Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Ausarbeitung
Erfahrungen im Projekt

Stephan Koops

Visualisierung von sich bewegenden Personen

Fakultät Technik und Informatik
Studiendepartment Informatik

Stephan Koops

Thema der Ausarbeitung
Erfahrungen im Projekt

Visualisierung von sich bewegenden Personen

Stichworte

Visualisierung, bewegt, VRML, VRML-Skript-Knoten, Cortona, Java, Microsoft-Java-VM

Kurzzusammenfassung

Dieses Dokument beschreibt den Projektteil *Visualisierung von sich bewegenden Personen* im Master-Projekt Ferienclub.
Es beschreibt die Schnittstelle zur RFID-Datenbank, die Architektur der Anwendung und als letztes Erfahrungen, die der Autor währende der Entwicklung mit VRML und Java gemacht hat.
Leider ist das Projekt aus zeitlichen Gründen nicht fertig geworden, so dass es leider nicht besonders viele Erfahrungen zu berichten gibt. Es wird etwas zur virtuellen Maschine innerhalb des Cortona-VRML-Browser-Plugins berichtet und aufgetretene Probleme bei der Schnittstelle zwischen Java und VRML.

Inhaltsverzeichnis

1 Ziel und Ergebnis

1.1 Ziel

ursprüngliches Ziel Im letzten Semester habe ich mir im Vortrag im Fach Anwendungen das Ziel gesetzt, den Ferienclub und die sich darin bewegenden Personen anzuzeigen (siehe (Koops, 2005c, Seiten 49-51), Koops (2005b); siehe auch Anwendungen I (2005)). Quelle der Bewegungen sollte die RFID-Datenbank von Martin Stein sein.

abgespecktes Ziel Von der ursprünglich geplanten statischen Anzeige („Foto“), Anzeige von zur Umgebung passender Kleidung, Fotos der Gesichter, u.ä. bin ich am 8.12.2005[1] mangels Zeit abgerückt. Der Punkt Foto ist ja nur eine minimale Untermenge eines Film, und dessen Fehlen ist deshalb sicher zu verschmerzen.

1.2 Ergebnis

Navigation implementiert Leider habe ich mangels Zeit nur erreicht, dass dreidimensionale Personen in einer VRML-Welt entlang eines gegebenen Graphen laufen. Neue zu erreichende Knoten im Graphen können durch den Server zu jedem Zeitpunkt vorgegeben werden. Wenn der Server für eine Person ein neues Ziel vorgibt, bewegt sich die Person entlang der Kanten des Graphen durch den (noch nicht modellierten) Ferienclub. Die Verbindung zur RFID-Datenbank ist leider nicht fertig geworden.

3D-Visualisierung nicht geschafft Die dreidimesionale Darstellung eines Ferienclubs mit VRML habe ich nicht geschafft, so dass ich mich nur wenig mit VRML selber beschäftigen konnte.
Der eben erwähnte Graph wäre in die Visualisierung des Ferienclubs eingebunden gewesen.

Bearbeitungsreihenfolge Da mir die Logik der sich bewegenden Personen am Anfang interessanter erschien, habe ich mit diesem Teil angefangen. Außerdem kann man bei der Visualisierung des Ferienclubs viele Abstriche machen, wenn am Ende weniger Zeit ist; die Bewegung der Personen hingegen war ein in sich sehr geschlossener Teil, bei dem nicht viele Abstriche ohne wesentliche Beeinträchtigung der Funktionalität möglich sind.

[1] An diesem Termin sollten alle Studenten ihr endgültiges Ziel zum Projektende angeben.

2 Schnittstellen

Die Anwendung sollte eine Schnittstelle zur RFID-Datenbank von Martin Stein haben. Diese ist leider - ebenfalls aus zeitlichen Gründen - nicht realisiert worden.

2.1 RFID-Datenbank

Eine Komponente des Visualisierungs-Servers sollte Kontakt zum Server mit den RFID-Daten halten.

Architektur-Skizzen Am einfachsten für die Komponente 3D-Visualisierung ist es, wenn die RFID-Komponente alle ihre „Neuigkeiten“ (festgestellte Bewegungen von Personen und erstellte Prognosen) zeitgleich mit dem Eintragen in die eigene Datenbank auch dem Visualisierungsserver meldet.

Eine andere Möglichkeit ist ständiges Pollen des Visualisierungsservers.

Die erste Variante ist eleganter, da die Differenzen zum vorigen Zustand nicht berechnet werden müssen. Außerdem ist sie auch ressourcenschonender, da kein ständiges Polling durch den Visualisierungsserver nötig ist.

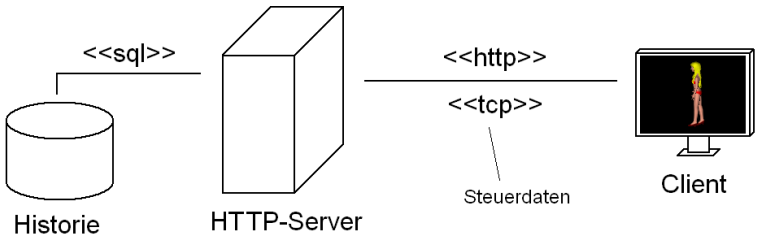
aktuelle Implementierung Im aktuellen Stand der Software wird ist diese Komponente nur als Thread implementiert, der nach eine zufälligen Zeit zufällige Knoten als neue Ziele einzelner Personen ausgibt.

3 Architektur der 3D-Visualisierung des Ferienclubs

Da die Personen im Projekt nur an gewissen Stellen (in Häusern z. B. an Türen) registriert werden, ist im Projekt eine exakte Positionsbestimmung nicht möglich, so dass mit der hier vorgestellte Projektanteil auf eine nicht-exakte Positionsdarstellung ausgerichtet ist. Wenn exaktere Daten vorliegen, kann die Darstellung entsprechend genauer werden.

3.1 Komponenten

Das System besteht aus folgenden Komponenten:



3.2 Steuerung

Der Ferienclub und jede Person besitzen eine Logik.[2] Die Ferienclub-Logik kommuniziert mit dem Server und gibt Aktionen an die Logiken der Sterungen der einzelnen Personen weiter. Diese steuern dann die Navigation und Bewegungen der jeweiligen Person.

Die gesamte Steuerung erfolgt mit Java. Die dynamischen Möglichkeiten von VRML werden nicht genutzt, damit die Logik der Steuerung nicht auf VRML-Script und Java verteilt wird. Sonst wäre wärend der Entwicklung ein ständiger Wechsel der Entwicklungsumgebung nötig gewesen, was nervig ist. Weiterhin habe ich keine kostenlose Entwicklungsumgebung für VRML gefunden, die ich über die ganze Projektdauer hätte nutzen können.

Aus Gründen des Software-Designs ist es wohl besser, wenn die Schnittstelle zwischen Java und VRML nicht mitten durch einer Komponente (in diesem Fall die Person) gehen würde. Andererseits hat das jetzige Konzept den Vorteil, dass sich der Änderungsaufwand innerhalb der Personen-VRML-Dateien reduziert, vermutlich sogar mit einem relativ einfachen Skript zu erledigen ist. Da das Projekt aber jetzt abgebrochen wird, wurden die Unterschiede zwischen dem aktuellen Stand und der Quelldatei nicht herausgesucht.
Details zu der Aufteilung finden sich im Kapitel 4.1.2: Erfahrungen, Schnittstelle zwischen Java und VRML.

[2] Inzwischen weiß ich, dass es auch das VRML Extrenal Authoring Interface (EAI) gibt, welches für genau die Fälle gedacht ist, bei denen die komplette Steuerung „von außen“ erfolgt.

3.3 Steuerdaten

Neue Aufenthaltspunkte von Personen werden vom Visualisierungsserver an alle aktiven Clients weitergeleitet, welche die Personen entsprechend durch den visualisierten Ferienclub bewegen. Wenn es sehr viele Clients gibt oder viele Personen, von denen die meisten zu einem Zeitpunkt nicht zu sehen sind, dann könnte der Server ggf. auch den einzelnen Clients gefilterte Daten zuschicken.

Dabei müssen potentiell immer, aber praktisch nur relativ selten Kommandos übertragen werden, d. h. der Server pusht die Daten.

Web-Service ? In dem Projekt werden für die Kommunikation überwieged Web-Services verwendet. Wenn der Server die Daten pushen soll, dann muss der Client einen Web-Service anbieten, dem der Server die Daten übergeben kann. Damit der Client jedoch einen WebService anbieten kann, muss er einen ServerSocket öffnen. Da sich jedoch auch Anwender zu Hause an ihrem eigenen Rechner den Ferienclub ansehen können sollen, muss die Visualisierung auch über ein Browser-Plugin erfolgen. Um ServerSockets zu öffnfn, sind im Browser entsprechende Einstellungen nötig. Dies ist auf den Geräten des Ferienclubs einfach möglich, für die potentiellen Gäste zu Hause ist dies jedoch eine nicht akzeptable Lösung.

Alternativ kann der Client ständig beim Server pollen, was jedoch unnötig viele Anfragen zur Folge hat. Diese erzeugen viel Datenverkehr, was sowohl im Funk-Netzwerk als auch bei Gästen, die den Ferienclub von zu Hause aus betrachten und keinen schnellen Internetanschluss haben, ungünstig ist.

Realisierte Datenübrtragung Wegen der eben aufgeführten Argumente gegen Web Services erfolgt die Steuerung direkt über TCP. Der Client baut Verbindung zum Server auf, über welche der Server dann einfache Befehle sendet. Da sich jetzigen Stand nur Personen durch den Ferienclub bewegen können, kann z.Zt. nur ein neues Ziel vorgegeben werden. Konzeptionell sollten hier auch Befehle wie das wechseln der Kleidung etc. übertragen werden.

3.4 Navigation der Personen

Die Positionsdaten für die Steuerung stammen - wie schon erwähnt - von RFID-Daten, die an den Türen und ähnlichen Punkten des Ferienclubs gesammelt werden. Die Bewegungen zwischen den RFID-Lesern sollten nach dem ursprünglichen Plan anhand der jeweiligen typischen Nutzung des Raumes, Platzes im Ferienclub u. ä. animiert werden. Dies ist aus zeitlichen Gründen leider nichts mehr geworden. Im Fall der Historie ist ja bekannt, wann die Person als nächstes durch welche Tür gegangen ist. Im Fall der aktuellen Anzeige und der Prognosen in die Zukunft, müssten die Daten aufgrund bisheriger Bewegungsmuster vorhergesagt werden. Diese Vorhersage ist nicht Teil der Visualierung, sondern wird vom Projektteil RFID zur Verfügung gestellt. (Siehe Kapitel 2.1: RFID-Datenbank)

Lineare Bewegungen von einer Tür zur nächsten (dort sind die ja RFID-Leser) sind leider unrealistisch (Es wäre ja auch zu einfach gewesen :-)), weil die Personen dann gnadenlos durch Wände und auch mal quer über den Pool oder so laufen würden.
Deshalb sind die Wege des Ferienclubs als Graph erfasst. Der Client sucht sich dann mittels Minimal-Cost-Suche einen wahrscheinlichen Weg, wie die Person vermutlich gegangen ist. Es war - wie erwähnt - vorgesehen, auch die Prognosen der RFID-Komponente anzuzeigen. Hierfür ist eine Planung des Weges mitsamt den dafür benötigten Wegen geplant, konnte aber wegen zeitlicher Probleme nicht fertig gestellt werden.

4 Erfahrungen

Beschäftigung mit vielem außer dem Ziel Wie viele andere auch habe ich die Erfahrung gemacht, dass man sich mit allem möglichem beschäftigt, außer mit dem, womit man sich eigentlich beschäftigen will oder wollte. In diesem Fall liegt es mit daran, dass ich die Logik der sich bewegenden Personen erst implementiert habe, und danach erst an die 3D-Darstellung des Ferienclubs angehen wollte. Letzteres ist leider der Zeit zum Opfer gefallen, so dass ich mich unterm Strich extrem wenig mit VRML u.ä. beschäftigt habe. Die einzige Ausnahme bildet die Schnittstelle zwischen Java und VRML, wo ich einige Problem hatte. Die Ursache dafür habe ich bis zum Ende zwar nicht verstanden, aber ich habe sie mit einer leichten Verschiebung der Schnittstelle umgehen können.

4.1 technisches

4.1.1 Microsoft-Java-VM

Vorraussetzungen und Annahmen Die Visualisierung sollte auch im Web von potentiellen Gästen genutzt werden können. Deshalb musste ich eine Tehnik nehmen, die möglichst viele Browser verstehen. Dann landet man leider häufig bei alten Spezifikationen. Außerdem sind die wenigsten Anwedner bereit, Policies oder ähnliches zu ändern.

Verwendete Software Ich habe den VRML-Viewer *Cortona VRML Client*[3] in der Version 4.2 benutzt. Die aktuellere Version 5.0 ist zur Zeit (2005-02-15, siehe Cortona) in der Beta-Phase und war (zumindest damals) damals nicht frei zu nutzen.

Der *BS Contact VRML / X3D*[4] unterstützte Java gar nicht, so dass er nicht in Frage kam.

Leider benutzt der *Cortona VRML Client* hartverdrahtet die von Microsoft mit Windows mitgelieferte virtuelle Maschine für Java. Deshalb musste ich auf Basis der von dieser virtuellen Maschine verwendeten Java-Spezifikation entwickeln. Die dort implementierte Version 1.1.4[5] (unter Windows XP, Service Pack 2) stammt von 1997 und ist damit reichlich veraltet.

Java 1.1.4 kann leider einiges nicht Den alten Java-Versionen fehlen leider viele Bibliotheken, die bei aktuellen (und auch nicht mehr ganz aktuellen) Versionen selbstverständlich sind. (Konkret fehlten mir Teile des Collection-Frameworks und das Interface Comparable.) Deshalb musste ich mir diese aus aktuellen Java-Versionen kopieren, was allerdings - dank des offenen Java-Quellcodes - keine Herausforderung darstellt, sondern nur Zeit kostet.

[3] http://www.parallelgraphics.com/products/cortona/

[4] http://www.bitmanagement.de/products/bs_contact_vrml.de.html

[5] Die implementierte Version der virtuellen Maschine kann in Java durch Abfragen des System-Property "java.version" herausgefunden werden. (Abfrage: `System.getProperty("java.version")`)

Einige schöne Sachen aus Java 1.5 / 5.0 (z.B. die generischen Datentypen) existierten in der Version Java 1.1.4 natürlich auch nicht.

Auch Klassen-Initializer werden von der VM in der Version nicht unterstützt. Bis ich dies herausgefunden habe, hat auch einige Zeit gedauert. Da die Java Platform Debugger Architecture (JPDA) erst seit Java 1.3 existiert, war ein einfaches Debuggen über Eclipse o.ä. nicht möglich, so dass ich mir mit in einen Logger gekapselten Ausgaben auf System.out behelfen musste. Eventuelle Ausgaben auf System.out und System.err werden von Cortona in einem Extra-Fenster ausgegeben.

4.1.2 Schnittstelle zwischen Java und VRML

Herkunft der VRML-Person Die Person ist eine von Seamless3d[6](Seamless3d) übernommene Figur, welche ich verändert habe. Dort werden die jeweils aktuellen Winkel der Beine, Knie, Arme, etc. durch Setzten von SFRotation[7] in ein bestimmtes Array gesetzt. Wie durch dieses Verändern von Array-Elementen VRML-technisch funktioniert, habe ich nicht verstanden, aber wenn man etwas benutzen will, muss man es ja nicht zwingend komplett verstehen, sondern nur das Interface benutzen können.

vom Java-Knoten durch einen VRML-Script-Knoten zu VRML Leider habe ich das System nicht dazu bringen können, von Java aus direkt die entsprechenden Werte setzen zu können, weil die Felder im VRML-Knoten implementiert waren.
Deshalb gibt es außer dem Java-Knoten - welcher die Logik enthält - einen VRML-Script-Knoten (vom Seamless3d-Original übernommen und angepasst), der die zu setzenden Winkel von Java übernimmt und dann die Werte in dem Array an der entsprechenden Position neu setzt.

Da die meisten (ca. 40 von 45) der zu setzenden Werte vom VRML-Typ SFRotation sind, wollte ich für diese zuerst eine einzige Übergabe-Route von Java zu VRML programmieren. Die Route sollte ein Array mit 2 Elementen haben: den Index im Array als ein Element eines VRML-SFRotations und das eigentliche SF-Rotation als 2. Element des Arrays. Dieser Weg ist semantisch nicht schön, hält aber die Schnittstelle extrem klein und übersichtlich. Leider hat dies nicht geklappt; die VRML-Anzeige hat nicht reagiert.

Deshalb habe ich für die vorhandenen 40 SFRotation-Array-Elemente jeweils eigenen Routen angelegt, und dort nur ein SFRotation zu übergeben. Dieser Weg hat funktioniert.

---

[6] http://www.seamless3d.com/av/index.html, Person 38

[7] Objekte vom Typ SFRotation werden - wie der Name andeutet - für Drehungen benötigt. Ein SFRotation enthält eine Drehachse im Raum (x, y, z) und den Winkel der Drehung (im Bogenmaß). Das S steht für Single, d.h. kein Array im Gegensatz zu M, was für Multiple steht und letztlich ein Array von Objekten ist.

Codegenerierung Bei dieser Konstruktion sind leider 3 mal 40 gleichartige Code-Stücke an verschiedenen Stellen nötig:

- VRML-Felder in Java[8] als Instanzvariablen deklarieren und initialisieren
- Route vom Java-VRML-Knoten in den VRMLScript-Knoten und
- Eingangs-Events in den VRML-Knoten

Um hier Tipparbeit und Fehler zu vermeiden, habe ich einen Code-Generator geschrieben, der aus einer Konfigurationsdatei eine Java-Klasse und zwei Teile der VRML-Datei erzeugt. Dieses Verfahren hat sich bewährt.

Von der genannten Java-Klasse erbt die eigentliche Personen-Klasse. So sind die Schnittstelle und die sie nutzende Implementierung einerseits getrennt, aber doch wieder in einer Klasse, was für die verwendete Schnittstelle nötig ist.

VRML-Datei per Skript zusammenbauen Die Person besteht aus einem großen IndexedFaceSet[9]. Dieses IndexedFaceSet enthält ca. 2200 Ecken, welche jeweils einen Richtungs-Vektoren besitzen, die Flächen haben Farben, etc., so dass die Personendatei ca. 360 KB groß ist. Hier habe ich einzelne Abschnitte (die eben genannten Begrenzungen des IndexedFaceSets, aber auch den automatisch generierten VRML-Code) ausgelagert, um die zentrale Datei zur Bearbeitung übersichtlich zu halten. Die Ziel-VRML-Datei habe ich dann per Skript zusammengebaut.

Sowohl die Code-Erzeugung als auch die Aufteilung der Datei haben sich bewährt.

4.2 persönliches Fazit dieses ganzen Semesters

Ursachen Wie schon an einigen Stellen erwähnt, ist das Teilprojekt aus zeitlichen Gründen gescheitert. Dies lag auch an meinem außeruniversitären Engagement, was viel Zeit in Anspruch genommen hat. Leider habe ich am Anfang den Aufwand für die einzelnen Projekte (Vorlesungen, Master-Projekt, Praxis-Erfahrungen in der Wirtschaft) vorher nicht oder sehr falsch eingeschätzt, so dass ich mir zuviel vorgenommen habe.

Leeren für die Zukunft Im Studium haben wir natürlich gelernt, dass man vor einem Angebot die Kosten überschlagen sollte, bevor man ein Angebot für eine Software (oder was auch immer) macht. Da aber für alle diese Dinge kein Kostenvoranschlag in Euro und Cent gemacht werden musste, haben ich das leider nicht abgeschätzt. Dies hat sich nun gerächt, und ab jetzt werde ich vorsichtiger mit der Verplanung meiner Zeit umgehen.

[8] Package vrml.field.*

[9] Mit IndexedFaceSets lassen sich beliebige Polyeder (Körper, die durch ebene Flächen begrenzt sind) erzeugen.

Gegebenenfalls sollte man auch Sachen wieder absagen oder verschieben, wobei es schwierig ist, einzelne Sachen im Studium zu verchieben, ohne dass das Studium sich massiv verlängert.

Literatur

[Anwendungen I 2005] : *Vorträge im Fach Anwendungen I*. April bis Juni 2005. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2005/vortraege.html

[Cortona] : *Cortona VRML Client*. – URL http://www.parallelgraphics.com/products/cortona/

[Koops 2005a] KOOPS, Stephan: *3D-Visualisierung von sich bewegenden Objekten*. Dezember 2005. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master05-06/koops/abstract.pdf. – Ausarbeitung zum Seminar-Vortrag vom 2005-12-02, Online-Zugriff 2006-01-02

[Koops 2005b] KOOPS, Stephan: *Ausarbeitung zum Vortrag: 3D-Visualisierung von bewegten Objekten*. Juni 2005. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2005/koops/abstract.pdf

[Koops 2005c] KOOPS, Stephan: *Folien zum Vortrag: 3D-Visualisierung von bewegten Objekten*. Juni 2005. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2005/koops/slides.pdf

[Seamless3d] SEAMLESS3D: *Thyme's Avatars*. – URL http://www.seamless3d.com/av/index.html