GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Ulrich Müller
Martin Leissler, Matthias Hemmje

Entwurf und Implementierung eines generischen Mechanismus zur dynamischen Einbettung multimedialer Daten in VRML-Szenen auf der Basis eines objektrelationalen DBMS

N° 23/1998

GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin
Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Anschriften der Verfasser/Address of the authors:
Ulrich Müller
Fliederstraße 55c
D-65396 Walluf
E-mail: ulrich.mueller@main-rheiner.de

Martin Leissler, Matthias Hemmje
Institut für Integrierte Publikations- und Informationssysteme
GMD – Forschungszentrum Informationstechnik GmbH
Dolivostraße 15
D-64293 Darmstadt
E-mail: {leissler,hemmje}@gmd.de

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:

Müller, Ulrich:
Entwurf und Implementierung eines generischen Mechanismus zur dynamischen Einbettung multimedialer Daten in in VRML-Szenen auf der Basis eines objektrelationalen DBMS / Ulrich Müller ; Martin Leissler ; Matthias Hemmje. GMD – Forschungszentrum Informationstechnik GmbH. - Sankt Augustin : GMD – Forschungszentrum Informationstechnik, 1998
(GMD Research Series ; No. 23)
Zugl.: Darmstadt, Techn. Univ., Diplomarbeit U. Müller, 1998
ISBN 3-88457-347-0

ISSN 1435-2699
ISBN 3-88457-347-0

ABSTRACT

Keywords: VRML, Databases, ORDBMS, Multimedia

This Document discribes the design and implementation of a concept which allows to integrate external (mulimedia) data into a VRML scene using a server side include mechanism. The dynamic VRML scenes are managed by an object-relational DBMS (Informix Dynamic Server) and can be manipulated through the VRML DataBlade module which has been developed at GMD-IPSI.

The implementation represents a an extension to this DataBlade. Besides the integration of data, a so called template mechanism has been implemented which allows to generate multiple instances of arbitrary VRML subgraphs and integrate them into an existing VRML scene. With this mechanism it is possible to generate spatially as well as logically scaled VRML scenes

ABSTRACT

Schlagworte:VRML, Datenbanken, ORDBMS, Multimedia

Es wurde ein Konzept entworfen und implementiert, das es erlaubt, mit Hilfe eines Server-Side-Include-Mechanismus externe (multimediale) Daten in eine VRML-Szene zu integrieren. Die dynamischen VRML-Szenen werden von einem objekt-relationalen DBMS (Informix Dynamic Server) verwaltet und sind mit Hilfe des in der GMD-IPSI entwickelten VRML-DataBlades manipulierbar.

Die Implementierung fand im Rahmen einer Erweiterung dieses DataBlades statt. Neben der Integration von Daten wurde ein sogenannter "Templatemechanismus" entwickelt, der es erlaubt, beliebige VRML-Teilszenen mehrfach zu instanziieren und in eine bestehende VRML-Szene einzubinden. Mit diesem Mechanismus kann man sowohl räumliche als auch logische, skalierte VRML-Szenen erzeugen.

INHALTSVERZEICHNIS

1 Einleitung

Zur Zeit existieren verschiedene Möglichkeiten, Informationen im World Wide Web (WWW) zu repräsentieren. Am häufigsten werden Informationen zweidimensional durch HTML (Hypertext Markup Language) präsentiert.

Man hat sehr schnell erkannt, daß die Visualisierung von Informationen durch diese Art der Präsentation stark eingeschränkt wird. Für eine im Internet vertretene Baugesellschaft wäre es z. B. sehr vorteilhaft, wenn ihre Kunden die Möglichkeit hätten, die angebotenen Häuser in einem dreidimensionalen virtuellen Szenario zu besichtigen. Sie wären durch diese neue Informationsvisualisierung mit einer integrierten, von ihnen beeinflußbaren Navigation in der Lage, sich einen persönlichen Eindruck zu verschaffen. Es spielt dabei keine Rolle, ob das Haus bereits real oder lediglich im Katalog existiert. Alle herkömmlichen zweidimensionalen Informationsdarstellungen, die eine derartige Beschreibung ermöglichen sollen, würden hier scheitern, da man in diesem Fall lediglich über Beschreibungen in Form von Texten, Bildern oder bestenfalls nicht intertaktiven Filmen verfügt.

Um eine virtuelle Welt - im oben angeführten Beispiel die dreidimensionale innere Ansicht eines Hauses - im WWW beschreiben zu können, wurde VRML (Virtual Reality Modeling Language) entwickelt. Diese 3D-Szenen-Beschreibungssprache schließt die Lücke, um neben zwei- auch dreidimensionale Darstellungen von Informationen visualisieren zu können. Man hat mit VRML die Möglichkeit geschaffen, virtuelle Welten plattformunabhängig auf einem Computer darzustellen. Man benötigt lediglich einen Browser, der mit einem VRML-Plug-in ausgestattet ist. Darüber hinaus ist der Zugriff auf VRML-Szenen belibig: über das Internet, Intranet oder durch lokalen Zugriff auf die eigene Festplatte.

Für das oben angeführte Beispiel würde es ausreichen, einen fertigen Satz statischer VRML-Szenenbeschreibungen zu jedem angebotenem Haus auf einem Internet-Server anzubieten. Es hat sich jedoch gezeigt, daß es unbedingt erforderlich ist, kundenspezifische Wünsche in die Informationsdarstellung zu integrieren. Diesem Wunsch ist man sehr schnell nachgekommen.

Zur Zeit verfügt man im zweidimensionalen Bereich über eine große Anzahl von HTML-Erweiterungen. Eine dieser Erweiterungen ist z.B. das "Active Server Pages"-Konzept von Microsoft mit dem man dynamische HTML-Seiten generieren kann. Diese HTML-Seiten werden erst auf Anfrage eines Benutzers erzeugt und liegen nicht fertig vor, da unter anderem auch aktuelle Daten z.B. aus einer Datenbank integriert werden können.

Im dreidimensionalen Bereich gibt es im Vergleich dazu zur Zeit lediglich konzeptionelle Ansätze und nur ganz wenige nicht proprietäre Realisierungen, obwohl bereits eine große Anzahl von Anwendungsmöglichkeiten existiert. Für Kunden in dem oben beschriebenen Beispiel wäre es vorteilhaft, wenn sie die Möglichkeit hätten, das Modell des Hauses nach eigenen Wünschen zu modifizieren (z.B. Einfügen zusätzlicher Fenster). Derartige Veränderungen müssen dann z.B. in einer Datenbank nachträglich gespeichert werden. Bei einem späteren erneuten Zugriff auf die gleiche virtuelle Szene würde dann der letzte benutzerspezifische Zustand angezeigt. Dies ist nur mit Hilfe einer dynamischen Generierung der betreffenden VRML-Szene und einer gekoppelten Einbettung aktueller Daten aus einer Datenbank möglich.

In der vorliegenden Arbeit wird ein Konzept entwickelt und implementiert, welches die oben angesprochene Dynamisierung von VRML-Szenen ermöglicht. Als Basis dient ein objektrelationales Datenbanksystem. Es verwaltet Informationen, die für den Aufbau einer online-berechneten VRML-Szene erforderlich sind. Alle zusätzlichen VRML-Sprachkomponenten sind konform der aktuellen VRML-Spezifikation. Die virtuellen Szenen sind ohne Einschränkungen auf jedem Browser darstellbar, der VRML unterstützt. Die Spezifikation und Implementierung dieses neuen Konzepts sollen spätere Erweiterungen vereinfachen.

Gliederung

Im Kapitel 2 werden die drei gebräuchlichsten Datenbanksystemformen und ihre Einsatzmöglichkeiten vorgestellt. Anschließend werden das verwendete objektrelationale Datenbanksystem "Informix Dynamic Server" (IDS) und seine Erweiterungsmöglichkeiten durch DataBlades erläutert.

Das Kapitel 3 bietet eine Übersicht über VRML, den heutigen Standard zur Beschreibung dreidimensionaler Szenen. Es werden die Entwicklung, die Darstellungs- und Einsatzmöglichkeiten sowie der Sprachaufbau beschrieben.

Im Kapitel 4 werden die bereits vorhandenen konzeptionellen Ansätze für eine Verknüpfung von VRML mit Datenbanken erklärt. Es folgt eine Überleitung und Motivation zu dem Konzept, das in dieser Arbeit entwickelt wird.

Im Kapitel 5 werden das Konzept und die daraus resultierenden Mechanismen mit Hilfe einer Spezifikation und diversen Beispielen erläutert. Darüber hinaus wird das bereits vorhandene VRML-DataBlade beschrieben, in das die erforderlichen Erweiterungen integriert werden.

Das Kapitel 6 beschreibt die Implementierung des Konzepts und die Anforderungen, die bei der Entwicklung von objektrelationalen Erweiterungen zu berücksichtigen sind.

Im Kapitel 7 wird der praktische Einsatz des erweiterten VRML-DataBlades anhand einer virtuellen Messe gezeigt, in der Standplätze gemietet und freigegeben werden können.

In Kapitel 8 wird abschließend auf künftige Erweiterungsmöglichkeiten hingewiesen.

2 Objektrelationale Datenbanken

Relationale Datenbanksysteme sind heute immer noch am weitesten verbreitet. Es hat sich jedoch in vielen Fällen gezeigt, daß diese Kategorie von Datenbanksystemen komplexen Anwendungen nicht mehr genügt. Möchte man z.B. multimediale Daten, wie Audio- oder Videodaten, in einem relationalen Datenbanksystem verwalten und verarbeiten, so stößt man sehr schnell auf unüberbrückbare Hindernisse. Es wurden deshalb neue Datenbanksysteme entwickelt, die im folgenden beschrieben werden. Der interessierte Leser findet eine detailliertere Beschreibung und Gegenüberstellung der verschiedenen Systeme in [STO96]. Das objektrelationale Datenbanksystem "Informix Dynamic Server" (IDS) wird im Anschluß vorgestellt. Es wird zur Umsetzung des in dieser Arbeit entwickelten Konzepts verwendet.

2.1.1 Relationale Datenbanksysteme

Charakteristik:

Alle relationalen Datenbanksysteme (RDBS) verwenden das relationale Datenmodell, das 1970 von Codd eingeführt wurde [COD70]. Dieses Datenmodell basiert auf einer simplen und einheitlichen Datenstruktur, welche die mathematische endliche Relation verkörpert.

Beliebige Ausschnitte der realen Welt können durch Tabellen abgebildet werden, welche die oben genannten Relationen repräsentieren. Diese Tabellen sind in den meisten Fällen vielfach miteinander verknüpft. Diese Verknüpfungen, sogenannte Beziehungen, haben die Form n:m mit $n \geq 1$ und $m \geq 1$. Man kann z.B. Einträge einer Tabelle X, welche Informationen über unterschiedliche Abteilungen in einer Firma enthält, mit Einträgen einer Tabelle Y, die spezifische Daten zu jedem Angestellten speichert, verknüpfen. Diese Beziehung gibt dann an, in welcher Abteilung ein Angestellter arbeitet.

Eine Zeile in einer Tabelle bezeichnet man als Tupel. Es besteht aus einer geordneten Menge von Werten. Mit Hilfe der Basisoperationen der relationalen Algebra (Vereinigung, Differenz, Kartesisches Produkt, Projektion und Selektion) kann man die Tupel verarbeiten. Es können z.B. Tupel aus verschiedenen Tabellen vereinigt und die darin enthaltenen Informationen gezielt selektiert werden [ELMASRI94].

Für die Definition des Datenmodells, die Datenmanipulation sowie die Datenselektion wird eine eigene Datenbanksprache, das sogenannte SQL (Structured Query Language), verwendet. SQL wurde 1986 international standardisiert (SQL1). Die derzeitige Version ist SQL92. Sie ist deklarativ aufgebaut, d.h. der Benutzer fragt an, was er möchte, und

nicht, wie er es bekommt. Dies bedeutet, daß alle Aktivitäten, die das Datenbank-Managementsystem (DBMS) für die Lösung der Anfrage durchführt, auf der Anwenderseite verborgen bleiben. Diese Kapselung ermöglicht eine ständige Optimierung des Anfragesystems durch den Hersteller.

Anwendungsgebiete:

Typische Einsatzgebiete relationaler Datenbanksysteme sind die klassischen betriebswirtschaftlichen und administrativen Anwendungen wie Warenwirtschaft, Lohn- und Gehaltsabrechnung, Buchhaltung und Management-Informationssysteme.

Kennzeichnend für diese Anwendungen ist die Tatsache, daß die Daten eine einfache Struktur haben; sie bestehen vornehmlich aus Zahlen und Buchstaben oder kurzen Texten.

Vor- und Nachteile:

Wesentliche Vorteile eines relationalen Datenbanksystems sind:

- Die Verarbeitung der Daten ist sehr effizient, d.h. Datenzugriffe und Datenveränderungen sind in der Regel sehr einfach und durch den Einsatz leistungsfähiger Verfahren äußerst schnell.
- Es besteht mit SQL/SQL92 eine intuitive und international standardisierte Sprache für alle relationalen Datenbanksysteme.
- Der Benutzer ist sehr gut vor den internen Verarbeitungsabläufen einer Anfrage an die Datenbank geschützt, da das relationale Paradigma die logisch-begriffliche Ebene betont, was z.B. durch die Deklarativitätseigenschaften von SQL zum Ausdruck kommt.

Die Erfahrungen mit dem relationalen Modell haben jedoch folgende grundsätzliche Schwächen herauskristallisiert [CT 5/97]:

- Das Problem der Identität: Jede Entität in relationalen Datenbanken kann lediglich durch einen Primärschlüssel identifiziert werden. Da es aber z.B. vorkommen kann, daß sich Werte von Primärschlüsselattributen nachträglich ändern, ist nicht gewährleistet, daß die Identität unberührt bleibt. Es fehlen systemgenerierte unveränderliche Standardnamen, die bei der Erzeugung einer Entität automatisch vergeben werden.
- Das Problem komplexer Werte: Es gibt keine Möglichkeit, komplexe Attribute zu bilden, die aus mehreren Werten eines gleichen oder unterschiedlichen Datentyps bestehen. Attribute sind im relationalen Datenmodell immer elementar d.h. es können lediglich Standardtypen verwendet werden. Dies führt zu einer unnatürlichen Repräsentation von Daten, welche die reale Welt widerspiegeln sollen. So kann z.B. ein geometrisches Objekt, welches durch Koordinaten spezifiziert wird, in einer re-

lationalen Datenbank nicht als Einheit angesehen werden. Die interne Repräsentation erfolgt lediglich in zerlegter Form durch Speichern der einzelnen Koordinatenwerte in unterschiedlichen Spalten.

- Das Problem großvolumiger Datenobjekte: Es ist unmöglich, großvolumige Daten, wie z.B. Multimediadokumente zu interpretieren oder zu manipulieren. Der Einsatz von BLOBs (Binary Large Objects) ist eine unsaubere Lösung, da sie im relationalen Modell nicht vorkommen und deshalb in einer SQL-Anfrage nicht verarbeitet werden können.
- Das Problem der Vererbung: Im Datenbankentwurf erhält man oft hierarchische Subsumptionsbeziehungen zwischen Daten. Diese können im relationalen Modell nicht abgebildet werden.
- Das 'Impedance Mismatch'-Problem: Es besteht eine Diskrepanz zwischen SQL und imperativen Programmiersprachen wie C/C++. SQL basiert auf mathematischer Mengenverarbeitung; das Ergebnis einer Anfrage wird immer in Form einer Menge von Werten zurückgeliefert. Imperative Programmiersprachen sind lediglich in der Lage, einzelne Tupel der Ergebnismenge zu verarbeiten, was zu einer umständlichen Weiterverarbeitung der Abfrageergebnisse führt.

2.1.2 Objektorientierte Datenbanksysteme

Charakteristik:

Die heutigen objektorientierten Datenbanksysteme (OODBS) weisen zwei verschiedene evolutionäre Ansätze auf. Es wurden entweder Datenbanksysteme um objektorientierte Konzepte erweitert oder eine objektorientierte Programmiersprache um diverse Datenbankfunktionalitäten ergänzt [NEU96].

Die zu verwaltenden Informationen werden in Form von Objekten als Instanzen einer Klasse gespeichert. Neben dem Anlegen von Objektattributen, die lediglich Daten repräsentieren, besteht die Möglichkeit, Methoden für Objekte einer Klasse zu erstellen. Mit diesen Methoden können z.B. mit Hilfe von Daten, die in dem gleichen Objekt gespeichert sind, komplexe Berechnungen durchgeführt werden.

In der Datenbankforschung gab es mehrere Versuche, die Eigenschaften eines objektorientierten Datenbanksystems zu definieren. Es gibt bis heute keinen einheitlichen Standard. Um dennoch eine Abgrenzung gegenüber den beiden anderen Systemvarianten zu ermöglichen, wird eine Charakterisierung mit Hilfe einer Zusammenfassung des wohl anerkanntesten Vorschlags, dem "Object-Oriented Databasesystem Manifesto", erstellt [MANIFESTO89]:

Laut den Verfassern des oben genannten Manifests muß ein OODBS folgende Eigenschaften aufweisen:

Es muß ein DBMS mit folgenden Eigenschaften enthalten:

- **Persistenz**: Jedes Objekt, unabhängig vom Typ, muß bei Bedarf persistent gemacht werden können. Dies bedeutet, daß ein Objekt auch nach Ablauf eines Prozesses, in dem es verwendet wird, dauerhaft erhalten bleibt.
- **Sekundärspeicherverwaltung**: Der Benutzer muß sich nicht um die Speicherung der Daten kümmern. Es existiert eine klare Trennung zwischen einer logischen und physikalischen Sicht auf die Daten.
- **Mehrbenutzerkontrolle**: Das System muß mehrere Benutzer, die alle gleichzeitig mit dem System arbeiten, unterstützen.
- **Recovery**: Im Falle eines Hard- oder Softwarefehlers muß das System in der Lage sein, einen konsistenten Datenbankzustand wiederherzustellen.
- **Abfragesprache**: Es muß eine deklarative High-Level-Abfragesprache ähnlich wie SQL vorhanden sein, die effizient und anwendungsunabhängig ist.

Es muß ein objektorientiertes System mit folgenden Eigenschaften sein:

- **Komplexe Objekte**: Mit Hilfe orthogonaler Typkonstruktoren können neue komplexe Objekte erzeugt werden, die sich aus anderen Objekten zusammensetzen.
- **Objektidentität**: Objekte haben eine von ihrem Zustand unabhängige Identität, d.h. selbst zwei Objekte mit identischen Eigenschaften sind **nicht** gleich, da sie jeweils über unterschiedliche Objektidentifikatoren (OIDs) angesprochen werden können.
- **Datenkapselung**: Jedes Objekt hat eine von außen sichtbare Schnittstelle und einen unsichtbaren Implementierungsabschnitt. Dies führt zu einer logischen Datenunabhängigkeit, bei der interne Veränderungen und Optimierungen an der Klasse des Objekts vorgenommen werden können, ohne die Anwendungen, die dieses Objekt benutzen, zu beeinflussen.
- **Klassen**: Man kann gleichartige Objekte in einer Kollektion, die sogenannte Klasse, zusammenfassen.
- **Vererbung**: Eine Klasse vererbt alle Attribute und Methoden an alle abgeleiteten Unterklassen.
- **Polymorphismus**: Eine Methode kann auf Objekte verschiedener Klassen angewendet werden, d.h. Methoden mit dem gleichen Namen können verschiedene Implementierungen besitzen.
- **Computational Completeness**: Im Gegensatz zu SQL soll die in einem OODBS integrierte Abfragesprache die mit einem Computer berechenbare Funktion ausführen können.

Um konzeptionelle Klarheit und einen einheitlichen Standard zu schaffen, haben sich 1991 fünf führende Hersteller von objektorientierten Datenbanksystemen (Objectivity, Ontos, Object Design, O2 und Versant) zur ODMG (Object Data Management Group) zusammengeschlossen. Die von ihnen entwickelte ODMG93-Spezifikation [ODMG93] enthält die Objektdefinitionssprache ODL, die deklarative Objektabfragesprache OQL und diverse Bindings, welche Objektmanipulationen in OO-Programmiersprachen wie C++ oder Smalltalk ermöglichen. Es ist unter anderem ein erklärtes Ziel der ODMG, die vorgeschlagene deklarative Objekt-Abfragesprache OQL kompatibel zu SQL92 zu machen.

Anwendungsgebiete:

Typische Einsatzgebiete eines OODBS sind Anwendungen, die komplexe Informationen wie z.B. Bilder, Filme oder Audioaufnahmen verwalten. Man verwendet ein OODBS z.B. bei Dokumentenverwaltungssystemen, Netzwerkmanagementsystemen, geographischen Informationssystemen und CAD/CAM/CIM-Anwendungen. Die Bearbeitung der Daten ist in diesen Applikationen oft komplex. Sie erstreckt sich in den meisten Fällen auf einen längeren Zeitraum, was durch die oben angeführten Eigenschaften unterstützt wird [CT 5/97]. Der Schwerpunkt eines OODBS liegt bei der sehr engen Integration einer Programmiersprache, der hohen Performanz für einen Zugriff auf persistente Objekte und der Verwaltung komplexer Daten [STO96].

Vor- und Nachteile:

Die wesentlichen Vorteile eines objektorientierten Datenbankssystems gegenüber einem relationalen Datenbanksystem sind [BUCH96]:

- Eine universelle Modellierungsform und effiziente Handhabung komplexer Objekte,
- systemweit eindeutige Objektidentifikatoren,
- Objekte beliebiger Komplexität werden intern immer als eine Einheit angesehen; man ist nicht wie bei einem RDBS gezwungen, die Struktur von Objekten in der realen Welt mit Hilfe von Fremdschlüsseln zu spezifizieren.
- benutzer-definierte Methoden für ein komplexes Objekt,
- kein "Impedance Mismatch"-Problem, da auch nicht mengenorientierte Operationen zugelassen sind.

Die heutigen objektorientierten Datenbanksysteme haben auch einige Nachteile:

- Es existiert keine international genormte deklarative Objektabfragesprache, also auch keine allgemeingültigen Abfrageoptimierungen. Die Abfragesprache eines OODBS ist oft nicht so leistungsfähig wie die eines RDBS. In der ODMG93-Spezifikation werden im Gegensatz zu SQL92 weder Integritätsbedingungen noch Trigger unterstützt.

- Gegenüber einem RDBS ist die Speicherung und Verwendung einfacher Datentypen erheblich aufwendiger, da diese ebenfalls als Objekttypen modelliert werden müssen.
- Der Benutzer eines OODBS muß wissen, wie man durch die Datenbank navigiert, um Informationen wiederzugewinnen. Dies ist im Vergleich zu einem RDBS ein unnötiger Mehraufwand und führt zu einem sehr komplizierten und komplexen Zugriffsmechanismus.
- Es existiert sehr oft keine einheitliche Schemaverwaltung. Dies hat unter anderem zur Folge, daß es keine benutzerspezifischen Zugriffsrechte gibt.

2.1.3 Objektrelationale Datenbanksysteme

Charakteristik:

Objektrelationale Datenbanksysteme (ORDBS) basieren auf einem objektrelationalen Datenmodell. Es handelt sich dabei um ein relationales Datenmodell, das um eine objektorientierte Kernfunktionalität erweitert wurde. Dies bedeutet, daß man die positiven Eigenschaften der relationalen Datenbanksysteme (z.B. die Entkopplung von logischer und physikalischer Struktur sowie die deklarative Abfragesprache SQL) beibehält. Die herkömmliche SQL-Funktionalität wird jedoch um objektorientierte Konzepte erweitert: Die Vorteile der relationalen und der objekt-orientierten Datenbanksysteme werden vereint.

Daten werden wie in einem RDBS in Tabellen gespeichert (s. Kapitel 2.1.1) mit dem Unterschied, daß Tabellen als Instanziierungen eines zuvor deklarierten Typs angesehen werden.

Michael Stonebreaker, einer der Väter des objektrelationalen Modells, charakterisiert ein ORDBS anhand von vier Eigenschaften folgendermaßen [STO96]:

- **Basisdatentypenerweiterung:** Im Gegensatz zu relationalen Modellen sind die verwendbaren Basisdatentypen nicht auf Standarddatentypen beschränkt. Man kann neue Basisdatentypen definieren, die in Tabellen und Abfragen genauso verwendet werden können, wie die bereits vorgegebenen Standarddatentypen und –methoden. Das System behandelt neu hinzugefügte Basisdatentypen genauso wie vorgegebene Standarddatentypen. So ist es beispielsweise möglich, einen neuen Basisdatentyp für die Speicherung und Manipulation kontinuierlicher Medien wie Audio oder Video einzuführen. Mit Hilfe von anschließend neu definierten Methoden kann man dann gespeicherte Audiodaten verändern.
- **Komplexe Datentypen:** Es lassen sich komplexe Datentypen definieren, die aus mehreren vorgegebenen oder neu definierten Basisdatentypen bestehen. Es ist auch möglich, daß ein komplexer Datentyp wiederum andere komplexe Datentypen ent-

hält oder von diesen abgeleitet wird. Die Objekte eines komplexen Datentyps erhalten einen systemweit eindeutigen Objektidentifikator (OID). Komplexe Datentypen werden mit Hilfe abstrakter Datentypen wie Mengen, Listen oder Schlangen erzeugt. Wie bei Basisdatentypen können auch für komplexe Datentypen Methoden eingeführt werden, die SQL unterstützt.

- **Vererbung:** Es besteht die Möglichkeit, Datentypen und Methoden zu vererben. Mit Hilfe der Vererbung kann man sogenannte Typhierarchien erstellen. Das Überladen von Funktionen und Mehrfachvererbung ist ebenfalls möglich.
- **Regeln:** Das Konzept des Triggers in RDBS wird durch Regeln erweitert. Dies sind durch Abfrage- oder Änderungsereignisse ausgelöste Operationen, mit deren Hilfe man die Integrität von Daten schützen kann. Die durch die Regeln eingeführten Integritätsbedingungen sorgen unter anderem dafür, daß der Datenbestand konsistent bleibt-

Man bemüht sich wie die ODMG, eine einheitliche international anerkannte Standardisierung für objektrelationale Datenbanksysteme zu entwickeln. Der momentan anvisierte Standard heißt SQL3 [SQL3] und seine Verabschiedung wird für 1999 erwartet. SQL3 soll SQL92 um objektorientierte Kernfunktionalitäten wie abstrakte Datentypen, OIDs und multiple Vererbung erweitern [CT 5/97].

Anwendungsgebiete:

Ein ORDBS wird bei Anwendungen eingesetzt, in denen die Funktionalität eines RDBS nicht mehr ausreicht. Dazu gehören Applikationen, wie sie bereits oben für ein OODBS beschrieben wurden. Der Schwerpunkt liegt im Vergleich zu einem OODBS jedoch auf dem effizienten Zugriff und der **Manipulation** komplexer Daten durch SQL-Anweisungen [STO96].

Vor- und Nachteile:

Ein ORDBS hebt die meisten Nachteile auf, die ein RDBS besitzt (s. Kapitel 2.1.1):

- Das Identitätsproblem wurde durch die Einführung von Objektidentifikatoren behoben.
- Das Problem komplexer Werte gibt es nicht mehr. Man kann komplexe Datentypen erzeugen, die aus beliebig verschachtelten Zusammensetzungen anderer Datentypen bestehen.
- Das Problem großvolumiger Datenobjekte ist durch die Einführung von LOBs (Large Objects) gelöst worden. Zu einem LOB können wie zu Basisdatentypen Methoden definiert werden. Diese Methoden werden sowohl in der Datenmanipulationssprache als auch in der Abfragesprache unterstützt.

- Es gibt kein Vererbungsproblem mehr: Neue Typen können von bereits definierten Typen abgeleitet werden.
- Mit einem gleichzeitigen Funktionalitätsgewinn bleibt die optimale Leistung eines ORDBS bei allen Erweiterungen erhalten. Viele Methoden, die bei einem RDBS auf der Anwenderseite ausgeführt werden, können bei einem ORDBS in den wesentlich leistungsfähigeren Kern des Datenbankservers verlagert werden.

Natürlich hat ein ORDBS auch Nachteile:

- Es existiert wie bei einem OODBS noch keine international standardisierte Datenmanipulations- und Abfragesprache. Jeder Hersteller hat eigene Erweiterungen eingeführt. Dies erschwert eine Portierung bereits existierender Anwendungen auf ein anderes ORDBS.
- Es werden nicht konsequenterweise alle OO-Konzepte übernommen, und es gibt auch keine allgemeingültigen Richtlinien, welche OO-Konzepte unterstützt werden sollen und welche nicht.

2.2 Informix Dynamic Server

In dieser Arbeit wurde ein objektrelationales Datenbanksystem um zusätzliche Funktionalitäten erweitert. Als ORDBS wurde der Informix Dynamic Server (IDS) verwendet. Die Architektur und die objektrelationalen Erweiterungsmöglichkeiten des IDS durch sogenannte DataBlades werden in diesem Kapitel näher erläutert.

2.2.1 Architektur

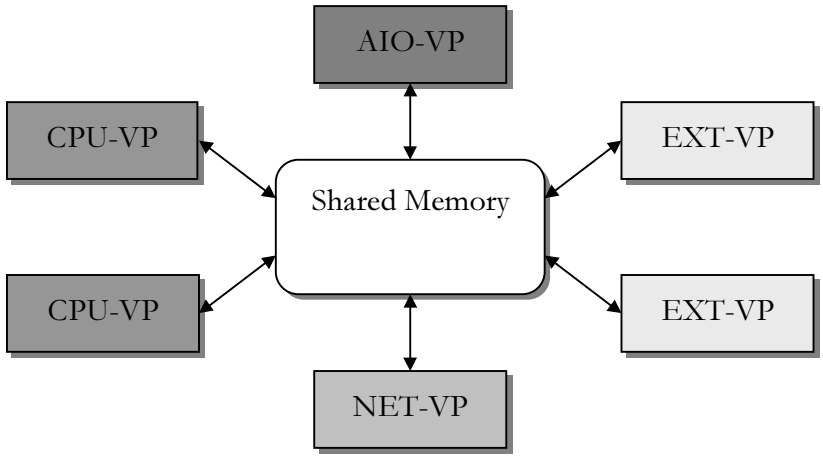
Der IDS ist ein hochperformanter Datenbankserver, der Multithreading unterstützt. Er kann auf einem Einprozessor- oder Multiprozessorsystem eingesetzt werden und ist für die Plattformen SGI IRIX, Sun Solaris, SNI, IBM/AIX, HP-UX, DEC-Unix und Microsoft Windows NT verfügbar.

Informix bezeichnet die Architektur des IDS als "Dynamic Scalable Architecture" (DSA). Das zentrale Element dieser Architektur ist der "virtuelle Prozessor", ein Datenbank-Serverprozeß der ab jetzt mit "VP" bezeichnet wird. Er wird virtueller Prozessor genannt, weil seine Funktionalität einer CPU in einem Computer ähnelt. So wie eine CPU verschiedene Systemprozesse bearbeitet, um mehrere Benutzer gleichzeitig zu bedienen, so bearbeitet ein VP verschiedene Threads, um mehrere SQL-Anfragen gleichzeitig zu verarbeiten. Die Architektur des IDS besteht aus mehreren VPs.

Der IDS teilt eine an ihn gestellte Aufgabe in Teilaufgaben auf und verschickt diese jeweils zu den VPs. Bei Verwendung eines Multiprozessorsystems werden die VPs den

zur Verfügung stehenden Prozessoren zugeordnet. Der IDS kann dann von den Vorteilen einer hochperformanten Parallelverarbeitung profitieren. Die Abbildung 1 zeigt die interne Architektur des IDS [DBDK97]:

ABBILDUNG 1: Architektur des IDS



In dem oben gezeigten Diagramm repräsentiert jedes Rechteck mit einem grauen Hintergrund einen VP. Die Anzahl der VPs ist nicht festgelegt und kann zur Laufzeit mit Hilfe administrativer Einstellungen dynamisch verändert werden. So können z.B. abhängig von Performanzvorgaben oder der zu verwaltenden Datenmenge mehr VPs hinzugefügt werden.

Shared Memory

Die VPs kommunizieren miteinander über einen gemeinsamen Speicherbereich (engl. shared memory). Die Größe dieses Speicherbereichs kann abhängig von der Auslastung einer Anfrage, variieren. Diese Veränderung ist für alle VPs sichtbar. In dem gemeinsamen Speicherbereich werden folgende Inhalte festgehalten:

- Daten zu Tabellen und Indices
- Informationen zu den aktiven Anfragen
- Informationen zu Sperren, die von den aktiven Anfragen erzeugt wurden.

Der Adressraum jedes VPs ist identisch. Unter Unix erreicht man dies z.B. durch Anwendung des bekannten `fork()`-Befehls. Alle VPs werden mit Hilfe dieses Befehls aus einem einzelnen Master-VP erzeugt. Das hat den Vorteil, daß der Aufwand für eine Threadmigration, d.h. die Übergabe eines Threads von einem VP zu einem anderen VP, sehr gering bleibt. Die Threadmigration ist der Schlüssel zur Parallelverarbeitung, da durch diesen Mechanismus verschiedene Threads gleichzeitig verarbeitet werden können.

DataBlade-Funktionen, mit denen man die Funktionalität des IDS erweitert, allokieren Speicherplatz über eigene DataBlade-API-Funktionen, die diesen aus dem gemeinsamen Speicherbereich zuweisen.

CPU-Virtual Processor

Der "CPU-Virtual Processor" (CPU-VP) ist das "Herz" des IDS. Er führt den Großteil der erforderlichen Arbeit für das Parsen, die Planung und die Optimierung einer SQL-Anfrage aus. Der IDS kann mehrere CPU-VPs enthalten. Wenn ein Benutzer eine Anfrage an den IDS stellt, wird sie von einem CPU-VP empfangen und geparst. Das Resultat des Parsens ist ein Parser-Baum. Mit Hilfe dieses Baumes wird ein Plan erstellt, aus dem z.B. später hervorgeht, welche Tabellenspalten für die Ermittlung der Ergebnisse benötigt werden. Nach einer Optimierung der Anfrage wird die effizienteste Ausführung berechnet.

Die Daten einer Tabelle können auf verschiedene Speichermedien verteilt sein. Falls nun eine Anfrage alle Daten einer solchen Tabelle benötigt, werden mehrere Subthreads gebildet, die die Verarbeitung der Anfrage übernehmen. Jeder von ihnen kann dann in unterschiedlichen Speicherbereichen agieren. Die Subthreads können nun in einem Mehrprozessorsystem parallel oder in einem Einprozessorsystem seriell verarbeitet werden. Die Resultate jedes Subthreads werden von dem Masterthread in der CPU-VP gesammelt und in aufbereiteter Form als Antwort auf die Anfrage an den Benutzer zurückgegeben.

Extension Virtual Processor

Der "Extension-Virtual Processor" (EXT-VP) führt prinzipiell die gleiche Arbeit aus wie der CPU-VP. Der Unterschied besteht in der Entschärfung einiger Restriktionen. Diese von dem CPU-VP auferlegten Einschränkungen sind z.B. bei der Entwicklung von Funktionen eines DataBlades (s. a. Kapitel 6.1.2) zu berücksichtigen. Das Wegfallen dieser Restriktionen im EXT-VP geschieht auf Kosten der Performanz, d.h. der EXT-VP ist in der Ausführung von Threads langsamer als der CPU-VP.

Asynchronous I/O Virtual Processor

Der "Asynchronous I/O Virtual Processor" (AIO-VP) übernimmt die Aufgaben der asynchronen Ein- und Ausgabe. Wird durch einen Thread in einem CPU-VP eine Anfrage an ein externes Medium wie z.B. eine Festplatte gestellt, so wird diese Anfrage von dem AIO-VP übernommen. Informationen zur asynchronen Ein-/Ausgabe und ihre Vorteile gegenüber einer synchronen Ein-/Ausgabe findet man in [TAN92].

Network Virtual Processor

Der "Network Virtual Processor" (NET-VP) ist für die asynchrone Netzwerkkommunikation zwischen Clients und dem IDS verantwortlich. Er übernimmt alle Kommunikationsaufgaben von Clients, die mit Hilfe der Informix API eine Verbindung zu dem IDS aufgebaut haben.

Es gibt noch andere Typen von VPs, deren spezifische Anzahl und Ausprägung von der Konfiguration des IDS abhängt. Der interessierte Leser findet eine detaillierte Beschreibung der übrigen VPs in [ADMIN98].

2.2.2 Erweiterungen mit DataBlades

Der IDS bietet die Möglichkeit, seine Kernfunktionalität um spezielle Eigenschaften zu erweitern. Dies geschieht mit Hilfe sogenannter DataBlades. Ein Data-Blade besteht aus einer Sammlung benutzerdefinierter objektrelationaler Erweiterungen und unterstützt einen ganz bestimmten Anwendungsbereich. So gibt es z.B. das Web-DataBlade, welches eine Verwaltung von HTML-Seiten mit einer gekoppelten dynamischen Einbettung externer Daten ermöglicht (s. a. Kapitel 5.3.3). Man kann für die Unterstützung einer Anwendung auch mehrere DataBlades gleichzeitig verwenden.

Ein DataBlade kann die folgenden objektrelationalen Erweiterungen enthalten:

Erweiterte Datentypen

Neben den bereits vorhandenen Standarddatentypen, die in [SQLREF97] ausführlich beschrieben sind, hat der Anwender die Möglichkeit, eigene Datentypen zu definieren. Der IDS bietet dafür vier verschiedene Möglichkeiten:

1. *Collection Data Type*

Ein *Collection Data Type* ist ein komplexer Datentyp, der aus einem oder mehreren Elementen aufgebaut sein kann. Jedes Element ist vom gleichen Datentyp, und es sind beliebige Verschachtelungen von *Collection Data Types* erlaubt. Der IDS unterstützt für den Aufbau eines *Collection Data Types* drei verschiedene Typkonstruktoren:

1.1 list

Eine Liste (engl. list) ist eine geordnete Sammlung von Elementen, die Duplikate erlaubt.

Beispiel:

```
create table Terminplaner
(
     Tag                date,
     Termine            list (datetime hour to minute),
     primary key        (Tag)
);
```

1.2 set

Eine Menge (engl. set) ist eine ungeordnete Sammlung von Elementen, die keine Duplikate erlaubt.

Beispiel:

```
create table Angestellter
(
     ID           varchar(40) NOT NULL,
     Name         varchar(100),
     Name_Kind    set (varchar (100)),
     primary key  (ID)
);
```

1.3 multiset

Eine Multimenge (engl. multiset) ist eine ungeordnete Sammlung von Elementen, die Duplikate erlaubt.

Beispiel:

```
create table Ersatzteil
(
     ID           integer,
     Name         varchar(100),
     Komponenten  multiset (integer),
     primary key  (ID)
);
```

2. Row Data Type

Ein *Row Data Type* besteht aus Datenfeldern, die jeweils wie in einer Tabellendefinition durch einen Bezeichner und einen Datentyp beschrieben werden. Es können mehrere Datenfelder vom gleichen Typ vorhanden sein und es können alle IDS-Datentypen als Element eines *Row Data Types* verwendet werden. Ein *Row Data Type* bildet eine Scha-

blone (engl. template), um Datenrecords zu erstellen. Er ist vergleichbar mit einem Struct in C oder einer C++-Klasse ohne Memberfunktionen.

Ein *Row Data Type* kann verwendet werden, um eine Tabelle zu erstellen. In einem solchen Fall werden die Spalten einer Tabelle durch die Datenfelder des *Row Data Types* definiert.

Row Data Types können sowohl benannt als auch unbenannt sein. Unbenannte *Row Data Types* werden lediglich durch ihre Struktur identifiziert und können ausschließlich für die Typdefinition einer Spalte oder eines Datenfeldes verwendet werden. Benannte *Row Data Types* werden dagegen durch ihren Namen identifiziert.

Beispiel eines benannten *Row Data Types*:

```
create row type Person
(
     Name          varchar(100),
     Geburtstag    date
);
```

3. Distinct Data Type

Ein *Distinct Data Type* hat die gleiche interne Speicherstruktur wie ein bereits existierender Datentyp, unterscheidet sich davon aber im Namen. Ein neu eingeführter *Distinct Data Type* erbt alle Funktionen und Casts des ursprünglichen Typs. Er muß jedoch explizit umgewandelt werden, falls man ihn mit dem ursprünglichen Typ vergleichen will.

Man verwendet *Distinct Data Types,* um z.B. Inhalte einer Tabellenspalte treffender beschreiben zu können.

Beispiel:

```
create distinct type html as clob;
create table Internetseiten
(
     ID            varchar(40) NOT NULL,
     Seite         html,
     primary key (ID)
);
```

4. Opaque Data Type

Der *Opaque Data Type* ist ein fundamentaler benutzerdefinierter Datentyp, der einen einzigen Wert speichert und vom IDS nicht weiter in Komponenten zerlegt werden kann. Dies liegt daran, daß seine interne Struktur dem IDS nicht bekannt ist. Die Werte eines *Opaque Data Types* werden im IDS lediglich als Kette von Bytes bestimmter Länge behandelt. Der Zugriff auf die interne Struktur, die mit Hilfe einer C-Struktur imple-

mentiert wird, erfolgt durch benutzerdefinierte Funktionen. Diese Funktionen werden in der Datenbank registriert und so dem IDS bekannt gemacht.

Beispiel:

```
create opaque type Person (internallength=104);

// C-Struktur (dem IDS nicht bekannt)
struct Person
{
     char Name[100];
     long Geburtstag;
}
```

UDR-Routinen

Ein DataBlade bietet dem Anwender die Möglichkeit, Funktionen und Prozeduren zu definieren und in einer Datenbank des IDS zu registrieren. Diese UDR-Routinen (engl. user defined routines) werden dynamisch zum Serverprozeß gelinkt und wie die Standardroutinen im Kern des IDS ausgeführt. Sie sind entweder in C/C++ oder in SPL (Stored Procedure Language) geschrieben. SPL ist eine von Informix entwickelte SQL-Erweiterung und führt Kontrollstrukturen wie Schleifen und bedingte Verzweigungen ein. Im Falle von C/C++ werden die UDR-Routinen in einer Bibliothek (engl. shared object file) zusammengefaßt. Diese wird dynamisch in den Shared-Memory-Bereich des IDS geladen, sobald eine in ihr enthaltene Routine aufgerufen wird. Die UDR-Routinen werden im CPU-VP oder im EXT-VP ausgeführt (s. a. Kapitel 2.2.1).

Sie werden von Client-Anwendungen verwendet und stellen Operationen für neu eingeführte erweiterte Datentypen bereit.

Beispiel für die Registrierung einer in C geschriebenen Funktion:

```
create function Wurzel(double precision) returning double precision
external name '/home/username/.../math.bld(math_sqrt)' language C end
function;
```

Casts

Falls man in einem DataBlade neue erweiterte Datentypen, z.B. mit Hilfe von *Distinct Data Types* einführt, so werden Datentypkonvertierungen (engl. casts) benötigt, die einen Datentyp in einen anderen Datentyp umformen können. Man hat dann unter anderem die Möglichkeit, Werte unterschiedlichen Datentyps miteinander zu vergleichen. Falls während der Konvertierung Daten modifiziert werden müssen, wird dazu eine UDR-Routine benutzt.

Beispiel für die Registrierung eines Casts, der eine UDR-Routine benötigt:

```
create distinct type vrml as clob;

create IMPLICIT cast (lvarchar AS vrml WITH lvarcharToVRML );

create function lvarcharToVRML(s lvarchar) returns vrml
external name '/home/username/.../vrdb.so(lvarcharToBlob)' language C
end function;
```

Tabellen- und Indexdefinitionen

Durch die Verwendung eines DataBlades können auch neue Tabellen eingeführt werden. Falls z.B. eine UDR-Routine die mitgegebenen Parameter mit Hilfe von Referenzwerten validieren muß, kann man diese Referenzwerte in einer neuen Tabelle speichern, die in das DataBlade integriert wird. Für neu eingeführte Tabellen, auf die sehr oft zugegriffen wird, können in einem DataBlade auch Indices für eine oder mehrere Spalten definiert werden. Ein Index ermöglicht eine schnellere Bearbeitung von SQL-Anfragen, falls bei der Suche entsprechend indexierte Spalten benutzt werden. Indices benötigen in der Regel sehr viel Speicherplatz und bei einer Modifikation werden nicht zu unterschätzende Server-Ressourcen beansprucht. Deshalb sollte man sie nur verwenden, nachdem man eine eingehende Analyse der Performanz aller SQL-Anfragen vorgenommen hat, welche die eventuell zu indexierenden Tabellenspalten verwenden.

Fehlerkodierungen und Fehlermeldungen

UDR-Routinen können bei einer fehlgeschlagenen Ausführung benutzerdefinierte Fehlermeldungen zurückgeben. Diese Fehlermeldungen sind ebenfalls Bestandteile des DataBlades. Sie werden in Systemtabellen gespeichert und können bei Bedarf in der Server-Log-Protokolldatei oder sonstigen Traversierungs-Protokolldateien ausgegeben werden.

Der IDS behandelt alle durch ein DataBlade neu eingeführten Komponenten, wie z.B. erweiterte Datentypen, genauso wie jene, die er bereits standardmäßig anbietet. Die Erweiterungen sind über SQL, die DataBlade-API oder Clients verfügbar, die die Informix-Client-API benutzen. DataBlades werden in dem IDS durch eine Registrierung angemeldet. Die Registrierung führt unter anderem zu einer Ergänzung der SQL-Sprache, um die in dem DataBlade enthaltenen Erweiterungen.

DataBlades können daher auch auf Datentypen und Routinen anderer DataBlades mit Hilfe von SQL oder der DataBlade-API zugreifen.

Der interessierte Leser kann ausführlichere Informationen und Beispiele zu DataBlades den Quellen [SQLTUT97] und [DBDK97] entnehmen.

3 Virtual Reality Modeling Language

3.1 Überblick

VRML ist die Abkürzung für "Virtual Reality Modeling Language" und existiert seit 1994. Es handelt sich um ein spezielles Dateiformat, mit dem man interaktive 3-dimensionale Welten und die darin enthaltenen Objekte beschreiben kann. VRML-Dateien können wie HTML-Dateien sowohl im Online-Bereich des World Wide Web (WWW) als auch im Offline-Bereich zur Unterstützung von Anwendungen genutzt werden, die dreidimensionale Visualisierungen integrieren.

3.2 VRML 1.0

Die erste Version von VRML 1.0 [VRML1.0] wurde 1995 als internationaler Standard für die Darstellung von 3D-Welten proklamiert. VRML 1.0 wurde von der Firma *Silicon Graphics Inc.* auf der Basis des "Open Inventor"-Dateiformats entwickelt. "Open Inventor" [OPENINVENTOR] ist eine C++-Klassenbibliothek, mit deren Hilfe man dreidimensionale Szenen generieren kann.

Einer der größten Nachteile dieser ersten VRML-Version ist die Einschränkung auf statische Szenen. Es ist zwar möglich, alle erdenkbaren geometrischen Figuren zu modellieren, nicht jedoch, irgendeine Bewegung oder Interaktion in eine VRML-Szene zu integrieren. Ein weiterer Nachteil ist, daß VRML 1.0 keine Programmierschnittstelle anbietet und somit kaum erweiterbar ist.

Die Hersteller der verschiedenen VRML-unterstützenden Browser haben daher schon kurz nach den ersten vollständigen Implementierungen ihre eigenen Erweiterungen zu dem vorgegebenen VRML 1.0 Standard entwickelt. Diese machten es möglich, die oben genannten Einschränkungen zumindest teilweise aufzuheben und animierte, interaktive 3D-Welten zu erzeugen. Die Erweiterungen waren durch keinen Standard festgelegt, sondern in ihrer Kompatibilität auf den jeweils benutzten Browser eingeschränkt.

3.3 VRML 2.0

Um die Weiterentwicklung des Standards zu koordinieren, die Mängel der ersten Version aufzuheben und vor allem eine herstellerunabhängige Basis für die Implementierung neuer Browser zu schaffen, wurde die "VRML Architecture Group" [VAG] gegründet. Der Arbeitstitel dieses neuen Standards hieß "Moving Worlds". Er deutete an, daß VRML vor allem im Bereich der dynamischen und interaktiven Welten verbessert werden sollte. Die neue Version wurde von *Silicon Graphics Inc.* mit Unterstützung von *Sony Research* und *Mitra* entworfen. Am 4. August 1997 proklamierte die "VRML Architecture Group" VRML 2.0 [VRML2.0] zum internationalen Standard.

Mit VRML 2.0 ist man in der Lage, in einer 3D-Welt statische und animierte Objekte darzustellen. Außerdem können URLs integriert werden, die auf die verschiedensten multimedialen Medien, wie z.B. Sounds, Filme und Bilder verweisen. Darüber hinaus existiert ein Erweiterungsmodell, welches die Definition neuer wiederverwendbarer Objekte erlaubt. Es wurde ein Registrierungsprozeß in VRML 2.0 integriert, mit dem Benutzergruppen untereinander austauschbare Erweiterungen zum Basisstandard entwickeln können. Außerdem gibt es eine definierte Abbildung zwischen VRML-Elementen und den Komponenten allgemein verwendeter 3D APIs (Anwendungs-Programmierschnittstellen).

Browser, die VRML 2.0 unterstützen, sind inzwischen auf einer Vielzahl von Rechnerplattformen verfügbar. Die Darstellung einer VRML-2.0-Szene wird mit einem sogenannten "VRML-Plug-in" ermöglicht.

Für VRML 2.0 existieren zur Zeit mehrere Plug-ins, die sich in Qualität, Geschwindigkeit und insbesondere im Grad der Umsetzung der Spezifikation sehr unterscheiden. Der "Cosmoplayer" der Firma *Silicon Graphics Inc.* in der aktuellen Vollversion 2.1 gehört zu dem am weitest entwickelten VRML-Plug-in und ist für die Betriebssystem-Plattformen "SGI IRIX" (in Version 1.0), "Microsoft Windows 95/NT" sowie "PowerMac" verfügbar.

3.3.1 Anforderungen

Ein wichtiges Ziel der Spezifikation von VRML 2.0 war ein einheitliches Dateiformat, mit dem man dreidimensionale interaktive Welten und die darin enthaltenen Objekte beschreiben kann. Dem Benutzer soll es erleichtert werden, komplexe 3D-Szenen, wie Illustrationen, Produktbeschreibungen oder Virtual Reality Umgebungen, zu erschaffen.

Die VRML 2.0 Spezifikation wurde entworfen, um folgende Anforderungen zu erfüllen [VRML2.0]:

- **Authorability.** Es soll für Entwickler auf einfache Weise möglich sein, Programme zu entwicklen, mit deren Hilfe man komplette VRML-Anwendungen realisieren kann. Außerdem soll der Im- und Export industriell weitverbreiteter Datenformate vereinfacht werden.
- **Vollständigkeit.** Sämtliche Informationen, die für eine spätere komplette Implementierung der VRML 2.0-Spezifikation notwendig sind, sollen zur Verfügung gestellt werden. Darüber hinaus soll ein funktional vollständiger Satz von VRML-Fähigkeiten angeboten werden, um breite industrielle Akzeptanz zu erreichen.
- **Kombinierbarkeit.** VRML 2.0 soll die Fähigkeit besitzen, einzelne Elemente miteinander zu kombinieren. Es soll dadurch unter anderem die Wiederverwendbarkeit gefördert werden.
- **Erweiterbarkeit.** Es muß möglich sein, neue Komponenten hinzuzufügen.
- **Implementierbarkeit.** Die Implementierung soll auf unterschiedlichen Plattformen möglich sein.
- **Multi-User Potential.** Es soll keine Einschränkungen für die Implementierung einer Mehrbenutzer-Umgebung geben.
- **Orthogonalität.** Die Elemente von VRML 2.0 sollen so weit wie möglich voneinander unabhängig sein. Bei unvermeidbaren Abhängigkeiten sollen sie besonders gut strukturiert und wohldefiniert sein.
- **Performanz.** Die einzelnen VRML-Elemente sollen unter dem Gesichtspunkt einer guten interaktiven Performanz, die für eine Vielzahl von Computer-Plattformen gelten soll, entworfen werden.
- **Skalierbarkeit.** Die Elemente von VRML 2.0 sollen so konzipiert werden, daß sie in der Lage sind, mit theoretisch unendlich großen Kompositionen umzugehen.
- **Wohlstrukturiertheit.** Jedes Element von VRML 2.0 soll eine wohldefinierte Schnittstelle sowie einen einfachen und nicht an Bedingungen geknüpften Verwendungszweck haben. Alle Mehrzweckelemente und Nebeneffekte sollten vermieden werden.

3.3.2 Fähigkeiten

Der folgende Überblick über die VRML 2.0-Spezifikation soll eine Vorstellung von den wichtigsten Fähigkeiten von VRML 2.0 vermitteln.

Während man mit VRML 1.0 lediglich die Möglichkeit hat, statische 3D-Welten zu entwickeln, bietet VRML 2.0 eine weitaus größere Darstellungsfunktionalität. Das übergeordnete Ziel von VRML 2.0 ist, dem Benutzer eine umfassendere, aufregendere und interaktivere Erfahrung zu ermöglichen, als es die statischen Grenzen von VRML 1.0

erlauben. Das sekundäre Ziel ist die Entwicklung eines soliden Fundaments für zukünftige VRML-Erweiterungen. Dies wird durch die Einführung von Prototypen erreicht (s. Kapitel 3.3.2.5). Diese Erweiterungen, welche die Funktionalität des Standards erhöhen können, sollen so einfach und schnell wie möglich sein. Sie sollen vom Browser-Entwickler über den Entwickler von 3D-Szenarien bis hin zum Endbenutzer jeden zufriedenstellen. Durch die starke Veränderung der Anforderungen und die daraus resultierenden Funktionalitäten konnte in VRML 2.0 keine Abwärtskompatibilität erreicht werden. Im Standard von VRML 2.0 ist jedoch festgelegt, daß ein VRML-2.0-Browser auch Szenen von VRML 1.0 darstellen können muß.

Die Fähigkeiten von VRML 2.0 lassen sich am einfachsten anhand der Erweiterungen und Verbesserungen gegenüber VRML 1.0 beschreiben [VRMLVIEW].

3.3.2.1 Erweiterte statische Welten

Neben den Standardgeometrieknoten (z.B. Kugel, Würfel), erlauben neue Knoten beliebige Boden- und Himmelhintergründe (engl. ground-and-sky backdrops) zu generieren. Man kann damit eine 3D-Szene auf einfache Weise z.B. um Berge und Wolken ergänzen. Darüber hinaus können entfernte Objekte mit Hilfe von Nebel verschleiert werden. Es ist auch möglich, beliebige Hintergründe durch Farbverläufe zu generieren. Ein weiterer neuer Knotentyp erlaubt es, mit einfachen Mitteln unregelmäßige Bodenstrukturen zu erzeugen, ohne daß dazu große Polygonflächen definiert werden müssen.

VRML 2.0 bietet darüber hinaus "räumliche" Soundknoten, um den bereits vorhandenen Klangeindruck weiter zu verstärken. Man kann damit z.B. zirpende Grillen, berstendes Glas, klingelnde Telefone oder jeden anderen Soundeffekt in eine 3D-Szene einbauen. Diese Knoten können Sounddateien in unterschiedlichen Formaten abspielen (s. Kapitel 5.4.4). Je nach Position des Betrachters im Raum kann eine dreidimensionale Klangwelt erzeugt werden, welche die 3D-Szene wie in der Realität unter räumlichen Gesichtspunkten akustisch wiedergibt.

Außerdem wurde die Szenengraphenstruktur (s. Kapitel 3.3.3.1) vereinfacht, mit der eine 3D-Welt in VRML beschrieben wird. Damit lassen sich VRML-Szenenbeschreibungen wesentlich leichter optimieren und parsen.

3.3.2.2 Interaktion

Man kann nun direkt mit Objekten interagieren, auf die man bei der Navigation durch ein 3D-Szenario stößt. Neue Sensoren erzeugen sogenannte Ereignisse (engl. events), sobald der Benutzer in bestimmte Bereiche der Szene vorstößt oder wenn er z.B. bestimmte Objekte mit der Maus anklickt. Der Benutzer ist in der Lage, Objekte einer 3D-Szene von einem Ort zu einem anderen zu verschieben. Ein weiterer Sensor repräsentiert die Zeit der Simulation, wodurch nun Objekte auf ein zeitgesteuertes Ereignis rea-

gieren können. Dies ermöglicht eine breite Palette von Effekten, von Alarmuhren bis zu sich wiederholenden komplexen Animationsequenzen.

Die Kollisionserkennung stellt sicher, daß sich feste Objekte auch wie feste Objekte verhalten, d.h. es wird verhindert, daß ein Benutzer durch geschlossene Objekte hindurchgehen kann. Eine automatische Verfolgung der Navigation über unebene Bodenstrukturen erlaubt es nun auch, Treppen und Rampen hinauf- und hinunterzugehen.

3.3.2.3 Animation

VRML 2.0 beinhaltet eine Vielzahl von Animationselementen, die man als Interpolatoren bezeichnet. Diese ermöglichen es, vordefinierte Animationen in einer 3D-Szene zu modellieren, die dann z.B. aufgrund eines Ereignisses oder dauerhaft abgespielt werden können.

Mit Interpolatoren kann man bewegte Objekte erzeugen, wie fliegende Vögel, automatische Türen, umherlaufende Roboter oder Ähnliches. Außerdem kann man Objekte implementieren, die ihre Farbe mit der Bewegung verändern, wie die Sonne, aber auch Objekte, die ihre Form verändern.

Es sind sogar "Führungen", sogenannte "guided tours", durch die VRML-Welt möglich, die den Benutzer entlang eines vordefinierten Pfades bewegen.

3.3.2.4 Scripting

Eine durch VRML 2.0 beschriebene 3D-Szene käme ohne Skriptknoten nicht wirklich in Bewegung (s. Kapitel 3.3.3.5). Über diese Knoten kann man nicht nur Objekte einer virtuellen Welt animieren, sondern ihnen sogar den Anschein von Intelligenz geben. So können z.B. animierte Hunde nach Zeitungen oder Frisbees schnappen, Uhrzeiger bewegen sich, Vögel fliegen oder Roboter jonglieren. Die Möglichkeiten sind nahezu unbegrenzt.

Die obengenannten Effekte werden durch Ereignisse ermöglicht. Alle Ereignisse, die ein beliebiger VRML-Knoten aussendet, können durch einen Skriptknoten verarbeitet werden. So kann ein solcher Knoten z.B. ein Ereignis von einem Sensor entgegennehmen. Die Ergebnisse der Verarbeitung dieses eingehenden Ereignisses werden wieder in Form von Ereignissen weitergereicht, die dann wiederum andere Knoten im Szenengraph beeinflussen können.

Der Skriptknoten kann darüber hinaus neue Knoten erzeugen oder beliebige Knoten löschen. Eine ständige Kommunikation mit dem Web- oder Datenbankserver ist nicht erforderlich, weil ein Skriptknoten immer auf der Seite des Clients verarbeitet wird. Lediglich bei Anforderungen zusätzlicher Informationen wird eine Verbindung zum Server

aufgenommen. Der Stand der Technik und die verschiedenen Konzepte, VRML mit Datenbanken zu verknüpfen, werden im Kapitel 4 noch näher beleuchtet.

Die Funktionalität eines Skriptknotens wird zur Zeit vornehmlich in Java oder JavaScript implementiert. Für beide Sprachen existieren alle notwendigen Klassen und Funktionen zur Bearbeitung einer VRML-Szenenbeschreibung.

3.3.2.5 Prototyping

Die Idee des Prototyping besteht darin, häufig benötigte Knotentypen, die in der Spezifikation noch nicht enthalten sind, als sogenannte Prototypen zu definieren (s. a. Kapitel 3.3.3.5). Damit läßt sich der Sprachumfang von VRML 2.0 erweitern.

Mit VRML 2.0 kann man z.B. eine Gruppe von mehreren Knoten zu einem Prototypen zusammenfassen, der dann allen Anwendern von VRML 2.0 zugänglich gemacht werden kann. Es lassen sich somit, analog zu Hochsprachen wie C++, Bibliotheken aufbauen, die von jedem Entwickler genutzt werden können.

Man kann wie bei VRML-Standardknoten beliebig viele Instanzen dieses neuen Knotentyps erzeugen.

3.3.3 Aufbau und Struktur

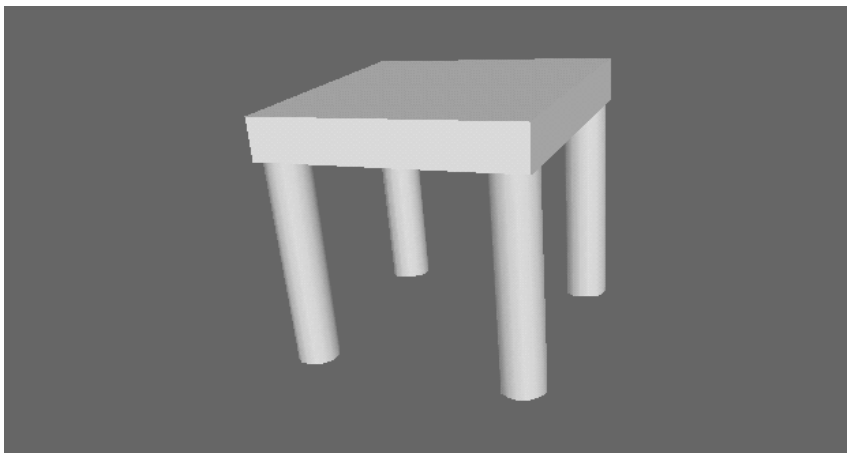
Im folgenden werden der Aufbau und die Struktur von VRML 2.0 beschrieben, indem die wichtigsten Schlüsselkonzepte zusammengefaßt werden.

3.3.3.1 Szenengraph

Eine 3D-Szene wird in VRML durch einen Szenengraphen beschrieben. Der Szenengraph ist ein gerichteter azyklischer Graph. Er besteht aus Knoten, welche die einzelnen Elemente der 3D-Szene und deren Eigenschaften modellieren, aus Sensorknoten, welche die Interaktion mit dem Benutzer unterstützen, und aus sogenannten Gruppenknoten, welche die zuvor genannten Knoten miteinander verknüpfen. Die verschiedenen Knotentypen werden im Kapitel 3.3.3.3 näher erläutert. Die Struktur eines VRML-Szenengraphen basiert auf der bei "Open Inventor" verwendeten (s. a. Kapitel 3.2).

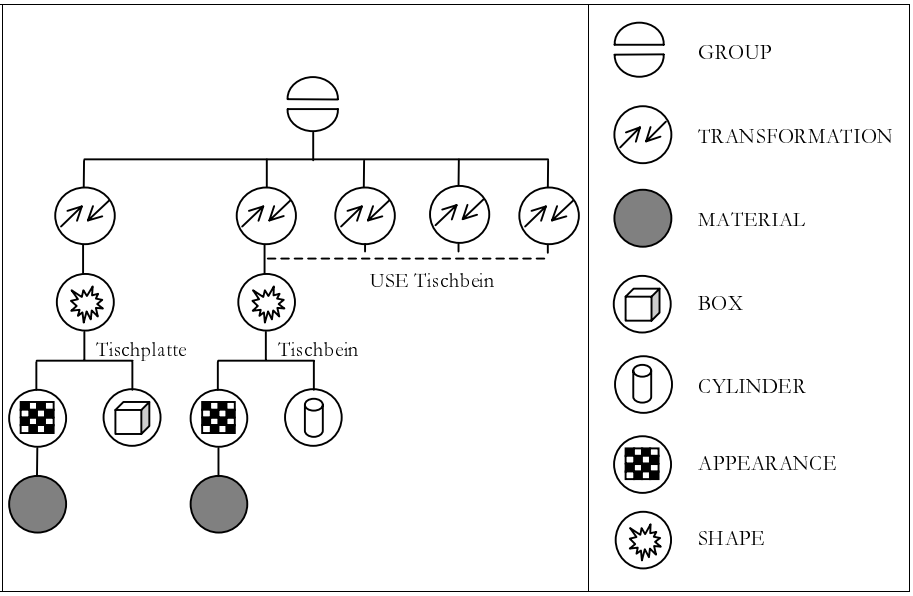
Der Szenengraph weist eine baum-ähnliche Struktur auf. Darüber hinaus läßt er jedoch Referenzen zwischen einem und beliebig vielen Knoten zu. Das ermöglicht die Wiederverwendung von bereits entwickelten Teilszenen. Im folgenden Beispiel wird die dreidimensionale Darstellung eines Tisches realisiert, der in der Abbildung 2 zu sehen ist:

ABBILDUNG 2: Darstellung eines Tisches mit VRML



Der zu diesem Tisch gehörende Szenengraph ist in Abbildung 3 dargestellt. Das Tischbein wird hier nur einmal definiert und durch Referenzen mehrmals benutzt.

ABBILDUNG 3: Szenengraph zur Darstellung eines Tisches



Die Wurzel eines Szenengraphen besteht in der Regel aus einem Gruppenknoten, der wiederum andere Knoten als Kinder umfaßt. In oben gezeigtem Beispiel enthält die Wurzel des Szenengraphen fünf weitere Gruppenknoten, sogenannte Transformknoten. Jeder dieser Transformknoten umfaßt wiederum Knoten für die Darstellung der Tischplatte und die der vier Tischbeine. Im Minimalfall kann die Wurzel auch aus einem einfachen Geometrieknoten bestehen. Die Blätter eines Szenengraphen bestehen aus Knoten, die in der Regel keine weiteren Knoten mehr aufnehmen können.

3.3.3.2 VRML-Datei

Die VRML-Datei ist eine aus reinem ASCII-Text bestehende Beschreibung der zu generierenden 3D-Welt. Der Kopf dieser Datei besteht aus einer verbindlichen Informationszeile mit dem Syntax: "#VRML V2.0 utf8". Sie gibt an, daß der nachfolgende Programmtext die Version VRML 2.0 und den UTF8-Zeichensatz, der gemäß ISO 10646-1:1993 spezifiziert ist, verwendet. Es folgen diverse Deklarationen, Kommentare sowie die Szenengraphen-Beschreibung und die Definition der Weiterleitung von Ereignissen durch die sogenannte ROUTE-Anweisung (s. Kapitel 3.3.3.4). VRML-Dateien besitzen immer die Endung `*.wrl`.

In dem nachfolgend abgebildeten VRML-Quelltext, die den oben gezeigten Szenengraphen beschreibt, sieht man, daß ein Tischbein nur einmal definiert (DEF-Konstrukt) und durch den Einsatz von Referenzierungen (USE-Konstrukt) mehrmals verwendet wird.

```
#VRML V2.0 utf8
# Ein Tisch
Group {
  children [
    Transform {   # Tischplatte
       translation 0 0.6 0
      children
        Shape {
          appearance Appearance {
            material Material { diffuseColor .6 .6 .1 }
          }
          geometry Box { size 1.2 0.2 1.2 }
        }
    }
    Transform {   # erstes Tischbein
      translation -.5 0 -.5
      children
        DEF Tischbein Shape {
          appearance Appearance {
            material Material { diffuseColor .8 .4 .7 }
          }
          geometry Cylinder { height 1 radius .1 }
        }
    }
    Transform {   # zweites Tischbein
      translation .5 0 -.5
      children USE Tischbein
    }
    Transform {   # drittes Tischbein
      translation -.5 0 .5
      children USE Tischbein
    }
    Transform {   # viertes Tischbein
      translation .5 0 .5
      children USE Tischbein
    } ]
}
```

3.3.3.3 VRML-Knoten

Ein VRML-Knoten besteht immer aus folgenden Elementen:

- einer Typangabe,
- einem Paar geschweifter Klammern,
- einer optionalen Anzahl von Feldern mit eingetragenen Werten, mit denen die Eigenschaften des Knotens spezifiziert werden.

Die in VRML verwendeten Knoten sind entweder gemäß der VRML 2.0-Spezifikation vordefiniert oder durch den Einsatz von Prototypen benutzerdefiniert (s. Kapitel 3.3.3.5).

Jedes Feld in einem VRML-Knoten ist einem bestimmten Typ zugeordnet. Eine genaue Beschreibung aller Feldtypen findet man in [VRML2.0]. Man unterscheidet zwischen Einzelwert- und Multiwertfeldern.

Die Einzelwertfelder haben nur einen Wert. Die Typen dieser Felder fangen immer mit "SF" an.

Die Multiwertfelder dagegen können beliebig viele Werte des angegebenen Typs aufnehmen. Die Typen dieser Felder fangen immer mit "MF" an. Die einzelnen Werte in einem Multiwertfeld können über einen Index angesprochen werden.

Jedes Einzelwertfeld in einem Knoten besitzt einen Standardwert, so daß man lediglich die Felder angeben muß, deren Werte nicht den Standardwerten entsprechen. Multiwertfelder sind dagegen standardmäßig leer. Der in dem oben gezeigten VRML-Quelltext verwendete Knoten `Cylinder` ist z.B. folgendermaßen definiert:

```
Cylinder {
     Field  SFBool      bottom      TRUE
     Field  SFFloat     height      2
     Field  SFFloat     radius      1
     Field  SFBool      side        TRUE
     Field  SFBool      top   TRUE
}
```

Die Felder `bottom`, `side` und `top` wurden in dem oben gezeigten Beispiel nicht angegeben, da ihre Standardwerte erwünscht sind.

Die in der VRML 2.0-Spezifikation definierten Knoten sind in folgende Gruppen eingeteilt:

- **Gruppenknoten**

  Hierzu zählen alle Knoten, die Kindknoten aufnehmen können. Sie werden benutzt, um einen hierarchisch aufgebauten Transformationsgraphen zu erzeugen. Jeder Gruppenknoten definiert einen eigenen Koordinatenraum für die Kinder. Somit

kann z.B. eine auf alle Kindknoten akkumulierende Transformation durchgeführt werden.

- **Geometrieknoten**
  Hierzu zählen alle Knoten, die geometrische Objekte beschreiben, z.B. eine Kugel, einen Quader oder einen Zylinder.
- **Geometrische Eigenschaften**
  Dies sind Knoten, die spezielle geometrische Eigenschaften beschreiben, wie z.B. die Farbe eines Objekts.
- **Darstellungsknoten**
  Darstellungsknoten beschreiben, wie ein geometrischer Knoten dargestellt wird, z.B. das Material eines Objekts, die Angabe des Schriftsatzes eines dargestellten Textes oder die durch URLs spezifizierten Texturen geometrischer Figuren.
- **Interpolationsknoten**
  Dies sind Knoten, die Zwischenwerte für vorgegebene Werten interpolieren. Sie werden unter anderem für Animationen und Farbveränderungen eingesetzt.
- **Sensorknoten**
  Sensorknoten generieren Ereignisse, die auf Benutzerinteraktionen basieren. So können z.B. nach einem Mausklick oder einer Annäherung an ein Objekt durch Navigation, ein oder mehrere Ereignisse an andere Knoten geschickt werden, die auf diese Interaktion entsprechend reagieren.
- **Anbindbare Knoten**
  Hierzu zählen Knoten, bei denen nur ein Exemplar pro Typ innerhalb einer VRML-Szene aktiviert ist, obwohl sie mehrfach in der VRML-Datei definiert werden dürfen. So kann man z.B. je nach Tageszeit einen anderen Hintergrund einstellen oder verschiedene Sichten auf die VRML-Szene ermöglichen, zwischen denen der Benutzer umschalten kann.
- **Sondergruppenknoten**
  Zu den Sondergruppen zählen z.B. Knoten wie der "Level Of Detail"-Knoten (LOD-Knoten), bei dem zwischen verschiedenen Knotengruppen umgeschaltet werden kann. Dadurch ist man u.a. in der Lage, je nach Entfernung zum Betrachter, eine gröbere oder feinere Darstellung des Objekts auszuwählen.
- **Verschiedene Knoten**
  Zu dieser Knotengruppe gehören Knoten, die Beleuchtungen, Geräusche, Multimediainhalte und Verhalten einer Szene repräsentieren.

Eine wesentlich detailliertere Beschreibung der verschiedenen Knotengruppen findet sich in [VRML2.0].

3.3.3.4 Verarbeitung von Ereignissen

Um die durch VRML beschriebene Welt dynamisch zu machen, gibt es spezielle Anweisungen um Knoten kommunikativ miteinander zu verknüpfen. VRML-Knoten kommunizieren miteinander über Ereignisse. Die meisten Knoten in VRML können sowohl Ereignisse empfangen als auch verschicken. Es gibt neben den normalen Feldern noch unidirektionale Ereignisfelder (engl. eventIn, event-Out) und bidirektionale Ereignisfelder (engl. exposedField).

Unidirektionale Ereignisfelder zeichnen sich dadurch aus, daß sie keine Standardwerte vorweisen. Sie können lediglich Ereignisse entgegennehmen oder verschicken.

Bidirektionale Ereignisfelder verhalten sich wie die herkömmlichen Felder (s. Kapitel 3.3.3.3), mit dem Unterschied, daß sich ihre Werte abhängig von empfangenen Ereignissen verändern bzw. daß bei einer Wertveränderung Ereignisse ausgesendet werden.

Die Verdrahtung zweier Knoten geschieht über die sogenannte ROUTE-Anweisung. Damit zwei Knoten eindeutig identifiziert werden können, müssen sie mit Hilfe des DEF-Konstrukts einen eindeutigen Namen erhalten.

Im folgenden Beispiel wird ein Kollisionsgeräusch erzeugt, wenn der Benutzer während einer Navigation auf eine Wand stößt. Die Kollision mit der Wand erzeugt ein Ereignis, die an einen Soundknoten weitergeleitet wird, der dann ein Kollisionsgeräusch erzeugt:

```
DEF Wand Collision { children [
  Transform {
    #... Geometrie der Wand ...
  }
  Sound {
    source DEF Kollisionsgeraeusch AudioClip {
      url "http://...../kollision.wav"
    }
  }
 ]
}
ROUTE Wand.collision TO Kollisionsgeraeusch.startTime
```

3.3.3.5 Erweiterungsmöglichkeiten

Dem Benutzer stehen verschiedene Möglichkeiten zur Verfügung, die Funktionalität von VRML zu erweitern:

- **DEF/USE-Konstrukt**

  Der Benutzer hat mit dem DEF-/USE-Konstrukt die Möglichkeit, eine bereits definierte Teilszene mehrfach zu verwenden (s. a. Kapitel 3.3.3.2). Allerdings ist die Verwendung des DEF-/USE-Konstrukts auf eine 3D-Szene beschränkt. Darüber hinaus ist man nicht in der Lage, der einzufügenden Teilszene Parameter mitzugeben, die z.B. die Farbe der Teilszene beeinflussen könnten.

- **Prototypen**

  Aufgrund der eingeschränkten Flexibilität des DEF-/USE-Konstrukts hat man in VRML 2.0 das Prototyping eingeführt (s. a. Kapitel 3.3.2.5). Mit Hilfe von Prototypen kann man, ähnlich wie Funktionen in herkömmlichen Programmiersprachen, durch die PROTO-Anweisung Objekte definieren, die von verschiedenen Parametern abhängen.

  Im Szenengraphen werden die Objekte eines Prototyps so behandelt wie gewöhnliche Knoten. Im folgenden Beispiel ist der bereits aus Abbildung 2 bekannte Tisch als Prototyp definiert worden. Man hat nun die Möglichkeit, Tische mit unterschiedlichen Farben zu erzeugen, indem jeweils die Farbe der Tischplatte und die Farbe der Tischbeine als Parameter mitgegeben werden:

```
#VRML V2.0 utf8

PROTO ZweiFarbenTisch [ field SFColor TischbeinFarbe  .8 .4 .7
                        field SFColor TischplatteFarbe .6 .6 .1
]
{
  Transform {
    children [
      Transform {   # Tischplatte
       translation 0 0.6 0
        children
          Shape {
            appearance Appearance {
              material Material { diffuseColor IS TischplatteFarbe }
            }
            geometry Box { size 1.2 0.2 1.2 }
          }
      }
      Transform {   # erstes Tischbein
       translation -.5 0 -.5
        children
          DEF Tischbein Shape {
            appearance Appearance {
              material Material { diffuseColor IS TischbeinFarbe }
            }
            geometry Cylinder { height 1 radius .1 }
          }
      }
      Transform {   # zweites Tischbein
       translation .5 0 -.5
        children USE Tischbein
      }
      Transform {   # drittes Tischbein
       translation -.5 0 .5
        children USE Tischbein
      }
      Transform {   # viertes Tischbein
       translation .5 0 .5
        children USE Tischbein
      }
```

```
    ]
  }
}

# Erzeugung eines Prototypenobjekts
ZweiFarbenTisch { TischbeinFarbe 1 0 0 TischplatteFarbe 0 1 0 }
```

Ein Prototyp kann auch extern, d.h. nicht in der aktuellen Szene, definiert werden. Er wird dann bei Bedarf nachgeladen. Dies geschieht durch die EXTERNPROTO-Anweisung:

```
EXTERNPROTO MeinZweiFarbenTisch [ ... ] "lib.wrl#ZweiFarbenTisch"
```

- **Skriptknoten**

Viele Aktionen sind zu komplex, um sie mit den in VRML 2.0 vorhandenen Standardknoten zu realisieren. Dazu zählen aufwendige Animationen, die z.B. eine künstliche Gravitation widerspiegeln sollen, komplexe Geometrien wie Fraktale und kollaborative Umgebungen wie Computerspiele.

Um solche komplexen Anwendungen in einem VRML-Szenengraphen zu integrieren, verwendet man Skriptknoten (s. a. Kapitel 3.3.2.4). Ein Skriptknoten enthält ein Skript, das in JavaScript, Java oder anderen Programmiersprachen geschrieben ist. Diese Skripte können unter anderem eingehende Ereignisse verarbeiten, Ereignisse erzeugen und Felder anderer Knoten lesen und schreiben. Das Spektrum der Funktionalität eines solchen Skripts ist nahezu unbegrenzt. Skripte, welche mit Hilfe von JavaScript erstellt wurden, können sowohl innerhalb der Knotendefinition als auch in einer externen Datei definiert sein. Skriptknoten-Funktionalitäten, die mit Hilfe von Java eingebunden werden, sind immer als Java-Klassen in einer externen Datei definiert.

3.3.4 Anwendungsgebiete

VRML wird überall dort eingesetzt, wo die zweidimensionale Darstellung von Informationen unzureichend ist. Die Bewegung durch eine mit VRML erzeugte virtuelle Welt ist dem Benutzer vertraut und erfordert keine speziellen Kenntnisse.

VRML hat längst das Interesse der meisten Softwarehersteller auf sich gezogen. Fast jede 3D-Software wird spätestens in ihrer nächsten Version VRML-2.0-Szenen exportieren oder sogar importieren können. Das Spektrum an Einsatzmöglichkeiten für VRML ist äußerst vielfältig:

Man findet im Internet eine Fülle von wissenschaftlichen VRML-Anwendungen aus den verschiedensten Forschungsbereichen. Diese ermöglichen den einfachen und sehr anschaulichen Zugang zu Forschungsergebnissen.

So wird VRML u.a. in der Chemie genutzt, um komplexe Strukturen wie Moleküle oder DNS-Ketten anschaulich zu präsentieren. Neben der reinen Darstellung werden Interaktionen integriert, um z.B. Moleküle in ihre Bestandteile zu zerlegen oder neue Moleküle zusammenzusetzen (siehe http://ws05.pc.chemie.th-darmstadt.de/vrml/)

Im medizinischen Bereich lassen sich z.B. komplexe anatomische Darstellungen implementieren, die von den verschiedensten Blickwinkeln aus betrachtet werden können (siehe http://reality.sgi.com/sambo/Oobe/CyberAnatomy/intro.html)

VRML kann als interaktives 3D-Lernmedium eingesetzt werden. Die Aus- und Weiterbildung in der Lehre und im Studium kann durch Lerninhalte, die mit VRML aufbereitet wurden, erheblich interessanter gestaltet werden. So kann der Lernende interaktiv in den zu lernenden Stoff eingreifen und seinen kreativen Fähigkeiten freien Lauf lassen.

Im kommerziellen Bereich wird VRML ebenfalls schon in den unterschiedlichsten Bereichen eingesetzt. Es fängt bei einfachen dreidimensionalen Firmen- und Produktpräsentationen an und endet bei komplexen Datenvisualisierungen, die z.B. im Börsenbereich genutzt werden. Der wesentliche Vorteil dieser neuen Darstellungsform ist die Möglichkeit, die Sicht nach dem persönlichen Geschmack oder zum besseren Verständnis zu verändern. Eine sehr gelungene dreidimensionale Darstellung von aufbereiteten Wirtschaftsdaten bietet die Firma *Online Environs Inc.* an (siehe http://dvf.environs.com/)

Darüber hinaus wird VRML bereits für die Realisierung virtueller Museen, z.B. die 3D-Visualisierung der "National Gallery of Ireland" in Dublin (siehe http://www.dmc.dit.ie/guests/eirenet/eirenet/eirenet/thic2.htm), in der Tourismusbranche, z.B. die 3D-Darstellung der Altstadt von Dublin (siehe http://www.dmc.dit.ie/guests/eirenet/eirenet/eirenet/sgithic.htm) und sogar im Sport z.B. anläßlich der Fußball-Weltmeisterschaft 98 (siehe http://www.ardwm98.de/virtual/complete/german/index_1.htm) verwendet.

Wenn in den nächsten Jahren die 3D-Darstellung und Navigation eine annehmbare Qualität erreicht hat (s. a. Kapitel 5.3.1), werden virtuelle Arbeitstreffen ohne räumliche und zeiliche Beschränkungen mit Hilfe von VRML möglich sein. Man kann dann z.B. mit einem Japaner und einem Amerikaner an einem Tisch sitzen, die beide ihr heimisches Büro nicht verlassen müssen, – und das ohne Kosten, abgesehen von den anfallenden Gebühren für Telefon und Internet-Zugang.

Virtuelle Welten, in denen sich mehrere Benutzer – symbolisch durch Stellvertreter (sogenannte "Avatare") – begegnen können, markieren die Zukunft von VRML.

3.4 VRML 97

Es sei an dieser Stelle darauf hingewiesen, daß VRML 2.0 im letzten Jahr als ISO-Standard ("ISO/IEC 14772-1:1997") ratifiziert wurde. Die Kurzschreibweise dieses Standards lautet "VRML97" [VRML97]. Dies gilt als entscheidender Schritt für eine breitere und kommerzielle Nutzung dieser 3D-Szenenbeschreibungssprache.

VRML97 ist nahezu identisch mit VRML 2.0; es weist lediglich einige Korrekturen und Verbesserungen des Spezifikationsdokuments sowie eine geringe Anzahl funktionaler Unterschiede auf. Genauere Information bezüglich der Unterschiede zwischen VRML 2.0 und VRML 97 findet man in [VRMLCHANGE].

Das VRML-DataBlade, welches im Rahmen dieser Diplomarbeit erweitert wurde, ist konform bezüglich des VRML2.0-Standards. Der Grund dafür liegt in der Verfügbarkeit der derzeitigen VRML-Interpreter. Zur Zeit unterstützen noch nicht alle VRML-Plug-ins den neuen Standard VRML97. Betrachtet man aber die wenigen funktionalen Unterschiede zwischen diesen beiden Standards, so sollte es einfach möglich sein, das DataBlade zu modifizieren, um eine Konformität mit VRML97 zu erreichen.

4 Datenbank-Anbindungen für VRML

4.1 Status Quo

VRML bietet als weltweit anerkannter ISO-Standard weitverbreitete Unterstützung für die Darstellung dreidimensionaler Informationen. Es existiert eine große Palette von Werkzeugen für die Erzeugung, Darstellung und Erkundung von VRML-Welten. Experten prophezeien einen stetig wachsenden Markt für integrierte 3D-Informationsvisualisierungen, die mit VRML realisiert werden können (s. a. Kapitel 3.3.4).

Im Dezember 1996 wurde eine Vereinigung für VRML, das sogenannte "VRML-Consortium", ins Leben gerufen mit Jahresgebühren zwischen 7500 und 15000 US-$ für stimmberechtigte Mitglieder [CONSORTIUM]. Es fungiert als Dachorganisation für verschiedene VRML-Arbeitsgruppen. Diese Arbeitsgruppen sind aus den unterschiedlichsten Initiativen hervorgegangen. Ihr gemeinsames Ziel ist eine Erweiterung von VRML um zusätzliche Funktionalitäten. So gibt es z.B. die Arbeitsgruppe "Living Worlds", die an der Spezifikation für eine Schnittstelle arbeitet, welche die Entwicklung von Multiuser-Anwendungen mit Hilfe von VRML ermöglichen soll. Eine andere Arbeitsgruppe mit dem Namen "VRML-CBF" diskutiert und entwickelt ein Binärformat, welches VRML-Dateien komprimiert. Mit diesem Format sollen unter anderem höhere Übertragungsraten von VRML-Dateien im Internet ermöglicht werden. Zur Zeit gibt es 20 verschiedene aktive Arbeitsgruppen.

Eine der gravierendsten Einschränkungen von VRML ist die starke Bindung an herkömmliche Dateisysteme. Zur Zeit exisitieren keine **standardisierten** Technologien für einen Zugriff auf VRML-Welten, welche sich außerhalb einer Fileserver-basierten Datenquelle befinden. Wenn 3D-Informationsvisualisierungen im Internet ihr volles Potential erreichen sollen, wird eine effiziente und flexible Form der Verwaltung von VRML-Dateien in Datenbanken benötigt. Es müssen generische Mechanismen für die Speicherung und den Zugriff von VRML-Daten entwickelt werden, die eine Modifizierung bereits existierender VRML-Welten, eine Einbettung externer Daten und die Verwaltung von Zugriffsrechten ermöglichen. Nur solche Technologien sind eine geeignete Basis für die Entwicklung komplexer und kooperativer VRML-Anwendungen, für skalierbare virtuelle Welten und ausgeklügelte Multiuser-Umgebungen.

Um diese dringend benötigten Datenbankanbindungen, die für HTML schon seit geraumer Zeit existieren, für VRML zu standardisieren, hat sich 1997 die Arbeitsgruppe "VRML Database Working Group" gebildet [DBWORK]. Ziel dieser Arbeitsgruppe ist es, standardisierte Schnittstellen zu spezifizieren, welche die Integration von Datenbankfunktionalitäten in VRML ermöglichen. Ein weiteres Ziel dieser Arbeitsgruppe ist die Popularisierung von VRML als Plattform für die Entwicklung ernstzunehmender kommerzieller Anwendungen. Dies soll erreicht werden, indem man erstens der "VRML-Gemeinschaft" von VRML als Medium im kommerziellen Kontext bewußt macht und zweitens die Wirtschaft von der Mächtigkeit und den Vorteilen von VRML als Anwendungsplattform für 3D-Informationsvisualisierungen überzeugt.

4.2 Datenbankanbindungskonzepte

Die "VRML-Database-Working-Group" hat zwei unterschiedliche Schnittstellenformen identifiziert, die eine Integration von Datenbankfunktionalität in VRML ermöglichen sollen. Es gilt zu beachten, daß im Falle einer Standardisierung dieser Schnittstellen die bisherige VRML-Spezikation um die entsprechenden Komponenten erweitert wird. Die Folge wäre eine Integration dieser neuen Funktionalität in die aktuellen VRML-Browser bzw. deren Plug-ins, damit eine Konformität zu der erweiterten VRML-Spezifikation hergestellt werden kann.

4.2.1 VRML Database API

Die "VRML Database API", repräsentiert eine Schnittstelle zu VRML-Daten, die in der Datenbank gespeichert sind. Sie kann von Entwicklern für VRML-Werkzeuge wie z.B. einen Browser verwendet werden. Diese Schnittstelle ist zur Zeit lediglich abstrakt definiert und soll konkretisiert werden, sobald alle Anwendungsanforderungen eingehend analysiert sind. Folgende Anforderungseigenschaften hat die "VRML-Database-Working-Group" bereits herauskristallisiert:

- **Persistenz**

Bisher fängt der Benutzer beim Laden einer VRML-Szene immer mit dem gleichen unveränderlichen Zustand an, obwohl VRML-Welten sich mit der Zeit verändern und auf Benutzerinteraktionen reagieren können. Alle Veränderungen, die man als Benutzer an Objekten in einer VRML-Welt vorgenommen hat, sind zurückgesetzt und damit nicht mehr erkennbar, wenn man die gleiche Welt erneut betritt. Die Persistenz soll nun VRML-Welten mit permanenten Zustandsveränderungen und die Konsistenz zwischen allen Benutzern einer Welt ermöglichen.

Die API soll dies durch Unterstützung von Sperren, Retrieval- und Update-Funktionalitäten für den VRML-Szenengraphen erreichen.

- **Skalierbarkeit**

Virtuelle Welten, die durch den VRML 2.0-Standard beschrieben werden, haben eine begrenzte Größe, weil der Browser des Clients allein verantwortlich ist für die Berechnung und Darstellung einer 3D-Szene. Die Inline-, LOD- und Anchorknoten (s. Kapitel 3.3.3.3) reichen nicht aus für eine nahtlose Navigation (engl. seamless navigation) durch eine sehr große und komplexe VRML-Welt.

Die API soll abhängig von logischen und räumlichen Gesichtspunkten mit Hilfe einer benutzerdefinerten Skalierung eine optimierte nahtlose Navigation durch VRML-Welten unterstützen.

- **Sicherheit**

Es gibt zur Zeit keine Standardmechanismen für die Zugriffskontrolle auf VRML-Welten oder die Kontrolle erlaubter Interaktionen.

Die API soll Benutzerrechte, Autorisierung und die benutzerspezifische Zughörigkeit von VRML-Objekten unterstützen.

4.2.2 VRML Database Extensions

Die "VRML Database Extensions" repräsentieren eine Schnittstelle zu relationalen Datenbankoperationen innerhalb einer VRML-Szene. Die Schnittstelle besteht aus einem Satz von PROTO-Anweisungen (s. Kapitel 3.3.3.5), die von VRML-Anwendungsentwicklern benutzt werden können. Diese neuen Prototypen repräsentieren standardisierte "High-Level"-Knoten, mit denen man allgemeine Datenbankoperationen innerhalb der VRML-Szenenbeschreibung angeben kann.

Die "VRML-Database-Working-Group" hat drei verschiedene Konzepte entwickelt, mit denen die Eignung von VRML als Entwicklungsplattform von kommerziellen 3D-Anwendungen im Internet entscheidend verbessert werden kann:

4.2.2.1 Embedded SQL

Es existieren momentan keine Standards für die Ausführung von SQL-Anweisungen innerhalb einer laufenden VRML-Anwendung. Man behilft sich mit Skriptknoten, die in Java geschriebene JDBC-Anfragen ausführen können (s. Kapitel 3.3.3.5). Nachteil dieser nicht standardisierten Methode ist die komplexe Technologie und der unvermeidbare Overhead, der auch bei einfachen SQL-Anweisungen auftritt. Darüber hinaus hängt der Nutzen eines solchen Skriptknotens von dem Grad der Unterstützung von Java in dem jeweiligen Browser ab.

Die "Extension" stellt einen "High-Level"-Knoten zur Verfügung, der die oben genannten Einschränkungen aufhebt. Er kann in einer VRML-Szene eingebettete SQL-Anweisungen, wie SELECT, INSERT oder UPDATE zur Laufzeit ausführen. Ein solcher Knoten wird durch ein Ereignis aktiviert. Er führt dann eine SQL-Anweisung aus, und die Ergebnisse werden wieder mit Hilfe von Ereignissen an andere Knoten geschickt.

4.2.2.2 Server Side Includes

Ein Server-Side-Include-Mechanismus kann eine gerade angeforderte VRML-Szene modifizieren. Diese Veränderungen werden ausschließlich auf der Serverseite vorgenommen. Das bedeutet, daß der Verbindungsaufbau zur Datenbank, die Selektion und das Einfügen der Daten in die VRML-Szene im Gegensatz zu dem vorherigen Konzept **nicht** im Browser (auf der Client-Seite) stattfinden. Dieser Mechanismus wird bei HTML in ähnlicher Form schon seit vielen Jahren angewendet.

Momentan exisitieren keine standardisierten Server-Side-Include-Mechanismen für die Integration von Daten in VRML-Szenen. Die "Extension" stellt einen "High-Level"-Knoten zur Verfügung, der die oben beschriebene Funktionalität erfüllt.

4.2.2.3 Trigger

Eine VRML-Welt ist eine abgeschlossene Welt, d.h. es findet keine Kommunikation mit der Außenwelt statt. Bisher kann diese Kluft nur mit Hilfe sehr aufwendiger Java-Netzwerk-Programmierung überbrückt werden. Damit hat man dann die Möglichkeit externe Ereignisse wie z.B. Datenbank-Trigger mit VRML-Ereignissen zu verbinden. Dabei werden Browser, die gerade VRML-Szenen aus der entsprechenden Datenbank anfordern, in einem speziellen Server registriert. Dieser baut eine TCP/IP-Socket-Verbindung zu einem Java-Programm auf, das in dem entsprechenden Browser läuft. Falls nun in der Datenbank eine Veränderung stattfindet, die angeforderte und noch aktive VRML-Szenen beeinflußt, so werden mit Hilfe der Socket-Verbindung alle betroffenen VRML-Szenen aktualisiert. So lassen sich Echtzeit-Multiuser-Anwendungen realisieren.

Die "Extension" stellt einen "High-Level"-Knoten zur Verfügung, der die oben erläuterte Funktionalität realisiert und den bisherigen Aufwand für die Generierung von VRML-Ereignissen, die auf Datenbank-Triggern basieren, minimiert.

4.3 Dynamisierung eines VRML-Knotens

Bei der Integration von (multimedialen) Daten in eine VRML-Szenenbeschreibung unterscheidet man zwischen zwei verschiedenen Formen der Dynamisierung eines VRML-Knotens. Der Begriff Dynamisierung bezeichnet hier die Fähigkeit eines Knotens externe Daten, die z.B. in einer Datenbank gespeichert sind, einer VRML-Szene hinzuzufügen und sie damit zu modifizieren.

Die erste Form ermöglicht die dynamische Integration externer Daten zur **Laufzeit** (s. a. Kapitel 4.2.2.1) und die zweite zur **Ladezeit**. Letztere wurde mit Hilfe des in dieser Arbeit entwickelten "SQLServerInclude-Knotens" entwickelt, der nachfolgend als "SSI-Knoten" bezeichnet wird.

Der SSI-Knoten verwendet einen Mechanismus, mit dem man ein Server-Side-Include-Mechanismus innerhalb einer VRML-Szene ausführen kann (s. a. Kapitel 4.2.2.2). Die einzufügenden Daten werden mit Hilfe einer SELECT-Anweisung ermittelt, die als Feldwert in dem SSI-Knoten steht.

Die Ergebnisse dieser Anweisung können auf zwei verschiedene Arten verarbeitet werden. Die erste Möglichkeit besteht darin, daß man sie in Felder anderer VRML-Knoten einsetzt, die zweite basiert auf einer mehrfachen Instanziierung einer in dem SSI-Knoten vorgegebenen VRML-Teilszene. Die Instanziierungsanzahl hängt von der Größe der Ergebnismenge der SELECT-Anweisung ab. Diese beiden Mechanismen werden in der Spezifikation des SSI-Knotens noch genauer erklärt (s. Kapitel 5.4).

Nachdem die Daten in die VRML-Szene eingefügt sind, wird der SSI-Knoten abhängig vom jeweiligen Mechanismus entweder gelöscht oder durch einen Gruppenknoten ersetzt. Der gesamte oben geschilderte Vorgang wird im folgenden als **Expandierung** des SSI-Knotens bezeichnet.

Durch die Einführung des SSI-Knotens ist der Anwendungsentwickler in der Lage, beliebige Abbildungen zwischen Daten in der Datenbank und VRML-Komponenten zu definieren. Das Abbildungsspektrum ist äußerst vielseitig. Es können sehr einfache Abbildungen, wie z.B. das Einfügen von externen Daten in beliebige Knotenfelder definiert werden, es sind aber auch sehr komplexe Abbildungen möglich, wie z.B. die Generierung einer Teilszene, deren Aussehen und Anzahl von externen Daten bestimmt wird. Durch Abbildungen wie im letzten Fall können benutzerdefinierte logische Skalierungen definiert werden (s. Kapitel 4.2.1). Die Art der Skalierung hängt von Metadaten ab, die in der Datenbank gespeichert sind und von dem SSI-Knoten verwendet werden.

Ein weiterer Vorteil besteht in der Integration multimedialer Daten, die sowohl auf einem Fileserver als auch in einer Datenbank gespeichert sein können.

Die in Kapitel 4.2.1 beschriebene Persistenz wird durch den SSI-Knoten gewährleistet, falls die darzustellenden VRML-Szenen in einer Datenbank gespeichert sind. Alle Ver-

änderungen, die z.B. direkt innerhalb der Datenbank oder mit Hilfe externer Zugriffe vorgenommen wurden, werden bei dem erneuten Laden der VRML-Szene sichtbar.

5 Konzeption des SSI-Knotens

5.1 Ausgangspunkt

Die Spezifikation und Implementierung des SSI-Knotens basieren auf der Erweiterung eines bereits existierenden VRML-DataBlades, das am Institut für integrierte Publikations- und Informationssysteme in der GMD entwickelt wurde.

5.1.1 VRML-DataBlade

Das VRML-DataBlade ist ein Modul, das den IDS um Kernfunktionalitäten erweitert (s. Kapitel 2.2.2). Es ermöglicht ein strukturiertes Management von VRML-Szenen. Der Benutzer verfügt über Datentypen und Funktionen für die Verwaltung von VRML-Szenen, die er direkt auf der SQL-Ebene verwenden kann. Mit Hilfe des VRML-DataBlades kann er VRML-Szenen in der Datenbank zu speichern und darauf zugreifen, wie er es von herkömmlichen Datentypen kennt. Das VRML-DataBlade bietet darüber hinaus Funktionen für die Manipulation und Komposition von VRML-Szenen. Es ist unter anderem möglich, ein durch VRML beschriebenes 3D-Szenario aus verschiedenen in der Datenbank gespeicherten Teilszenen dynamisch zusammenzusetzen.

Die Implementierung des VRML-DataBlades fand in mehreren Schritten statt. Als erstes eine VRML-Klassenbibliothek entwickelt. Sie bildet das Fundament des VRML-DataBlades. Die VRML-Klassenbibliothek wird im Anschluß erläutert; genauere Informationen findet man in der GMD-Studie von Thomas Risse und Matthias Hemmje [RISSE97].

Die erste Version der VRML-Klassenbibliothek verwendete den Informix Illustra Server als DBMS. Der IDS ist eine Weiterentwicklung des Informix Illustra Servers. Der zweite Schritt bestand aus der Portierung der VRML-Klassenbibliothek auf den IDS. Zusätzlich wurde in diesem zweiten Schritt die SQL-Schnittstelle erweitert, die ebenfalls im Anschluß erläutert wird.

5.1.1.1 VRML-Klassenbibliothek

Die wichtigste Komponente des VRML-DataBlades ist die oben erwähnte in C++ implementierte VRML-Klassenbibliothek. Mit den in dieser Bibliothek definierten Klassen und Funktionen können VRML-Szenen entwickelt und bearbeitet werden.

Die Speicherung einer VRML-Szene in der Datenbank erfolgt durch ein CLOB, eine spezielle Form eines BLOBs, welcher ASCII-Texte "beliebiger" Länge speichern kann. In diesem CLOB liegt die VRML-Szene in ihrer "Reinform" vor, d. h. als VRML 2.0-Quelltext. Mit Hilfe eines *Distinct Data Types* (s. Kapitel 2.2.2) wurde für VRML-Szenen ein eigener Datentyp erzeugt:

```
create distinct type vrml as clob;
```

Die VRML-Klassenbibliothek ist objektorientiert aufgebaut, wodurch eine sehr elegante Datenkapselung erreicht wird. Alle Attribute eines Objektes können nur mit Hilfe spezieller Funktionen verändert und gelesen werden. Dies bedeutet, daß die Objekte nur über eine festgelegte Schnittstelle manipuliert werden können.

Die Datentypen und Klassen innerhalb der Bibliothek lassen sich in drei Gruppen einteilen:

- **Datentypen für Felder und Ereignisse**

  Jeder VRML-Knoten besteht aus Feldern und Ereignissen (s. Kapitel 3.3.3.3). Die Felder unterteilt man in Einzel- und Multiwertfelder. Für jeden einfachen Feldtypen, z.B. SFInt32, existiert ein entsprechender Datentyp in der Klassenbibliothek, der entweder mit Hilfe eines einfachen C++-Datentyps oder durch eine eigenständige Klasse erzeugt wurde. Die Multiwertfelder wie z.B. MFNode sind als Listen implementiert, auf deren Elemente man mit Hilfe von Iteratoren zugegreifen kann.

- **VRML-Knoten**

  Jeder VRML-Knoten wurde durch eine eigene Klasse implementiert. Die Eigenschaften des VRML-Knotens werden durch seine Felder und Ereignisse charakterisiert. Jede dieser Klassen besitzt Methoden, um Felder und Ereignisse zu lesen oder zu beschreiben.

- **Sonstige Klassen**

  Hierzu gehört die Parserklasse, die ein VRML-Quelltext parst und daraus einen Objektbaum erzeugt, sowie die Generatorklasse, die aus diesem Objektbaum wieder einen VRML-Quelltext generiert. Die Szenenklasse, repräsentiert den gesamten VRML-Szenengraphen in Form des zuvor genannten Objektbaumes. Diese drei Klassen werden im folgenden näher erläutert, da sie eine wichtige Rolle bei der Integration des SSI-Knotens spielen.

VRML-Parser

Der Parser wird immer dann eingesetzt, wenn eine VRML-Szenenbeschreibung in die interne Darstellung (Objektbaum) transformiert werden soll. Der Konstruktor der Parserklasse bekommt als Parameter entweder einen Zeiger auf eine VRML-Datei oder ein Handle (Referenz) auf ein CLOB mit. Er arbeitet objektorientiert und stellt Funktionen für das Einlesen von Wörtern und die Erzeugung von Knotenobjekten zur Verfügung.

Die VRML-Datei wird wortweise geparst. Sobald Knotennamen im VRML-Quelltext auftauchen, wird in einer Liste binär nach dem zugehörigen Klassenkonstruktor gesucht. Der interne Aufbau eines Knotens ist nur dem jeweiligen Knotenobjekt selbst bekannt. Die Werte für die einzelnen Felder des Knotens werden daher innerhalb des entsprechenden Knotenobjekts eingelesen, d.h. jeder Knoten parst sich selbst. Dabei wird eine Methode des Knotenobjekts mit einer Referenz des Parsers als Parameter aufgerufen.

VRML-Generator

Der Generator ist das Gegenstück zum Parser. Er wird immer dann eingesetzt, wenn aus einem Objektbaum eine VRML-Datei bzw. ein CLOB erzeugt werden soll. Er arbeitet wie der Parser vollständig objektorientiert und stellt Funktionen für das Schreiben in einen Datenstrom zur Verfügung. Der interne Aufbau eines Knotens ist ihm ebenfalls unbekannt. Alle Knotenklassen besitzen eine virtuelle Funktion, mit deren Hilfe aus einem Knotenobjekt eine VRML-Beschreibung in Textform erzeugt werden kann, d.h. jeder Knoten generiert sich selbst.

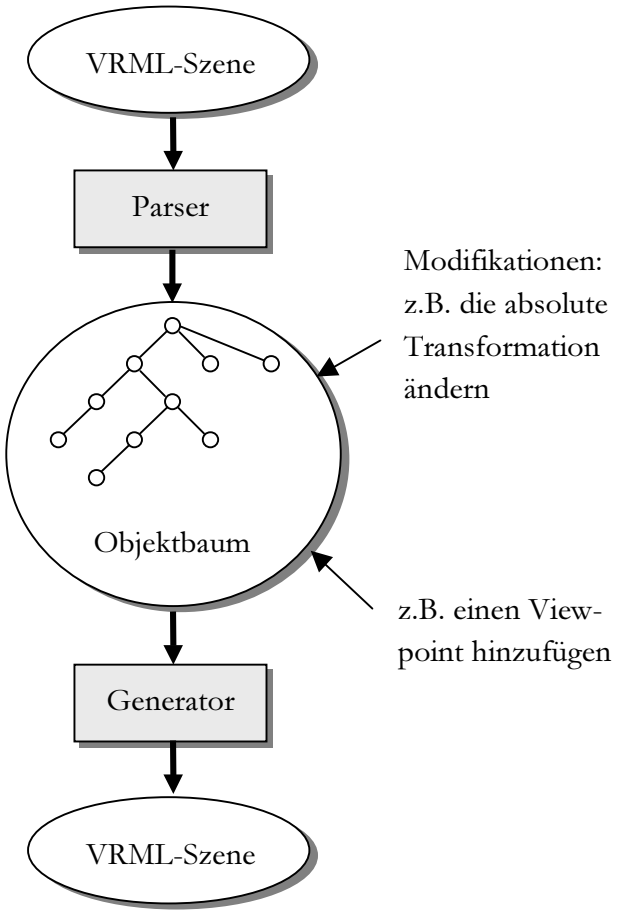
VRML-Szenenklasse

Ein Objekt dieser Klasse enthält den kompletten VRML-Szenengraphen in Form eines Objektbaumes. Er wird intern durch ein Objekt des Typs MFNode repräsentiert, d.h. als Liste, deren Elemente Wurzelknoten sind. Die Speicherung der Knoten erfolgt auf allen Ebenen des Objektbaumes nach demselben Prinzip. Damit wird eine rekursive Traversierung und Bearbeitung aller Knoten durch den Generator ermöglicht.

Mit dieser Klasse kann entweder ein VRML-Szenengraph neu erstellt werden, indem neue Knotenobjekte erzeugt und in die entsprechenden Listen eingehängt werden, oder man kann einen bestehenden Szenengraphen modifizieren, indem man durch den Objektbaum traversiert und entsprechende Modifikationen vornimmt.

Alle Funktionen der SQL-Schnittstelle des VRML-DataBlades (s. nächstes Kapitel) arbeiten nach diesem Prinzip. Die folgende Abbildung zeigt den Ablauf einer VRML-Szenenmodifikation.

ABBILDUNG 4: Ablauf einer VRML-Szenen-Modifikation



5.1.1.2 SQL-Schnittstelle

Das VRML-DataBlade bietet Funktionen an, die über die SQL-Schnittstelle angesprochen werden können. Sie ermöglichen den Zugriff, die Speicherung und die Modifikation von VRML-Szenen in der Datenbank. Es folgt eine kurze Übersicht der vorhandenen Funktionen:

Alle nachfolgenden Beispiele arbeiten mit folgender Tabelle:

```
create table Szenen
(
     ID           integer,
     VRMLWorld    vrml,
     primary key  (ID)
);
```

VRMLCreateWorld

Diese SQL-Funktion ermöglicht das Erzeugen oder Zusammenfügen von zwei VRML-Szenenbeschreibungen. Im folgenden Beispiel werden zwei Szenen miteinander verschmolzen und das Ergebnis in die oben gezeigte Tabelle eingefügt.

```
INSERT INTO Szenen (10, VRMLCreateWorld('select VRMLWorld from Szenen
where ID=4', 'select VRMLWorld from Szenen where ID=5'));
```

FileToVRML, VRMLToFile

Diese beiden SQL-Funktionen dienen dem Im- und Export von VRML-Szenen in und aus der Datenbank. Das folgende Beispiel importiert eine VRML-Szenenbeschreibung, die in der Datei `messe1.wrl` gespeichert ist:

```
INSERT INTO Szenen VALUES (11, FileToVRML('/home/messe1.wrl'));
```

VRMLSetTranslation, VRMLTranslation

Diese SQL-Funktionen ermöglichen das Setzen einer absoluten oder relativen Translation für eine VRML-Szene. Das folgende Beispiel setzt bei allen VRML-Szenen, deren ID kleiner als 4 ist, die absolute Translation auf die Werte X=10, Y=-5 und Z=7:

```
UPDATE Szenen SET vrml=VRMLSetTranslation( vrml, 10, -5, 7)) where
ID<4;
```

VRMLSetRotation, VRMLRotation

Mit diesen SQL-Funktionen kann man die absolute oder relative Rotation für eine VRML-Szene setzen.

VRMLSetScale, VRMLScale

Diese SQL-Funktionen setzen die absolute oder relative Skalierung einer VRML-Szene fest.

VRMLAddViewPoint

Mit dieser SQL-Funktion kann man zu einer bestehenden VRML-Szene einen neuen fest definierten Aussichtspunkt (engl. viewpoint) hinzufügen. Alle für eine VRML-Szene definierten Aussichtspunkte werden in einer speziellen Liste dargestellt, die man in dem VRML-Browser aufrufen kann. Sie können mit Hilfe der Maus ausgewählt werden, und die Szenen-Darstellung wird dann entsprechend dynamisch verändert.

Die oben angeführten Funktionen wurden unter Verwendung der in der VRML-Klassenbibliothek zur Verfügung stehenden Klassen und Funktionen implementiert. Je

nach Anwendungsprogramm kann man neue Funktionen dieser Art mit Hilfe der DataBlade-API und der Schnittstelle zur VRML-Bibliothek entwickeln. Sie werden dem IDS durch eine Registrierung als UDR-Funktion bekannt gemacht (s. Kapitel 2.2.2).

5.1.1.3 Fehlende Funktionalitäten

In der aktuellen Version der VRML-Klassenbibliothek fehlt die Umsetzung von zwei Konzepten der VRML 2.0 Spezifikation:

Es werden zur Zeit noch keine Prototypen und Skriptknoten (s. Kapitel 3.3.3.5) unterstützt. Die Erweiterung der VRML-Klassenbibliothek um diese Komponenten ist nicht Bestandteil dieser Arbeit. Die Spezifikation und Implementierung des SSI-Knotens berücksichtigt das Fehlen dieser Komponenten. Es wird jeweils an den entsprechenden Stellen darauf hingewiesen, welche Veränderungen vorgenommen werden müssen, sobald diese fehlenden Erweiterungen implementiert worden sind.

5.2 Anforderungen

Der SSI-Knoten wird in die oben beschriebene VRML-Klassenbibliothek (s. a. Kapitel 5.1.1.1) integriert. Die bereits in [RISSE97] erläuterten Anforderungen und Erwartungen an diese Klassenbibliothek sollen nach Möglichkeit weiterhin komplett erfüllt werden, da es sich hier lediglich um eine Erweiterung und nicht um eine Neukonzeption handelt.

Der Anwender soll wie bisher von der Technik und jeglichem Hintergrundwissen zur Benutzung der Funktionen des VRML-DataBlades verschont bleiben. Für ihn sind vielmehr Stabilität und Ausführungsgeschwindigkeit der Anwendung wichtig.

Der Anwendungsentwickler erwartet folgende Eigenschaften von der VRML-Klassenbibliothek, die die Spezifikation des SSI-Knotens ebenfalls erfüllen muß:

- Die Freiheiten, die VRML 2.0 bei der Entwicklung einer dreidimensionalen Anwendung bietet, sollen in keiner Weise eingeschränkt werden.
- Die Erweiterung muß sich an den VRML 2.0-Standard halten, d.h. es dürfen keine neuen Strukturen für den VRML-Szenengraphen eingeführt werden.
- Neu eingeführte Klassen müßen flexible Konstruktoren aufweisen, die neben dem Setzen von Standardwerten auch die Möglichkeit bieten, häufig geänderte Attribute als Parameter mit anzugeben.
- Jede Klasse muß eine vollständige Menge an Methoden aufweisen, die das Setzen und die Abfrage aller relevanten Werte eines Objekts ermöglichen.

- Syntaktische Fehler des Anwenders sollen durch eine vollständige Typüberprüfung vermieden werden.
- Die in der SQL-Schnittstelle neu eingeführten Funktionen sollen konform und orthogonal mit den bisherigen Funktionen sein.

Durch die Integration von externen (multimedialen) Daten in eine VRML-Szene, die mit Hilfe des SSI-Knotens ermöglicht wird, ergeben sich neue zusätzliche Anforderungen:

- Im Falle einer fehlerhaften Expandierung des SSI-Knotens, die vom Anwender verursacht wurde, muß eine hinreichende Rückmeldung erfolgen, damit der Fehler behoben werden kann.
- Die Persistenz von Modifikationen innerhalb einer in der Datenbank gespeicherten VRML-Szene durch den Anwender muß gewährleistet sein: Vorgenommene Veränderungen müssen beim erneuten Anfordern einer VRML-Szene sichtbar sein.
- Die Lokalität des SSI-Knotens in der VRML-Datei darf keinen Einschränkungen unterliegen, d.h. er wird wie alle anderen VRML-Knoten behandelt.
- Der SSI-Knoten soll Metadaten-Management unterstützen. Damit kann man VRML-Szenen logisch partionieren. Durch die Verwendung von Metadaten werden verschiedene logische Sichten auf VRML-Szenen ermöglicht, z.B. interessiert sich ein Elektriker bei der virtuellen Darstellung einer Büroetage lediglich für die Visualisierung elektronischer Leitungen und Anschlüsse in den Wänden.

5.3 Architektur

Die interne Architektur des VRML-DataBlades ist im Rahmen der Beschreibung der VRML-Klassenbibliothek in [RISSE97] bereits ausführlich erläutert worden. Für das Verständnis der Konzeption des SSI-Knotens ist die Darstellung der umgebenden Architektur, in der sich das VRML-DataBlade befindet, weitaus wichtiger. Die folgende Abbildung zeigt den Aufbau der Architektur bei Anwendungen, die das VRML-DataBlade und den darin integrierten SSI-Knoten benutzen.

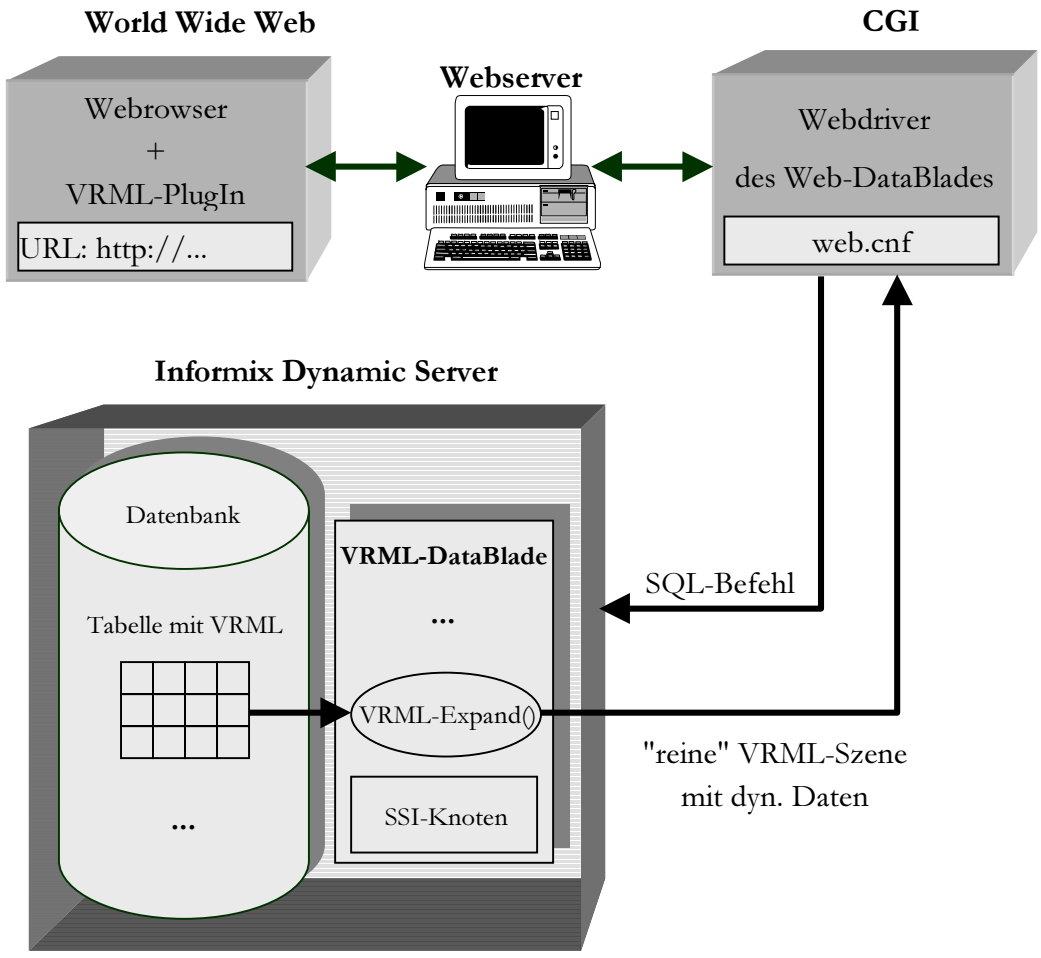


ABBILDUNG 5: Architektur einer VRML-DataBlade-Anwendung

5.3.1 World Wide Web

Der Anwender greift über das WWW auf die in der Datenbank gespeicherten VRML-Szenen zu. Er benötigt einen Browser, der VRML 2.0 unterstützt. Diese Unterstützung wird bei den meisten Standardbrowsern wie z.B. dem "Netscape Communicator" oder dem "Microsoft Internet Explorer" durch spezielle Erweiterungen (Plug-ins) erreicht. Die bekanntesten VRML-Plug-ins sind der "CosmoPlayer" von *Silicon Graphics Inc.* und der "WorldView" von *Intervista.*

Man sollte bei der Entwicklung von 3D-Welten darauf achten, daß dem potentiellen Kunden nur eine begrenzte Rechenleistung zur Verfügung steht. VRML-Szenen sind in der Berechnung der Darstellung erheblich aufwendiger als z.B. zweidimensionale HTML-Seiten. Bei zu komplexen VRML-Szenen ist keine flüssige Navigation mehr möglich, und der Kunde verliert sehr schnell die Lust sie zu betrachten. Den gleichen Effekt kann man auch bei zu langen Übertragungszeiten für HTML-Seiten beobachten, die mit speicherintensiven Bildern oder Java-Applets überfrachtet sind.

5.3.2 Webserver

Die Schnittstelle zwischen dem WWW und dem Datenbankserver ist der Webserver. Er leitet alle Zugriffe auf VRML-Szenen zu der CGI-Schnittstelle um, welche durch den Webdriver des Web-DataBlades (s.u.) repräsentiert wird. Es können beliebige Webserver wie. z.B. der Webserver von *Netscape*, der Internetserver von *Microsoft* oder der Apache-Webserver eingesetzt werden.

5.3.3 Web-DataBlade/Webdriver

Der Webdriver ist ein Bestandteil des Web-DataBlades von *Informix* und verbindet den Webserver mit dem IDS. Durch den Einsatz des Web-DataBlades kann ein Benutzer im WWW in der Datenbank gespeicherte HTML-Seiten anfordern. Dies geschieht durch eine in der URL kodierte Datenbankabfrage, die von einem CGI-Programm (hier der Webdriver) in eine gültige SQL-Anfrage umgewandelt wird. Das folgende Beispiel zeigt die Umsetzung einer URL in eine SQL-Anfrage, die von dem Webdriver durchgeführt wird:

```
http://myhost:port/cgi-bin/webdriver.cgi/?MivalObj=my_image
?MItypeObj=image/jpeg
                                    ↓

SELECT object FROM webImages WHERE ID='my_image';
```

Wie man in der Umsetzung sieht, müssen nicht immer alle Werte für die SQL-Anfrage in der URL angegeben werden, da die meisten Werte bereits in einer Konfigurationsdatei (hier das `web.cnf`) eingetragen sind. In dem oben gezeigten Beispiel ist der Name der Tabelle (`webImages`) und der Name der Spalte (`object`) bereits vorgegeben.

Die aus der URL generierte SQL-Anfrage wird vom IDS ausgeführt, und die Ergebnisse werden wiederum an den Webdriver geschickt. Der Webdriver leitet die Antwort an den Webserver, der diese an den anfragenden Benutzer zurücksendet.

Damit der Webdriver weiß, in welchem Format er die Antwort generieren soll, wird der MIME-Typ mitgeteilt. Der MIME-Typ spezifiziert die Art der gesendeten Daten. Im

oben gezeigten Beispiel bedeutet die Angabe `image/jpeg`, daß es sich bei der generierten Antwort um ein Bild im JPEG-Format handelt.

Eine weitere Eigenschaft des Web-DataBlades ermöglicht die Einbettung externer Daten in HTML-Seiten, d.h. es bietet im Prinzip eine ähnliche Funktionalität, wie der SSI-Knoten im VRML-DataBlade.

Die Einbettung der externen Daten erfolgt durch eigene Tags. Das folgende Beispiel zeigt die Integration von Daten in eine HTML-Seite. Die Daten werden mit Hilfe einer SQL-Anweisung ermittelt: Es werden alle in der Tabelle `employee` enthaltenen Namen und Gehälter von Angestellten in die entsprechenden Platzhalter `$1` und `$2`, die sich innerhalb einer HTML-Tabelle befinden, eingesetzt. Die HTML-Zeilen zwischen den <MISQL>-Tags werden entsprechend der Anzahl dupliziert:

```
...
<BR><TABLE>
<?MISQL SQL="select name, salary from employee;">
<TR><TD><B>Name:</B></TD><TD>$1</TD>
<TD><B>Salary:</B></TD><TD>$2</TD></TR>
<?/MISQL>
</TABLE>
...
```

Dieser Mechanismus wird benutzt, um den Zugriff von außen (WWW) auf in der Datenbank gespeicherte VRML-Szenen zu ermöglichen. Es bieten sich je nach Art der Anwendung zwei verschiedene Möglichkeiten an:

Die erste Variante besteht darin, VRML-Szenen in eine HTML-Szene einzubetten, d.h. im Browser erscheint eine HTML-Seite, in der an einer bestimmten Stelle eine VRML-Szene eingeblendet ist. Das folgende Beispiel integriert in eine HTML-Seite die in der Tabelle `vrml_scenes` gespeicherte VRML-Szene `test`:

```
...
<?MISQL SQL="select scene from vrml_scenes where ID='test';">
<embed src="$WEB_HOME?LO=$1&MItypeObj=model/vrml" align="baseline"
border="0" width="100%" height="75%">
<?/MISQL>\
...
```

Die VRML-Szene ist wie alle anderen multimedialen Daten als ein Large Object in der Datenbank gespeichert (s. a. Kapitel 5.4.4.2). Der Zugriff auf die VRML-Szene erfolgt in dieser Variante über die Angabe eines Handles, welches nach der Ausführung der SELECT-Anweisung im Platzhalter `$1` eingetragen wird. Ein Handle wird vornehmlich verwendet bei der dynamischen Erzeugung einer darzustellenden Liste, die multimediale Daten enthält.

Die Übertragung von HTML-Seiten im Internet erfolgt mit Hilfe des HTTPs (Hypertext Transfer Protocol). Das Web-DataBlade ist in der Lage, den Kopf dieses Proto-

kolls, der Informationen z.B. über den Inhalt eines Dokumentes enthält, zu beeinflussen.

In der zweiten Variante kann ein Dokument erstellt werden, dessen Inhalt eine VRML-Szene ist. Dazu muß das Feld `content-type` des HTTPs auf den Wert `x-model/x-vrml` gesetzt werden. Das folgende Beispiel erzeugt ein "reines" VRML-Dokument, welches die gleiche VRML-Szene enthält, wie die HTML-Seite im vorherigen Beispiel:

```
<?MIVAR>$(HTTPHEADER,content-type,x-world/x-vrml)<?/MIVAR>
<?MISQL SQL="select scene::clob from vrml_scenes where
ID='test';">$1<?/MISQL>
```

VRML-Driver

Der oben besprochene Webdriver ließe sich auch durch ein CGI-Programm ersetzen, welches im folgenden als "VRML-Driver" bezeichnet wird. Dieser VRML-Driver kann als eine Komponente des VRML-DataBlades implementiert werden. Damit wäre ein autonomer Einsatz dieses DataBlades möglich.

Bei den meisten Anwendungen werden jedoch neben der Darstellung von VRML-Szenen auch zusätzliche HTML-Seiten benötigt. Falls die HTML-Seiten ebenfalls in der Datenbank verwaltet werden, ist es ist eleganter, den Webdriver für den Zugriff auf beide Formate zu benutzen. Er erfüllt durch die oben gezeigten Varianten nahezu alle Funktionalitäten eines eigenständigen VRML-Drivers, deshalb wurde in dieser Arbeit auf die Entwicklung eines solchen CGI-Programms verzichtet.

5.3.4 VRMLExpand-Funktion

Die VRMLExpand-Funktion ist ein Bestandteil des VRML-DataBlades und erzeugt dynamisch generierte VRML-Szenen (s. a. Kapitel 6.3). Das Aussehen der VRML-Szenen wird von den Daten bestimmt, die in der IDS-Datenbank gespeichert sind. Sie parst die angeforderte VRML-Szene und expandiert alle eingebetteten SSI-Knoten. Die resultierende "reine" VRML-Szene, die nun dynamisch integrierte (multimediale) Daten enthält, wird an die entsprechende Client-Applikation - in der Regel ein CGI-Programm (hier der Webdriver) - geliefert.

5.4 Spezifikation des SSI-Knotens

Unter Berücksichtigung der oben angeführten Anforderungen wird der SSI-Knoten nun spezifiziert. Es folgt eine genaue Beschreibung der Syntax und der Funktionalität, wobei letztere mit Hilfe von Abbildungen und Beispielen erklärt wird.

5.4.1 Syntax

Der SSI-Knoten wird durch die PROTO-Anweisung definiert (s. Kapitel 3.3.3.5). Da das VRML-DataBlade Prototypen zur Zeit noch nicht unterstützt (s. Kapitel 5.1.1.3), entfällt diese Anforderung. Es wird jedoch an dieser Stelle der Vollständigkeit halber die Definition des SSI-Knotens unter Verwendung eines PROTOs angegeben:

```
PROTO SQLServerInclude [
  field SFString   connection ""
  field SFInt32    maxRow -1
  field SFString   sqlStatement ""
  field SFNode     template NULL
  field SFInt32    error_no 0
  field SFString   error_message ""
]
{ }
```

Die fehlende Unterstützung von Prototypen schränkt die Funktionalität des SSI-Knotens nicht ein. Er taucht auf der Client-Seite nicht auf, da die Expandierung aller SSI-Knoten innerhalb des IDS vorgenommen wird. Lediglich im Falle eines Fehlers, z.B. bei der Angabe einer ungültigen SELECT-Anweisung im Feld `sqlStatement`, erscheint er in der expandierten VRML-Szenenbeschreibung (s. a. Kapitel 5.4.5). Der betroffenene VRML-Browser meldet in einem solchen Fall, daß eine ungültige VRML-Syntax vorliegt. Das ist für Anwendungsentwickler vorteilhaft, da sie damit über einen Indikator für die fehlerhafte Expandierung eines SSI-Knotens verfügen.

Wenn die VRML-Klassenbibliothek um den PROTO-Mechanismus erweitert wurde, kann der SSI-Knoten wie oben gezeigt in der VRML-Szene definiert werden.

Es wird nun die Bedeutung der einzelnen SSI-Knotenfelder erklärt:

connection

Im Feld `connection` des Typs SFString stehen Informationen, die eine bestimmte Datenbankverbindung definieren. Dieses Feld dient zukünftigen Erweiterungen des VRML-DataBlades und repräsentiert in dieser Version lediglich ein Provisorium. Es besteht aus einer Zeichenkette, die eine JDBC-ähnliche Syntax enthalten soll, die man für den Aufbau einer Datenbankverbindung benötigt. Die Informationen für eine solche Verbindung können abhängig von der benutzten Form eines DBMS variieren. Eine mögliche Variante könnte folgendermaßen aussehen:

```
connection "DatabaseName@ServerName:LoginName.Password"
```

Man soll später die Möglichkeit haben auf verschiedene Datenbanken zuzugreifen, die sich auf unterschiedlichen Datenbankservern befinden können. Dies geschieht durch Angabe der Datenbank, des Datenbankservers, des Benutzernamens und des Paßwortes.

In der jetzigen Version wird lediglich eine bestimmte hardcodierte Datenbankverbindung (zu dem IDS) verwendet, so daß die Angabe einer Verbindung in diesem Feld vorerst entfällt.

- **maxRow**

Mit dem Feld `maxRow` des Typs SFInt32 kann man durch Angabe eines ganzzahligen Wertes spezifizieren, wie viele Ergebnistupel nach der Ausführung der SELECT-Anweisung im Feld `sqlStatement` berücksichtigt werden. Falls `maxRow` den voreingestellten Wert –1 besitzt, werden alle Ergebnistupel berücksichtigt. Es werden nur die ersten n-ten Ergebnistupel ausgewertet, wenn n der Wert von `maxRow` ist.

Dieses Feld dient der Visualisierung, d.h. hat man in der VRML-Szene für die Darstellung der Ergebnisse nur einen bestimmten Bereich zur Verfügung, so kann man an dieser Stelle die in der Regel unvorhergesehene Ergebnisgröße einer SELECT-Anweisung einschränken.

- **sqlStatement**:

In dem Feld `sqlStatement` des Typs SFString wird eine SELECT-Anweisung angegeben, die innerhalb des IDS ausgeführt wird. Die Ergebnisse der Anfrage werden in die Felder anderer VRML-Knoten abgebildet (s. a. Kapitel 5.4.2) oder für die Instanziierung des Templates benötigt (s. a. Kapitel 5.4.3).

Die SELECT-Anweisung ist bis auf zwei kleine Erweiterungen, die die Abbildung der SQL-Ergebnisse auf die Knotenfelder und umgekehrt definieren, konform der SQL92-Spezifikation. Die abstrahierte Syntax der SELECT-Anweisung entspricht folgendem Muster:

```
SELECT [SPALTENNAME]+ INTO [KNOTENNAME.FELDNAME]+ FROM [TABELLENNAME]
WHERE [BEDINGUNG]
```

Die erste Erweiterung der SELECT-Anweisung besteht aus der Einführung des INTO-Konstrukts, mit dem man angeben kann, in welche Knotenfelder die Ergebniswerte der Abfrage abgebildet werden sollen. Die Anzahl der selektierten Spalten muß mit der Anzahl der angegebenen Ziel-Knotenfelder übereinstimmen, wenn nur einfache Feldtypen das Ziel der Abbildung sind. Zu den einfachen Feldtypen gehören alle Felder, deren Werte elementar sind wie z.B. SFInt32 oder MFFloat. Die Abbildung wird durch die Reihenfolge der Spalten bzw. der Ziel-Knotenfelder festgelegt. Ein Ziel-Knotenfeld wird durch Angabe des Knoten- sowie des Feldnamens, die durch einen Punkt getrennt sein müssen, bestimmt:

```
SELECT Vorname, Nachname, Gehalt INTO Knoten1.string, Knoten2.string,
Knoten3.string FROM Angestellte WHERE Gehalt>10000
```

In der oben gezeigten Anweisung werden unter anderem alle selektierten Werte aus der Spalte `Nachname` in in das Feld `string` des Knotens `Knoten2` abgebildet.

Falls ein komplexer VRML-Feldtyp wie SFVec2F oder MFVec3F das Ziel einer Abbildung ist, muß die Anzahl der selektierten Spalten mit der Anzahl aller Elemente der angegebenen Zielknotenfelder übereinstimmen. Im folgenden Beispiel werden die Ergebnisse der Selektion in ein Feld vom Typ SFVec3F geschrieben. Der Typ SFVec3F repräsentiert drei verschiedene Koordinaten:

```
SELECT Xpos, Ypos, Zpos INTO Knoten1.translation FROM Objektposition
WHERE ID='STUHL2'
```

In der oben gezeigten Anweisung wird unter anderem der selektierte Wert aus der Spalte `Ypos` in die Y-Wert-Komponente des Feldes `translation` des Knotens `Knoten1` abgebildet.

Im WHERE-Konstrukt befindet sich eine Bedingung, in der auch Platzhalter für Feldwerte anderer VRML-Knoten verwendet werden können. Dies ist die zweite Erweiterung, die oben erwähnt wurde. Vor der Ausführung der Selektion werden die Platzhalter durch den jeweiligen referenzierten Feldwert ersetzt. Die Syntax der Platzhalter wird wie im INTO-Konstrukt durch Angabe eines Knoten- und eines Feldnamens, die durch einen Punkt getrennt sein müssen, bestimmt:

```
SELECT Xpos, Ypos, Zpos INTO Knoten1.translation FROM Objektposition
WHERE ID=Knoten2.string
```

Die erlaubten Abbildungen zwischen VRML-Feldtypen und SQL-Datentypen und umgekehrt werden im Kapitel 5.4.1.1 spezifizert.

- **template**

Der SSI-Knoten besitzt neben der Möglichkeit SQL-Ergebnisse in Felder anderer VRML-Knoten abzubilden, noch eine zweite Anwendungsform:

In dem Feld `template` kann eine VRML-Teilszene beliebiger Komplexität angegeben werden, die später abhängig von der Anzahl der Ergebnistupel (mehrfach) instanziiert wird. Genauer gesagt ist die Anzahl der Instanziierungen genauso groß wie die Anzahl der Ergebnistupel der SELECT-Anweisung. Die Ergebniswerte der SQL-Anfrage können wie in der ersten Anwendungsform auf Feldwerte von VRML-Knoten, die **in** dem Template enthalten sind, abgebildet werden. Falls das Feld `template` den voreingestellten Wert NULL enthält, finden keine Instanziierungen statt, d.h. hiermit wird eingestellt, welche der beiden Anwendungsformen verwendet werden soll. Genauere Erklärungen und Beipiele zu dem Templatemechanismus befinden sich in Kapitel 5.4.3.

- **error_no**

In dem Feld `error_no` des Typs SFInt32 wird vom System eine Fehlerkonstante angegeben, falls die Expandierung des SSI-Knotens nicht durchgeführt werden konnte. Dieses Feld dient lediglich der Ausgabe von Fehlerinformationen des IDS und wird in

der nicht expandierten VRML-Datei mit keinen Werten belegt. Genauere Informationen zur Fehlerausgabe werden im Kapitel 5.4.5 erläutert

- **error_message**

Im Feld `error_message` des Typs SFString wird vom System eine Fehlermeldung gegeben, die den aufgetretenen Fehler näher beschreibt. Dieses Feld dient lediglich der Ausgabe von Fehlerinformationen des IDS und wird in der nicht expandierten VRML-Datei mit keinen Werten belegt. Genauere Informationen zur Fehlerausgabe werden im Kapitel 5.4.5 erläutert

5.4.1.1 Abbildung von Datentypen

Die folgende Tabelle definiert die erlaubten Abbildungen der einfachen SQL-Datentypen auf VRML-Feldtypen, die im INTO-Konstrukt angegeben werden. Eine Zusammenfassung und genaue Erklärung der SQL-Datentypen befindet sich in [SQLREF97]:

SQL-Datentyp	VRML-Feldtyp
BOOLEAN	SFBool
CHAR	SFString, MFString
CHARACTER	SFString, MFString
DATE	SFString, MFString
DATETIME	SFString, MFString
DEC	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
DECIMAL	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
DOUBLE PRECISION	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
FLOAT	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F

SQL-Datentyp	VRML-Feldtyp
INT	SFColor*, MFColor*, SFFloat, MFFloat, SFInt32, MFInt32, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
INTEGER	SFColor*, MFColor*, SFFloat, MFFloat, SFInt32, MFInt32, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
INTERVAL	SFString, MFString
MONEY	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
NCHAR	SFString, MFString
NUMERIC	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
NVARCHAR	SFString, MFString
REAL	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
SMALLFLOAT	SFColor*, MFColor*, SFFloat, MFFloat, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
SMALLINT	SFColor*, MFColor*, SFFloat, MFFloat, SFInt32, MFInt32, SFRotation**, MFRotation**, SFString, MFString, SFTime***, MFTime***, SFVec2F, MFVec2f, SFVec3F, MFVec3F
TEXT	SFString, MFString
VARCHAR	SFString, MFString

* Nur Werte n mit $0 \leq n \leq 1$ erlaubt.

** Bei 4. Element nur Werte n mit $0 \leq n \leq 2\pi$ erlaubt.

*** Nur Werte n mit $0 \leq n$ erlaubt.

Beispiel: Wird in der SELECT-Anweisung eine Spalte vom Typ VARCHAR selektiert, so kann das Ergebnis nur in Knotenfelder vom Typ SFString oder MFString geschrieben werden, die im INTO-Konstrukt angegeben werden.

Die nächste Tabelle definiert die erlaubten Abbildungen der VRML-Feldtypen auf einfache SQL-Datentypen, die als Platzhalter im WHERE-Konstrukt angegeben werden können:

VRML-Feldtyp	SQL-Datentyp
SFBool	BOOLEAN
SFInt32	INT, INTEGER, SMALLINT
SFFloat	DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NUMERIC, REAL, SMALLFLOAT, SMALLINT
SFString	CHAR, CHARACTER, DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, NCHAR, NUMERIC, NVARCHAR, REAL, SMALLFLOAT, SMALLINT, VARCHAR

Beispiel: Wird in dem WHERE-Konstrukt als Platzhalter ein Feldwert vom Typ SFInt32 verwendet, so kann er nur als INT, INTEGER oder SMALLINT Typ verwendet werden.

5.4.2 Abbildungsmechanismus

Der Abbildungsmechanismus ermöglicht die Projektion von Daten (wie Zeichenketten oder Zahlenwerte) in die VRML-Szene. Diese Daten werden mit Hilfe der SELECT-Anweisung, die im Feld `sqlStatement` steht, aus der Datenbank gewonnen. Der Mechanismus läßt sich anhand eines Beispiels am besten erklären:

Es soll der Wert für das Feld `string` eines Textknotens und die Position des Textes im dreidimensionalen Raum (das Feld `translation` des Transformknotens) durch eine SELECT-Anweisung bestimmt werden. Die in der Datenbank gespeicherte und noch nicht expandierte VRML-Szene sieht folgendermaßen aus:

```
...
SQLServerInclude {
      connection ""
      maxRow -1
      sqlStatement "SELECT mieter, Xpos, Ypos, Zpos INTO
                    Firma.string, Position.translation FROM messe
```

```
                    where standNr='A1';"
      template NULL
}

DEF Position Transform {
      bboxCenter 0 0 0
      bboxSize   -1 -1 -1
      children   Shape {
            appearance Appearance {
                  material Material {
                        ambientIntensity 1
                        diffuseColor     1 1 1
                  }
            }
            geometry DEF Firma Text {
                        string []
                        fontStyle FontStyle {
                        horizontal  TRUE
                        justify     "MIDDLE"
                        style       "BOLD"
                  }
            }
      }
      center           0 0 0
      rotation         0 0 1 0
      scale            0.4 0.4 0.4
      scaleOrientation 0 0 1 0
      translation      0 0 0
}
...
```

Bei der Expandierung der Szene wird die SELECT-Anweisung evaluiert, d.h sie wird in eine SQL92-konforme SELECT-Anweisung umgewandelt und ausgeführt. Die evaluierte SELECT-Anweisung für das oben gezeigte Beispiel sieht dann folgendermaßen aus:

```
SELECT mieter, Xpos, Ypos, Zpos FROM messe where standNr='A1';
```

Das Feld `template` ist NULL, d.h. es handelt sich um den Abbildungsmechanismus. Die Ergebnisse werden in die mit grauen Hintergrund markierten Felder eingetragen und der SSI-Knoten wird aus der VRML-Szene gelöscht. Das Ergebnis der Expandierung sieht folgendermaßen aus:

```
...
DEF Position Transform {
     bboxCenter 0 0 0
     bboxSize   -1 -1 -1
     children   Shape {
      appearance Appearance {
            material Material {
                  ambientIntensity 1
                  diffuseColor     1 1 1
            }
      }
      geometry DEF Firma Text {
            string      "Informix"
            fontStyle   FontStyle {
                  horizontal  TRUE
                  justify     "MIDDLE"
                  style       "BOLD"
                  }
            }
     }
     center           0 0 0
     rotation         0 0 1 0
     scale            0.4 0.4 0.4
     scaleOrientation 0 0 1 0
     translation      0 2 0.5
}
...
```

In dem oben gezeigten Beispiel wird lediglich ein Tupel als Ergebnis erzeugt. Falls die SELECT-Anweisung mehrere Ergebnistupel liefert, z.B. durch das Weglassen des WHERE-Konstrukts, werden bei Einzelwertfeldern nur das erste Ergebnistupel und bei Multiwertfeldern alle Ergebnistupel berücksichtigt. Die expandierte VRML-Szene sieht folgendermaßen aus, falls drei Ergebnistupel erzeugt wurden:

```
...
DEF Position Transform {
      bboxCenter 0 0 0
      bboxSize   -1 -1 -1
      children   Shape {
            appearance Appearance {
                  material Material {
                        ambientIntensity 1
                        diffuseColor     1 1 1
                  }
            }
            geometry DEF Firma Text {
                  string      [ "Informix" "Oracle" "GMD-IPSI" ]
                  fontStyle   FontStyle {
                        horizontal  TRUE
                        justify     "MIDDLE"
                        style       "BOLD"
                  }
            }
      }
```

```
      center            0 0 0
      rotation          0 0 1 0
      scale             0.4 0.4 0.4
      scaleOrientation  0 0 1 0
      translation       0 2 0.5
}
...
```

Das Feld `string` im Textknoten ist vom Typ MFString (Multiwertfeld) und enthält somit die Werte aller Ergebnistupel. Das Feld `translation` im Transformknoten ist jedoch vom Typ SFVec3f (Einzelwertfeld) und enthält daher nur die Werte des ersten Ergebnistupels.

Die Positionierung des SSI-Knotens in der VRML-Datei unterliegt keinen Beschränkungen, d.h. er kann überall eingesetzt werden, wo VRML-Knoten erlaubt sind. Es gibt drei Möglichkeiten:

- Als Wurzelknoten, d.h. oberster Knoten im Szenengraphen.
- Als Kindknoten in einem MFNode-Feld, z.B. als Element des Feldes `children` im Gruppenknoten.
- Als Wert eines SFNode-Feldes.

Beim Abbildungsmechanismus wird der SSI-Knoten nach erfolgreicher Expandierung in den ersten beiden Fällen gelöscht und im dritten Fall durch den Wert NULL ersetzt.

5.4.3 Templatemechanismus

Der Templatemechanismus ermöglicht die mehrfache Instanziierung eines vorgegebenen beliebig komplexen VRML-Knotenbaums, der im Feld `template` steht. Die Anzahl der Instanziierungen richtet sich nach der Größe der Ergebnistupelmenge der SELECT-Anweisung, die sich im Feld `sqlStatement` befindet. Man hat dadurch die Möglichkeit, verschiedene Teilszenen beliebiger Größe dynamisch in die gerade angeforderte VRML-Datei einzubetten.

Besteht das Ergebnis der SELECT-Anweisung z.B. aus fünf Ergebnistupel und gibt es keine Einschränkungen auf die Anzahl der Ergebnisse im Feld `maxRow`, so wird der VRML-Knotenbaum entsprechend fünfmal instanziiert. Die nachfolgende Abbildung soll den Ablauf des Templatemechanismus verdeutlichen:

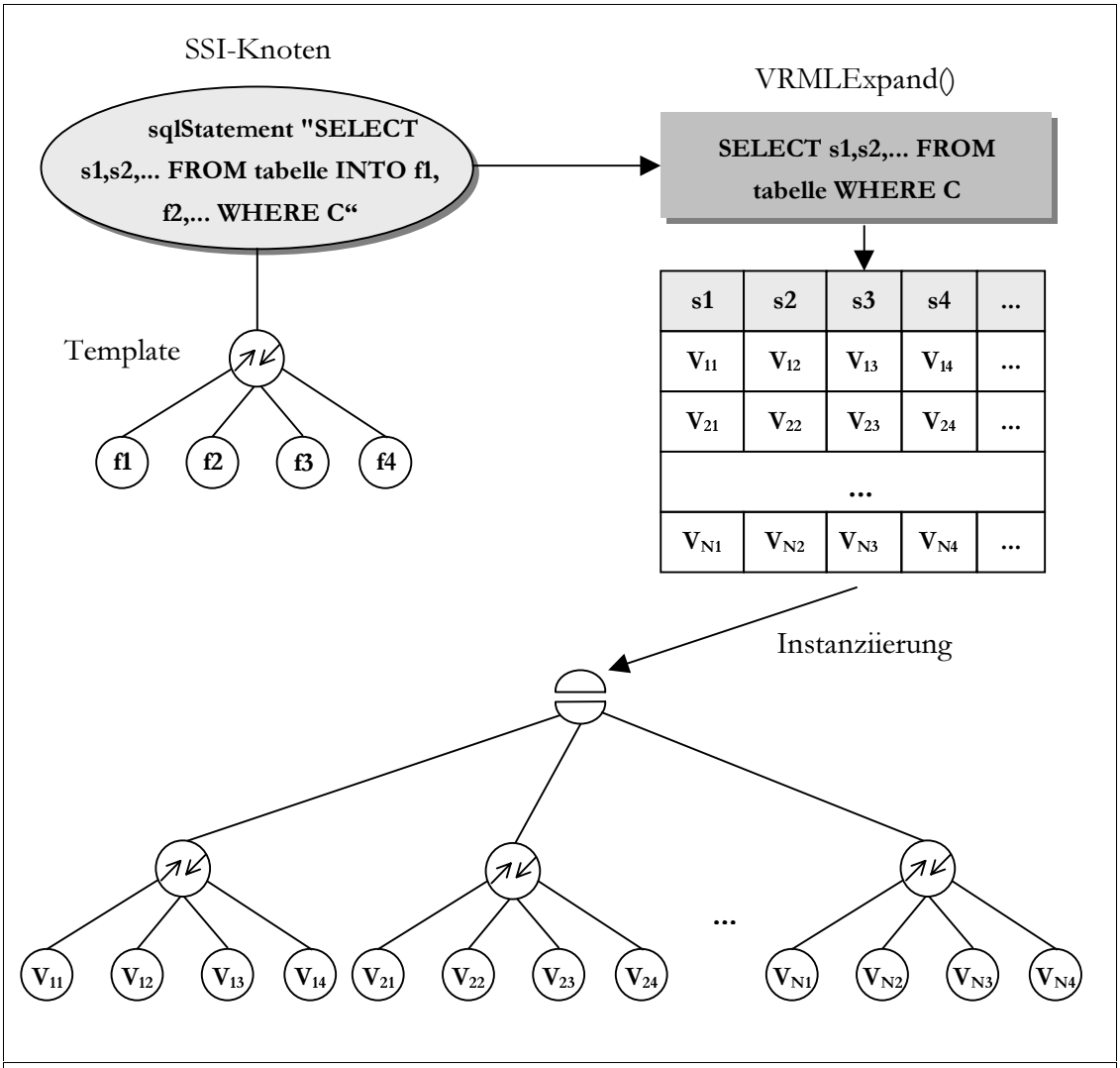


ABBILDUNG 6:
Ablauf des Templatemechanismus

Nach der reinen Instanziierung eines Knotenbaums kommt der bereits erläuterte Abbildungsmechanismus hinzu. In der oben gezeigten Abbildung besteht das Template aus fünf Knoten, von denen vier Knoten (f1–f4) jeweils Daten für ein bestimmtes Feld zugewiesen bekommen sollen. Die VRMLExpand-Funktion evaluiert die in dem Feld `sqlStatement` angebene SELECT-Anweisung und führt sie aus.

Die Ergebnisse der selektierten Spalten (s1-s4) werden zeilenweise jeder Instanziierung zugeordnet, d.h. bei n Ergebnissen werden n Instanziierungen vorgenommen und die n-te Instanziierung erhält die Werte des n-ten Ergebnistupels.

Die VRMLExpand-Funktion ersetzt danach den SSI-Knoten durch einen Gruppenknoten, in dessen Feld `children` die Instanziierungen angehängt werden.

Das folgende Beispiel instanziiert eine Teilszene, die aus einem Transformknoten besteht, der als Kind einen Textknoten enthält. Es wird jeweils der Text und die Position dieser Teilszene durch Ergebnisse der Anfrage ersetzt. Die noch nicht expandierte VRML-Szene sieht folgendermaßen aus:

```
...
DEF TextPath OrientationInterpolator { ... }
...
SQLServerInclude {
      connection ""
      maxRow 3
      sqlStatement "SELECT mieter, Xpos, Ypos, Zpos INTO
                    Firma.string, Position.translation FROM messe;"
      template DEF Position Transform {
            children [
                  Shape {
                        appearance Appearance {
                              material Material { }
                        }
                        geometry DEF Firma Text {
                              string []
                        }
                  }
                  ...
                  USE Firma
                  ...
            ]
            rotation          0 1 0 0
            translation       0 0 0

            ROUTE TextPath.value_changed TO Position.set_rotation
      }
}
```

Da der Wert von maxRow 3 beträgt, werden nur drei Ergebnistupel ausgewertet und somit auch nur drei Instanziierungen vorgenommen. Alle Knotennamen, die innerhalb des Templates durch ein DEF-Konstrukt definiert wurden, werden in jeder Instanziierung um einen zusätzlichen Index ergänzt. Der Index ist eine Zahl, die bei 1 anfängt und pro Instanziierung um 1 erhöht wird. So ist eine spätere eindeutige Zuordnung der Knotennamen nach der Expandierung des SSI-Knotens gewährleistet:

```
...
DEF TextPath OrientationInterpolator { ... }
...
Group {
      children [
            DEF Position1 Transform {
                  children [
                        Shape {
                              appearance Appearance {
                                    material Material { }
                              }
                              geometry DEF Firma1 Text {
                              string "Informix"
```

```
                              }
                        }
                        ...
                        USE Firma1
                        ...
                  ]
                  rotation          0 1 0 0
                  translation       0 2 0.5

                  ROUTE TextPath.value_changed TO
                        Position1.set_rotation
            }
            DEF Position2 Transform {
                  children [
                        Shape {
                              appearance Appearance {
                                    material Material { }
                              }
                              geometry DEF Firma2 Text {
                              string "Oracle"
                              }
                        }
                        ...
                        USE Firma2
                        ...
                  ]
                  rotation          0 1 0 0
                  translation       0 2.5 0.5

                  ROUTE TextPath.value_changed TO
                        Position2.set_rotation
            }
            DEF Position3 Transform {
                  children [
                        Shape {
                              appearance Appearance {
                                    material Material { }
                              }
                              geometry DEF Firma3 Text {
                              string "GMD-IPSI"
                              }
                        }
                        ...
                        USE Firma3
                        ...
                  ]
                  rotation          0 1 0 0
                  translation       0 3 0.5

                  ROUTE TextPath.value_changed TO
                  Position3.set_rotation
            }
      ]
}
```

In der expandierten Szene ist der Knotenname "Firma" im USE-Konstrukt und der Knotenname "Position" in der ROUTE-Anweisung für jede Instanziierung um einen

zusätzlichen Index ergänzt worden. Der Knotenname "TextPath" wurde nicht indexiert, da der Knotenname außerhalb des Templates definiert wurde.

5.4.4 Einbettung von multimedialen Daten

In VRML werden multimediale Daten wie z.B. Bilder, Audio und Video durch eine URL integriert, die in speziell dafür vorgesehene Knoten angegeben wird. Die nachfolgende Aufstellung zeigt alle VRML-Knoten, in denen multimediale Daten verwendet werden. Eine ausführlichere Beschreibung findet man in [AMES97]:

ImageTexture-Knoten

Dieser Knoten spezifiziert Texturen und kann z.B. in dem Feld `texture` eines Appearance-Knotens verwendet werden. Die Texturen werden durch Bilder angegeben, die in Dateien mit dem Format JPEG oder GIF gespeichert sein müssen.

MovieTexture-Knoten

In diesem Knoten werden ebenfalls Texturen spezifiziert, die jedoch nicht statisch sind, sondern als eine Folge von Einzelbildern (engl. frames) eines Videos abgespielt werden, welches im MPEG-1-Format vorliegen muß.

AudioClip-Knoten

In diesem Knoten werden Audioquellen beschrieben. Er wird vornehmlich als Wert des Feldes `source` im Sound-Knoten angegeben. Die Audioquellen werden als Sounddateien im General MIDI- oder WAV-Format bereitgestellt. Eine Datei im General MIDI-Format enthält Instruktionen, um Musik mit Hilfe eines in der Soundkarte integrierten Synthesizers zu erzeugen. WAV-Dateien enthalten dagegen unkomprimierte, digitalisierte Musikaufnahmen. Außerdem können die Soundkanäle von Videos abgespielt werden, die im MPEG1- oder MPEG2-Fomat komprimiert sind.

Anchor-Knoten

Dieser Knoten erfüllt in VRML die gleiche Funktionalität wie ein Link in einer HTML-Seite. Der Anchor-Knoten umschließt eine Gruppe von VRML-Objekten, wie z.B. eine geometrische Figur. Wenn man als Betrachter auf ein Element dieser Gruppierung klickt, wird eine im Feld `url` angegebene neue VRML-Szene in den Browser geladen.

Inline-Knoten

Der Inline-Knoten enthält einen Verweis auf eine VRML-Teilszene in Form einer URL, die in der aktuellen Szene bereits dargestellt ist oder bei Bedarf nachgeladen wird.

Es gibt unterschiedliche Möglichkeiten, multimediale Daten in eine VRML-Szene mit Hilfe des SSI-Knotens zu integrieren. Die verschiedenen Varianten unterscheiden sich

im Zugriff auf diese Daten. Multimediale Daten können auf einem herkömmlichen Fileserver oder in einer Datenbank gespeichert sein.

5.4.4.1 Multimediale Daten auf dem Fileserver

Wenn die Daten auf einem Fileserver abgelegt sind, so müssen die entsprechenden URLs in der Datenbank verwaltet werden. Der Nachteil dieser Variante besteht darin, daß eine Veränderung, wie z.B. das Löschen einer Bilddatei auf dem Fileserver, in der Datenbank nicht automatisch nachvollzogen werden kann. Dies hat zur Folge, daß eventuell URLs in eine VRML-Szene integriert werden, die auf nicht mehr vorhandene multimediale Daten verweisen. Diese Daten können dann in der VRML-Szene nicht angezeigt werden.

Ein nicht sehr effizienter Ausweg besteht darin, mehrere URLs in der Datenbank zu speichern, die jeweils auf eine Kopie der gleichen Datei verweisen. Das Feld `url` der oben angeführten Knoten ist vom Typ MFString und kann mehrere URLs enthalten.

Der SSI-Knoten müßte daher alle vorhandenen URLs, die auf die gleiche Datei verweisen, in das Feld abbilden. Der Sinn besteht darin, daß wie in einer Prioritätenliste, beginnend mit der ersten URL nachgeschaut wird, ob man die entsprechend referenzierte Datei öffnen kann. Falls dies nicht klappt, wird die zweite URL in der Liste verwendet usw.

5.4.4.2 Multimediale Daten in der Datenbank

Multimediale Daten, die in der IDS-Datenbank gespeichert sind, werden mit Hilfe des Webdrivers (s. a. Kapitel 5.3.3) in die VRML-Szene integriert. Es gibt zwei Möglichkeiten, sie mit Hilfe einer URL zu referenzieren:

In der ersten Variante wird das entsprechende Objekt durch seinen Namen in der URL angegeben wird. Das nachfolgende Beispiel zeigt einen Ausschnitt einer bereits expandierten VRML-Szene. Es wird ein Bild mit dem eindeutigen Namen `GMD-Logo` aus der Tabelle `Images` referenziert. Dieses Bild befindet sich in der Spalte `obj` und wird durch den Vergleich des Namens mit dem Primärschlüssel `ID` gefunden:

```
...
     texture DEF Logo ImageTexture {
            url "http://myhost:port/cgi-bin/Webdriver.cgi?
                 table=Images&column=obj&name=ID&value=GMD-Logo"
     }
...
```

Man kann diese Form der URL auch komplett ohne das VRML-DataBlade verwenden, falls man lediglich ein in der Datenbank gespeichertes multimediales Objekt in eine **statische** VRML-Szene integrieren will.

Die zweite Variante spricht das multimediale Objekt über ein Large Object Handle (LO-Handle) an, das in der URL angegeben wird. Ein LO-Handle identifiziert ein in der Datenbank gespeichertes Large-Object eindeutig. Das nachfolgende Beispiel zeigt einen Ausschnitt aus einer bereits expandierten VRML-Szene, in der ein Bild durch das LO-Handle referenziert wird:

```
...
     texture DEF Logo ImageTexture {
            url "http://myhost:port/cgi-bin/Webdriver.cgi?
                 LO=...&MItypeObj=image/gif"
     }
...
```

Das LO-Handle ist in Wirklichkeit eine sehr lange hexadezimale Zahlenkette und wird nur verwendet, wenn die URL generiert wird, d.h. eine manuelle Eingabe entfällt.

In beiden Varianten kann die URL in der Datenbank bereits als ganzes gespeichert werden. Sie wird dann in dem SELECT-Statement des SSI-Knotens spezifiziert:

```
...
sqlStatement "SELECT imageURL FROM Images INTO Logo.url WHERE
ID='GMD-Logo'"
...
```

In diesem Fall müßten aber Triggerfunktionen für die Aktualisierung der gespeicherten URLs sorgen, falls Veränderungen, wie z.B. das Hinzufügen eines Bildes in die Tabelle, vorgenommen werden.

Eine andere und wesentlich effizientere Möglichkeit besteht aus der dynamischen Generierung dieser URLs durch einen "intelligenten" Abbildungs- und Templatemechanismus. Beide Mechanismen müssen das Ergebnis der Selektion überprüfen, ob es sich dabei um Large-Objekts handelt oder nicht. Im Falle eines Large Objects wird eine URL generiert und in allen anderen Fällen werden die Ergebnisse direkt in die Knotenfelder abgebildet. Damit würde eine zusätzliche Speicherung von URLs für jedes multimediale Datenobjekt sowie aufwendige Triggermechanismen entfallen.

Der SSI-Knoten sollte auf jeden Fall sowohl multimediale Daten, die sich auf einem Fileserver befinden, als auch solche, die in der IDS-Datenbank gespeichert sind, in VRML-Szenen dynamisch einbetten können. Im nächsten Schritt kann dann eine dynamische Generierung von URLs implementiert werden.

5.4.5 Fehlerbehandlung

Neben den Fehlern, die bei der Verwendung von Funktionen der VRML-Klassenbibliothek auftreten können und deren Behandlung in [RISSE97] beschrieben wurde, ist durch die Einführung des SSI-Knotens eine neue Kategorie von Benutzerfehlern entstanden. Diese unterscheiden sich von den bisherigen Fehlern dadurch, daß die gerade laufende Expandierung aller SSI-Knoten nach Möglichkeit nicht abgebro-

chen, sondern der auftauchende Fehler in der expandierten VRML-Szene beschrieben wird.

Falls ein SSI-Knoten nicht expandiert werden konnte, wird in den Felder `error_no` und `error_message` eine Fehlerkonstante und eine Fehlermeldung zurückgegeben. Der SSI-Knoten bleibt in diesem Fall bestehen, d.h. er wurde während der Expandierung nicht gelöscht.

Die zurückgegebene Fehlermeldung soll dem Anwendungsentwickler einen Hinweis auf die evtl. falsche Benutzung des SSI-Knotens liefern. Die Fehlermeldungen sind expandierungsspezifisch, d.h. es werden nur diejenigen angegeben, die etwas mit der Anwendung des SSI-Knotens zu tun haben. Die evtl. auftauchenden Fehler werden folgendermaßen kategorisiert:

- **SQL-Syntaxfehler**

  Dies sind Fehler, die der SQL-Parser liefert, falls die im Feld `sqlStatement` angegebene SELECT-Anweisung eine falsche Syntax aufweist, z.B. wenn erforderliche Schlüsselwörter wie z.B. "FROM" fehlen oder falsch geschrieben wurden.

- **Ausführungsfehler**

  Hierzu gehören Fehler, die bei einer nicht ausführbaren SELECT-Anweisung entstehen, z.B. die Angabe einer nicht vorhandenen Tabelle.

- **Benennungsfehler**

  Diese Fehler entstehen, wenn falsch geschriebene Knoten- oder Feldnamen verwendet wurden.

- **Abbildungsfehler**

  Hierzu gehören Fehlermeldungen, die gegen die in Kapitel 5.4.1.1 definierten Abbildungsvorschriften und Sonderregeln verstoßen.

Das folgende Beispiel zeigt einen SSI-Knoten nach der Expandierung, der in der SELECT-Anweisung einen nicht vorhanden Knotennamen verwendet:

```
...
SQLServerInclude {
      connection ""
      maxRow 3
      sqlStatement "SELECT Xpos, Ypos, Zpos INTO Positin.translation
                    FROM messe;"
      template DEF Position Transform { ... }
      error_no -7
      error_message "ERROR: Unknown nodename 'Positin' !"
}
...
```

5.4.6 Designerweiterungen

Bei der Spezifizierung des SSI-Knotens wurden verschiedene Eigenschaften diskutiert und einige wenige verworfen. Die nicht in der Spezifikation aufgenommenen Funktionalitäten werden an dieser Stelle kurz erläutert.

Man hat in der jetzigen Spezifikation keine Möglichkeit, mit dem SSI-Knoten direkt auf einzelne Elemente eines komplexen VRML-Feldtyps zuzugreifen. So wird z.B. durch eine Spezifikationserweiterung im folgenden Beispiel ein Wert in die Y-Komponente eines SFVec3F-Feldtyps eingesetzt:

```
...
SQLServerInclude {
      connection ""
      maxRow 1
      sqlStatement "SELECT price INTO value.size.y FROM stock"
      template Transform {
            children [
                  Shape {
                        appearance Appearance {
                              material Material { }
                        }
                        geometry DEF value Box {
                              size 1 1 1
                        }
                  }
            ]
      }
}
...
```

Der Zugriff auf ein Element eines komplexen VRML-Feldtyps erfolgt wie bisher durch Angabe des Knotennamens und des Feldnamens. Neu hinzugekommen ist noch eine Angabe des Elements. In oben gezeigtem Beispiel wird mit einem zusätzlich angehängten ".y" die Y-Komponente des Typs SFVec3F spezifiert, in die der Ergebniswert abgebildet werden soll.

Diese zusätzliche Funktionalität wurde verworfen, da das Konzept eines komplexen Feldelements wie die oben gezeigte Y-Komponente kein Bestandteil der aktuellen VRML-Spezifikation ist. Man kann diese Funktionalität jedoch indirekt nachahmen, indem der Wert mit Hilfe eines SSI-Knotens in ein SFInt32-Feld abgebildet wird und ein Skriptknoten bei der Initialisierung der VRML-Datei diesen in das gewünschte SFVec3F-Feld einsetzt.

Eine weitere nicht in die Spezifikation aufgenomme Funktionalität besteht darin, mit Hilfe eine **Platzhaltermechanismus** bestimmte Elemente eines Multiwertfeldes zu ersetzen. Das folgende Beispiel bildet einen Ergebniswert in ein einzelnes Element eines MFString-Feldes ab:

```
...
SQLServerInclude
{
     connection ""
     maxRow 1
     sqlStatement "SELECT name INTO info.$name FROM employee
               WHERE ID<10;"
     template NULL
}
...
DEF info Text {
     string [ "Name: ", $name ]
     ...
}
...
```

Damit der SQL-Parser unterscheiden kann, ob es sich um einen Feldnamen oder um einen Platzhalter handelt, wird dem Namen ein Sonderzeichen, hier ein "$", vorangestelllt.

Diese Spezifikationserweiterung wurde ebenfalls verworfen, da sie einerseits sehr umfangreiche Änderungen des bestehenden VRML-Parsers hervorgerufen hätte und man andererseits diese Funktionalität in den meisten Fällen sehr einfach durch Verwendung zusätzlicher Knoten in der VRML-Szenenbeschreibung imitieren kann. So könnte man im oben gezeigten Beispiel den Textknoten in zwei Textknoten aufspalten, wobei in einem der vorgegebene Feldwert "Name:" steht und der andere den Wert des Platzhalters durch den herkömmlichen Abbildungsmechanismus erhält.

6 Realisierung des SSI-Knotens

6.1 Anforderungen

Ein Großteil der Funktionalität, die sich hinter dem SSI-Knoten verbirgt, wurde durch die Implementierung von DataBlade-Funktionen realisiert. Jede DataBlade-Funktion wird im Kern des IDS ausgeführt. Bei der Entwicklung von DataBlade-Funktionen sind einige Restriktionen zu berücksichtigen, welche die Architektur des IDS bedingt (s. Kapitel 2.2.1). Die Einschränkungen werden von *Informix* in Form von "DataBlade Module Programming Guidelines" vorgestellt. Es folgt eine kurze Zusammenfassung dieser Programmierungsrichtlinien. Detailliertere Informationen bezüglich der DataBlade-Entwicklung findet der interessierte Leser in [DBDK97].

6.1.1 Speicherverwaltung

Der IDS verfügt über ein eigenes Speicher-Managementsystem, das er über das Betriebssystem legt. Dadurch besitzt er die vollständige Kontrolle über den Speicherverbrauch. Dies hat den Vorteil, daß u. a. bei der Bestimmung der optimalen Ausführung einer SQL-Anweisung Informationen über den benötigten und den noch zur Verfügung stehenden Speicherplatz mit einbezogen werden können. Damit kann der IDS dann ein unnötiges Auslagern von Speicherbereichen auf die Festplatte verhindern. Die Folge ist eine beachtenswerte Beschleunigung der Ausführungsgeschwindigkeit von SQL-Anfragen durch das DBMS.

Die einzelnen virtuellen Prozessoren (VPs) des IDS kommunizieren über einen gemeinsam benutzten Speicherbereich (s. Kapitel 2.2.1). Die Verwendung dieses Speicherbereichs hat diverse Vorteile:

- Die beteiligten VPs sind nicht gezwungen Kopien von Daten mitzuführen, die in diesem Speicherbereich angelegt sind.
- Die benötigten Zugriffe auf die Festplatte werden reduziert, da sich alle Buffer in einem gemeinsamem Pool befinden, auf den alle VPs zugreifen können.
- Es wird seltener auf die Festplatte als auf den gemeinsam genutzten Speicherbereich zugegriffen, da die benötigten Daten dort meist als Ergebnis einer vorherigen Leseoperation vorhanden sind.

- Die Verwendung eines gemeinsamen Speicherbereichs für alle VPs realisiert die schnellstmögliche Interprozesskommunikation, da Schreib- und Lesezugriffe sehr oft mit der hohen Geschwindigkeit von Speicherzugriffen stattfinden.

Bei der Entwicklung von DataBlades sollte man daher unbedingt darauf achten, daß nur die eigenen IDS-Speichermanagementfunktionen wie `mi_alloc()` und `mi_free()` verwendet werde, sonst besteht keine Garantie für eine einwandfreie Durchführung aufgerufener DataBlade-Funktionen. Verwendet man nämlich die herkömmlichen Speichermanagementfunktionen wie `alloc()` und `free()`, wird ein Speicherbereich in dem Stapel des VPs allokiert, in dem die basierende Funktionsbibliothek (engl. shared library) geladen wurde. Findet nun eine Threadmigration statt (die Übergabe eines Threads von einem VP zu einem anderen VP) so sind danach alle Variablen ungültig, die auf Adressen innerhalb des VP eigenen Stapels zeigen. Das Resultat wären unvorhergesehene Nebeneffekte.

6.1.2 Parallele Verarbeitung

Aufgrund einer eigenen Prozeßverwaltung kann der IDS die bereits standardmäßig vorhandenen oder die durch DataBlades neu eingeführten Funktionen parallel verarbeiten. Man unterscheidet hier zwischen echter Parallelverarbeitung auf einem Multiprozessorsystem und Quasi-Parallelverarbeitung auf einem Einprozessorsystem. In letzterem Fall findet keine echte Parallelverarbeitung statt; jeder Prozeß wird nur solange auf einem Prozessor durchgeführt, bis er die Ausführung selbst beendet oder von einem anderen Prozeß verdrängt wird. Nähere Informationen zur Prozeßverwaltung findet man in [TAN92]. In beiden Fällen kann jedoch nicht vorhergesagt werden, zu welcher Zeit welcher Prozeß aktiv ist.

Jeder VP hat einen eigenen Stapel (s.o.) für die Speicherung von Rücksprungadressen, lokalen und temporären Variablen sowie Funktionsparametern. Die Speicherbereiche für Programmcode, statische und globale Variablen werden von allen VPs gemeinsam verwendet.

Die Prozeßsynchronisation ist vollkommen autark, d.h. nicht vom Entwickler mittels irgendwelcher Funktionen beeinflußbar. Die Funktionen eines DataBlade können von mehreren VPs unabhängig voneinander aufgerufen und somit in mehreren Threads gleichzeitig verarbeitet werden. Daher gibt es Einschränkungen, die der Entwickler beachten muß, um eine reibungslose vom IDS gesteuerte Prozeßsynchronisation zu gewährleisten:

Ein DataBlade sollte **niemals** statische oder globale Variablen enthalten. Das bedeutet, daß DataBlade-Funktionen sowie externe Funktionen, die von einem DataBlade aufgerufen werden, sich nicht gegenseitig beeinflussen dürfen. Darüber hinaus ist es äußerst unsicher statische oder globale Variablen zu verwenden, da sie in den Datensegmenten

der VPs angelegt werden und ihre Adressen nach außen hin nicht immer für jeden VP stabil sind. Unter Umständen treten bei einer Threadmigration die gleichen Probleme auf, die bereits in Kapitel 6.1.1 erläutert wurden.

Zusätzlich darf man in einem DataBlade keine Funktionen wie z.B. `fopen()` verwenden, die blockierende I/O-Betriebssystem-Funktionen auslösen können. Eine solche Funktion würde den CPU-VP blockieren und damit die vom IDS forcierte Parallelverarbeitung verhindern. Es wird daher dringend empfohlen, die in der DataBlade-API angebotenen I/O-Funktionen (`mi_file_*`) zu verwenden. Alle POSIX-Funktionen, die weder direkt noch indirekt blockierende Betriebssystemaufrufe auslösen, sind explizit in [DBDK97] von *Informix* freigegeben worden.

Man kann DataBlade-Funktionen gezielt VPs zuweisen: Der CPU-VP ist da am besten geeignet (s. Kapitel 2.2.1). In diesem Fall sollte man jedoch darauf achten, sehr aufwendige Funktionen mit der DataBlade-API-Funktion `mi_yield()` zu koppeln. Diese Funktion transferiert einen Thread zeitweise in die Warteschlange, um andere Threads auf dem CPU-VP verarbeiten zu können. Dies dient einem ausgeglichenen Daten-Durchsatz, da der IDS eine nicht verdrängbare (engl. non-preemptive) Multithreading-Strategie verfolgt. Das bedeutet, daß im Normalfall ohne Verwendung der `mi_yield()`-Funktion eine sehr komplexe Funktion, die momentan von einem Thread ausgeführt wird, andere Threads nicht zum Zuge kommen ließe. Viele DataBlade-API-Funktionen verwenden implizit die `mi_yield()`-Funktion.

6.2 Implementierung

Der SSI-Knoten wurde wie bereits erwähnt als Erweiterung der VRML-Klassenbibliothek implementiert. Daher werden in diesem Kapitel lediglich die vorgenommen Veränderungen und Erweiterungen erläutert. Die bereits vorhandenen Komponenten der VRML-Klassenbibliothek werden nur beschrieben, falls ihre Funktionalität für das Verständnis der Implementierung wichtig ist. Eine Beschreibung der übrigen und hier nicht erläuterten Komponenten findet man in [RISSE97].

6.2.1 Basisklassen

Ein DataBlade unterliegt einer speziellen Speicherverwaltung, damit es innerhalb des IDS einwandfrei läuft (s. Kapitel 6.1.1). Die in C++ herkömmlichen Freigaben sowie Reservierungen von Speicherbereichen werden über die Operatoren `new()` und `delete()` durchgeführt, die intern wiederum die Standard-C-Funktionen `malloc()` und `free()` verwenden. Diese Operatoren werden mit Hilfe der Basisklasse `VRDBBase` überladen, d.h. anstelle des Aufrufs von `malloc()` und `free()` werden

die von der *Informix* DataBlade-API bereitgestellten Funktionen `mi_alloc()` und `mi_free()` verwendet. Alle weiteren Klassen, und dazu gehören auch diejenigen, die den SSI-Knoten und seine Funktionalität realisieren, erben direkt oder indirekt von der Basisklasse `VRDBBase`. Das gewährleistet ein korrektes Speichermanagement gemäß den in Kapitel 6.1 erläuterten Anforderungen.

Alle Klassen für Knoten und komplexe Felder erben von der Klasse `VRDBField`, die wiederum von `VRDBBase` abgeleitet ist. Die Klasse `VRDBField` definiert virtuelle Basisfunktionen, z.B. Funktionen für das Parsen und die Generierung von VRML-Knoten und das Kopieren von Objekten. Alle Klassen, die von VRDBField erben, müssen diese virtuellen Funktionen überschreiben. Klassen, die keine Felder und Knoten repräsentieren, erben direkt von `VRDBBase`.

6.2.2 SSI-Knoten

Die erforderlichen Erweiterungen der VRML-Klassenbibliothek wurden in mehreren Schritten vorgenommen. In diesem Kapitel werden die einzelnen Phasen der Implementierung und die dabei auftauchenden Probleme erklärt. Darüber hinaus wird der Ablauf einer Expandierung eines SSI-Knotens in einer VRML-Szene genau beschrieben. Eine detaillierte Beschreibung der spezifischen Veränderungen und Erweiterungen der VRML-Klassenbibliothek findet man im Kapitel 6.2.3.

6.2.2.1 Implementierungsabschnitte

Integration des SSI-Knotens

Der erste Schritt bestand darin, den SSI-Knoten in die VRML-Klassenbibliothek zu integrieren, damit er erst einmal geparst und generiert werden kann. Dazu wurde eine neue Knotenklasse `VRDBSQLServerInclude eingeführt`, die von der Klasse `SFNode` abgeleitet ist.

Die Klasse `SFNode,` die wiederum von der Klasse `VRDBField` abgeleitet ist, bildet die Basisklasse aller VRML-Knoten. Sie beschreibt die grundlegenden Eigenschaften eines Knotens. Hierzu gehören unter anderem Attribute, die den Knotennamen und den Definitionsnamen speichern. Außerdem findet man hier die virtuellen Funktionen für das Parsen und die Generierung eines VRML-Knotens. Wie in Kapitel 5.1.1.1 bereits erklärt wurde, parst und generiert sich jedes Knotenobjekt selbst, so daß diese Funktionen erst in den Knotenklassen implementiert werden.

Die Klasse `VRDBSQLServerInclude` enthält wie alle anderen Knotenklassen Funktionen für das Setzen und das Lesen der Feldwerte sowie Funktionen für das Parsen und die Generierung des SSI-Knotens.

Speicherung von Knotenpositionen

Im zweiten Schritt wurde ein Mechanismus entwickelt, der einen effizienten Zugriff auf die SSI-Knoten im Objektbaum ermöglicht (s. a. Kapitel 5.1.1.1). Nach eingehender Analyse des Parsers boten sich zwei verschiedene Möglichkeiten:

Die erste Variante besteht darin, jeden SSI-Knoten im Objektbaum mit Hilfe einer rekursiven Traversierung zu finden. Dieser Ansatz wurde schnell verworfen, da es wesentlich einfacher und effizienter ist, die Position der SSI-Knoten mit Hilfe von Zeigern zu speichern, die während des Parsens ermittelt werden. Die Zeiger werden in einer Liste eingetragen, die in der Szenenklasse `VRDBScene` verankert ist. Mit Hilfe eines Listeniterators kann man später auf die SSI-Knoten zugreifen und sie dementsprechend verarbeiten. Da der SSI-Knoten je nach Anwendungsfall im laufe der Expandierung entweder gelöscht oder durch einen Gruppenknoten ersetzt wird (s. Kapitel 5.4), wurde die Knotenklasse `VRDBSQLServerInclude` um zwei Zeiger erweitert:

Der erste Zeiger speichert die Adresse des Vaterknotens, falls der SSI-Knoten in einem Feld vom Typ SFNode steht, und der zweite Zeiger speichert die Anfangsadresse der MFNode-Liste, falls der SSI-Knoten darin als Kindknoten eines Gruppenknotens eingetragen ist.

Der SSI-Knoten referenziert in seiner SELECT-Anweisung andere VRML-Knoten über ihren Definitionsnamen. Ein effizienter Zugriff auf diese Knoten wird ebenfalls mit Hilfe von Zeigern ermöglicht, die in Listen gespeichert werden. Hierbei werden referenzierte Knoten, die innerhalb der Teilszene des Feldes `template` stehen, mit zusätzlichen Informationen wie z.B. einem Index, der die SSI-Knoten-Zugehörigkeit bestimmt, in einer eigenen Liste gesammelt. Die Elemente dieser Liste sind Zeiger auf Objekte der neu eingeführten Klasse `NodeInfo`. Referenzierte Knoten, die sich nicht innerhalb der im Feld `template` angegebenen Teilszene befinden, werden wie SSI-Knoten mit Hilfe von Zeigern auf das Knotenobjekt in einer eigenen Liste gespeichert.

Die oben angeführten Listen wurden mit Hilfe der bereits implementierten Multiwertfelder erstellt. Hierzu mußte jedoch die basierende Templateklassenstruktur erweitert werden, um Listen von Zeigern auf Objekte zu realisieren, **ohne** daß diese Objekte kopiert werden.

Entwicklung eines SQL-Parsers

Der nächste Schritt bestand in der Implementierung eines rudimentären SQL-Parsers. Er überprüft die SELECT-Anweisung, die in dem Feld `sqlStatement` des SSI-Knotens eingetragen ist, auf korrekte Syntax. Der SQL-Parser wird durch die neue Klasse `VRDBSQLParser` repräsentiert.

Die komplette Expandierung eines SSI-Knotens findet mit Hilfe eines sogenannten "Evaluierers" statt, der durch die Klasse `VRDBVRML2Evaluator` verkörpert wird. Der SQL-Parser wird innerhalb des Evaluierers aufgerufen. Während der Syntaxüberprüfung der SELECT-Anweisung werden die in dem INTO-Konstrukt angegebenen Ziele extrahiert und in einer Liste gespeichert. Die Elemente dieser Liste sind Zeiger auf Objekte der neu eingeführten Klasse `SQLData`. Nach einem erfolgreichen Aufruf des SQL-Parsers liegt eine modifizierte ausführbare SELECT-Anweisung vor.

Integration des Abbildungsmechanismus

Die Ergebnisse der SELECT-Anweisung werden in einem sogenannten "Saveset" gespeichert. Dies ist eine Datenstruktur, die es erlaubt beliebig oft über die Ergebnistupelmenge einer SQL-Anfrage zu iterieren. Weitere Informationen zu dem Saveset findet man in [DBDK97].

Da die VRML-Klassenbibliothek objektorientiert aufgebaut ist und jeder Knoten sich selbst parst und generiert, bot es sich an, dieses Konzept auch bei dem Einfügen der SQL-Ergebnisse in die Knotenfelder zu verwenden. Jeder Knoten soll selbst wissen, in welche Felder er die (als Parameter übergebenen) Ergebnisse eintragen muß. Dadurch vermeidet man ein Einfügen der SQL-Ergebnisse von "außen" mit umständlichen CASE-Abfragen, welche die in dem Knotenobjekt zu verwendende Funktion bestimmen.

Der einzige realisierbare Ansatzpunkt für ein solches Vorgehen bestand in der Modifikation des VRMLParsers. Es wurde ein neuer Parser implementiert, der sich von dem bisherigen dadurch unterscheidet, daß er nur **eine** VRML-Zeile parst. Der neue Parser ist in der Klasse `VRML2LineParser` implementiert, die von der Klasse `VRDBVRML2Parser` abgeleitet ist. Dieser Parser erhält als Parameter eine generierte VRML-Zeile, in die zuvor die Ergebnisse der SELECT-Anweisung eingebettet wurden.

Der SSI-Knoten wird durch den Aufruf einer Funktion des Knotenobjekts des Vaters bzw. dessen MFNode-Listenobjekts gelöscht. Der Zugriff auf den jeweiligen Ahnen ist mit Hilfe von Zeigern gewährleistet, die innerhalb des SSI-Knotenobjekts gespeichert sind.

Integration des Templatemechanismus

Das Hauptproblem bei der Integration des Templatemechanismus bestand darin, einen flexiblen und effizienten Instanziierungsvorgang für die in dem Feld `template` angegebene Teilszene zu entwickeln.

Die Instanziierung wurde dadurch ermöglicht, daß in der Teilszene, die natürlich auch im Objektbaum vorhandenen ist, für jedes Ergebnistupel der SELECT-Anweisung neue Werte in die Knotenfelder eingesetzt werden. Außerdem wird jeder Knotenname inner-

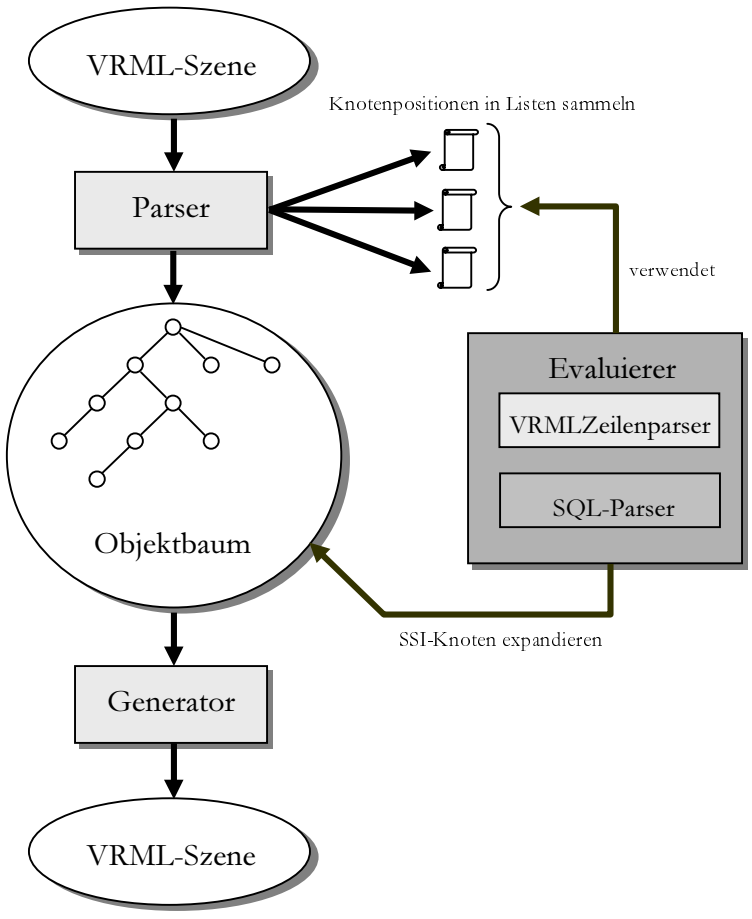
halb dieser Teilszene umbenannt, der durch das DEF-Konstrukt definiert oder der in einem USE-Konstrukt verwendet wird. (s. Kapitel 5.4.3). Danach wird die Teilszene mit Hilfe der rekursiv arbeitenden Kopierfunktion dupliziert, die in jeder Knotenklasse implementiert ist. Diese Kopie wird als Kindknoten an die MFNode-Liste des zuvor erzeugten Gruppenknotens angehängt.

Nachdem alle Ergebnisse der SELECT-Anweisung abgearbeitet worden sind, wird der SSI-Knoten durch den Gruppenknoten ersetzt.

6.2.2.2 Ablauf der Expandierung

Die folgende Abbildung zeigt den Ablauf der SSI-Knoten-Expandierung (vergleiche auch Abbildung 4: Ablauf einer VRML-Szenen-Modifikation):

ABBILDUNG 7: Ablauf der SSI-Knoten-Expandierung



Die noch nicht expandierte VRML-Szene wird zu Beginn geparst. Während des Parsens wird der Objektbaum aufgebaut. In drei unterschiedlichen Listen werden die Positionen und weitere Merkmale von VRML-Knoten gespeichert, die bei der Expandierung benötigt werden.

Nachdem der Objektbaum erzeugt wurde beginnt die eigentliche Expandierung der SSI-Knoten durch den Evaluierer. Er greift mit Hilfe einer Liste, die während des Parsens erzeugt wurde, auf die einzelnen SSI-Knoten zu und extrahiert die für die Expandierung benötigten Informationen aus deren Feldern.

Die SELECT-Anweisung wird mit Hilfe des eingebauten SQL-Parsers überprüft und umgewandelt. Die umgewandelte SELECT-Anweisung wird ausgeführt und die von dem IDS gelieferten Ergebnistupel werden in verschiedene, jeweils neu generierte VRML-Zeilen eingebettet. Diese VRML-Zeilen werden zusammen mit dem VRML-Zeilenparser an die jeweiligen Knotenobjekte übergeben, welche die Ergebniswerte selbständig in ihre Felder einsetzen.

Nachdem alle SSI-Knoten expandiert wurden, erfolgt der Aufruf des Generierers, der die "reine" VRML-Szene erzeugt, die der Anwender in seinem Browser zu sehen bekommt.

6.2.3 Erweiterungen der VRML-Klassenbibliothek

Im folgenden werden alle spezifischen Erweiterungen und Veränderungen der VRML-Klassenbibliothek erklärt, die im Rahmen der SSI-Knoten-Implementierung vorgenommen wurden. Jede neu hinzugefügte Klasse ist innerhalb der jeweiligen Klassenbaumabbildung als Rechteck mit dunkelgrauen Hintergrund dargestellt, nicht veränderte Klassen haben einen weißen Hintergrund. Die bereits vorhandenen, aber modifizierten Klassen besitzen einen hellgrauen Hintergrund. Die in diesen Klassen vorgenommenen Veränderungen oder hinzugefügten Funktionen sind in den Klassendefinitionstabellen ebenfalls durch einem grauen Hintergrund gekennzeichnet.

Dieses Kapitel dient u.a. als Grundlage für spätere Ergänzungen des VRML-DataBlade und soll künftigen Entwicklern die Einarbeitung in die VRML-Klassenbibliothek erleichtern. Zusätzliche Informationen findet man in [RISSE97].

6.2.3.1 Typen

Für die Unterstützung einer effizienten Expandierung des SSI-Knotens wurden verschiedene neue Typen eingeführt. Sie dienen der temporären Speicherung von Informationen.

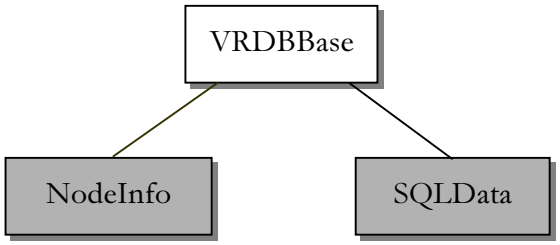
Ein Zeiger auf Objekte der VRML-Knotenklassen wurde als neuer Typ definiert:

```
typedef SFNode* SFPtrNode;
```

Der Typ `SFPtrNode` wird verwendet, um Positionen von VRML-Knoten im Objektbaum zu speichern. Werte dieses Typs werden während des Parsens in Listen vom Typ `MFPtrNode` eingetragen (s. a. Kapitel 6.2.3.3).

Alle weiteren eingeführten Typen wurden als Zeiger auf Objekte neu implementierter Klassen definiert. Die zugehörige Klassenhierarchie ist in der nachfolgenden Abbildung zu sehen:

ABBILDUNG 8: Klassenbaum der Neuen Typen



Für die Speicherung der Position von VRML-Knoten mit definiertem Namen, die sich in der im Feld `template` angegebenen Teilszene befinden, wurde die Klasse `NodeInfo` implementiert. Objekte dieser Klasse speichern neben der Knotenposition den ursprünglichen Knotennamen (`OrigNodename`) und die SSI-Knotenzugehörigkeit (`TemplateID`) mit Hilfe einer ID:

Klasse	NodeInfo
Basisklasse	VRDBBase
Konstruktor & Destruktor	NodeInfo (const SFInt32& id, const SFPtrNode& node); virtual ~NodeInfo();
Public	SFString OrigNodename() const; SFInt32 TemplateID() const; NodeInfo& setTemplateID(const SFInt32 id); SFPtrNode NamedNode() const; NodeInfo& setNamedNode(const SFPtrNode node);
Protected	-

Der neue Typ `SFPtrNodeInfo`, der Zeiger auf Objekte der Klasse `NodeInfo` repräsentiert, wurde folgendermaßen definiert:

```
typedef NodeInfo* SFPtrNodeInfo;
```

Werte dieses Typs werden während des Parsens in Listen vom Typ `MFPtrNodeInfo` eingetragen (s. a. Kapitel 6.2.3.3).

Für die Zwischenspeicherung der in dem INTO-Konstrukt angegebenen Zielfelder wurde die Klasse `SQLData` implementiert. Objekte dieser Klasse enthalten den extra-

hierten Knotennamen (`Nodename`), den extrahierten Feldnamen (`Fieldname`), den Feldtyp (`Fieldtype`) und das einzufügende Ergebnis der SELECT-Anweisung (`Fielddata`):

Klasse	SQLData
Basisklasse	VRDBBase
Konstruktor & Destruktor	SQLData(const SFString& nodename, const SFString& fieldname); virtual ~SQLData();
Public	SFString OrigNodename() const; SFString Nodename() const; SQLData& setNodename(const SFString name); SFString Fieldname() const; SFInt32 Fieldtype() const; SQLData& setFieldtype(const SFInt32 type); SFString Fielddata() const; SQLData& setFielddata(const SFString data);
Protected	-

Der neue Typ `SFPtrSQLData`, der Zeiger auf Objekte dieser Klasse repräsentiert, wurde folgendermaßen definiert:

```
typedef SQLData* SFPtrSQLData;
```

Werte dieses Typs werden während des Parsens in Listen vom Typ `MFPtrSQLData` eingetragen (s. a. Kapitel 6.2.3.3).

6.2.3.2 Multiwertfelder

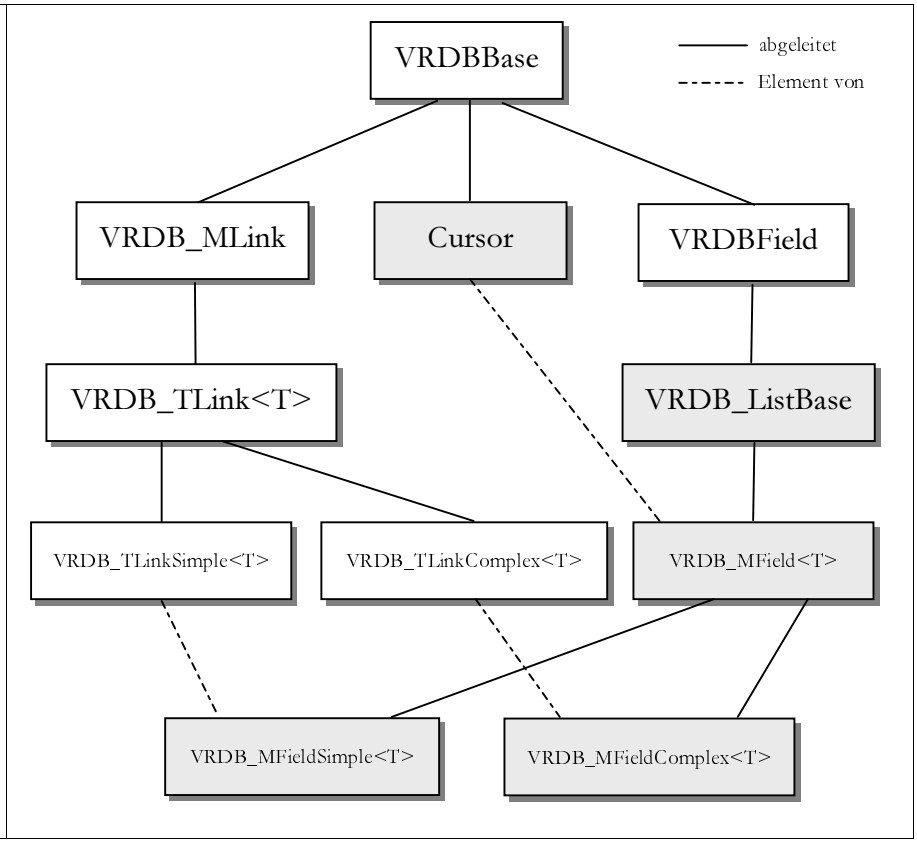
Die Multiwertfelder sind in der VRML-Klassenbibliothek mit Hilfe von Listen-Templates implementiert. Ein Listen-Template ist mit einer Schablone vergleichbar, die es ermöglicht Listenfunktionalitäten, wie Einfügen und Löschen eines Elements, unabhängig von einem konkreten Typen zu implementieren.

Die Initialisierung mit einem konkreten Datentypen wird durch die Übergabe des Typs als Parameter bei der Deklaration der Liste erreicht. Der Begriff des Templates und dessen Anwendungsmöglichkeiten werden ausführlich in [STRO92] erklärt.

Die Liste ist doppelt verkettet, und der Zugriff auf die Elemente erfolgt mit Hilfe eines Listeniterators. Die Werte einfacher Multiwertfelder wie z.B. MFInt32 werden direkt in dem Listenelement und die Werte komplexer Multiwertfelder wie MFNode werden als Verweis auf eine Kopie des Objekts gespeichert.

Die folgende Abbildung zeigt den Klassenbaum der Listentemplates:

ABBILDUNG 9: Klassenbaum der Templates für Multiwert-felder



Die Klasse `VRDB_ListBase` wurde um Funktionen erweitert, die das Spektrum der möglichen Modifikationen einer Liste erweitern. Man kann nun an der momentanen Position des Listeniterators, welcher durch die Klasse `Cursor` repräsentiert wird, Elemente austauschen oder Elemente davor bzw. dahinter einfügen. Diese Erweiterungen waren notwendig, um z.B. den SSI-Knoten in einem Multiwertfeld mit einen Gruppenknoten auszutauschen:

Klasse	VRDB_ListBase
Basisklasse	VRDBField
Konstruktor & Destruktor	VRDB_ListBase(); virtual ~VRDB_ListBase();
Public	virtual VRDBField *copy() const;

	VRDB_MLink *first() const; VRDB_MLink *last() const; long numberOfElements() const; VRDB_ListBase& append(VRDB_MLink *ptr); VRDB_ListBase& remove(VRDB_MLink *ptr); VRDB_ListBase& insertbefore(VRDB_MLink *before, VRDB_MLink *ptr); VRDB_ListBase& insertafter(VRDB_MLink *after, VRDB_MLink *ptr); VRDB_ListBase& replace(VRDB_MLink *oldptr, VRDB_MLink *newptr);
Protected	VRDB_ListBase& setFirst(VRDB_MLink *ptr); VRDB_ListBase& setLast(VRDB_MLink *ptr);

Für die Unterstützung der neuen Listenfunktionen mußte die Klasse `Cursor` um zusätzliche Überprüfungsfunktionen erweitert werden:

Klasse	Cursor
Basisklasse	VRDBBase
Konstruktor & Destruktor	Cursor(); Cursor(const VRDB_MField<T>& mfield); Cursor(VRDB_MField<T> *mfield); virtual ~Cursor();
Public	Cursor<T>& setToFirst(); Cursor<T>& setToLast(); Cursor<T>& setToNext(); Cursor<T>& setToPrev(); SFBool isValid() const; SFBool isNextValid() const; SFBool isPrevValid() const; T const& element() const;
Protected	-

Bedingt durch die oben gezeigte Vererbungsstruktur sind die neuen Listenfunktionen in allen Nachkommen vorhanden. In der Klasse `VRDB_MField` werden sie als abstrakte

Funktionen deklariert, deren Implementierungen man erst in den Template-Klassen `VRDB_MFieldSimple` und `VRDB_MFieldComplex` findet:

Klasse	VRDB_MField
Basisklasse	VRDB_ListBase
Konstruktor & Destruktor	VRDB_MField(); virtual ~VRDB_MField();

Public	VRDB_MField<T> &operator= (const VRDB_MField<T> &field);
	virtual VRDB_MField<T>& setMField(const VRDB_MField<T> &field);
	virtual VRDB_MField<T>& setArray(T *a, unsigned long number);
	virtual VRDB_MField<T>& setSingle(const T& a);
	virtual VRDB_MField<T>& appendMField(const VRDB_MField<T> &field);
	virtual VRDB_MField<T>& appendArray(T *a, unsigned long number);
	virtual VRDB_MField<T>& append(const T& a) = 0;
	virtual VRDB_MField<T>& insertbefore(Cursor<T>& cur, const T& a) = 0;
	virtual VRDB_MField<T>& insertafter(Cursor<T>& cur, const T& a) = 0;
	virtual VRDB_MField<T>& replace(Cursor<T>& cur, const T& a) = 0;
	
	VRDB_TLink<T> *first() const;
	VRDB_TLink<T> *last() const;
	
	VRDB_MField<T>& remove(Cursor<T>& cur);
	VRDB_MField<T>& removeAll();

Protected	-

Klasse	VRDB_MFieldSimple<T>
Basisklasse	VRDB_MField<T>
Konstruktor & Destruktor	VRDB_MFieldSimple(); VRDB_MFieldSimple(const T& a); VRDB_MFieldSimple(const VRDB_MFieldSimple<T>& field); virtual VRDB_MField<T>& append(const T& a);

Public	virtual VRDB_MField<T>& append(const T& a);
	
	virtual VRDB_MField<T>& insertbefore(Cursor<T>& cur, const T& a);
	virtual VRDB_MField<T>& insertafter(Cursor<T>& cur, const T& a);
	virtual VRDB_MField<T>& replace(Cursor<T>& cur, const T& a);
	
	virtual SFString asString() const;
	virtual const VRDBField& asVRML(VRDBVRML2Generator& generator, long indentLevel=0) const;
	
	virtual VRDBField &setValues(VRDBVRML2Parser& parser);

Protected	-

Klasse	VRDB_MFieldComplex<T>
Basisklasse	VRDB_MField<T>
Konstruktor & Destruktor	VRDB_MFieldComplex(); VRDB_MFieldComplex(const T& a); VRDB_MFieldComplex(const VRDB_MFieldComplex<T>& field);

Public	VRDB_MFieldComplex<T> &operator= (const VRDB_MFieldComplex<T> &field);
	
	virtual VRDB_MField<T>& append(T *a);
	virtual VRDB_MField<T>& append(const T& a);
	
	virtual VRDB_MField<T>& insertbefore(Cursor<T>& cur, const T& a);
	virtual VRDB_MField<T>& insertafter(Cursor<T>& cur, const T& a);
	virtual VRDB_MField<T>& replace(Cursor<T>& cur, const T& a);
	
	virtual SFString asString() const;
	virtual const VRDBField& asVRML(VRDBVRML2Generator& generator, long indentLevel=0) const;
	
	virtual VRDBField &setValues(VRDBVRML2Parser& parser);

Protected	-

6.2.3.3 Hilfslisten

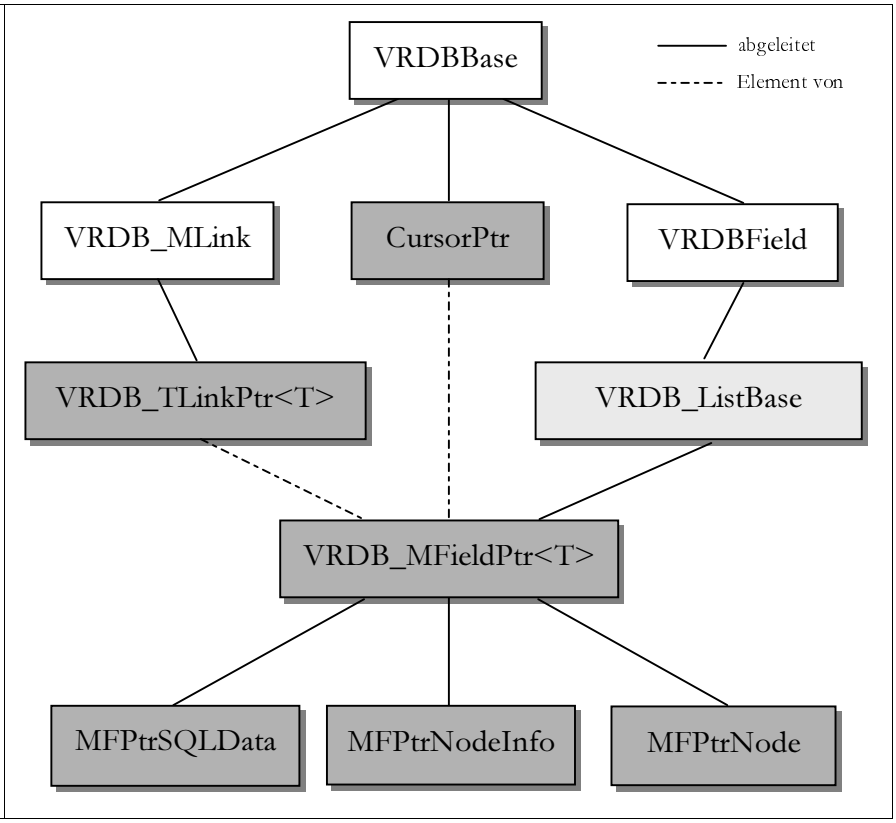
Die Hilfslisten enthalten wichtige Informationen für die Expandierung und basieren wie die Multiwertfelder auf Listen-Templates. Da sie jedoch beim Einfügen eines Listenelements dieses **nicht** vorher kopieren sollen, mußten umfangreiche Veränderungen und Erweiterungen an der bisherigen Template-Klassenhierarchie vorgenommen werden. Bis auf die Klasse `VRDB_ListBase` und `VRDB_MLink`, welche ein Listenelement repräsentiert, konnten keine der bisherigen Template-Klassen übernommen werden.

Die Template-Klasse `VRDB_TLinkPtr` erweitert die Klasse `VRDB_MLink` um die Speicherung eines Zeigers auf Objekte beliebiger Klassen. Die Klasse `VRDB_MFieldPtr` erweitert die Klasse `VRDB_ListBase` um Funktionen zur Erstellung und Bearbeitung von Listen, wie z.B. die Verknüpfung zweier Listen.

Die Klassen `MFPtrSQLData`, `MFPtrNodeInfo` und `MFPtrNode` repräsentieren Listen eines konkreten Datentyps.

Die folgende Abbildung zeigt den Ableitungsbaum der Template-Klassen, die für die Erzeugung der Hilfslisten benötigt werden:

ABBILDUNG 10: Klassenbaum der Templates für Hilfslisten



Die Klasse `VRDB_TLinkPtr` implementiert ein Template-Listenelement:

Klasse	VRDB_TLinkPtr
Basisklasse	VRDB_MLink
Konstruktor & Destruktor	VRDB_TLinkPtr(const T& a); virtual ~VRDB_TLinkPtr();
Public	VRDB_TLinkPtr<T> *next() const; VRDB_TLinkPtr<T> *prev() const; virtual T const& element() const;
Protected	-

Der Listeniterator wird durch die Klasse `CursorPtr` repräsentiert:

Klasse	CursorPtr
Basisklasse	VRDBBase
Konstruktor & Destruktor	CursorPtr(); CursorPtr(const VRDB_MFieldPtr<T>& mfieldptr); CursorPtr(VRDB_MFieldPtr<T> *mfieldptr); virtual ~CursorPtr() ;
Public	CursorPtr<T>& setToFirst(); CursorPtr<T>& setToLast(); CursorPtr<T>& setToNext(); CursorPtr<T>& setToPrev(); SFBool isNextValid() const; SFBool isPrevValid() const; SFBool isValid() const; T const& element() const;
Protected	-

Die Template-Klasse `VRDB_MFieldPtr` repräsentiert alle Listen, die Zeiger auf Objekte beliebiger Klassen speichern können:

Klasse	VRDB_MFieldPtr
Basisklasse	VRDB_ListBase
Konstruktor & Destruktor	VRDB_MFieldPtr(); VRDB_MFieldPtr(const T& a);

	VRDB_MFieldPtr(const VRDB_MFieldPtr<T>& field); virtual ~VRDB_MFieldPtr();

Public	VRDB_MFieldPtr<T> &operator= (const VRDB_MFieldPtr<T> &field);
	
	virtual VRDB_MFieldPtr<T>& setMFieldPtr(const VRDB_MFieldPtr<T> &field);
	virtual VRDB_MFieldPtr<T>& setArray(T *a, unsigned long number);
	virtual VRDB_MFieldPtr<T>& setSingle(const T& a);
	
	virtual VRDB_MFieldPtr<T>& appendMFieldPtr(const VRDB_MFieldPtr<T> &field);
	virtual VRDB_MFieldPtr<T>& appendArray(T *a, unsigned long number);
	virtual VRDB_MFieldPtr<T>& append(const T& a);
	
	VRDB_TLinkPtr<T> *first() const;
	VRDB_TLinkPtr<T> *last() const;
	
	VRDB_MFieldPtr<T>& remove(CursorPtr<T>& cur);
	VRDB_MFieldPtr<T>& removeAll();

Protected	-

Mit der Klasse `MFPtrNode` wurde eine Liste implementiert, die Zeiger auf Objekte beliebiger Knotenklassen speichert (`SFPtrNode`), d.h. sie enthält die Adressen von Knotenobjekten im Objektbaum. Zwei Listen dieses Typs werden während des Parsens der VRML-Szene verwendet. Die erste Liste speichert die Adressen aller SSI-Knotenobjekte und die zweite Liste speichert die Adressen aller definierten VRML-Knotenobjekte, die sich außerhalb der SSI-Knoten befinden:

Klasse	MFPtrNode
Basisklasse	VRDB_MFieldPtr<SFPtrNode>
Konstruktor & Destruktor	MFPtrNode(); MFPtrNode(const SFPtrNode& a); MFPtrNode(const MFPtrNode& field);
Public	-
Protected	-

Die Klasse `MFPtrNodeInfo` repräsentiert ein Liste, die Zeiger auf Objekte der Klasse `NodeInfo` speichert (`SFPtrNodeInfo`). Eine Liste dieses Typs wird während des Parsens der VRML-Szene verwendet. Sie speichert die Adressen und weitere Informationen von VRML-Knotenobjekten, die sich innerhalb der im Feld `template` (SSI-Knoten) angegebenen Teilszene befinden:

Klasse	MFPtrNodeInfo
Basisklasse	VRDB_MFieldPtr<SFPtrNodeInfo>
Konstruktor & Destruktor	MFPtrNodeInfo(); MFPtrNodeInfo(const SFPtrNodeInfo& a); MFPtrNodeInfo(const MFPtrNodeInfo& field);
Public	-
Protected	-

Mit der Klasse `MFPtrSQLData` kann man Listen erzeugen, die Zeiger auf Objekte der Klasse `SQLData` speichern (`SFPtrSQLData`). Listen dieses Typs werden während des Parsens der SELECT-Anweisung des SSI-Knotens benötigt. Die im INTO-Konstrukt angegebenen Ziele werden extrahiert und in dieser Liste zwischengespeichert:

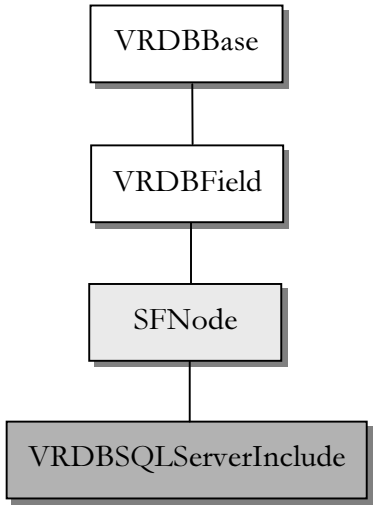
Klasse	MFPtrSQLData
Basisklasse	VRDB_MFieldPtr<SFPtrSQLData>
Konstruktor & Destruktor	MFPtrSQLData(); MFPtrSQLData(const SFPtrSQLData& a); MFPtrSQLData(const MFPtrSQLData& field);
Public	-
Protected	-

6.2.3.4 Knotenklassen

SSI-Knotenklasse

Die Klasse `VRDBSQLServerInclude` beschreibt den SSI-Knoten. Sie ist wie alle anderen Knotenklassen von der Klasse `SFNode` abgeleitet:

ABBILDUNG 11: Klassenbaum des SSI-Knotens



Neben den Funktionen für das Lesen und Setzen eigener Feldwerte gibt es Funktionen, die den Zeiger auf den Vaterknoten (`ParentNode`) speichern bzw. löschen, falls der SSI-Knoten in einem SFNode-Feld steht. Darüber hinaus gibt es Funktionen, die den Zeiger auf die MFNode-Liste (`ParenList`) speichern bzw. löschen, falls der SSI-Knoten als Kindknoten in einem der Gruppenknoten eingetragen ist (s. a. Kapitel 5.4):

Klasse	VRDBSQLServerInclude
Basisklasse	SFNode
Konstruktor & Destruktor	VRDBSQLServerInclude(); VRDBSQLServerInclude(const VRDBSQLServerInclude& node); VRDBSQLServerInclude(const SFString& Connection, const SFInt32& MaxRow, const SFString& SQLStatement); VRDBSQLServerInclude(const SFString& Connection, const SFInt32& MaxRow, const SFString& SQLStatement, const SFNode& Template); virtual ~VRDBSQLServerInclude();

Public	static SFString typeName();
	virtual SFString type() const;
	
	SFString Connection() const;
	VRDBSQLServerInclude& setConnection(SFString connect);
	
	SFInt32 MaxRow() const;
	VRDBSQLServerInclude& setMaxRow (SFInt32 max=-1);
	
	SFString SQLStatement() const;

	VRDBSQLServerInclude& setSQLStatement(SFString sql); SFNode& Template() const; VRDBSQLServerInclude& setTemplate(const SFNode& templ); VRDBSQLServerInclude& clearTemplate(); SFBool isTemplate() const; SFInt32 Error_no() const; VRDBSQLServerInclude& setError_no (SFInt32 error); SFString Error_message() const; VRDBSQLServerInclude& setError_message (SFString errormsg); MFNode *ParentList() const; VRDBSQLServerInclude& setParentList (MFNode *mfnodeptr); VRDBSQLServerInclude& clearParentList (); SFBool isParentList() const; SFNode **ParentNode() const; VRDBSQLServerInclude& setParentNode (SFNode **sfnodeptr); VRDBSQLServerInclude& clearParentNode (); SFBool isParentNode() const; virtual VRDBField *copy() const ; SFInt32 getFieldtype(SFString& fieldname) const;
Protected	virtual SFNode& setItem(VRDBVRML2Parser& parser); virtual const SFNode& _parameterAsVRML (VRDBVRML2Generator& generator, long indentLevel=0) const;

Sonstige Knotenklassen

Die Klasse `SFNode`, von der alle anderen Knotenklassen direkt oder indirekt erben, wurde um zwei virtuelle Funktionen erweitert. Die Funktion `getFieldtype()` liefert bei Angabe eines Feldnamens als Parameter den entsprechenden Feldtyp als Konstante zurück. Sie wird während der Expandierung bei der Generierung der VRML-Zeilen benötigt. Jede Knotenklasse wurde um diese Funktion erweitert. Es wird an dieser Stelle lediglich die Klassenbeschreibung der Klasse `SFNode` angegeben, da die Syntax in allen anderen Knotenklassen gleich ist:

Klasse	SFNode
Basisklasse	VRDBField
Konstruktor & Destruktor	SFNode(); SFNode(const SFNode& node); virtual ~SFNode();

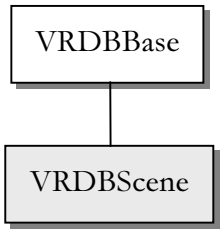
Public	static SFString typeName();
	virtual SFString type() const;
	
	SFNode& setDefinition(const SFString& name);
	SFNode& clrDefinition();
	
	SFBool isDefinition() const;
	SFString definitionName() const;
	
	virtual SFInt32 getFieldtype(SFString& fieldname) const;
	
	virtual VRDBField& setValues(VRDBVRML2Parser& parser);
	virtual const VRDBField& asVRML(VRDBVRML2Generator& generator, long indentLevel=0) const;
	operator SFString() const;
	
	virtual VRDBField *copy() const;
	SFNode *copyNode() const;

Protected	virtual SFNode& setItem(VRDBVRML2Parser& parser);
	virtual const SFNode& _parameterAsVRML (VRDBVRML2Generator& generator, long indentLevel=0) const;
	SFInt32 parseInt32(VRDBVRML2Parser& parser, SFInt32 defValue) const;
	SFFloat parseFloat(VRDBVRML2Parser& parser, SFFloat defValue) const;
	SFBool parseBoolean(VRDBVRML2Parser& parser, SFBool defValue) const;
	SFString booleanAsVRML(SFBool value) const;

6.2.3.5 Szenenklasse

Die Objekte der Klasse `VRDBScene` repräsentieren vollständige VRML-Szenengraphen. Die Klasse `VRDBScene` ist direkt von der Klasse `VRDBBase` abgeleitet:

ABBILDUNG 12: Klassenbaum der Szenenklasse



In dieser Klasse sind die drei oben genannten Listen als Objektattribute verankert (s. a. Kapitel 6.2.3.3). Es wurden neue Funktionen (`DefNodes()`, `TempDefNodes()` und `DynNodes()`) implementiert, die den Zugriff auf diese Listen ermöglichen. Die Expandierung aller SSI-Knoten wird mit dem Aufruf der Funktion `evaluateScene()` gestartet:

Klasse	VRDBScene
Basisklasse	VRDBBase
Konstruktor & Destruktor	VRDBScene(); VRDBScene(const SFString& vrmlScene); VRDBScene(_SFString *vrmlScene); VRDBScene(const SFNode& vrmlScene); virtual ~VRDBScene();

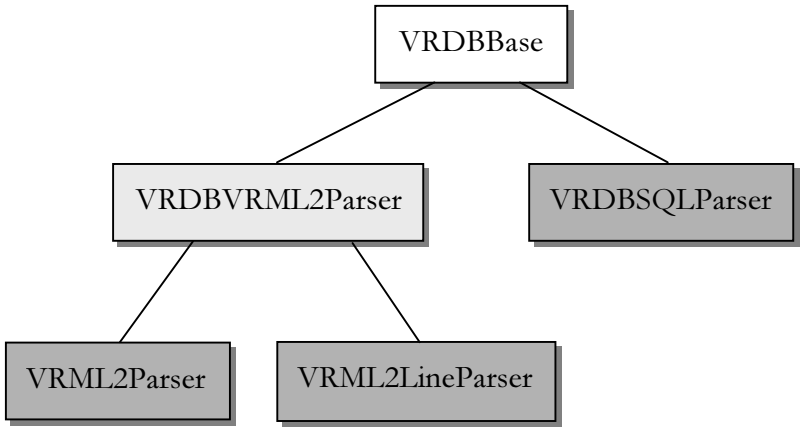
Public	VRDBScene& setScene(const SFString& vrmlScene);
	VRDBScene& setScene(_SFString *vrmlScene);
	VRDBScene& setScene(const SFNode& vrmlScene);
	
	const MFNode& scene() const;
	MFNode& scene();
	
	MFPtrNode& DefNodes();
	MFPtrNode& DynNodes();
	MFPtrNodeInfo& TempDefNodes();
	VRDBScene& evaluateScene(MI_CONNECTION *connection);
	
	const VRDBScene& asVRML(VRDBVRML2Generator& generator) const;

	SFString asVRMLString() const; VRDBScene(MI_CONNECTION *connection, MI_LO_FD desc); VRDBScene& setScene(MI_CONNECTION *connection, MI_LO_FD desc); const VRDBScene& generateVRML(MI_CONNECTION *connection, MI_LO_FD desc) const; MI_LO_HANDLE *generateVRML(MI_CONNECTION *connection) const;
Protected	-

6.2.3.6 Parserklassen

Es wurden neue Parserklassen implementiert, die in der nachfolgenden Klassenbaumabbildung zu sehen sind:

ABBILDUNG 13: Klassenbaum der Parser-klassen



VRML-Parser

Die Konstruktoren des ursprünglichen VRML-Parsers (Klasse `VRDBVRML2Parser`) wurden modifiziert. Sie erhalten neben den alten Parametern noch zusätzliche Zeiger auf die in der Klasse `VRDBScene` verankerten Listen. Die Funktion `createNode()` wurde erweitert. Hier werden alle signifikanten Informationen in Listen eingetragen, die für die Expandierung des SSI-Knoten benötigt werden:

Klasse	VRDBVRML2Parser
Basisklasse	VRDBBase
Konstruktor & Destruktor	VRDBVRML2Parser(const SFString& string, MFPtrNode *DynNodes, MFPtrNode *DefNodes, MFPtrNodeInfo *TempDefNodes); VRDBVRML2Parser(MI_CONNECTION *connection, MI_LO_FD desc, MFPtrNode *DynNodes, MFPtrNode *DefNodes, MFPtrNodeInfo *TempDefNodes); virtual ~VRDBVRML2Parser();

Public	VRDBVRML2Parser& skipAllSpace(SFBool remarks=true);
	VRDBVRML2Parser& nextLine();
	VRDBVRML2Parser& nextWord();
	
	const SFString& word();
	
	char getChar();
	char nextChar();
	
	virtual SFBool initParse();
	SFNode *parse(MFNode *mfnodeptr = NULL);
	SFBool isMoreToParse();
	
	SFNode *createNode(MFNode *mfnodeptr = NULL, SFNode **sfnodeptr = NULL);
	SFBool isVRML20() const;

Protected	-

Die Klasse `VRML2Parser` repräsentiert den bisher in der VRML-Klassenbibliothek vorhandenen VRML-Parser. Sie erbt von der Klasse `VRDBVRML2Parser` und implementiert die virtuelle Funktion `initParse()`. In dieser Funktion wird unter anderem geprüft, ob eine VRML-2.0-Datei vorliegt:

Klasse	VRML2Parser
Basisklasse	VRDBVRML2Parser
Konstruktor & Destruktor	VRML2Parser(const SFString& string, MFPtrNode *DynNodes, MFPtrNode *DefNodes, MFPtrNodeInfo *TempDefNodes); VRML2Parser(MI_CONNECTION *connection, MI_LO_FD desc, MFPtrNode *DynNodes, MFPtrNode *DefNodes,

	MFPtrNodeInfo *TempDefNodes); virtual ~VRML2Parser();
Public	SFBool initParse();
Protected	-

Die Klasse `VRML2LineParser` unterscheidet sich von der Klasse `VRML2Parser` lediglich durch die Implementierung der Funktion `initParse()`. In dieser Funktion wird auf die Überprüfung der VRML2.0-Kennung verzichtet, damit der entsprechende VRML-Knoten eine VRML-Zeile direkt parsen und die darin enthaltenen Werte in seine Felder eintragen kann:

Klasse	VRML2LineParser
Basisklasse	VRDBVRML2Parser
Konstruktor & Destruktor	VRML2LineParser(const SFString& string, MFPtrNode *DynNodes, MFPtrNode *DefNodes, MFPtrNodeInfo *TempDefNodes); virtual ~VRML2LineParser();
Public	SFBool initParse();
Protected	-

SQLParser

Für die Analyse der SELECT-Anweisung des SSI-Knotens wurde ein rudimentärer SQL-Parser durch die Klasse `SQLParser` implementiert. Sie ist von der Basisklasse `VRDBBase` abgeleitet. Der Konstruktor initialisiert den SQL-Parser durch den Aufruf der Funktion `initParse()`. Die Funktion `skipAllSpace()` überspringt die eventuell vorhandenen Leerzeichen. Mit der Funktion `nextWord()` springt man zum nächsten Wort, welches mit `word()` abgefragt werden kann. Einzelne Zeichen werden mit den Funktionen `getChar()` und `nextChar()` gelesen. Die Funktion `isMoreToParse()` überprüft, ob es noch weitere Daten zum Parsen gibt:

Klasse	SQLParser
Basisklasse	VRDBBase
Konstruktor & Destruktor	VRDBSQLParser(SFString& string); virtual ~VRDBSQLParser();
Public	VRDBSQLParser& skipAllSpace(); VRDBSQLParser& nextWord(); const SFString& word();

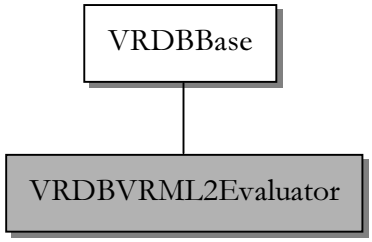
	char getChar();
	char nextChar();
	
	SFBool initParse();
	SFBool isMoreToParse();

Protected	-

6.2.3.7 Evaluatorklasse

Die Klasse `VRDBVRML2Evaluator` enthält die Kernfunktionalität für die Expandierung der SSI-Knoten in einer VRML-Szenenbeschreibung. Sie ist von der Klasse `VRDBBase` abgeleitet:

ABBILDUNG 14: Klassenbaum der Evaluator-klasse



Dem Konstruktor der Klasse `VRDBVRML2Evaluator` werden Zeiger auf Listen übergeben, die während des Parsens erzeugt wurden. Diese Listen enthalten wichtige Informationen für die Expandierung.

Der Zeiger `DynNodes` verweist auf die Liste aller SSI-Knoten, der Zeiger `DefNodes` verweist auf die Liste aller definierten VRML-Knoten, die sich **außerhalb** des SSI-Knotens befinden, und der Zeiger `TempDefNodes` verweist auf die Liste aller definierten VRML-Knoten, die sich **innerhalb** des SSI-Knotens befinden. Für jede dieser Listen wurden eigene Navigations- und Zugriffsfunktionen implementiert.

Die Expandierung der SSI-Knoten wird mit der Funktion `runEvaluation()` gestartet. In dieser Funktion wird über die SSI-Knotenliste (`DynNodes`) iteriert. Die Überprüfung der SELECT-Anweisung des aktuellen SSI-Knotens, die Extraktion der Zielfelder und die Erzeugung einer in der Datenbank ausführbaren SELECT-Anweisung erfolgt durch den Aufruf der Funktion `analyzeSQL()`. Mit Hilfe der Funktion `determineFieldtypes()` werden die Feldtypen der in dem INTO-Konstrukt angegebenen Zielfelder bestimmt.

Durch die Funktion `executeSQL()` wird die umgewandelte SELECT-Anweisung ausgeführt und die Ergebnisse in einer Hilfsstruktur der Data-Blade-API abgespeichert.

Der Abbildungsmechanismus wird mit Hilfe der Funktionen `BuildSQLResults()` und `insertResIntoDefNode()` ausgeführt, wobei erstere die einzufügenden VRML-Zeilen aus den Ergebnissen der SELECT-Anweisung generiert und letztere diese mit Hilfe des VRML-Zeilenparsers in die entsprechenden Knoten einfügt. Nach der Expandierung werden alle SSI-Knoten, die einen Abbildungsmechanismus ausgelöst haben, mit der Funktion `ClearDynNodes()` aus dem VRML-Szenengraphen gelöscht.

Der Templatemechanismus wird mit Hilfe der Funktionen `buildTemplates()` und `insertResIntoTempDefNode()` ausgeführt, die ähnlich arbeiten wie die entsprechenden Funktionen für den Abbildungsmechanismus. Die SSI-Knoten, die einen Templatemechanismus ausgelöst haben, werden durch die Funktion `exchangeDynNode()` gegen einen Gruppenknoten in der VRML-Szenenbeschreibung ausgetauscht.

Klasse	VRDBVRML2Evaluator
Basisklasse	VRDBBase
Konstruktor & Destruktor	VRDBVRML2Evaluator(MFPtrNode *DynNodes, MFPtrNode *DefNodes, MFPtrNodeInfo *TempDefNodes); virtual ~VRDBVRML2Evaluator();
Public	VRDBVRML2Evaluator& setToFirstDynNode(); VRDBVRML2Evaluator& setToLastDynNode(); VRDBVRML2Evaluator& setToNextDynNode(); SFBool isValidDynNode() const; VRDBVRML2Evaluator& clearDynNodes(); VRDBVRML2Evaluator& exchangeDynNode(SFNode& newNode); SFString getSQLStatement() const; SFInt32 getMaxRow() const; SFNode& getTemplate() const; SFBool isTemplate() const; VRDBVRML2Evaluator& setError(SFInt32 error_no, SFString error_msg) SFBool isError() const; VRDBVRML2Evaluator& setToFirstDefNode(); VRDBVRML2Evaluator& setToLastDefNode(); VRDBVRML2Evaluator& setToNextDefNode(); SFBool setToDefNode(SFString& defname); SFBool isValidDefNode() const; SFBool insertLineIntoDefNode(SFString& line); VRDBVRML2Evaluator& insertResIntoDefNode();

	VRDBVRML2Evaluator& setToFirstTempDefNode(); VRDBVRML2Evaluator& setToLastTempDefNode(); VRDBVRML2Evaluator& setToNextTempDefNode(); SFBool setToTempDefNode(SFInt32 id); SFBool setToTempDefNode(SFInt32 id, SFString& defname); SFBool isValidTempDefNode() const; SFBool insertLineIntoTempDefNode(SFString& line); VRDBVRML2Evaluator& insertResIntoTempDefNode(SFInt32 id); VRDBVRML2Evaluator& clearListTempDefNodes(); SFString getCurSQL() const; VRDBVRML2Evaluator& setCurSQL(SFString& sql); SFString getEvalSQL() const; VRDBVRML2Evaluator& setEvalSQL(SFString& sql); VRDBVRML2Evaluator& analyzeSQL(SFString& SQLStatement); VRDBVRML2Evaluator& executeSQL(SFString& SQLStatement, MI_CONNECTION *connection); VRDBVRML2Evaluator& buildSQLResults(); SFBool buildTemplates(SFInt32 id); VRDBVRML2Evaluator& clearListData(); VRDBVRML2Evaluator& clearsqlresults(MI_CONNECTION *connection); VRDBVRML2Evaluator& determineFieldtypes(SFInt32 id); SFBool runEvaluation(MI_CONNECTION *connection);
Protected	-

6.3 SQL-Schnittstelle

Die in Kapitel 5.1.1.2 beschriebene SQL-Schnittstelle des VRML-DataBlades wurde lediglich um eine Funktion erweitert. Die Funktion `VRMLExpand()` ruft die in der VRML-Klassenbibliothek gleichnamige Expandierungsfunktion auf. Die Deklaration sieht folgendermaßen aus:

```
create function VRMLExpand(s vrml) returning vrml external name
'/home/.../vrdb.so(VRMLExpand)' language C
end function;
```

Sie erhält bei ihrem Aufruf als Parameter eine VRML-Szene, die keine oder mehrere SSI-Knoten enthält, und liefert als Ergebnis eine vollständig expandierte VRML-Szene. Das Ergebnis der Funktion ist vom Typ CLOB, so daß es mit weiteren SQL-Anweisungen verarbeitet werden kann.

6.4 Fehlende Funktionalitäten

Aus Zeitgründen konnten nicht alle Spezifikationskonzepte des SSI-Knotens implementiert werden. Es fehlen folgende Eigenschaften:

- **Platzhalter für Feldwerte in der SELECT-Anweisung des SSI-Knotens**

  Laut der SSI-Knotenspezifikation soll es möglich sein, Platzhalter für Feldwerte im WHERE-Konstrukt der SELECT-Anweisung anzugeben (s. Kapitel 5.4.1). Diese Eigenschaft des SSI-Knotens wird in der momentanen Version des VRML-DataBlade noch nicht unterstützt. Die Implementierung dieser Erweiterung ist jedoch sehr einfach, da lediglich die Klasse `SFNode` um eine virtuelle Funktion erweitert werden muß, deren spezifische Implementierung in den Knotenklassen erfolgt. Darüber hinaus muß in der Klasse `VRDBVRML2Evaluator` die Funktion `analyzeSQL()` um einem Mechanismus erweitert werden, der Feldwerte in die SELECT-Anweisung einbettet.

- **ROUTE-Anweisungen im Template des SSI-Knotens**

  Neben den verwendeten Knotennamen in DEF- und USE-Konstrukten sollen (gemäß der SSI-Knotenspezifikation) auch Knotennamen in ROUTE-Anweisungen bei der Instanziierung der Teilszene indexiert werden (s. a. Kapitel 5.4.3). Dies ist für eine spätere eindeutige Identifizierung instanziierter Knoten unerläßlich. Die ROUTE-Anweisungen werden in der VRML-Klassenbibliothek wie Knotenklassen behandelt und können daher nicht an beliebigen Stellen in der VRML-Szenenbeschreibung stehen, was aber eine Grundvoraussetzung für die Verwendung von ROUTE-Anweisungen in der im Feld `template` angegebenen Teilszene ist. Die aktuelle VRML-Spezifikation besagt, daß eine ROUTE-Anweisung keinen VRML-Knoten darstellt und es keine Einschränkungen bezüglich der Lokalität gibt. Um dieses Spezifikationskonzept zu implementieren, muß die VRML-Klassenbibliothek geändert werden.

7 Evaluierung

Das Ziel der Evaluierung ist, den Einsatz des SSI-Knotens in der Praxis zu zeigen. Die dafür implementierte Demo-Anwendung sollte im besten Fall alle funktionalen Merkmale des SSI-Knotens ausnutzen und im Einsatz validieren. Zur Demonstration wurde eine virtuelle Messeanwendung entwickelt, die im folgenden beschrieben wird.

7.1 Visualisierung einer virtuellen Messe

Die Messe befindet sich auf einem Platz, der von Marmorsäulen begrenzt ist. Sie besteht aus sechs Messeständen, die entweder noch frei oder bereits vermietet sind. Der Anwender hat die Möglichkeit, die Messe virtuell zu begehen und sie aus verschiedenen Perspektiven zu betrachten. Die verschiedenen Sichten kann er mit Hilfe von Navigationselementen wie Zoomen, Verschieben oder Rotieren, einstellen. Die verschiedenen Navigationsmöglichkeiten werden durch Betätigung der Maus ausgelöst. Es gibt darüber hinaus verschiedene vordefinierte Betrachtungspositionen (engl. viewpoints), die er über eine Liste auswählen kann.

Freie Messestände sind mit einem Schild (auf dem "Frei" steht) markiert und vermietete Messestände bestehen aus Tisch und Stühlen sowie einem frei schwebenden Firmennamen und einem Schild, auf dem das Firmenlogo abgebildet ist.

Der Betrachter kann zwischen einer Darstellung der Messe mit allen, mit den vermieteten oder mit den freien Messeständen wählen. Bei diesen Ansichten befindet sich über den nicht belegten Messeständen jeweils eine frei schwebende Standnummer und über den vermieteten der frei schwebende Firmenname. Die folgende Abbildung zeigt die Darstellung der Messe mit allen Messeständen:

ABBILDUNG 15: Ansicht der Messe mit allen Ständen

Neben der Gesamtansicht der kompletten Messe hat der Anwender die Möglichkeit, alle Messestände einzeln zu betrachten. Handelt es sich in diesem Fall um einen freien Messestand, so kann er ihn mieten durch Angabe eines Firmennamens und einer URL die angibt, wo sich das auf einem Fileserver gespeicherte Firmenemblem befindet. Die folgende Abbildung zeigt die Einzelansicht eines freien Messestandes:

ABBILDUNG 16: Ansicht eines freien Messestands

Bereits vermietete Messestände kann der Anwender in der Einzelansicht durch den Button "Stand löschen" wieder frei geben. Die folgende Abbildung zeigt die Ansicht eines einzelnen, bereits vermieteten Messestandes:

ABBILDUNG 17: Ansicht eines vermieteten Stands

7.2 Aufbau der virtuellen Messe

Alle Daten für die Messeanwendung sind in fünf Tabellen gespeichert. Die Spalten einer Tabelle werden mit Hilfe eines *Row Data Type* festgelegt (s. Kapitel 2.2.2). In der Tabelle `messe` sind die spezifischen Informationen zu einem Messestand abgelegt. Dies sind im einzelnen die Standnummer, der Status (vermietet/frei), der Firmenname des Mieters, die URL für das Firmenlogo, die Position des Messestandes in der Halle und die Position des Firmenschildes:

```
create row type messe_t
(
     standNr       varchar(254),
     vermietet     boolean,
     mieter        lvarchar,
     ImageURL      lvarchar,
     xPosition     real,
     yPosition     real,
     zPosition     real,
     xPosSign      real,
     yPosSign      real,
     zPosSign      real
);
```

```
create table messe of type messe_t
(
     primary key(standNr),
     check(standNr is not null),
     check(halle is not null)
);
```

Alle HTML-Seiten sind in der Tabelle `web_pages` im Feld `page` gespeichert. Jede einzelne HTML-Seite wird durch eine eindeutige ID identifiziert, mit der man die entsprechende Seite später durch eine URL referenzieren kann. Das Feld `descr` enthält eine optionale kurze Beschreibung der jeweiligen HTML-Seite:

```
create row type web_pages_t
(
     ID varchar(254),
     descr lvarchar,
     page  html
);
create table web_pages of type web_pages_t
(
     primary key (ID),
     check (ID is not null)
);
```

Alle graphischen Objekte, die Bestandteil einer HTML-Seite sind, werden in der Tabelle `web_obj` als BLOBs im Feld `obj` gespeichert. Sie werden ebenfalls durch eine eindeutige ID identifiziert:

```
create row type web_obj_t
(
        ID  varchar(254),
        obj blob
);
create table web_obj of type web_obj_t
(
        primary key(ID),
        check(ID is not null)
);
```

In der Tabelle `vrml_init` werden alle Teilszenen im Feld `vrml` gespeichert, die man für die Generierung kompletter VRML-Szenen benötigt. Die hier gespeicherten Teilszenen sind Bestandteile komplexer Szenen und wegen dieser logischen Trennung in einer eigenen Tabelle untergebracht. Über eine eindeutige ID können sie wie die HTML-Seiten referenziert werden:

```
create row type vrml_init_t
(
     ID     varchar(254),
     vrml   vrml
);
create table vrml_init of type vrml_init_t
(
     primary key(ID),
     check(ID is not null)
);
```

In der Tabelle `vrml_scenes` sind alle kompletten, aber noch nicht expandierten VRML-Szenen im Feld `scene` gespeichert. Sie werden ebenfalls durch eine eindeutige ID identifiziert:

```
create row type vrml_scenes_t
(
     ID varchar(254),
     scene  vrml
);
create table vrml_scenes of type vrml_scenes_t
(
     primary key (ID),
     check (ID is not null)
);
```

Jeder Messestand ist als eine einzelne VRML-Szene in der Tabelle `vrml_scenes` gespeichert und gehört einem Tupel der Tabelle `messe` an. Werden nun Daten eines Messestandes in der Tabelle `messe` gelöscht oder aktualisiert, wird durch Verwendung von Triggern sichergestellt, daß die VRML-Szene in der Tabelle `vrml_scenes` ebenfalls geändert wird. Wird z.B. ein Messestand vermietet, so wird die VRML-Szene generiert und in der Tabelle `vrml_scenes` gespeichert.

Die Generierung der VRML-Szenen aus den Teilszenen, die in der Tabelle `vrml_init` gespeichert sind, erfolgt durch eine neu eingeführte SQL-Funktion:

```
create function CreateStand(standNr varchar(254), vermietet boolean)
returning vrml
```

Ist der Parameter `vermietet` TRUE, so wird ein vermieteter Messestand und im anderen Fall (FALSE) ein freier Messestand generiert.

Die in der Datenbank gespeicherten VRML-Szenen werden erst bei einer Online-Anfrage expandiert. Der Vorteil von online-expandierten VRML-Szenen besteht darin, daß sie den höchsten Aktualitätsgrad haben. Es müssen keine bereits expandierten VRML-Szenen in der Datenbank gespeichert werden, was je nach Anwendung viel Speicherplatz sparen kann.

Man könnte aber auch häufig angefragte VRML-Szenen, wie z.B. die Messeübersichten, bereits expandiert in der Datenbank speichern. Dies entlastet den Server bei vielen Anfragen und wirkt sich positiv auf die Antwortzeit aus. Man muß aber dafür sorgen, daß u.a. mit Hilfe eines Triggers bei Veränderungen in der Datenbank die VRML-Szenen aktualisiert werden.

Alle VRML-Szenen werden mit Hilfe des Web-DataBlades in die entsprechenden HTML-Frames eingeblendet (s. a. Kapitel 5.3.3):

```
...
<?MIVAR>$(HTTPHEADER,content-type,x-world/x-vrml)<?/MIVAR><?MISQL
SQL="select VRMLExpand(scene)::clob from vrml_scenes where
ID=\'HALLFULL\';">$1<?/MISQL>
...
```

7.3 Validierung des SSI-Knotens

Der Abbildungsmechanismus wird durchgeführt, wenn man einen einzelnen Messestand als VRML-Szene anfordert. Im Falle eines vermieteten Messestandes wird der Name der Firma und das Firmenemblem mit Hilfe des SSI-Knotens auf die Felder abgebildet:

```
...
sqlStatement "SELECT mieter INTO Company.string FROM messe WHERE
standNr='A1';"
...
sqlStatement "SELECT ImageURL INTO Logo.url FROM messe WHERE
standNr='A1';"
...
```

Der Templatemechanismus wird bei den Hallenübersichten verwendet. Jeder Messestand repräsentiert eine Instanziierung einer vorgegebenen Teilszene. Jeder instanziierte vermietete Messestand erhält mit Hilfe des SSI-Knotens den Namen der Firma, das Firmenemblem als Bild, die Position des Schildes und die Position des Messestandes in der Messe. In diesem Fall handelt es sich um eine komplexe Teilszene mit fast 900 Zeilen VRML-Quellcode:

```
...
sqlStatement "Select xPosition, yPosition, zPosition, xPosSign,
yPosSign, zPosSign, mieter, ImageURL Into Messestand.translation,
Sign.translation, Company.string, Logo.url from messe where
vermietet='t';"
...
```

Jeder instanziierte leere Messestand erhält lediglich die Standnummer und seine Position in der Messe:

```
...
sqlStatement "Select xPosition, yPosition, zPosition, standNr Into
Stand.translation, StandNr.string from messe where vermietet='f';"
...
```

Als Beispiel für die Einbettung multimedialer Daten wurde das Firmenemblem verwendet. Es wird mit Hilfe des Zugriffs auf eine in der Datenbank gespeicherte URL, die auf eine Graphik auf einem Fileserver verweist, eingebunden (s. a. Kapitel 5.4.4.1). Da man nicht voraussetzen kann, daß alle auf der Messe präsentierten Firmen Zugriff auf die gleiche Datenbank haben, macht es keinen Sinn die Firmenembleme in einer Datenbank zu speichern.

Neben diesem Anwendungsszenario wurde der SSI-Knoten mit allen anderen zulässigen Datentypabbildungen (s. Kapitel 5.4.1.1) und einer großen Anzahl von Kombinationsmöglichkeiten validiert.

8 Perspektiven

Das VRML-DataBlade muß zunächst um die in Kapitel 5.1.1.3 angeführten fehlenden Funktionalitäten der VRML-Klassenbibliothek erweitert werden. Dann kann unter anderem der SSI-Knoten, wie in der Spezifikation vorgesehen, als Prototyp deklariert werden und man muß nicht länger auf Skriptknoten verzichten.

Im nächsten Schritt können die im Kapitel 6.4 angegebenen noch fehlenden Komponenten des SSI-Knotens integriert werden. Durch die Korrektur der VRML-Klassenbibliothek bezüglich der Verwendung der ROUTE-Anweisung, wäre eine vollständige Umsetzung der VRML2.0-Spezifikation erreicht.

Eine zusätzliche Erweiterung betrifft die Implementierung eines eigenen CGI-Programms (VRML-Driver) für das VRML-DataBlade (s. a. Kapitel 5.3.3). Mit einem solchen CGI-Programm hat man die Möglichkeit das VRML-DataBlade autonom einzusetzen, d.h. ohne Unterstützung durch das Web-DataBlade und dessen Webdrivers.

Im Rahmen einer Implementierung der bisher fehlenden Funktionalität für das Feld `connection` des SSI-Knotens (s. a. Kapitel 5.4.1) lassen sich Sicherheitsaspekte, wie z.B. ein Zugriffsschutz auf VRML-Szenen, die in einer Datenbank gespeichert sind, integrieren.

Zu den Erweiterungen größeren Umfangs zählen die in Kapitel 4.2.2 beschriebenen Konzepte. Durch die Integration eines "High-Level"-Knotens, der Embedded SQL ermöglicht, und eines Triggermechanismus in das VRML-DataBlade, wäre nahezu das vollständige Spektrum der Verbindungsmöglichkeiten zwischen VRML und Datenbankensystemen abgedeckt.

Das VRML-DataBlade stellt dann ein ideales Werkzeug für die Realisierung datenbankbasierter dreidimensionaler Informationsvisualisierungen im Internet dar.

ANHANG A: Abbildungen

ANHANG B: Literaturverzeichnis

[ADMIN98] ***Administrator's Guide for Informix Dynamic Server™*** Version 7.3, Informix Press 1998

[AMES97] Andrea l.Ames,David R.Nadeau and John L.Moreland: ***VRML 2.0 Sourcebook Second Edition*** John Wiley & Sons, Inc. 1997

[BUCH96] A. Buchmann: Skript zur Vorlesung "*Datenbanksysteme I*" WS 96/97 TU-Darmstadt

[COD70] E.F.Codd: ***A Relational Model for Large Shared Data Banks*** Communications of the ACM, Vol. 13, No.6, Seite 377-387, Juni 1970

[CONSORTIUM] Http://www.vrml.org/: Offizielle WWW-Informationseite über das VRML-Consortium

[CT 5/97] Gerd Wagner: ***Jenseits von Schema F, Datenmodelle – Strickmuster für Datenbanken*** C'T 5/1997, Seite 276-282, Verlag Heinz Heise, Hannover

[DBDK97] ***DataBlade Developers Kit User's Guide*** Version 3.4, INFORMIX Press 1997

[DBWORK] Http://www.vrml.org/WorkingGroups/dbwork/charter.html: Offizielle WWW-Informationsseite über die VRML-Database Working Group

[ELMASRI94] Ramez Elmasri; Shamkant B. Navathe: ***Fundamentals of Database Systems (Second Edition)*** The Benjamin/ Cummings Publishing Company, Inc. 1994

[LEIS97] Martin Leissler, Matthias Hemmje: ***Portierung und Erweiterung der Relevanzkugel- metapher des LyberWorld-Werkzeuges um interaktive***

Informationsvisualisierungsmechanismen GMD-Studien Nr. 321, September 1997, GMD - Forschungszentrum Informationstechnik, Sankt Augustin

[NEU96] E.J.Neuhold, K. Aberer, W. Klas, T. Rakow: Skript zur Vorlesung "***Multimediale und Objekt-Orientierte Datenbanken***" WS 95/96 TU-Darmstadt

[MANIFESTO89] M. Atkinson, D. DeWitt, D. Maier, F. Bancilhon, K.Dittrich, S. Zdonik: ***The Object Oriented Database System Manifesto*** in W. Kim, J.Nicholas, und S. Nishio (Hrsg.), Proceedings of DOOD'89, North Holland, 1990

[ODMG93] Francois Bancilhon, Guy Ferran: ***ODMG93: Object Database Standard*** Bulletin of the Technical Committee of Data Engineering, December 94, Vol.17 No.4

[OPENINVENTOR] Http://www.sgi.com/Technology/Inventor/: Offizielle WWW-Informationsseite über OpenInventor von Silicon Graphics Inc.

[RISSE97] Thomas Risse, Matthias Hemmje: ***Entwurf und Implementierung einer VRML Bibliothek für ein objektrelationales Datenbanksystem auf der Basis des VRML 2.0 Standards*** GMD-Studien Nr. 324, September 1997, GMD – Forschungszentrum Informationstechnik, Sankt Augustin

[SQL3] Http://www.jcc.com/sql_stnd.html: Offizielle WWW-Informationsseite und vorläufige Spezifikation über SQL3

[SQLREF97] ***Informix Guide To SQL: Reference*** Version 9.1, INFORMIX Press 1997

[SQLTUT97] ***Informix Guide To SQL: Tutorial*** Version 9.1, INFORMIX Press 1997

[STO94] Michael Stonebraker: ***Object-relational DBMSs: the next great wave; 1996*** Morgan Kaufmann Publishers, Inc.,

San Francisco

[STRO92] Bjarne Stroustrup: ***The C++ Programming Language, 2nd Edition*** Addison-Wesley, 1992

[TAN92] Andrew S. Tanenbaum: ***Modern Operating Systems*** Prentice Hall Inc., Simon & Schuster Company 1992

[VAG] Http://vag.vrml.org/: Offizielle WWW-Informationsseite über die VRML Architecture Group

[VRML1.0] Http://www.vrml.org/VRML1.0/vrml10c.html: Die Spezifikation zu VRML 1.0

[VRML2.0] Http://www.vrml.org/Specifications/VRML2.0.old/FINAL/spec/index.html: Die Spezifikation zu VRML 2.0

[VRMLVIEW] Http://www.vrml.org/Specifications/VRML2.0.old/FINAL/Overview.html#Summary: Eine Übersicht der Fähigkeiten zu VRML 2.0

[VRML97] Http://www.vrml.org/Specifications/VRML97/: Die Spezifikation zu VRML 97

[VRMLCHANGE] Http://cosmosoftware.com/developer/moving-worlds/spec.DIS/index.html#Summary: Liste der Unterschiede zwischen VRML 2.0 und VRML97

ANHANG C: Glossar

- **API (Application Programming Interface):** Eine Softwareschnittstelle, über die man auf Funktionen und Klassen eines Betriebssystems oder einer Applikation zugreifen kann. Eine solche Applikation kann z.B. ein Datenbank-Managementsystem sein.
- **ASCII (American Standard Code for Information Interchange):** ASCII ist das weitest verbreitete Format für Textdateien. In einer ASCII-Datei werden jedes alphabetische oder numerische Zeichen sowie alle Sonderzeichen durch eine 7-stellige Binärzahl dargestellt. Es sind 128 verschiedene Zeichen definiert.
- **BLOB (Binary Large Object):** Großvolumiges Binärdatenobjekt, welches verwendet wird, um Multimediadaten (z.B. Videos) in einer Datenbank zu speichern.
- **Browser:** Ein Client-Programm mit dem man Informationen aus dem WWW (z.B. HTML-Dokumente) darstellen, lesen und hören kann. Es benutzt das HTTP, um Anfragen abhängig von den jeweiligen Benutzereingaben über das Internet an Webserver zu leiten. Die populärsten Browser sind zur Zeit der Netscape Communicator und der Microsoft Internet Explorer.
- **CGI (Common Gateway Interface):** Dies ist eine standardisierte Schnittstelle, die es erlaubt, im WWW über einen Webserver auf bestimmte Programme zuzugreifen, die z.B. mit einem Datenbankserver kommunizieren und Ergebnisse an den Webserver zurücksenden.
- **CLOB:** Spezieller BLOB, der Daten speichert, die in Form von ASCII-Texten vorliegen.
- **Datenbank:** Eine einheitliche beschriebene Darstellung eines Weltausschnittes mittels diskreter Daten auf externen und persistenten Speichermedien. Typischerweise werden dazu Magnetplatten verwendet.
- **Datenbankmanagementsystem (DBMS):** Ein Softwaresystem, das eine einheitliche Beschreibung und sichere Bearbeitung einer Datenbank ermöglicht. Es garantiert die Korrektheit der Daten durch Einhaltung definierter Konsistenzregeln. Es garantiert die Korrektheit beim konkurrierenden Zugriff im Mehrbenutzerbetrieb und ermöglicht eine feinkörnige Zugriffskontrolle. Beim fehlerhaften Ablauf von Transaktionen oder bei Systemabstürzen stellt es den letzten konsistenten Zustand der Datenbank wieder her.
- **Datenbanksystem (DBS):** Komposition aus Datenbanken und einem DBMS.

- **Fremdschlüssel:** Ein oder mehrere Attribute eines Datensatzes, die einen Datensatz in einer anderen Tabelle eindeutig referenzieren. Fremdschlüssel werden gebraucht, um referentielle Integrität zu bewahren. Die referentielle Integrität ist die Garantie, daß die Einträge, die ein Fremdschlüssel referenziert, immer vorhanden sind.
- **GIF (Graphics Interchange Format):** Ein Dateiformat für Graphikdaten. Es wurde von *CompuServe, Inc.* entwickelt und ist auf vielen verschiedenen Plattformen und Systemen verfügbar. GIF ist neben JPEG eines der am häufigsten verwendeten Graphikformate und kann als Inline-Graphik in HTML-Dokumente eingebunden werden.
- **HTML (Hypertext Markup Language):** Die Auszeichnungssprache des World Wide Web (WWW). Es enthält Sprachelemente für den Entwurf von Hypertext-Dokumenten, die mit Browsern dargestellt werden können. Mit HTML werden die Struktur und die Hyperlink-Informationen in einem Dokument beschrieben.
- **HTTP (Hypertext Transfer Protocol):** Dieses Protokoll beschreibt einen festdefinierten Satz von Regeln, mit denen ein Client und eine Server während einer HTML-Sitzung kommunizieren können. Es wird vornehmlich für den Austausch von Daten (z.B. Texte, Bilder und Videos) im WWW benutzt. Es basiert auf TCP/IP.
- **ISO (International Organization for Standardization):** ISO wurde 1946 gegründet und ist eine Föderation nationaler Standardisierungskörperschaften aus über 100 Ländern. Sie fördert Normierungen auf internationaler Ebene, um den Austausch von Gütern und Dienstleistungen zu erleichtern.
- **Java:** Eine objektorientierte Programmiersprache, die 1995 von *SUN Microsystems* entwickelt wurde. In Java geschriebene Programme werden nicht in Maschinencode umgewandelt, sondern in einen sogenannten "Bytecode" kompiliert. Dieser Code kann dann auf unterschiedlichen Betriebssystemen und Rechnerarchitekturen mit Hilfe einer virtuellen Java-Engine ausgeführt werden. Java-Programme genießen den Ruf, plattformunabhängig zu sein.
- **JavaScript:** Eine für das World Wide Web entwickelte Skriptsprache. Sie wurde von Netscape entwickelt und kann in HTML-Seiten eingebettet werden, um diese mit zusätzlichen Funktionalitäten zu erweitern.
- **JDBC (Java Database Connectivity):** Eine API-Spezifikation, mit der man in Java geschriebene Programme mit Daten in populären Datenbanken verknüpfen kann. Die API ermöglicht einen Zugriff auf diese Daten mit Hilfe von SQL-Anweisungen.
- **JPEG (Joint Photographic Experts Group):** Es steht für a) ein Standardisierungs-Gremium, b) eine Methode der Datenkompression und c) ein graphisches Datei-

format. Das Gremium entstand innerhalb der International Standards Organization (ISO) mit dem Ziel, einen Standard für die Übertragung von Bilddaten zu entwikkeln. Die Resultate waren ein sehr gutes Datenkompressionsverfahren und verschiedene, eng miteinander verbundene Datenformate. JPEG-Dateien können Fotografien, Still-Video oder andere komplexe Graphiken enthalten. JPEG ist neben GIF eines der am häufigsten verwendeten Graphikformate und kann als Inline-Graphik in HTML-Dokumente eingebunden werden.

- **Large Object Handle:** Eine opake Datenstruktur, die den Ort des in einer Datenbank gespeicherten Large Objects eindeutig identifiziert.
- **Metadaten:** Dies sind Daten, die andere Daten beschreiben. Hierzu zählen z.B. die in HTML verwendeten Tags, die die Dokumentenstruktur beschreiben, aber in der Dokumentendarstellung nicht zu sehen sind.
- **MIME (Multipurpose Internet Mail Extensions Protocol):** Eine Erweiterung des ursprünglichen Internet E-Mail-Protokolls. Es teilt verschiedene Datentypen in Haupt- und Untergruppen auf. Die Verwendung dieses Protokolls erlaubt einem Browser, unterschiedliche Dateiformate zu erkennen und korrekt zu verarbeiten. Die MIME-Hauptgruppe definiert, um welchen Datentyp es sich handelt, also z.B. Graphik, Audio oder Video, die Untergruppe beschreibt den Typ genauer.
- **MPEG (Motion Picture Experts Group):** Es steht für a) ein Standardisierungs-Gremium, b) eine Methode der Datenkompression und c) ein graphisches Dateiformat. Die Hauptanwendung von MPEG liegt in der Speicherung und Kompression von Audio- und Videodaten für Multimedia-Systeme. Es gibt mittlerweile verschiedene Versionen von MPEG-Formaten, die jeweils abhängig von der Übertragungsart eingesetzt werden.
- **Multimedia:** Dies ist vereinfacht ausgedrückt ein Sammelbegriff für Dokumente, die verschiedene Datentypen wie beispielsweise Text, Audio und Video enthalten.
- **Primärschlüssel:** Ein oder mehrere Attribute, die jeden Datensatz (Tupel) in einer Tabelle eindeutig referenzieren können.
- **Plug-in:** Ein Plug-in erweitert einen Browser um neue Funktionalitäten, die in der Standardversion nicht verfügbar sind. Mit dieser Technologie hat man die Möglichkeit, die Darstellung neuer multimedialer Datenformate wie z.B. Audio, Video oder VRML auf einfache Art und Weise in einen Browser zu integrieren.
- **Tags:** Mit Hilfe von Tags werden z.B. in HTML die Struktur und die Darstellung von Informationen festgelegt. Tags werden meist durch in spitze Klammern eingeschlossene Bezeichner dargestellt.
- **TCP/IP (Transmission Control Protocol/Internet Protocol):** Ein Kommunikationsprotokoll, das sowohl im Internet als auch in privaten Netzwerken wie Intra-

nets oder Extranets verwendet wird. Es besteht aus zwei Schichten: Die höhere Schicht (TCP) verwaltet die Aufteilung zu übertragender Nachrichten in Pakete und generiert beim Empfang von Paketen die ursprüngliche Nachricht. Die untere Schicht (IP) übernimmt die Verwaltung der Adressen eines jeden Pakets, so daß es immer zum richtigen Ziel gesendet wird.

- **Thread:** Ein Thread ist ein sogenannter Leichtgewichtsprozeß. Er hat die gleichen Eigenschaften wie ein gewöhnlicher Prozeß mit dem Unterschied, daß man durch Threads mehrere Kontrollflüsse innerhalb desselben Adressraums ermöglichen kann, d.h. Threads teilen sich im Gegensatz zu Prozessen den gleichen Adressraum.
- **Trigger:** Eine vom Benutzer definierte, deklarative Bedingung sowie eine dazugehörige Aktion, die ausgeführt wird, wenn die Bedingung erfüllt ist. Trigger werden u.a. benötigt, um die Integrität eines Datenbestandes in einer Datenbank zu bewahren.
- **URL (Uniform Resource Locator):** Dies ist die Adresse eines Dokuments im World Wide Web. Die Adresse ist Teil des Links auf ein anderes Dokument und wird von der Client-Software interpretiert, um eine Verbindung zum entsprechenden Server aufzubauen. Eine URL enthält den Namen des erforderlichen Kommunikationsprotokolls, den Domänenname, der einen bestimmten Computer im Internet identifiziert sowie eine hierarchische Beschreibung der Position der zu übertragenden Datei.
- **World Wide Web (WWW):** Ein Hypertext-basiertes System für die Suche nach und den Zugriff auf Internet-Ressourcen.
- **X Window System:** Ein netzbasiertes Window-System, das ursprünglich am Massachusetts Institute of Technology (MIT) entwickelt wurde. X-Windows oder kurz X wird häufig auf UNIX-Systemen eingesetzt.

ANHANG D: Index