; Fröhlich, T.; Knöpfle, C.; Lutz, B.; Müller, S.; Roth, M.: „Cybernarium Days 2002 – Proving Virtual and Augmented Reality In A Public Exhibition.“ Proceedings of Cyberworlds 2002, Tokyo, Japan, 2002.

[2] Bernd Schwald, Helmut Seibert: „Registration Tasks for a Hybrid Tracking System for Medical Augmented Reality.“ WSCG 2004, Pilsen, Tschechien, Februar 2004.

[3] Goebbels et al.: „Development of an Augmented Reality System for intra-operative navigation in maxillo-facial surgery.“ Internationale Statustagung „Virtuelle und Erweiterte Realität“, Leipzig, Deutschland, Februar 2004.

[4] Vlahakis, Vassilios; Ioannidis, Nikos; Karigiannis, John; Tsotros, Manolis; Gounaris, Michael; Stricker, Didier; Gleue, Tim; Dähne, Patrick; Almeida, Luis: „Archeoguide: An Augmented Reality Guide for Archaeological Sites.“ IEEE Computer Graphics and Applications 22 (2002), 5, pp. 52-60.

[5] G. Papagiannakis, S. Schertenleib, M. Ponder, M.Arévalo, N. Magnenat-Thalmann, D. Thalmann: „Real-Time Virtual Humans in AR Sites.“ IEEE Visual Media Production (CVMP), pp. 273-276, 15-16 March 2004, London, UK, March 2004.

[6] Weidenhausen, Jens; Knöpfle, Christian; Stricker, Didier: „Lessons learned on the way to industrial augmented reality applications, a retrospective on ARVIKA.“ Computers & Graphics (2003), 27, pp. 887-891.

[7] Bernd Schwald, Blandine de Laval: „An Augmented Reality System for Training and Assistance to Maintenance in the Industrial Context.“ WSCG 2003, Pilsen, Tschechien, Februar 2003.

[8] A. Makri, D. Arsenijevic, J. Weidenhausen, P. Eschler, D. Stricker, O. Machui, C. Fernandes, S. Maria, G. Voss, N. Ioannidis: „ULTRA: An Augmented Reality system for handheld platforms, targeting industrial maintenance applications.“ 11th International Conference on Virtual Systems and Multimedia, Ghent, Belgium.

[9] Seibert, Helmut; Dähne, Patrick: „System Architecture of a Mixed Reality Framework.“ Journal of Virtual Reality and Broadcasting, 3(2006), no. 7, May 2007, urn:nbn:de:0009-6-7774, ISSN 1860-2037.

[10] Mark Livingston, Lawrence Rosenblum, Simon Julier, Dennis Brown, Yohan Baillot, J. Edward Swan II, Joseph L. Gabbard, Deborah Hix: „An Augmented Reality System for Military Operations in Urban Terrain.“ Proceedings of the Interservice / Industry Training, Simulation, & Education Conference (I/ITSEC '02), Orlando, FL, December 2-5, 2002.

[11] Ursula Kretschmer, Volker Coors, Ulrike Spierling, Dieter Grasbon, Kerstin Schneider, Isabel Rojas, Rainer Malaka: „Meeting the Spirit of History.“ VAST 2001, Glyfada, 2001, 161-172.

[12] Bruce Thomas, Ben Close, John Donoghue, John Squires, Phillip De Bondi, Michael Morris and Wayne Piekarski: „ARQuake: An Outdoor/Indoor Augmented Reality First Person Application.“ 4. International Symposium on Wearable Computers, pp 139-146, Atlanta, Georgia, Oktober 2000.

[13] Chandaria, J.; Thomas, G.; Bartczak, B.; Koeser, K.; Koch, R.; Becker, Mario; Bleser, Gabriele; Stricker, Didier; Wohlleber, Cedric; Felsberg, M.; Gustafsson, F.; Hol, J.; Schön, T.B.; Skoglund, J.; Slycke, P.J.; Smeitz, S.: „Realtime Camera Tracking in the MATRIS Project.“ In: SMPTE Motion Imaging Journal 116 (2007), 7/8, pp. 266-271.

[14] Jung, Yvonne; Franke, Tobias; Dähne, Patrick; Behr, Johannes: „Enhancing X3D for advanced MR appliances.“ In: ACM SIGGRAPH u.a.: Proceedings WEB3D 2007: 12th International Conference on 3D Web Technology. New York: ACM Press, 2007, pp. 27-36.

[15] Gleue, Tim; Dähne, Patrick: „Design and Implementation of a Mobile Device for Outdoor Augmented Reality in the ARCHEOGUIDE Project.“ VAST 2001: Proceedings of the International Symposium on Virtual Reality, Archaeology and Cultural Heritage: Delegates’ Edition. Glyfada, 2001, S. 181-187.

[16] Peter Ebbesmeyer, Markus Knobel, Holger Krumm, Jürgen Fründ, Carsten Matysczok: „AR-PDA: Innovative Product Marketing for Innovative Products.“ Internationale Statustagung „Virtuelle und Erweiterte Realität“, Leipzig, Deutschland, Februar 2004.

[17] Wouter Pasman, Charles Woodward: „Implementation of an Augmented Reality System on a PDA.“ ISMAR 2003.

[18] Stuart Goose, Sinem Güven, Xiang Zhang, Sandra Sudarsky and Nassir Navab: „PARIS: Fusing Vision-based Location Tracking with Standards-based 3D Visualization and Speech Interaction on a PDA.“ Proc. IEEE DMS 2004 (International Conference on Distributed Multimedia Systems), p 75-80, San Francisco, CA, September 8-10, 2004.

[19] Daniel Wagner, Dieter Schmalstieg: „First Steps Towards Handheld Augmented Reality.“ Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003).

[20] Mathias Möhring, Christian Lessig, and Oliver Bimber: „Video See-Through AR on Consumer Cell Phones.“ In proceedings of International Symposium on Augmented and Mixed Reality (ISMAR'04), pp. 252-253, 2004.

[21] Dieter Schmalstieg, Daniel Wagner: „Experiences with Handheld Augmented Reality.“ Proc. 6h IEEE International Symposium on Mixed and Augmented Reality (ISMAR'07), pp. 3-18, Nara, Japan, Oct. 2007.

[22] Gerhard Reitmayr, Tom Drummond: „Going Out: Robust Model-based Tracking for Outdoor Augmented Reality.” Proc. IEEE ISMAR'06, 2006, Santa Barbara, California.

[23] Behr, Johannes; Dähne, Patrick; Roth, Marcus: „Utilizing X3D for Immersive Environments.“ In: Spencer, Stephen N. (Ed.); ACM SIGGRAPH: Web3D 2004. Proceedings: Nineth International Conference on 3D Web Technology. New York: ACM, 2004, pp. 71-78, 182.

[24] Kato, H., Billinghurst, M.: „Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System.“ Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99). Oktober 1999, San Francisco, USA.

[25] Daniel Wagner, Dieter Schmalstieg: „ARToolKitPlus for Pose Tracking on Mobile Devices.“ Proceedings of 12th Computer Vision Winter Workshop (CVWW'07). St. Lambrecht, Austria, February 6–8 2007.

[26] Daniel Wagner, Dieter Schmalstieg: „First Steps Towards Handheld Augmented Reality.“ ISWC 03.

[27] Blair MacIntyre, Steven Feiner: „Language-Level Support for Exploratory Programming of Distributed Virtual Environments.“ Proceedings of ACM UIST ‘96 (Symp. on User Interface Software and Technology), Seattle, WA, November 6–8, 1996, pp. 83–94.

[28] D. Schmalstieg, A. Fuhrmann, G. Hesina, Zs. Szalavari, L. Miguel Encarnação, M. Gervautz, and W. Purgathofer: „The Studierstube Augmented Reality Project.“ Presence, 11(No.1), 2002.

[29] Gerd Hesina, Dieter Schmalstieg, Anton Fuhrmann, and Werner Purgathofer: „Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics.“ In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'99), pp. 74-81, 1999.

[30] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reicher, S. Riß, C. Sandor, M. Wagner: „Design of a Component-Based Augmented Reality Framework.“ Proceedings of The Second IEEE and ACM International Symposium on Augmented Reality (ISAR 2001).

[31] F. Michahelles: „Designing an Architecture for Context-Aware Service Selection and Execution.“ Master’s thesis, Ludwig–Maximilians–Universität München, 2001.

[32] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams, and J. Shuster: „UIML: An Appliance Independent XML User Interface Language.“ 1999.

[33] Johannes Behr: „Avalon: Ein skalierbares Rahmensystem für dynamische Mixed-Reality Anwendungen.“ Darmstadt, Technische Universität, Dissertation, 2005.

[34] C. Sandor and T. Reicher: „CUIML: A Language for the Generation of Multimodal Human-Computer Interfaces.“ In: Proceedings of the European UIML conference, 2001.

[35] Piekarski, W. and Thomas, B. H: „An Object-Oriented Software Architecture for 3D Mixed Reality Applications.“ In: 2nd Int’l Symposium on Mixed and Augmented Reality, Tokyo, Japan, Oct 2003.

[36] Charles Owen, Arthur Tang, and Fan Xiao: „ImageTclAR: A Blended Script and Compiled Code Development System for Augmented Reality.“ In: Proceedings of the International Workshop on Software Technology for Augmented Reality Systems (STARS), October 2003.

[37] Ousterhout, J.K.: „Tcl and the Tk Toolkit.“ 1994, Reading, MA.

[38] Owen, C.: „The ImageTcl Multimedia Algorithm Development System.“ In: Fifth Annual Tcl/Tk Workshop, 1997, Boston, MA.

[39] Jan Ohlenburg, Iris Herbst, Irma Lindt, Thorsten Fröhlich, Wolfgang Broil: „The MORGAN Framework: Enabling Dynamic Multi-User AR and VR Projects.“ VRST’04, November 10-12, 2004, Hong Kong.

[40] Wolfgang Broll, Irma Lindt, Jan Ohlenburg: „A Framework for Realizing Multi-Modal VR and AR User Interfaces.“ HCI International 2005.

[41] Blair MacIntyre, Maribeth Gandy, Steven Dow, Jay David Bolter: „DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences.“ Conference on User Interface Software and Technology (UIST’04), October 24-27, 2004, Sante Fe, New Mexico.

[42] J. Rohlf, J. Helman: „IRIS Performer: A High Performance Multiprocessing Toolkit for Real–Time 3D Graphics.“ In A. Glassner, editor, Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994), Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[43] R. K. Dybvig: „The Scheme Programming Language: ANSI Scheme.“ P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, second edition, 1996.

[44] E. Frécon, M. Stenius: „DIVE: A Scaleable network architecture for distributed virtual environments.“ Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100.

[45] Emmanuel Frécon: „DIVE on the Internet.“ Dissertation, IT University of Göteborg, ISBN 91-628-6134-4, May 2004.

[46] The Virtual Reality Modeling Language (VRML). International Standard ISO/IEC 14772:1997.

[47] Extensible 3D (X3D). International Standard ISO/IEC 19775:2004.

[48] Standard ECMA-262: ECMAScript Language Specification 3rd edition (December 1999).

[49] Stiles, R., Tewari, S., and Metha, M.: „Adapting VRML 2.0 for Immersive Use.“ VRML 97, Second Symposium on the Virtual Reality Modeling language.

[50] Carolina Cruz-Neira, Allen Bierbaum, Patrick Hartling, Christopher Just, and Kevin Meinert: „VR Juggler – An Open Source Platform for Virtual Reality Applications.“ 40th AIAA Aerospace Sciences Meeting and Exhibit 2002, Reno, Nevada, Januar 2002.

[51] Allen Bierbaum, Christopher Just, Patrick Hartling, Kevin Meinert, Albert Baker, and Carolina Cruz-Neira: „VR Juggler: A Virtual Platform for Virtual Reality Application Development.“ IEEE VR 2001, Yokohama, Japan, März 2001.

[52] Christopher Just, Allen Bierbaum, Patrick Hartling, Kevin Meinert, Carolina Cruz-Neira, and Albert Baker: „VjControl: An Advanced Configuration Management Tool for VR Juggler Applications.“ IEEE VR 2001, Yokohama, Japan, März 2001.

[53] Allen Bierbaum and Carolina Cruz-Neira: „Run-Time Reconfiguration in VR Juggler.“ 4th Immersive Projection Technology Workshop (IPT2000), Ames, Iowa, Juni 2000.

[54] Christopher Just, Allen Bierbaum, Albert Baker, and Carolina Cruz-Neira: „VR Juggler: A Framework for Virtual Reality Development.“ 2nd Immersive Projection Technology Workshop (IPT98), Ames, Iowa, Mai 1998.

[55] Gerhard Reitmayr, Dieter Schmalstieg: „An Open Software Architecture for Virtual Reality Interaction.“ VRST 2001, Banff, Alberta, Canada, November 2001.

[56] J. von Spiczak, E. Samset, S. DiMaio, G. Reitmayr, D. Schmalstieg, C. Burghart, R. Kikinis: „Multi-modal event streams for virtual reality.“ Proc. 14th SPIE Annual Multimedia Computing and Networking Conference (MMCN'07), San Jose, California, Jan. 2007.

[57] Joseph Newman, M. Wagner, M. Bauer, A. MacWilliams, Thomas Pintaric, D. Beyer, D. Pustka, F. Strasser, Dieter Schmalstieg, G. Klinker: „Ubiquitous Tracking for Augmented Reality.“ Proceedings of the 3rd International Symposium on Mixed and Augmented Reality (ISMAR'04), pp. 192-201, Arlington VA, USA, Oktober 2004.

[58] Joseph Newman, Alexander Bornik, Daniel Pustka, Florian Echtler, Manuel Huber, Dieter Schmalstieg, Gudrun Klinker: Tracking for Distributed Mixed Reality Environments. Proceedings of IEEE Virtual Reality Workshop on Trends and Issues in Tracking for Virtual Environments, Charlotte NC, USA, March 2007.

[59] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, Aron T. Helser: „VRPN: A Device-Independent, Network-Transparent VR Peripheral System.“ VRST 2001, Banff, Alberta, Kanada, November 2001.

[60] Lance E. Arsenault, John Kelso: „The DIVERSE Toolkit: A Toolkit for Distributed Simulations and Peripheral Device Services.“ IEEE VR 2002.

[61] John Kelso, Lance E. Arsenault, Steven G. Satterfield, Ronald D. Kriz: „DIVERSE: A Framework for Building Extensible and Reconfigurable Device Independent Virtual Environments.“ IEEE VR 2002.

[62] Torsten Fröhlich, Marcus Roth: „Integration of Multidimensional Interaction Devices in Real-Time Computer Graphics Applications.“ Computer Graphics Forum 19 (2000), 3, S. C-313 - C-319.

[63] Torsten Fröhlich: „Dynamisches Objektverhalten in virtuellen Umgebungen.“ Dissertationsschrift, Technische Universität Darmstadt, Fachgebiet Graphisch-Interaktive Systeme, 2003.

[64] Marcus Roth: „Entwicklung eines Client-Server Systems zur Handhabung multidimensionaler Interaktionsgeräte in 3D Echtzeit-Graphiksystemen.“ Diplomarbeit, Fachhochschule Mannheim, FG Informatik, Deutschland, 1997.

[65] Paul S. Strauss, Rikk Carey, „An object-oriented 3D graphics toolkit.“ SIGGRAPH Computer Graphics, Vol. 26, pp. 341-349, 1992.

[66] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau (editors): „Extensible Markup Language (XML) 1.0 (Fourth Edition).“ World Wide Web Consortium, 2006.

[67] James Clark (editor): „XSL Transformations (XSLT) Version 1.0.“ World Wide Web Consortium, 1999.

[68] Florian Ledermann, István Barakonyi, Dieter Schmalstieg: „Abstraction and Implementation Strategies for Augmented Reality Authoring.” Emerging Technologies of Augmented Reality: Interfaces and Design (ed. M. Billinghurst, M. Haller, B. Thomas), IDEA Group, 2006, pp. 138-159

[69] R. Droms: „Dynamic Host Configuration Protocol.“ RFC 2131, März 1997.

[70] S. Cheshire und B. Aboba: „Dynamic Configuration of IPv4 Link-local Addresses.“ Internet draft, Zeroconf WG, März 2001, work in progress.

[71] S. Thomson und T. Narten: „IPv6 Stateless Address Autoconfiguration.“ RFC 2462, Dez. 1998.

[72] Yaron Y. Goland: „Multicast and Unicast UDP HTTP Messages.“ IETF Draft, 1999.

[73] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, Shivaun Albright: „Simple Service Discovery Protocol.“ IETF Draft, 1999.

[74] J. Cohen, S. Aggarwal: „General Event Notification Architecture.“ IETF Draft, 1998

[75] Stuart Cheshire, Marc Krochmal: „Multicast DNS.“ IETF Draft, 2006.

[76] Stuart Cheshire, Marc Krochmal: „DNS-Based Service Discovery.“ IETF Draft, 2006.

[77] E. Guttman, C. Perkins, J. Veizades, M. Day: „Service Location Protocol, Version 2.“ IETF RFC 2608, 1999.

[78] E. Guttman, C. Perkins, J. Kempf: „Service Templates and Service: Schemes.“ IETF RFC 2609, 1999.

[79] J. Kempf, E. Guttman: „An API for Service Location.“ IETF RFC 2614, 1999.

[80] Anne van Kesteren (editor): „The XMLHttpRequest Object.“ W3C Working Draft, World Wide Web Consortium, 2007.

[81] Reiners, Dirk: „OpenSG: A Scene Graph System for Flexible and Efficient Realtime Rendering for Virtual and Augmented Reality Applications.“ Darmstadt, Technische Universität, Dissertation, 2002.

[82] Henrik Tramberend: „Avocado: A Distributed Virtual Reality Framework.“ In: Virtual Reality, 1999. Proceedings. Houston, TX, USA.

[83] Henrik Tramberend: „Avocado: A Distributed Virtual Environment Framework.“ Dissertation, Universität Bielefeld, Technische Fakultät, 2003.

[84] IEEE Standard 1278.1-1995, Standard for Distributed Interactive Simulation – Application Protocols, 1995.

[85] Nicholas F. Polys, Andrew Ray: „Supporting Mixed Reality Interfaces through X3D Specification.“ Workshop on Mixed-Reality Interface Specification, Proceedings of IEEE Virtual Reality, IEEE Press. 2006.

[86] Frank Althoff, Herbert Stocker, Gregor McGlaun, Manfred K. Lang: „A Generic Approach for Interfacing VRML Browsers to Various Input Devices and Creating Customizable 3D Applications.“ Web3D 2002 Proceedings.

[87] Pablo Figueroa, Omer Medina, Roger Jiménez, José Martíınez, Camilo Albarracín: „Extensions for Interactivity and Retargeting in X3D.“ Proceedings of the tenth international conference on 3D Web technology. Bangor, United Kingdom 2005.

[88] Yoshiaki Araki: „VSPLUS: A High-level Multi-user Extension Library For Interactive VRML Worlds.“ Proceedings of the third symposium on Virtual reality modeling language. Monterey, California, United States 1998.

[89] R. Y. Tsai: „An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision.“ Proceedings CVPR’86, 1986, pp. 364-374.

[90] Anton Fuhrmann, Dieter Schmalstieg, Werner Purgathofer: „Practical Calibration Procedures for Augmented Reality.“ Proceedings of the 6th EUROGRAPHICS Workshop on Virtual Environments (EGVE 2000), pp. 3-12, Amsterdam, June 1-2, 2000.

[91] Shreiner, Dave; Woo, Mason; Neider, Jackie; Davis, Tom: „OpenGL Programming Guide. The Official Guide to Learning OpenGL.“ München: Addison-Wesley Verlag, 2007.

[92] MacWilliams, A., Sandor, C., Wagner, M., Bauer, M., Klinker, G., Bruegge, B.: „Herding Sheep: Live System Development for Distributed Augmented Reality.” In Proceedings of the the 2nd IEEE and ACM international Symposium on Mixed and Augmented Reality (October 07 – 10, 2003). ISMAR. IEEE Computer Society, Washington, DC, 123.

[93] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee: „Hypertext Transfer Protocol – HTTP/1.1.“ RFC 2616, 1999.

[94] Steven Feiner, Blair MacIntyre, Tobias Höllerer, Anthony Webster: „A Touring Machine: Prototyping 3D Mobile Augmented Reality Systems for Exploring the Urban Environment.“ In Proc ISWC ‘97 (Int. Symp. on Wearable Computing), Cambridge, MA, October 13–14, 1997, pages 74–81.

[95] Dave Raggett, Arnaud Le Hors, Ian Jacobs (editors): „HTML 4.01 Specification.“ W3C Recommendation, World Wide Web Consortium, 1999.

[96] Behr, Johannes; Fröhlich, Torsten; Knöpfle, Christian; Kresse, Wolfram; Lutz, Bernd; Reiners, Dirk; Schöffel, Frank: „The Digital Cathedral of Siena - Innovative Concepts for Interactive and Immersive Presentation of Cultural Heritage Sites.“ In: Bearman, David (Ed.) u.a.: International Cultural Heritage Informatics Meeting. Proceedings: Cultural Heritage and Technologies in the Third Millennium. Mailand, 2001, 57-71.

[97] Nilo Mitra, Yves Lafon (editors): „SOAP Version 1.2 Part 0: Primer (Second Edition).“ W3C Recommendation, World Wide Web Consortium, 2007.

[98] Weidenhausen, Jens-Martin: „Mobile Mixed Reality Platform.“ Darmstadt, Technische Universität, Dissertation, 2006.

[99] Knöpfle, Christian; Weidenhausen, Jens; Chauvigné, Laurent; Stock, Ingo: „Template Based Authoring for AR based Service Scenarios.“ In: Fröhlich, Bernd (Ed.) u.a.; Institute of Electrical and Electronics Engineers (IEEE): IEEE Virtual Reality 2005. Proceedings.

10 Eigene Veröffentlichungen

Jung, Yvonne; Franke, Tobias; Dähne, Patrick; Behr, Johannes: „Enhancing X3D for advanced MR appliances.“ In: ACM SIGGRAPH u.a.: Proceedings WEB3D 2007: 12th International Conference on 3D Web Technology. New York: ACM Press, 2007, pp. 27-36.

Behr, Johannes; Dähne, Patrick; Jung, Yvonne: „Beyond the Web Browser – X3D and Immersive VR.“ In: IEEE Computer Society u.a.: IEEE Virtual Reality 2007. VR Tutorial and Workshop Proceedings [CD-ROM]: IEEE Symposium on 3D User Interfaces. Piscataway, NJ: IEEE Service Center, 2007.

Seibert, Helmut; Dähne, Patrick: „System Architecture of a Mixed Reality Framework.“ Journal of Virtual Reality and Broadcasting, 3(2006), no. 7, May 2007, urn:nbn:de:0009-6-7774, ISSN 1860-2037.

Seibert, Helmut; Dähne, Patrick: „System Architecture of a Mixed Reality Framework.“ In: Braz, José (Ed.) u.a.; Institute for Systems and Technologies of Information, Control and Communication (INSTICC): GRAPP 2006. Proceedings: Proceedings of the First International Conference on Computer Graphics Theory and Appilications. Setúbal: INSTICC Press, 2006, pp. 200-207.

Behr, Johannes; Dähne, Patrick: „Avalon - Ein skalierbares Rahmensystem für dynamische Mixed-Reality Anwendungen.“ In: Schenk, Michael (Hrsg.); Fraunhofer-Institut für Fabrikbetrieb und -automatisierung (IFF): 8. IFF-Wissenschaftstage 2005. Tagungsband: Virtual Reality and Augmented Reality zum Planen, Testen und Betreiben technischer Systeme. Magdeburg: Fraunhofer IFF, 2005, S. 121-130.

Malerczyk, Cornelius; Dähne, Patrick; Schnaider, Michael: „Exploring Digitized Artworks by Pointing Posture Recognition.“ In: Mudge, Mark (Ed.) u.a.; ACM SIGGRAPH u.a.: VAST 2005. Proceedings: The 6th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage. Aire-la-Ville: Eurographics Association, 2005, pp.113-119.

Behr, Johannes; Dähne, Patrick; Knöpfle, Christian: „A Scaleable Sensor Approach for Immersive and Desktop VR Applications.“ HCI International 2005. Proceedings of the 12th International Conference on Human-Computer Interaction. Las Vegas, USA.

Malerczyk, Cornelius; Dähne, Patrick; Schnaider, Michael: „Pointing Gesture-Based Interaction for Museum Exhibits.“ HCI International 2005. Proceedings of the 12th International Conference on Human-Computer Interaction. Las Vegas, USA.

Dähne, Patrick; Seibert, Helmut: „Managing Data Flow in Interactive MR Environments.“ In: Skala, Vaclav (Ed.); European Association for Computer Graphics (Eurographics): WSCG 2005 [CD-ROM]: Full Papers, Short Papers, Posters, Journal of WSCG. Plzen: University of West Bohemia, 2005.

Behr, Johannes; Dähne, Patrick; Roth, Marcus: „Utilizing X3D for Immersive Environments.“ In: Spencer, Stephen N. (Ed.); ACM SIGGRAPH: Web3D 2004. Proceedings: Nineth International Conference on 3D Web Technology. New York: ACM, 2004, pp. 71-78, 182.

Behr, Johannes; Dähne, Patrick: „AVALON: Ein komponentenorientiertes Rahmensystem für dynamische Mixed-Reality Anwendungen.“ In: Thema Forschung (2003), 1, S. 66-73.

Vlahakis, Vassilios; Ioannidis, Nikos; Karigiannis, John; Tsotros, Manolis; Gounaris, Michael; Stricker, Didier; Gleue, Tim; Dähne, Patrick; Almeida, Luis: „Archeoguide: An Augmented Reality Guide for Archaeological Sites.“ In: IEEE Computer Graphics and Applications 22 (2002), 5, pp. 52-60.

Stricker, Didier; Karigiannis, John; Dähne, Patrick; Ioannidis, Nikos: „ArcheoGuide - A Mobile Augmented Tracking System for Archeological Sites – A Solution to the Tracking Problematic.“ In: Stanke, Gerd (Ed.) u.a.; Gesellschaft zur Förderung angewandter Informatik e.V. (GFaI): EVA 2002. Proceedings: Elektronische Bildverarbeitung & Kunst, Kultur, Historie 2002. Berlin: Gesellschaft zur Förderung angewandter Informatik e.V., 2002, pp. 33-40.

Sá, Vítor; Dähne, Patrick: „Accessing Financial Data through Virtual Reality.“ In: Figueiredo, Antonio Dias de (Ed.) u.a.: Proceedings of 3ª Conferência da Associação Portuguesa de Sistemas de Informação 2002. CDROM. Coimbra, 2002

Dähne, Patrick; Karigiannis, John N.: „Archeoguide: System Architecture of a Mobile Outdoor Augmented Reality System.“ In: Müller, Stefan (Ed.) u.a.; Institute of Electrical and Electronics Engineers (IEEE) u.a.: IEEE and ACM International Symposium on Mixed and Augmented Reality 2002. Proceedings. Los Alamitos, Calif.: IEEE Computer Society, 2002, pp. 263-264.

Vlahakis, Vassilios; Karigiannis, John.; Ioannidis, Nicos; Tsotros, Manolis; Gounaris, Michael; Stricker, Didier; Dähne, Patrick; Almeida, Luis: „3D Interactive, On-site Visualization of Ancient Olympia.“ In: Institute of Electrical and Electronics Engineers (IEEE): 3D Data Processing Visualization and Transmission, 2002. Proceedings. Los Alamitos, Calif.: IEEE Computer Society, 2002, pp. 337-345.

Gleue, Tim; Dähne, Patrick: „Design and Implementation of a Mobile Device for Outdoor Augmented Reality in the ARCHEOGUIDE Project.“ In: ACM SIGGRAPH u.a.: VAST 2001: Proceedings of the International Symposium on Virtual Reality, Archaeology and Cultural Heritage : Delegates' Edition. Glyfada, 2001, S. 181-187.

Stricker, Didier; Dähne, Patrick; Seibert, Frank; Christou, Ioannis T.; Almeida, Luis; Carlucci, Renzo; Ioannidis, Nikos: „Design and Development Issues for ARCHEOGUIDE: An Augmented Reality based Cultural Heritage On-site Guide.“ In: Giagourta, Venetia (Ed.) u.a.; European Commission / European Project INTERFACE IST u.a.: International Conference on Augmented, Virtual Environments and Three-Dimensional Imaging. Proceedings 2001. 2001, S. 1-5.

Hergenröther, Elke; Dähne, Patrick: „Real-time Virtual Cables based on Kinematic Simulation.“ In: Skala, Vaclav; IFIP Working Group 5.10 on Computer Graphics and Virtual Worlds u.a.: WSCG 2000. Conference Proceedings Vol. 2. Plzen: University of West Bohemia, 2000, S.402-409.

11 Lebenslauf

Persönliche Informationen	Name:	Patrick Dähne
	Geburtsdatum:	4. Juni 1973
	Geburtsort:	Hamburg
	Nationalität:	deutsch
	Familienstand:	ledig
	Wohnort:	Mathildenplatz 11, 64283 Darmstadt

Schulausbildung	1979-1992	Schulbesuch in Hamburg, München und Reinbek (Schleswig-Holstein) Abschluss Abitur am Sachsenwald-Gymnasium in Reinbek, Schleswig-Holstein Leistungsfächer: Physik, Deutsch Wahlpflichtfächer: Mathematik, Geschichte
Studium	1992-1999	Technische Universität Darmstadt Studiengang Informatik mit Abschluß Diplom-Informatiker Diplomarbeit „Echtzeitsimulation virtueller Kabel“ am Fraunhofer Institut für Graphische Datenverarbeitung in Darmstadt, Abteilung „Virtuelle und Erweiterte Realität“
Beruf	1999-2006	Wissenschaftlicher Mitarbeiter am Zentrum für Graphische Datenverarbeitung, Darmstadt
	seit 2006	Wissenschaftlicher Mitarbeiter an der Technischen Universität Darmstadt, Fachbereich Informatik, Fachgebiet Graphisch-Interaktive Systeme