Wissensverarbeitung und Inferenz für Gebäudenavigation

Diplomarbeit

nach einem Thema von
Prof. Dr. Dr. hc. Wolfgang Wahlster

an der Universität des Saarlandes,
Fachbereich für Informatik,

vorgelegt von

Tim vor der Brück

22. September 2001

Danksagung

Ich danke Professor Wahlster für die Vergabe des interessanten Themas und die ausführliche Beratung bei meiner Suche nach einer Diplomarbeit.
Ich danke Jörg Baus und Antonio Krüger für die gute Betreuung und vorbildliche Hilfsbereitschaft.
Ich danke Cyrille Breihof für viele nützliche Hinweise sowie für die kooperative Zusammenarbeit.
Ich danke Boris Brandherm für seine hilfreichen Anregungen und Vorschläge bezüglich des schriftlichen Teils meiner Diplomarbeit

Hiermit erkäre ich an Eides Statt, dass ich zum Anfertigen der vorliegenden Arbeit keine anderen als die angegebenen Hilfsmittel benutzt habe.

Saarbrücken, den 22. September 2001

Tim vor der Brück

Ich widme meine Diplomarbeit der Allgemeinheit

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Where do you have to go today?

Finally you reach the airport,
the time to departure is short.
You need to buy a gift for your wife,
for your child a pocket knife.
You need to write postcards to your friends,
where do you can get some stamps?

You are running quickly down the hall,
just came for your flight the last call.
Where to go you cannot say
and finally you lose your way ...

1.1 Motivation

Wer kennt diese Situationen nicht? Man befindet sich auf dem Flughafen, das Flugzeug startet in kurzer Zeit, man muss noch zahlreiche Besorgungen vorher erledigen und sucht verzweifelt die entsprechenden Geschäfte. Oder man befindet sich im Kaufhaus und sucht stundenlang den gewünschten Artikel. In einer Behörde wird man von einem Büro zum anderen verwiesen ohne dass sich jemand zuständig fühlen würde.

Oft existieren in Gebäuden zwar verschiedene Navigationshilfen, wie statische Übersichtstafeln über mögliche Benutzerziele, doch besitzen diese meist deutliche Schwachpunkte. Betrachte man zum Beispiel Abbildung 1.1, die eine solche Navigationstafel für ein Kaufhaus zeigt. Rechts sind verschiedene Begriffe angegeben, anhand derer der Benutzer den gesuchten Artikel klassifizieren kann. Links sieht man die entsprechende Etage, wo sich die bezeichneten Produkte jeweils befinden. Dabei kann es zum einen passieren, dass der Benutzer keine Kategorie findet, in die er seinen Artikel einordnen kann oder dass Ambiguitäten entstehen können, weil mehrere Möglichkeiten der Klassifizierung möglich sind. Semantisch gesehen kann dies auftreten, wenn ein Begriff mehrere Oberbegriffe besitzt

Abbildung 1.1: Navigationstafel

(siehe Abschnitt 2.1.2). Sei nun der Benutzer beispielsweise auf der Suche nach einem *Computerbuch*. Dieses könnte nach der Navigationstafel sowohl im 2. Stockwerk bei Büchern als auch im 3. Stockwerk bei Computerzubehör erhältlich sein. Ohne beide Stockwerke abzusuchen könnte man dieses Problem lösen, indem man entweder jedes Produkt einzeln auflistet (was allerdings zu unübersichtlich wäre) oder aber ein Computersystem mit natürlichsprachlicher Eingabe benutzt, so dass eine durch den Benutzer durchzuführende Klassifizierung nicht mehr notwendig ist.

Ein weiterer Nachteil der Navigationstafel besteht darin, dass das Produkt nur sehr ungenau lokalisiert wird. Man weiss zwar in welcher Etage es sich befindet aber nicht wo dort.

Für all die oben beschriebenen Probleme kann ein Gebäudenavigationssystem hilfreich sein. Die Navigation mit einem solchen System erfolgt dabei so, dass der Benutzer als erstes sein Ziel auf eine möglichst einfache und intuitive Weise dem System mitteilt. Dieses berechnet einen optimalen Weg dorthin, welchen es dem Benutzer anschließend beschreibt. Die Wegbeschreibung kann entweder natürlichsprachlich oder auch grafisch erfolgen. Eine solche grafische Darstellung kann zum einen aus einer Übersichtdarstellung bestehen, auf der das entsprechende Objekt markiert ist. Zum Anderen wäre auch eine Animation denkbar, bei der ein Präsentationsagent den Weg zum Ziel anschaulich darstellt. Das Wegauskunftssystem kann dabei in einen mobilen Computer, wie beispielsweise einen Palm-Pilot oder in ein stationäres System integriert sein. Ein mobiles Gerät hat den

Vorteil, dass die Wegbeschreibung inkrementell erfolgen kann, d.h. sie gibt dem Benutzer Hinweise, welche Richtung er vom augenblicklichen Standpunkt einschlagen muss.

1.2 Ziel- und Gegenstand der Arbeit

Thema dieser Arbeit ist die Verarbeitung von räumlichen und semantischen Wissen für die Gebäudenavigation sowie sich daraus ableitende Inferenzprozesse. Diese beinhalten:

- die Semantikextraktion aus der Benutzereingabe zur Identifizierung des gesuchten Zielobjektes
- Inferenz der räumlichen Lage von Objekten aus deren geometrischer Beschreibung zur sprachlichen und grafischen Generierung von Wegauskünften und Lokalisationsbeschreibungen

Um diese Aspekte in einer konkreten Anwendung umzusetzen, wurde das System GECL[1] implementiert. GECL definiert eine für die Gebäudenavigation geeignete Ontologie, es implementiert eine natürlichsprachliche Eingabe, mit Hilfe derer der Benutzer sein Navigationsziel spezifizieren kann und stellt verschieden Hilfsmittel bereit, um semantische Information geeignet zu visualisieren, wie beispielsweise das Einblenden von Textinformationen über Objekte in verschiedener Detailliertheit. Diese Funktionen werden von einem in GECL enthaltenden Objektlokalisationssystem sowie vom System RANA (siehe nächster Abschnitt) verwendet. Die zwei zum Einsatz kommenden Beispielszenarios sind dabei Wegsuche im Lehrstuhl und im Flughafen.

Die vorliegende Arbeit berührt dabei verschieden Gebiete der Informatik, wie Computergrafik (Visualisierung), Computerlinguistik (natürlichsprachliche Ein- und Ausgabe), Kognitionswissenschaft (Berechnung räumlicher Relationen) sowie Wissensrepräsentation.

1.3 Einordnung der Arbeit

Die vorliegende Arbeit ist Teil des Projektes REAL [2] (siehe [REA01]), dessen Architektur in Abbildung 1.2 dargestellt ist. REAL ist ein Bestandteil des von der deutschen Forschungsgemeinschaft geförderten Sonderforschungsbereichs 378 für „Ressourcenadaptive Kognitive Prozesse "(siehe [WT98]) und beschäftigt sich mit Objektlokalisationen und Wegauskünften unter Ressourcenbeschränkungen. Es besteht aus den Teilprojekten IRREAL [3], ARREAL [4] und einem Informationskiosksystem [5] für Wegauskünfte mit Namen RANA [6], dessen Bestandteil auch diese Diplomarbeit ist.

Dabei sind diese drei Teilprojekte aber nicht gänzlich unabhängig voneinander entwickelt sondern es existieren verschiedene Verbindungen zwischen diesen. So ermittelt

[1] GECL ist das Akronym für **G**eneral **E**nvironment **C**onstruction **L**ibrary
[2] REAL ist das Akronym für **re**ssourcenad**a**ptierende Navigationshilfen
[3] IRREAL ist das Akronym für **I**nfra**r**ed **REAL**
[4] ARREAL ist das Akronym für **A**ugmented **R**eality REAL (Outdoor-GPS-Navigation)
[5] Abbildung 1.3 zeigt zum Vergleich den Informationskiosk vom Frankfurter Flughafen
[6] RANA ist das Akronym für **R**essouren**a**daptive **N**avigational **A**id

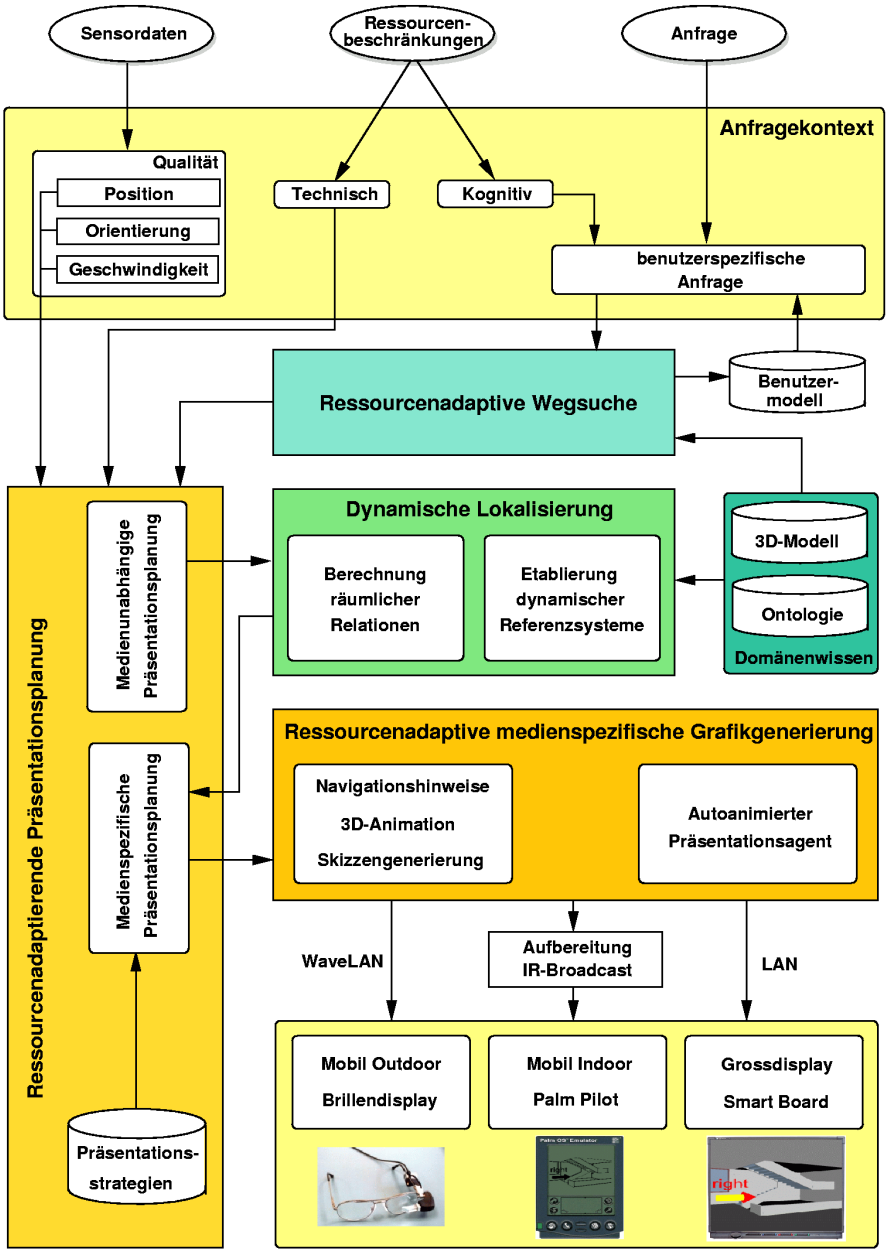

Abbildung 1.2: Architektur von REAL

Abbildung 1.3: Informationskiosk am Frankfurter Flughafen

IRREAL beispielsweise seine Übersichtsskizze aus den im Informationskiosk vorliegenden 3D-Geometrien. Im System IRREAL erhält der Anwender mittels Infrarotsignalen Wegauskunftshinweise auf seinem Palm-Pilot. Diese können je nach Bewegungsgeschwindigkeit des Anwenders aus Pfeilen oder auch aus einer Übersichtsskizze bestehen. Im Gegensatz zu IRREAL ist ARREAL für die Benutzung außerhalb von Gebäuden gedacht. Die Wegauskunft erfolgt hier durch eine Wegskizze, die auf einen Miniaturbildschirm projiziert wird, der an einer Brille befestigt werden kann. Die Position des Benutzers wird dabei mit Hilfe von GPS-Signalen ermittelt.

Der Informationskiosk besteht aus einem Computersystem, welches fest an einem bestimmten Platz installiert ist. Dieses System kann aufgrund der höheren Systemleistung und dem größeren Bildschirm dem Benutzer eine ausführlichere Darstellung geben als die Systeme der beiden anderen Teilprojekte. Dafür ist es allerdings nicht mobil einsetzbar. Der Informationskiosk zeigt sowohl eine Übersichtskarte als auch eine 3D-Animation, in der der Benutzer einen Präsentationsagenten sieht, der den Weg vom Standort des Anwenders bis zum Ziel abläuft. Dies folgt dem Prinzip, dass der Mensch sich grafische Darstellungen oft besser merken kann, als textuelle Beschreibungen. Während der Animation gibt der Präsenationsagent Hinweise zu verschiedenen Landmarken, anhand derer der Wegsuchende sich die Strecke besser merken kann, als auch zu durch den Benutzer zu spezifizierenden Zwischenzielen. Die Darstellung ist dabei ressourcenadaptiv. So erfolgt, falls der Benutzer mehr Zeit zur Verfügung hat, eine ausführlichere Präsentation. Dabei ist allerdings zu beachten, dass im Gegensatz zu der Navigation mittels Palm-Pilot, es nicht unbedingt der Fall sein muss, dass der Anwender bei besonderer Eile unbedingt eine knappere Darstellung wünscht. So bevorzugen manche Personen gerade in diesem Fall eine ausführlichere Darstellung, weil ein Verlaufen aufgrund einer zu knappen Erläuterung hier fatal sein kann. Das System besteht aus den Modulen Planung, Wegsuche (siehe [Pos01]), Animation des Präsentationsagenten (siehe [Bre01]) und der semantischen und räumlichen Wissensverarbeitung, die Bestandteil dieser Arbeit ist.

Weitere Informationen über REAL findet man in [WBKK01].

1.4 Aufbau der Arbeit

Die vorliegende Arbeit besteht insgesamt aus acht Kapiteln. Im folgenden Kapitel werden Grundlagen der Wissensrepräsentation und Computergrafik vorgestellt. Kapitel drei betrachtet bestehende Systeme und Vorgehensmodelle zur semantischen und räumlichen Verarbeitung sowie zur Domänenvisualisierung. In Kapitel vier werden eigene Konzepte und Verfahren erläutert. Die Systemarchitektur von GECL und die dort benutzte Ontologie wird in Kapitel fünf beschrieben. Die Implementierung von GECL wird anschließend in Kapitel sechs vorgestellt. Kapitel sieben erklärt die Bedienung einschließlich einiger Beispielabläufe und Kapitel acht gibt dann noch einen Ausblick auf Verbesserungsmöglichkeiten sowie künftige Entwicklungsmöglichkeiten von Wegauskunftssystemen. Der Anhang beschreibt die in dieser Arbeit verwendeten mathematischen Bezeichnungen, erklärt den Paketaufbau von GECL, beschreibt wie man das System an eigene Objektlokalisationsszenarien anpassen kann als auch wie man die in GECL definierten Klassen und Funktionen allgemein in eigenen Programmen verwenden kann.

Die Einteilung innerhalb der einzelnen Kapitel orientiert sich im allgemeinen an der Reihenfolge, mit der bestimmte Funktionen vom System ausgeführt werden, diese sind

- einladen der 3D-Welt, einschließlich semantischer und grafischer Objektrepräsentationen
- natürlichsprachliche Benutzereingabe und Lokalisation des Zielobjektes
- Berechnung räumlicher Relationen, welche für eine natürlichsprachliche Wegbeschreibung benötigt werden
- Visualisierung des zu lokalisierenden Objektes in einer 3D-Umgebung

Kapitel 2

Grundlagen

Im disem Kapitel werden Grundlagen der Wisssensrepräsentation und Computergrafik beschrieben, auf denen die im folgenden beschriebenen Konzepte aufbauen.

2.1 Wissensrepräsentation

2.1.1 Objekte und Klassen

Nach [Wah00a] bezeichnet man als Wissensrepräsentation die operationale sowie formale und damit computergerechte Darstellung von Wissensinhalten. Weit verbreitet ist dabei die objektorientierte Wissensrepräsentation (siehe [RBL91] und [Haa99]), die auch in der folgenden Arbeit verwendet wird. [1].

Zentraler Gegenstand bei dieser Modellierung ist das **Objekt**. Ein Objekt bezeichnet alles was im linguistischen Sinne ein Substantiv sein kann, das heißt Objekte können Gegenstände aber auch Gefühle oder Ereignisse sein. Ein wichtiges Merkmal eines Objektes ist seine Einzigartigkeit, d.h. zum Beispiel dass jedes Lebewesen ein eigenes Objekt ist. Ähnliche Objekte können zu Klassen zusammengefaßt werden.

Definition 1 *Ein Objekt obj ist eine* ***Instanz*** *der Klasse* $cl :\Leftrightarrow obj \in cl$

Objekte einer Klassen können sich zudem durch gewisse Eigenschaften auszeichnen, man nennt diese Eigenschaften auch **Attribute** einer Klasse. Der Wert eines Attributs kann direkt im Objekt selber definiert oder aber auf Klassenebene festgelegt werden. Im Objekt selber definiert werden Attributwerte, die normalerweise für die meisten Objekte der Klasse verschieden sind (Beispiel: Name oder Adresse einer Person). In der Klasse definierte Attributwerte sind im allgemeinen für alle Objekte in der Klasse identisch, z.B. die Eigenschaft von Fischen, dass sie schwimmen können.

Neben Attributen können auch **Methoden** (Operationen) auf Klassen definiert werden. Eine für eine Klasse definierte Methode ist auf allen Instanzen dieser Klasse anwendbar.

[1] Eine Vielzahl anderer Wissensrepräsentationsformalismen findet sich beispielsweise in [Wah00a]

2.1.2 Vererbung

Klassen lassen sich in Unterklassen unterteilen, so kann man beispielsweise die Klasse *Zimmer* in *Wohnzimmer*, *Schlafzimmer*, *Büro* u.s.w unterteilen. Eine solche Unterklasse nennt man eine **Spezialisierung** der Oberklasse, die Oberklasse eine **Generalisierung** oder Subsumption der Unterklasse.

Definition 2 *Eine Klasse clLower ist Spezialisierung einer Klasse clUpper* $:\Leftrightarrow$ $clLower \subseteq clUpper$

Der Vorgang der Spezialisierung wird auch als Vererbung bezeichnet. Eine spezialisierte Klasse übernimmt (erbt) automatisch alle Methoden, Attribute, sowie deren Werte von der Oberklasse.

Die Attribute, die in einer Klasse definiert wurden, müssen nicht unbedingt auch für jede Instanz dieser Klasse gelten (siehe [RN95]). Es gibt zwei Möglichkeit, diese zu **überschreiben**.

- einmal im Objekt selbst. Beispielsweise haben Autos normalerweise vier Räder, allerdings kann an einem konkreten Auto eines fehlen.

- In einer Unterklasse. Beispielsweise läuft die Klasse der Menschen normalerweise auf zwei Beinen, die Klasse der Babys ist eine Spezialisierung von Mensch. Elemente dieser Klasse laufen aber auf allen Vieren.

Dieser Sachverhalt lässt sich folgendermaßen formal darstellen:

$Holds, Val$ und $ClassRel$ seinen wie folgt definiert:

- $Holds(A, O, v) :\Leftrightarrow$ Für ein Objekt O kann für ein Attribut A der Wert v ermittelt werden. Dieser Wert wurde entweder explizit im Objekt gespeichert oder aus der Klassendefinition übernommen.

- $Val(A, O, v) :\Leftrightarrow$ Für ein Objekt O wurde für ein Attribut A der Wert v definiert. Dies entspricht der Zuweisung eines Attributes zu einem Wert direkt im Objekt.

- $ClassRel(A, cl, v) :\Leftrightarrow$ ein Attribut A der Klasse cl nimmt den Wert v an

Ein Attribut A eines Objektes O nimmt also den Wert v an ($Holds(A, O, v)$), wenn es entweder direkt für dieses Objekt auf diesen Wert gestzt wurde ($Val(A, O, v)$) oder wenn es in einer Klasse, von der dieses Objekt eine Instanz ist, so definiert wurde ($ClassRel(A, cl, v)$), falls dieser Wert nicht in einer Unterklasse überschrieben wurde

$$\forall A, O, v\ Holds(A, O, v) \Leftrightarrow Val(A, O, v) \vee$$
$$(\exists cl : O \in cl \wedge ClassRel(A, cl, v) \wedge \neg InterveningRel(O, cl, A))$$

Der Wert v eines Attributes A wird überschrieben mit v', wenn eine Unterklasse i von cl existiert ($Invervening(o, i, cl)$), die dem Attribut A den Wert v' zuweist.

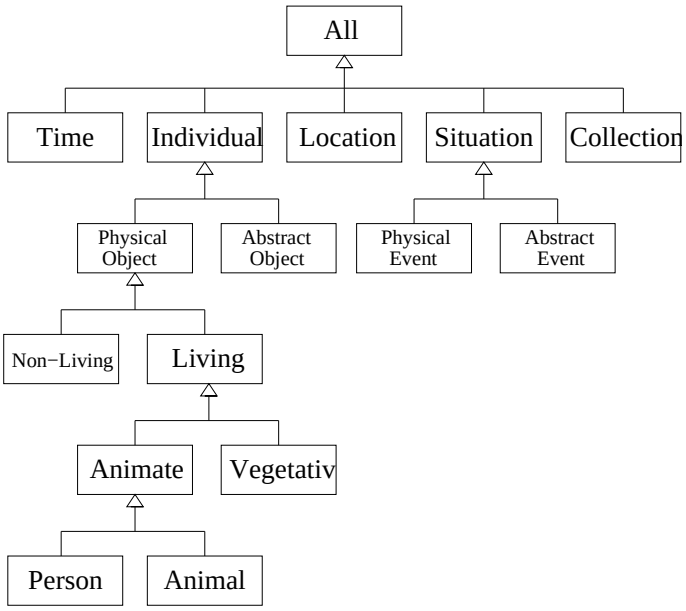

Abbildung 2.1: Beispielontologie

$$\forall O, cl, A\ InterveningRel(O, cl, A) :\Leftrightarrow$$
$$\exists i : Intervening(O, i, cl) \wedge \exists v'\ ClassRel(A, i, v')$$

$$\forall O, i, cl : Intervening(O, i, cl) :\Leftrightarrow$$
$$(O \in i) \wedge (i \subset cl)$$

Ähnlich Attributen lassen sich auch Operatoren umdefinieren. Eine umfangreiche Vererbungshierarchie, in die sich die meisten Objekte aus der realen Welt einordnen lassen bezeichnet man auch als Ontologie (siehe Abbildung 2.1 aus [All95]).

2.1.2.1 Mehrfachvererbung und -instantiierung

Manchmal ist es wünschenswert, dass eine Klasse als eine Spezialisierung mehrerer Objekte definiert ist, d.h. es gibt Klassen
$clLower$, $clUpper_1$,... $clUpper_n : clLower \subset clUpper_1, ..., clUpper_n$.
So ist ein Werkstudent beispielsweise sowohl ein Student als auch ein Arbeiter.

Eng verwandt damit ist der Fall, dass ein Objekt Instanz mehrerer Klassen sein soll, im folgenden auch als **Mehrfachinstantiierung** bezeichnet. Formal bedeutet das: $\exists O, cl_1, ..., cl_n : O \in cl_1, ...cl_n$ (siehe Seite 12 in [Mül88]). Es wäre möglich die Mehrfachinstantiierung auf die Mehrfachvererbung zurückzuführen, indem man eine neue Klasse $clLower$ einführt, die eine Spezialisierung von $cl1, ..., cn$ ist und erzeugt dann O als Instanz von $clLower$. Allerdings ist manchmal erst zur Laufzeit des Programmes bekannt, in

welchen Klassen sich das Objekt befinden soll. Außerdem existiert oft gar keine sinnvolle Bezeichnung für diese spezialisierte Klasse. Nun gibt es beispielsweise ein Geschäft, das sowohl Sonnenbrillen als auch Zeitschriften verkauft. Man müßte also nach obiger Methode die Unterklasse Sonnenbrillenzeitschriftenladen bilden, nur existiert solch ein Begriff überhaupt nicht. Daher ist eine explizite Modellierungsmöglichkeit solch einer Beziehung wünschenswert. Allerdings existiert keine objektorientierte Programmiersprache, die zulassen würde, dass ein Objekt tatsächlich direkt als Instanz mehrerer Klassen definiert werden kann. Hinweise, wie dieses Problem in dieser Arbeit gelöst wurde, finden sich in Abschnitt 4.1.4.

2.1.3 Abstraktionsebenen der Modellierung

Objekte und Klassen können gemäß ihrem Abstraktionsniveau in verschiedene Ebenen unterteilt werden. In [Sch98] werden dabei die folgenden vier Ebenen unterschieden:

1. Ausprägungsebene: Elemente der Ausprägungsebene sind alle *Objekte*, so beispielsweise *Prof. Newton* oder *Raumschiff Enterprise*

2. Typebene: Elemente der Typebene bezeichnen Objekttypen wie Person, Raumschiff, Fernseher, Büro.... Dabei sind alle Elemente der Ausprägungsebene Element von mindestens einem Element der Typebene.

3. Metaebene: Diese enthält Klassen, die unabhängig von der Anwendung benutzt werden können. Es wird sozusagen vom Anwendungsbezug abstrahiert. Beispiele für Elemente der Metaebene sind: *Leistung, Datenobjekt, Funktion, Gegenstand, Ereignis*. Elemente der Typebene sind Teilmengen von Elementen der Metaebene.

4. Meta^2-Ebene: Diese enthält nur noch ein Element, oft *Objekt* genannt. Dies ist nicht zu verwechseln mit den Elementen der Ausprägungsebene, die ebenfalls als *Objekte* bezeichnet werden. Alle Objekte der Meta-Ebene sind Teilmenge von diesem *Objekt*.

Eine Inkonsistenz, die hier besteht, ist die Tatsache, dass Elemente der Ausprägungsebene Elemente der Elemente der Typebene sind, Elemente der Typebene aber Teilmengen der Elemente der Metaebene. So ist beispielsweise ein bestimmter Tisch (enthalten in der Ausprägungsebene) ein Element der Klasse Tisch (enthalten in der Typebene). Die Menge aller Tische ist aber Teilmenge von der Menge aller Gegenstände (enthalten in der Meta-Ebene).

Daher existiert noch eine zweite Abstraktionsmethode, die dieses vermeidet. Die sieht so aus, dass die Meta-Ebene nur ein einziges Element enthält, oft als *Class* (siehe [Sun01a]) oder Metaklasse bezeichnet. Diese Metaklasse ist die Menge aller Klassen. Instanzen der Metaklasse sind daher wieder Klassen. Die Metaklasse beinhaltet als Operationen oft die Extraktion von Informationen über die von einer Klasse verwendeten Attribute und Methoden, sowie die Instantiierung von neuen Objekten einer Klasse, deren Namen erst zur Laufzeit feststeht. Diese letztere erfolgt dabei so, dass zuerst eine Instanz der Metaklasse erzeugt wird. Die erzeugte Instanz ist nun eine Klasse und kein Objekt. Sie entspricht dem Typ, zu dem das zu erzeugende Objekt gehören soll. Auf ihr sind alle Operationen aus der

Metaklasse erlaubt, d.h. auch die Instantiierung eines neuen Objektes. Hinweise, wie diese beiden Abstraktionsmodelle in dieser Arbeit verwendet wurden, finden sich in Abschnitt 5.2 und Abschnitt C.4.

2.1.4 Assoziation

Es gibt vielfältige Arten von Beziehungen, die zwischen zwei Objekten bestehen können. Allgemein nennt man das eine Assoziation. Man unterscheidet dabei 1:1, 1:n und n:m-Assoziationen (siehe [RBL91]).

In einem Wegauskunftssystem häufig verwendete Assoziationen sind räumliche Relationen. Das sind Relationen, die die Lage zwischen Objekten im Raum beschreiben. In welchem Umfang eine räumliche Relation zwischen zwei Objekten anwendbar ist, läßt sich durch den Anwendbarkeitsgrad ausdrücken, der Werte von null bis eins annimmt.
Beispiel: Im Satz „Der Computer steht *vor* dem Fernseher "ist *vor* eine räumliche Relation. Eine ausführliche Erläuterung dieses Begriffs folgt in Abschnitt 3.4.

Weiterhin kann durch eine Assoziation ausgedrückt werden, auf welche Weise ein Objekt aus anderen zusammengestzt ist, zum Beispiel besteht ein Auto aus einer Karosse, Rädern, Motor, Autositzen u.s.w.. Solch eine Assoziation nennt man auch **Aggregation**. Dabei sind auch überlappende Aggregationen zulässig d.h. ein Objekt kann auch Teil mehrerer übergeordneter Objekte sein. Ein Beispiel einer überlappenden Aggregation wäre eine Wand, die zu mehreren Räumen gehört.

2.1.5 Klassendiagramme

Um Klassenstrukturen zu beschreiben, existieren verschiedene Modellierungsverfahren. Als Standardmodellierungsverfahren durchgesetzt hat sich heute die UML (Unified Modeling Language, siehe [Obj01]). Sie wird für alle Klassendiagramme in dieser Arbeit verwendet. Ein weiteres beliebtes Modellierungsverfahren ist beispielsweise das ERM (Entity Relationship Diagram, siehe [Che77]).

UML stellt Klassen durch Rechtecke und Assoziationen durch Kanten zwischen Rechtecken dar. Ist die Assoziation eine Aggregation fügt man an das Ende der Kante an der zusammengesetzten Klasse eine auf den Kopf gestellte Raute ein (siehe Abbildung 5.6). Wird eine Klasse von einer anderen vererbt, verbindet man beide Klassen durch eine Kante und fügt an der Oberklasse ein Dreieck ein (siehe Abbildung 2.1). Zusätzlich existieren in UML Konstrukte für die Definition von Interfaces, Attributen und Memberfunktionen. Außer Klassendiagrammen gibt es noch viele andere Diagrammarten, die in UML definiert werden, wie Anwendungsfalldiagramme oder Aktivitätsdiagramme. 2.1.

2.1.6 Identifizierung von Objekten

Zur Identifizierung von Objekten werden gewöhnlich Schlüsselattribute benutzt. Diese haben oft die Form von Zahlen (z.B. Kundennummer), können aber auch Kombinationen (kartesisches Produkt) von mehreren Attributen sein. Wichtiges Merkmal von Schlüsselattributen ist ihre Einzigartigkeit, d.h. durch Angabe eines Schlüsselattributes ist ein Objekt eindeutig zu identifizieren.

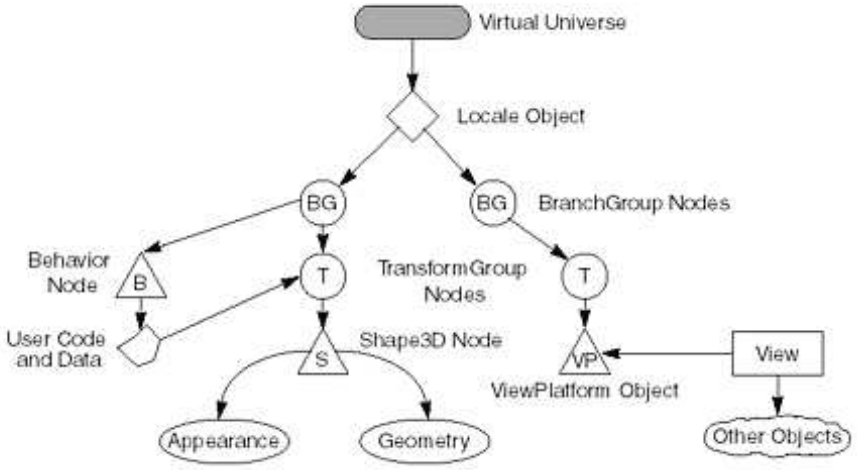

Abbildung 2.2: Beispielszenegraph

2.2 Grundlagen der Computergrafik

2.2.1 Szenegraph und Transformation

Die semantische Information muss mit den entsprechenden Geometrien verknüpft werden, so dass die Geometrien zu einem Objekt ermittelt und umgekehrt auch semantische Informationen zu Geometrien abgefragt werden können. Die Geometrien werden dabei ähnlich den semantischen Klassen aus elementareren Strukturen zu Gruppen aggregiert. Den dadurch definierten Baum nennt man auch Szenegraph (siehe [Bou99] sowie Abbildung 2.2 aus [Bou99]). Ein Knoten des Szenegraphes darf dabei nicht zwei übergeordnete Gruppen als Väter haben. Dies steht im Gegensatz zur Wissensrepräsentation, in der ein Objekt oft Bestandteil mehrerer übergeordneter Objekte sein kann, was allerdings durch einen Szenegraph nicht darstellbar ist.

Um die Geometrien auf dem Bildschirm darzustellen, muss dem System ein Szenegraph übergeben werden. Einen solchen visualisierten Szenegraph nennt man auch **lebendig**. Aus einem lebendigen Szenegraphen dürfen nur noch eingeschränkt Knoten entfernt oder hinzugefügt werden. Ein Knoten eines Szenegraphes kann u.a. sein:

- eine Gruppe
- eine Transformationsgruppe
- ein Behavior
- eine Lichtquelle
- ein 3D-Modell (Shape3D)

Durch eine Gruppe fasst man Geometrien wie oben beschrieben zu komplexeren Einheiten zusammen.
Mit einer Transformationsgruppe bezeichnet man eine Gruppe, der man eine Transformation zuordnen kann. Eine **Transformation** bezeichnet dabei eine Skalierung, Drehung und/oder Verschiebung eines geometrischen Objektes. Sie wird oft durch eine 4x4-Matrix repräsentiert. Man transformiert nun ein Objekt einfach dadurch, dass jeder Punkt von diesem mit der Transformationsmatrix multipliziert wird. Allerdings müssen die entsprechende Punktkoordinaten dazu erst in homogene Koordinaten umgewandelt werden. (siehe [FvDFH90]).

Alle Kinder einer Transformationsgruppe werden mit der entsprechenden Transformationsmatrix multipliziert. Dabei kann ein Kind einer Transformationsgruppe auch wieder eine Transformationsgruppe sein. In diesem Fall wird die am weitesten untenliegende Transformation zuerst auf die Geometrien angewandt. Das Konzept der Transformationsgruppe hat den großen Vorteil, dass Geometrien einfach durch Änderung an dieser bewegt (gedreht,skaliert) werden können. Es ist also nicht wie bei 2D-Animationen für den Systementwickler oft nötig, erst das Objekt explizit an der alten Stelle vom Bildschirm zu löschen und es dann anschließend neu zu zeichnen.

Ein Behavior kann nur in einem Blattknoten auftreten. Es definiert eine Funktion, die immer bei Eintreten eines bestimmten Ereignisses (Ablauf einer Zeitspanne, Änderung des Transformationswertes einer Transformationsgruppe, Kollision ...) aufgerufen wird.

2.2.2 Repräsentation der Geometrien

In der Computergrafik sind folgende Repräsentationen von Geometrien geläufig (siehe [Dom99])

- die Polygondarstellung
- Splinemodellierung
- Solid-Modeling
- Darstellung durch Voxel

Eine beliebte Repräsentationsform ist die Polygondarstellung. In dieser wird lediglich die zweidimensionale Oberfläche des entsprechenden Objektes modelliert. Das 3D-Modell ist also tatsächlich innen hohl, was aber nicht auffällt, wenn man es von außen betrachtet. Die Oberfläche des Objektes wird dabei durch Dreieckspolygone zusammengesetzt.
Gut geeignet ist dieses Verfahren bei ebenen Flächen. Sind diese dagegen stark gekrümt, müssen, um die Krümmung gut zu approximieren, sehr viele Dreieckspolygone verwendet werden. Daher ist es in diesem Falle oft günstiger, die Oberfläche mit Hilfe von Kurven (sogenannten Splines) zu modellieren.
Eine dritte Darstellungsform ist das sogenannte Solid-Modeling. Dabei wird das Objekt durch Vereinigung, Differenzbildung und Durchschnitt aus einigen Grundprimitiven zusammengesetzt. Diese können beispielsweise Quader, Zylinder, Kugel, Kegel oder Pyramiden beinhalten. Als Spezialfall dieses Verfahrens kann die Modellierung durch Voxel

betrachtet werden. Dabei wird das zu modellierende Objekt durch kleine Würfel gleicher Größe (sogenannte Voxel[2]) zusammengesetzt.

Allerdings unterstützt das hier verwendete 3D-Renderingsystem die drei letzteren Methoden nicht, sodass im folgenden immer von einer Polygonrepräsentation ausgegangen wird.

2.3 Zusammenfassung

Die hier erläuterten Grundladen der Wissensrepräsentation werden verwendet, um eine für ein Gebäudenavigation geeignete Ontologie zu konstruieren.
Zusätzlich wurden wichtige Prinzipien der Computergrafik erläutert, die beim Aufbau einer intelligenten virtuellen Umgebung zur Domänenvisualisierung und zur Berechnung von Objektapproximationen zum Einsatz kommen. Diese werden benötigt um räumliche Relationen zwischen Objekten zu berechnen.

[2]Voxel ist das Akronym für **Vo**lume Pi**xel**

Kapitel 3

Stand der Forschung

Im ersten Abschnitt werden mehrere Informationssysteme vorgestellt, die im Laufe der Arbeit bei verschiedenen Verfahren von Relevanz sind. Anschließend wird die semantische Objektrepräsentation an konkreten Objektlokalisations- oder Wegauskunftssystem untersucht sowie allgemeine Probleme diskutiert, die bei einer solchen Repräsentation auftreten können.
Im dritten Abschnitt werden verschiedene prozedurale Verfahren zur Suche nach Objekten angegeben. Dies beinhaltet Vorgehensweisen zum flexiblen und performanten Auffinden von Objekten in der Wissensbasis sowie die natürlichsprachliche Formulierung von Suchanfragen.
Im dritten Abschnitt folgt eine Beschreibung verschiedener Berechnungsverfahren für räumliche Relationen, die für sprachliche und grafische Wegbeschreibungen benötigt werden.
Der vierte Abschnitt befaßt sich mit der Visualisierung von Wissen. Dies beinhaltet sowohl den Aufbau einer intelligenten virtuellen Umgebung als auch die Erstellung von Objektannotationen.

3.1 Verwandte Systeme

3.1.1 CITYGUIDE

Ein leistungsfähiges Stadtwegauskunftssystem ist CITYGUIDE (siehe [Mül88] sowie Abbildung 3.1), das Nachfolgesystem von CITYTOUR (siehe [ABGT85]). CITYGUIDE wurde im Rahmen des Projektes VITRA[1] (siehe [VIT97]) entwickelt. Der Benutzer hat in CITYGUIDE die Möglichkeit Ausgangs- und Zielort in natürlicher Sprache zu spezifizieren, woraufhin das System eine natürlichsprachliche Wegbeschreibung generiert. Falls der Ausgangsort weggelassen wird, nimmt das System an, dass der Benutzer den Weg von der Stelle beschrieben haben will, an der er sich gerade befindet. CITYGUIDE berücksichtigt bei der Wegbeschreibung vielfältige Verkehrsmittel wie Bus oder Straßenbahn.

[1] VITRA ist das Akronym für **Vi**sual **Tra**nslator

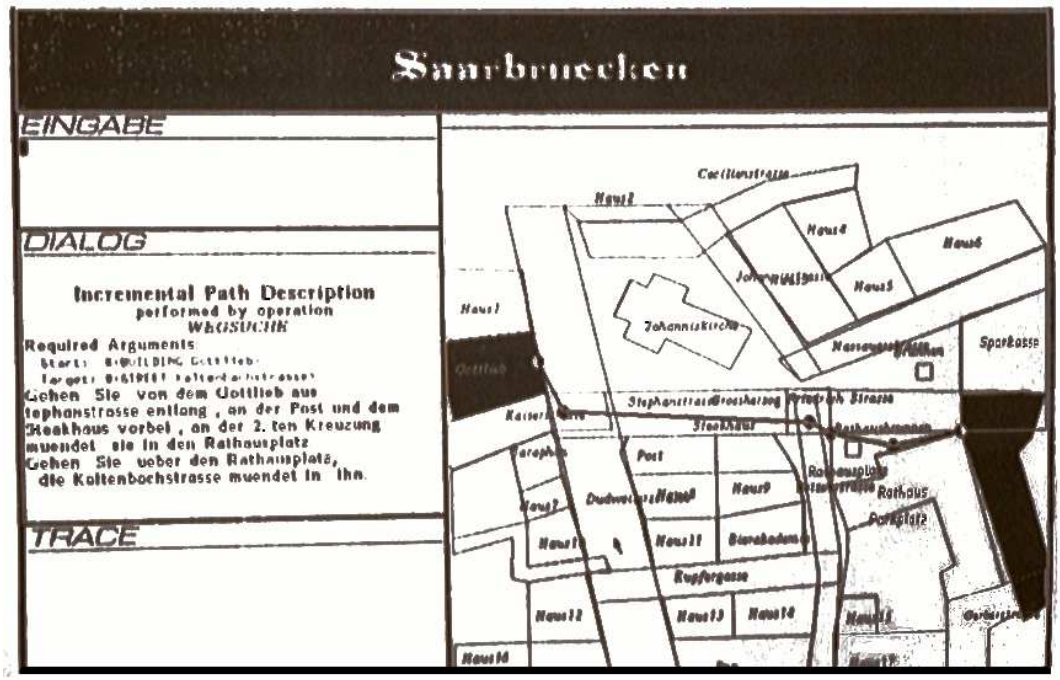

Abbildung 3.1: CITYGUIDE

Dabei kann der Benutzer auch die explizite Benutzung bestimmter Verkehrsmittel erzwingen z.B. mit folgender Anfrage:
„Wie komme ich mit dem Bus zum Media Markt? “
Eine Besonderheit besteht bei CITYGUIDE besteht darin, dass der Benutzer auch allgemeine Fragen über die räumliche Lage von Objekten stellen kann, beispielsweise:
„Befindet sich die Post vor dem Kaufhof? “
Die von CITYGUIDE zur Weg- und Lokalisationsbeschreibung verwendeten räumlichen Relationen (siehe Abschnitt 3.4) sind externe projektive Relationen (z.B. *rechts neben*, *vor*) und dynamische Pfadrelationen (z.B. *entlang*, *vorbei*), wobei es sich auf zwei Dimensionen beschränkt.

Insgesamt ist CITYGUIDE ein ausgezeichnetes Wegauskunftssystem, welches auch eine sehr gute praktische Anwendbarkeit besitzt.

3.1.2 MOSES

Das Wegauskunftssystem MOSES wurde ebenfalls als Teil des Projektes VITRA entwickelt (siehe [Maa96]). Während sich konventionelle Wegauskunftssysteme normalerweise auf eine statische grafische Darstellung beschränken, wird in MOSES die aktuelle Umgebung durch ein 3D-Modell dargestellt und der Wegverlauf durch eine Animation simuliert, in der ein kognitiver Agent die zurückzulegende Strecke abläuft und an bestimmten Zwischenpunkten inkrementelle Wegbeschreibungen generiert. Für diese Beschreibung werden

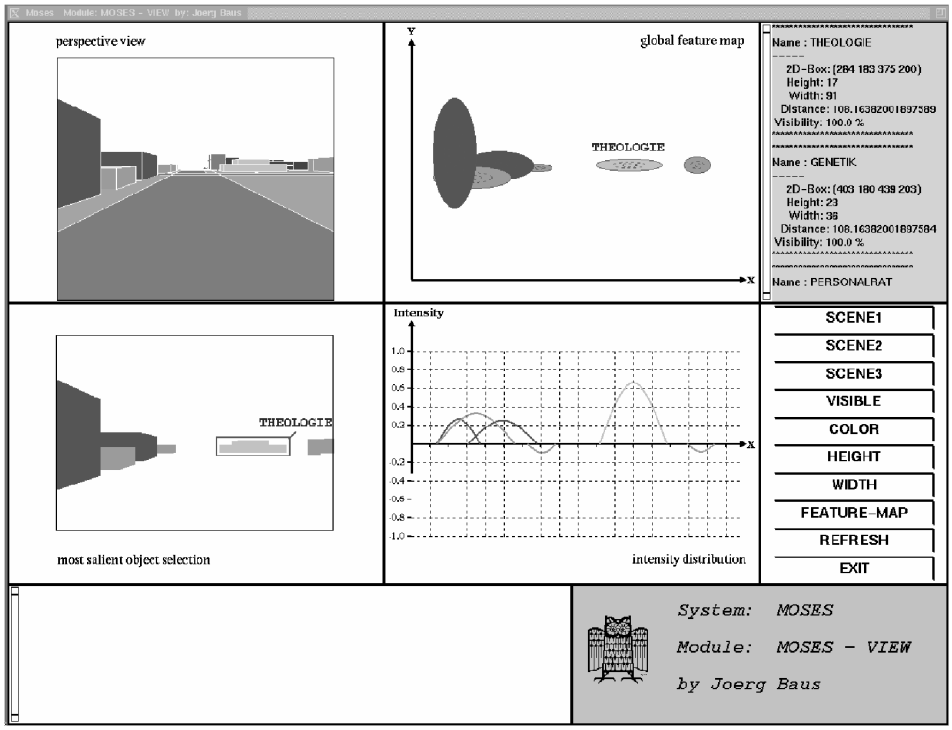

Abbildung 3.2: MOSES

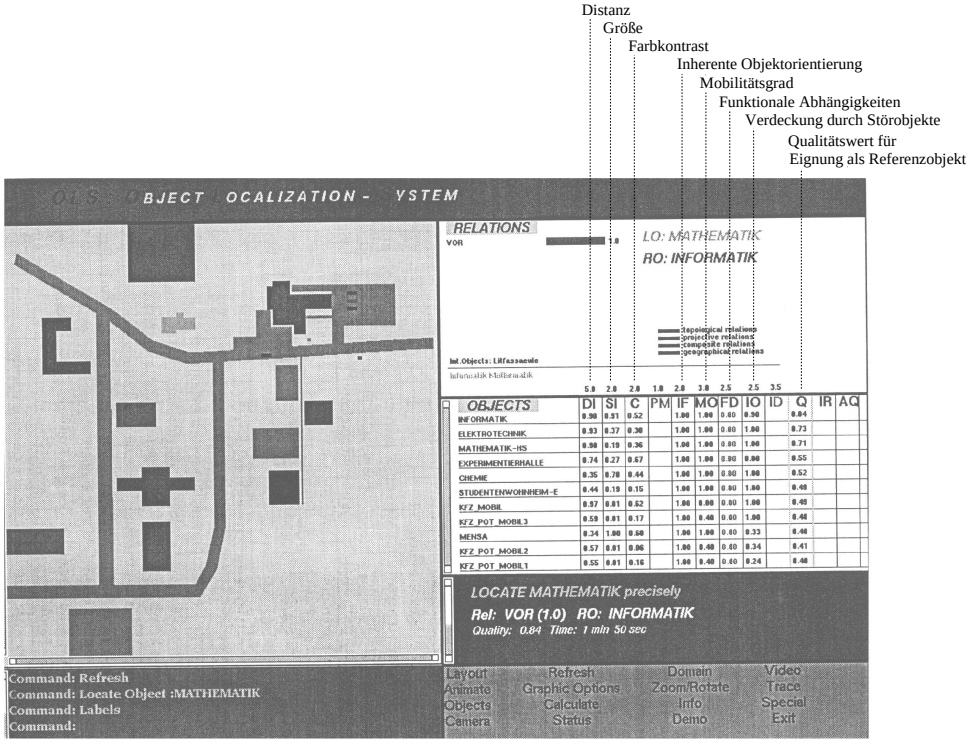

Abbildung 3.3: OLS

Referenzobjekte herangezogen, welche nicht einfach vorher festgelegt, sondern auf Grund ihrer Salienz (Auffälligkeit) (siehe [Bau96]) ausgewählt worden sind. Die Geschwindigkeit der Animation wird dabei an das benutzte Verkehrsmittel (Fußgänger, Fahrradfahrer oder Autofahrer) angepasst.

Im MOSES-View-Fenster (siehe Abbildung 3.2) sieht man vier Teilfenster. Links unten erhält man eine Übersicht auf die 3D-Welt aus der Sicht des kognitiven Agenten.
Im linken oberen Bildabschnitt werden die Objekte durch Ellipsen approximiert. Die Salienzwerte der Objekte werden dabei in verschiedene Intervalle eingeordnet, denen jeweils eine Farbe zugeordnet ist. Diese Farbe wird nun für die Darstellung der im Bildauschnitt angezeigten Ellipsen verwendet. Im Fenster rechts unten erfolgt eine funktionale Darstellung der Salienz. Die Salienz wird dabei aus Größe und Farbe sowie einiger pfadspezifischer Parameter berechnet. Das salienteste Objekt wird nun unten links durch ein Rechteck markiert.

3.1.3 OLS

Im Gegensatz zu den beiden zuvor beschriebenen Systemen ist das von Klaus-Peter Gapp entwickelte OLS [2] (siehe Abbildung 3.3 sowie [Gap97]) kein Wegauskunfts -, sondern nur

[2] OLS ist das Akronym für **O**bject**l**okalisationssystem

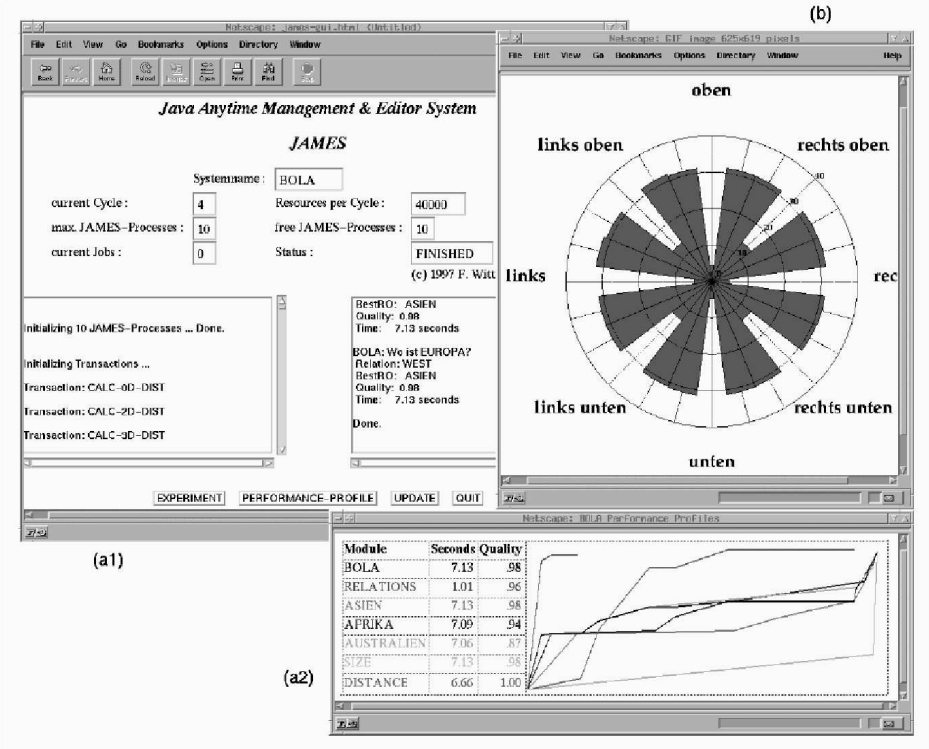

Abbildung 3.4: BOLA

ein Objektlokalisationssystem, d.h. die Lage eines Objektes im Raum kann beschrieben, allerdings kein Weg dorthin berechnet werden.

Durch das Subsystem CBR-3D [3] werden dazu zuerst geeignete Referenzobjekte automatisch nach ihrer Salienz bestimmt, für die dann das Modul CSR-3D [4] die entsprechenden externen räumlichen Relationen berechnet und ausgibt. Verwendet werden dabei projektive externe räumliche, geographische sowie Distanzrelationen. Die Anwendungsdomäne ist dabei im Gegensatz zu CITYGUIDE und MOSES ein dreidimensionaler Raum. Mehrere der in OLS verwendeten Berechnungsverfahren werden im Verlauf dieser Arbeit vorgestellt.

3.1.4 BOLA

Ein weiteres Objektlokalisationssystem ist BOLA (siehe [Blo99] und Abbildung 3.4). Dieses System lässt den Benutzer ein zu lokalisierendes Objekt spezifizieren, für das es anschließend ein geeignetes Referenzobjekt sowie die räumliche Relation, die die Lage des zu lokalisierenden Objektes bezüglich dieses Referenzobjektes am besten charakterisiert, be-

[3] CBR-3D ist das Akronym für **C**omputation of **B**est **R**eferences in 3D Space
[4] CSR-3D ist das Akronym für **C**omputation of **S**patial **R**elations in 3D Space

stimmt und ausgibt. Das Besondere an BOLA besteht in dessen Anytime-Fähigkeit. Der Benutzer erhält nähmlich zuerst eine (eventuell) ungenaue Auskunft, die nach und nach präzisiert wird. Zusätzlich teilt es ihm den Qualitätswert der angezeigten Relation mit, der sich aus dem Anwendbarkeits- und Präzisionsgrad errechnet. Die Verfahren zur Anwendbarkeitsberechnung [5] basieren dabei zum großen Teil auf denen in OLS verwendeten Algorithmen.

3.1.5 Faircar

FAIRCAR (siehe [Wah00b] und [Fai01] sowie Abbildung 3.5) ist eine Börse für Gebraucht- und Neuwagen. Es ist für diese Arbeit deswegen relevant, da es eine natürlichsprachliche Eingabe in Verbindung mit einer Datenbankanfrage implementiert. Der Benutzer spezifiziert bei FAIRCAR in einem Eingabefeld in natürlicher Sprache, was für ein Auto er sucht, beispielsweise:

„Suche ein Auto, höchstens 4 Jahre alt mit mindestens 200 PS."

Das System parst daraufhin die vom Benutzer eingegebenen Daten und generiert daraus eine Datenbankanfrage, mit der in Frage kommende Fahrzeuge aus der Wissensbasis ausgelesen und nach Übereinstimmungsgrad mit der Anfrage sortiert, ausgegeben werden. Der Benutzer kann die Erfüllung von Kriterien mit Schlüsselwörtern wie *unbedingt* oder *keinesfalls* erzwingen.

Beispiel: „Suche Auto mit keinesfalls weniger als 200 PS. "

3.1.6 Visdok

VISDOK (siehe [HHSR98]) ist ein System zur multimedialen Informationspräsentation. Dieses enthält Methoden zum Aufbau von Animationen, Berücksichtigung von Benutzerinteraktionen sowie zur Generierung von sprachlichen Beschreibungen. Das integrierte Textgenerierungssystem vereinfacht dabei die Übersetzung in andere Sprachen. Die Präsentation erfolgt durch einen Planungsprozess, bei dem ein vorgegebenes Präsentationsziel erreicht werden soll. Planungsoperatoren können vom Ersteller der Präsentation in einer formalen Sprache spezifiziert werden. Um bestimmte Objekte hervorzuheben, besteht die Möglichkeit die übrigen im Bildausschnitt befindlichen Gegenstände durch Skizzen darzustellen. Die im Ausgabefenster angezeigten Objekte können vom Benutzer selektiert werden, woraufhin das System aus der Wissensbasis die für dieses Objekt definierten Operatoren ermittelt und dem Anwender in einem Auswahlmenü präsentiert. Das System ist für diese Arbeit insofern relevant als in dieser Arbeit die Konzeption einer grafischen und textuellen Lokalisationsbeschreibung von Objekten beschrieben wird.

[5] siehe Abschnitt 3.4.3

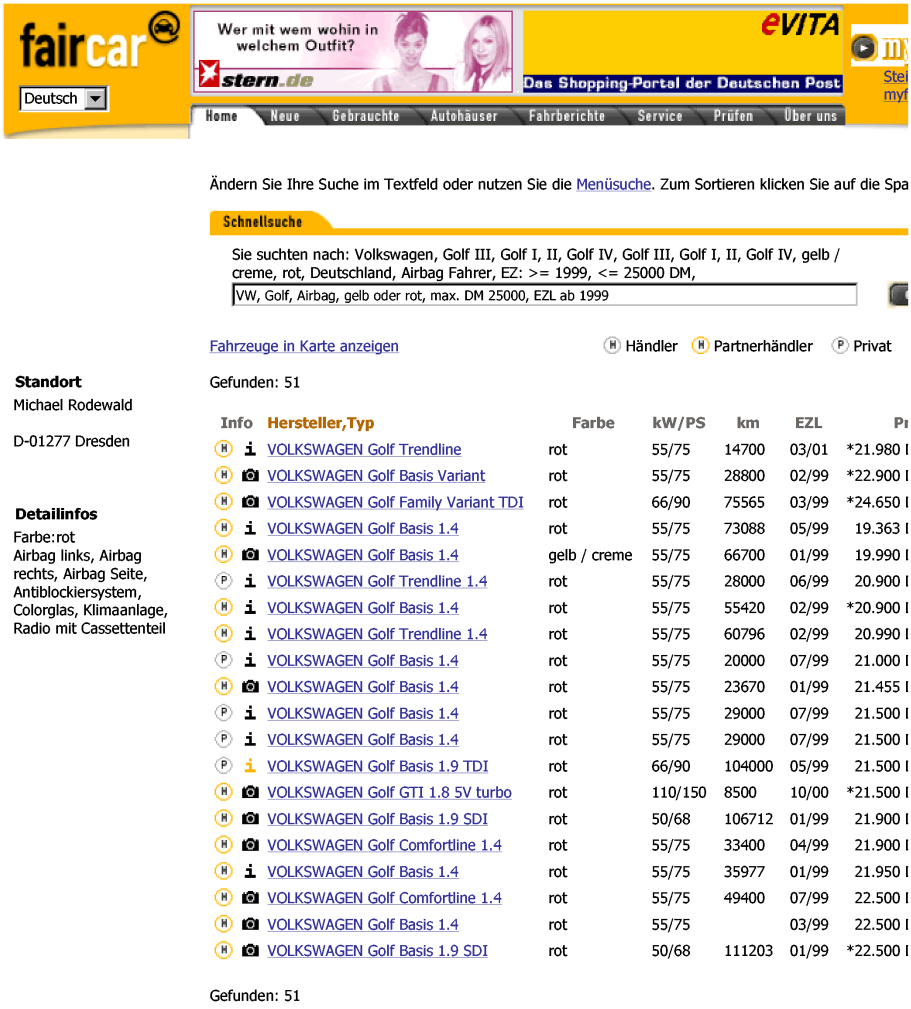

Abbildung 3.5: FAIRCAR

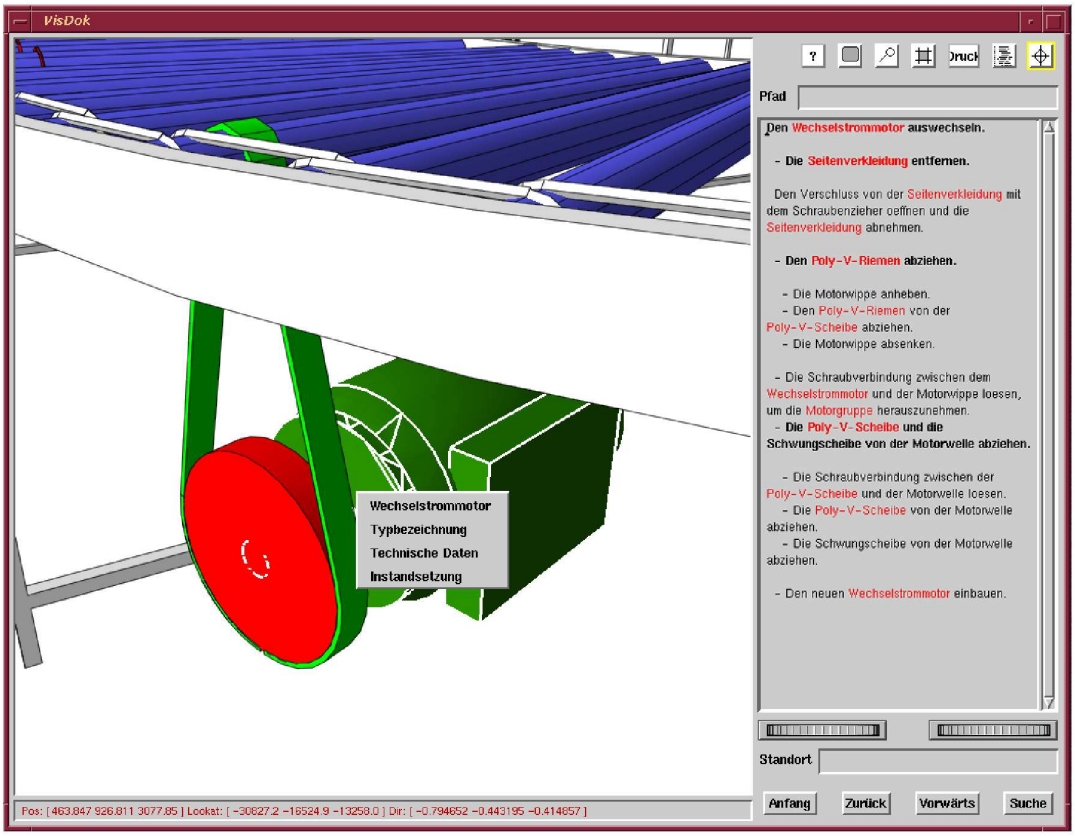

Abbildung 3.6: VISDOK

3.2 Wissensrepräsentation

Im folgenden sollen Möglichkeiten betrachtet werden, wie semantische Informationen in einem Navigationssystem repräsentiert werden können.

3.2.1 Wissensrepräsentation in Navigationssystemen

Dieser Abschnitt untersucht die semantische Repräsentation in den im letzten Abschnitt beschriebenen Navigationssystemen CITYGUIDE, MOSES, BOLA und OLS.

In System CITYGUIDE existieren fünf verschiedene Arten von Objekten. Dies sind Gebäude, Straßen, Plätze, Fußgänger und Fahrzeuge.

- Straßen werden durch die Mittellinie sowie ihren rechten und linken Rand repräsentiert.
- Gebäude werden durch ein Polygon, Mittelpunkt, Vorderkante, begrenzendes Rechteck und Auffälligkeitswert beschrieben. Die Vorderkante definiert dabei die intrinsische Orientierung des Objektes.
- Plätze sind wie die Gebäude repräsentiert mit der Ausnahme, dass sie keine intrinsische Orientierung, daher auch keine Vorderkante besitzen.
- Fußgänger und Fahrzeuge werden durch ihre Mittelpunkte repräsentiert.

Zusätzlich ist jedes Objekt für eine natürlichsprachliche Beschreibung mit seinem Genus annotiert. Weiterhin existieren zu allen Objekten Referenzen von den entsprechenden Kanten und Knoten des Wegenetzes. CITYGUIDE besitzt gegenüber anderen Navigationssystemen eine recht umfangreiche semantische Repräsentation. Allerdings sind die im System verwendeten Objekte fest vorgegeben, was die Anpassung auf andere Szenarien erschwert. Auch werden bei der natürlichsprachlichen Eingabe keine Spezialisierungsbeziehungen ausgenutzt (siehe Abschnitt 4.2.1), was die Eingabefunktion etwas unflexibel macht.

In MOSES existieren Klasse für Straßen, Landmarken, Kreuzungen, Polygone und allgemeine Gegensände. Die verwendete Vererbungshierarchie verdeutlicht Abbildung 3.7. Zu allen Objekten existieren Referenzen zu den entsprechenden Kanten oder Knoten des Wegenetzes. Zusätzlich werden Objekte mit ihren Namen sowie geometrischen Informationen (z.B. begrenzender Quader und Farbe) annotiert.

Sowohl OLS als auch BOLA sind keine Wegauskunfts- sondern nur Objektlokalisationssysteme. Sie sind in erster Linie Anwendungen zur Berechnung räumlicher Relationen und Salienzbestimmung. Die semantische Repräsentation kommt dabei zu kurz. So werden die Objekte nur minimal mit ihrem Namen beschrieben.

3.2.1.1 Zusammenfassung

CITYGUIDE besitzt von den hier vorgestellten vier Systemen die umfassendste semantische Repräsentation. Im Unterschied zu MOSES werden Fußgänger und Fahrzeuge durch

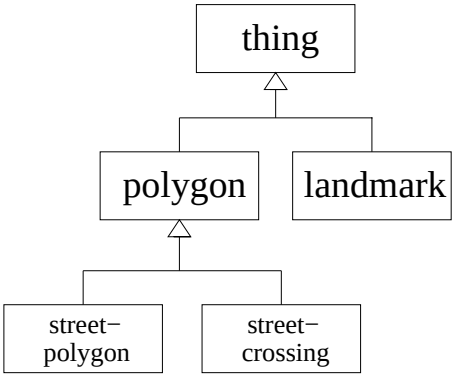

Abbildung 3.7: Vererbungshierarchie in MOSES

eigene Objekte repräsentiert. Außerdem wird bei jedem Objekt noch dessen Genus vermerkt.
OLS und BOLA dagegen legen den Schwerpunkt weniger auf praktische Anwendbarkeit zur Generierung von Wegauskünften als auf Berechnungsverfahren für räumliche Relationen und besitzen daher nur eine sehr einfache Wissensrepräsentation. Alle vier Systeme verwenden dabei eine feste Anzahl von Objekttypen. Wünschenswert wäre hier die Möglichkeit, beliebige Ontologien benutzen zu können. Außerdem macht keines der hier vorgestellten Systeme von Generalisierungs-, Aggregations- oder Innerhalb-Beziehungen Gebrauch, was diese Systeme etwas inflexibel macht (siehe Abschnitte 4.2.1, 4.1.5 sowie 4.4.2.1 wie deren explizite Modellierung in einem Navigationssystem verwendet werden kann).

Im folgenden Abschnitt soll nun ein mehr theoretischer Aspekt von semantischer Repräsentation in Wegauskunftssystemen betrachtet werden.

3.2.2 Überlappende Aggregation bei Räumen

Bei Wegauskunftssystemen innerhalb von Gebäuden werden oft Wege durch verschiedene Räume beschrieben. Ein Raum setzt sich dabei wiederum aus Wänden zusammen. Eine Wand kann nun allerdings auch wieder zu mehreren Räumen gehören, was insofern problematisch ist, dass dieser Zusammenhang nicht in einem Szenegraph ausgedrückt werden kann. Im Gegensatz zur semantischen Repräsentation, wo ein Objekt als Bestandteil mehrerer übergeordneter Objekt definiert werden kann, ist es in der Szenegraphrepräsentation in der Computergrafik nicht möglich ein Objekt als Bestandteil mehrerer Gruppen zu definieren, was die gemeinsame Repräsentation von Geometrien und semantischen Informationen in einer Datenstruktur erschwert.

Ein weiteres Problem verdeutlicht Abbildung 3.8. Kann nämlich jede Wand eindeutig einem Raum zugeordnet werden, berrechnet sich der den Raum begrenzende minimale Quader einfach aus dem minimalen Quader, der alle minimalen Quader der Wände enthält. In der obigen Abbildung dagegen sind die beiden horizontal verlaufenden Wände Bestand-

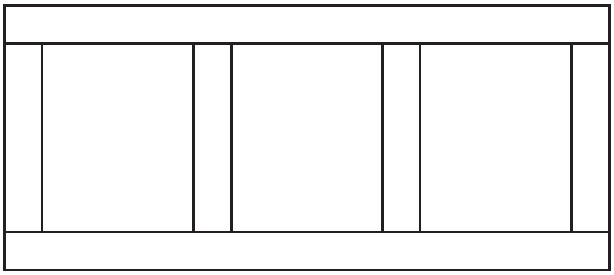

Abbildung 3.8: Drei aneinandergrenzende Räume

teil dreier Räume, weswegen der durch diesen Algorithmus berechnete Quader dreimal so groß wie der zugehörige Raum wäre.

Daher wird in [EPT96] der folgende Vorschlag gemacht:
Es werden zwei Datenstrukturen angelegt, in der ersten kommen gar keine Räume vor, sondern die Wohnung, deren Bestandteil die Räume sind, setzt sich direkt aus den Wänden zusammen. Die Türen wiederum werden als Bestandteil der Wände betrachtet. In der zweiten Datenstruktur wird jetzt die ganze Hierarchie abgebildet, wobei ein Raum statt aus Wänden aus Wandoberflächen zusammengesetzt wird. Dabei wird eine Wand so in Wandoberflächen zerlegt, dass eine Wandoberfläche auch nur genau einem Raum zugeordnet wird. Eine Wandoberfläche besitzt dabei eine Referenz zu der zugehörigen Wand und umgekehrt.

Dieser Ansatz hat nun die oben beschriebenen Nachteile nicht mehr. Allerdings ist er recht kompliziert und erfordert eine Zerlegung der Wände in Wandoberflächen, was erheblichen zusätzlichen Aufwand in der Modellierung bzw. der Berechnung benötigt.

3.3 Suche nach Objekten

In diesem Abschnitt werden verschiedene Verfahren beschrieben, wie semantisch repräsentierte Objekte in der Wissensbasis lokalisiert werden können. In einem Wegnavigationssystem ist die Suche von Objekten in der Wissensbasis beispielsweise nötig, wenn das System den vom Benutzer eingegebenen Zielpunkt ermitteln muss.

Auch ein dynamisch erzeugtes Menü könnte eine solche Funktion benötigen, um sich wärend der Laufzeit alle Objekte eines bestimmten Typs zusammenzusuchen und diese dann gemeinsam unter einem Menüpunkt zu präsentieren.

3.3.1 Grundprinzipien der Objektsuche

Man kann prinzipiell zwei Aspekte der Suche unterscheiden

- möglichst effiziente Suche
- möglichst flexible Suche

Um ein Objekt möglichst effizient in der Datenbasis zu suchen, spezifiziert man normalerweise einige Attribute oder Attributkombinationen, nach deren Werten besonders effiziente Anfragen gestellt werden können. Für diese Attribute legt man oft Hashtabellen oder B*-Bäume an (siehe [Wei99]).

Für die flexible Suche nach Objekten existieren eigene sehr mächtige Anfragesprachen. Die wohl bekannteste davon ist SQL (Structured Query Language, siehe [SQL01]) eine von IBM entwickelte Datenbankanfragesprache.
Eine einfache Operation in SQL ist die Suche nach Objektwerten:
Der Befehl

```
SELECT * from Relation WHERE Attribut=Wert
```

ermittelt beispielsweise alle Tupel aus einer Relation, deren Attribut <Attribut> den Wert <Wert> annimmt. Zusätzlich kann man Bedingungen mit Hilfe logischer Operatoren verknüpfen. Auf die Ergebnismenge der Suche können weiterhin Vereinigungs- und Schnittmengenoperationen angewendet werden. Außerdem möglich sind Aufsummieren von Werten, Durchschnittsberechnungen, Minima- und Maximabildung.

Allerdings sind in SQL einige Einschränkungen enthalten, die bei einer Koppelung mit einer natürlichsprachlichen Eingabefunktion zu Problemen führen. So sind Anfragen der Art, dass ein Attribut einen Wert annimmt, der einer bestimmten nur Zeichenkette ähnlich ist, nicht möglich. So ein Mechanismus wäre allerdings nötig, um Tippfehler des Benutzers automatisch zu korrigieren.

Auch die Suche nach Objekten, die Unterklasse eines bestimmten Typs sind, ist nicht vorgesehen, da SQL Anfragesprache für ein relationales Datenbanksystem ist, welches grundsätzlich keine Vererbung unterstützt. Solch eine Anfrage benötigt man zum Beispiel für den Fall, dass man nach einem Auto sucht, in der Wissensbasis ist das entsprechende Objekt aber unter der von der Klasse *Auto* spezialisierten Klasse *Sportwagen* abgelegt.

3.3.2 Natürlichsprachliche Suche

Für ein Wegnavigationssystem ist es von besonderer Wichtigkeit, dass der Benutzer sein Ziel möglichst einfach und schnell angeben kann. Je größer die Anzahl möglicher Zielpunkte, desto unübersichtlicher und zeitraubender wird eine Anfrage durch ein Auswahlmenü, in dem der Benutzer sich zum Zielpunkt durchnavigieren muss. Hier ist eine natürlichsprachliche Anfrage, eventuell gekoppelt mit einer Spracherkennungssoftware eine sinnvolle Eingabeerleichterung.

Laut [Wah90] sollte ein natürlichsprachliches Dialogsystem folgende Fähigkeiten besitzen:

- Es sollte Tipp- und Übertragungsfehler korrigieren können.
- Es sollte ein Synonymwörterbuch besitzen.
- Es sollte eine Flexionsanalyse besitzen, die zu einem Wort den Wortstamm ermittelt.
- Es sollte adhoc zusammengesetzte Wört erkennen, wie z.B. *Kreuzschraubenzieher*.

- bei Unklarheiten oder nicht vollständiger Information sollte es Rückfragen an den Benutzer stellen.

Die Erkennung wie Wörter zusammengesetzt sind, könnte in einem Navigationsszenario z.B. folgendermaßen verwendet werden: Angenommen der Anwender möchte in einem Kaufhaus einen Kreuzschraubenzieher kaufen. Er gibt dazu ein: „Wo sind Kreuzschraubenzieher? “. Das System besitzt aber keine Informationen über *Kreuzschrabenzieher*, erkennt aber das das Wort aus *Kreuz* und *Schraubenzieher* zusammengesetzt wurde. Da der letzte Teil eines zusammengesetzten Wortes im Deutschen dessen Typ angibt (so ist ein Stammbaum ein abstrakter Baum, ein Baumstamm aber ein Stamm) ist dem System nun bekannt, dass der Benutzer eine spezielle Art von Schraubenzieher sucht und kann ihm Hinweise geben, wo in diesem Kaufhaus *Schraubenzieher* zu finden sind, da die Wahrscheinlichkeit groß ist, dass der Anwender dort auch *Kreuzschraubenzieher* erhalten kann. In den verwendeten Beispieldomänen spielte dies allerdings keine Rolle, weswegen dieser Fall im folgenden auch nicht weiter berücksichtigt wird.

Die Semantikextraktion kann man oft als eine Funktion darstellen, die als Eingabe einen geschrieben Text erhält und als Ausgabe in einer fest vorgegebenen Datenstruktur Attributen bestimmte Werte zuweist. Die Attribute können Felder beinhalten wie Art des Ereignisses, Zeitpunkt, Ort und handelnde Person. Zeit- Orts und Personenangaben können dabei in eine vordefinierte Form gebracht werden, so dass das System die Daten anschließend leicht weiterverarbeiten kann. Ein Eigenname kann beispielsweise in Titel, Vor- und Nachname zerlegt werden, eine Uhrzeit in Jahr, Monat, Tag, Stunde und Minute (siehe [Neu01] und [Neu]). Für das Erkennen bzw. Parsen von Eigennamen wird oft eine Vornamens- und Titelliste benutzt (siehe [CV01]).

Bei einem Weginformationssystem besteht die extrahierte Informationen in erster Linie aus Beschreibungen des gesuchten Zielortes. Dies kann im einfachsten Fall einfach der Name des entsprechenden Ortes sein oder aber auch ein komplexerer Ausdruck wie in

„Wo ist der Raum, in dem sich der schnellste Computer befindet? “

Ein Wegnavigationssystem mit natürlichsprachlicher Eingabe ist CITYGUIDE (siehe Abschnitt 3.1.1). In diesem System sind verschiedene Arten von Anfragemöglichkeiten im System gespeichert wie:

„Wie komme ich [bitte] von X nach Y?“
„Können Sie mir [bitte] den Weg nach X beschreiben? “

Einige Wörter in diesen Anfragen sind optional (wie *bitte*) und können weggelassen werden. Die Benutzeranfrage wird dann auf eine dieser Möglichkeiten abgebildet (Pattern Matching).

Ein Beispiel für ein System mit Schlüsselwortsuche ist FAIRCAR (siehe Abschnitt 3.1.5). Angenommen die Benutzereingabe lautet:

„Suche Auto mit mindestens 200 PS “

Das System funktioniert nun so, dass der eingegebene Text zuerst nach Schlüsselwörtern abgesucht wird, hier *Jahre* und *PS*. Anschließend werden die dazu benachbarten Wörter betrachtet (sogenanntes Insel-Parsing), in diesem Fall *höchstens vier* und 200. Die extrahierten Informationen werden dann verwendet, um eine Datenbankanfrage zu generieren.

3.3.2.1 Tippfehlerkorrektur

Eine weitere wichtige Aufgabe eines natürlichsprachlichen Eingabesystems ist eine Tippfehlerkorrektur. Ich beschreibe hier den in [Boc86] beschriebenen Algorithmus. Es werden dabei die drei häufigsten Arten von Fehlern korrigiert, diese sind:

- Vertipper
- Vertauscher
- Auslasser

Beim Vertipper hat der Benutzer bei der Eingabe des Textes einmal eine falsche Taste gedrückt, d.h. die Hammingdistanz [6] beider Eingabezeichenketten beträgt eins.

Der Vertipper lässt sich mit den folgenden Bezeichnungen

- $|w| : A^* \rightarrow \aleph$ ermittelt die Länge der Zeichenkette w
- $w[i] : \aleph \rightarrow A$ bezeichnet den i.ten Buchstaben von der Zeichenkette w

definieren als:

$$\begin{aligned} Vertipper(text_1, text_2) &:\Leftrightarrow \\ |text_1| = &|text_2| \wedge \\ \exists p : text_1[p] &\neq text_2[p] \wedge \\ \forall q : 1 \leq q \leq |text_1| \wedge &\; q \neq p : \\ text_1[q] &= text_2[q] \end{aligned}$$

Um eine möglichst performante Bearbeitung zu gewährleisten, sollte zuerst überprüft werden, ob beide Texte die gleiche Länge besitzen.

Beim Vertauscher wurden zwei benachbarte Zeichen vertauscht.
Beispiel: Der Benutzer hat Moedgeschäft statt Modegeschäft eingegeben

$$\begin{aligned} Vertauscher(text_1, text_2) &:\Leftrightarrow \\ |text_1| &= |text_2| \wedge \\ \exists p : text_1[p] &= text_2[p+1] \wedge \\ text_1[p+1] &= text_1[p] \wedge \\ \forall 0 \leq q \leq |text_1| \wedge q &\neq p \wedge \end{aligned}$$

[6] Die Hammingdistanz gibt an, an wie vielen Stellen sich zwei gleichlange Zeichenketten unterscheiden (siehe auch [KPS96])

$$\begin{aligned} q &\neq p+1: \\ text_1[q] &= text_2[q]. \end{aligned}$$

Wiederum sollte zuerst die Einhaltung der Textlängenbedingung sichergestellt werden.

Beim Auslasser (oder Hinzufüger) läßt der Benutzer ein Zeichen des Orginaltextes aus (oder schreibt eines zuviel).

$$\begin{aligned} Auslasser(text_1, text_2) &:\Leftrightarrow \\ |text_1| &= |text_2| + 1 \wedge \\ \exists p : text_1 &= text_2[1]...text_2[p-1]text_1[p]text_2[p]...text_2[|text_2|] \end{aligned}$$

Nachdem das System nun durch Benutzung der Tippfehlerkorrektur den vom Benutzer gesuchten Gegenstand lokalisiert hat, muss dieser nun geeignet präsentiert werden. Dies ist Gegenstand der beiden nächsten Abschnitte.

3.4 Verarbeitung von räumlichem Wissen

Der Weg zum Zielort kann entweder durch eine grafische Darstellung oder natürlichsprachlich beschrieben werden. Räumliche Relationen sind in erster Linie für eine solche sprachliche Beschreibung von Nutzen, können aber auch bei einer grafischen Animation verwendet werden, um beispielsweise durch Pfeile anzudeuten, dass sich eine bestimmte Landmarke links oder rechts vom Weg befindet.

Für eine sprachliche Wegauskunft müssen zuerst **Referenzobjekte** ermittelt werden. Dies sind Objekte, die besonders auffällig sind. Die Auffälligkeit kann darin bestehen, dass sie eine besondere Form, Größe oder Farbe besitzen, die sie von der übrigen Umgebung abhebt (siehe [Bau96]).

Anhand dieser Referenzobjekte kann dann der Weg bis zum Ziel beschrieben werden. Eine Wegbeschreibung könnte so aussehen: „Gehen sie gerade aus bis Sie zur Apotheke kommen, biegen Sie dort rechts ab, links neben dem Modegeschäft Timberland befindet sich ein Telefon “[7] (siehe Abbildung 3.9) a). Das *Modegeschäft Timberland* wäre in diesem Fall das Referenzobjekt für zu lokalisierende Telefon.

Da große Objekte im allgemeinen vom Betrachter eher wahrgenommen werden als kleinere, sollte das Bezugsobjekt normalerweise größer sein als das zu lokalisierende. Ansonsten würde man das letztere vermutlich zuerst wahrnehmen und das Referenzobjekt wäre überflüssig (siehe [DH90]).

Um die obige Wegauskunft durch einen Computer zu erzeugen muß nun berechnet werden, dass sich das Telefon links vom Geschäft Timberland befindet. Dazu wiederum muss die Vorderseite dieses Geschäftes bestimmt werden.
Formal kann man also eine räumliche Relation wie folgt definieren:

[7] Zur sprachlichen Generierung von Wegauskünften siehe auch [Hor01a]

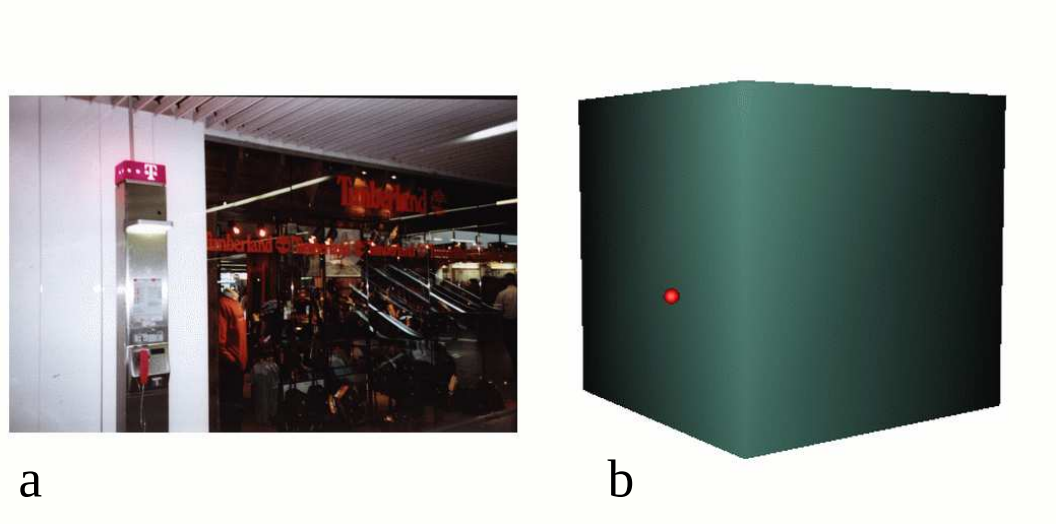

Abbildung 3.9: Das Telefon befindet sich links von Timberland

Definition 3 *Eine Assoziation zwischen einem* ***zu lokalisierendem Objekt*** *und einem* ***Referenzobjekt*** *heißt eine räumliche Relation, wenn diese die räumliche Lage des zu lokalisierenden Objektes bezüglich des Referenzobjektes beschreibt.*

3.4.1 Arten von räumlichen Relationen

Räumliche Relationen lassen sich wie in Abbildung 3.10 angegeben unterteilen (siehe [Gap97] und [Kam93]). Die verschiedenen Relationsarten enthalten dabei die folgenden Elemente:

Typ der räumliche Relation	Instanzen
Distanzrelationen	an, bei, in der Nähe von, weit entfernt
sonstige topologische Relationen	auf, zwischen, neben
geographische Relationen	Nord, Süd, West, Ost
externe projektive Relationen	über, unter, vor, hinter, rechts von, links von
interne projektive Relationen	oben (in), unten (in), vorne (in), hinten (in), rechts (in), links (in)

Dabei lassen sich Relationen kombinieren wie z.B.:
„Die Lampe hängt rechts hinten über dem Tisch. “oder
„Nord-Nord-West “.

Neben den statischen räumlichen Relationen gibt es auch dynamische Relationen wie entlang, vorbei u.s.w. (siehe Seiten 27–36 in [Blo99]). Diese werden verwendet um die Bewegung eines Objektes bezüglich eines Referenzobjektes anzugeben. Bei einer Wegauskunft beschreibt man damit meist den vom Wegsuchenden einzuschlagenden Pfad. Seltener dagegen benutzt man sie, um direkt die Lage des zu lokalisierenden Objektes zu bezeichnen,

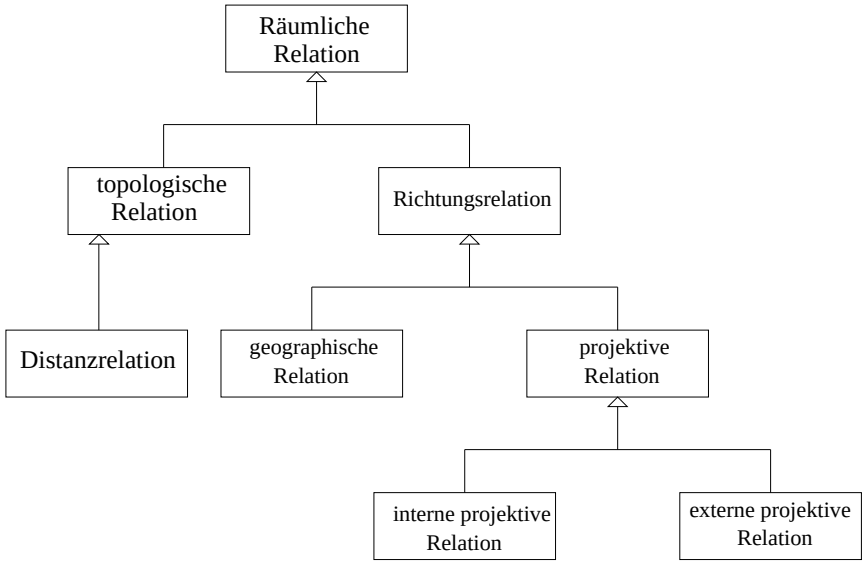

Abbildung 3.10: Räumliche Relationen

wie beispielsweise in
„Sie wollen den Zug nach London nehmen? Der fährt gerade *aus* dem Bahnhof heraus. “

Räumliche Relationen direkt aus den Polygonmodellen zu berechnen ist sehr zeitaufwendig, sodass man meist mit Approximationen arbeitet (siehe Abbildung 3.9) b), von denen im folgenden einige vorgestellt werden.

3.4.2 Objektidealisierungen

Mögliche Objektidealisierungen sind (mit steigendem Komplexitätsgrad) [8]:

- Approximation zu einem Punkt
  - Zentrum
  - Schwerpunkt
- Approximation durch Rechteck
- Approximation durch einen Quader
  - achsenparallelen Quader
  - Quader, der beliebig im Raum orientiert sein kann
- Approximation durch die konvexe Hülle (siehe [Krü01])

[8] siehe [Gap97]

Welche Approximation gewählt werden soll, ist auch abhängig von der zu berechnenden räumlichen Relation. So reicht zur Berechnung von projektiven Relationen oft eine Punktapproximation des zu lokalisierenden Objektes aus. Soll dagegen bestimmt werden, ob sich ein Objekt *auf* einem anderen befindet, benötigt man eine genauere Repräsentation.

3.4.3 Anwendbarkeitsgrade

Man kann nicht immer genau spezifizieren, ob zwei Objekte in einer bestimmten räumlichen Relation zueinander stehen oder nicht. Es sind auch Fälle möglich, wo eine solche Relation nur teilweise anwendbar ist. So ist es vorstellbar, dass sich ein Objekt nur halb auf einem anderen befindet oder dass es nicht genau vor einem anderen steht, sondern leicht versetzt.

Der Anwendbarkeitsgrad gibt an, in welchem Umfang eine räumliche Relation gilt oder nicht. Er nimmt dabei Werte von null (gar nicht anwendbar) bis eins (voll anwendbar) an. Der Anwendbarkeitsgrad ist dann von Bedeutung, wenn mehrere Objekte als Bezugsobjekt bezüglich einer bestimmten räumlichen Relation und einem zu lokalisierenden Objekt in Frage kommen. Man bevorzugt dann dasjenige Referenzobjekt mit dem höheren Anwendbarkeitsgrad für diese Relation (siehe [Blo99]).

3.4.3.1 Berechnungsarten

Grundsätzlich gibt es zwei Möglichkeiten der Berechnung räumlicher Relationen. Zum einen kann man bestimmen, welche Relation zwischen zwei Objekten am ehesten anwendbar ist, zum anderen kann man den Anwendbarkeitsgrad für eine bestimmte Relation ermitteln. Der erste Fall kann dabei auf den zweiten zurückgeführt werden, indem man zuerst für jede in Frage kommende Relation den Anwendbarkeitsgrad berechnet und dann diejenige Relation mit dem höchsten Wert auswählt. Allerdings ist dies ziemlich ineffizient, da z.B. bei projektiven Relationen dazu 26 Werte berechnet und verglichen werden müssen (über, unter, rechts, links von, vor, hinter und Kombinationen von diesen). Im System BOLA (siehe Abschnitt 3.1.4) wird deswegen zuerst eine grobe Abschätzung vorgenommen, wo das zu lokalisierende Objekt vom Referenzobjekt aus liegt. Dadurch müssen deutlich weniger Werte berechnet und verglichen werden.

Bei den folgenden Berechnungsverfahren wird meist implizit angenommen, dass der Anwendbarkeitsgrad für eine bestimmte räumliche Relation bestimmt werden soll.

3.4.4 Distanzrelationen

Distanzrelationen klassifizieren eine Distanz zwischen einem zu lokalisierenden Objekt und einem Referenzobjekt qualitativ. Sie beinhalten die Relationen *bei*, *an*, *nah* und *weit entfernt*. Kriterien für die Anwendbarkeit einer solchen Relation sind dabei euklidische Distanz und gute Erreichbarkeit. Die letztere kann durch Hindernisse wie z.B. einen Fluß (siehe [LP92]) oder auf der anderen Seite durch besonders gute Verkehrsanbindungen zwischen beiden Objekten beeinflußt werden. Weiterhin ist die Ausdehnung des Referenzobjektes vom Bedeutung. Je größer das Referenzobjekt ist, desto eher würde man das

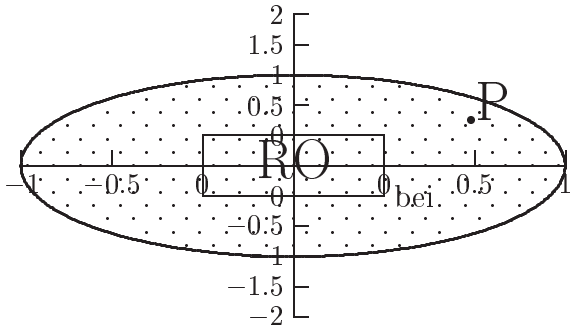

Abbildung 3.11: Berechnung von Distanzrelationen

zu lokalisierende Objekt, bei gleichbleibender absoluter Entfernung, als in dessen Nähe angeben.

Die Distanz zweier Objekte wird in [Dav90] als der Abstand der beiden nächsten Punkte dieser Objekte definiert. Da das zu lokalisierende Objekt normalerweise eine deutlich geringere Ausdehnung als das Referenzobjekt besitzt (siehe Abschnitt 3.4), berücksichtigt man oft auch nur die Ausdehnung des letzteren und verwendet für das zu lokalisierende Objekt eine Punktapproximation (siehe [Gap97]).

Alternativ könnte man die Distanz auch als Abstand der Schwerpunkte definieren. Das hat allerdings den Nachteil, das beide Objekte als weit voneinander entfernt betrachtet würden könnten, auch wenn sie sich direkt berührten.

3.4.4.1 Distanzberechnungsverfahren von Gapp

Gapp berechnet zuerst die Distanz der beiden Objektschwerpunkte in den drei Dimensionen getrennt (siehe [Gap97]). Anschließend reduziert er jeden dieser Werte um die Hälfte der Ausdehnung des Referenzobjektes in der entsprechenden Dimension und dividiert sie anschließend durch dessen ganze Ausdehnung, reduziert um einen nicht näher definierten Faktor, der mit zunehmender Objektgrösse zunimmt. Die euklidische Norm des durch diese drei Werte definierten Vektors benutzt er als Maß für den Abstand. Das Koordinatensystem wird also mit der Größe des Referenzobjektes skaliert und der Nullpunkt auf dessen Größe aufgebläht (siehe Abbildung 3.11).

Zuerst wird also der Distanzvektor $\vec{dist}$ zwischen Referenzvektor und zu lokalisierendem Objekt berechnet mit

$$\begin{aligned} \vec{dist}_i &= \max\{0, |Z(LO) - Z(RO)| - ext_i(RO)/2\} \\ &\quad \bullet sign(Z(LO) - Z(RO)) \\ &\quad \text{mit } i \in \{1, 2, 3\} \end{aligned}$$

Dazu wird zuerst die Distanz zwischen den Schwerpunkten bestimmt und diese gemäß Abschnitt 3.4.4 um die Hälfte der Ausdehnung des Referenzobjektes reduziert.

Anschließend skaliert man diesen Vektor mit der Ausdehnung des Referenzobjektes in der entsprechenden Dimension, verringert durch eine Funktion w_i:

$$\vec{dist}_{scaled} = (\vec{dist}_{scaled,x}, \vec{dist}_{scaled,y}, \vec{dist}_{scaled,z})$$

mit

$$\vec{dist}_{scaled,i} = \frac{\vec{dist}_i}{w_i(ext_i(RO))}$$

wobei $w_i(ext_i(RO)) < ext_i(RO)$ und $w_i'(j) < 0$, d.h. die Skalierung nimmt mit zunehmender Objektausdehnung ab.

Die skalierte Distanz ergibt sich dann aus: [9]

$$dist_{scaled} = ||\vec{dist}_{scaled}||.$$

Mit ihrer Hilfe wird nun die Entfernung zwischen beiden Objekten geeignet klassifiziert. Dafür werden 4 Intervalle definiert mit $i_{directlyAt} = [0, a], i_{at} = (a, b], i_{near} = (b, c), i_{farAway} = (d, \infty)$.
Das zu lokalisierende Objekt befindet sich beispielsweise *bei* dem Referenzobjekt $\Leftrightarrow S(LO) \in i_{at}$, analog für die anderen Distanzrelationen.

Gapp berücksichtigt also nur die Ausdehnung des Objektes in denjenigen Dimensionen, in denen das zu lokalisierende Objekt vom Referenzobjekt entfernt ist. Sollten sich die Schwerpunkte beider Objekte auf einer Höhe befinden, würde demnach die vertikale Ausdehnung des Referenzobjektes für die Distanzberechnung ignoriert, was doch etwas unrealistisch scheint. So sind sehr hohe Objekte normalerweise besonders auffällig und andere Objekte werden wohl eher in deren Nähe bezeichnet als bei niedrigen.

Betrachte man beispielsweise einen Gegenstand mit sehr geringer Tiefe wie eine Tür und einen anderen Gegenstand (z.B. einen Sessel), der kleiner aber dafür deutlich tiefer ist. Ein Objekt, das sich vor dem Sessel befindet, liegt bei Benutzung dieses Verfahrens viel eher in dessen Nähe als eines, das sich vor der Tür befindet, da bei Benutzung der Tür als Referenzobjekt der Raum deutlich stärker skaliert würde. In Abbildung 3.12 beträgt beispielsweise bei gleichem absoluten Abstand die skalierte Distanz des zu lokalisierenden Objekts vom Sessel 1.0, von der Tür dagegen 23.3, d.h. LO wäre von der Tür mehr als 23 mal soweit entfernt wie vom Sessel.

Weiter scheint die von Gapp vorgeschlagene Verringerung der Skalierung bei großen Objekten fragwürdig. So scheint der Satz: *Der Mond befindet sich in der Nähe der Erde* durchaus plausibel. Würde man dagegen auf eine Skalierung verzichten müsste dagegen die Aussage gelten:
Der Mond ist sehr weit von der Erde entfernt.

[9] $||x||$ bezeichnet in der folgenden Rechnung die normale euklidische Norm mit $||x|| = \sqrt{\sum_{i=1}^{n} x_i^2}$

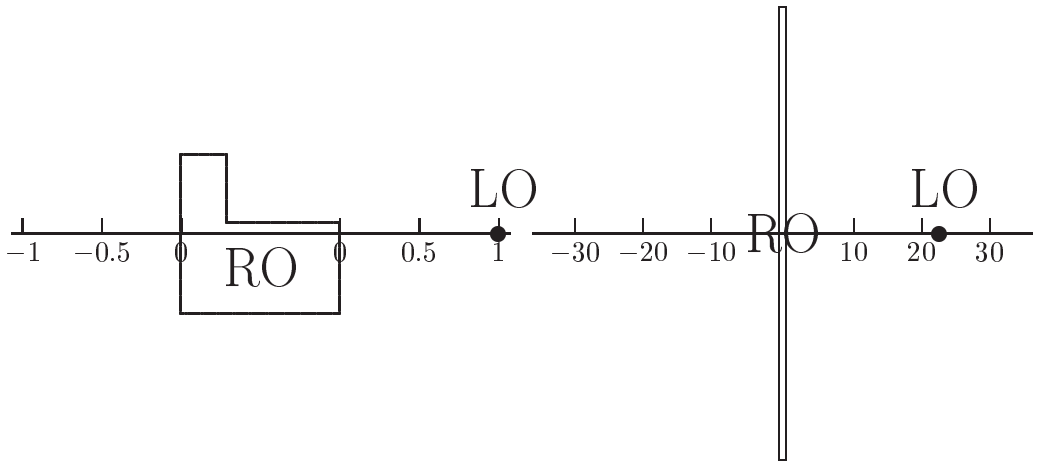

Abbildung 3.12: Probleme bei der Distanzberechnung

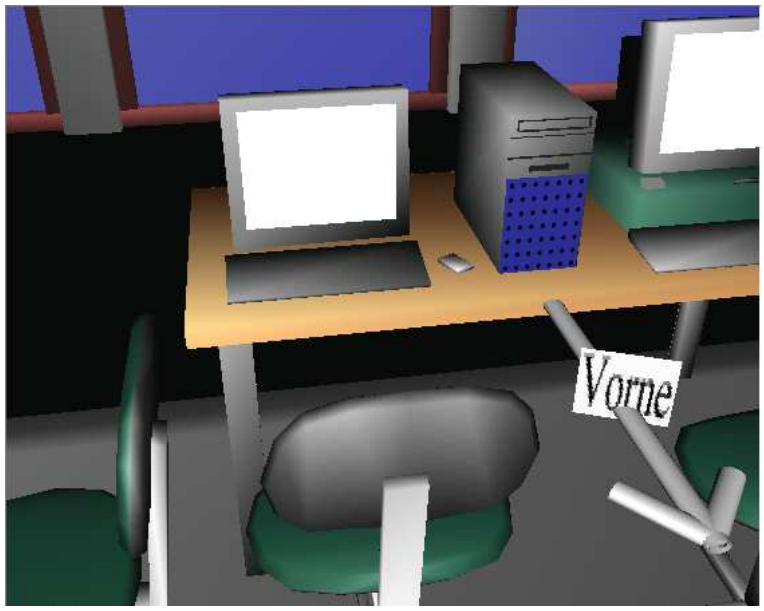

Abbildung 3.13: Beispiel für extrinsische Orientierung

3.4.4.2 Verfahren in CITYTOUR

In CITYTOUR [ABGT85] wird ausschließlich die Distanzrelation *bei* betrachtet. Der Anwendungsgrad dieser Relation wird durch eine Exponentialfunktion berechnet:

$$ag(RO, LO, bei) = \begin{cases} e^{-\frac{dist}{d}} & , \text{falls } dist > 0 \\ 0 & , \text{ sonst} \end{cases}$$

Dabei bezeichnet *dist* die skalierte Entfernung von zu lokalisierendem zum Referenzobjekt. d ist ein Dämpfungsfaktor, der bestimmt wie stark der Anwendungsgrad bei zunehmender Entfernung sinkt. CITYTOUR verwendet für diesen den Wert 2.5. Zusammenfassend kann man sagen, dass das vorgeschlagene Berechnungsverfahren durchaus plausible Werte liefert.

3.4.5 Projektive Relationen

3.4.5.1 Bestimmung der Objektvorderseite

Um eine projektive Relation wie *vor*, *hinter*, *rechts von* und *links von* u.s.w. zwischen Referenzobjekt und zu lokalisierendem Objekt zu berechnen, muß zuerst die Objektvorderseite des Referenzobjektes bestimmt werden (siehe [Gap93]). Man unterscheidet dabei zwischen den folgenden drei Fällen der Orientierung des Referenzobjektes:

- intrinsisch
- extrinsisch

- deiktisch

Wenn ein Objekt intrinsisch orientiert ist, ist die Orientierung des Objektes durch das Objekt selber gegeben. Die Vorderseite eines Raumes ist beispielsweise die Seite, wo man diesen betritt (sofern nur ein Eingang existiert). Bei einem Fahrzeug orientiert sich die Front an der normalen Fahrtrichtung.

Bei einer extrinsischen Orientierung wird die Orientierung des Objektes aus der Umgebung abgeleitet. So wird auf den in Abbildung 3.13 dargestellten Tisch durch die Orientierung der daraufbefindlichen Computer und Monitore sowie den davorstehende Stuhl eine extrinsische Orientierung induziert.

Die deiktische Orientierung ist ein Sonderfall der extrinsischen. Hierbei wird die Orientierung des Objektes aus der Position des Betrachters inferiert, die zum Betrachter zeigende Seite ist dann die Vorderseite. Bei einer Objektlokalisation kann der Betrachter entweder der Sprecher der Positionsbeschreibung oder der Höhrer sein. Wenn sich diese beide Personen frontal gegenüber befinden und der Sprecher ein Objekt in ihrer beider Mitte bezeichnet, müssen je nachdem, aus wessen Perspektive die Position beschrieben wird, rechts und links vertauscht werden. Rechts vom Sprecher entspricht dann links vom Hörer und umgekehrt, was für Konfusion sorgen kann, weil der Hörer nicht weiß, ob der Sprecher die Position aus seiner eigenen oder der Position des anderen beschreibt (siehe Seite 29 in [Maa96]).

Besitzt ein Objekt keine intrinsische oder extrinsische Front muss für die Wegbeschreibung die deiktische Orientierung ausgewählt werden. Ansonsten muss nun erst entschieden werden, welche Orientierung verwendet werden soll. Ein Kriterium ist z.B., ob Referenzobjekt und zu lokalisierendes Objekt in einer funktionalen Beziehung stehen (beispielsweise: *Die Kassette liegt vor dem Kassettenrekorder* oder *der Videorekorder steht vor dem Fernseher*). In diesem Fall wird nach [TGE01] die intrinsische Orientierung bevorzugt.

Bei der Berechnung des Anwendbarkeitsgrades spielen auch Distanzbetrachtungen eine Rolle. So verringert sich bei zunehmender Distanz die Anwendbarkeit einer Richtungsrelation (siehe [ABGT85]). Statt die Anwendbarkeit zu verringern, könnte man in diesem Fall auch alternativ die Salienz des Referenzobjektes herabsetzen, da dieses bei größerer Entfernung kleiner erscheint, also weniger auffällig wäre. Das würde dazu führen, dass seine Benutzung als Referenzobjekt weniger präferiert würde.

Weiterhin können Objekte zwischen zu lokalisierendem und Referenzobjekt (Störobjekte) den Anwendbarkeitsgrad abschwächen (siehe [Gap97]).

3.4.5.2 Naives Berechnungsverfahren

Ein einfaches Berechnungsverfahren besteht darin, die Schwerpunkte des Referenzobjektes und des zu lokalisierenden Objektes miteinander zu verbinden und den Winkel zwischen dieser Strecke und des die räumliche Relation bestimmenden Richtungsvektors zu ermitteln. Je größer der Winkel ist, desto geringer ist auch die Anwendbarkeit der entsprechenden Relation (siehe [Her86]).
Für das folgende Berechnungsverfahren werden folgende Bezeichnungen verwendet:

- *RO*: Referenzobjekt

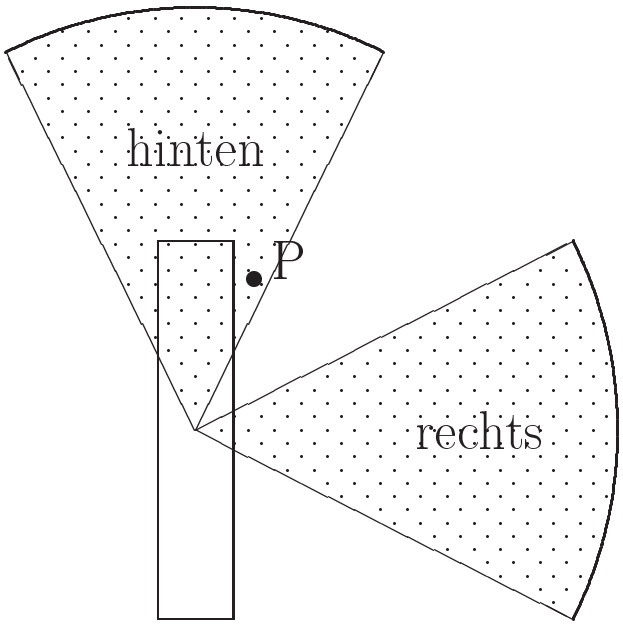

Abbildung 3.14: Naive Berechnung von projektiven Relationen

- LO: das zu lokalisierende Objekt.
- $S(obj)$: Schwerpunkt des Objektes obj.
- $\vec{Rel}$: der zu Rel gehörige Richtungsvektor, für den gilt:

$$\forall \lambda > 0 : ad(RO, o, Rel) = 1 \; mit \; o = \lambda \vec{Rel} + S(RO)$$

Der Anwendbarkeitsgrad der Relation Rel ergibt sich nun durch

$$ad(RO, LO, Rel) = f(\angle(S(LO) - S(RO), \vec{Rel}))$$

mit einer Funktion f mit $f(0) = 1$, $f(\frac{\pi}{2}) = 0$ und $\frac{df}{dx}(x) \leq 0$.

Das Problem an diesem Verfahren verdeutlicht Abbildung 3.14. Die Begrenzungslinien der Anwendbarkeitsbereiche von *rechts* und *hinter* sind angegeben. Der Mittelpunkt P des zu lokalisierenden Objektes würde nach diesem Verfahren als *hinter* RO eingestuft werden, obwohl die Relation *rechts von* RO vorzuziehen wäre.

3.4.5.3 Verfahren von Hernández

Um dieses Problem zu vermeiden schlägt Hernández in [Her86] vor, bei nahe an RO liegenden zu lokalisierenden Objekten die Punkte, von denen die Anwendbarkeitsbegrenzungslinien ausgehen zu verschieben, so dass diese gerade die Eckpunkte des umfassenden Quaders schneiden (siehe Abbildung 3.15). Liegt P dagegen weiter von RO entfernt, wird weiterhin das naive Verfahren angewendet.

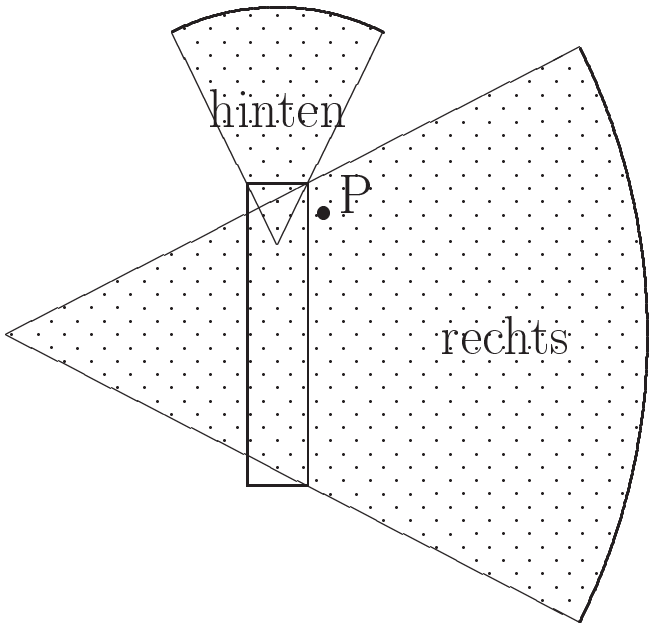

Abbildung 3.15: Verfahren von Hernández

Im Prinzip ist diese Vorgehensweise recht plausibel, allerdings stört dabei, dass der Übergang zwischen beiden Verfahren abrupt und nicht fließend ist.

3.4.5.4 Verfahren von Fuhr et al.

In [VSF+97] wird die Umgebung in mehrere Segmente unterteilt, indem die Oberflächen des umschließenden Quaders vom Referenzobjektes in den Raum hinein verlängert werden. Zusätzlich wird jeder, nicht durch eine Fläche des Quaders begrenzte Halbraum, nochmals durch eine Diagonale in zwei Halbräume aufgeteilt. Jedem Halbraum AV_i^O ist zudem ein Richtungsvektor $d(AV_i^O)$ zugeordnet, der dessen Verlauf vom Referenzobjekt aus angibt. Dieses Vorgehen ist in Abbildung 3.16 illustriert. Der besseren Übersichtlichkeit wegen, beschränkt sie sich dabei auf eine zweidimensionale Darstellung. Im Gegensatz zu den meisten anderen Verfahren wird die Ausdehnung des zu lokalisierenden Objektes hier berücksichtigt ohne dabei allerdings eine bestimmte Objektrepräsentation (z.B. Quader) vorzuschreiben.

Um nun die Anwendbarkeit einer bestimmten Relation zu berechnen, wird für jeden Halbraum der Anteil des Volumens des zu lokalisierenden Objektes, der in diesen Halbraum fällt, im Verhältnis zu seinem Gesamtvolumen berechnet und gemäß der Abweichung des dem Halbraum zugeordneten Richtungsvektors von dem die räumliche Relation definierenden Richtungsvektors gewichtet. Der Volumenenthaltungsgrad eines Halbraumes berechnet sich durch

$$containment(LO, RO, i) = \frac{vol(LO \cap AV_i^{RO})}{vol(LO)}.$$

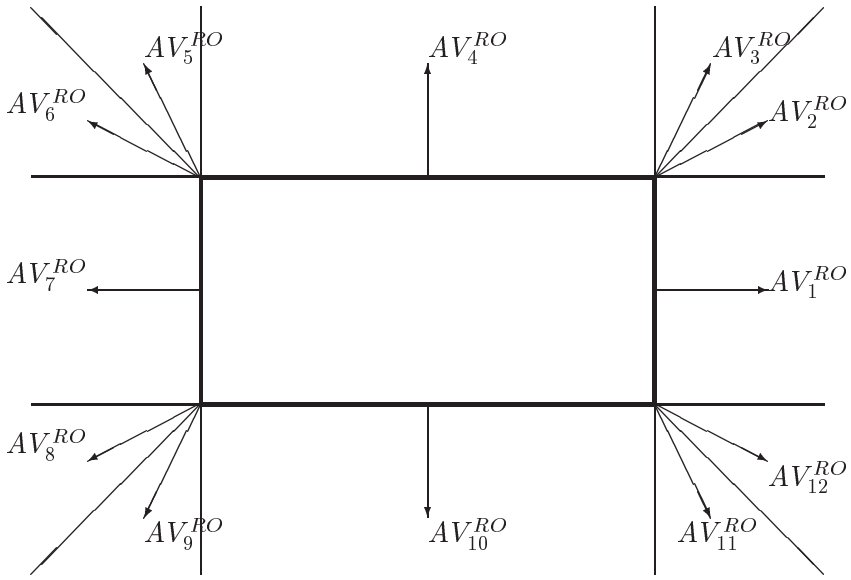

Abbildung 3.16: Akzeptanzregionen

Für die verschiedenen kanonischen Richtungen *rechts*, *links*, *oben*, *unten*, *vor*, *hinter* sind die zugehörigen Richtungsvektoren $Directions = \{\vec{right}, \vec{left}, ..., \vec{behind}\}$ definiert. So entspricht $\vec{right}$ beispielsweise dem Wert $(1, 0, 0)$, d.h. ein Objekt befindet sich rechts vom Referenzobjekt, wenn es im Koordinatensystem des Referenzobjektes eine höhere x-Koordinate und gleiche y und z-Werte besitzt.
$weigth(RO, \vec{Rel}, i)$ bezeichnet die Gewichtung des Volumenanteils, die sich errechnet aus:

$$weigth(RO, \vec{Rel}, i) = 1 - 2 \bullet \frac{arccos(d(AV_i^{RO}) \bullet \vec{Rel})}{\pi}.$$

mit $\vec{Rel} \in Directions$ Die Anwendbarkeit einer Relation Rel eines zu lokalisierenden Objektes LO und eines Referenzobjektes RO definiert sich dann durch

$$ad(RO, LO, Rel) = \sum_i weight(RO, \vec{Rel}, i) \bullet containment(LO, RO, i).$$

Falls beispielsweise der relationsdefinierende Vektor $\vec{Rel}$ zu $d(AV_i^{RO})$ senkrecht steht, ergibt sich für die Gewichtung

$$weight(RO, Rel, i) = 1 - 2 \bullet \frac{arccos(0)}{\pi} = 1 - 2 * \frac{\frac{\pi}{2}}{\pi} = 0.$$

d.h. der Anteil des Objektes, der in dem durch AV_i^{RO} definierten Sektor liegt, würde in diesem Fall für die Anwendbarkeit der Relation Rel nicht berücksichtigt werden.
Sind beide Richtungsvektoren dagegen identisch, ergibt sich das Skalarprodukt zu eins was in voller Berücksichtigung des Volumenanteils resultiert.

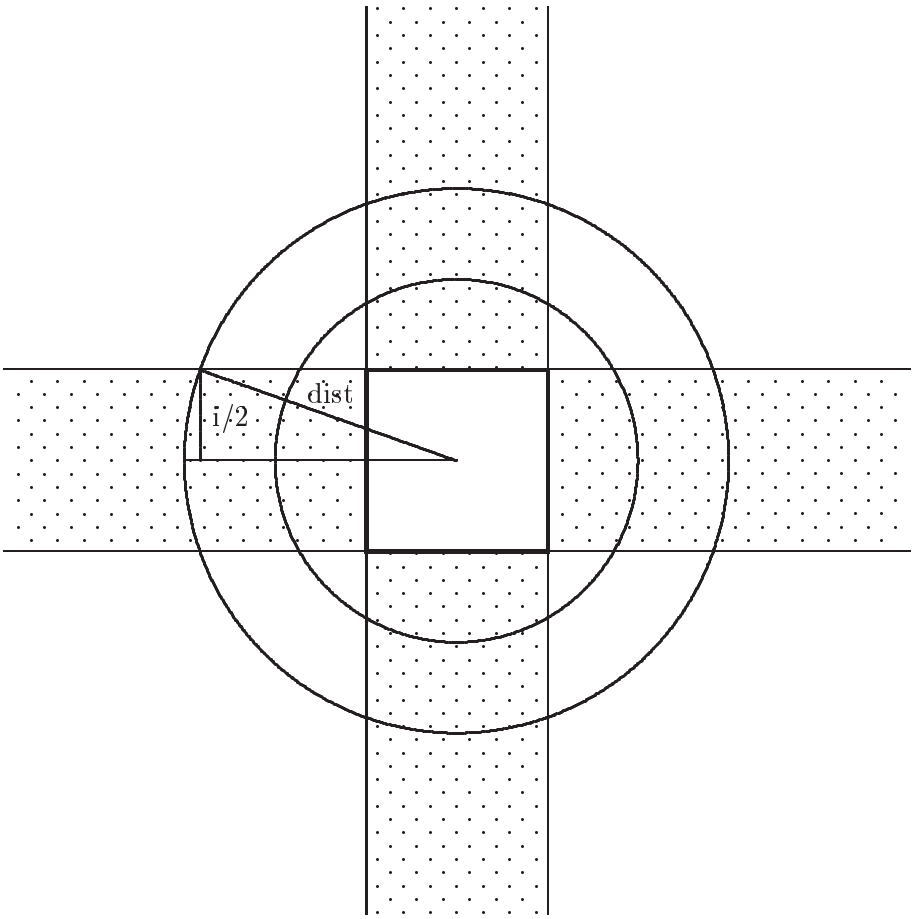

Abbildung 3.17: geringer werdende Anwendbarkeit kanonischer Relationen

Interessant an diesem Verfahren ist, dass auch die räumliche Ausdehnung des zu lokalisierenden Objektes für die Anwendbarkeitsberechnung verwendet wird. Allerdings führt dies auch zu einem erhöhten Rechenaufwand. Ferner ergibt sich, solange ein Objekt nur einen Halbraum überdeckt, immer derselbe Anwendungsgrad. Hier wäre eine weitere Differenzierung wünschenswert.

Einen weiteren Nachteil illustriert Abbildung 3.17. Mit zunehmender Entfernung (oder abnehmender Objektausdehnung) wird der Bereich, an dem die kanonischen Relationen (*rechts von*, *links von*, *über*, *unter*, *vor*, *hinter*) voll anwendbar sind, immer geringer. In der Abbildung sind die Anwendungsbereiche der nicht zusammengesetzten Relationen schraffiert. Man kann erkennen, dass der größere Kreis im Verhältnis zu seiner gesamten Umfangslänge einen deutlich geringeren Teil der schraffierten Fläche überdeckt als der kleinere. Dies kann man auch dadurch formal nachweisen, dass man zeigt, dass der Quotient aus dem Teil der Kugeloberfläche, in dem kanonische Relationen anwendbar sind und

der Gesamtkugeloberfläche bei steigender Distanz gegen null konvergiert.

$$\lim_{dist \to \infty} A_{covered}/A_{total} = 0$$

Eine Kugel mit Radius *dist* hat eine Oberfläche von

$$A_{total} = 4\pi dist^2$$

Der Anwendbarkeitsbereich für die kanonischen Relationen bei der Entfernung *dist* besteht aus 6 Rechtecken mit gekrümmter Oberfläche. Länge und Breite dieser Rechtecke bestimmt man aus der Länge des entsprechenden Kugelumfangs an dieser Stelle ($U = \alpha dist$). Siehe dazu auch Abbildung 3.17.

$$Dim := \{Length, Width, Depth\}$$
$$A_{covered} = \sum_{i,j \in Dim, j \neq i} 2dist \bullet \arcsin(\tfrac{i/2}{dist}) \bullet dist \bullet \arcsin(\tfrac{j/2}{dist})$$

Insgesamt ergibt sich dann:

$$\lim_{dist \to \infty} A_{covered}/A_{total} =$$
$$\lim_{dist \to \infty} \frac{2 \sum_{i,j \in Dim, j \neq i} \arcsin(\frac{i/2}{dist}) \bullet \arcsin(\frac{j/2}{dist})}{4\pi} = 0. (q.e.d.)$$

Gleiches gilt bei abnehmender Größe des Referenzobjektes ($i, j \to 0$).

Um diesen Effekt zu kompensieren, könnte man Entfernung und Objektgröße bei Auswahl der geeigneten Relation mitberücksichtigen.

3.4.5.5 Berechnungsverfahren von Gapp für projektive Relationen

Das Verfahren von Gapp (siehe [Gap97]) funktioniert ganz ähnlich wie sein Algorithmus zur Berechnung von Distanzrelationen, d.h. es wird auch wieder der skalierte Differenzvektor $\vec{dist}_{scaled}$ berechnet (siehe Abschnitt 3.4.4.1). Der Anwendbarkeitsgrad bestimmt sich dann einfach durch

$$ad = f(\angle(\vec{dist}_{scaled}, \vec{Rel}))$$

Wie Gapp in seiner Dissertation ausführt ([Gap97]), ist f näherungsweise eine lineare Funktion. Somit gilt:

$$f(x) = \frac{\max\{0, \pi/2 - x\}}{\pi/2}$$

Das Problem beim naiven Verfahren (siehe Abschnitt 3.4.5.2) wird dadurch vermieden, dass eine Koordinate des Differenzvektors erst dann einen von null verschiedenen Wert annimmt, wenn sie die Hälfte der Ausdehnung des Referenzobjektes in der jeweiligen Dimension überschreitet, was durchaus plausibel zu sein scheint.

Fragwürdig scheint dagegen die hier wie bei der Distanzberechnung vorgenommene Skalierung der Distanz. Diese kann zu recht merkwürdigen Resultaten führen. In Abbildung 3.18 besitzt das zu lokalisierende Objekt bei voller Skalierung mit der Ausdehnung des Referenzobjektes die Koordinaten (-0.5;4); würde also als *vor* dem Referenzobjekt

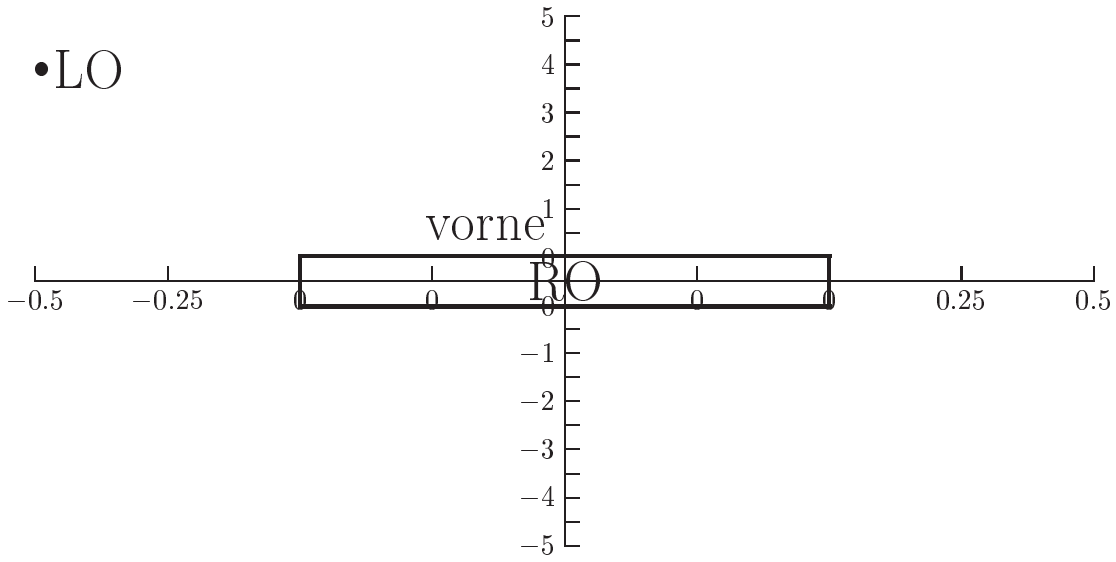

Abbildung 3.18: Probleme bei der Raumskalierung

eingestuft, obwohl die korrekte Bezeichnung eher *schräg vor* wäre. Gapp versucht diesen Effekt zwar durch einen Ausgleichsfaktor zu kompensieren. Trotzdem stellt sich die Frage, ob die Skalierung in diesem Fall konzeptuell sinnvoll ist, auch weil der Betrachter eventuell gar keine Möglichkeit die Tiefe des Objektes zu bestimmen, z.B. bei aneinanderliegenden Häusern [10]. Verzichtet man dagegen auf eine Skalierung liefert dieses Verfahren durchaus brauchbare Ergebnisse.

3.4.5.6 Weitere Berechnungsverfahren

In erster Linie für Straßenzüge als Referenzobjekte ist das in [ZF93] beschriebene Verfahren gedacht. Zusätzlich besteht dabei die Möglichkeit aus bereits berechneten Relationen andere zu inferieren.

Für denselben Anwendungszweck ist auch der Algorithmus von [LR93] gedacht. Zur Bestimmung einer geeigneten Relation wird die Eigenschaft von Winkeln (rechter, stumpf oder spitz), sowie die Orientierung von Dreiecken verwendet. Die Straßen werden dabei durch Strecken zusammengesetzt, für die jeweils Anfangs- und Endpunkt gegeben sind.

3.4.5.7 Sonderfälle

In manchen Fällen kann es vorkommen, dass sich das zu lokalisierende Objekt im begrenzenden Quader des Referenzobjektes befindet. Nach dem Berechnungsverfahren von Gapp (siehe Abschnitt 3.4.5.5 und [Gap97]) entspricht der skalierte Differenzvektor $dist_{scaled}$ in diesem Fall dem Nullvektor. Damit ist die Berechnung des Abweichungswinkels nicht

[10] Nötig wäre hier eine psychologische Validierung mit einem repräsentativen Test

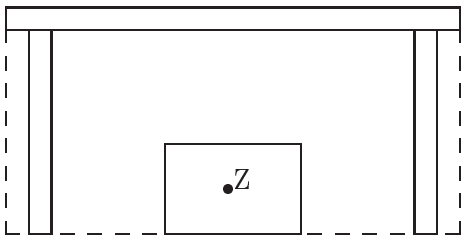

Abbildung 3.19: Eine Kiste unter einem Tisch

durchführbar und damit keine der oben beschriebenen projektiven Relationen anwendbar. Gapp ist der Ansicht, dass in diesem Fall eine Anwendung externer projektiver Relationen gar nicht erwünscht sondern interne Relationen (Abschnitt 3.4.1) vorzuziehen sind. Dies ist aber nicht immer der Fall, wie in Abbildung 3.19 illustriert ist. Dort befindet sich die Kiste *unter* dem Tisch, obwohl sie sich innerhalb des den Tisch umgebenden Quaders befindet. Eine mögliche Vorgehensweise besteht darin, in diesem Fall das zu lokalisierende Objekt aus dem Quader heraus zu verschieben, bis wieder eine externe Relation anwendbar ist. So verfährt man beispielsweise im System CITYTOUR (siehe [ABGT85]). Dort wird jedes Objekt durch ein zweidimensionales Polygon beschrieben. Ein innerhalb des begrenzenden Rechtecks liegender Punkt wird dann aus diesem heraus verschoben, wenn er sich nicht in dem entsprechenden Polygon befindet. Dieses Verfahren auf den dreidimensionalen Raum zu erweitern würde allerdings auf Grund der oft recht hohen Komplexität von 3D-Modellen zu einem enormen Rechenaufwand führen.

In den vorliegenden Fällen wurde stets eine intrinsische Orientierung des Referenzobjektes vorausgesetzt (siehe Abschnitt 3.4.5.1) Soll dagegen die deiktische Orientierung verwendet werden, wird die Orientierung des Betrachters am Referenzobjekt gespiegelt (siehe [HRA90]). Das Bezugsobjekt ist also nun dem Betrachter entgegengesetzt orientiert, zugleich werden linke und rechte Seite vertauscht. Liegt demnach ein zu lokalisierendes Objekt links(rechts) vom Betrachter, befindet es sich dann rechts(links) vom Referenzobjekt.

Normalerweise sind *recht*, *links*, *vor* und *hinter* horizontale Relationen, *über* und *unter* dagegen vertikale. Dies muß aber nicht notwendigerweise so sein. So sind Fahrzeuge beispielsweise normalerweise nach ihrer Bewegungsrichtung hin orientiert. Man betrachte nun ein Flugzeug, das fast vertikal in die Höehe fliegt. Bei solch einem Flug wäre nun *vor* dem Flugzeug auf dessen Flugbahn, das heißt die Richtungsvektoren für *über* und *vor* könnten jetzt ganz dicht zusammenliegen, im Extremfall sogar zusammenfallen. Dies kommt dadurch, dass im Unterschied zu *vor*, *hinter*, *rechts* und *links* die Relationen *über* und *unter* von der Orientierung des Referenzobjektes unabhängig sind.

3.4.6 Berechnung der Relation *auf*

Die Relation auf ist anwendbar, wenn sich ein zu lokalisierendes Objekt über einem Referenzobjekt befindet und zusätzlich ein physischer Kontakt zwischen beiden besteht (siehe [Gap97]). Sie kann transitiv anwendbar sein oder auch nicht. So würde man ein Buch, das

sich auf einem anderen Buch befindet, das wiederum auf dem Tisch liegt, als auch auf dem Tisch befindlich einordnen, ein Glas, das auf dem Tisch steht, der auf dem Boden steht, befindet sich aber nicht auf dem Boden (siehe auch [Gra00]).

In [Gap97] wird $on(LO, RO)$ approximiert durch

$$on(LO, RO) = at(LO, RO) \wedge above(LO, RO)$$

d.h. die Ausdehnung des zu lokalisierenden Objektes wird ganz ignoriert, allein dessen Schwerpunkt wird bei der Berechnung berücksichtigt. Dies hat den Vorteil der einfachen Berechnung, allerdings liefert das Verfahren auch in vielen Fällen falsche Ergebnisse. So kann bei gleicher Lage des Schwerpunktes des zu lokalisierenden Objektes, abhängig von dessen Ausdehnung, es sich entweder *auf* oder nicht *auf* dem Referenzobjekt befinden.

In [Blo99] wird dieses Problem dadurch vermieden, dass statt dem Schwerpunkt, der dem Referenzobjekt am nächsten liegende Punkt des zu lokalisierenden Objektes zur Relationsbestimmung herangezogen wird. Allerdings hat dies den Nachteil, dass es nun nicht mehr möglich ist, zu unterscheiden, ob sich ein Objekt vollständig oder nur zum Teil auf dem Referenzobjekt befindet.

Für die Berechnung dieser Relation ist daher wohl doch eine exaktere Objektapproximierung vorzuziehen (siehe auch [Krü01]). Dies kann beispielsweise der begrenzende Quader sein.

3.5 Domänenvisualisierung und Interaktion

Neben der natürlichsprachlichen Lokalisationsbeschreibung besteht auch die Möglichkeit der grafischen Darstellung. Siehe [Hor01b] und [RD00] zur Diskussion der Vor- und Nachteile beider Verfahren. Eine grafische Darstellung bietet dabei eine ganze Reihe von Möglichkeiten, von der einfachen Markierung des gesuchten Objektes bis zu einer 3D-Animation, in der ein vom System gesteuerter Agent den Weg vom Benutzerstandpunkt zum gesuchten Objekt abläuft. Falls der Agent dazu eingesetzt werden soll, dem Anwender Informationen zu präsentieren, nennt man ihn auch Präsentationsagent. Er kann dazu Wissen über den Benutzer und dessen Ziele (siehe Seite 118 in [Bor94]) verwenden. Zusätzlich zu der Animation können Informationen über am Weg liegende Objekte eingeblendet werden.

Sinnvollerweise sollte der Benutzer die 3D-Welt aus mehreren Perspektiven gleichzeitig betrachten können und die angezeigte Information in verschiedener Ausführlichkeit abrufbar sein (siehe [Ner95]). Objekte, die beim Entwurf einer solchen 3D-Umgebung berücksichtigt werden müssen, sind

- der Benutzer
- ein oder mehrere Agenten
- der Rest der 3D-Welt, d.h. die virtuelle Umgebung im engeren Sinne

Man kann dabei zwischen mehreren Arten von Interaktionsbeziehungen unterscheiden:

- Umgebung und Agent

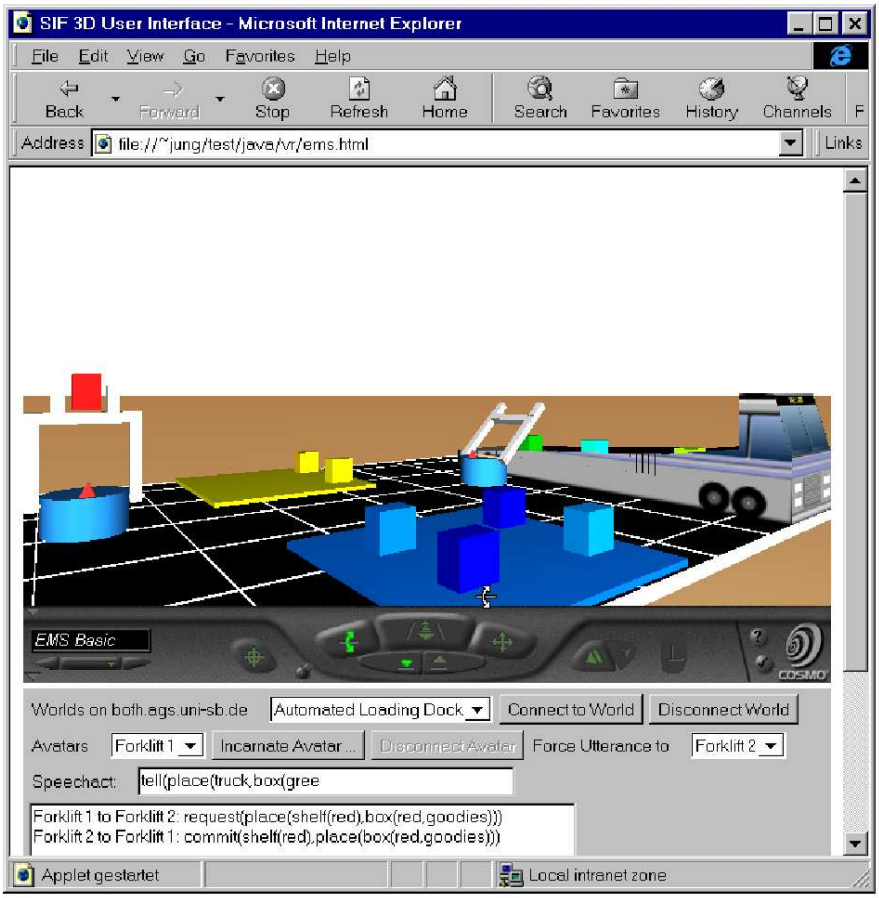

Abbildung 3.20: SIF-VW

- Interaktion zwischen Agenten
- Umgebung und Benutzer
- Benutzer und Agent

Interaktionsbeziehungen zwischen Agenten werden in Multiagentensystemen betrachtet. Ein solches Multiagentensystem für virtuelle Umgebungen ist beispielsweise SIF-VW[11]. Da hier aber immer nur ein Agent verwendet wird, soll dies im folgenden allerdings nicht weiter betrachtet werden. Interaktionen zwischen Benutzer und Agent ist Gegenstand einer anderen Diplomarbeit im Projekt REAL (siehe System AJAPA [12] in [Bre01]).

[11] SIF ist das Akronym für **S**ocial **I**nteraction **F**ramework (siehe [GFS+00] und [SFGJ99] sowie Abbildung 3.20)

[12] AJAPA steht für **A** **J**ava **P**resentation **A**gent

3.5.1 Interaktion zwischen Agent und Umgebung

Eine intelligente Umgebung soll sich auf den Agent einstellen und auf seine Aktionen angemessen reagieren können. Dazu gehören Tätigkeiten wie das automatische Öffnen der Türen, wenn der Agent den Raum dahinter betreten will sowie das automatische Laden und Entladen von Räumen. Eine Möglichkeit das letztere zu realisieren, besteht darin, einen Sichtbarkeitsgraph anzulegen, der angibt welche Nachbarräume von einem Raum aus jeweils eingesehen werden können (siehe [Zer99]). Man braucht dann immer nur die Geometrien dieser Räume vorzuhalten.

Solch eine virtuelle intelligente Umgebung hat dabei durchaus Ähnlichkeiten mit der realen Welt. So wird an intelligenten Räumen geforscht, die automatisch das Licht an- und ausschalten, wenn eine Person diese betritt bzw. wenn alle Personen diesen verlassen haben (siehe [Ben00]), was in der virtuellen Umgebung in etwa dem dynamischen Laden von Räumen entspricht. Automatisch auf- und zugehende Türen sieht man öfters in Supermärkten.

Weiterhin sollte ein sich in der virtuellen Umgebung aufhaltender Agent mit Sensoren ausgestattet sein, mit denen er seine Umgebung abtasten, in der Nähe liegende Geometrien erkennen und ihre semantischen Annotationen auslesen kann (siehe [BBK+00]). Auf diese Art kann er verschiedene Informationen über seine Umgebung erlangen. Annotationen können beinhalten, zu was für einem Typ von Objekt die Geometrie gehört (z.B. Tür), sie können den Agenten bei Problemlösungen unterstützen, die Funktion von Gegenständen beschreiben oder auch bestimmte Gefühle bei ihm erwecken. So könnte ein dunkles Gewölbe beispielsweise Angst kodieren (siehe [DHR98]).

3.5.2 Interaktion zwischen Benutzer und Umgebung

Die im System gespeicherte semantische Information kann benutzt werden, um Informationen über bestimmte Objekte der 3D-Umgebungen einzublenden. Die Verknüpfung des entsprechenden Textfeldes mit dem annotierten Objekt kann zum einen durch eine Verbindungslinie realisiert werden oder zum anderen dadurch, dass das Textfeld in die Nähe des Objektes plaziert wird (siehe [Zim93]). Textfelder in einem 3D-Szenario sollten nach [Bar01] zur besseren Lesbarkeit und Aussagekraft folgende Eigenschaften haben.

- Sie sollten mit der entsprechenden Geometrie verknüpft werden und nur sichtbar sein, falls das annotierte Objekt im Sichtfeld liegt.
- Sie sollten immer zum Benutzer orientiert sein. Dies kann beispielsweise durch Benutzung eines Billboards (siehe [Bou99]) erreicht werden.
- Sie sollten immer eine konstante Größe besitzen, wie Abbildungen 3.21 (a-d) aus [Bar01] illustrieren.
- Sie sollten sich immer im Vordergrund befinden.

Die semantische Information kann aber auch dazu verwendet werden, die Relevanz eines Objektes festzulegen, so dass für manche Objekte detailliertere Informationen angezeigt

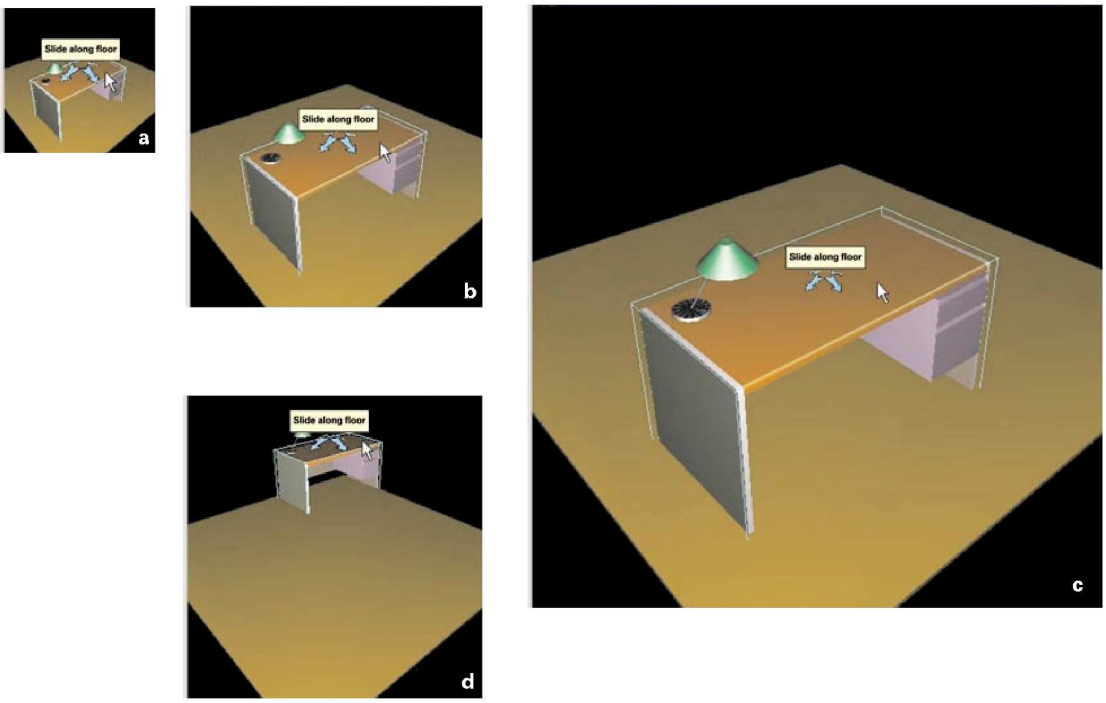

Abbildung 3.21: Objektannotation

werden als für andere (siehe auch [Wex93]). So können zum Beispiel in einer Übersichtskarte bei für den Benutzer interessanten Objekten mehr Informationen angegeben werden. Die wichtigsten Objekte in einem Navigationssystem sind dabei sicherlich die vom Benutzer angegebenen Zielobjekte. Weitere interessierende Objekte sind aufgrund allgemeiner Erfahrungen oder auch durch ein Benutzermodell bestimmbar. Am wenigsten Relevanz besitzen Dinge, zu denen der Benutzer normalerweise gar keinen Zugang hat. Diese können eventuell in einer Karte durch *Black Boxes* nur angedeutet werden. Zusätzlich kann ein Objekt detaillierter dargestellt werden, wenn der Benutzer sich diesem nähert.

3.6 Zusammenfassung und Forderungen

Zuerst wurde die semantische Repräsentation von verschiedenen Wegauskunfts- und Objektlokalisationssystemen untersucht. Anschließend folgte eine Beschreibung der Probleme, die dadurch entstehen können, dass Objekte Bestandteil mehrerer übergeordneter Objekte sein dürfen. Eine Konsequenz, die sich daraus ergibt, besteht darin, dass der einen Raum begrenzende Quader nicht mehr so einfach zu berechnen ist.Daher muss dafür explizit ein Berechnungsverfahren entwickelt werden.

Weiterhin ist eine für die semantische Repräsentation von Gebäuden geeignete Ontologie aufzubauen, die die in einem Gebäude vorkommenden Objekte wie Etage, Raum, Boden, Decke, Möbel u.s.w. beinhaltet.

Außerdem müssen im folgenden zwei verschiedene Suchverfahren für die Suche nach Objekten in der Wissensbasis entwickelt werden, zum einen ein möglichst performante zum anderen eine möglichst flexible Suche. Für die erstere empfiehlt sich dabei die Verwendung von B*-Bäumen oder Hashtabellen, mit Hilfe derer ein Objekt durch ein dasselbe eindeutig identifizierendes Schlüsselattribut aufgefunden werden kann.

Eine möglichst flexible Suche sollte berücksichtigen, dass die Suchparameter natürlichsprachlich eingegeben sein könnten. Daher empfiehlt sich die Verwendung einer Tippfehlerkorrektur sowie weitere im nächsten Kapitel beschriebene Maßnahmen, die die Suche für den Benutzer möglichst einfach und intuitiv machen. Für eine natürlichpsrachliche Eingabe benötigt man einen Parser, der aus einem vom Benutzer eingegebenen Eingabestring die entsprechenden Suchattribute ermittelt. Dies kann beispielsweise durch Pattern-Matching wie in CITYGUIDE (siehe [Mül88]) oder auch durch eine Schlüsselwortsuche wie in dem System FAIRCAR (siehe [Wah00b]) erfolgen.

Eine natürlichsprachliche Lokalisationsbeschreibung macht die Berechnung von räumlichen Relationen erforderlich. Brauchbare Verfahren dafür findet man in [Gap97]. Dabei müssen einige Schwachstellen behoben, für die Relation *auf* ein gänzlich neues Verfahren entwickelt, sowie einige dort nur oberflächlich behandelte Fälle berücksichtigt werden.

Zuletzt soll eine virtuelle Umgebung entwickelt werden, die intelligent auf einen sich in dieser befindlichen Agenten reagiert. Dies beinhaltet Aspekte wie das automatische Öffnen und Schließen von Türen sowie das dynamische Laden- und Entladen von Geometrien. Zusätzlich sollen Objektannotation eingeblendet werden können, deren Konzeption die in [Bar01] (siehe Abschnitt 3.5.2) beschriebenen Prinzipien berücksichtigt.

Kapitel 4

Konzepte für die Wissensverarbeitung in der Gebäudenavigation

In diesem Kapitel werden auf den Verfahren aus den vorigen Kapiteln aufbauende Konzepte entwickelt, wie eine semantische und räumliche Wissensverarbeitung in einem Navigationssystem realisiert werden kann.
Im ersten Abschnitt erfolgt dabei eine Beschreibung der in dieser Arbeit verwendeten Konzepte für die Wissensrepräsentation.
Der zweite Abschnitt erläutert verschiedene Suchveralgorithmen zum Auffinden von Objekten in der Wissensbasis einschließlich einer natürlichsprachlichen Suche.
Im dritten Abschnitt werden verschiedene Berechnungsverfahren für räumliche Relationen beschrieben, welche für eine sprachliche Lokalisationsbeschreibung benötigt werden.
Der vierte Abschnitt beschreibt schließlich die Konzeption einer intelligenten virtuellen Umgebung, die verwendet werden kann, um den gefundenen Weg anhand einer Animation darzustellen. Außerdem werden verschiedene Vorgehensweisen zum Einblenden von Objektannotationen sowie zur Interaktion mit dem Benutzer entwickelt.

4.1 Wissensrepräsentation

4.1.1 Objektrepräsentationen

In einem Gebäudenavigationssystem sind drei verschiedene Arten der Objektrepräsentation von Bedeutung (siehe Abbildung 4.1). Diese sind

- die grafische Repräsentation
- die Objektidealisierung
- die semantische Repräsentation

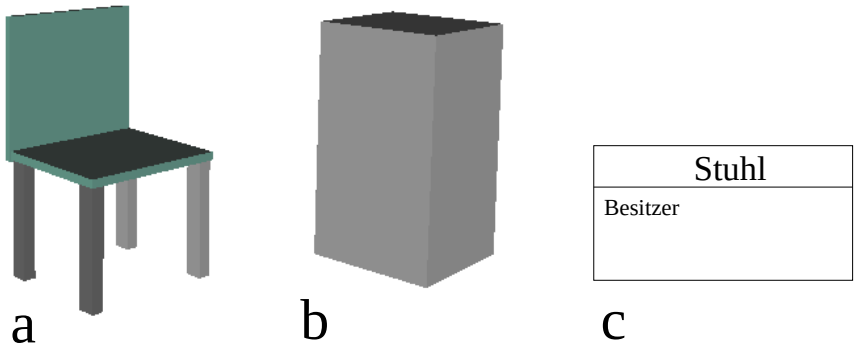

Abbildung 4.1: Objektrepräsentationen

Die grafische Repräsentation (siehe Abbildung 4.1a) besteht aus dem zu einem Objekt erstellten 3D-Modell. Dabei sollten Methoden definiert sein, um die zugehörige semantische Repräsentation zu bestimmen.

Die Objektidealisierung (siehe Abbildung 4.1b) besteht aus einer approximativen geometrischen Beschreibung (siehe Abschnitt 3.4.2). In dieser Arbeit werden Objekte durch den begrenzenden Quader sowie durch ihren Mittelpunkt approximiert. Die Objektidealisierung wird in erster Linie zur Berechnung von räumlichen Relationen benötigt. Sie kann dabei automatisch aus der grafischen Repräsentation berechnet werden.

Die semantische Repräsentation (siehe Abbildung 4.1.1c) enthält Informationen über die Klassen, in denen sich ein Objekt befindet sowie über seine Zusammensetzung aus anderen Objekten. Zusätzlich können bestimmte das Objekt beschreibende Attribute repräsentiert werden. Von der semantischen Repräsentation sollte eine Verknüpfung zu der entsprechenden grafischen Repräsentation existieren.

Im folgenden wird dabei zunächst die semantische Repräsentation von Objekten genauer betrachtet.

4.1.2 Identifizierung von Objekten

Eine Methode wie Objekte identifiziert werden können, besteht darin, dass jedem Objekt eine eindeutige Nummer zugewiesen wird. Wenn nun ein neues Objekt dazukommt, zählt man einfach eine Zahl hoch und vergibt diese als nächste ID. Für einen Computer ist dies eigentlich die ideale Form der Identifizierung. Für einen menschlichen Anwender dagegen wären Lokalisationsanfragen der Art

„Wo befindet sich Objekt Nummer 1432? “

doch ziemlich unkomfortabel. Daher ist es zu empfehlen andere Möglichkeiten der Objektidentifizierung einzuführen. Denkbar sind dabei:

- Klasse eines Objektes (Beispiel: Modegeschäft, Post)

- Namen eines Objektes (Beispiel: McDonalds, Palmers)
- Klasse und Nummer eines Objektes (Beispiel: Raum 114)
- Besitzer und Klasse eines Objektes (Beispiel: Das Büro von Herrn Baus)
- Klasse und Namen eines Objektes (Beispiel: Modegeschäft Palmers)

4.1.3 Generierung einer textuellen Objektbeschreibung

Wie im vorigen Abschnitt erwähnt, können Objekte durch Namen, Klasse, Nummer und Besitzer beschrieben werden, wobei nicht immer alle dieser Bezeichner angegeben werden müssen. Einfach ist die textuelle Ausgabe, falls nur die Klasse oder der Name für dieses Objekt definiert ist.

Sind Name und Klasse bekannt, bestehen im Deutschen grundsätzlich zwei Möglichkeiten der linguistischen Beschreibung. Sie kann entweder in der Form <Klasse> <Name> (z.B. Villa Kunterbunt) oder <Name>-<Klasse> (z.B. Watergate-Affäre) erfolgen.

Wird das Objekt durch Klasse und Nummer bezeichnet, kann es der Fall sein, dass sich die Nummerierung nicht auf die Klasse selber sondern auf eine Oberklasse bezieht. Betrachte man beispielsweise einen Raum mit Nummer 124, der ein Sekretariat ist. Korrekt wäre hier normalerweise die Bezeichnung *Raum 124, Sekretariat* und nicht *Sekretariat 124*. Falls das Objekt zusätzlich noch einem Besitzer zugeordnet ist, teilt man die Beschreibung oft in zwei Nominalphrasen auf, eine, die die Oberklasse mit der Numerierung enthält und eine die aus der Unterklasse und dem Besitzer besteht.
Beispiel: *Raum 124, Sekretariat von Prof. Einstein*

4.1.4 Spezialisierungsattribut

Unter Umständen entstehen beim Aufbau einer Ontologie zahlreiche Klassen, bei denen nur der Objekttyp von Bedeutung ist. Für alle diese Dinge extra eine Klasse zu definieren, wäre recht umständlich und unübersichtlich. Dies läßt sich vermeiden, indem man für alle Klassen ein zusätzliches Attribut einführt, mit dem sich der Typ eines Objektes noch weiter spezialisieren läßt. Dadurch können zahlreiche Klassen eingespart werden. Einen Nachteil besitzt dieser Ansatz allerdings. Die durch dieses Attribut eingeführten (virtuellen) Klassen lassen sich nähmlich nicht weiter vererben, d.h. dieser Mechanismus kann nur bei Blattknoten im Vererbungsbaum zum Einsatz kommen. Dieses Prinzip wird in Abbildung 4.2 dargestellt. Die gestrichelten Linien symbolisieren dabei eine Instantiierungsbeziehung zwischen einem Objekt und seiner Klasse. Dies ist nicht Teil des UML-Standards, da dort keine Darstellungsform für solch eine Beziehung definiert wurde.

Durch dieses Konzept läßt sich auch das in Abschnitt 2.1.2 angesprochene Problem der Mehrfachinstantiierung lösen, sofern die Klassen, deren Instanz das Objekt sein soll, durch solch ein Typattribut modelliert worden sind (siehe Abbildung 4.3). Dazu spezifiziert man im Typattribut der Instanz einfach, zu welchen Klassen sie gehören soll.

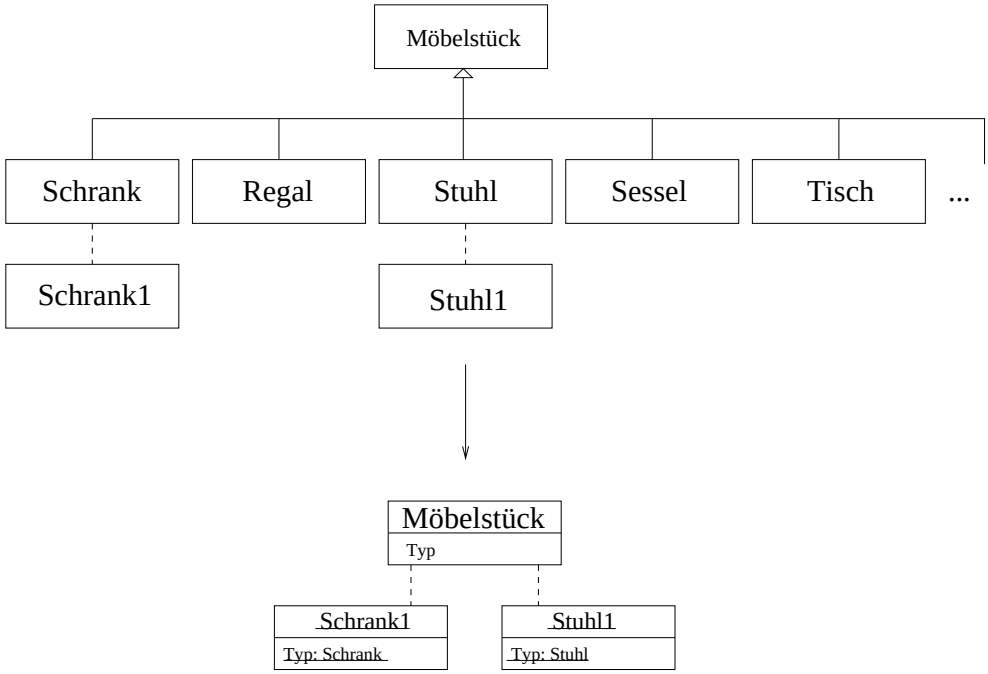

Abbildung 4.2: Spezialisierungsattribut

4.1.5 Objektgraph

Das Äquivalent zu der Gruppierung in der Computergrafik (siehe Abschnitt 2.2.1) ist in der semantischen Beschreibung die Aggregation (siehe Abschnitt 2.1.4). Damit lassen sich elementarere Objekte zu komplexeren zusammenfassen. Unterschiedlich zum Szenegraph darf ein und dasselbe Objekt durchaus Bestandteil mehrerer übergeordneter Objekte sein. So kann eine Wand beispielsweise zu mehreren Räumen gehören. Aufgrund dieser Tatsache erscheint eine Objektrepräsentation geometrischer und semantischer Repräsentation in einem gemeinsamen Graphknoten nicht sinnvoll (siehe auch Seite 101 in [Krü01]).

Allein durch Aggregationsbeziehungen können meist nicht alle Objekte in einem Graph repräsentiert werden, da es oft nicht möglich ist, zwischen zwei Objekten eine solche Kompositionsbeziehung herzuleiten. Man kann sich nun dadurch helfen, dass man in Analogie zum Szenegraph als zusätzliches Objekt das Universum einführt und alle Objekte als dessen Bestandteil definiert. Somit wäre das Universum einzige Wurzel in einem zusammenhängenden aggregierten Graph.

Alternativ kann auch eine zusätzliche Relation hinzugenommen werden. Dabei empfiehlt sich die Verwendung von *innerhalb*, da es sich damit nun sehr leicht bestimmen lässt, welche Objekte sich innerhalb eines anderen befinden und umgekehrt. Dies ist für ein Gebäudenavigationssystem von großem Vorteil, weil dort oft der Raum ermittelt werden muss, in dem sich ein gesuchtes Objekt aufhält.

Durch diese beiden Beziehungen lassen sich nun alle Objekte durch einen einzigen zu-

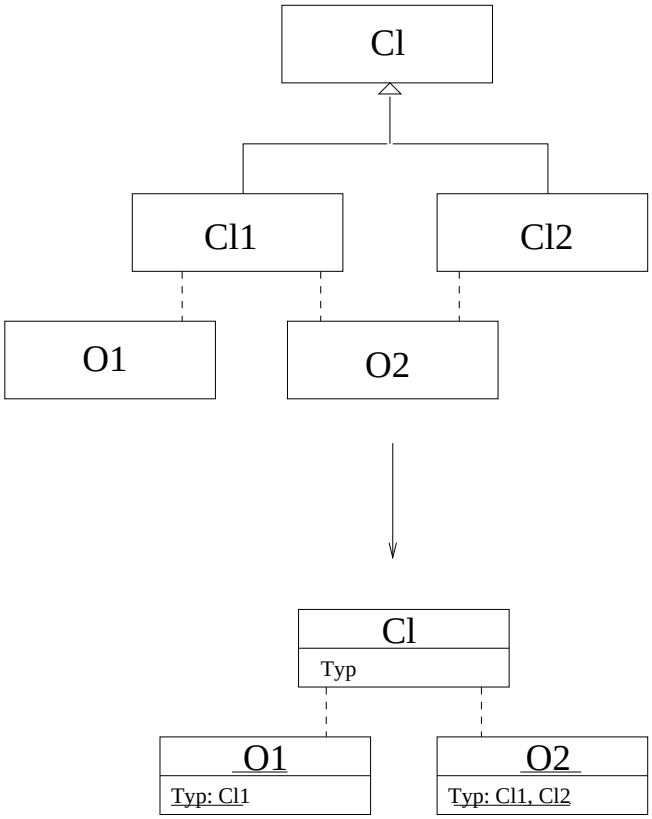

Abbildung 4.3: Mehrfachinstantiierung

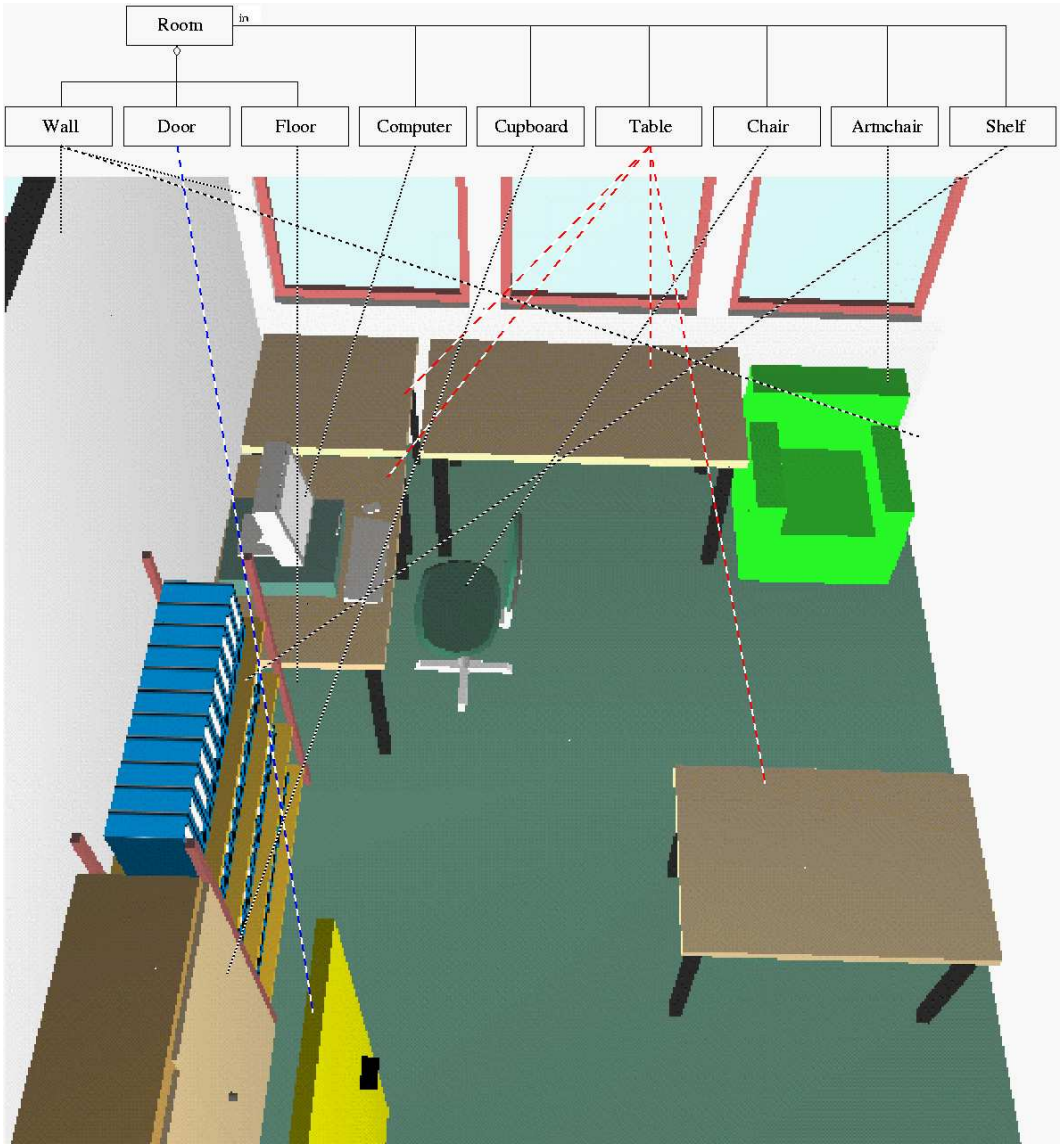

Abbildung 4.4: Objektgraph

sammenhängenden azyklischen Graphen mit einer Wurzel beschreiben. Bei einem Gebäudenavigationssystem besteht diese Wurzel normalerweise aus dem für die Navigation relevanten Gebäude. Dieser Graph wird im folgenden auch als **Objektgraph** bezeichnet. Ein Beispiel für einen Objektgraphen findet sich in Abbildung 4.4.

4.2 Suche nach Objekten

Bei der Suche nach Objekten in der Wissensbasis existieren zwei miteinander in Konflikt stehende Ziele. Zum einen soll das gesuchte Objekt möglichst schnell aufgefunden werden können. Zum anderen ist es wünschenswert, wenn das System die Benutzeranfrage möglichst flexibel bearbeiten und auch Tippfehler korrigieren kann. Daher kommen hier zwei verschieden Suchverfahren zum Einsatz.

Soll eine Suchanfrage möglichst performant ausgeführt werden, wird das Objekt wie in Abschnitt 3.3 aus einer Hashtabelle ausgelesen. Der dazu benötigte Schlüssel berechnet sich wie folgt:

- explizit gespeicherte ID, falls definiert
- sonst Nummer des Objektes, falls definiert
- sonst Name des Objektes, falls definiert
- sonst Klassenbezeichnung des Objektes

Dieses Vorgehen hat den Vorteil, dass in vielen Fällen Objekte statt durch eine kryptische ID einfach durch ihren Namen bzw. ihre Nummer referenziert werden können. Nur wenn durch die letzten drei Bezeichner keine eindeutige Identifizierung möglich ist, muss explizit eine ID definiert werden.

Obwohl diese Suchanfragemöglichkeit schon bedeutend einfacher ist als durch die Eingabe irgendeiner Zahl, wie dies oft bei Datenbanksystemen realisiert ist [1], setzt es doch voraus, dass dem Anwender bekannt ist, ob für ein bestimmtes Objekt eine Nummer oder ein Name definiert ist. Außerdem kann ein Objekt damit nicht über seinen Besitzer ermittelt werden. Diese Probleme entfallen alle bei der im folgenden beschriebenen Fuzzy-Suche.

4.2.1 Konzeption der Fuzzy-Suche

Ein zu suchendes Objekt kann bei Benutzung der Fuzzysuche durch seinen Namen, Nummer, Besitzer und seine Klasse beschrieben werden. Dabei muss berücksichtigt werden, dass nicht alle diese Informationen bekannt sein müssen. So kennt der Benutzer vielleicht nur die Nummer und Klasse eines Objektes, nicht jedoch dessen Namen. Grundsätzlich sieht eine solche Suche nun so aus, dass jedes in Frage kommende Element der Wissensbasis mit den angegebenen Suchattributen verglichen wird. Um eine zufriedenstellende Performance zu erreichen, sollten manche Objekte allerdings von vornherein ausgeschlossen werden. So ist es beispielsweise recht unwahrscheinlich, dass jemand eine bestimmte Wand oder einen Stuhl sucht.

[1] z.B. Kundennummer, Produktnummer, ...

Bei der Benutzung der oben angegebenen Vergleichsattribute (im folgenden auch als Suchmuster bezeichnet) können die folgende Fälle auftreten:

- Vergleich von Personennamen. Das ist der Fall, wenn nach dem Besitzer eines Objektes gesucht werden soll.
- Vergleich von Objektnamen
- Vergleich von Klassennamen
- Vergleich von Nummern, die das Objekt identifizieren, z.B. *Raum 121*

Werden die Suchparameter direkt vom Benutzer eingegeben ist die Verwendung einer Tippfehlerkorrektur zu empfehlen (siehe Abschnitt 3.3.2.1).

Beim Vergleich von Klassennamen muß überprüft werden, ob die Klasse im Suchmuster eine Generalisierung (siehe Abschnitt 2.1.2) der Klasse des Vergleichsobjektes ist. Wird beispielsweise ein Fahrzeug von Herrn Mampf gesucht und in der Wissensbasis ist ein *Auto* verzeichnet, das Herrn Mampf gehört, sollte dieses Auto als Ergebnis der Suchanfrage gefunden werden, da ein Auto eine Spezialisierung von einem Fahrzeug ist.

4.2.1.1 Vergleich von Eigennamen

Beim Vergleich von Eigennamen sollte berücksichtigt werden, dass der angegebene Name eventuell nicht vollständig spezifiziert sein muß. Beispielsweise könnte der Benutzer nach einem in der Wissensbasis gespeicherten *Hadschi Halef Omar Ben Hadschi Abbul Abbas ibn Hadschi Dawuhd al Gossarah* einfach mit „Wo ist Halef? "fragen. Obwohl diese beiden Zeichenketten sich deutlich unterscheiden, wird beidesmal dieselbe Person damit bezeichnet. Daher sollten diese zwei Eingaben miteinander identifiziert werden.

Auf jeden Fall unterschiedlich sind zwei Namensbezeichnungen, wenn die beiden Nachnamen nicht übereinstimmen.

Bei den Vornamen besteht die zusätzliche Schwierigkeit darin, dass eine Person mehrere davon besitzen kann. Damit beide Personen als identisch betrachtet werden können, wird in diesem Fall verlangt, dass alle Vornamen der einen Person in der Liste der Vornamen der anderen auftauchen oder umgekehrt.

Noch schwächer sind die Anforderungen bei der Übereinstimmung bei den Titeln, so können *Herr Einstein* und *Prof. Einstein* dieselbe Person bezeichnen, obwohl die Titelbezeichnungen (Herr, Prof.) unterschiedlich sind. Eine notwendige Bedingung besteht allerdings darin, dass eine Person mit weiblicher Form einer Anrede (z.B. Frau) nicht mit einer Person mit männlicher Anredeform (z.B. Herr) identifiziert werden darf. Dies erfordert allerdings eine explizite semantische Repräsentation der verschiedenen Titelbezeichnungen. Da dieser Fall in der Praxis nicht von besonders hoher Relevanz ist, wird dies hier nicht überprüft.

Im folgenden werden folgende Kurzschreibweisen verwendet:

- $GivNames(name)$ bezeichnet die Menge der Vornamen von $name$.
- $lastName(name)$ bezeichnet den Nachnamen von $name$.

- ε bezeichnet das leere Wort.

Die Äquivalenz zweier Eigennamen lässt sich dann folgendermaßen definieren:

Definition 4 *Zwei Eigennamen $name_1$ und $name_2$ sind äquivalent* $:\Leftrightarrow$

$$\begin{gathered}[(lastName(name_1) \neq \varepsilon \wedge lastName(name_2) \neq \varepsilon) \\ \Rightarrow lastName(name_1) = lastName(name_2)] \wedge \\ (GivNames(name_1) \subseteq GivNames(name_2) \vee \\ GivNames(name_2) \subseteq GivNames(name_1)) \wedge \\ [(lastName(name_1) \neq \varepsilon \wedge lastName(name_2) \neq \varepsilon) \vee \\ (GivNames(name_1) \neq \emptyset \wedge GivNames(name_2) \neq \emptyset)]\end{gathered}$$

4.2.2 Natürlichsprachliche Suchanfragen

Der Benutzer hat die Möglichkeit, Lokalisationsanfragen nach Objekten in natürlicher Sprache zu stellen. Die hier vorgestellte natürlichsprachliche Eingabe beinhaltet die folgenden Funktionen:

- In einer Anfrage kann nach mehreren Objekten gesucht werden z.B. „Wo ist die Post, eine Sparkasse und McDonalds? “.
- Es kann ein Objekt über Name, Besitzer, Nummer, Klasse sowie durch einige Kombinationen davon identifiziert werden.
- Personenbezeichnungen können auf vielfältige Arten angegeben werden z.B. *Wo ist das Büro von Prof. Moriarty/Herr Moriarty/ Prof. John Moriarty?*.
- Für Objekt- und Personennamen ist eine Tippfehlerkorrektur integriert (siehe Abschnitt 3.3.2.1).
- Anfragen können umformuliert werden, falls kein passendes Objekt ausfindig gemacht werden konnte.
- Benutzung eines Synonymwörterbuches
- Eingaben sind in verschiedenen Sprachen möglich (Multilingualität). Dazu werden die Eingaben grundsätzlich ins Englische übersetzt. Die Übersetzung erfolgt dabei durch das Synonymwörterbuch der entsprechenden Sprache.
- Groß- und Kleinschreibung werden ignoriert.

Das Umformulieren von Anfragen bezieht sich ausschließlich auf Personenanfragen. So gibt der Benutzer, um das Büro von Prof. Einstein ausfindig zu machen, möglicherweise ein: „Wo ist Prof. Einstein? “Dies kann das System eigentlich nicht herausfinden, da nicht bekannt ist, wo eine Person sich zu einem bestimmten Zeitpunkt aufhält. Da Personen im Modell deswegen nicht direkt repräsentiert sind, wird das System dieses Objekt auch nicht in der Wissensbasis finden können. Solche Fragen werden nun umformuliert zu: „Wo ist

ein Objekt von Prof. Einstein? "Da jede Klasse Spezialisierung von *Objekt* ist, wird nun tatsächlich das *Büro von Prof. Einstein* als Ergebnis der Suche zurückgeliefert. Allerdings benötigt das System dazu auch etwas mehr Zeit, als wenn der Benutzer direkt nach dem *Büro von Prof. Einstein* gefragt hätte.

Man kann die sprachliche Eingabefunktion in folgende Unterpunkte zerlegen:

- Semantikextraktion aus der Eingabe
- Umwandeln der semantischen Information in ein Suchmuster
- Auffinden der Objekte mit Hilfe des Suchmusters. Dabei wird der in Abschnitt 4.2.1 geschilderte Mechanismus benutzt.
- mögliche Nachfrage beim Benutzer um Unklarheiten aufzulösen

Ambiguitäten können entstehen, wenn mehrere Objekte gefunden wurden, die der Benutzereingabe entsprachen. Allerdings ist ein explizites Nachfragen nicht immer gewünscht. Sucht der Benutzer einen Supermarkt, ist es ihm eventuell gleichgültig, welchen das System zurückliefert. Daher kann in einem solchen Fall die Selektion auch automatisch getroffen werden. Auswahlkriterien sind dabei geringe Distanz zum Benutzer, niedrige Preise oder gute Merkbarkeit des Weges. Alternativ wäre auch vorstellbar alle gefundenen Supermärkte zu markieren. Sucht der Anwender dagegen nach dem *Büro vom Otto* ist es im allgemeinen nicht gleichgültig, welcher Otto gemeint ist. In diesem Fall wäre ein Auswahlmenü sinnvoll, in dem das System alle möglichen *Ottos* auflistet, einschießlich weitergehender Informationen wie der entsprechenden Raumnummern, in dem der Benutzer das gesuchte Büro auswählen kann.

4.2.2.1 Semantikextraktion

Die Semantikextraktion kann in drei Teile unterteilt werden.

- Segmentierung der Eingabe
- Ermittlung von Klasse, Namen, Besitzer und Nummer
- Parsen von Eigennamen in Titel, Vornamen und Nachnamen. Dies ist nötig, um auch Namen miteinander zu identifizieren, die zwar dieselbe Person bezeichnen aber in einer anderen Weise (siehe Abschnitt 4.2.1.1).

Der Benutzer hat die Möglichkeit mehrere zu suchende Objekte gleichzeitig zu spezifizieren. Der Eingabestring muss in diesem Fall in Komponenten zerlegt werden, die jeweils nur noch ein einziges zu findendes Objekt beinhalten. Für diese Zerlegung werden bestimmte Trennzeichen definiert wie Konjunktionen (*und*), bestimmte Artikel und Kommas.

Nach der Segmentierung werden aus den Eingabezeichenketten Klasse, Name, Besitzer und Nummer der zu suchenden Objekte ermittelt. Hierzu werden verschiedene Textmuster definiert, aus denen man dann versucht, eines mit dem Eingabestring in Einklang zu

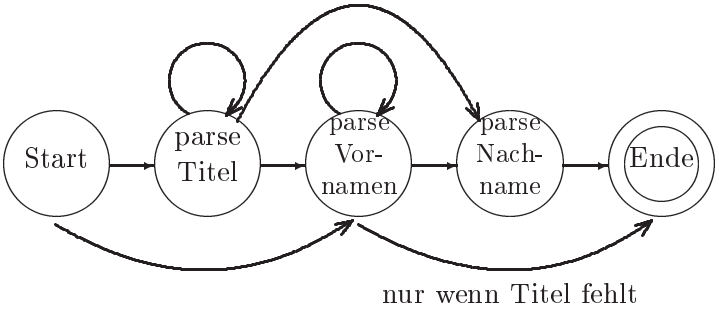

Abbildung 4.5: Parsen von Eigennamen

bringen. Dazu kommt eine Kombination aus Schlüsselworsuche (wie bei *Faircar*, siehe Abschnitt 3.3.2) und Pattern-Matching (wie bei CITYGUIDE, siehe Abschnitt 3.3.2) zum Einsatz. Die verwendeten Textmuster lauten:

1. [<Artikel>] <Klasse> von(m) <Person>
2. [<Artikel>] <Klasse> <Name>
3. [<bestimmter Artikel>] <Klasse> [Nummer] <Zahl>
4. <Person>'s <Klasse>
5. [<Artikel>] <Name>
6. [<Artikel>] <Klasse>

Das Büro von Heinz würde beispielsweise mit der 1.Vorlage identifiziert.

Parsen von Personennamen Das Parsen von Personennamen erfolgt über einen endlichen Automaten [2] (siehe Abbildung 4.5). Dabei werden verschiedene Gesetzmäßigkeiten der deutschen und englischen Sprache ausgenützt. So gelten folgende Regeln:

- Wenn ein Titel angegeben wurde, muss auch ein Nachname im Namen auftauchen.
- Wenn kein Titel angegeben wurde, muss mindestens ein Vorname im Namen vorkommen.
- Wenn Nachnamen aus mehreren Wörtern bestehen, beginnen sie normalerweise mit Präpositionen wie von, vor, zum , u.s.w..
- Titel werden stets zuerst genannt, dann die Vornamen und am Schluss der Nachname.

Den Anfang eines zusammengesetzten Nachnamens wie *vom See* bei *Lancelot vom See* erkennt man auch daran, dass das erste Wort vom Nachnamen klein geschrieben wird. Diese Gesätzmäßigkeit kann in dieser Arbeit allerdings nicht ausgenutzt werden, weil vom Benutzer keine korrekte Groß- und Kleinschreibung erwartet werden soll.

[2] zur Erläuterung eines endlichen Automaten siehe [Hot90]

4.3 Verarbeitung von räumlichen Wissen

Nachdem die Eingabe vom System eingelesen und geparst wurde, müssen die entsprechenden Objekte lokalisiert und eine natürlichsprachliche Lokalisationsbeschreibung generiert werden, wozu die Berechnung räumlicher Relationen erforderlich ist (siehe Abschnitt 3.4). Die Benutzung semantischer Informationen kann die Berechnung räumlicher Beziehungen dabei an vielen Stellen vereinfachen. Manche Eigenschaften, die nur schwierig oder auch gar nicht aus den Geometrien berechnet werden können, lassen sich leicht aus der semantischen Repräsentation ableiten.

So kann diese bei der Entscheidung hilfreich sein, ob Objekte eine intrinsische Front haben oder nicht. Wände und Bäume haben beispielsweise im allgemeinen keine Orientierung im Raum, Häuser, Schränke oder Regale dagegen schon. Wenn nun eine Wand bei einer Wegbeschreibung benutzt werden soll, kann allein aus der semantischen Information, dass dieses Referenzobjekt eine Wand ist, gefolgert werden, dass eine Benutzung einer intrinsischen Orientierung hier nicht in Frage kommt.

Der Schlußfolgerungsmechanismus (einschließlich Ausnahmebehandlung) läuft dabei ab wie in Abschnitt 2.1.2 beschrieben, d.h.
ein Objekt o besitzt eine intrinsische Orientierung $\Leftrightarrow$
$holds(o, intrinsicOrientation, true)$.

Zusätzlich kann man aus der semantischen Information inferieren, ob ein Objekt gut durch einen begrenzenden Quader approximiert werden kann. Bei Schränken und Räumen geht das normalerweise gut, bei Tischen oder Stühlen schlecht, da hier das Volumen des Tisches/Stuhles meist deutlich kleiner ist als das Volumen der das Objekt begrenzenden Box (siehe Abschnitt 3.4.5.7). Falls das Referenzobjekt nicht gut durch einen Quader approximiert werden kann, muss berücksichtigt werden, dass, auch wenn sich das zu lokalisierende Objekt innerhalb dieses Quaders befindet, eine externe projektive Relation anwendbar sein kann.

4.3.1 Objektidealisierungen

In dieser Arbeit werden Zentrum und ein nicht notwendigerweise achsenparalleler Quader als Approximation für die Objektgeometrien verwendet. Dieser Quader wird durch einen achsenparallelen Quader und eine Transformation spezifiziert, mit der der entsprechende Quader verschoben, rotiert oder gestreckt werden kann. Dabei werden Berechnungsverfahren vorgestellt, um Zentrum und einen achsenparallelen Quader automatisch aus den Geometrien zu errechnen. Die automatische Berechnung von beliebig orientierten begrenzenden Quadern scheint aufgrund der oft hohen Komplexität von 3D-Modellen sowie der meist achsenparallelen Ausrichtung von Räumen innerhalb Gebäuden als auch darin befindlicher für eine Lokalisationsanfrage relevanter Einrichtungsgegenstände nicht gerechtfertigt. Falls wirklich einmal Objekte nicht achsenparallel ausgerichtet sein sollten, muss der entsprechende schiefe Quader manuell spezifiziert werden.

Bei Objekten, die Geometrien gemeinsam verwenden werden, ist dies sehr einfach möglich, da in diesem Fall das gemeinsam benutzte 3D-Modell achsenparallel und nullpunktzentriert modelliert sein muss und die Positionierung und Orientierung des dieses

Modell benutzenden Objektes allein durch eine anzugebende Transformation bestimmt wird. Man ermittelt nun einfach den achsenparallelen Quader vom (nicht-transformierten) 3D-Modell und ordnet diesem Quader die entsprechende Transformation des Objektes zu. Dies hat den zusätzlichen Vorteil, dass die Berechnung des Quaders nur ein einziges Mal erfolgen muss, auch wenn das zugehörige 3D-Modell von sehr vielen Objekten verwendet wird.

4.3.2 Zentrum vs Schwerpunktapproximation

In vielen wissenschaftlichen Arbeiten (z.B. [Gap97]) werden Objekte anstatt durch das Zentrum durch den Schwerpunkt idealisiert. Dies führt möglicherweise zu unterschiedlichen Ergebnissen, falls das Objekt nicht gut durch einen Quader angenähert werden kann. Intuitiv ist der Schwerpunkt besser zur Approximation geeignet als das Zentrum, allerdings ist er auch viel schwieriger aus den Geometrien zu berechnen. Wenn man dies nun bei jedem Programmstart automatisch durchführt, würde eine Schwerpunktapproximation, wie im folgenden aufgeführt, zu einer beträchtlichen Verzögerung führen. Darum wird in dieser Arbeit auch die Schwerpunktapproximation nicht verwendet.

Um die obige Aussage zu untermauern, folgt eine Aufstellung für Berechnungsverfahren für Zentrums- und Schwerpunktberechnung.

4.3.2.1 Berechnung des Zentrums aus einem 3D-Modell

Hierzu ermittelt man zuerst die Extremwerte der Koordinaten von denen in der Geometrie auftretenden Eckpunkte: $x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}$. Nun ergibt sich das Zentrum einfach durch:

$$Z = \begin{pmatrix} \frac{x_{max}+x_{min}}{2} \\ \frac{y_{max}+y_{min}}{2} \\ \frac{z_{max}+z_{min}}{2} \end{pmatrix}$$

Aus diesen Werten läßt sich zusätzlich noch leicht der kleinste achsenparallele umfassende Quader berechnen.

4.3.2.2 Berechnung des Schwerpunktes aus einem 3D-Modell

Der Schwerpunkt eines Körpers mit homogener Masse ist definiert als (siehe [Bar94]):

$$S = \begin{pmatrix} s_x \\ s_y \\ s_z \end{pmatrix} mit$$
$$s_x = \frac{\int_V x dV}{V}$$
$$s_y = \frac{\int_V y dV}{V}$$
$$s_z = \frac{\int_V z dV}{V}$$

Dabei bezeichnet V das Volumen des Objektes. Den Schwerpunkt für ein 3D-Modell zu berechnen ist wesentlich komplizierter und aufwendiger als die Zentrumsberechnung. Eine

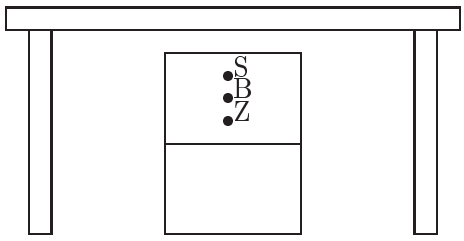

Abbildung 4.6: Zwei Kisten unter einem Tisch

mögliche Vorgehensweise besteht darin, dass man zuerst den Bereich, den das Polygon im Raum einnimmt durch Voxel (3D-Pixel) überdeckt. Man addiert nun die Mittelpunkte aller Voxel, die innerhalb des Polygons liegen aufeinander auf und gewichtet sie mit dem Anteil des entsprechenden Volumens des Voxels am Gesamtvolumen.

4.3.2.3 Vor- und Nachteile der Schwerpunktapproximation gegenüber der Zentrumsapproximation

In Abbildung 4.6 sieht man zwei Kisten, die unter einem Tisch stehen. Dabei bezeichnet S den Schwerpunkt des Tisches, Z dessen Zentrum und B das Zentrum der oberen Kiste (in diesem Fall identisch mit deren Schwerpunkt). Die Aufgabe besteht darin, die Lage der oberen Kiste bezüglich des Tisches zu bestimmen. Einmal soll dabei eine Zentrums- und einmal eine Schwerpunktapproximation des Tisches verwendet werden.

Bei Benutzung des Schwerpunktes als Tischapproximation würde die obere Kiste korrekterweise als *unterm* Tisch lokalisiert, da das Zentrum der Kiste unterhalb des Schwerpunktes des Tisches liegt. Das Zentrum des Tisches liegt dagegen unter dem Zentrum der oberen Kiste, d.h. bei der Zentrumsapproximation würde die Kiste fälschlicherweise als *über* dem Tisch liegend klassifiziert werden.

Allerdings gibt es auch Fälle, wo die Zentrumsapproximation bessere Resultate liefert. Man betrachte beispielsweise den Fall, das ein Punkt P sich direkt vor dem Gegenstand in Abbildung 4.7 befindet. Bei einer Mittelpunktapproximation würde P, sofern man sich auf horizontale Relationen beschränkt, korrekterweise als vor dem Objekt, bei einer Schwerpunktapproximation dagegen als hinter dem Objekt befindlich betrachtet werden.

Es gibt also sowohl Fälle, in denen eine Mittelpunktapproximation zum besseren Ergebnis als eine Schwerpunktapproximation führt als auch Fälle, wo die Schwerpunktapproximation vorzuziehen ist. Welche der beiden nun im allgemeinen bessere Resultate liefert, kann in dieser Arbeit nicht engültig geklärt werden.

4.3.3 Berechnung von Distanzrelationen

Das hier vorgestellte Verfahren beruht bis auf zwei Unterschiede auf dem Algorithmus von Gapp (siehe Abschnitt 3.4.4). Zum einen wird die Skalierung des Raumes bei der Di-

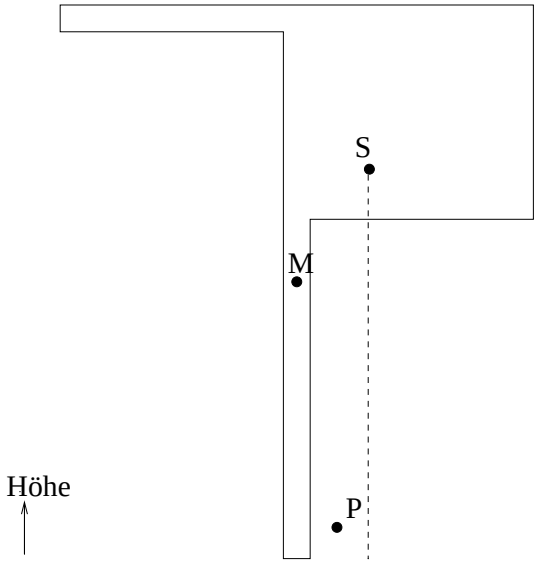

Abbildung 4.7: bessere Approximation durch den Mittelpunkt

stanzberechnung nicht abgeschwächt sondern voll berücksichtigt [3] zum anderen wird die Höhe des Referenzobjektes in besonderem Maße berücksichtigt. Die vertikale Ausdehnung eines Objektes spielt eine besondere Bedeutung, da hohe Objekte oft schon von weiter Ferne sichtbar sind. Man wird diese Objekte bevorzugt als Bezugsobjekte benutzen und auch das zu lokalisierende Objekt wohl eher als in dessen Nähe bezeichnen als bei einem niedrigeren Objekt von gleicher horizontaler Ausdehnung.

Beim folgenden Berechnungsalgorithmus werden folgende Bezeichnungen verwendet:

- RO bezeichnet das Referenzobjekt.
- LO bezeichnet das zu lokalisierende Objekt.
- $Z(obj)$ bezeichnet das Zentrum von obj.
- $ext_i(obj)$ bezeichnet die Ausdehnung eines Objektes in Dimension i mit $i \in \{1, 2, 3\}$.
- $ad(RO, LO, Rel)$ bezeichnet den Anwendungsgrad der Distanzrelation Rel. Er kann Werte von null bis eins annehmen.

Die Anwendbarkeitsbereiche für Distanzrelationen berechnen sich dann wie folgt:

- Man berechnet zwischen beiden Objekten den Distanzvektor $\vec{dist}$ mit

[3] siehe Beispiel am Ende von Abschnitt 3.4.4.1, warum dies notwendig erscheint

$$\begin{aligned} \vec{dist}_i &= \max\{0, |Z(LO) - Z(RO)| - ext_i(RO)/2\} \\ &\bullet \; sign(Z(LO) - Z(RO)) \end{aligned}$$

Dieser errechnet sich aus der Distanz beider Mittelpunkte, abzüglich der Hälfte der Ausdehnung des Referenzobjektes in den entsprechenden Koordinaten. Dies ist nötig, da die Distanz zweier Objekte in dieser Arbeit nicht als der Abstand deren Schwerpunkte sondern als Abstand der beiden nächsten Punkte der jeweiligen Objekte definiert ist (siehe Abschnitt 3.4.4). Normalerweise besitzt das zu lokalisierende Objekt eine wesentlich geringere Größe als das Referenzobjekt, so dass dessen Ausdehnung bei der Distanzberechnung hier nicht berücksichtigt wird.

- Nun wird der Differenzvektor mit dem Maximum der Höhe des Referenzobjektes und dessen Ausdehnung in der entsprechende Dimension skaliert:

$$\vec{dist}_{scaled2,i} = \vec{dist}_i / \max\{ext_y(RO), ext_i(RO)\}$$

- Man berechnet daraus die skalierte Distanz:

$$dist_{scaled2} = ||\vec{dist}_{scaled2}||$$

- entsprechende Intervalle für $i_{directlyAt} = (0, a),\ i_{at} = (a, b),\ i_{near} = (b, c)$ und $i_{far} = (d, \infty)$ werden definiert

- Nun ist Relation $Rel \in \{directlyAt, at, near, far\}$ anwendbar $:\Leftrightarrow distance_{scaled2} \in i_{Rel}$
  In dieser Arbeit werden dabei die folgende Werte für die Intervalle benutzt: a=1, b=2, c=5, d=10.

4.3.4 Berechnung von projektiven räumlichen Relationen

4.3.4.1 Berechnung des Anwendbarkeitsgrades

Wie bei der Berechnung von Distanzrelationen basiert auch der Algorithmus zur Bestimmung des Anwendbarkeitsgrades von projektiven Relationen auf dem Verfahren von Gapp mit der Ausnahme, dass auf eine Skalierung des Raumes mit der Ausdehnung des Referenzobjektes verzichtet wird (siehe Abschnitt 3.4.5.5).

Anschaulich kann man sich dieses Verfahren so verdeutlichen, dass das Lot von dem zu lokalisierenden Objekt LO auf die am nächsten liegende Seite des das Referenzobjekt umgrenzenden Quaders gefällt wird (siehe LO_1 in Abbildung 4.8). Falls dies nicht möglich sein sollte, wird eine Verbindungslinie der LO am nächsten liegenden Ecke bis zu LO gezogen (siehe LO_2). Der Winkel α zwischen der Verbindungsstrecke und des der Relation zugeordneten Vektors ergibt den für die Anwendbarkeit relevanten Abweichungswinkel (siehe naives Verfahren in Abschnitt 3.4.5.2). Dies entspricht dem folgenden Algorithmus:

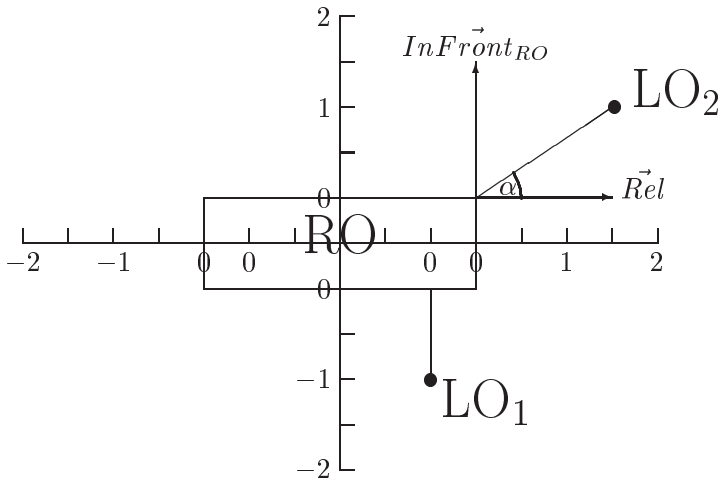

Abbildung 4.8: Berechnung von Richtungsrelationen

1. Es wird der Distanzvektor $\vec{dist}$ berechnet, der die Entfernung von Referenzobjekt und zu lokalisierendem Objekt als Vektor darstellt (siehe Distanzberechnung in Abschnitt 4.3.3).

$$\begin{aligned} \vec{dist}_i &= \max\{0, |Z(LO) - Z(RO)| - ext_i(RO)/2\} \\ &\bullet \; sign(Z(LO) - Z(RO)) \end{aligned}$$

2. Der die Relation definierende Vektor $\vec{Rel}$ wird bestimmt, für diesen gilt:

$$\begin{aligned} &Rel = above \Rightarrow \vec{Rel} = (0, 1, 0) \\ &Rel = below \Rightarrow \vec{Rel} = (0, -1, 0) \\ &Rel = right \Rightarrow \vec{Rel} = Rot_y(-\pi/2) \bullet \vec{inFront}_{RO} \\ &Rel = left \Rightarrow \vec{Rel} = Rot_y(\pi/2) \bullet \vec{inFront}_{RO} \\ &Rel = inFront \Rightarrow \vec{Rel} = \vec{inFront}_{RO} \\ &Rel = behind \Rightarrow \vec{Rel} = Rot_y(\pi) \bullet \vec{inFront}_{RO} \end{aligned}$$

3. Der Winkel zwischen dem die Relation definierenden Vektor und dem Differenzvektor wird ermittelt.

$$\alpha = \angle(\vec{dist}, \vec{Rel})$$

4. Der Anwendbarkeitsgrad wird berechnet durch

$$ad = \frac{\max\{0, \pi/2 - \alpha\}}{\pi/2}$$

(hier wird das von Gapp herausgefundene Ergebnis verwendet, dass die Anwendbarkeitsfunktion in etwa linear fällt (siehe Abschnitt 3.4.5.5)).

4.3.4.2 Sonderfälle

Falls der das Referenzobjekt begrenzende Quader nicht achsenparallel ist, kann $\vec{dist}$ nicht mehr wie oben beschrieben berechnet werden. Ein *schiefes* Quader ist in diesem System durch einen achsenparallelen Quader und eine Transformation definiert. Das obige Problem kann nun dadurch gelöst werden, dass der Mittelpunkt des zu lokalisierenden Objektes mit der inversen Transformationsmatrix des Quaders multipliziert wird. Anschließend kann man so weiterrechnen als wäre der Quader achsenparallel.

Die vorliegende Berechnung bezieht sich nur auf intrinsisch orientierte Referenzobjekte. Bei einer deiktischen Orientierung muss nach dem Spiegelprinzip verfahren werden, d.h. als Objektorientierung wird die negierte Betrachterorientierung verwendet, außerdem müssen rechts und links vertauscht werden. Konkret bedeutet das, dass in Schritt zwei $in\vec{Front}_{RO}$ durch $-in\vec{Front}_{Viewer}$ ersetzt werden muss. Zusätzlich müssen bei der Berechnung von $\vec{right}$ und $\vec{left}$ die entsprechenden Drehwinkel negiert werden.

Zusammengesetzte Relationen werden nach dem von Gapp vorgeschlagenen Berechnungsverfahren berechnet, d.h.

$$ad(RO, LO, Rel_1 \wedge .... \wedge Rel_n) = \min\{1, n \bullet min\{ad(Rel_1), ..., ad(Rel_n)\}\}$$

Weiterhin kann der Fall eintreten, dass der Mittelpunkt des zu lokalisierenden Objektes innerhalb des das Referenzobjekt begrenzenden Quaders liegt (siehe Abschnitt 3.4.5.7). In diesem Fall ist der berechnete Differenzvektor $\vec{dist}$ nach Ausführung von Schritt 1 null und damit kann Schritt 3 der Berechnung nicht durchgeführt werden. Wie in Abschnitt 3.4.5.7 aufgeführt, kann es durchaus möglich sein, dass trotzdem eine projektive Relation anwendbar sein soll. So kann sich ein Objekt *unter* einem Tisch befinden, obwohl es innerhalb des den Tisch umfassenden kleinsten Quaders lokalisiert ist. Eine mögliche Vorgehensweise besteht darin, den Tisch in mehrere Komponenten (Tischplatte, Tischbeine) zu zerlegen. Ein Objekt befindet sich dann *unter* dem Tisch, wenn es sich *unter* der Tischplatte befindet. Alternativ kann in diesem Fall auch das naive Verfahren (siehe Abschnitt 3.4.5.2) zum Einsatz kommen, bei dem das Referenzobjekt durch seinen Mittelpunkt bzw. Schwerpunkt approximiert wird. Eine dritte Möglichkeit besteht darin, das zu lokalisierende Objekt aus dem begrenzenden Quader heraus zu verschieben, so dass wieder eine externe projektive Relation anwendbar ist. Welches der drei Verfahren nun angewendet werden soll, hängt auch vom konkreten Anwendungsfall ab. So kann durch eine Zerlegung des Objektes das genaueste Resultat erzielt werden. Allerdings ist der dadurch erforderliche Mehraufwand nicht in jedem Fall gerechtfertigt. Daher soll die Entscheidung, welches Verfahren nun in diesem Fall konkret zu verwenden ist, offen gelassen werden.

4.3.4.3 Selektion der bestanwendbaren Relation

Statt den Anwendungsgrad für eine bestimmte externe projektive Relation zu bestimmen, ist man manchmal mehr daran interessiert zu wissen, welche externe projetive Relationen am ehesten anwendbar ist (siehe Abschnitt 3.4.3.1).

Bei der Berechnung eines Anwendbarkeitsgrades für eine bestimmte Relation war die Vorgehensweise so, dass zuerst aus der Position und Orientierung von Referenzobjekt und

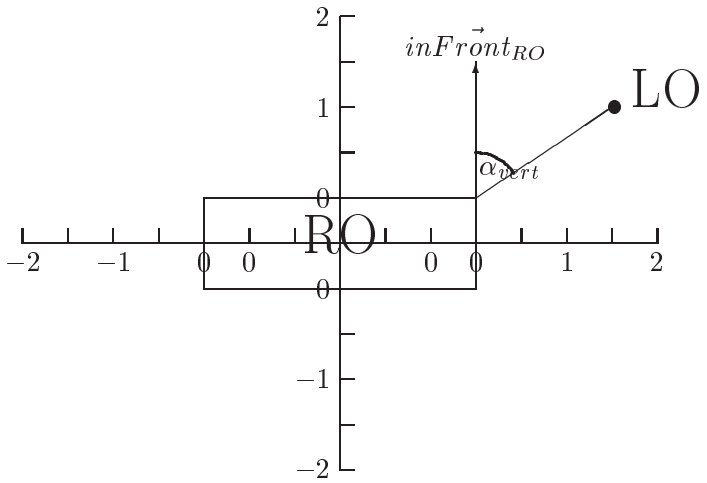

Abbildung 4.9: Bestimmung des Abweichungswinkels

zu lokalisierendem Objekt ein Differenzvektor $\vec{dist}$ ermittelt wurde. Der Abweichungswinkel zwischen diesem und dem die entsprechende Richtungsrelation definierenden Vektor bestimmte dann den Anwendungsgrad.

Bei der Selektion der besten anzuwendenden Relation wird ebenfalls zuerst dieser Richtungsvektor ($\vec{dist}$) berechnet. Anschließend ermittelt man aber nicht nur einen Winkel, sondern zwei. Dazu projiziert man $\vec{dist}$ und den Orientierungsvektor $\vec{inFront}$ des Referenzobjektes auf die x-y-Ebene, indem man die y-Koordinaten beider Vektoren eliminiert. Der erste der beiden zu berechnenden Winkel ergibt sich dann einfach aus dem Winkel zwischen den beiden projizierten Vektoren mit

$$\alpha_{horiz} = \angle\left(\begin{pmatrix} \vec{Front}_{RO,z} \\ \vec{Front}_{RO,x} \end{pmatrix}, \begin{pmatrix} \vec{dist}_z \\ \vec{dist}_x \end{pmatrix}\right)$$

Der zweite Winkel entspricht dem Steigungswinkel von $\vec{dist}$ mit

$$\alpha_{vert} = \arctan(\vec{dist}_y, \sqrt{\vec{dist}_x^{\,2} + \vec{dist}_z^{\,2}})$$

Für beide Winkel definiert man mehrere Anwendbarkeitsintervalle. Dies sind für α_{horiz} die Regionen *inFront*, *right*, *left* sowie *behind*. Dabei überlappen sich die Bereiche für die horizontale Winkeleinteilung, so dass ein entsprechender Winkel α_{horiz} beispielsweise als *rechts vor* klassifiziert würde, falls er sowohl im dem Intervall für *vorne* als in dem für *rechts* liegt.

Für den Steigungswinkel α_{vert} definiert man Intervalle für *above*, *below* sowie einen Zwischenbereich. Genauso wie im horizontalen Fall überlappen sich auch die vertikalen Intervalle, und zwar überschneidet sich der Zwischenbereich jeweils mit den beiden übrigen Intervallen. Diese Tatsache wird allerdings etwas anders verwendet als beim horizontalen

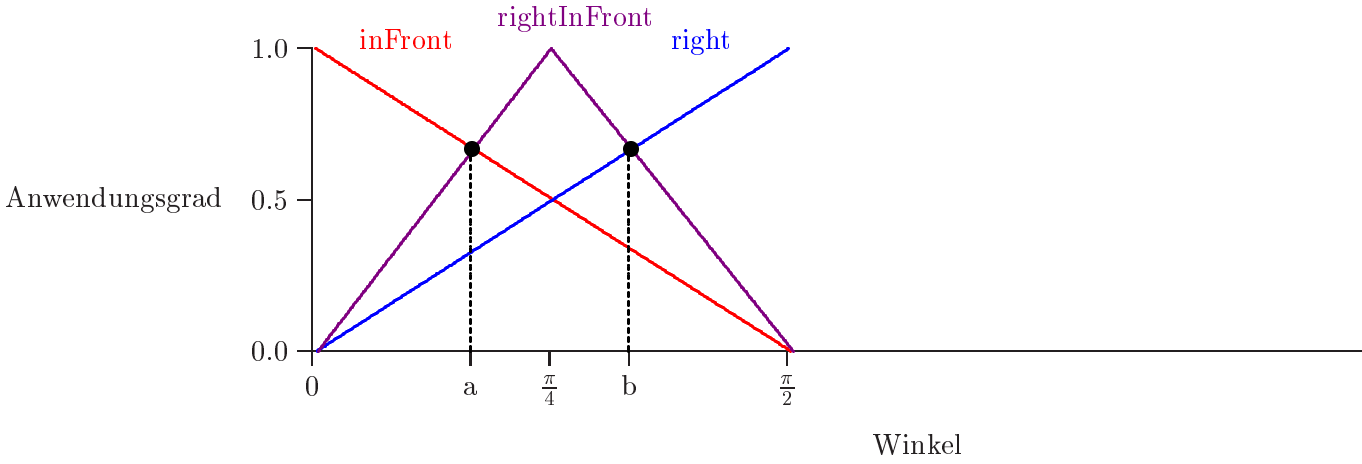

Abbildung 4.10: Anwendungsbereiche

Fall. Horizontale Relationen sind nämlich nur anwendbar, solange der Steigungswinkel noch im Zwischenbereich liegt, d.h. signifikant von $\pm 90°$ abweicht (ansonsten wäre LO direkt über bzw. unter RO).

Nun müssen allerdings noch die verschiedenen Anwendungsbereiche voneinander abgegrenzt werden. Dies sollte möglichst konsistent mit der Anwendbarkeitsgradberechnung sein, d.h. die selektierte Relation sollte also auch einen höheren Anwendbarkeitsgrad besitzen als jede andere.

Wie die einzelnen Intervalle zu wählen sind, wird im folgenden am Beispiel der Relationen *vor* und *rechts* erläutert. α bezeichnet dabei den Abweichungswinkel von demjenigen Vektor, der die Front des Referenzobjektes angibt.

$$
\begin{aligned}
ad(RO, LO, inFront) &= \frac{\max\{0, \frac{\pi}{2} - \alpha\}}{\frac{\pi}{2}} = \max\{1 - \frac{\alpha}{\pi/2}, 0\} \\
ad(RO, LO, right) &= \frac{\max\{0, \frac{\pi}{2} - |\alpha - \frac{\pi}{2}\}}{\frac{\pi}{2}} \\
ad(RO, LO, rightInFront) &= 2\min\{ad(RO, LO, inFront), \\
&\quad ad(RO, LO, right)\} \\
&= \begin{cases} \frac{2\alpha}{\pi/2}, 0 \leq \alpha \leq \pi/4 \\ 2(1 - \frac{\alpha}{\pi/2}), \pi/4 < \alpha \leq \pi/2 \\ 0, \text{sonst} \end{cases}
\end{aligned}
$$

Den Verlauf der drei Anwendbarkeitsfunktionen verdeutlicht Abbildung 4.10. Die fallende Kurve entspricht der Anwendbarkeitsfunktion von *inFront*, die steigende der von *right*. Die dritte Funktion *rightInFront* erkennt man in der Skizze an ihrem lokalen Maximum bei $\frac{\pi}{4}$. Von 0 bis a besitzt die Relation *inFront* den höchsten Anwendungsgrad, von a bis b *rightInFront* und danach *right*. Um die drei Anwendungsbereiche abzugrenzen, müssen also nur noch a und b bestimmt werden, welche sich folgendermaßen berechnen

lassen:

$$\frac{2a}{\frac{\pi}{2}} = 1 - \frac{a}{\frac{\pi}{2}} \Rightarrow a = \frac{\pi}{6}$$
$$2(1 - \frac{a}{\frac{\pi}{2}}) = \frac{a}{\frac{\pi}{2}} \Rightarrow b = \frac{\pi}{3}$$

Daraus ergeben sich die Anwendbarkeitsbereiche mit

$$in\ front = [-\pi/3, \pi/3]$$
$$right = [\pi/6, 5\pi/6]$$
$$...$$

4.3.4.4 Berechnung von internen projektiven Relationen

Interne projektive Relationen bezeichnen, an welcher Stelle innerhalb eines Objektes sich ein anderes Objekt befindet, z.B. *Das Regal befindet sich im Wohnzimmer hinten rechts.* Sie sind in erster Linie bei Wegbeschreibungen *innerhalb* von Gebäuden relevant. Vorteilhaft ist in diesem Zusammenhang, dass bei ihrer Verwendung nicht erst aufwendig ein Referenzobjekt bestimmt werden muss, da es sich anbietet, einfach das kleinste Objekt, in dem sich das zu lokalisierende Objekt befindet, als Bezugsobjekt zu verwenden.

Die hier zum Einsatz kommende Berechnung ist ganz ähnlich zu der bei den externen projektiven Relationen (siehe Abschnitt 4.3.4), weshalb sie nicht mehr im Detail beschrieben wird. Grundlage für den Algorithmus ist das in Abschnitt 3.4.5.2 beschriebene *naive* Verfahren, da dies auch problemlos bei Objekten im inneren des Referenzobjektes angewendet werden kann. Allerdings wird in diesem Fall der Differenzvektor zwischen zu lokalisierendem und Bezugsobjekt mit der Ausdehnung des letzteren skaliert. So bezeichnet die Ortsangabe *hinten links* immer einen Bereich nahe der linken hinteren Ecke des Bezugsobjektes und zwar unabhängig von dessen Ausdehnung. Dies kann demnach auch bedeuten, dass man von der Raummitte zehn Schritte nach links und nur einen nach vorne gehen muss.

Für jede externe projektive Relation gibt es eine korrespondierende interne, *hinten* für *hinter*, *vorne* für *vor*, *oben* für *über*, *unten* für *unter* und *rechts/links* für *rechts/links von.* Zusätzlich existiert eine weitere Relation, die keine Entsprechung bei den externen Relationen besitzt und zwar *in der Mitte von.* Diese bezeichnet einen Raum, der durch eine Ellipse mit gleicher skalierter Distanz vom Raummittelpunkt begrenzt ist (ich benutzte hier die Hälfte der Raumausdehnung).

Soll die deiktischen Orientierung verwendet werden, ist zu beachten, dass der entsprechende Orientierungsvektor an die Objektausrichtung angepaßt werden muss. Im dem Fall, dass das Referenzobjekt ein Raum ist, muss er also parallel zu einer Wand verlaufen.

4.3.5 Berechnung von *auf*

In diesem Abschnitt wird ein Verfahren vorgestellt, mit dessen Hilfe bestimmt werden kann, ob ein Objekt sich *auf* einem anderen befindet. Die Berechnung soll dabei nicht etwa einen booleschen Wert (wahr oder falsch) zurückliefern, sondern die Anwendbarkeit in einen kontinuierlichen Wertebereich von null bis eins abbilden (siehe Abschnitt 3.4.3).

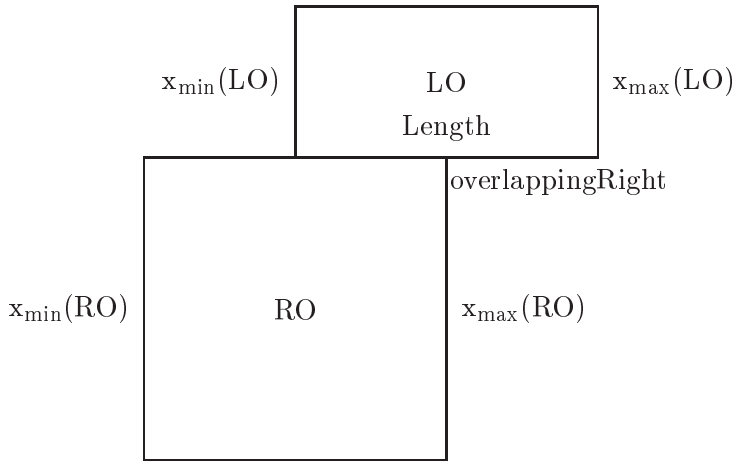

Abbildung 4.11: *LO* überlappt *RO*

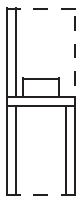

Abbildung 4.12: Eine Kiste auf einem Stuhl

Im Gegensatz zu den Berechnungsverfahren für projektive und Distanzrelationen wird diesmal die geometrische Ausdehnung des zu lokalisierenden Objektes bei der Berechnung mitberücksichtigt.
Zunächst wird überprüft, ob sich der unterste Punkt des zu lokalisierenden Objektes auf der Höhe des obersten Punktes des Referenzobjektes befindet. Um kleine Modellierungsfehler auszuschließen, wird eine gewisse Abweichung a zugelassen (hier a=10cm). Stimmen beide Ordinaten nicht überein, beträgt der Anwendungsgrad null. Ansonsten ist für dessen Berechnung entscheidend, inwiefern das zu lokalisierende Objekt vom Referenzobjekt gestützt wird (siehe Abschnitt 3.4.6). Überlappt das zu lokalisierende Objekt das Bezugsobjekt auf allen Seiten gleich (die Überlappung kann auch null sein), befindet es sich im Gleichgewicht und somit vollständig *auf* dem Referenzobjekt. Siehe auch Abbildung 4.11 zur Vedeutlichung des Verfahrens.

$$
\begin{aligned}
&overlappingRight = \max\{0, x_{max}(LO) - x_{max}(RO)\}\\
&overlappingLeft = \max\{0, x_{min}(RO) - x_{min}(LO)\}\\
&overlappingFront = \max\{0, z_{max}(LO) - z_{max}(RO)\}\\
&overlappingBack = \max\{0, z_{min}(RO) - z_{min}(LO)\}\\
&length_{LO} = x_{max}(LO) - x_{min}(LO)\\
&depth_{LO} = z_{max}(LO) - z_{min}(LO)
\end{aligned}
$$

$$
\begin{aligned}
ad(RO, LO, on) \quad = \quad & (1 - \frac{|overlappingRight - overlappingLeft|}{length_{LO}}) \bullet \\
& (1 - \frac{|overlappingFront - overlappingBack|}{depth_{LO}})
\end{aligned}
$$

Ein Problem, das sich bei dieser Berechnung ergeben kann, tritt dann auf, wenn sich wie in Abbildung 4.12 das Referenzobjekt schlecht durch einen Quader approximieren lässt. Dort befindet sich eine Kiste *auf* einem Stuhl, obwohl der unterste Punkt der Kiste deutlich unter dem obersten des Stuhles liegt. Nach dem hier besprochenen Verfahren wäre also die Relation auf in dieser Situation nicht anwendbar. Dieses Problem kann man dadurch lösen, in dem man den Stuhl als zusammengesetztes Objekt modelliert, der aus Lehne, Sitz und Stuhlbeinen besteht.
Die Funktion auf kann man dann folgendermaßen umdefinieren:

$$ad(Stuhl, LO, on) = max\{ad(Sitz, LO, on), ad(Lehne, LO, on)\}$$

Weiterhin können Fälle auftreten, in denen eine transitive Interpretation der Relation gewünscht ist (siehe Abschnitt 3.4.6). In diesem Fall betrachtet man ein Objekt o_1 als *auf* einem anderen Objekt o_2 befindlich, falls es sich direkt auf o_1 befindet oder es sich direkt auf einem anderen Objekt o_3 befindet, welches dann wiederum auf o_2 liegt. Dabei kommen als zwischen o_1 und o_2 liegend alle Objekte in Frage, die sich im selben Objekt (meistens Raum) wie o_2 befinden.

4.4 Domänenvisualisierung und Interaktion

Dieser Abschnitt schildert, wie semantische Informationen geeignet für eine grafische Darstellung verwendet werden können. Das beinhaltet auch eine Animation, in der ein Präsentationsagent den vom Benutzer einzuschlagenden Weg zum Zielobjekt abläuft. Die Bewegung des Agenten selber, ist dabei allerdings Thema einer anderen Diplomarbeit (siehe [Bre01]).

Bei der grafische Darstellung werden dabei die folgenden Aspekte berücksichtigt:

- Für Objekte können achsenparallele begrenzende Quader automatisch berechnet werden (schiefe Quader müssen manuell spezifiziert werden).
- Türen öffnen und schließen sich automatisch, wenn der Agent in deren Nähe kommt und zu der entsprechenden Tür orientiert ist.
- Abhängigkeiten zwischen Objekten werden berücksichtig. So impliziert das Laden eines Raumes auch dass die zugehörigen Wandgeometrien bereitgestellt werden müssen.
- Geometrien werden abhängig von der Agentenposition automatisch geladen und wieder entladen.
- Zu jeder Geometrie kann deren semantische Repräsentation ermittelt werden und umgekehrt.
- Objektinformationen können als Textfeld in der virtuellen Umgebung eingeblendet werden.

Grundlage für viele der folgenden Verfahren ist die Berechnung der begrenzenden achsenparallelen Quader der in der virtuellen Umgebung vorkommenden Räume, welche im folgenden beschrieben wird.

4.4.1 Berechnung der raumbegrenzenden achsenparallelen Quader

Der folgende Algorithmus benutzt zur Berechnung des Quaders grundsätzlich nur vier Wände. Sollte ein Raum mehr besitzen, müssen vier herausgesucht werden, die den Raum in alle vier Richtungen begrenzen. Sie müssen diesen allerdings nicht vollständig abschließen, sondern es darf durchaus eine Lücke vorhanden sein.

Die Wände werden dabei durch Strecken angenähert, die deren horizontalen Verlauf beschreiben. Dies entspricht quasi einer Grundrißzeichnung aus der Vogelperspektive, bei der die Wände durch Linien approximiert werden. Zwischen diesen Strecken werden dann Schnittpunkte bestimmt. Diese vier Schnittpunkte sowie die vertikalen Extremwerte der Wände bestimmen zusammen den gesuchten Quader. Dies kann folgendermaßen formal beschrieben werden:

- Für jede Wand werden die begrenzenden achsenparallelen Quader ermittelt. Diese können leicht aus den Extremwerten der entsprechenden Wandgeometrien bestimmt

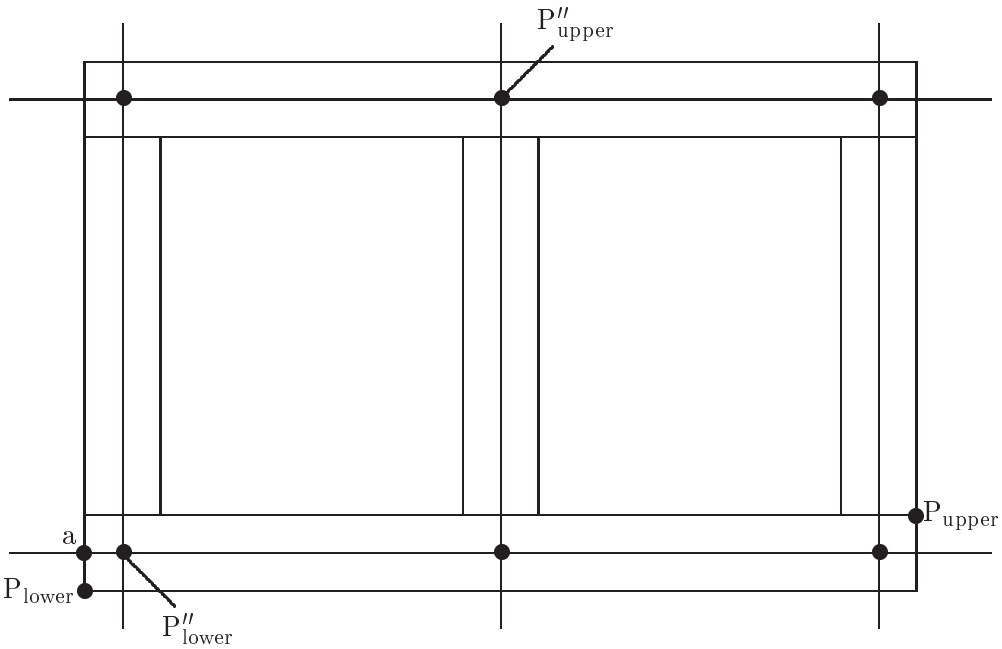

Abbildung 4.13: Raumberechnung

werden.

$$P_{lower} = \begin{pmatrix} x_{min} \\ y_{min} \\ z_{min} \end{pmatrix}, wobei$$
$$x_{min} = \min\{x|(x,y,z) \in Vertices(Wall)\},$$
$$y_{min} = \min\{y|(x,y,z) \in Vertices(Wall)\},$$
$$z_{min} = \min\{z|(x,y,z) \in Vertices(Wall)\},$$
analog für P_{upper},

Die Punkte P_{lower} und P_{upper} definieren das achsenparallele Begrenzungsquader.

- Die erhaltenen Quader werden dazu benutzt, die Wände durch Strecken zu approximieren, die deren horizontale Verläufe beschreiben. Falls die Breite des Quaders dessen Tiefe übersteigt, benutzt man eine Strecke, die parallel zur x-Achse verläuft, andernfalls eine Strecke parallel zur z-Achse. Die vertikale Ausdehnung der Räume wird dabei ignoriert.

  $route(Wall) = \overleftarrow{a} + \lambda \bullet \overleftarrow{b}, mit$

$$\vec{a} = \begin{pmatrix} P_{lower,x} \\ \frac{P_{lower,z}+P_{upper,z}}{2} \end{pmatrix} und\ \vec{b} = \begin{pmatrix} P_{upper,x} - P_{lower,x} \\ 0 \end{pmatrix}$$

  falls $P_{upper,x} - P_{lower,x} > P_{upper,z} - P_{lower,z}$,
  ansonsten gilt:

$$\vec{a} = \begin{pmatrix} \frac{P_{lower,x}+P_{upper,x}}{2} \\ P_{lower,z} \end{pmatrix}, \vec{b} = \begin{pmatrix} 0 \\ P_{upper,z} - P_{lower,z} \end{pmatrix}$$

wobei $0 \leq \lambda \leq 1$

- Man verlängert die Strecken etwas, damit dieser Algorithmus auch bei Räumen zum Erfolg führt, die nicht vollständig geschlossen sind. d.h. für λ muß nun gelten:

$$-\delta \leq \lambda \leq 1 + \delta \text{ für ein } \delta > 0.$$

- Man bestimmt die Schnittpunkte zwischen den vier zu einem Raum gehörenden Strecken (siehe Abbildung 4.13).
  Es sollte genau vier Schnittpunkte geben, d.h. $|Crosspoints| = 4$ mit

$$\begin{aligned} &Crosspoints = \{P : \exists \overleftarrow{a_1}, \overleftarrow{b_1}, \lambda_1, \vec{a_2}, \vec{b_2}, \lambda_2 \text{ mit} \\ &\vec{a_1} + \lambda_1 \bullet \vec{b_1} = \vec{a_2} + \lambda_2 \bullet \vec{b_2} \wedge \\ &-\delta \leq \lambda_1 \leq 1 + \delta \text{ sowie } -\delta \leq \lambda_2 \leq 1 + \delta\} \end{aligned}$$

- Von diesen vier Schnittpunkten berechnet man nun das begrenzende achsenparallele Rechteck. Dies erhält man einfach durch Ermittlung der Extremwerte der vier Schnittpunkte (s.o.). Der minimale bzw. maximale Punkt dieses Rechteckes wird im folgenden mit P'_{lower} und P'_{upper} bezeichnet.

- Man ermittelt die minimale und maximale Ordinate y_{min} und y_{max} der vier Wände.

- Nun braucht man nur noch alles zusammenzusetzen. Die horizontale Begrenzung des Quaders ist durch das Rechteck gegeben, die vertikale durch y_{min} und y_{max}. Die den gesuchten Quader definierenden zwei Punkte sind demnach:

$$P''_{lower} = \begin{pmatrix} P'_{lower,x} \\ y_{min} \\ P'_{lower,z} \end{pmatrix}$$

und

$$P''_{upper} = \begin{pmatrix} P'_{upper,x} \\ y_{max} \\ P'_{upper,z} \end{pmatrix}$$

Ein Fall, wo der obige Algorithmus nur unzureichend genau einen Raum approximiert hat, trat im Flughafenszenario mehrmals auf, wenn mehrere aneinander angrenzende Räume eine gemeinsame lange schräge Wand besaßen. So wird die schräge Wand in Abbildung 4.14 von P_1 bis P_4 nur sehr ungenau durch die gestrichelte Linie approximiert. In diesem Fall muss diese Wand in mehrere Wände gesplittet werden, um wieder eine hinreichend genaue Approximation zu erhalten. Die Wand in Abbildung 4.14 würde man beispielsweise in drei Teile aufteilen (P_1 bis P_2, P_2 bis P_3 und P_3 bis P_4).

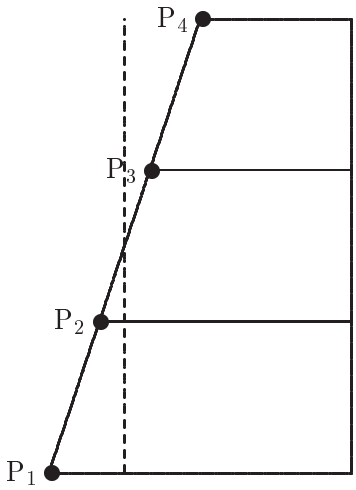

Abbildung 4.14: ungenaue Approximation bei langer schräger Wand

4.4.2 Dynamisches Laden von Geometrien

Nicht immer ist es in einem 3D-Visualisierungssystem möglich, alle Geometrien ständig vorzuhalten. Zum einen benötigen 3D-Polygone oft riesige Mengen an Hauptspeicher, zum anderen steigt die zum Rendern der Umgebung benötigte Zeit mit der Anzahl und Komplexität der Geometrien. Dabei sieht der sich in der virtuellen Welt aufhaltende Agent ja immer nur einen kleinen Ausschnitt der Umgebung. Es sollte also reichen, nur die Geometrien von diesem Teil vorrätig zu halten. Dynamisches Laden beinhaltet dabei sowohl dynamisches Laden als auch Entladen, wobei beide Aspekte im folgenden betrachtet werden.

4.4.2.1 Automatisches Laden der Geometrien

Falls sich ein Agent außerhalb von Gebäuden aufhält, ist er möglicherweise im Stande weite Teile der Umgebung einzusehen. In diesem Fall ist ein dynamisches Laden von Geometrien recht schwierig zu realisieren. Ist sein Blickfeld dagegen auf das Innere von mehreren Räumen beschränkt wie das bei Gebäudenavigation oft der Fall ist, genügt es, gerade die Geometrien dieser Räume vorrätig zu halten.

Betrachte man den Fall, das nur ein einziger Raum geladen ist und der Agent sich innerhalb desselben befindet. Dieser Raum besitzt dabei eine Tür, die zu einem Nachbarraum führt. Die Geometrien des Nachbarraumes müssen genau dann sichtbar sein, wenn diese Tür geöffnet wird (siehe Abbildung 4.15). Allgemein kann man also sagen, dass, wenn eine Tür sich öffnet, beide an die Tür grenzenden Räume geladen sein müssen.

$$open(d) \Rightarrow (\forall r(Room(r) \wedge hasDoor(r,d)) \Rightarrow load(r))$$

Dass ein Raum geladen wird, heißt wiederum, dass alle seine Komponenten (Decke, Boden, Wände, Türen) sowie seine Einrichtung geladen werden müssen.

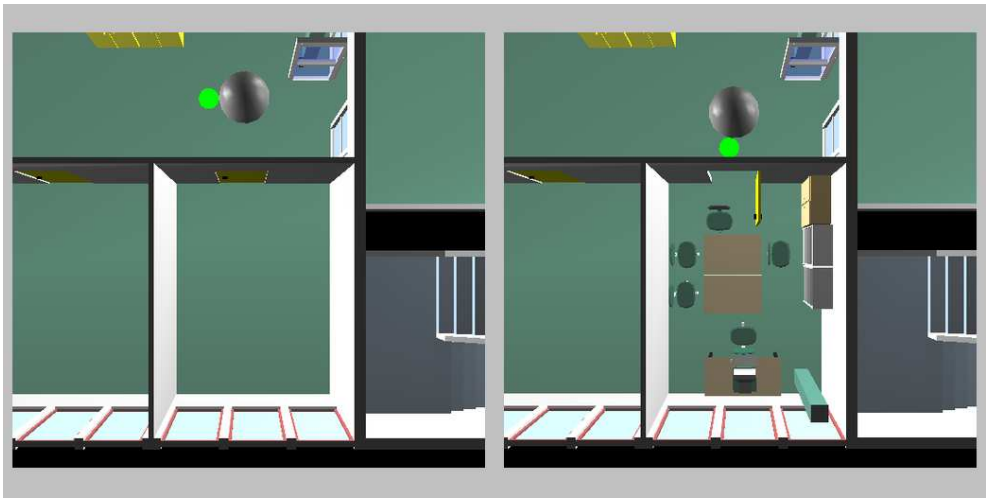

Abbildung 4.15: automatisches Laden von Geometrien

Der obige Algorithmus versagt allerdings, wenn Räume ohne eine Tür miteinander verbunden sind, man also von dem einen Raum ohne Probleme in den anderen sehen kann. Eine Lösungsmöglichkeit besteht darin, dass beim Laden eines Raumes auch alle mit diesem direkt (ohne Tür) verbundenen Räume mitgeladen werden. Allerdings ist zu beachten, dass es dabei zu beträchtlichen Verzögerungen kommen kann, wenn mit dem automatisch mitgeladenen Raum wieder andere Räume direkt verbunden sind, bei denen diese Konstellation möglicherweise wiederum vorliegt.

Ein anderes Problem tritt auf, wenn ein Raum geladen wird, bei dem noch eine andere Tür offen steht, der Agent also dann möglicherweise in einen Raum sehen könnte, dessen Geometrien momentan nicht sichtbar sind. Dieser Fall kann analog zu den direkt miteinander verbundenen Räumen behandelt werden. Im System GECL werden Türen (normalerweise) nur bei Bedarf geöffnet, d.h. der obige Fall kann dadurch gar nicht auftreten.

4.4.2.2 Automatisches Entladen von Geometrien

Ohne einen Mechanismus zum automatischen Entladen von Geometrien, wird der Vorteil des automatischen Ladens bei längerer Programmlaufzeit möglicherweise wieder egalisiert, da bei Besuch des Agenten von allen im System vorkommenden Räumen auch alle Geometrien wieder eingeladen wären. Der Algorithmus zum automatischen Entladen verläuft übrigens nicht, wie man denken könnte, spiegelbildlich zum automatischen Laden. Dies würde nähmlich heißen, dass beim Schließen einer Tür beide an die Tür angrenzenden Räume einfach wieder entladen werden würden. Wie nun, wenn der Agent sich gerade in dem an die Tür angrenzenden Raum befindet? Er würde anschließend im Nichts stehen.

Stattdessen wird ein an eine schließende Tür angrenzender Raum entladen, wenn der

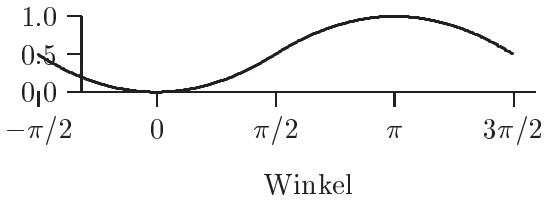

Abbildung 4.16: Kosinusfunktion

Agent sich nicht in diesem befindet und alle Türen dieses Raumes geschlossen sind.

$$
\begin{aligned}
&closingFinished(d) \wedge \\
&Room(r) \wedge \\
&hasDoor(r, d) \wedge \\
&(\forall d' \text{mit } hasDoor(r, d') : closed(d')) \wedge \\
&\neg in(agent, r) \Rightarrow unload(r)
\end{aligned}
$$

Ob ein Agent sich in einem bestimmten Raum aufhält kann mit Hilfe der Raum begrenzenden Quader ermittelt werden, die nach dem in Abschnitt 4.4.1 beschriebenen Algorithmus berechnet wurden.

4.4.2.3 Öffnen und Schließen von Türen

Den Vorgang des Türöffnen und Schließens beinhaltet zum einen die Erkennungsprozedur, dass eine bestimmte Tür geöffnet werden soll, zum anderen die Modellierung der Türrotation.

Ob eine bestimmte Tür geöffnet werden soll, kann dadurch festgestellt werden, dass ein Picking-Strahl vom Avatar abgeschickt wird. Nun ermittelt man die erste Geometrie, die dieser Strahl schneidet. Ist ihr ein semantisches Objekt vom Typ *Door* zugeordnet, muss die entsprechende Tür geöffnet werden.

Die Drehung der Tür erfolgt beim Öffnen nicht linear, sondern zuerst langsam, wird dann schneller, bis die Geschwindigkeit der Rotation am Schluss wieder abnimmt, wodurch die Massenträgheit simuliert werden soll. In dieser Arbeit wird für den Rotationsverlauf die Funktion $(-\cos x + 1)/2$ verwendet, die zwischen null und π von der beschriebenen Form ist (siehe Abbildung 4.16).

Die Berechnung der für die Türrotation benötigten Transformation wird in Abbildung 4.17 dargestellt. Um die Abbildung übersichtlicher zu machen, wurde dabei eine Rotation von 180° benutzt. In Wirklichkeit wählt man die einzelnen Rotationen sehr gering, um eine möglichst flüssige Animation zu erreichen. Die visuelle Darstellung einer Tür erfolgt in Abhängigkeit von ihrer geometrischen Modellierung und einer Transformation. Jede Tür wird dabei grafisch so modelliert, dass sie achsenparallel und nullpunktzentriert ist. Ihre Position und Ausrichtung bestimmt sich dann allein durch die Transformation.

Um die Tür D_0 in der Abbildung zu rotieren reicht es nicht, sie einfach mit der entsprechenden Rotation zu multiplizieren. Das Ergebnis einer solchen Multiplikation wäre

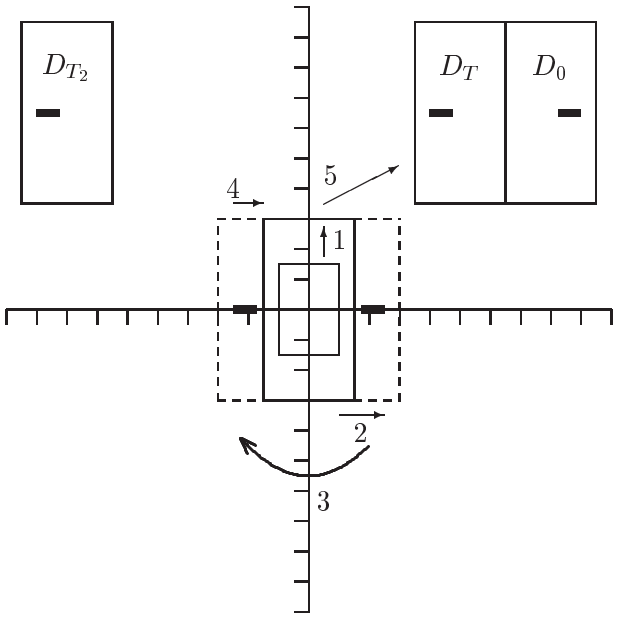

Abbildung 4.17: Türrotation

D_{t_2}. Stattdessen erfolgt die Berechnung der Rotation in den folgenden Schritten:

1. Die Skalierung der Tür wird aus der Transformation herausgezogen und zuerst auf die Türgeometrie angewendet. Dies ist notwendig, weil die Tür, wenn sie skaliert wird, noch nicht rotiert worden sein darf.

2. Die Tür wird so verschoben, dass sie mit ihrer linken (rechten) Seite genau an der y-Achse liegt.

3. Die Tür wird rotiert.

4. Die Tür wird zurück zum Nullpunkt verschoben.

5. Die Tür wird mit ihrer Orginaltransformation multipliziert. Die Skalierung muß dazu vorher aus dieser entfernt werden. Damit ist die Rotation abgeschlossen.

Insgesamt kann man die Rotation darstellen durch

$$doorRot(\alpha) = Transl_0(door) \bullet Rot_0(Door) \bullet Transl(\mp width(door)/2) \bullet Rot_y(\alpha) \bullet Transl(\pm width(door)/2) \bullet Skal_0(door)$$

Dabei bezeichnen $Rot_0(Door)$, $Transl_0(Door)$ und $Skal_0(Door)$ die anfängliche Transformation vor der durchzuführenden Drehung.

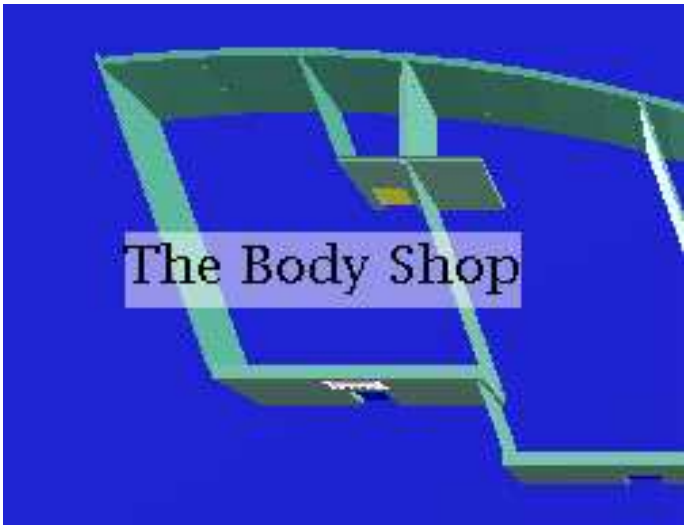

Abbildung 4.18: Einzelannotation

4.4.3 Objektannotationen

Bei Objektannotationen kann man zwischen Annotationen unterscheiden, die Informationen über ein bestimmtes Objekt ausgeben sollen und solchen, die in einem Gesamtüberblick alle in der Umgebung vorkommenden Gebiete annotieren. Bei der Einzelannotation kann die angezeigte Information recht ausführlich sein, während bei einer Überblicksdarstellung eine knappe Objektbeschreibung bevorzugt wird. Zusätzlich muss in diesem Fall sichergestellt werden, dass sich die angezeigten Textfelder nicht überlappen.

4.4.3.1 Einzelannotation

Die Konzeption der Einzelannotation erfolgt dabei nach den in Abschnitt 3.5.2 dargestellten Prinzipien. Das heißt unter anderem, dass die Einzelannotation in ihrer Größe konstant bleibt, auch wenn sich der Betrachter vom annotierten Objekt entfernt. Dies wird dadurch erreicht dass die Ausgabe der Objektannotationen direkt in das Fenster erfolgt, in das die virtuelle Umgebung gerendert wird (siehe Abbildung 4.18), d.h. der Ausgabetext ist nicht Bestandteil der 3D-Welt. [4] Dadurch entsteht aber das Problem, das die Position der Annotation nicht mehr so einfach mit der des zugehörigen Objektes verknüpft werden kann. Obwohl das Textfeld sich nicht in der 3D-Welt befindet, muss es nun explizit so positioniert werden, als ob dies der Fall wäre. Dabei sind die folgenden Parameter bekannt. (siehe Abbildung 4.19)

- der Winkel α, der das Sichtfeld des Betrachters definiert. Ein hoher Winkel entspricht im Prinzip einer Betrachtung der Umgebung mit einem Weitwinkelobjektiv, ein geringer einer Sicht mit einem Fernrohr.

[4] Dies entspricht einer Projektion eines Punktes der 3D-Welt auf die Projektionsebene

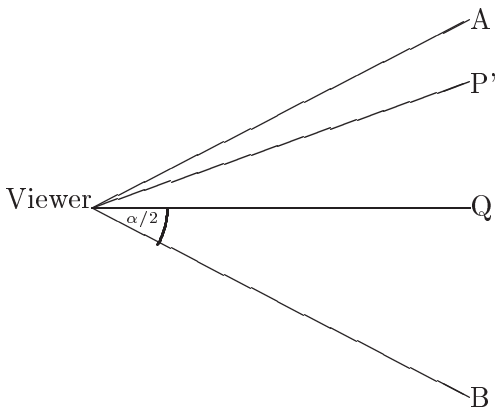

Abbildung 4.19: Positionierung der Einzelannotation

- die Höhe $height_{canvas}$ und Breite $width_{canvas}$ des Fensters (in Pixeln), auf dem die 3D-Umgebung projiziert wird. Bei Entfernung $||Q - Viewer||$ wird der ganze Bereich zwischen A und B dargestellt.
- die Transformation $transf_{viewer}$, mit der die Sicht des Betrachters festgelegt wird. [5]
- der Punkt P mit dem das Textfenster verknüpft werden soll

Gesucht sind die Koordinaten Mx, My, die den Mittelpunkt des Textes innerhalb des Fensters angeben, das die 3D-Umgebung auf dem Bildschirm darstellt.

Das Berechnungsverfahren läuft folgendermaßen ab (siehe dazu auch Abbildung 4.19):

- Zuerst wird der Punkt P in das Koordinatensystem des Betrachters transformiert. Dadurch sind die horizontalen und vertikalen Abweichungen von der Betrachterorientierung sehr leicht zu bestimmen und zwar betragen diese:

$$P - Q = \begin{pmatrix} P'_x \\ P'_y \\ 0 \end{pmatrix}$$

mit $P' = transf^{-1}_{viewer} \bullet P$

[5] Keine Transformation entspricht dem Fall, dass der Betrachter am Ursprung positioniert ist und dessen Beobachtungsrichtung parallel mit der z-Achse verläuft. In seiner Sicht befinden sich dann ausschließlich Objekte mit negativem z-Wert.

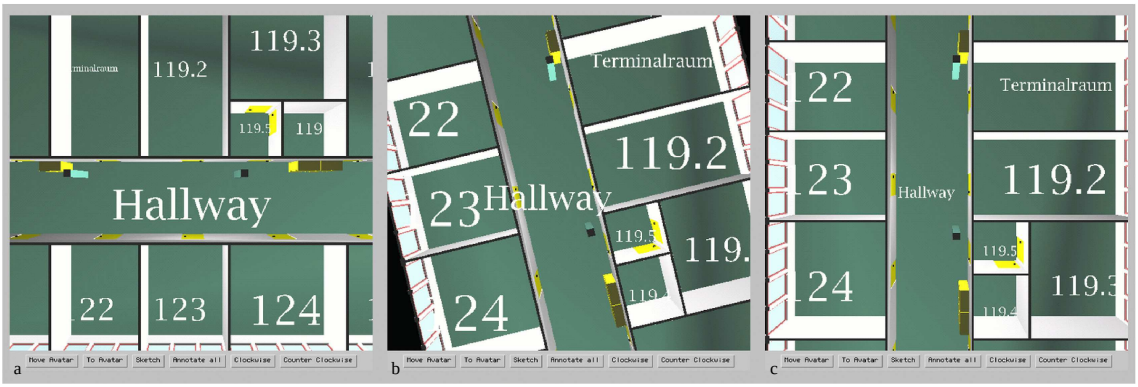

Abbildung 4.20: Übersichtsannotation

- Auf den Bildschirm wird der ganze Bereich von A bis B projiziert. Es gilt also folgendes Verhältnis:

$$\begin{aligned}
\frac{(P'-B)_x}{||A-B||_x} &= \frac{Mx}{width_{canvas}} \\
\frac{(P'-B)_y}{||A-B||_y} &= \frac{My}{height_{canvas}} \text{ mit} \\
P'-B &= (P'-Q)+(Q-B) \\
(Q-B) &= \tan(\frac{1}{2} \bullet \alpha) \bullet P'_z \\
A-B &= 2 \bullet (Q-B)
\end{aligned}$$

  Daraus lassen sich die gesuchten Werte (Mx, My) für den Mittelpunkt der Annotation bestimmen.

4.4.3.2 Übersichtsannotation

Diese Annotation ist dafür konzipiert, dem Anwender einen groben Überblick zu geben. In diesem Fall ist es durchaus sinnvoll, mehrere Textannotationen gleichzeitig einzublenden, um eine gute Übersicht über die Umgebung zu erhalten (siehe Abbildung 4.20). Allerdings sollten sich die Textfelder, um eine zufriedenstellende Lesbarkeit zu gewährleisten, nicht überlappen. Um dies zu gewährleisten wurde diese Annotation, im Gegensatz zur Einzelannotation, als Bestandteil der 3D-Welt definiert. Damit kann deren Darstellungsgröße auf die Ausdehnung des annotierten Objektes beschränkt werden. Dieses Textfeld wird also desto kleiner dargestellt, je weiter sich der Betrachter von dem entsprechenden Objekt entfernt. Weil dadurch die Lesbarkeit beeinträchtigt wird, besteht der Text nur noch aus einer recht knappen Objektbeschreibung. Zusätzlich wird er den Ausdehnungen der Räume optimal angepasst. Dadurch dass die Annotation immer zum Betrachter hin orientiert sein soll (sie darf beispielsweise nicht auf dem Kopf stehen), muss dazu eventuell die Schriftgrösse angepasst werden.

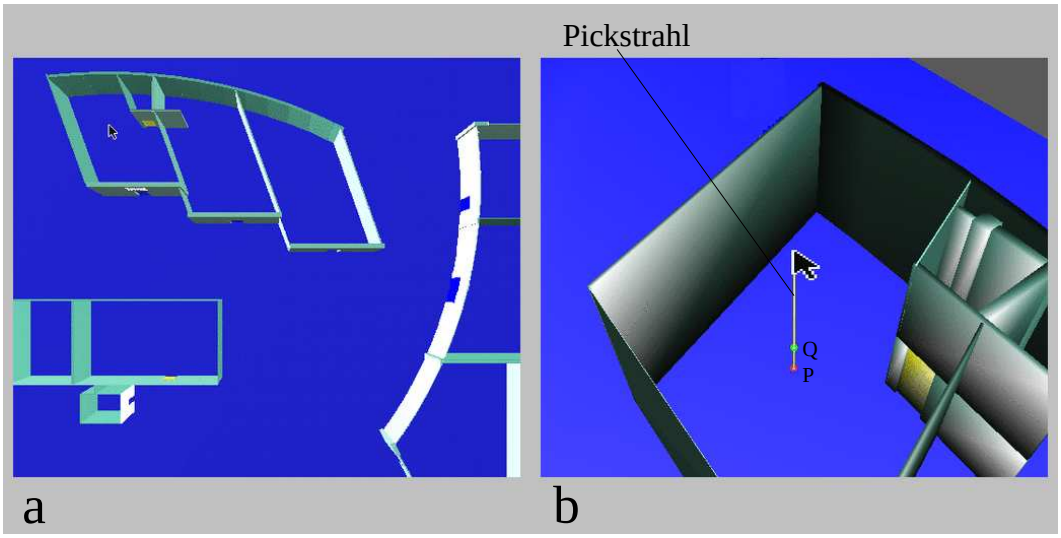

Abbildung 4.21: Anklicken eines Raumes (aus Anwender- und Schrägansicht)

Betrachte man beispielsweise eine Annotation in einem sehr langen und schmalen Flur. Für die Annotation steht die ganze Länge des Flurs zur Verfügung. Wird dieser plötzlich um 90° gedreht muss der Text so stark gestaucht werden, dass seine Länge die schmale Seite des Flurs nicht überschreitet (siehe Abbildung 4.20) (a–c)).

Die für die Textannotation zur Verfüfung stehende Fläche *width*, *depth* ist abhängig von der Ausdehnung eines Raumes *size* und der Schriftorientierung $\vec{orientation}$ (gibt an, wo oben ist) mit
$||\vec{orientation}|| = 1$. Sie ergibt sich dabei aus

$$\begin{aligned} width &= |\vec{orientation_x}| \bullet size_z + |\vec{orientation_z}| \bullet size_x \\ depth &= |\vec{orientation_x}| \bullet size_x + |\vec{orientation_z}| \bullet size_z \end{aligned}$$

Wenn z.B. die Schrift in Richtung der z-Achse orientiert ist, ergibt sich die für den Text zur Verfügung stehende Tiefe einfach aus der Tiefe des Raumes. Für die Orientierung gilt in diesem Fall:

$$\vec{orientation_x} = 0, \vec{orientation_z} = -1$$

Ist die Schrift dagegen parallel zur x-Achse orientiert muss die Textlänge in die Tiefe des Raumes eingepaßt werden ($\vec{orientation} = (-1, 0)$).

4.4.4 Eingabetechniken

Im vorigen Abschnitt wurden Methoden vorgestellt, um dem Benutzer verschiedene Informationen zu präsentieren. Neben der Ausgabe von Daten, muss das System aber auch in der Lage sein, auf Benutzereingaben zu reagieren. Ein beliebtes Instrument ist dabei die

Auswahl von Objekten durch Anklicken. Bei atomaren, nicht zusammengesetzten Objekten läßt sich das leicht realisieren, indem man einen Picking-Strahl (siehe [Bou99]) von der Mausposition in die 3D-Welt hinein generiert und das erste Objekt ermittelt, den dieser schneidet.

Komplizierter wird der Fall, wenn auch aggregierte Objekte wie Räume angeklickt werden können sollen (siehe Abbildung 4.21). Die einfache Vorgehensweise, dass man zuerst mit dem Picking-Strahl das Bodenpolygon ermittelt und den zugehörigen Raum als angeklicktes Objekt identifiziert führt nicht zum Erfolg, da ein Boden Bestandteil von sehr vielen Räumen, im Extremfall sogar von allen Räumen, sein kann. Eine mögliche Vorgehensweise wäre, über jeden Raum ein unsichtbares Objekt zu legen, das eindeutig dem darunterliegenden Raum zugeordnet wäre. Der Picking-Strahl würde nun zuerst dieses unsichtbare Objekt schneiden und der gesuchte Raum wäre damit bestimmt. Allerdings ist dieses Verfahren recht umständlich, weil erst für jeden vorkommenden Raum ein solches Objekt modelliert oder berechnet werden müsste. Daher kommt diese Verfahren auch in diesem System nicht zum Einsatz.

Der hier verwendete Algorithmus bestimmt zuerst den Schnittpunkt mit dem Bodenpolygon (siehe Punkt P in Abbildung 4.21b). Um etwaige Störobjekte wie geöffnete Türen auszuschließen, wird der Picking-Strahl solange verfolgt, bis er auf ein Polygon trifft, dass mit dem semantischen Typ *Floor* annotiert wurde. Den Schnittpunkt hebt man nun um einen Meter an, d.h. man erhöht seine y-Koordinate um 1m, damit er sich vollständig innerhalb eines Raumes befindet (Punkt Q). Allen Räumen in der Umgebung wurde durch den Algorithmus in Abschnitt 4.4.1 ein umfassender achsenparalleler Quader zugeordnet. Man ermittelt nun einfach derjenige Quader, in dem sich dieser Punkt befindet und erhält dadurch auch den entsprechenden angeklickten Raum.

Ein Problem kann bei diesem Verfahren auftreten, wenn der Betrachter nicht genau von oben, sondern etwas von der Seite auf das Modell blickt (z.B. aus der Perspektive aus Abbildung 4.21 b) und auf eine Wand klickt. Bei dem hier beschriebenen Algorithmus würde der Raum hinter der Wand annotiert werden. Möglicherweise möchte der Benutzer aber Informationen über den vor der angeklickten Wand befindlichen Raum erhalten. Welche der beiden Anzeigen normalerweise vom Benutzer intendiert ist, ist nicht eindeutig bestimmbar und müßte anhand von psychologischen Tests ermittelt werden.

4.5 Zusammenfassung

In diesem Kapitel wurden Konzepte für den Entwurf eines Wegauskunfts- oder Objektlokalisationssystems vorgestellt.

Dies beinhaltet Methoden zum Auffinden von Objekten, wie performante und flexible Suchalgorithmen, sowie die Konzeption einer Freitextsuche, die sich durch Merkmale wie Multilingualität, Berücksichtigung von Generalisierungbeziehungen, Tippfehlerkorrektur und integriertem Synonymwörterbuch auszeichnet.

Außerdem wurden Berechnungsverfahren für Distanzrelationen, externe und interne räumliche Relationen sowie für dir Relation *auf* entwickelt, deren Bestimmung Vorraussetzung für eine linguistische Wegbeschreibung ist.

Schließlich folgte die Konzeption einer intelligenten, interaktiven, virtuellen Umgebung. Dies beinhaltet das dynamische und automatische Laden von benötigten Geometrien, die Modellierung von Türrotatationen sowie das Einblenden von Objektannotationen.

In den nächsten Kapitel wird nun betrachtet, wie diese Konzepte in GECL umgesetzt wurden.

Kapitel 5

GECL: Ein System zur Wissensmodellierung in der Gebäudenavigation

In diesem Kapitel wird die Architektur von dem in GECL enthaltenden Objektlokalisationssystems, seine Bedienung sowie die verwendete Ontologie beschrieben.

5.1 Architektur

Die Architektur ist in Abbildung 5.1 dargestellt. Gestrichelte Linien markieren Zugriffe auf die Wissensbasis, die Pfeile zeigen dabei in Richtung des Informationsflusses.

Zuerst gibt der Benutzer das gesuchte Ziel in das System ein. Die *Objektsuche* versucht anschließend ein mit der Benutzereingabe übereinstimmendes Objekt in der Wissensbasis zu finden. Das Zielobjekt wird nun grafisch markiert (Visualisierung) sowie eine natürlichsprachliche Lokalisationsbeschreibung generiert [1], wofür zuvor die entsprechenden räumlichen Relationen bestimmt werden müssen.

Alle in der Abbildung dargestellten Komponenten benötigen dabei Zugriff auf das vorliegende semantische Wissen und zwar wie folgt:

- Die natürlichsprachliche Eingabe verwendet semantische Informationen, um die Präzision des Parsings zu verbessern.
- Die Objektsuche vergleicht das vom Benutzer spezifizierte Objekt mit Objekten aus der Wissensbasis.
- Die Visualisierung stellt semantische Informationen über Objekte grafisch dar.
- Semantische Informationen können die Berechnung räumlicher Relationen vereinfachen (siehe Abschnitt 4.3).

[1] Zur Zeit nur implementiert für interne projektive Relationen

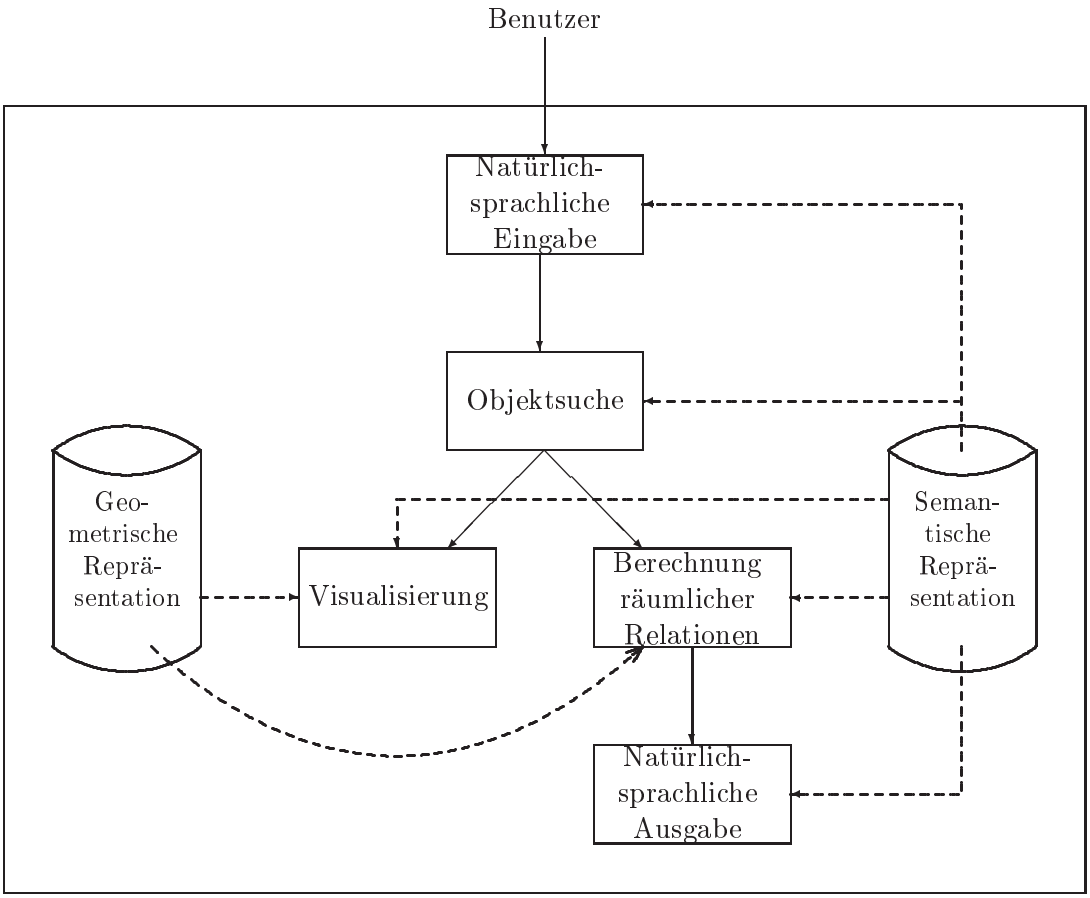

Abbildung 5.1: Systemarchitektur

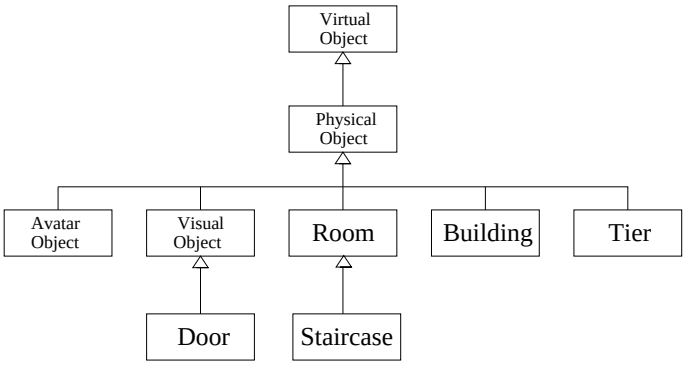

Abbildung 5.2: Ontologie

- Zur sprachlichen Lokalisationsbeschreibung wird Zugriff auf die Wissensbasis benötigt, um verwendete Referenzobjekte linguistisch beschreiben zu können.

Die geometrische Repräsentation wird einerseits für die Domänenvisualisierung verwendet. Andererseits werden aus ihr auch die für die Berechnung räumlicher Relationen benötigten Objektapproximationen berechnet.

5.2 Ontologie des Gebäudekonzepts

Dieser Abschnitt behandelt die in diesem System verwendete semantische Repräsentation sowie ihre Verbindung zu den Objektgeometrien.

5.2.1 Vererbungshierarchie in GECL

Einziges Element der Meta^2-Ebene (siehe Abschnitt 2.1.3) ist das *VirtualObject*, von dem alle übrigen Objekte abgeleitet sind. Die Metaebene enthält *VisualObject*, *PhysicalObject* und *AvatarObject*. Mit *PhysicalObject* werden Objekte bezeichnet, die eine Materie besitzen, also Lebewesen, Sachen oder Pflanzen, nicht dagegen Gefühle oder Ereignisse.

AvatarObject kennzeichnet den sich in der virtuellen Umgebung aufhaltenden Agenten. Dies ist im in GECL enthaltenen Objektlokalisationssystem ein Avatar, in RANA (siehe Abschnitte 1.2 und 7.3) ein Präsentationsagent.

Alle Objekte, für die eine geometrische Repräsentation definiert ist, befinden sich in der Klasse *VisualObject*. Elemente dieser Klasse sollten nicht mehr aus anderen Objekten zusammengesetzt sein. Ein Raum ist dagegen nicht von *VisualObject* abgeleitet, da er keine eigene geometrische Repräsentation besitzt, sondern nur deshalb sichtbar ist, weil seine Komponenten (Türen, Wände, Boden und Decke) es sind (siehe auch Abbildungen 5.2 und 5.3).

5.2.2 In GECL verwendete Assoziationen

Folgende Assoziationen werden explizit am Objekt annotiert:

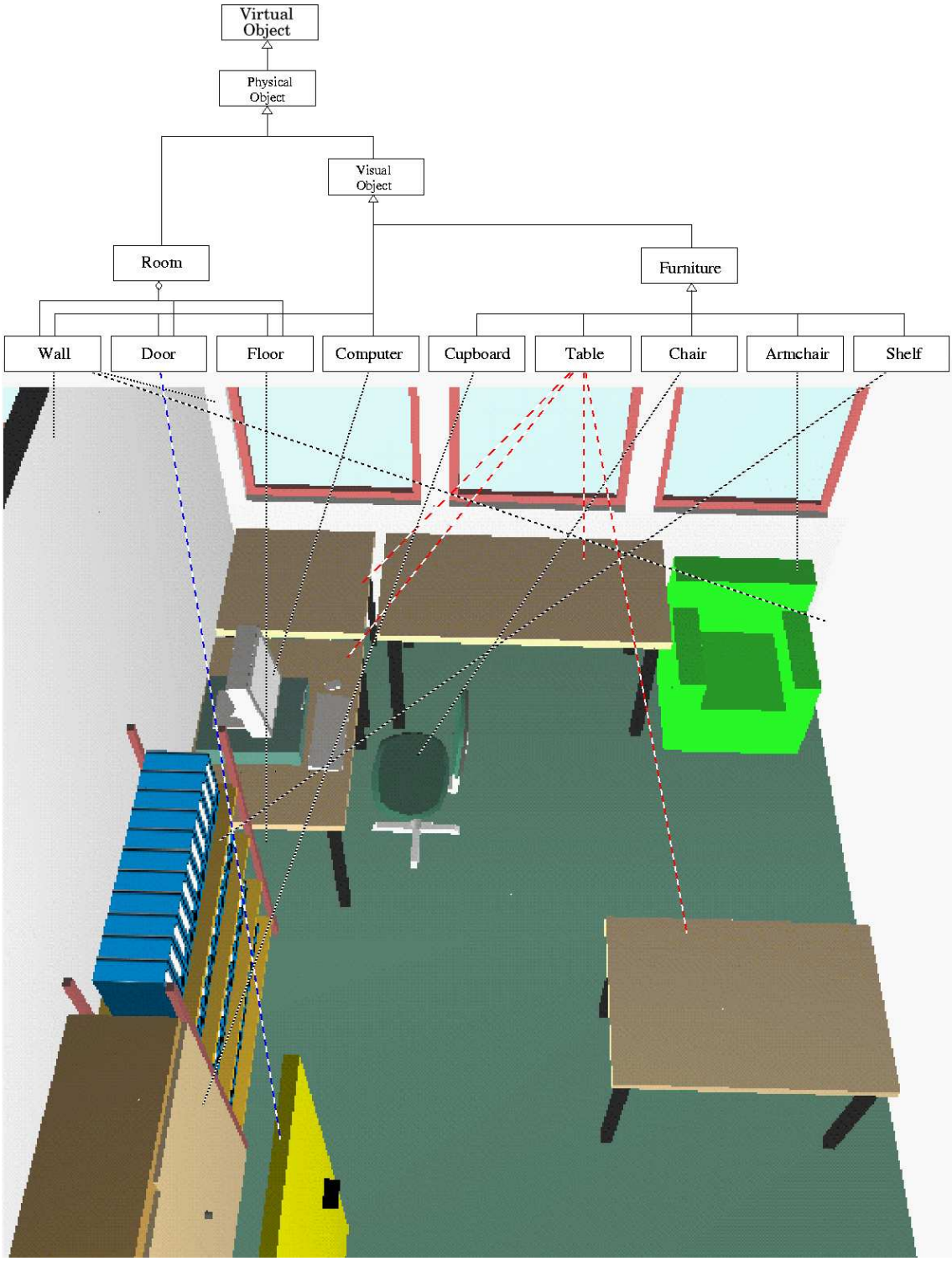

Abbildung 5.3: Beispiel für die Ontologie

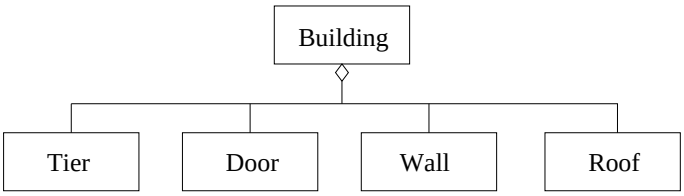

Abbildung 5.4: Gebäude

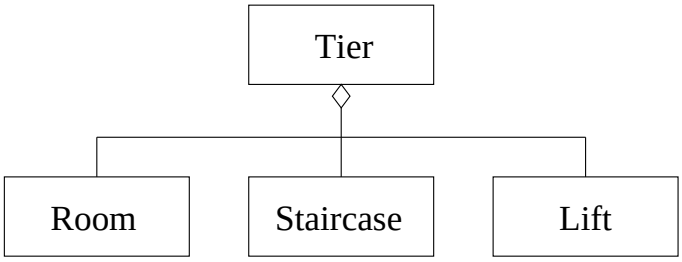

Abbildung 5.5: Etage

- Für jedes *PhysicalObject* wird angegeben, welche Objekte in dessen Inneren lokalisiert sind. Dies ist eine 1:n Relation, d.h. innerhalb eines Objektes können mehrere Objekte liegen, aber ein Objekt kann sich direkt nur innerhalb eines einzigen Objektes befinden. Dabei müssen die enthaltenen Objekte nicht zwangsläufig Gegenstände sein. In einem Raum kann beispielsweise eine dort stattfindende Party, also ein Ereignis, lokalisiert sein.

- Zu einem *PhysicalObject* existiert eine Assoziation zu seinem begrenzenden Quader. Dieses muß nicht notwendigerweise achsenparallel sein.

- Zu jedem *VisualObject* existiert eine Assoziation zu der entsprechenden geometrischen Darstellung

5.2.3 In GECL verwendete Aggregationen

Ein Gebäude setzt sich aus mehreren Etagen, einer Eingangstür, Außenwänden und einem Dach zusammen (siehe Abbildung 5.4). Eine *Etage* besteht aus Treppenhaus, Aufzug und Räumen (siehe Abbildung 5.5), wobei Aufzug und Treppenhaus normalerweise zu mehreren Etagen gehören. Die Komponenten eines Raumes sind Wände, Türen, Boden und Decke (siehe Abbildung 5.6). Dabei gehört die Decke auch zum Typ *Floor*, weil sie wiederum ein Boden eines Raumes der nächsthöheren Etage sein kann. Türen werden also nicht als Bestandteil der Wände definiert, was auch denkbar wäre, aber den Nachteil hätte, das eine Tür nicht mehr eindeutig einem Raum zugeordnet werden könnte.
Ein *Lift* setzt sich aus zwei Räumen zusammen, einmal aus der Aufzugskabine und zum

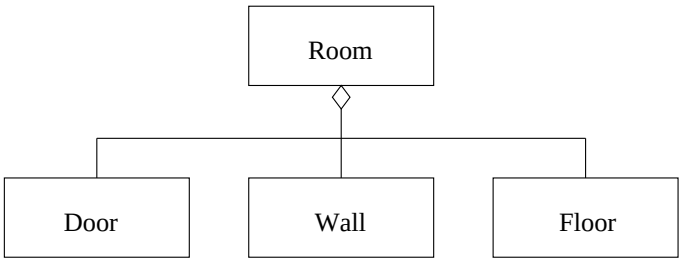

Abbildung 5.6: Raum

anderen aus dem Bereich, in dem sich diese bewegt.
Das *Treppenhaus* ist eine Spezialisierung von *Raum*, es besitzt als zusätzliche Komponenten die Treppe, das Geländer und die Böden von den Zwischenstockwerken.
Auch vom *Raum* abgeleitet ist die *Halle*. Sie definiert zusätzliche Wände, die bei der Errechnung des begrenzenden Quaders nicht berücksichtigt werden sollen. Dies sind normalerweise Wände im Innern.

In diesem Modell sind überlappende Aggregationen zugelassen, d.h. der Zusammensetzungsgraph ist kein Baum, sondern nur azyklisch. So können Wände als zu mehreren Zimmern gehörig gekennzeichnet werden.

Einrichtungsgegenstände, die sich innerhalb eines Raums befinden, sind nicht Teil des Raumes. Vielmehr besteht zwischen dem Raum und den Objekten in seinem Inneren die räumliche Beziehung *in*. Dies wird explizit im Modell abgespeichert und braucht daher nicht berechnet zu werden.

5.3 Benutzerschnittstelle

Im folgenden folgt eine Beschreibung, auf welche Weise das in GECL enthaltene Objektlokalisationssystem vom Anwender benutzt werden kann. Das beinhaltet Systemanforderungen, Beschreibung der Basisfunktionalität sowie dynamische Änderungen am 3D-Modell.

5.3.1 Systemanforderungen

Um das Objektlokalisationssystem zu benutzen werden folgende Komponenten benötigt:

- Java Development Kit 1.2
- Java3D 1.2
- der VRML-Loader für Java3D

Getestet wurde dieses Programm unter den Betriebssystemen:

- Microsoft Windows NT 4.0
- Debian GNU/Linux 2.2 (siehe [Deb01])

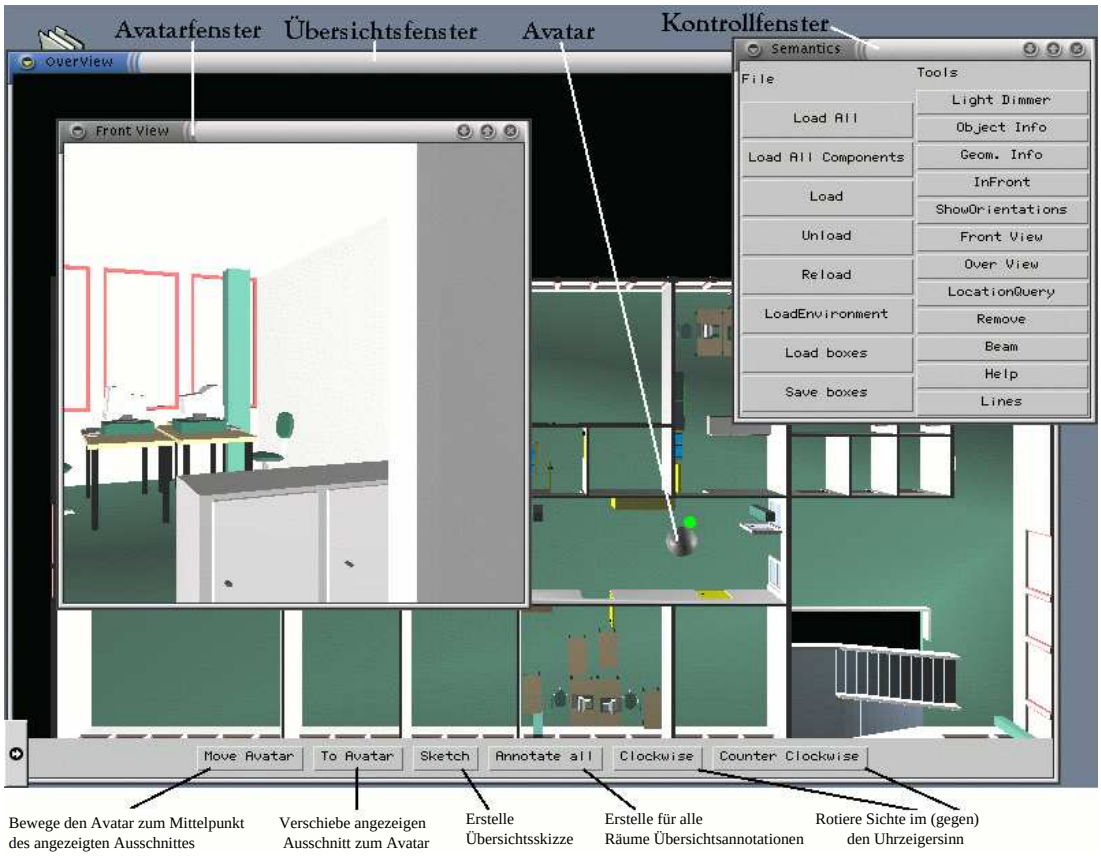

Abbildung 5.7: Übersichts, Avatar- und Kontrollfenster

Unter Linux wird die Benutzung der Blackdown-Portierung von Java bzw. Java3D empfohlen (siehe [Bla01]).

5.3.2 Basisfunktionalität

Das Programm kann durch das mitgelieferte Batch-Programm `av` oder direkt durch Eingabe von `java semantics.Semantics` gestartet werden. Optional kann als Parameter noch der Name des zu benutzenden Umgebungsladers angegeben werden. Mögliche Werte sind dabei `semantics.loaders.ObjectLoaderChair` für den Lehrstuhl bzw. `semantics.loaders.ObjectLoaderAirport` für den Flughafen

Falls kein Lader angegeben wurde, erscheint ein Auswahldialog, in dem man die zu ladende Umgebung anklicken kann. Bei der folgenden Beschreibung wird davon ausgegangen, dass der Anwender den Lehrstuhl (Chair) ausgewählt hat.

Kurz darauf öffnen sich dann drei Fenster, ein Übersichtsfenster, ein Fenster aus der Sicht des Avatars und ein Kontrollfenster (siehe Abbildung 5.7). Der Avatar wird dabei durch zwei Kugeln symbolisiert (siehe Abbildung 5.9). Das Übersichtsfenster ermöglicht

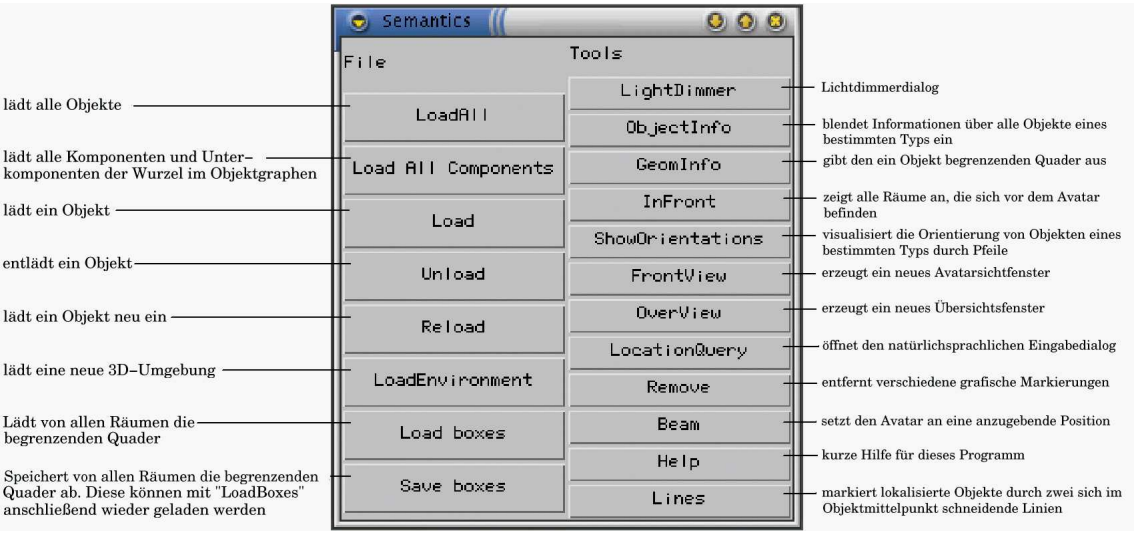

Abbildung 5.8: Kontrollfenster

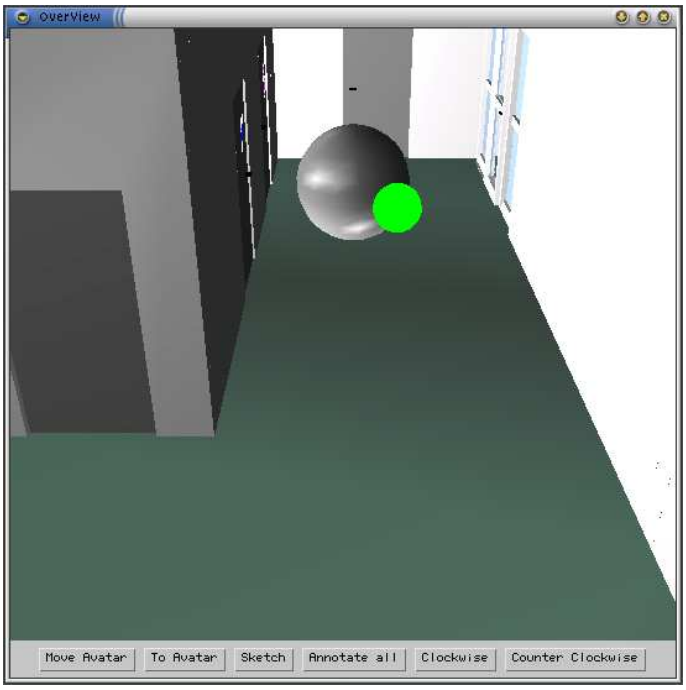

Abbildung 5.9: Idealisierung des Avatars

eine Betrachtung der Umgebung aus der Vogelperspektive und größerer Distanz. Das Avatarsichtfenster zeigt eine Abbildung der 3D-Welt aus der Sicht des Avatars. Im Kontrollfenster befinden sich zahlreiche Buttons mit denen der Benutzer verschiedene Funktionen des Programmes ausführen kann (siehe Abbildung 5.8).

Den Avatar navigiert man mit den Cursor-Tasten, sowie mit *u* wie *upper* um den Avatar nach oben und *l* wie *lower* um ihn nach unten zu bewegen. Um statt dem Avatar das Übersichtsfenster zu verschieben, muss zusätzlich die *Shift*-Taste gedrückt werden. Zusäzlich ist auch eine Navigation mit der Maus möglich. Mit der linken Maustaste rotiert man die Sicht, mit der mittleren verschiebt man sie und mit der rechten zoomt man heran oder heraus.

Objekte können durch eine natürlichsprachliche Eingabe lokalisiert werden. (siehe Abbildung 7.1), die man durch Klick auf *LocationQuery* im Kontrollfenster aufrufen kann. Der Eingabedialog kann multilinguale Eingaben bearbeiten, zur Zeit werden Deutsch und Englisch unterstützt.

Zusätzlich können Textannotationen eingeblendet werden, die Informationen über bestimmte Räume anzeigen (siehe Abschnitte 4.4.3 und 7.1).

5.3.3 Dynamische Modelländerungen

Um eine neue Welt einzuladen ist es nicht nötig das Programm nochmals zu starten. Man klickt dazu auf den Button *LoadEnvironment* des Kontrollfensters. Daraufhin öffnet sich ein Auswahlmenü, in dem die gewünschte Umgebung ausgewählt werden kann.

Um Teile eines verwendetes 3D-Modell zu modifizieren, ist es ebenfalls nicht nötig das Programm neu zu starten. Nachdem man die in den Verzeichnissen `semantics/resources/airport` bzw. `semantics/resources/chair` befindlichen Geometrien abgeändert hat, kann man die Änderungen in der 3D-Welt sichtbar machen, indem man das zugehörige Objekt über den Button *Reload* des Kontrollfensters neulädt. Dazu muss allerdings die Objekt-Identifizierung bekannt sein.

Diese Vorgehensweise des Neuladens ist allerdings nur möglich, falls die entsprechende Geometrie nicht als gemeinsam benutzt deklariert wurde. Ansonsten muss die gesamte Umgebung neu eingeladen werden. Dazu klickt man im Kontrollfenster auf den Button *LoadEnvironment* und wählt das augenblickliche Szenario aus.

Kapitel 6

Implementierung von GECL

Implementiert wurde dieses System in der Programmiersprache Java. Es verwendet dabei die Java3D-Klassenbibliothek (siehe [Bou99]). In Java3D werden für die bei 3D-Anwendungen geläufigen Konzepte wie Transformationsgruppe, Szenegraph, Lichtquelle u.s.w. entsprechende Klassen definiert. Empfehlenswert für eine ansprechende Performance ist dabei die Benutzung einer 3D-Grafikkarte.

6.1 Repräsentation der geometrischen Modelle

Die verwendeten Geometrien liegen im VRML97-Format (siehe [KRSD98]) vor, einem Dateiformat zur Beschreibung von 3D-Welten. Es kann direkt mit einem Texteditor eingegeben oder auch durch ein entsprechendes 3D-Modellierungsprogramm erzeugt werden. Mit speziellen Plugins kann man diese durch VRML-Dateien erstellten virtuellen Umgebungen auch mit einem Internetbrowser betrachten. Mehr Informationen zu VRML findet man unter [KRSD98]. Das VRML-Format wird in Zukunft veraussichtlich durch das XML-kompatible X3D-Format (siehe [Web01]) ersetzt werden.

6.2 Objektrepräsentationen

Jeder Geometrie wird in GECL ein Java-Objekt zugeordnet, wobei umgekehrt aber nicht für jedes Objekt auch eine Geometrie definiert sein muss. Alle Java-Objekte enthalten semantische Informationen wie den Objektnamen, seine Nummer, den Besitzer u.s.w.. Die darin abgelegte Information ist dabei keineswegs fest begrenzt, sondern kann vom Benutzer beliebig erweitert werden. Die Basisklasse aller dieser Objekte ist das *VirtualObject*. Es besitzt den folgenden Konstruktor:

```
public VirtualObject(String name)
```

Eine Unterklasse von *VirtualObject* ist das *PhysicalObject*. Es enthält alle Objekte, die aus Materie bestehen. Der Konstruktor entspricht demjenigen des *VirtualObject*s.

Objekte, denen eine Geometrie zugeordnet ist, sind grundsätzlich von *VisualObject* abgeleitet. Eine solche geometrische Beschreibung kann dabei von einem Objekt alleine oder auch von mehreren Objekten zusammen verwendet werden.

6.3 Eingabedaten

Die Beschreibung der zu verwendeten Ontologie muss nicht in einem bestimmten Datenformat vorliegen. Statt dessen müssen zwei Java-Klassen konstruiert werden, ein Objektlader und eine virtuelle Umgebung (siehe Abschnitt A). Um diesen Objektlader zu verwenden, muss er beim Programmstart als Argument übergeben werden. Falls er im Verzeichnis `semantics/loaders` abgelegt wird, kann man ihn auch aus einem Auswahlmenü auswählen.

Eingabedaten für Synonym-, Übersetzungs- und Worttypenlexikon sind jeweils im Java-Properties-Format anzugeben. Dabei ist zu beachten, dass Synonym- und Übersetzungswörerbuch vom System als konzeptionell identisch angesehen werden.

Beim Synonymwörterbuch erscheint am Anfang jeder Zeile das Wort, für das ein Synonym definiert werden soll. Anschließend folgt stets ein Gleichheitszeichen und dahinter durch Kommas getrennte Synonyme.
Beispiel:

```
Grocery store= grocery shop, supermarket, grocery's
```

Auf der linken Seite des Gleichheitszeichens befindet sich beim Übersetzungswörterbuch immer die englische Übersetzung der auf der rechten Seite durch Kommas getrennten Wörter in der anderen Sprache. Beispiel:

```
Hairdesser= frisörladen, friseur
```

Die Worttypenliste spezifiziert, welche Wörter zu welchen Worttypen gehören. Beispiel:

```
ART=the,a,an
```

Beim Systemstart zu ladende Plugins können in der
Datei `semantics/resources/classlist1` bzw. `semantics/resources/classlist2` spezifiziert werden. Die Plugins werden dabei in der Reihenfolge ihres Auftretens in den entsprechenden Dateien initialisiert. Der anzugebende Klassenname muss dabei die Paketbezeichnung beinhalten.

6.4 Klassenstrukturbeziehungen

6.4.1 Wissensrepräsentation

Jedem `ObjectLoader` ist genau eine `VirtualEnvironment` zugeordnet (siehe Abbildung 6.1). Der `ObjektLoader` baut die 3D-Welt zusammen und ordnet den semantischen Objekten die entsprechenden Geometrien zu. Außerdem erstellt er eine `VirtualEnvironment`, die

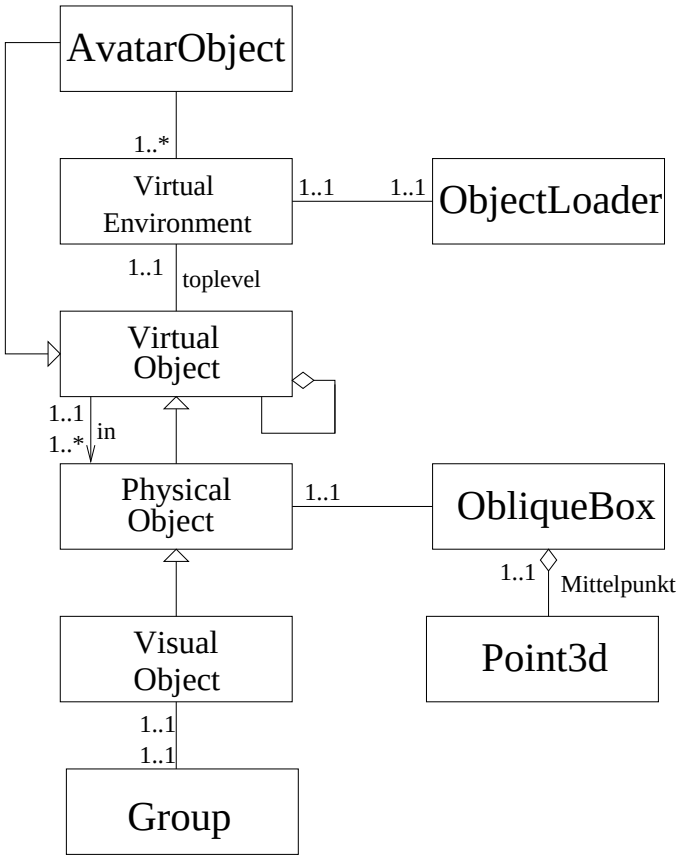

Abbildung 6.1: Wissensrepräsentation

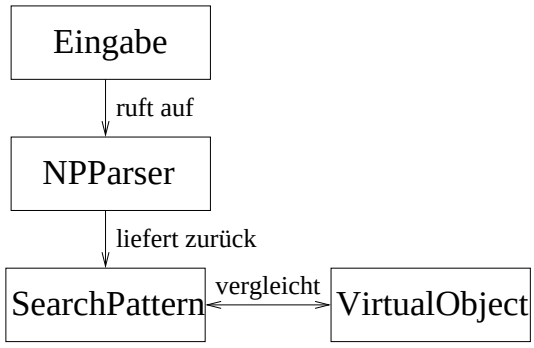

Abbildung 6.2: Grobarchitektur der natürlichsprachlichen Eingabe

verschiedene für das Szenario relevante Funktionen enthält. Es wird deswegen überhaupt zwischen `ObjectLoader` und der `VirtualEnvironment` unterschieden, weil der `ObjectLoader` normalerweise ein sehr umfangreiches Objekt ist. So kann dieser nach dem Laden der Umgebung, um Speicherplatz zu sparen, wieder dereferenziert werden, da alle für die Umgebung relevanten Funktionen sich in der `VirtualEnvironment` befinden.

Eine Referenz auf das von `VirtualObject` abgeleitete Wurzelobjekt des Objektgraphen findet sich im Attribut `m_toplevel` der `VirtualEnvironment`. Ein `VirtualObject` kann dabei wieder aus anderen `VirtualObjects` zusammengesetzt sein. `PhysicalObjects` bezeichnen `VirtualObjects` mit Materie. In ihrem inneren können andere Objekte lokalisiert sein. Außerdem besitzen sie eine Referenz zu ihrer Objektidealisierung (`ObliqueBox`), mit deren Hilfe räumliche Relationen berechnet werden können. Objekte, denen direkt eine geometrische Beschreibung (`Group`) zugeordnet ist, müssen Unterklassen von `VisualObject` sein.

6.4.2 Die natürlichsprachliche Anfrage

Grob gesehen wandelt der `NPParser` die natürlichsprachliche Eingabe in ein Suchmuster (`SearchPattern`) um, mit dessen Hilfe das gesuchte Objekt dann lokalisiert wird (siehe Abbildung 6.2).

Für die natürlichsprachliche Eingabe wird als Eingabe eine Anfragezeichenkette sowie Informationen über die benutzte Sprache benötigt. Mit diesen beiden Informationen wird die statische Methode `query()` der Klasse `Query` aufgerufen, die ein `VDict`-Objekt abhängig von der gewählten Sprache erzeugt. Durch `NPParser.segmentate()` wird der Eingabestring in mehrere Einzelanfragen zerlegt und mittels `NPParser.parse()` die enthaltene Information über das gesuchte Zielobjekt in ein `NP`-Objekt extrahiert.

Das `NP` wird anschließend in ein `SearchPattern` konvertiert. Personennamen werden dabei durch das Objekt `Name` repräsentiert, das diese in Titel, Vor- und Nachnamen zerlegt. Nun wird eine Liste von in Frage kommenden `VirtualObjects` mit dem `SearchPattern` verglichen und bei Übereinstimmung in einem Vektor gespeichert. Mehrdeutigkeiten müssen dabei eventuell vom Benutzer durch einen `ElementSelector` (z.b.

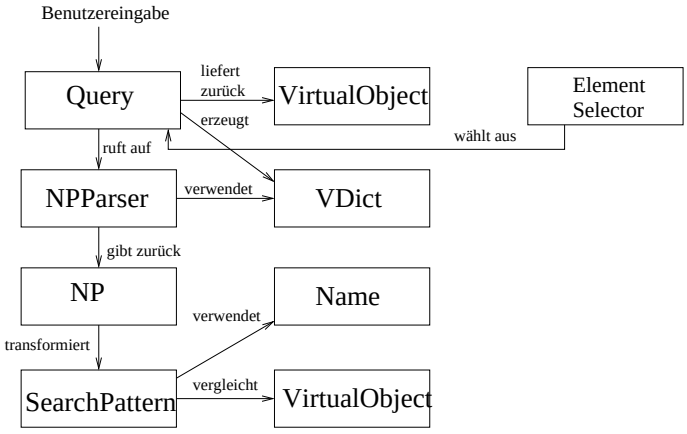

Abbildung 6.3: Feinarchitektur der natürlichsprachlichen Eingabe

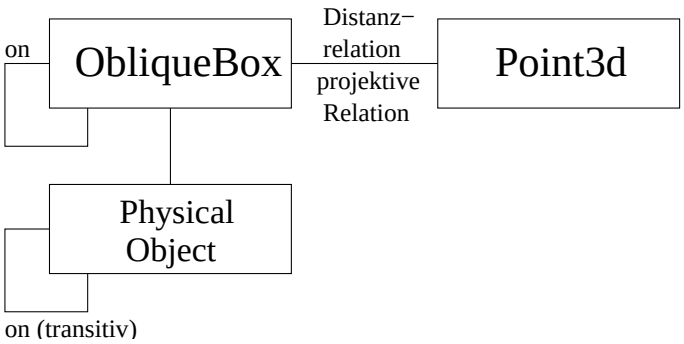

Abbildung 6.4: räumliche Relationen

`ListElementSelector`) aufgelöst werden. Einen fehlertoleranten Vergleich von zwei Zeichenketten implementiert `Fuzzy.equals()`, welcher beim Vergleich von Besitzer- und Objektnamen zum Einsatz kommt.

6.4.3 Berechnung von räumlichen Relationen

Zur Berechnung von Distanz- und projektiven räumlichen Relationen wird für das Referenzobjekt eine Approximation durch einen schiefes Quader (`ObliqueBox`) und für das zu lokalisierende eine Punktapproximation (`Point3d`) benötigt (siehe Abbildung 6.4). Soll dagegen die Anwendbarkeit von *auf* bestimmt werden, kommen Quaderapproximation von beiden Objekten zur Verwendung. Bei transitiver Verwendung von *auf* muß sogar die semantische Repräsentation (`PhysicalObject`) bekannt sein.

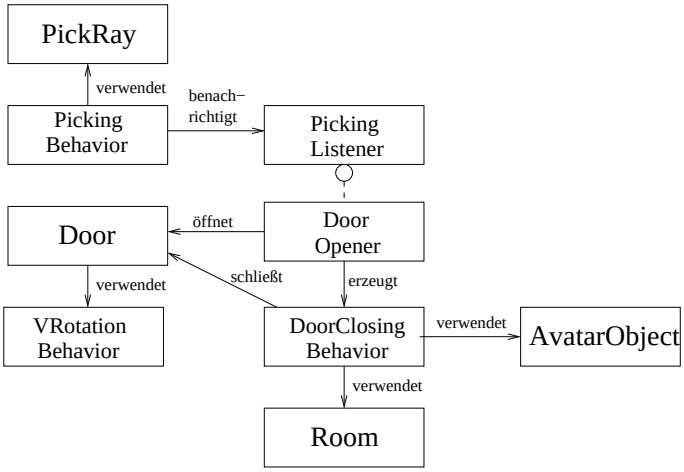

Abbildung 6.5: Picking

6.4.4 Interaktionen mit der 3D-Welt

Dieser Abschnitt behandelt die Implementierung des automatischen Ladens von Geometrien, die Türanimationen sowie die eingeblendeten Textannotationen.

Während der Avatar sich durch die virtuelle Umgebung bewegt, tastet er die Umgebung mittels einem oder mehrerer `PickRay`s ab. Beim `PickingBehavior` können sich dabei verschiedene `PickingListener`s registrieren, die dazu angeben müssen, an welchen der `PickRay`s sie interessiert sind.

Ein solcher `PickingListener` ist auch der `DoorOpener`, der ausschließlich den nach vorne verlaufenden `PickRay` benutzt. Falls dieser eine Geometrie schneidet, ermittelt der `DoorOpener` mit Hilfe der `VirtualEnvironment` das dazugehörige `VisualObject`. Sollte dieses eine `Door` sein, gibt er ihr den Hinweis sich zu öffnen. Die zugehörige Bewegungsanimation modelliert das `VRotationBehavior`.

Zusätzlich erzeugt der `DoorOpener` in diesem Fall ein `DoorClosingBehavior`, welches den Abstand vom `AvatarObject` mit der `Door` in bestimmten Zeitintervallen überprüft. Falls dieser einen bestimmten Schwellenwert überschreitet, wird die `Door` durch das `VRotationBehavior` wieder geschlossen. Außerdem registriert sich das `DoorClosingBehavior` noch als `RotationListener` beim `VRotationBehavior`, was dazu führt, das ersterer bei Beendigung der Rotation benachrichtigt wird.
Das `DoorClosingBehavior` überprüft nun, ob die an die geschlossene `Door` angrenzenden `Room`s entladen werden dürfen. Falls dies möglich ist, werden die `load()`-Methoden der entsprechenden `Room`s mit dem Parameter zum Entladen aufgerufen.

6.4.4.1 Textannotationen

Die Basisklasse für die Einzelannotation heißt `Annotation`. Der Informationstext wird damit so im Fenster positioniert als ob er sich an einer vom Anwender zu bestimmenden

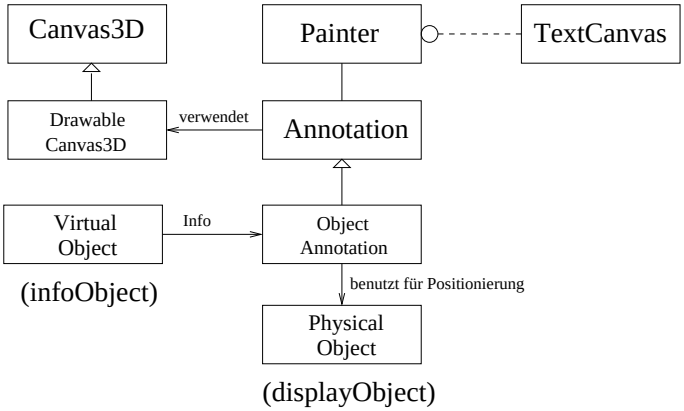

Abbildung 6.6: Objektannotation

Stelle in der 3D-Welt befinden würde.

Davon abgeleitet ist die `ObjectAnnotation`. In diesem Fall verknüpft man die Annotation mit zwei Objekten. Dabei bezieht sich die angezeigt Information auf das `infoObject`, positioniert wird diese an der Stelle des `displayObject`. Dabei können beide natürlich auch zusammenfallen. Wird diese Annotation aus der Avatarsicht eingeblendet, sieht man die über einen Raum angezeigte Information an einer seiner Türen, d.h. das `displayObject` ist die Tür, das `infoObject` aber der Raum. Für die Annotation kann spezifiziert werden, ob sie automatisch wieder ausgeblendet werden soll, falls der Avatar sich um eine bestimmte Strecke von dem `displayObject` entfernt hat oder einen zu spezifizierenden Raum verlässt

Ähnlich wie die Einzelannotation wird auch die Übersichtsannotation (`vr.behaviors.OverViewInfo`) mit dem zu annotierenden Objekt verknüpft. Dabei gibt es in diesem Fall allerdings keine Unterscheidung zwischen `displayObject` und `infoObject`. Zusätzlich benötigt die Übersichtsannotation eine Referenz auf die Transformationsgruppe, die die Transformation für die Sicht definiert, mit der der Benutzer auf die 3D-Umgebung blickt. Dies ist notwendig, um die eingeblendete Schrift immer in Richtung des Anwenders zu orientieren.

Kapitel 7

Arbeiten mit GECL

7.1 Suche nach Person und Objekt

Nehmen wir einmal an, jemand möchte im Lehrstuhl zu Frank Wittig, wüßte aber weder dessen Vornamen noch die genaue Schreibweise des Nachnames. Zusätzlich wolle er wissen, wo sich der Computer Abacus befindet. Er würde also dazu im Kontrollfenster auf *LocationQuery* klicken und im daraufhin geöffneten Dialog beispielsweise eingeben (siehe Abbildung 7.1):

„(Wo sind) Herr Wittik und der Computer Abacus? “

Der natürlichsprachliche Parser würde die Eingabe segmentieren in
(Herr Wittik, Computer Abacus) (siehe Abschnitt 4.2.2.1). Der 1. Teil der Eingabe wird daraufhin geparst zu

Objektname	Herr Wittik
Objektbesitzer	-
Objektnummer:	-
Objektklasse:	-

Anschließend überprüft die Fuzzysuche in der Wissensbasis enthaltene Objekte auf Übereinstimmung mit den angegebenen Attributen. Da Personen in diesem System nur als Besitzer von Objekten auftreten können, kann kein entsprechendes Objekt gefunden werden.

Abbildung 7.1: natürlichsprachlicher Eingabedialog

In einem zweiten Suchdurchlauf wird nun auf Übereinstimmung mit Hilfe der Tippfehlerkorrektur geprüft. Nachdem auch dies erfolglos blieb, erfolgt eine Umformulierung des Suchmusters zu

Objektname:	-
Objektbesitzer:	Titel: Herr, Vornamen: ∅, Nachname: Wittik
Objektnummer:	-
Objektklasse	Objekt

Es wird also nun irgendein Objekt gesucht, welches Herrn Wittik gehört. Allerdings bleibt auch der dritte Suchdurchlauf aufgrund des Tippfehlers erfolglos. Im vierten und letzten Suchprozess stößt die Fuzzy-Suche irgendwann auf folgenden Eintrag:

Objektname:	-
Objektbesitzer:	Titel: Herr, Vornamen: Frank, Nachname: Wittig
Objektnummer:	118
Objektklasse:	Büro

Dieses Objekt kann nun mit dem Suchmuster identifiziert werden, da

- Objektname und Objektnummer im Suchmuster nicht angegeben wurden und daher hier ein Vergleich entfällt.
- Die im Suchmuster spezifizierte Klasse *Objekt* eine Generalisierung von *Büro* ist.
- nach Anwendung der Tippfehlerkorrektur beide Nachnamen als identisch betrachtet werden (Vertipper beim letzten Buchstaben in *Wittik*).
- ein Vorname im Suchmuster nicht spezifiziert wurde, wodurch ein sonst durchzuführender Vergleich entfällt.

Nun muss noch der Computer Abacus lokalisiert werden. Das Ergebnis des Parsingprozesses ergibt:

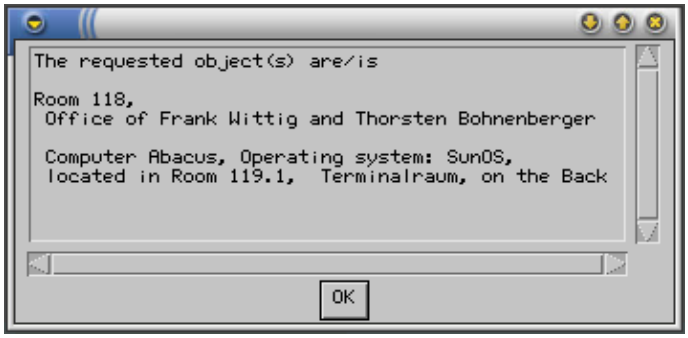

Abbildung 7.2: Objektbeschreibung

Objektname:	Abacus
Objektbesitzer:	-
Objektnummer:	-
Objektklasse:	Computer

Dies korrespondiert direkt zu dem in der Wissensbasis enthaltenen Objekt:

Objektname:	Abacus
Objektbesitzer:	-
Objektnummer:	-
Objektklasse:	Computer

Für beide Objekte wird nun gemäß Abschnitt 4.1.3 eine sprachliche Beschreibung generiert und ausgegeben. Da sich der Computer Abacus innerhalb eines Raumes befindet, wird in diesem Fall zusätzlich die bestanwendbare interne projektive Relation bestimmt, die die Lage des Computers innerhalb des entsprechenden Raumes angibt, und in die textuelle Beschreibung integriert.

Außerdem werden beide Objekte durch eine grüne Kugel markiert. Dazu werden die aus den Geometrien berechneten Objektmittelpunkte verwendet. Der Mittelpunkt eines Raumes ergibt sich dabei aus dem Zentrum des den Raum umschließenden Quaders, welches gemäß Abschnitt 4.4.1 bestimmt wurde.

Eine andere Methode um eine Beschreibung eines Raumes zu erhalten, besteht darin, auf den entsprechenden Raum mit der Maus zu klicken. Nehme man beispielsweise an, jemand klickt mit der Maus auf das oben erwähnte Büro von Frank Wittig. Nun wird zuerst mit dem in Abschnitt 4.4.4 beschriebenen Algorithmus festgestellt welcher Raum vom Benutzer selektiert wurde. Anschließend wird wie in Abschnitt 4.1.3 beschrieben eine entsprechende Raumbeschreibung generiert und im Ausgabefenster positioniert (siehe Abschnitt 4.4.3.1). Die Genauigkeit der Darstellung lässt dabei durch weitere Klicks auf den Raum erhöhen (siehe Abbildung 7.4). Durch Drücken der Taste T kann man die Transparenz des Hintergrundes der Annotation von nicht transparent, halbtransparent bis ganz transparent ändern (siehe Abbildung 7.5). Dieser Typ von Annotation wird in dieser Arbeit als Einzelannotation bezeichnet, weil zur gleichen Zeit nur ein einziges Objekt damit annotiert werden kann. Sie besitzt immer eine konstante Größe, was man nachprüfen kann,

Abbildung 7.3: Markierung der Objekte

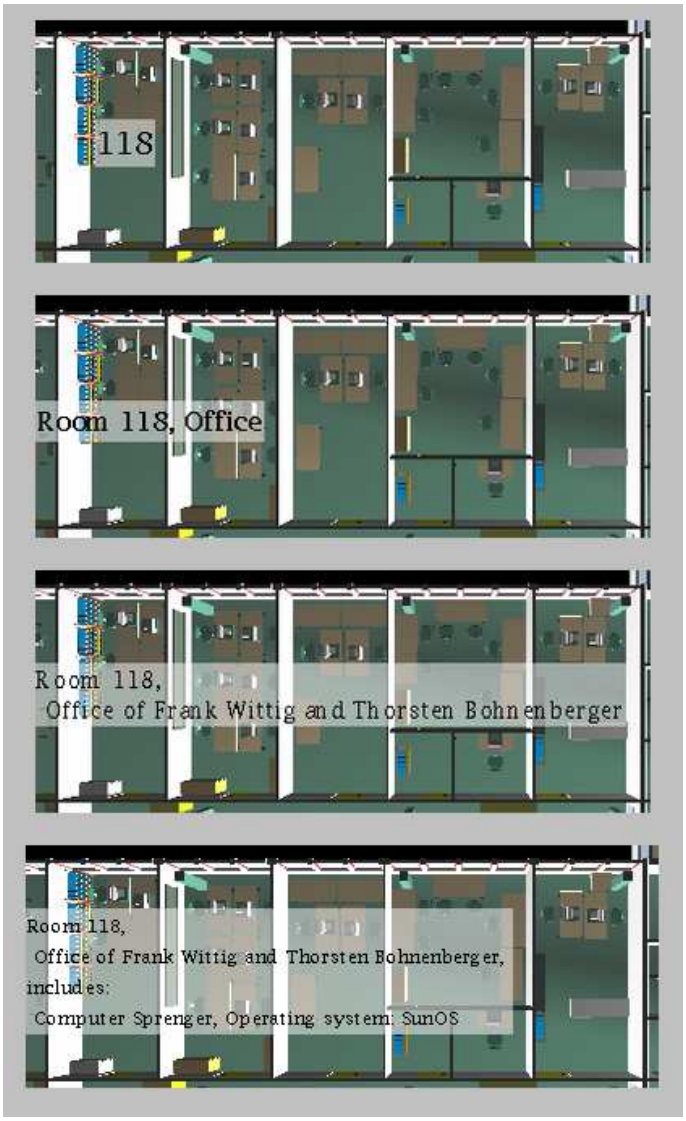

Abbildung 7.4: Darstellung in unterschiedlicher Granularität

Abbildung 7.5: Darstellung in unterschiedlicher Transparenz

indem man mit der rechten Maustaste vor- und zurückzoomt. Neben der Einzelannotation gibt es auch noch die Übersichtsannotation. Aktivieren kann man diese durch Klick auf den Button *Annotate All* im Übersichtsfenster. Daraufhin werden für alle Räume der Umgebung kurze Informationstexte eingeblendet (siehe Abbildung 4.20). Wenn man jetzt die Sicht mit dem Button *Clockwise* rotiert, bleibt die Schrift dabei immer zum Betrachter orientiert und paßt sich wie in Abschnitt 4.4.3.2 beschrieben dynamisch dem verfügbaren Platz bei dem entsprechenden annotierten Objekt an. Wenn man jetzt mit der rechten Maustaste herauszoomt, ist zu erkennen, dass sich die Schriftgröße im Gegensatz zur Einzelannotation verringert.

Bei zu hoher Lichintesität können manchmal Objekte nicht mehr richtig erkannt werden. Dieses Problem kann mit dem integrierten Lichtdimmer behoben werden. Aufrufen kann man ihn durch Klick auf *Light Dimmer* im Kontrollfenster. Mithilfe eines Schieberegler kann die Helligkeit auf den gewünschten Wert eingestellt werden. Dabei können die durchgeführten Einstellungen pro Benutzer gespeichert werden.

Abbildung 7.6: Dynamisches Austauschen von Geometrien

7.2 Dynamisches Ändern einer Geometrie

Dieses Beispiel beschreibt, wie man ein 3D-Modell während der Laufzeit des Programmes austauscht. Angenommen es soll im Flughafenszenario das Aussehen einer Wand geändert werden, und zwar soll die rechte lange Wand vom Geschäft Harrods rot eingefärbt werden (siehe Abbildung 7.6). Dazu klickt man zuerst die entsprechende Wandgeometrie mit der Maus an. Das System ermittelt daraufhin das zu dieser Geometrie gehörende semantische Objekt (siehe Abschnitt B.2) und gibt eine Kurzbeschreibung desselben auf der Befehlskonsole aus (in diesem Fall: `wharrods3`).

Wenn man nun in das Verzeichnis, in dem sich die Wandgeometrien des Flughafens befinden (`semantics_dist/semantics/resources/walls`) wechselt, kann man dort die Datei finden, die die Geometrieinformation für dieses Objekt enthält. Der Name einer solchen Datei setzt sich im allgemeinen aus dem Objektnamen (hier *wharrods3*) und der Endung *.wrl* zusammen. Dabei sollte die Datei *wharrods3.wrl* gefunden werden.

Nun kann man die Farbe des in dieser Datei enthaltenden 3D-Modells beispielsweise durch Eingabe des Befehls

```
java vr.tools.ChangeColor wharrods3.wrl -c 1 0 0
```

in rot ändern

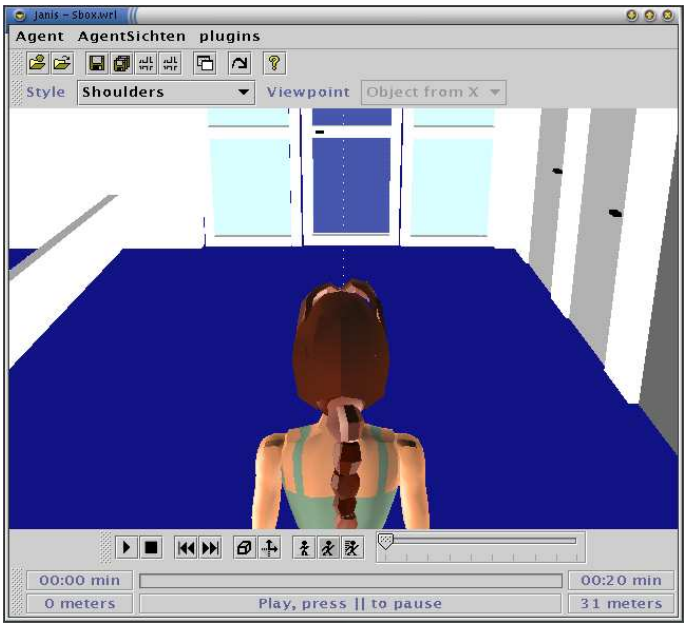

Abbildung 7.7: Lara in unserem Flughafenlehrstuhl

Anschließend klickt man auf den Button *Reload* im Kontrollfenster, gibt den Objektnamen *wharrods3* in den sich öffnenden Dialog ein und klickt auf OK. [1]. Daraufhin wird durch die Funktion *getAnyObject* (siehe Abschnitt B.2) das zugehörige semantische Objekt durch Absuchen des Objektgraphen ermittelt und neu eingeladen. Man sieht nun, dass die Farbe der Wand sich von grün in rot geändert hat.

7.3 Integriertes Projekt RANA

Dieses System vereinigt die Komponenten Planung, Wegsuche (Komponente RAW [2]), Avatarvisualisierung (AJAPA [3]), semantische Repräsentation und grafische Abstraktion.

Für dieses Gesamtsystem wurde weder der Lehrstuhl noch der Flughafen benutzt sondern Flughafen und Lehrstuhl aneinandergebaut. Dies hat den Vorteil der größeren Übersichtlichkeit, weil der Benutzer sich nur noch mit einem Szenario auseinandersetzen muss. Um die Navigation im Lehrstuhl realistischer zu machen und dem Flughafen anzugleichen, wurden aus manchen Büros Geschäfte gemacht (Raum 125=McDonals).

Wechseln Sie zum Start der Anwendung in das Verzeichnis `janis` und geben Sie

[1] Dies ist nicht ausreichend bei Geometrien, die von mehreren Objekten gemeinsam verwendet werden. In diesem Fall muss die aktuelle Umgebung mit dem Button *Load Environment* aus dem Kontrollfenster neu eingeladen werden.

[2] RAW ist das Akronym für **R**essourcen**a**daptive **W**egsuche

[3] AJAPA ist das Akronym für **A** **Ja**va **P**resentation **A**gent

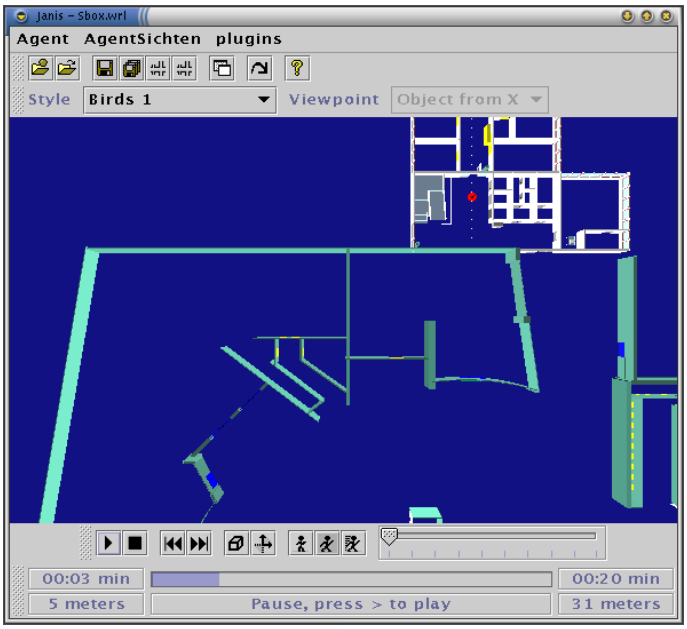

Abbildung 7.8: Vogelperspektive

`./sjanis` ein . Das System baut nun die semantische Repräsentation des Flughafenlehrstuhls auf (siehe Abschnitt 5.2) und weist den Objekten die entsprechenden Geometrien zu. Anschließend erscheint ein grafischer Eingabedialog, bei der der Anwender die Auswahl zwischen mehreren verschiedenen Wegen besitzt, von denen er einen anklicken kann.

Nach der Auswahl des gewünschten Weges durch den Anwender, erfolgt eine Animation mit einem Präsentationsagenten, bei der dieser den angeklickten Weg abläuft (siehe Abbildungen 7.7 und 7.8). Die Geometrien dieses Agenten sind dynamisch austauschbar. Man hat dabei die Wahl zwischen Lara Croft und Jim (siehe [Bre01]). Wenn dieser Agent sich nun einer Tür nähert, öffnet sich diese und die Geometrien des dahinterliegenden Raumes werden automatisch eingeladen [4].

Die Animation erfolgt dabei ressourcenadaptiv. Hat der Benutzer viel Zeit zur Verfügung, erfolgt eine ausführlichere Darstellung aus der Ego-Perspektive, bei der der Präsentationsagent auf Landmarken mit Hilfe von Zeigegesten aufmerksam macht. Zusätzlich werden Objektinformationen eingeblendet, zu deren Positionierung auf den Mittelpunkt der Quaderapproximation des entsprechenden Objektes zurückgegriffen wird (siehe Abschnitte 4.4.1 und 4.3.2). Als Landmarken werden dabei vom System auffällige Objekte an Abbiegepunkten ausgewählt.

Bei Zeitknappheit erfolgt eine beschleunigte Darstellung aus der Vogelperspektive.

[4] siehe Abschnitte 4.4.2 und 4.4.2.3

Kapitel 8

Zusammenfassung und Ausblick

8.1 Erzielte Ergebnisse

In dieser Arbeit wurden folgende Ergebnisse erzielt:

- das System ist in der Lage natürlichsprachliche Lokalisationsanfragen zu verarbeiten.
- Es wurden verschiedene Verfahren zur Berechnung räumlicher Relationen entwickelt und implementiert. Diese werden benötigt, falls der Weg natürlichsprachlich beschrieben werden soll
- Informationen über Objekte können auf unterschiedliche Weise und in mehreren Granularitätsstufen visualisiert werden.
- Es wurden Verfahren diskutiert und implementiert, wie eine virtuelle Umgebung intelligent auf einen sich in dieser befindlichen Agenten reagieren kann.
- Es wurde eine Ontologie definiert, mit deren Hilfe Objekte in einem Gebäudenavigationssystem geeignet semantisch repräsentiert werden können.
- es wurde eine Klassenbibliothek und ein Objektlokalisationssystem implementiert, welches die obigen Konzepte geeignet umsetzt. Dieses Objektlokalisationssystem ist in der Lage dem Benutzer das gesuchte Objekt sprachlich zu beschreiben und dessen Position grafisch zu markieren. Falls sich das lokalisierte Objekt innerhalb eines anderen befindet, wird seine Lage bezüglich des ihn enthaltenden Objektes mittels interner projektiver Relationen beschrieben.

Außerdem wurde untersucht auf welche Weise semantische Informationen über Objekte bei verschiedenen Aufgaben in einem Gebäudenavigationssystem verwendet werden können. Diese beinhalten:

- Durch Wissen über vorkommende Objekte kann die Präzision des Parsings der natürlichsprachlichen Benutzereingabe erhöht werden.

- Es können bei der Lokalisationsanfrage Unterklassenbeziehungen berücksichtigt werden. So ist der vom Benutzer gesuchte *Raum von Frank* möglicherweise als *Büro von Frank* in der Wissensbasis repräsentiert. Wenn dem System nun bekannt ist, dass *Büro* eine Spezialisierung von *Raum* ist, kann es trotzdem beide Begriffe miteinander identifizieren

- Objektinformationen können verschieden detailliert ausgegeben werden.

- Aus der Vererbungshierarchie lassen sich verschiedene für die Bestimmung räumlicher Relationen erforderliche Informationen ableiten, die sonst nur schwierig aus den Objektgeometrien zu bestimmen wären (beispielsweise, ob ein Objekt eine intrinsische Front besitzt oder nicht).

- Kompositionale Zusammensetzungsinformationen und innerhalb-Beziehungen können zum inkrementellen Laden von Objektgeometrien verwendet werden. So impliziert das Laden eines Zimmers beispielsweise das Laden dessen Wände, Türen, Decke, Boden und Einrichtung.

- Für eine sprachliche Wegbeschreibung ist semantische Informationen über die dort verwendeten Bezugsobjekte erforderlich.

8.2 Erweiterungsmöglichkeiten

Im folgenden werden verschieden Möglichkeiten besprochen, wie dieses System erweitert und verbessert werden könnte.

8.2.1 Mögliche Erweiterungen bei der Eingabe

Wünschenswert wäre hier die Möglichkeit das aufzufindende Zielobjekt auch in gesprochener Sprache zu beschreiben. So gibt es seit geraumer Zeit ein Java API (JavaSpeech, siehe [Sun01b]), dass mit verschiedenen Spracherkennungssystemen (z.B. Voicetype, siehe [IBM01]) gekoppelt werden kann. Dabei muss berücksichtigt werden, dass in gesprochener Sprache oft andere Formulierungen benutzt werden als in geschriebener. Beispielsweise sagt man im Deutschen oft: „Wo ist dem Prof. Newton sein Büro “anstatt „Wo ist das Büro von Prof. Newton? “.

Weiterhin könnte man den natürlichsprachlichen Parser auf andere Konstrukte als Nominalphrasen erweitern, so dass beispielsweise auch Relativsätze korrekt erkannt werden (z.B. „Wo ist der Raum, in dem sich der Computer Eugene befindet? “). Ferner wäre eine Erweiterung des Vokabulars auf qualitative Ausdrücke vorstellbar (z.B. „Wo befindet sich der schnellste Rechner? “). Zur Zeit muss man die benutzte Sprache noch explizit angeben. Dies könnte durch eine automatische Erkennung der benutzten Sprache vermieden werden. Oft werden daür Wortlisten oder Trigramme eingesetzt (siehe [Lan01]).

8.2.2 Sonstige Erweiterungsmöglichkeiten

Für eine natürlichsprachliche Lokalisationsbeschreibung werden zur Zeit nur interne projektive Relationen verwendet. Hier wäre eine Erweiterung auf externe projektive Relatationen und Distanzrelationen sowie die Einbeziehung von linguistischen Hecken [1] vorstellbar.

Außerdem wäre sicherlich eine noch realistischere grafische Darstellung vor allem im Flughafenszenario wünschenswert. Zudem ist die Annotation der geometrischen Objekte mit semantischer Information etwas mühsam. Von Nutzen wäre hier, wenn direkt im 3D-Editor den Geometrien diese Informationen zugeordnet werden könnte.

8.3 Ausblick

Für Fußgänger wäre eine weitere Miniaturisierung der tragbaren Wegauskunftssysteme sinnvoll, da das Mitschleppen benötigter Hardware den Benutzer dazu veranlassen könnte auf ein solches System zu verzichten. In der Zukunft werden Wegauskunftssystem daher möglicherweise in Armbanduhren, Handys u.s.w.. integriert sein, so dass endlich gilt:

You can walk all day
without loosing your way...

[1] Als linguistische Hecke bezeichnet man die Präzisierung bzw. Abschwächung von räumlichen Relationen durch Wörter wie genau bzw. ungefähr (z.B. *das Regal steht genau vor der Tür*) (siehe [Kra98])

Anhang A

Anpassung des Programmes an eigene Szenarien

Dieser Abschnitt beschreibt, wie eigene Szenarien in dieses System eingebunden werden können. Üblicherweise erstellt man dazu zuerst ein 3D-Modell mit einem 3D-Modellierungsprogramm. Außerdem sollte man die in der Umgebung vorkommenden Klassen und ihre Beziehungen untereinander identifizieren. Anschließend erstellt man für jede dieser Klassen eine entsprechende Java-Klasse. Dabei können bereits im Verzeichnis *vr.objects* vordefinierte Klassen mitverwendet werden. Wichtig dabei ist, dass jede Klasse direkt oder indirekt von *VirtualObject* abgeleitet sein muss.

Anschließend muss das 3D-Modell auseinandergebaut werden, d.h. jede Geometrie, die einem Objekt zugeordnet wurde, sollte in eine eigene Datei gespeichert werden. Dabei empfiehlt es sich, möglichst viele Geometrien von mehreren Objekten gemeinsam zu verwenden. Eine solche gemeinsam verwendete 3D-Grafik muss nullpunktzentriert und achsenparallel positioniert und nicht für jedes Objekt, sondern nur einmal abgespeichert werden.

Nach Durchführung der obigen Schritte muss ein Objektlader definiert werden. Diese Klasse muss von *vr.ObjectLoader* abgeleitet sein und hat die Aufgabe die Welt zusammenzubauen und den Objekgraph zu konstruieren. Dies erfolgt in der Methode `construct`, die ein von `VirtualEnvironemnt` abgeleitetes Objekt zurückliefert, dass wiederum eine Referenz auf die Objektgraphwurzel im Attribut `m_toplevel` besitzen muss. Damit der Objektlader beim Programmstart automatisch selektiert werden kann, muss er sich im Verzeichnis `semantics/loaders` befinden.

Schließlich müssen Synonyme und Übersetzungen definiert werden (Dateien `semantics/resources/syn_<language>.properties`), eventuell die Worttypenliste ergänzt werden (Datei `semantics/resources/wordTypes_<language>.properties`) sowie Beispielsätze für die natürlichsprachliche Eingabe geschrieben werden (`semantics/resources/semantics.properties`).

Anhang B

Programmierschnittstelle

B.1 Laden/Entladen von Objekten

Eine einfache aber dafür auch recht spezielle Funktion zum Laden/Entladen von Objektgeometrien ist

```
void VirtualObject.load(boolean mode);
```

wobei `mode` auf *wahr* gesetzt wird, falls das entsprechende Objekt geladen werden soll, andernfalls auf *falsch*. Dieser Befehl lädt/entlädt sowohl die direkt mit diesem Objekt verbundenen Geometrien als auch alle Geometrien von seinen Nachkommen im Objektbaum (siehe Abschnitt 4.1.5).

Eine differenziertere Spezifizierung erlaubt die Verwendung des Befehls
`void VirtualObject.load(int mode)`.
`mode` kann dabei folgende Werte annehmen:

- `UNLOAD`, falls das Objekt entladen werden soll
- `ATTACH`. Soll ein Objekt geladen werden, bewirkt dieses Flag, dass die entsprechenden Geometrien auch in den Szenegraphen eingehängt werden. Andernfalls befindet sich das Objekt zwar im Speicher, ist aber nicht sichtbar. Wenn das entsprechende Objekt dagegen entladen werden soll, bewirkt das Setzen dieses Flags, dass die Geometrien nicht aus dem Speicher entfernt sondern lediglich aus dem Szenegraph ausgehängt werden.
- `LOAD_COMPONENTS`: Es werden auch alle Geometrien der Objekte geladen (bzw. entladen), die Bestandteil dieses Objektes sind (part-of-Beziehung). Beim Entladen ist zu beachten, dass eine Komponente nur aus dem 3D-Modell entfernt werden kann, wenn keines der Objekte, dessen Bestandteil sie ist, mehr geladen ist.
- `LOAD_INTERIOR`: Es werden auch alle Geometrien der Objekte geladen (bzw. entladen), die sich innerhalb dieses Objektes befinden (z.B. Möbel innerhalb von Räumen).

- `FORCE`: Besitzt nur beim Entladen eine Bedeutung. Wenn dieses Flag gesetzt wird, werden die entsprechenden Objektgeometrien auch entladen, wenn sie noch von anderen Objekten benötigt werden. Diese Funktionalität braucht man, wenn Objektgeometrien wärend der Laufzeit verändert wurden und neu eingeladen werden sollen. Dazu erzwingt man die Entfernung des alten 3D-Modells durch Angabe dieses Flags und lädt das entsprechende Objekt anschließend neu.

- `LOAD_NEIGHBOURS`: es werden mit diesem Raum direkt (ohne Tür) verbundene Räume mit geladen, sofern bei diesen das `autoload`-Flag gesetzt ist.

Das anfänglich beschriebene
`void VirtualObject.load(true)` entspricht dabei
`void VirtualObject.load(ATTACH|LOAD_COMPONENTS|LOAD_INTERIOR)` und
`void VirtualObject.load(false)` entspricht
`void VirtualObject.load(UNLOAD|ATTACH|LOAD_COMPONENTS|LOAD_INTERIOR)`.

B.2 Auffinden von Objekten

Allgemein können Objekte aufgefunden werden, in dem der ganze Objektgraph (angefangen vom toplevel-Objekt) abgesucht wird. Von Nutzen sind dabei die Methoden `getNumOfComponent()`, `getComponent(int)` und `getElementsInside()` jedes `VirtualObjects`, welche die folgenden Funktionen besitzen.

- `getNumOfComponents()` ermittelt die Anzahl der Komponenten, aus denen dieses Objekt zusammengesetzt ist.

- `getComponent(int i)` ermittelt die i.te Komponente.

- `getElementsInside()` liefert eine Aufzählung (`Enumeration`) aller Objekte zurück, die sich unmittelbar innerhalb dieses Objektes befinden.

Neben diesem sehr allgemeinen Verfahren existieren auch noch einfachere und effizientere Möglichkeiten, Objekte aufzuspüren. So kann jedes Objekt auch anhand seines Identifizierers ermittelt werden. Dieser bestimmt sich folgendermaßen:

- explizit gespeicherte ID, falls definiert

- sonst Objektnummer, falls definiert

- sonst Objektnamen, falls definiert

- sonst Objektklasse

Eine ID muss also nur angegeben werden, falls das Objekt anhand der übrigen Attribute nicht eindeutig identifiziert werden kann. Soll das Objekt über seinen Identifizierer in der Wissensbasis aufgefunden werden kann dafür die Methode

```
VirtualEnvironment.getAnyObject(String identifier)
```

benutzt werden. Ein performantere Möglichkeit bietet die Suche mittels

`VirtualEnvironment.getObject(String identifier).`

In diesem Fall wird das Objekt über eine Hashtabelle ermittelt. Allerdings sind darin nicht alle Objekte enthalten, sondern nur diejenigen, für die die Methode

`VirtualEnvironment.include(VirtualObject obj)`

wahr zurückgibt. Dadurch soll der Speicher- und Rechenaufwand begrenzt werden.
Eine Aufzählung aller Objekte dieser Hashtabelle erhält man durch

`VirtualEnvironment.getObjects().`

Die geometrische Beschreibung eines Objektes ermittelt man mit

`VisualObject.getBranchGroup().`

Umgekehrt erhält man bei gegebener `BranchGroup` die entsprechende semantische Repräsentation durch

`VirtualEnvironment.getObject(BranchGroup bg).`

Komplexere Suchmöglichkeiten können mittels Suchmustern definiert werden. Konstruieren kann man ein solches mittels:

```
public SearchPattern(String sname,
                     String classname,
                     String number,
                     String owner,
                     VDict dict,
                     boolean fuzzy)
```

Die Parameter bezeichnen dabei

- `sname`: den Namen des gewünschten Objektes, falls bekannt, sonst `null`
- `classname`: den englischen Klassennamen (z.B. Grocery Store) falls bekannt, sonst `null`
- `number`: die Nummer des Objektes, falls bekannt, sonst `null`
- `owner`: den Namen des Objektbesitzers, falls bekannt, sonst `null`
- `dict`: ein Wörterbuch, das benutzt wird, um den Besitzernamen in Titel, Vor- und Nachnamen zu zerlegen
- `fuzzy`: spezifiziert, ob eine Tippfehlerkorrektur für Namensvergleiche verwendet werden soll oder nicht

Nach einem Objekt, das mit einem gegebenen Suchmuster übereinstimmt, kann mit

```
public static void search(SearchPattern pattern,
                          VirtualEnvironment env,
                          Vector vec,
                          boolean onlyOne)
```

gefunden werden. Dabei spezifiziert

- `pattern`: das Suchmuster
- `env`: die 3D-Umgebung
- `vec`: den Vektor, in den diejenigen Objekte abgelegt werden, mit denen das Suchmuster identifiziert werden kann
- `onlyOne`: Falls dieses Flag gesetzt ist, wird der Suchvorgang abgebrochen, sobald ein passendes Element gefunden wurde

Eine Beispielanfrage nach allen in der Umgebung befindlichen Geschäften könnte folgendermaßen spezifiziert werden:

```
SearchPattern pattern=new SearchPattern(null,''Store'',
null,null,null,true);
Vector vec=new Vector();
Query.search(pattern,env,vec,true);
if (!vec.isEmpty()) System.out.println(vec.firstElement());
```

B.2.1 Natürlichsprachliche Suche

Um die Freitextsuche zu verwenden, muss zunächst die Wörterbuchklasse `VDict` im Paket `vjavalib.ling` aufgebaut werden. Dies erfolgt mit dem folgenden Konstruktor:

```
public VDict(Locale locale,String bundleName, String synName)
throws IOException
```

Die Parameter haben dabei folgende Bedeutung:

- `locale`: spezifiziert die zu benutzende Sprache, beispielsweise
  `locale.getDefault()` für die Standardsprache,
  `new Locale(''en'',''EN'')` für englisch,
  `new Locale(''de'',''DE'')` für deutsch.
- `bundleName`: Name des Worttypenlexikons (siehe Abschnitt 6.3). Hierin sind den verschiedenen Worttypen (Artikel, Substantiv, ...) die entsprechenden Vokabeln zugeordnet. Es handelt sich dabei um eine Java-Properties-Datei mit Namen <`bundleName`>_<`Sprache-Kürzel`>`.properties`. Die CLASSPATH-Umgebungsvariable sollte das Verzeichnis dieser Datei beinhalten. Das verwendete Sprachkürzel ist identisch mit demjenigen, das die zu verwendende Sprache bei der `java.util.Locale`-Klasse angibt. Eine solche Worttypendatei findet sich beispielsweise in `janis/semantics/resources/wordTypes.properties`.

- `synName`: Name des Synonymwörterbuchs (bzw. des Übersetzungswörterbuchs in die englische Sprache). Für die Bildung des Dateinamens gilt dasselbe wie für das Worttypenlexikon
  (siehe `semantics/resources/syn.properties` als Beispiel).

Für natürlichsprachliche Anfrage können die zwei Funktionen
`vr.locquery.Query.multiquery()` und `vr.locquery.Query.query()` verwendet werden. Bei `multiquery()` dürfen in einer Anfrage mehrere zu lokalisierende Objekte gleichzeitig angegeben werden, bei `query()` dagegen nur ein einziges.

Im Gegensatz zu `query()` werden die Ergebnisobjekte bei Verwendung von `multiquery()` nun in einem Feld von Vektoren abgelegt. Dabei enspricht jeder Feldeintrag einer unterschiedlichen Teilanfrage im Eingabestring. Sei die Anfrage beispielsweise: *Wo sind ein Supermarkt und die Post?*, so würde der erste Vektor alle gefundenen Supermärkte, der zweite die Postfilialen enthalten.

B.3 Berechnung räumlicher Relationen

Bei der Erzeugung eines `VisualObject`s können mittels der Klasse `CoordinatesFinder` [1] kleinste umfassende Quader (`ObliqueBox`) von Objekten berechnet werden. In dieser `ObliqueBox` finden sich auch die entsprechenden Methoden zur Berechnung räumlicher Relationen

Eine `ObliqueBox` entspricht also einer Idealisierung der entsprechenden Objektgeometrie. Sie kann mit der Funktion `PhysicalObject.getBox()` abgefragt werden. Alle Methoden dieser Klasse zur Berechnung von räumlichen Relationen gehen davon aus, dass das `this`-Objekt das Referenzobjekt ist und die Box (bzw. Mittelpunkt) des zu lokalisierenden Objektes als Parameter übergeben wird. Dabei können folgende räumliche Relationen berechnet werden:

- Distanzrelationen
- externe projektive Relationen
- interne projektive Relationen
- die Relation “auf”

Die folgenden Funktionen setzen meist voraus, dass das zu lokalisierende Objekt durch eine Punktapproximation repräsentiert ist. Sollte dieses Objekt dagegen durch einen Quader idealisiert worden sein, lässt sich die Punktapproximation ermitteln durch:

```
ObliqueBox box;
Point3d center=new Point3d();
box.getCenter(center);
box.transform(center);
```

[1] befindet sich im Paket vjavalib3d

B.3.1 Berechnung von Distanzrelationen

Die Berechnung von Distanzrelationen erfolgt dabei durch die Funktion

```
int dist=getQualifiedDistance(Point3d p3d);
```

Der zurückgegebene Wert entspricht einer der Konstanten
`DIRECTLY_AT,AT,NEAR,0` oder `FAR`. [2] Eine Stringrepräsentation erhält man mit der Funktion:

```
String getDistanceToString(distType);
```

B.3.2 Berechnung von externen projektiven Relationen

Prinzipiell ermittelt man die Anwendbarkeitsbereiche für externe projektive Relationen auf ähnliche Weise wie dies bei den Distanzrelationen der Fall war. Zuerst berechnet man wiederum den Mittelpunkt des zu lokalisierenden Objektes. Die Anwendbarkeit einer bestimmten Relation (z.B. *rechts hinten*) erhält man dann mit

```
double appl=getApplOfCombinedDir(p3d,
                                 viewerBox,
                ObliqueBox.RIGHT|ObliqueBox.BEHIND);
```

wobei `p3dLocBox` den begrenzenden Quader des zu lokalisierenden Objektes und `viewerBox` den begrenzenden Quader vom Betrachter bezeichnet. Dabei wird von dem Betrachterquader nur der Mittelpunkt und der Orientierungsvektor verwendet.

Die bestanzuwendene Relation liefert dagegen

```
int direction=getDirection(ObliqueBox p3dLocBox,
                           ObliqueBox viewerBox);
```

`direction` ist dabei eine ODER-Verknüpfung der Werte
`RIGHT`, `LEFT`, `BEHIND`, `IN_FRONT`, `ABOVE` und `BELOW`. Falls statt einer deiktischen eine intrinsische Orientierung verwendet werden soll, setzt man viewer auf `null`.

B.3.3 Berechnung von internen projektiven Relationen

Die Ermittlung der am besten anwendbaren Relation erfolgt einfach durch

```
int dir=getInternDirection(ObliqueBox pLocBox,
                           Vector3d forwards);
```

`forwards` bezeichnet die Richtung, in der vom Objektmittelpunkt aus gesehen die Objektfront liegt, normalerweise benutzt man hierfür einfach das Feld `m_forwards` von der `ObliqueBox` des Referenzobjektes. Der Parameter `dir` besteht aus einer logischen ODER-Verknüpfung der Werte
`ON_THE_FRONT,ON_THE_BACK,RIGHT,LEFT,ON_TOP, ON_THE_BOTTOM`.
Eine textuelle Darstellung des Richtungswertes erhält man durch

```
refBox.internDirectionToString(dir)
```

[2] zur Wahl der Intervalle siehe Ende Abschnitt 4.3.3

B.3.4 Berechnung von *auf*

Den Anwendungsgrad dafür, dass ein Objekt sich direkt *auf* einem anderen befindet, erhält man durch

```
double appl=refObj.directlyOn(ObliqueBox obj2);
```

Soll die Funktion dagegen transitiv interpretiert werden, muss auch das Objekt selber bekannt sein, nicht nur der begrenzende Quader.

Der zugehörige Funktionsaufruf, um die Anwendbarkeit dafür zu ermitteln, dass obj2 (transitiv) *auf* obj1 liegt, lautet in diesem Fall:

```
PhysicalObject obj1,obj2;
double appl=obj1.on(obj2);
```

Anhang C

Aufbau der Pakete

Insgesamt existieren vier Hauptpakete (siehe Abbildung C.1), diese heißen `vjavalib`, `vjavalib3d`, `vr` und `semantics`.

- `vjavalib` ist eine allgemeine, Java-Klassenbibliothek
- `vjavalib3d` ist eine auf Java3D basierende Klassenbibliothek
- `vr` (Virtual Reality) besteht aus einer Sammlung von Klassen, die für *Virtual Reality*-Anwendungen entworfen wurden. Es beinhaltet verschiedene Klassen, die Gegenstände der realen Welt modellieren (`VirtualObject`, `VisualObject`, `Door`, `Room`,...) sowie einige Behaviors (siehe Abschnitt 2.2.1), Objektannotationen und Tools.
- `semantics`: Dieses Paket besteht aus einer Beispielanwendung, den verschiedenen Ladern sowie der Schnittstelle zum JANIS-Programm.

Von diesen vier Paketen enthält `vjavalib` die allgemeinsten Funktionen. Etwas spezifischere Funktionalität besitzt `vjavalib3d`, gefolgt von `vr`. In `semantics` befinden sich dann die speziellsten Funktionen. Das heißt aber auch, dass es einer Funktion aus `vjavalib` untersagt ist, auf eine `vjavalib3d` oder `vr`-Klasse zuzugreifen. Umgekehrt darf eine `vr`-Funktionen dagegen `vjavalib3d` und `vjavalib`-Klassen verwenden.

C.1 `vjavalib`

Dieses Pakte unterteilt sich in die folgenden Unterpakete

- `base`: Basisfunktionalität wie String- und Array-Behandlung
- `awt`: Funktionen für das AWT (Abstract Windowing Toolkit). Das beinhaltet einige allgemein benutzbare Dialoge.
- `ling`: linguistische Funktionen und Klassen, die von der natürlichsprachlichen Eingabe verwendet werden. Die bestehen häuptsächlich aus:
  - der Wörterbuchmanagementklasse (`VDict`),

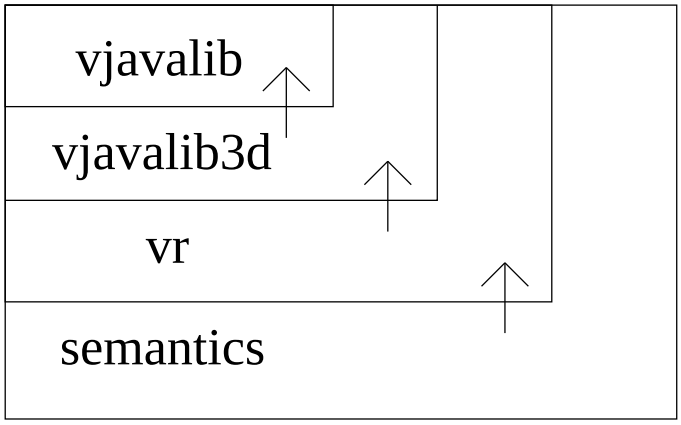

Abbildung C.1: Vierschicht-Architektur

  - dem Parser für Nominalphrasen (`NPParser`) und
  - einer Klasse zum Parsen und Vergleich von Eigennamen (`Name`).

- math: beinhaltet Klassen für Linien und Strecken, benutzt bei der Berechnung des begrenzenden Quaders von Räumen (siehe Abschnitt 4.4.1)

C.2 `vjavalib3d`

`vjavalib3d` ist eine auf Java3D basierende Klassenbibliothek. Sie beinhaltet die Quaderrepräsentation von Objekten (`ObliqueBox`) und enthält das Unterpaket `behaviors`, in dem sich verschiedene allgemeine Navigationshilfen sowie die Basisklasse für die Textannotationen befinden.

C.3 `vr`

`vr` steht für *Virtual Reality*. Darin befinden sich die Basisklassen für den Objektlader sowie für die virtuelle Umgebung (`vr.VirtualEnvironment`). Außerdem besteht es aus den Teilen

- `objects`
- `tools`
- `behaviors`
- `janis`

Im Unterpaket `objects` befinden sich Klassen für die Gegenstände, die Bestandteil der 3D-Umgebung sind, wie Räume, Türen, Möbel, Flure und Gebäude. Weiterhin enthält es

ein Wrapper-Objekt für den Avatar, womit dieser in die verwendete Ontologie eingeordnet werden kann.

`tool` beinhaltet einige Tools zur Unterstützung der 3D-Modellierung und der Verbindung von grafischer und semantischen Repräsentation.

Das Unterpaket `behaviors` enthält die Repräsentation des durch Picking realisierten Sensors des Avatars (`PickingBehavior`) sowie Klassen zum Öffnen und Schließen von Türen (`DoorOpener`, `DoorClosingBehavior`) und Anzeigen von Text-Annotationen (`ObjectAnnotation`).

Im Paket `janis` befinden sich Funktionen zur Integration mit dem JANIS-System von Marco Lohse (siehe [Loh01]). Zentrale Klassen sind dabei `SMetaNode` und `SCompactNode`, die intern aus `MetaNodes` und `CompactNodes` [1] bestehen, aber im Gegensatz zu diesen bei beliebig strukturierten Szenegraphen zum Einsatz kommen können.

C.4 semantics

Im Paket `semantics` befindet sich das in GECL enthaltene Objektlokalisationssystem sowie folgende Unterpakete

- `plugins`: Plugins für die Beispielapplikation
- `janis`: die Startklasse für die integrierte Projektapplikation RANA (`SJanis`)
- `loaders`: zu verwendende Lader (hier Flughafen und Lehrstuhl)
- `env`: zu verwendende Umgebungen

Ein Großteil der Funktionalität des dieser Arbeit zugrundeliegenden Systems wurde durch die Verwendung von Plugins [2] implementiert. So werden beispielsweise bei Klicks auf Buttons des Kontrollfensters (siehe Abbildung 5.8) ausnahmslos Funktionen aus Plugin-Klassen aufgerufen. Der vollständige Klassenname eines *Plugins* (einschließlich Paketbezeichnung) muss zu dessen Aktivierung in der Textdatei `semantics/resources/classlist1.txt` bzw. `semantics/resources/classlist2.txt` eingetragen werden. Ein Plugin muss das Interface `semantics.SemanticsPlugin` oder das allgemeinere `semantics.GeneralSemanticsPlugin` implementieren. Es hat die Möglichkeit auf dem Kontrollfenster einen Button zu erzeugen, mit dem es jederzeit aktiviert werden kann. Da bei der Erzeugung eines Plugins ein Objekt einer Klasse generiert wird, deren Namen erst zur Laufzeit feststeht, muss dazu das in Abschnitt 2.1.3 beschriebene Konzept zur dynamischen Instantiierung verwendet werden.

[1] Meta- und CompactNodes werden für die Abstraktion von Geometrien benötigt (siehe [Loh01])

[2] Als *Plugins* bezeichnet man Programmteile, die ohne Änderung am Programmcode der Anwendung dem System hinzugefügt oder aus ihm entfernt werden können.

Anhang D

The Geometry Reduction Tool (GRT)

Das Erstellen von 3D-Modellen ist sehr zeitintensiv. Daher gibt es einige Werkzeuge (z.B. 3D-Scanner), die diese automatisch erstellen. Allerdings können die damit erzeugten Modelle extrem groß werden, sodass man sie mit entsprechenden Werkzeugen vereinfachen muss. So betrug die Gesamtgröße der Geometrien, die wir vom Frankfurter Flughafen erhalten hatten 25 MB.

Grundsätzlich gibt es für die automatische Vereinfachung die Möglichkeiten benachbarte Dreieckspolygone zusammenzufassen oder auch kaum sichtbare Polygone wegzulassen. Die zweite Vorgehensweise verwendet auch *GRT*.

Dabei wird jedes Objekt von allen sechs Seiten (vorne, hinten, rechts, links, oben, unten) betrachtet und die entsprechenden Schnappschüsse abgespeichert. Anschließend entfernt man nacheinander jedes Polygon aus dem Objekt und vergleicht die sechs Seitenansichten des entstandenen Objektes mit denen vom Ursprungsobjekt. Falls die erzeugten Bilder sich von den Orginalen signifikant [1] unterscheiden, wird das Polygon wieder in die Geometrie eingefügt, andernfalls bleibt es entgültig draußen. Am Schluss enthält das Objekt also nur noch die relevanten Polygone.

Um die Ähnlichkeit von zwei Abbildungen zu bestimmen, addiert man die Differenz von jedem der drei Farbwerte (rot,gelb,blau) für jeden Pixel auf und dividiert sie durch die maximal mögliche Differenz. Dabei werden nur Pixel berücksichtigt, die innerhalb der das Objekt begrenzenden Kugel liegen.

Das Ergebnis des Vereinfachungsverfahrens ergab, dass tatsächlich um die 50% der Polygone eingespart werden konnten. Allerdings zeigte es sich auch, dass eine manuelle Bearbeitung noch weit bessere Resultate liefern konnte.

Verbessern könnte man das System dahingehend, dass bei der Berechnung von Farbdifferenzen diese nach ihrem Auffälligkeitsgrad gewichtet werden. So sind Unterschiede im Roten beispielsweise deutlich besser wahrzunehmen als im grünen Bereich (siehe [Bau96]).

[1] Die Ähnlichkeit, die zwischen dem vereinfachten und dem ursprünglichen 3D-Modell mindestens bestehen muss, kann vom Benutzer mit Werten von null bis eins spezifiziert werden

Anhang E

Mathematische Bezeichnungen

Im folgenden werden einige in dieser Arbeit benutzten mathematischen Bezeichnungen vorgestellt. Einige davon sind allgemein standardisiert, andere wiederum werden nur in dieser Arbeit verwendet.

$||\vec{a}||$: Euklidische Norm: $||\vec{a}|| = \sqrt{\vec{a}_x^2 + \vec{a}_y^2 + \vec{a}_z^2}$
$\vec{dist}(obj_1, obj_2)$: Distanzvektor zwischen zwei Objekten, die Distanz selber ergibt sich dann aus $||\vec{dist}||$.
$\vec{dist}_{scaled}(obj_1, obj_2)$: Mit der Ausdehnung von obj_1 skalierter Distanzvektor
$ext_i(obj)$: Ausdehnung eines Objektes in der entsprechenden Dimension i
ε: Die leere Zeichenkette
LO: Zu lokalisierendes Objekt
$\vec{Rel}$: Eine räumliche Relation definierender Richtungsvektor
RO: Referenzobjekt
$Rot_y(\alpha)$: Rotation um die y-Achse um den Winkel α
$S(obj)$: Schwerpunkt eines Objektes
$sign(i)$: Vorzeichen: $sign(0) = 0, sign(x) = 1 :\Leftrightarrow x > 0$,
$sign(x) = -1$ sonst
$Z(obj)$: Zentrum eines Objektes

Literaturverzeichnis

[ABGT85] E. André, G. Bosch, Herzog. G., und Rist. T. CITYTOUR - Ein natürlichsprachliches Anfragesystem zur Evaluierung räumlicher Präpositionen, Fortgeschrittenenpraktikum, 1985.

[All95] James Allen. *Natural Language Understanding.* The Benjamin/Cummings Publishing Company, 1995.

[Bar94] Hans-Jochen Bartsch. *Taschenbuch mathematischer Formeln.* Fachbuchverlag Leipzig, Köln, 1994.

[Bar01] Jon Barrilleaux. *3D User Interfaces with Java 3D.* Manning Publication Corporation, 2001.

[Bau96] Jörg Baus. Objektauswahl in 3D-Umgegungen basierend auf visueller Salienz als Wegbeschreibungsagenten. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 1996.

[BBK+00] Jörg Baus, C. Breihof, A. Krüger, Marco Lohse, und Andreas Butz. Some aspects of scouting smart environments. Technischer Bericht SS-00-04, Universität des Saarlandes, URL: http://w5.cs.uni-sb.de/ baus/publications/sg2000.ps.gz, 2000.

[Ben00] Maria Benning. Das intelligente Haus. *c't Magazin für Computertechnik*, 15:133–137, 2000.

[Bla01] Blackdown, URL: http://www.blackdown.org, 2001.

[Blo99] Anselm Blocher. *Ressourcenadaptierende Raumbeschreibung: Ein beschränkt-optimaler Lokalisationsagent.* Dissertation, Universität des Saarlandes, 1999.

[Boc86] Richard Bock. BUDA - Ein System zur Beantwortung unterspezifizierter Datenbankanfragen. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 1986.

[Bor94] Sven Bormann. *Virtuelle Realität.* Addison Wesley, 1994.

[Bou99] Dennis J. Bouvier. *Getting Started with the Java 3D API.* Sun Microsystems, URL: http://java.sun.com/products/java-media/3D/collateral, 1999.

[Bre01] Cyrille Breihof. Ein Autoanimierter Präsentationsagent für Ressourcenadaptierende Navigationssysteme. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 2001.

[Che77] P.P. Chen. The Entity-Relationship Model: Towards a unified view of data. *Transactions on Database Systems*, 1:9–36, 1977.

[CV01] Simon Clematide und Martin Volk. Linguistische und semantische Annotation eines Zeitungskorpus. In *Proceedings von der Jahrestagung der Gesellschaft für linguistische Datenverarbeitung*, Seite 202, 2001.

[Dav90] Earnest Davis. *Representation of Commonsense Knowledge.* Morgan Kaufmann Publishers, Inc., 1990.

[Deb01] Debian GNU/Linux, URL: http://www.debian.org, 2001.

[DH90] Sabine Dittrich und Theo Herrmann. „Der Dom steht hinter dem Fahrrad"-Intendiertes Objekt oder Relatum?, Arbeiten aus dem Sonderforschungsbereich 245, Sprechen und Sprachverstehen im sozialen Kontext. Technischer Bericht 16, Universität Mannheim, Lehrstuhl für Psychologie, 1990.

[DHR98] Patrick Doyle und Barbara Hayes-Roth. Agents in annotated worlds. In *Proceedings of the 2nd International Conference on Autonomous Agents*, Seiten 173–180, 1998.

[Dom99] Gitta Domik. Computergrafik, Gesamthochschule Paderborn, Vorlesungsskript, 1999.

[EPT96] Anne Engelhausen, Simone Pribbenow, und Ulf Töter. Multiple Part-Hierarchies. Technischer Bericht, Deutsches Forschungszentrum für künstliche Intelligenz (DFKI),
URL: http:/www.dfki.uni.sb.de/ hjb/WRKP-96/Pribbenow-E.html, 1996.

[Fai01] Faircar, URL: http://www.faircar.de, 2001.

[FvDFH90] J.D. Foley, A. van Dam, S. K. Feiner, und J.F. Hughes. *Computer Graphics. Prinicples and Practice.* Addision-Wesley, 1990.

[Gap93] Klaus-Peter Gapp. Berechnungsverfahren für räumliche Relationen in 3D-Szenen. Diplomarbeit, Unversität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 1993.

[Gap97] Klaus-Peter Gapp. *Objektlokalisation.* Deutscher Universitätsverlag, Dissertation, 1997.

[GFS+00] C. Gerber, P. Funk., M. Schillo, A. Burt, und C. Jung. SIF-VW: Eine integrierte Systemarchitektur für Agenten und Benutzer in virtuellen Welten. *Künstliche Intelligenz*, 2:12–15, 2000.

[Gra00] Joachim Grabowski. Ein psychologisch-onomasiologischer Ansatz zur Auffassung räumlicher Objektrelationen und ihrem sprachlichen Ausdruck: Inklusion und Kontakt. *Kognitionswissenschaft*, Seiten 63–76, 9 2000.

[Haa99] Stephanie W. Haas. Object oriented modeling tutorial components of an OO model, URL: http://ils.unc.edu/ stephani/safa99/oo-tut-components.html, 1999.

[Her86] Daniel Hernández. *Qualitative Representations of Spatial Knowledge*, Band 84 der *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1986.

[HHSR98] Knut Hartmann, Ralf Helbing, Thomas Strohotte, und Dietmar Rösner. Visdok: Ein Ansatz zur interaktiven Nutzung von technischer Dokumentation. Technischer Bericht, Universität Magdeburg, 1998.

[Hor01a] Helmut Horacek. Diskursplanung und Planrealisierung, 2001.

[Hor01b] Helmut Horacek. Textgenerierung und -zusammenfassung, Vorlesungsskript, 2001.

[Hot90] Günter Hotz. *Einführung in die Informatik*. B.G. Teubner Stuttgart, 1990.

[HRA90] Gerd Herzog, Thomas Riest, und Elizabeth André. Sprache und Raum, Natürlichsprachlicher Zugang zu visuellen Daten. In C. Freska und C. Habel, Herausgeber, *Repräsentation und Verarbeitung räumlichen Wissens*, Seite 212. Springer Verlag, 1990.

[IBM01] IBM. Voicetype, URL: http://www.ibm.com/voicetype, 2001.

[Kam93] Vera Kamp. Rämliches Schließen. SEKI working paper, Universität Kaiserslautern, 1993.

[KPS96] Herbert Klimant, Rudi Piotraschke, und Dagmar Schönfeld. *Informations- und Kodierungstheorie*. B.G. Teubner Verlagsgesellschaft, 1996.

[Kra98] Christian Kray. Ressourcenadaptierende Verfahren zur Präzisionsbewertung und zur Generierung von linguistischen Hecken, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster. Diplomarbeit, Unversität des Saarlandes, 1998.

[KRSD98] Jörg Kloss, Robert Rockwell, Karnél Szabó, und Martin Duchrow. *VRML 97 - Der neue Standard für interaktive 3D-Welten im World Wide Web*. Addision Wesley, 1998.

[Krü01] Antonio Krüger. *Automatische Abstraktion in 3D-Grafiken*. Nummer 232 in DISKI. infix, Dissertation, 2001.

[Lan01] Stefan Langer. Sprachen auf dem WWW. In *Proceedings von der Jahrestagung der Gesellschaft für linguistische Datenverwaltung*, Seiten 85–91, 2001.

[Loh01] Marco Lohse. Ein System zur Visualisierung von Navigationsauskünften auf stationären und mobilen Geräten. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 2001.

[LP92] Longan Latecki und Simone Pribbenow. On hybrid reasoning for processing spatial expressions. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI)*, Seiten 389–393, 1992.

[LR93] Longin Latecki und Ralf Rohrig. Orientation and qualitative angle for spatial reasoning. In *Proceedings of the 13th Inernational Joint Conference of Artificial Intelligence (IJCAI)*, Band 2, Seiten 1544–1549. Universität Hamburg, 1993.

[Maa96] Wolfgang Maas. *Von visuellen Daten zu inkrementellen Wegbeschreibungen in dreidimensionalen Umgebungen. Das Modell eines kognitiven Agenten.* Dissertation, Universität des Saarlandes, 1996.

[Mül88] Sabine Müller. CITYGUIDE. Diplomarbeit, Universität des Saarlandes, 1988.

[Ner95] Aaron Nerus. Principles of effective visual communication for graphical user interface design. In *Human-Computer Interaction*, Seite 429. Morgan Kaufmann Publishers, 1995.

[Neu] Günter Neumann. Methoden zur intelligenten Informationsextraktion im Internet, URL: http://www.dfki.de/ neumann.

[Neu01] Günter Neumann. Informationsextraktion,
URL: http://www.dfki.de/ neumann/ie.pdf, 2001.

[Obj01] Object Management Group. *UML Specification*, URL: www.omg.org, 2001.

[Pos01] Stefan Post. Ressourcenadaptive Wegplanung. Diplomarbeit, Unversität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 2001.

[RBL91] James Rumbaugh, Michael Blaha, und William Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[RD00] Ehud Reiter und Robert Dale. *Building Natural Language Generation Systems.* Camebridge University Press, 2000.

[REA01] REAL, http://w5.cs.uni-sb.de/real, 2001.

[RN95] Stuard Russel und Peter Norvig. *Artificial Intelligence - A Modern Approach.* Prentice Hall, 1995.

[Sch98] August Wilhelm Scheer. *ARIS - Vom Geschäftsprozeß zum Anwendungssystem.* Springer Verlag, 1998.

[SFGJ99] Michael Schillo, Petra Funk, Christian Gerber, und Christoph Jung. SIF - the Social Interaction Framework system description and user's guide to a multi-agent system testbed, DFKI research report. Technischer Bericht RR-99-02, Deutsches Forschungszentrum für künstliche Intelligenz (DFKI), URL: http://www.dfki.de/ sif/other_doc/users_guide.ps, 1999.

[SQL01] SQL Resource Page, URL: http://www.sql.org, 2001.

[Sun01a] Sun Microsystems. *Java 2 Platform Standard Edition Version 1.3, API Specification*, URL: http://java.sun.com/j2se/1.3/docs/api/index.html, 2001.

[Sun01b] Sun Microsystems. *Java Speech*,
URL: http://www.java.sun.com/products/java-media/speech, 2001.

[TGE01] E. Tamsen Taylor, Christina Gagné, und Roy Eagleson. Producing spatial descriptions: Effects of object familiarity. In *Proceedings of the 1st Conference on Smart Graphics*, Seite 69. ACM Press, 2001.

[VIT97] Connecting Vision und Natural Language Systems - SFB 314 Projekt VITRA, URL: http://www.dfki.uni-sb.de/vitra, 1997.

[VSF+97] Constanze Vorweg, Gudrun Socher, Thomas Fuhr, Gerhard Sagerer, und Gert Rickheit. Projective relations for 3D space: Computational model, application, and psychological evaluation. In *Proceedings of the Forteenth National Conference on Artificial Intelligence (AAAI)*, Seiten 159–162. Universität Hamburg, 1997.

[Wah90] Wolfgang Wahlster. Natürlichsprachliche Dialogsysteme, Vorlesungsskript, Universität des Saarlandes, Fachbereich für Informatik, 1990.

[Wah00a] Wolfgang Wahlster. Einführung in die Methoden der künstlichen Intelligenz, Vorlesungsskript, Universität des Saarlandes, Fachbereich für Informatik, 2000.

[Wah00b] Wolfgang Wahlster. Internet-Agenten, Vorlesungsskript, Universität des Saarlandes, Fachbereich für Informatik, 2000.

[WBKK01] W. Wahlster, J. Baus, C. Kray, und A. Krüger. REAL: Ein ressourcenadaptierendes mobiles Navigationssystem. *Informatik Forschung und Entwicklung*, URL: http://w5.cs.uni-sb.de/ baus/publications/ife_draft.ps.gz, 2001.

[Web01] X3D - Extensible 3D , New-Generation Open Web3D Standard, URL: http://www.web3d.org/x3d, 2001.

[Wei99] Gerhard Weikum. Datenbanksysteme, Vorlesungsskript, Universität des Saarlandes, Fachbereich für Informatik, 1999.

[Wex93] Alan Wexelblat. *Virtual Reality - Applications und Explorations.* Verlag Academic Press Professional, 1993.

[WT98] Wolfgang Wahlster und Werner Tack. Sonderforschungsbereich 378: Ressourcenadaptive Kognitive Prozesse, Bericht Nummer 143. In M. Jarke, Pasedach, und K. Pohl., Herausgeber, *Informatik'97 - Informatik als Innovationsmotor, 27. Jahrestagung der Gesellschaft für Informatik*, Seiten 51–57. Springer Verlag, URL: ftp://ftp.cs.uni-sb.de/pub/papers/SFB378/b143.ps.gz, 1998.

[Zer99] Klaus Zerbe. Virtuelle Welten. *c't - Magazin für Computertechnik*, 12:195, 1999.

[ZF93] Kai Zimmermann und Christian Freska. Qualitatives räumliches Schließen mit Wissen über Richtungen, Entfernungen und Pfade. *Künstliche Intelligenz*, 4:21–28, 1993.

[Zim93] Detlev Zimmermann. AnnA II: Ein wissensbasiertes System zur automatischen Annotation von Grafiken. Diplomarbeit, Universität des Saarlandes, Fachbereich für Informatik, bei Prof. Dr. Dr. hc. Wolfgang Wahlster, 1993.

Index