Ein abstraktes Komponentenframework für interaktive 3D-Grafikanwendungen

Andreas Braig, Raimund Dachselt
TU Dresden, Fakultät Informatik,
Heinz Nixdorf Stiftungslehrstuhl für Multimediatechnik
Andreas.Braig@objectfab.de, dachselt@irz.inf.tu-dresden.de

Keywords: Virtual Environments, 3D-Interfaces, 3D-Widgets, Software Components, VRML, UML, XML.

Abstract

Even though 3D toolkits and technologies are rapidly evolving, the development of three-dimensional applications is still very complicated and hard work due to format dependencies, missing standards and lack of software engineering support. Component-based development seems to be promising, where interactive 3D software will be composed of flexible and reusable 3D interface components. We present an abstract component framework, which allows the toolkit-independent modeling of components for 3D applications. Using an XML-based framework description language and XSL Transformations these descriptions can be transformed to "real" 3D formats like VRML, as shown in an example.

Einleitung

Im Bereich der aktuellen PC- und Unterhaltungstechnik ist gegenwärtig eine rasante Entwicklung leistungsfähiger 3D-Grafikhardware zu beobachten. Auch für neuartige, portable Geräte zeichnen sich bereits erste Entwicklungen 3D-fähiger Hardware ab. Durch diese Trends sind Anwendungen mit dreidimensionaler Grafik künftig nicht mehr nur auf die Forschungs- und Industriedomäne beschränkt, auch neue Anwendungsklassen für den Heimanwender-Bereich entstehen. Insbesondere E-Commerce- und Serviceanwendungen, Produktpräsentationen, Lehr-/Lernsysteme sowie Unterhaltungssoftware werden vom Einsatz interaktiver 3D-Grafik profitieren.

Zur Realisierung derartiger Anwendungen wurden in den letzten Jahren verschiedene 3D-Technologien entwikkelt, vom Open Inventor 3D-Toolkit für Grafikworkstations über den VRML97-Standard für 3D-Internetgrafik bis hin zu schlankeren, *streaming*-fähigen Webformaten wie Metastream. Obwohl gerade mit der Entwicklung des VRML-Nachfolgeformates X3D ein vielversprechender, modularer und mit anderen Standards interoperabler Ansatz vorliegt, ist anzunehmen, daß sich nicht eine einzelne Technologie für alle Plattformen durchsetzen wird, sondern verschiedene Technologien koexistieren werden.

Bisherige Anwendungsentwicklung auf diesem Gebiet ist also geprägt von Formatabhängigkeit und stellt aufgrund fehlender Standards, Methodologien und softwaretechnologischer Unterstützung einen aufwendigen Prozeß dar. Komponentenbasierte Softwareentwicklung, die generell Vorteile verspricht, könnte deshalb auch für interaktive Grafikanwendungen das Mittel der Wahl sein. Die Realisierung virtueller Umgebungen sollte durch die Wiederverwendung funktionaler Bestandteile deutlich erleichtert werden. Insbesondere bei dokumentenbestimmten Anwendungen mit dreidimensionaler Benutzungsschnittstelle, wie z.B. einem 3D-Modellierungswerkzeug, sind wiederverwendbare Schnittstellenelemente von hohem Nutzen. Solche 3D-Widgets kapseln Geometrie und Interaktionsverhalten [1]. Ziel unserer Forschung ist deren Spezifikation und Implementierung in Form von flexiblen, wiederverwendbaren Komponenten, die sich zu komplexen 3D-Anwendungen zusammensetzen lassen.

Für einen derartig vereinfachten Erstellungsprozeß von 3D-Grafik gibt es bisher jedoch noch kein komponentenbasiertes 3D-Toolkit, mit dem sich Grafikapplikationen aus vorgefertigten Bausteinen leicht zusammensetzen ließen. Um von der Entwicklung konkreter Toolkits unabhängig zu sein, wurde ein Framework entworfen, das es gestattet, Elemente von 3D-Grafikanwendungen auf abstraktem Niveau als Softwarekomponenten zu modellieren.

Dieses mit der *Unified Modeling Language* (UML) beschriebene Komponentenframework wird nach einem Abschnitt zu verwandten Arbeiten im Hauptteil des Artikels vorgestellt und am Beispiel eines Ringmenü-Widgets illustriert. Um die Entwicklung, Portierung und Anpassung von Anwendungen auf verschiedene Technologien zu unterstützen, lassen sich die entwickelten Modelle mittels definierter Abbildungen auf Implementationen mit konkreten 3D-Grafiksystemen übertragen. Für VRML97 wird eine solche Abbildung unter Nutzung von XML und XSL *Transformations* vorgestellt.

Verwandte Arbeiten

Döllner und Hinrichs stellen in [2] eine Architektur für interaktive, animierte 3D-Widgets vor, bei der geometrische Struktur und Interaktionsverhalten in separaten Geometrie- und Verhaltensgraphen modelliert werden. Diese Trennung diente auch als Grundlage für die Graphen-Repräsentation im hier vorgestellten Framework.

Im Rahmen des Spezifikationsprozesses für X3D [12] wurde die *Jamal Component Architecture* für die Integration von Softwarekomponenten vorgeschlagen [4], [5]. Die Architektur definiert ein Format für die deklara-

tive Anwendungsentwicklung und identifiziert zudem Isomorphismen zwischen Schnittstellenbeschreibungen in VRML, Java und CORBA. Solche Abbildungsmöglichkeiten sind für die konkrete Umsetzung eines abstrakten Komponentenframeworks von Bedeutung.

Schönhage et al. haben Softwarekomponenten für 3D-Visualisierungen betriebswirtschaftlicher Sachverhalte auf Basis von VRML und CORBA sowie alternativ mit Java und JavaBeans vorgestellt [6].

Das von Geiger et al. entwickelte System *i4D* dient dem *Rapid Prototyping* von Anwendungen mit animierter, interaktiver 3D-Grafik [7]. *I4D* basiert auf dem Konzept von *Actors*, die aktive Objekte der 3D-Szene kapseln und mittels *Actions* gesteuert werden. Durch eine Schichtenarchitektur kann *i4D* als Gesamtsystem auf verschiedene Plattformen und Grafiktoolkits portiert werden. Zusätzliche Funktionalität läßt sich mit Softwarekomponenten z.B. via *COM* integrieren.

Dörner und Grimm stellen mit *3D Beans* eine Lösung vor, 3D-Webinhalte unter Nutzung der Technologien JavaBeans und Java3D komponentenbasiert zu realisieren [3].

Die erwähnten Arbeiten enthalten interessante Ansätze zur Realisierung komponentenbasierter 3D-Grafik. Sie setzen jedoch auf der Kombination konkreter 3D-Grafiksysteme und Komponentenarchitekturen auf, während das hier vorgestellte Framework einen flexiblen und allgemeineren Rahmen für die abstrakte Modellierung von Softwarekomponenten in interaktiven 3D-Grafikanwendungen bereitstellt. Die erwähnten Systeme sind jedoch mögliche Zielarchitekturen für die Abbildung des Frameworks.

Spezifikation des Komponentenframeworks

Charakterisierung und Modell

Das Framework erlaubt die Modellierung von Elementen interaktiver 3D-Grafikanwendungen als abstrakte Softwarekomponenten unabhängig von konkreten 3D-Grafiktoolkits bzw. Komponententechnologien. Die entwickelten Modelle lassen sich mittels definierter Abbildungen auf konkrete 3D-Grafiksysteme übertragen. Das abstrakte Framework definiert ein Objektmodell für die Elemente des Frameworks, eine Szenengraphstruktur für die Repräsentation der Objekte, Interaktionen und Animationen der 3D-Szene sowie Dienste der Laufzeitumgebung für die Darstellung und Manipulation der 3D-Szene. Das Framework ist in UML (Teildefinition siehe Abb. 1) auf der Abstraktionsebene eines Entwurfsmodells spezifiziert. Dies läßt ausreichende Flexibilität für Abbildungen auf eine breite Palette von 3D-Grafiksystemen.

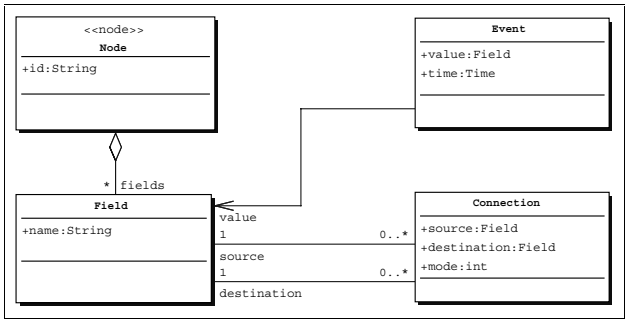

Abbildung 1: Objektmodell des Frameworks

Das Objektmodell definiert Knoten, Felder, Verbindungen und Ereignisse zur Modellierung des Szenengraphen. Knoten sind die grundlegenden Elemente der Szenengraphstruktur. Sie repräsentieren Funktionalität zur Modellierung der Darstellung und des Verhaltens von 3D-Objekten. Erweiterungen des Frameworks um zusätzliche Funktionalität durch Komponenten werden ebenfalls als Knoten modelliert. Die übrigen Elemente (Felder, Verbindungen und Ereignisse) können nicht erweitert werden. Knoten können durch einen Bezeichner identifiziert sein und damit mehrfach referenziert werden. Ebenso lassen sie sich anonym instanziieren und damit nur exklusiv referenzieren. Felder repräsentieren die Eigenschaften von Knoten. Die Felder eines Knoten werden durch Namen identifiziert und bilden die Schnittstelle zur Manipulation von Knoten. Der Wert eines Feldes kann statisch bei Instanziierung des Knotens gesetzt werden oder dynamisch zur Laufzeit der Anwendung mittels Verbindungen. Verbindungen definieren Datenflüsse, die Wertveränderungen von Feldern als Ereignisse von einem Quell- zu einem Zielfeld propagieren. Felder können auf das Aussenden bzw. den Empfang von Ereignissen beschränkt sein oder nur statisch belegt werden. Dies schränkt die Möglichkeit zur Deklaration von Verbindungen ein. Diese Charakteristika werden als UML Stereotypen definiert.

Das Objektmodell ist äquivalent zu dem von VRML97 und bildet die Konzepte der *Properties* bzw. *Slots* und des *Connection-oriented Programming* [8] der komponentenbasierten Softwareentwicklung ab.

Für Felder werden Datentypen für Skalare, Vektoren, Zeichenketten, Referenzen auf Knoten und multimediale Objekte definiert, die in Tabelle 1 dargestellt sind. Dabei wird jeder Datentyp sowohl als einzelner Wert als auch als Liste (Array) von Werten definiert. Im Gegensatz zu VRML werden auch multimediale Objekte als Datentypen definiert, um die Darstellung der Objekte (z.B. als räumliche Schallquelle) von der Datenrepräsentation (z.B. als PCM-kodierte Samples) zu trennen.

Einwertige Felder	Mehrwertige Felder
NodeField	NodeArrayField
BooleanField	BooleanArrayField
IntegerField	IntegerArrayField
FloatField	FloatArrayField
StringField	StringArrayField
TimeField	TimeArrayField
ColorField	ColorArrayField
Vector2fField	Vector2fArrayField
Vector3fField	Vector3fArrayField
RotationField	RotationArrayField
ImageSourceField	ImageSourceArrayField
VideoSourceField	VideoSourceArrayField
AudioSourceField	AudioSourceArrayField

Tabelle 1: Datentypen für Felder

Repräsentation durch verschiedene Graphen

Unter Verwendung der Bestandteile des Objektmodells werden die statische Geometrie der 3D-Szene und das dynamische Verhalten der Objekte in der 3D-Szene separat modelliert.

Die räumliche und logische Struktur einer Szene sowie die geometrischen Objekte und ihre Darstellungseigenschaften werden durch einen hierarchischen *Geometriegraphen* beschrieben. Der Geometriegraph dient als Abstraktion für die Darstellung der 3D-Szene unabhängig von einer konkreten 3D-Grafikbibliothek. Das Framework definiert Knotentypen zur Strukturierung, Repräsentation von Geometrie, Darstellungseigenschaften, Audioquellen und Beleuchtung, die in Abbildung 2 aufgelistet sind. Die Struktur des Geometriegraphen wird durch die Knotentypen *Group*, *Transform* und *Switch* definiert, deren Feld *children* Referenzen auf untergeordnete Knoten enthält. Der Geometriegraph wird von der Laufzeitumgebung traversiert, die dabei auf die Felder der Knoten zugreift und daraus die Darstellung generiert.

```
Node
  GeometryGraphNode
    ChildrenNode
      Shape
      Group
        Transform
        Switch
      LightNode
        DirectionalLight
        PointLight
        Spotlight
      Sound
    PropertyNode
      Geometry
        IndexedFaceSet
        IndexedLineSet
        PointSet
      Appearance
        Material
        Texture
          ImageTexture
          MovieTexture
```

Abbildung 2: Klassenhierarchie der Geometriegraph-Knotentypen

Die Elemente des Geometriegraphen sind aus dem Szenengraphmodell von VRML97 [9] abgeleitet, jedoch auf die Kernbestandteile für die Repräsentation geometrischer Objekte und ihrer Struktur reduziert.

Interaktionen und Animationen von Objekten der 3D-Szene werden durch *Verhaltensgraphen* modelliert. Verhaltensgraphen realisieren Datenflüsse zwischen einer Quelle, die ein Feld des Geometriegraphen oder ein Wert eines Eingabegerätes sein kann, und einer Senke, die ein Feld des Geometriegraphen ist. So läßt sich der Geometriegraph und damit die Darstellung der 3D-Szene zur Laufzeit manipulieren. Die zwischen Quelle und Senke plazierten inneren Knoten des Verhaltensgraphen setzen eingehende Ereignisse in ausgehende um, indem sie den Wert des Ereignisses verändern oder selbständig neue Ereignisse erzeugen. Zwischen den Feldern der Knoten werden Verbindungen deklariert, die den Wert des Quellfeldes an das Zielfeld propagieren. Mit Verhaltensgraphen können komplexe Datenflüsse realisiert werden, wie sie von Jacob zur visuellen Modellierung von Interaktionen beschrieben werden [10].

Für Eingaben des Benutzers werden sogenannte Sensoren u.a. für Translationen und Rotationen mit unterschiedlich vielen Freiheitsgraden sowie für Text- und Symbol-Eingaben definiert. Das Konzept der Sensoren dient dazu, von konkreten Eingabegeräten zu abstrahieren. Die Laufzeitumgebung bindet dann die vorhandenen Eingabegeräte einer konkreten Systemkonfiguration an die Sensoren und realisiert zusätzliche Interaktionstechniken, um z.B. mit einem 2D-Eingabegerät eine 3D-Rotation auszuführen. Für die Bindung an Eingabegeräte werden Sensoren mit Geometrieknoten assoziiert. Selektiert der Benutzer z.B. mit einer Raycasting-Interaktionstechnik eine Geometrie, werden die damit assoziierten Sensoren aktiviert, d.h. an konkrete Eingabegeräte gebunden. Um Verhaltensgraphen zu realisieren, die Ereignisse nicht nur propagieren, sondern auch verarbeiten und manipulieren, muß der Anwendungsentwickler weitere Knoten implementieren, die eintreffende Ereignisse verarbeiten und über Veränderung ihrer Felder auf ausgehende Ereignisse abbilden.

Komponenten und Definition von 3D-Widgets

Um ein 3D-Grafiksystem um zusätzliche Funktionalität zu erweitern, können neue Knotentypen des Geometrie- oder Verhaltensgraphen als Softwarekomponenten implementiert werden. Die Schnittstelle einer solchen Komponente besteht aus Feldern nach den Konventionen des Objekt-Modells. Komponenten können beliebig komplexe Geometrie- und Verhaltensgraphen – oder auch nur einen der beiden – kapseln. Eine typische Gruppe von Komponenten stellen 3D-Widgets dar, die Geometrie und Interaktionsverhalten kapseln. Somit besteht ein 3D-Widget im vorgestellten Framework aus einem Geometriegraph, der Möglichkeiten zur Anpassung der Erscheinung des 3D-Widgets bietet, und einem Verhaltensgraph, der i.allg. aus mehreren, nicht zusammenhängenden Teilgraphen besteht, die einzelne Interaktionstechniken realisieren.

Beispielhaft wird die Modellierung eines Ringmenü-Widgets mit den Mitteln des Frameworks demonstriert. In einem Ringmenü sind dreidimensionale Objekte bzw. 3D-Icons um eine zentrale Achse angeordnet, wobei durch Rotation oder direkte Selektion das gewünschte Element in den Fokus gebracht und damit ausgewählt wird (s. Abb. 3). Der Geometriegraph des Ringmenü-

Widgets besteht aus den Menüelementen, die auf einer Drehbühne angeordnet sind, einer Drehachse und einem feststehenden Hintergrund. Die Wurzel des Geometriegraphen bildet ein *Transform*-Konten, durch den das komplette Widget in der 3D-Szene positioniert, rotiert und skaliert werden kann. Dem Wurzelknoten untergeordnet sind weitere *Transform*-Knoten, denen Subgeometriegraphen für Drehbühne, -achse und Hintergrund untergeordnet sind, wie in Abb. 4 gezeigt. Der Übersichtlichkeit halber sind in dieser Darstellung die Felder nicht als separate Objekte dargestellt. Ebenso wurden die Subgeometriegraphen weggelassen. Grau hinterlegt sind jene Subgeometriegraphen, die dynamisch beim Hinzufügen von Menüelementen erzeugt werden.

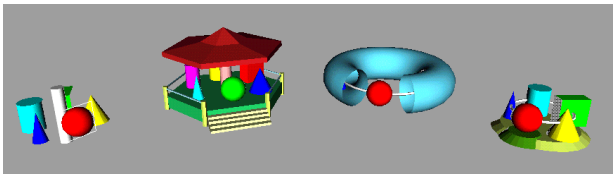

Abbildung 3: Verschiedene Ausprägungen eines Ringmenü-Widgets

Das Verhalten des Ringmenüs besteht aus der Rotation der Menüelemente um die Drehachse, der Selektion eines Menüelementes und dem Hinzufügen bzw. Entfernen von Menüelementen. Für die Modellierung der Rotation der Menüelemente wird ein Sensor verwendet, der Rotationen in einem Freiheitsgrad vom Eingabegerät des Benutzers aufnimmt, über eine Verbindung an einen Knoten weiterleitet, der den Rotationswert so begrenzt, daß die Drehbühne auf die Position eines Menüelementes einrastet, und über eine Verbindung den Geometriegraph der Drehbühne manipuliert. Für die Selektion von Menüelementen nimmt ein Sensor Tastendrücke auf, ein Verhaltensknoten bildet den Tastendruck auf den Index des gewählten Menüelementes ab, der von einem weiteren Verhaltensknoten auf eine Rotation der Drehbühne abgebildet wird. Dieser Verhaltensgraph muß für jedes Menüelement dynamisch erzeugt werden. Zum Hinzufügen und Entfernen von Menüelementen dient der Knoten *arrangeItems*, der Subgeometriegraphen erzeugt.

Die Komplexität dieser Modellierung läßt sich verbergen, in dem das 3D-Widget als Komponente gekapselt wird. Die Komponente wird als Knoten modelliert, der in seiner Schnittstelle ausschließlich die Felder des Geometrie- und Verhaltensgraphen deklariert, die für die Verwendung des 3D-Widgets in Anwendungen notwendig sind (s. Abb. 5).

Die Funktionalität von 3D-Widgets läßt sich in verschiedene Schnittstellen partitionieren, die im folgenden beschrieben werden.

Basisschnittstelle: Die Basisschnittstelle dient zur Integration des 3D-Widgets in den Geometriegraphen der Anwendung. Damit kann das Widget in der 3D-Szene positioniert, rotiert und skaliert werden. Über das Feld

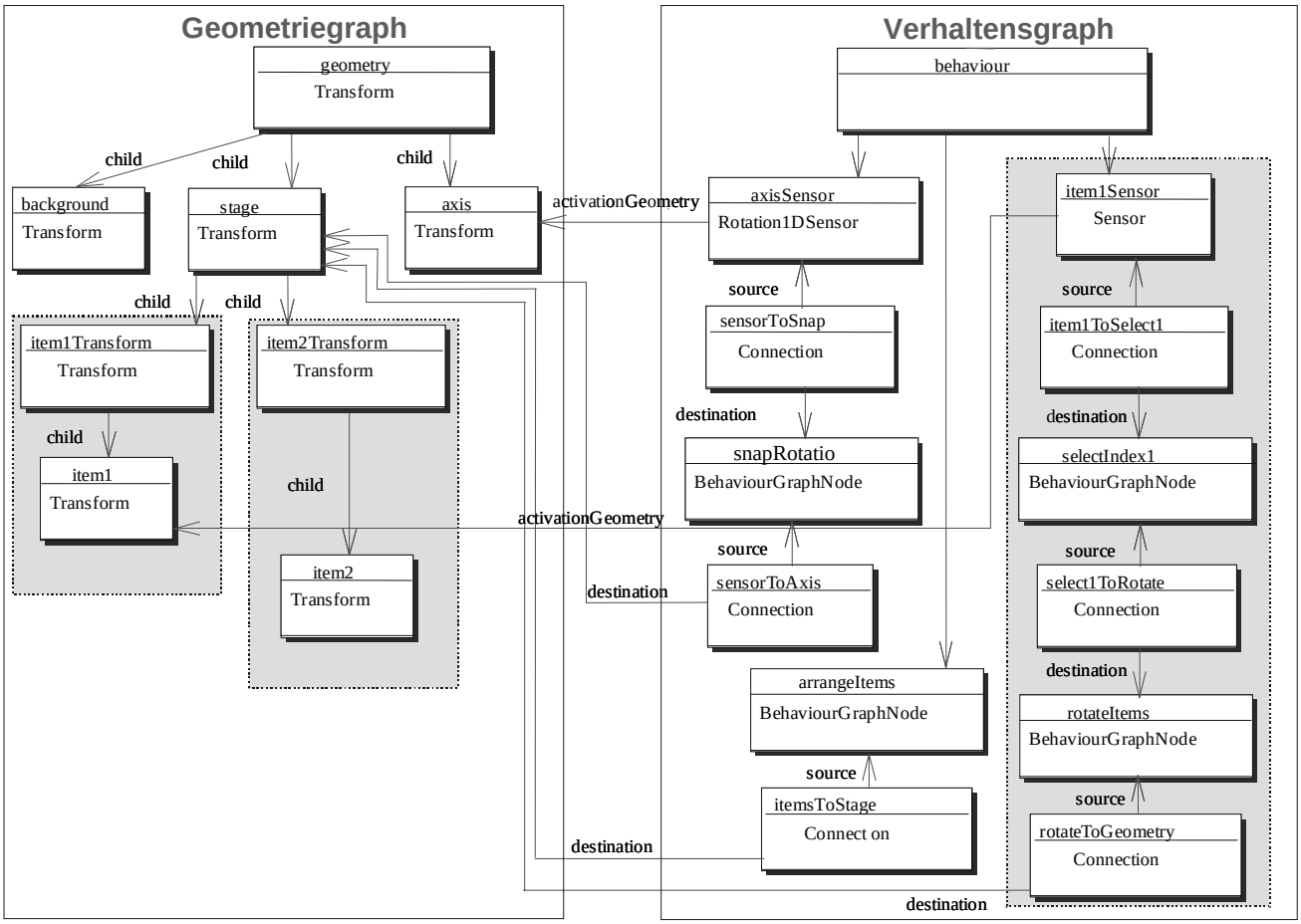

Abbildung 4: Modellierung des Ringemenü-Widgets

children traversiert die Laufzeitumgebung den Geometriegraphen für die Darstellung. Diese Schnittstelle ist für alle 3D-Widgets identisch.

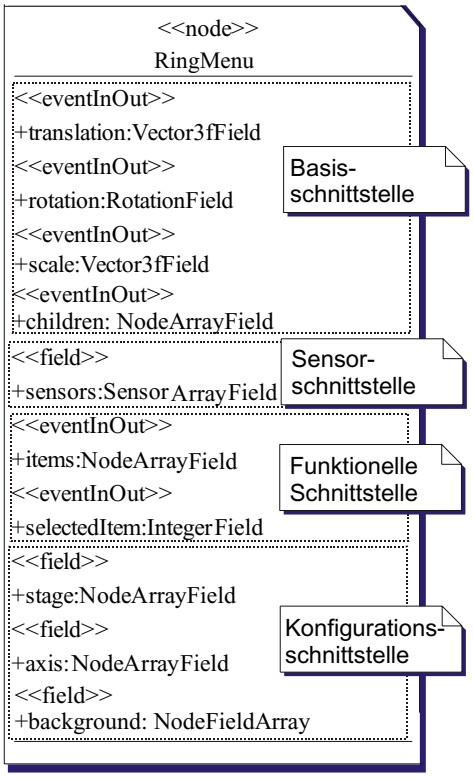

Abbildung 5: Schnittstelle des Ringmenü-Widgets

Sensorschnittstelle: 3D-Widgets veröffentlichen die Sensoren ihrer Interaktionstechniken über diese Schnittstelle, mit der die Laufzeitumgebung Eingabegeräte an die Sensoren des 3D-Widgets bindet.

Funktionelle Schnittstelle: Die funktionelle Schnittstelle realisiert die für die Interaktionsaufgabe charakteristische Funktionalität des 3D-Widgets. Sie ist spezifisch für die Interaktionsaufgabe und eröffnet die Möglichkeit, Widgets in einer Anwendung gegen andere Implementierungen austauschen zu können. Im Falle des Ringmenüs definiert sie ein Feld für das Hinzufügen und Entfernen von Menüelementen sowie ein Feld, das den Index des gewählten Menüelements wiedergibt bzw. über das ein bestimmtes Menüelement ausgewählt werden kann.

Konfigurationsschnittstelle: Die Konfigurationsschnittstelle bietet die Möglichkeit, Teile des 3D-Widgets an die konkrete Anwendung anzupassen. So werden beim Ringmenü Felder definiert, um die Geometrie der Drehachse, der Drehbühne und des Hintergrundes zu setzen. Diese Schnittstelle ist spezifisch für die konkrete Implementierung eines Widgets. Soll ein Widget ausgetauscht werden, so muß auch immer die Konfiguration ersetzt werden.

3D-Widgets werden zu Anwendungen komponiert, indem sie mittels Basisschnittstelle in der 3D-Szene positioniert werden und durch Verbindungen bzw. Verhaltensgraphen zwischen den Feldern ihrer funktionellen Schnittstellen die Funktionalität der Anwendung realisiert wird. Während Basisschnittstelle und funktionelle Schnittstelle dynamische Felder deklarieren, sind die Felder der Konfigurationsschnittstelle statisch (d.h. sie können zur Laufzeit nicht mehr verändert werden).

Abbildung auf konkrete Toolkits

Für die Implementierung der modellierten 3D-Widgets müssen Abbildungen des abstrakten Komponentenframeworks auf konkrete 3D-Grafiksysteme definiert werden. Dies entspricht einer Verfeinerung des UML Entwurfsmodells in ein Implementierungsmodell. Die nötigen Transformationen sollen als formale Abbildungsregeln für verschiedene 3D-Grafiksysteme spezifiziert werden. Objektorientierte, szenengraphbasierte 3D-Toolkits bieten dabei die größte Flexibilität, da sie einfach um zusätzliche Klassen erweiterbar sind und bereits ein Szenengraphmodell implementieren, so daß nur wenige funktionale Ergänzungen nötig sind. Außerdem können Komponententechnologien wie Java Beans oder COM einfach integriert werden. Die Abbildung auf nicht-szenengraphbasierte APIs wie OpenGL oder auf Grafikformate wie Metastream ist mit höherem Implementierungsaufwand bzw. eingeschränkter Funktionalität ebenfalls möglich.

Für die Transformation von Modellen in Implementierungen müssen die Elemente des Objektmodells und die Knotentypen des Geometrie- und Verhaltensgraphen auf entsprechende Konstrukte des 3D-Grafiksystems abgebildet werden. Die Elemente *Node* und *Field* des Objektmodells werden auf entsprechende Klassen des Szenengraphmodells abgebildet (z.B. auf *Node* bzw. *get/set* Methoden in Java3D). Für die Elemente *Connection* und *Event* definieren Komponententechnologien i.allg. äquivalente Konzepte. Die Stereotypen für die Beschränkung des Zugriffs auf Felder definieren Randbedingungen, die bei der Abbildung eingehalten werden müssen. Für Knoten- und Feldtypen des Frameworks, die kein Äquivalent im Szenengraphmodell des 3D-Grafiksystems besitzen, müssen sogenannte *Wrapper* oder *Adapter* implementiert werden, die eine entsprechende Funktionalität realisieren.

Exemplarisch wurde eine Abbildung auf VRML97 definiert, da das vorgestellte Framework an VRML orientiert ist und für die Evolution zu X3D [12] die Integration von Komponententechnologie bereits geplant ist. VRML97 wird der komponentenbasierten Softwareentwicklung durch das deklarative Dateiformat gerecht, ermöglicht jedoch nur eine eingeschränkte Abbildung, da z.B. das Konzept der Sensoren im Framework allgemeiner und umfassender definiert wurde als in VRML97. Zudem sind in VRML Darstellungsfunktionalität und Datenrepräsentation nicht konsequent voneinander getrennt, was jedoch für die Definition von Abbildungsregeln kein Problem darstellt.

Für die Beschreibung der abstrakten Modelle des Komponentenframeworks wurde ein XML-basiertes Format definiert. Auf der Grundlage dieser Grammatik können Abbildungsregeln dann als *XSL Transformations* [11] formuliert werden, mit denen sich aus den abstrakten Modellen automatisch Code-Gerüste für die Implementierung auf konkreten 3D-Grafiksystemen erzeugen lassen. Da das Framework lediglich eine Teilmenge von

UML zur Modellierung verwendet, wäre XMI – das XML Format für UML Modelle – unnötig komplex. Für das Framework wurde statt dessen eine *Document Type Definition* entworfen, die auf im Framework definierte Klassen und deren Beziehungen beschränkt ist.

Die folgenden Abbildungen geben die verschiedenen Definitionsebenen am Beispiel des *Group*-Knoten wieder. Abb. 6 zeigt diesen Knoten als UML-Modell, Abb. 7 die korrespondierende XML-Beschreibung und Abb. 8 einen Ausschnitt der *XSL Transformations* für die Verarbeitung dieses Knotens. Das Resultat dieser Transformationen ist in Abb. 9 dann als VRML97-Datei zu sehen.

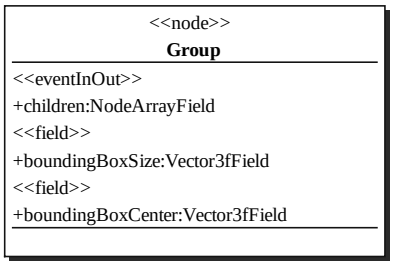

Abbildung 6: UML Modell des *Group*-Knoten

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Model SYSTEM
"file://localhost/D:/home/ /xml/framework.dtd">
<Model>
  <Group>
    <id id="_1"/>
    <children>
      <NodeArrayField>
        <id id="_2"/>
        <eventInOut/>
      </NodeArrayField>
    </children>
    <boundingBoxCenter>
      <Vector3fField>
        <id id="_3"/>
        <field/>
      </Vector3fField>
    </boundingBoxCenter>
    <boundingBoxSize>
      <Vector3fField>
        <id id="_4"/>
        <field/>
      </Vector3fField>
    </boundingBoxSize>
  </Group>
</Model>
```

Abbildung 7: XML Beschreibung des *Group*-Knoten

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform">

<xsl:template name="root" match="Model">
#VRML V2.0 utf8
<xsl:apply-templates select="*"/>
</xsl:template>

<xsl:template match="Group">
  <xsl:apply-templates select="id"/>
  <xsl:text>Group { </xsl:text>
  <xsl:apply-templates select="children |
         boundingBoxCenter | boundingBoxSize"/>
  <xsl:text>} </xsl:text>
</xsl:template>
```

```
<xsl:template match="id">
  <xsl:text>DEF </xsl:text>
  <xsl:value-of select="@id"/>
  <xsl:text> </xsl:text>
</xsl:template>

<xsl:template match="children">
  <xsl:text>children </xsl:text>
  <xsl:apply-templates
select="NodeArrayField"/>
</xsl:template>

<xsl:template match="boundingBoxCenter">
  <xsl:text>boundingBoxCenter </xsl:text>
  <xsl:apply-templates select="*"/>
</xsl:template>

<xsl:template match="boundingBoxSize">
  <xsl:text>boundingBoxSize </xsl:text>
  <xsl:apply-templates select="*"/>
</xsl:template>

<xsl:template match="NodeArrayField">
  <xsl:text>[] </xsl:text>
</xsl:template>

<xsl:template match="Vector3fField">
  <xsl:text>0 0 0 </xsl:text>
</xsl:template>

</xsl:stylesheet>
```

Abbildung 8: *XSL Transformations* für *Group*-Knoten

```
#VRML V2.0 utf8
DEF _1 Group {
   children []
   boundingBoxCenter 0 0 0
   boundingBoxSize 0 0 0
}
```

Abbildung 9: Erzeugter VRML97 Code

Trotz der strukturellen Nähe des Frameworks zu VRML folgt es anderen Entwurfskriterien als das XML-Format des VRML Nachfolgers X3D. Während X3D zur Reduzierung der Dateigröße keine Typinformationen für Felder definiert, werden sie im XML-Format des Frameworks explizit beschrieben. In der *Document Type Definition* lassen sich Einschränkungen zur Verwendung von Modellelementen formal beschreiben, für die im UML Modell Stereotypen definiert werden, die in natürlicher Sprache formuliert sind. Darunter ist die Einschränkung, daß Knoten (und damit auch Komponenten) lediglich Felder als Attribute besitzen dürfen, die durch den Stereotyp *«node»* beschrieben wird. XML an sich weist aber den Nachteil auf, daß Referenzen auf Objekte nur über den Attributtyp *ID* realisiert werden können, und sich somit keine getypten Referenzen definieren lassen.

Für die Definition von Komponenten und deren Schnittstellen ergeben sich zwei Entwurfsvarianten. Komponenten erweitern die Menge der vordefinierten Knotentypen des Frameworks und damit das Typsystem. Daher sollten Komponenten und ihre Schnittstellen sinnvollerweise in der Schemabeschreibung, d.h. in der *Document Type Definition*, definiert werden. Dadurch ist die Schemadefinition nicht abgeschlossen und keine generische Definition von Abbildungsregeln als *XSL Transformations* möglich. Somit müßte jede abstrakte Komponentenbeschreibung eine explizite DTD und Abbildungsregeln auf konkrete 3D-Grafiksysteme mitführen.

Um das zu vermeiden, wurde für das XML Beschreibungsformat des Frameworks ein weiteres Modellelement *ComponentDeclaration* eingeführt, das die Beschreibung der Schnittstelle und Implementierung einer Komponente ohne Erweiterung des Schemas ermöglicht – ein analoges Vorgehen zur Definition von Prototypen in X3D.

Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Framework vorgestellt, mit dem sich Softwarekomponenten für interaktive 3D-Grafikanwendungen abstrakt modellieren und mittels definierter Abbildungen auf konkrete 3D-Grafiksysteme übertragen lassen. Dies ermöglicht die Spezifikation von Softwarekomponenten und komponentenbasierten Anwendungen unabhängig von konkreten 3D-Grafiksystemen und künftigen Entwicklungen komponentenorientierter 3D-Grafiktoolkits. Das XML Beschreibungsformat des vorgestellten Frameworks erlaubt die Definition formaler Abbildungsregeln als *XSL Transformations*, mit denen abstrakt modellierte Komponenten automatisiert in Code-Gerüste für Implementierungen auf verschiedenen konkreten 3D-Grafiksystemen überführt bzw. Modell-Subgraphen auf 3D-Dateiformate abgebildet werden können.

Um die generelle Anwendbarkeit des Frameworks zu verdeutlichen, müssen Abbildungsregeln auf weitere 3D-Grafikssysteme, z.B. Java3D, definiert werden. So sollte das abstrakte Framework verfeinert und z.B. um formale Verhaltens- und Constraintbeschreibung erweitert werden. Für die Implementierung des Frameworks in Form eines komponentenorientierten 3D-Grafiksystems bietet sich X3D – evtl. mit der *Jamal Component Architecture* [4] als Basis – an. Die Spezifikation von häufig verwendeten 3D-Widgets mit Hilfe des Frameworks soll perspektivisch die Standardisierung von 3D-Benutzeroberflächen durch wiederverwendbare 3D-Komponenten erleichtern, wobei die Flexibilität des Frameworks Grundlage für eine Vielzahl von potentiellen 3D-Zielsystemen ist.

Literatur

[1] D.B. Conner, S.S. Snibbe, K.P. Herndon, R.C. Zeleznik, A. van Dam.: Three-Dimensional Widgets. In *Proceedings of the 1992 Symposium on Interactive Three-Dimensional Graphics*, 1992.

[2] J. Döllner, K. Hinrichs: Interactive, Animated 3D Widgets. In *IEEE Proceedings of Computer Graphics International '98*, Seiten 278-286, 1998.

[3] R. Dörner, P. Grimm: Three-dimensional Beans – Creating Web Content Using 3D Components in a 3D Authoring Environment. In *Proceedings of the Web3D-VRML 2000 Symposium*, Monterey, USA, Seiten 69-74, 2000.

[4] M. Rudolph: Jamal: Components Frameworks and Extensibility. http://www.web3d.org/TaskGroups/x3d/lucidActual/jamal/Jamal.html, 1999.

[5] M. Rudolph: X3DComponents. Web3D Consortium, http://www.web3d.org/TaskGroups/x3d/lucidActual/X3DComponents/X3DComponents.html, 1999.

[6] B. Schönhage, A. van Ballegooij, A. Eliens: 3D Gadgets for Business Process Visualization - A Case Study. In *Proceedings of the Web3D-VRML 2000 fifth symposium on the Virtual Reality Modeling Language*, Seiten 131-138, 2000.

[7] C. Geiger, C. Reiman, W. Rosenbach: Design of Reusable Components for Interactive 3D Environments. In *Proceedings of the Workshop on Guiding Users through Interactive Experiences*, Paderborn, April 13-14, 2000.

[8] C. Szyperski: Component Software - Beyond Object oriented Programming. Addison-Wesley, 1997.

[9] R. Carey, G. Bell, C. Marrin: The Virtual Reality Modeling Language - International Standard ISO/IEC 14772-1:1997. http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[10] R.J.K. Jacob, L. Deligiannidis, S. Morrison: A Software Model and Specification Language for Non-WIMP User Interfaces. In: *ACM Transactions on Computer-Human Interaction*, Vol. 6, No. 1, 1999.

[11] XSL Transformations (XSLT) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xslt, 1999.

[12] X3D: The Virtual Reality Modeling Language - International Standard ISO/IEC 14772:200x. http://www.web3D.org/TaskGroups/x3d/specification/