Interactive Systems Laboratory (ISL),
Fakultät für Informatik, Universität Karlsruhe (TH),
Postfach 6980, D-76128 Karlsruhe

Studienarbeit im Diplomstudiengang Informatik

Natürlichsprachliche Wegbeschreibungen für LingWear
Ein tragbares Touristeninformations- und -Navigationssystem

Stefan Jansen

30. April 2002

Betreuer
Dipl.-Inform. Christian Fügen
Prof. Dr.rer.nat. Alex Waibel

Inhaltsverzeichnis

1 Einleitung

1.1 Vorbemerkungen

Navigationssysteme sind ein Thema, das in den letzten Jahren zunehmend an Beachtung gewinnt. Das *Global Positioning System (GPS)* ist wohl jedem bekannt. Auch weiß heute jede, daß Flugzeuge, Schiffe und auch moderne Raketen des Militärs eigenständig ihr Ziel finden und nicht mehr auf die Steuerung durch den Benutzer angewiesen sind. Routenplaner mit Navigationshilfe halten jetzt auch schon in Wagen der oberen Mittelklasse Einzug und können auch schon als Nachrüstsatz erworben werden. Sogar die Mobilfunk-Provider bieten ortsabhängige Dienste an und geben Ausgeh-Tips, wenn auch die kleinen Displays und geringe Rechenleistung der Mobiltelefone hier enge Grenzen setzen. Warum also **noch** ein Navigationssystem?

Das in dieser Arbeit behandelte System ist für Touristen gedacht, die sich in erster Linie zu Fuß durch eine Stadt bewegen, um dort bestimmte Sehenswürdigkeiten zu besuchen oder ihr Hotel zu erreichen. Es stellt außerdem eine Reihe von Zusatzdiensten bereit, die über die reine Navigation weit hinausgehen. Dieser Anwendungsfall führt zu neuen Anforderungen an das System, die ich später beschreiben werde. So soll dieses System zum Begleiter werden, der neben der Wegbeschreibung auch weitere Informationen bereit hält, bei der Routenplanung hilft und sogar noch übersetzt, wenn man beispielsweise eine Postkarte kaufen will.

Als blinder Informatikstudent interessiert mich das Thema ‚Navigation für Fußgänger' so sehr, weil auch im Hilfsmittelbereich sehr stark an solchen Lösungen geforscht wird und auch schon erste Produkte erhältlich sind. Ich selbst habe es täglich mit Wegbeschreibungen zu tun. Wenn ein Blinder in eine neue Umgebung kommt, ist er darauf angewiesen, daß er eine möglichst detailgenaue Beschreibung bekommt, um ein bestimmtes Ziel selbständig zu erreichen. Ich werde in dieser Arbeit auch darauf eingehen, welche Orientierungspunkte sich zur Wegbeschreibung anbieten, wie eine Route in Abschnitte aufgeteilt werden kann oder welche Ordnungsstrategie bei der Beschreibung von Objekten eingesetzt werden kann.

1.2 Dank

Mein Dank gilt dem **ISL**, das mich bei dieser Studienarbeit sehr gut unterstützt hat, ebenso wie dem **Studienzentrum für Sehgeschädigte (SZS)** der Universität Karlsruhe, das mir an den dort eingerichteten speziell ausgerüsteten Computerarbeitsplätzen die Möglichkeit zur Programmierung und zum Test der Anwendung gab. Weiterhin danke ich all denen, die mich durch ihren Zuspruch in dieser Arbeit unterstützt haben. Ein besonderes Dankeschön gilt all denen, die sich die Zeit genommen haben, um mit mir über ihre Wünsche, Vorstellungen und Ideen zu **LingWear** zu diskutieren.

1.3 Zum Aufbau der Ausarbeitung

Der wohlgestaltete Entwurf eines Systems sollte sich im Idealfall nicht mit den Details der Implementierung befassen. Dennoch weiß man, daß im praktischen Einsatz eine solche Trennung nicht möglich ist, da die Implementierung sich oft in einer Rückkopplung auf den eigentlichen Entwurfsprozeß auswirkt. Im Falle dieser Arbeit geht die Verflechtung leider noch etwas weiter. Viele Ideen sind erst während der Implementierung entstanden, teilweise auch, weil manche Entscheidungen sehr weit nach hinten verschoben wurden oder manche Dinge nicht wie vorher beabsichtigt umgesetzt werden konnten. Dies lag beispielsweise daran, daß ein Großteil der Komponenten, mit denen mein kleines Navigationssystem zusammenarbeiten sollte, schon festgelegt waren. Mangels einer ausreichenden Dokumentation mußte viel experimentiert werden, was zur Revidierung getroffener Entscheidungen führte.

Ich werde in den folgenden Kapiteln auf einige grundlegende Aspekte eingehen, danach anhand einer Benutzerstudie die Problematik weiter erläutern und dann am Ende auf meine Lösungsansätze kommen. Dabei werde ich verschiedene Möglichkeiten ansprechen und jeweils darauf hinweisen, welche ich in der Implementierung umgesetzt habe. Manche Abschnitte werden aber einen speziellen Unterabschnitt enthalten, in dem ich mich nur auf die Implementierung des Teilthemas konzentriere. Dies geschieht dann, wenn das besprochene Problem weitgehend unabhängig von Implementierungsentscheidungen behandelt werden kann.

2 Problembeschreibung

2.1 Ziele dieser Arbeit

Ziel dieser Arbeit ist es, ein Navigationsmodul für **LingWear** zu entwickeln. Es soll, ausgehend von gegebenen Start- und Zielpunkten, eine Route ausfindig machen, diese in angemessene und sinnvolle Teilabschnitte zerlegen und dazu eine verbale Beschreibung erzeugen. Der Benutzer soll durch diese Beschreibung zum Ziel geführt werden — notfalls auch ohne eine Landkarte. Trotzdem sollen auch entsprechende Kartenausschnitte bereitgestellt werden.

Das Navigationsmodul arbeitet dazu einerseits mit einem Map Server und weiteren Datenbanken zusammen, die geographische Informationen und beispielsweise Zusatzinformationen über die Lage und Öffnungszeiten von Sehenswürdigkeiten bereitstellen. Andererseits gibt es seine Ausgaben an eine Dialogsteuerungseinheit weiter, die diese dann an eine Anzeigeeinheit oder eine Sprachsynthese weiterleitet und die Navigation steuert, indem es Anfragen des Benutzers an das Navigationsmodul weiterleitet.

Die Ausgaben des Moduls soll in englischer Sprache erfolgen, allerdings soll es leicht um andere Sprachen erweiterbar sein.

Die Implementierung soll in **Tcl/Tk** erfolgen, da fast alle Komponenten, die in **LingWear** eingesetzt werden, auch auf **Tcl/Tk** basieren und so eine leichte Integration möglich ist.

2.2 Anforderungen

Die hier genannten Anforderungen sind im Gegensatz zu den o.g. Zielen für die Funktion des Systems nicht notwendig. Sie erleichtern aber den Umgang mit der Software, steigern die Leistungsfähigkeit und erlauben die Erweiterbarkeit.

Die Einzelteile des Systems sollten, wenn möglich und sinnvoll, nur über enge Schnittstellen miteinander kommunizieren. Außerdem sollte jede Teilkomponente unabhängig vom Rest implementiert werden. Nur dadurch kann man leicht zu einem späteren Zeitpunkt einzelne Teile aus dem Gesamtsystem herauslösen und durch andere ersetzen. Sind die Einzelteile zu sehr miteinander verstrickt, so wirken sich auch kleine Änderungen schnell auf das Gesamtsystem aus.

Der hierdurch entstehende Vorteil ist auch, daß das Navigationsmodul leicht auf andere Komponenten eingestellt werden kann. Wird z.B. ein neuer Map Server gewünscht, dann muß lediglich die Schnittstelle zum Map Server neu implementiert werden. Der Rest des Systems ist davon unabhängig.

Eine Anforderung an ein Gesamtsystem wie **LingWear** ist eine gute Benutzerführung. Solche Systeme werden nur dann akzeptiert werden, wenn sie den Benutzer so wenig wie möglich belasten. Schließlich soll das Anwendungsgebiet u.a. der Tourismus sein. Touristen wollen sich nicht in ihrer Freizeit mit komplizierten Maschinen abgeben. Ein akzeptabler Preis für solche

Systeme sowie gute Mobilität und Flexibilität sind ebenso wichtige Aspekte, die allerdings im Rahmen dieser Arbeit nicht berücksichtigt werden, da es sich hier nur um ein Versuchsmodell handelt. In der Praxis wird man aber dem Benutzer niemals ein Set aus Laptop, GPS-Empfänger und -Antenne, Handy und einem kleinen Bildschirm zumuten können, das er sich erst umständlich anlegen muß, um es überhaupt bequem transportieren zu können. Vielmehr muß alles in einem handlichen Gerät verpackt sein.

Was die Navigation betrifft, so ist es nicht damit getan, den Benutzer von A nach B zu führen. Man muß ihm auch eine Hilfe an die Hand geben, damit er überhaupt weiß, was sich anzusehen lohnt. Desweiteren will ein Tourist bestimmt früher oder später wissen, wo er zu Mittag essen oder eine Postkarte kaufen kann. Auch auf diese Fragen sollte das System eine Antwort wissen, indem es eine Route zum nächsten Restaurant oder zu einem Andenkenladen oder Kiosk berechnet. Diese Aufgaben fallen zwar nicht direkt der Navigation zu und sind deshalb nicht Hauptthema dieser Arbeit. Allerdings sind sie sehr eng mit dem Navigationsmodul verbunden, wie wir später sehen werden.

Auch die Genauigkeit und Detailgenauigkeit des Kartenmaterials ist viel wichtiger als bei Navigationssystemen für Autos. Beim PKW-Verkehr reicht im einfachsten Fall eine reine Straßenkarte — angereichert um Hausnummern — aus. Ein Fußgänger ist aber viel flexibler. Er kann den Straßen folgen, kann aber auch viele kleine Schleichwege nutzen, Plätze beliebig überqueren, Unter- und Überführungen nutzen. Schließlich steht ihm natürlich auch der öffentliche Personennahverkehr zur Verfügung. All dies muß bei der Berechnung der Route berücksichtigt werden. Außerdem wird ein Fußgänger viel öfter die berechnete Route verlassen, weil er irgendwo etwas Interessantes entdeckt, als ein Autofahrer.

Hat man diese Ausführlichkeit der Karten erreicht, folgt fast unweigerlich der Wunsch, die Navigation auch ins Innere von Gebäuden zu bringen. So könnte das System den Museumsführer ersetzen oder zumindest als Ergänzung dienen.

2.3 Nützliche Zusatzfunktionalität

Zu einem System wie **LingWear** gehören auch Zusatzfunktionen, die nichts mit der Navigation zu tun haben. Einen wichtigen Teil dazu trägt die schon im System enthaltene Übersetzung bei, deren Möglichkeiten ich später genauer beschreiben werde. Weitere Zusatzfunktionen könnten z.B. eine Verbindung zum Internet über ein GSM- oder UMTS-Handy sein, um aktuelle Informationen wie Theaterprogramme oder Ausstellungspläne anzufordern. Außerdem könnte man damit beispielsweise auch einen E-Mail-Dienst anbieten. So wird das System nicht nur ein Navigationssystem, sondern der digitale Begleiter durch die Stadt, der mehr als ein Reiseführer ist.

Um dem Benutzer die Arbeit mit dem System zu erleichtern, muß dieses auch personalisierbar sein. So könnte ein Benutzer festlegen, daß er hauptsächlich an Kirchen und Schlössern interessiert ist und unterwegs möglichst wenig durch andere Dinge aufgehalten werden will. Ein anderer könnte festlegen, daß er auf jeden Fall alles Wichtige, was er auf seinem Weg passiert, auch erkunden will. Auch der Umfang der ausgegebenen Informationen könnte eingestellt werden.

Das Navigationsmodul ist so zu entwerfen, daß man diesen Aspekten Rechnung tragen kann und sie leicht integrieren kann. Deshalb muß es viele Einstellungsmöglichkeiten bieten und darf durch die eingesetzten Verfahren nicht schon ein starres Verhaltensmuster aufweisen, welches nicht von anderen Komponenten beeinflußbar ist.

3 Systemüberblick

In diesem Kapitel will ich einerseits die grobe Struktur und Implementierung von **LingWear** beschreiben, andererseits aber auch meine Vorstellungen von der logischen Struktur eines Navigationssystems darstellen. Dabei werden einige Komponenten nur vereinfacht dargestellt, da sie eher schwach mit der eigentlichen Navigationsproblematik verbunden sind. Andere Komponenten stelle ich nur als Teil der logischen Struktur vor, da sie bis jetzt nicht im vorgestellten System implementiert wurden.

3.1 Verteilungsaspekte, Dialog und Systemzustand

Gewiß wird man das in dieser Arbeit besprochene System nicht monolitisch, sondern in Komponenten realisieren. Dabei stellt sich zuerst die Frage, wie man das System verteilt. Die Verteilung bezieht sich sowohl auf die Steuerung als auch auf den allgemeinen Zustand des Systems. Zwar wird jede Komponente einen internen Zustand haben, den nur sie kennt und der für die anderen Systembestandteile ohne Bedeutung ist. Aber es gibt auch einen übergeordneten Systemzustand, der die Zusammenarbeit der Komponenten beeinflußt. Ist das Gesamtsystem z.B. im Zustand *Navigation*, so müssen Eingaben des Benutzers an das Navigationsmodul weitergeleitet werden. Ist das System beispielsweise hingegen im Zustand *Übersetzung*, dann haben die selben Benutzereingaben eine andere Wirkung, sie werden nämlich in eine andere Sprache übersetzt und ausgegeben. Systemzustand und Steuerung des Gesamtsystems sind eng miteinander verbunden.

Wenn es eine zentrale Steuerungseinheit gibt, die die Arbeitsweise des Systems im Groben steuert, dann muß diese natürlich auch den Systemzustand kennen. Dafür brauchen die anderen Komponenten sich nicht um den globalen Zustand zu kümmern, da sie sich darauf verlassen können, daß sie immer nur die für sie relevanten Informationen und Befehle erhalten. Bei einer Erweiterung des Systems muß außerdem nur die zentrale Steuerung angepaßt werden.

Verteilt man den Systemzustand über die Komponenten, dann ist jede Komponente selbst dafür verantwortlich, im Zusammenspiel aller Teile richtig zu agieren. Sie muß sich mit der Arbeitsweise und der Funktion der anderen Teile befassen. Allerdings kann sie u.U. auch flexibler reagieren, wenn sie den Gesamtzustand kennt. Erweiterungen wirken sich allerdings dann auf mehrere Komponenten aus, worin auch der große Nachteil dieses Ansatzes besteht.

Das vorgestellte System verfolgt den zentralen Ansatz, allerdings in einer Form, die jeder Komponente gestattet, Einsicht in den Gesamtablauf zu nehmen (vgl. 3.2 und 3.3.3.6). Das Navigationsmodul wurde so implementiert, daß nur die unbedingt notwendigen Informationen im Modulzustand abgelegt sind. Dies ermöglicht, Anfragen nicht nur in einem starren Muster, sondern flexibel zu stellen (vgl. 5.5).

Die Implementierung des Navigationsmoduls in dieser Art beeinflußt auch den Benutzerdialog. Starre Abläufe innerhalb des Moduls würden sich bis zur Benutzungsschnittstelle hin

durchschlagen und der Benutzer müßte sich genau an diese Abläufe halten. In unserem Fall kann er aber flexibel zwischen den einzelnen vom Navigationsmodul angebotenen Diensten wechseln. Dies ist vergleichbar mit dem Unterschied zwischen modalen und nichtmodalen Dialogen oder Fenstern bei einer graphischen Benutzungsoberfläche.

3.2 Komponenten und logische Struktur

Ich will nur näher auf die logische Struktur des Systems eingehen, wie sie sich aus dem Standpunkt eines Entwerfers darstellt, und kurz die Komponenten beschreiben, aus denen es besteht. Das Navigationsmodul wird später in Kapitel 5 ausführlich beschrieben. Am Ende dieses Abschnittes findet sich eine graphische Darstellung des Systems.

3.2.1 Dialogmanager

Die zentrale Steuerung regelt — wie in 3.1 beschrieben — den Gesamtablauf innerhalb des Systems. Alle Hauptmodule wie Navigationsmodul, Übersetzung, Sprachein- und Ausgabe werden über sie angesprochen und kommunizieren untereinander im allgemeinen nicht, da dies ein Seiteneffekt aus Sicht der Steuerungskomponente wäre. Direkte Kommunikation zwischen den anderen Komponenten ist nur dann erlaubt, wenn diese ohne Änderung des Zustands erfolgen kann. So können die in 3.2.6 behandelten Informationsmodule sich direkt gegenseitig nutzen, dürfen aber keine für andere Komponenten sichtbaren Änderungen in den direkt benutzten Modulen hinterlassen, die für die zentrale Steuerung wichtig sind. Bei späterer Nutzung eines so durch einen Seiteneffekt veränderter Moduls durch eine andere Komponente, die von der vorherigen direkten Kommunikation nichts weiß, kann leicht zu Fehlfunktionen führen.

In der Steuerungskomponente ist auch die Benutzerführung verankert. So muß hier entschieden werden, ob eine bestimmte Aktion des Benutzers momentan zulässig ist und Fehler müssen abgefangen und dem Benutzer gemeldet werden. Somit braucht diese Komponente Wissen über alle anderen Module des Systems und über die möglichen Ablaufszenarien. Sie stellt daher den *Dialogmanager* des Systems dar. Abschnitt 3.3.3.1 zeigt, wie die Steuerungskomponente hier implementiert wurde.

3.2.2 Eingabemethoden

Für die Interaktion mit dem System ist eine normale Computertastatur aus verständlichen Gründen unterwegs nicht einsetzbar. Es gibt zwar Spezialtastaturen [8], die klein sind und mit nur einer Hand bedient werden können, allerdings werden die Zeichen hier durch Tastenkombinationen erzeugt, die erst erlernt werden müssen. Bei einem Informationssystem für Touristen ist aber gerade eine einfache und intuitive Bedienung wichtig. Eine andere Möglichkeit ist eine Schrifterkennung, wie sie z.B. beim allseits bekannten **Palm Pilot** [5] und anderen PDAs zur Eingabe dient. Aber auch hier muß der Anwender eine spezielle Schrift erlernen. Die Erkennung von normaler Handschrift [6] ist zwar schon gut einsetzbar, benötigt aber genau wie die Spracherkennung mehr als die Rechenleistung eines einfache PDA und natürlich spezielle Hardware, ein Touchpad oder besser noch ein Touchscreen.

Das hier behandelte System nutzt Spracherkennung, kann aber durch den modularen Aufbau leicht um weitere Eingabemethoden erweitert werden.

Die Spracheingabe zerfällt in zwei Teile. Der Spracherkenner analysiert Audiodaten, die er entweder direkt vom Mikrofon oder aus einer Audiodatei einliest, und liefert als Ergebnis die Eingabe des Benutzers in Textform. Dazu muß der Erkenner auf die verwendete Sprache, den Wortschatz und die Eigenheiten des Sprechers eingestellt sein. In unserem Fall können nur sprecherunabhängige Systeme eingesetzt werden, die ohne Trainingszeiten auskommen.

Zwar ist der Wortschatz bei einem Navigationssystem, was die möglichen Befehle und Anfragen angeht, stark eingeschränkt, andererseits ist er auch sehr speziell, da er viele Orts-, Straßen- und Gebäudenamen enthält, die mit dem Einsatzort wechseln. Will man allerdings das System um Zusatzfunktionen wie E-Mail oder Übersetzung erweitern, kommt man in den Bereich der *Large Vocabulary Conversational Speech Recognition (LVCSR)*. Die Alternative wäre, für die reine Navigation auch einen eigenen Erkenner einzusetzen. Ebenso kann für jede Eingabesprache entweder ein eigener Erkenner laufen, oder man setzt einen Erkenner ein, der mehrere Sprachen gleichzeitig versteht, wie es in [7] beschrieben wird.

Der vom Spracherkenner gelieferte Text ist für die Verarbeitung durch den Dialogmanager und andere Komponenten nicht geeignet. Es handelt sich hier nämlich um natürliche Sprache. Man könnte zwar dem Benutzer auch feste Befehlsfolgen vorschreiben, allerdings leidet die intuitive Bedienung des Systems dadurch sehr und der Anwender müßte zuerst alle möglichen Befehle lernen. Man verlagert diese Arbeit nun auf den Parser, der die Eingabe anhand einer Grammatik in Teile zerlegt. Das Ergebnis ist die ursprüngliche Eingabe des Benutzers, jetzt allerdings in einer strukturierten Form, die sich an den Produktionsregeln der Grammatik orientiert. Ein Beispiel wird im Zusammenhang mit der von **Miso** verwendeten Typeed Feature Structure in 3.3.3.1 auf Seite 16 vorgestellt.

Die Spracheingabe kann auf zwei Arten betrieben werden: auf Benutzerwunsch oder im Dauerbetrieb. Hier kommt nur die erste Variante zum Einsatz. Diese ist für den Benutzer leichter verständlich, da er durch Knopfdruck genau festlegt, welche Eingaben für das System bestimmt sind. Dadurch vermeidet er Fehlinterpretationen von Nebengeräuschen und anderen parallel ablaufenden Dialogen. Diese Betriebsart ist auch einfacher zu realisieren. Die Spracheingabe bekommt hier nämlich voneinander abgesetzte Eingaben und muß sich nicht damit beschäftigen, aus einem kontinuierlichen Strom von an das System gerichteten Anfragen und anderen zufällig mit aufgenommenen Sätzen die passenden Teile herauszufiltern.

3.2.3 Ausgabemethoden

Die Ausgabe des Systems soll — wie auch die Eingabe — mehrere Sprachen unterstützen. Dazu ist nicht nur ein geeignetes Verfahren zur Konstruktion von Ausgabetexten nötig (vgl. 5.7.5), sondern es müssen auch für die einzelnen Sprachen Sprachsynthese-Programme verfügbar sein. Auch hier kann man Sprachsynthesen einsetzen, die mehrsprachig arbeiten können und die Sprache im laufenden Betrieb umschalten.

Bei einem portablen System ist zwar der Einsatz von Displays wegen ihrer Größe nur begrenzt möglich, allerdings hilft ein Kartenausschnitt bei der Navigation sehr viel weiter. Zusätzlich können Bilder und Videos von Sehenswürdigkeiten angezeigt werden. Das System sollte aber so gestaltet sein, daß sowohl Sprachausgabe als auch ein Bildschirm in Kombination oder alleine eingesetzt werden können. Die Ausgabeeinheit muß dann entsprechend der jeweiligen Konfiguration die Ausgaben des Systems an die passenden Geräte weiterleiten. Denkbar ist auch, gezielt Meldungen nur an die Sprachausgabe oder an den Bildschirm zu senden. So könnte man die ausführliche Beschreibung eines Gebäudes ansagen lassen, jedoch nur eine Kurzbeschreibung am Bildschirm zeigen.

Der Ausgabe durch synthetische Sprache ist noch eine Vorverarbeitung voranzustellen. Gerade Eigennamen, die natürlich nicht in die verschiedenen Ausgabesprachen übersetzt werden, können von Sprachausgabesoftware oft nur schlecht verarbeitet werden. So fehlen im Englischen die deutschen Umlaute, aber auch andere deutsche Begriffe werden im Englischen falsch betont oder sind nicht wiederzuerkennen. Deshalb muß dafür gesorgt werden, daß solche Wörter und Satzteile so in ihrer Schreibweise korrigiert werden, daß die Umsetzung in synthetische Sprache gelingt. Eine Alternative wäre, diese Satzteile speziell zu kennzeichnen und so der Sprachsynthese die Ausgangssprache mitzuteilen. Eine geeignete Markup-Sprache wäre z.B. *JSML* [9]. Aus eigener Erfahrung weiß ich allerdings, daß die Umschaltung der Ausgabesprache während eines Satzes oft zu Brüchen in der Satzmelodie und Betonung führt.

3.2.4 Navigationsmodul

Das Navigationsmodul, um das es in meiner Arbeit hauptsächlich geht, hat folgende Aufgaben, die später in eigenen Abschnitten genauer beschrieben werden sollen.

- Bestimmung einer Route zwischen einem gegebenen Start- und Zielpunkt oder von der aktuellen Position aus. Optional könnten Zwischenpunkte gegeben sein, die der Benutzer unbedingt, nur mit einer gegebenen Priorität, oder in einer bestimmten Reihenfolge passieren will. Außerdem sollten verschiedene Berechnungsverfahren angeboten werden. Manchmal wird eine möglichst schnelle, kurze oder nach den Vorstellungen des Anwenders interessante Route gewünscht sein. In dieser Arbeit werden diese optionalen aber trotzdem wünschenswerten Aspekte der Routenberechnung nicht berücksichtigt, da der Kartenserver **Rhea** diese nicht hinreichend unterstützt und sie für die Erzeugung von Wegbeschreibungen und für die eigentliche Führung des Anwenders zu seinem Ziel nur Beiwerk sind (5.6).

- Aufbereitung der Route. Die nackte Route, wie sie vom später angesprochenen Kartenserver geliefert wird, besteht gerade einmal aus einer Folge von Straßenabschnitten und Kreuzungen. Sie muß nun in für den Anwender sinnvoll zu überblickende Abschnitte zerlegt und mit Zusatzinformationen z.B. über Sehenswürdigkeiten angereichert werden, auf die der Anwender während seines Weges automatisch hingewiesen wird. Diese Aufgabe setzt wesentlich mehr Information voraus, als sie ein reiner Kartenserver liefert und ist daher auf weitere Datenquellen, wie sie kurz in Abschnitt 3.2.8 beschrieben werden, angewiesen. (5.6.3)

- Ausgabe der Beschreibung. Die Wegbeschreibung zerfällt in mehrere Teile. Erstens soll der Benutzer die Möglichkeit haben, sich die Route auf einem Display anzeigen zu lassen. Dazu sollen zu Beginn eine Gesamtansicht und während der Navigation weitere Ansichten der Route erzeugt werden, die den Fortschritt des Anwenders auf seinem Weg zum Ziel zeigen und seinen derzeitigen Standort angeben. Zweitens wird die verbale Beschreibung des gerade aktuellen Wegabschnittes erzeugt und ausgegeben. Diese kann in mehreren Sprachen angefordert werden (5.2, 5.7)

- Korrektur und Nachbesserung. Falls es dem Navigationsmodul möglich ist, die aktuelle Position des Benutzers zu ermitteln, dann sollte es den Benutzer auch dann unterstützen, wenn er die Route verläßt. Dies kann in Form von Hinweisen oder durch die Berechnung einer neuen Route geschehen. (3.2.8.3.5)

3.2.5 Tourmanager

Der Tourmanager soll den Anwender in der Suche nach einer geeigneten Route unterstützen und ihm den Zugang zum Navigationsmodul erleichtern. Dies kann dadurch geschehen, daß er einige Beispielrouten vom Navigationsmodul berechnen läßt und dem Benutzer zur Auswahl stellt, oder daß er einige vordefinierte Routen vorschlägt. Gerade in einem Einsatz-Szenario, in dem ein Tourist ein solches Navigationssystem z.B. an Bahn- oder Flughäfen ausleiht, kann es für ihn sehr hilfreich sein, wenn ihm der Tourmanager einige besonders interessante oder berühmte Dinge zeigen kann. Er kann somit einen Teil der Aufgabe eines Informationsprospekts übernehmen, den man sonst üblicherweise zur Hand nimmt, um sich einen ersten Überblick über die Möglichkeiten einer Stadt zu informieren.

3.2.6 Informationsmodul

Das Informationsmodul ist nicht viel mehr als ein Zugang zu einer Datenbank, in der ausführliche Informationen zu den verschiedensten Objekten abgelegt sind, die dem Anwender auf seinen Wegen mit dem Navigationssystem gegeben werden können. Es handelt sich hier nicht um Informationen, die für die Navigation wichtig sind. Vielmehr sind es Informationstexte, Bilder oder Videos zu Sehenswürdigkeiten, Speisekarten von Restaurants oder Programme von Theatern und Kinos. Sie können völlig frei genutzt werden, ohne in einem Zusammenhang mit anderen Modulen zu stehen und dienen somit als Nachschlagewerk und allgemeine Informationsquelle, die sowohl zu Hause als auch unterwegs immer dabei ist.

3.2.7 Zusatzmodule

Als Zusatzmodule bezeichne ich hier Dienste wie z.B. Schnittstellen zum Web oder E-Mail, die Übersetzung oder auch Module, die nur dazu dienen, das System zu konfigurieren oder im Testbetrieb Daten zu sammeln. Eines davon ist das *User Model*, in dem alle persönlichen Einstellungen des Anwenders abgelegt sind.

Andere Module protokollieren die Kommunikation zwischen den Modulen und ermöglichen so, Abläufe innerhalb des Systems zu reproduzieren, was für Tests und Fehlersuche unerläßlich ist, zumal es sich hier um ein System handelt, in dem mehrere Komponenten parallel laufen.

3.2.8 Datenquellen

Woher erhalten nun die soeben beschriebenen Komponenten ihre Daten? Diese Frage soll hier geklärt werden. Die hier vorgestellten Datenquellen sind nur als Anhaltspunkt zu sehen. Sollte eine Datenquelle in der Implementierung mehrere hier besprochene Quellen zusammenfassen, so ist dies natürlich kein Problem. Wir werden sowieso später sehen, daß sich die Implementierung von der hier vorgestellten logischen Struktur stark unterscheidet. Trotzdem hat die hier vorgestellte Struktur des Systems ihre Berechtigung, da sie als Grundlage für ein eventuelles Redesign dienen kann, sehr flexibel bezüglich Ersetzung und Austausch von Komponenten ist, sowie als Einstieg in die Arbeitsweise und das Verständnis des Systems dient.

3.2.8.1 Map Server

Der Map Server stellt eine elektronische Landkarte zur Verfügung und bildet die Grundlage, auf der sich die Navigation und viele andere Informationsquellen abstützen. Unerläßlich für die Navigation ist ein Koordinatensystem, in dem alle Positionen von Objekten auf der Karte oder in anderen Datenbeständen angegeben werden. Über den Map Server erhält das Navigationsmodul Informationen über Straßen und Kreuzungen, die Lage von Gebäuden und Plätzen, über Bus- oder Straßenbahnlinien und vieles mehr. Jeder weiß aus eigener Erfahrung, was man aus einer normalen gedruckten Landkarte alles ablesen kann. In diesem Umfang bewegen sich auch die Leistungen dieser Datenquelle.

Für die Navigation für Fußgänger braucht man allerdings sehr genaue Karten, die viel Detailinformation enthalten. Normale Autokarten sind hier wohl kaum zu gebrauchen, da ein Fußgänger wesentlich mehr Möglichkeiten zur Wahl seines Weges als ein Fahrzeug hat. So kann er einen für den Verkehr gesperrten Platz in fast jeder denkbaren Richtung überqueren, während ein Auto sich an dessen Begrenzungen orientieren muß.

In den letzten Jahren werden vermehrt detaillierte Karten angeboten. Allerdings reicht die dort enthaltene Information immer noch nicht aus, wie wir gleich sehen werden.

3.2.8.2 Sehenswürdigkeiten und Zusatzdaten

Informationen, die nicht durch den Map Server bereitgestellt werden, können in einer eigenen Datenbank abgelegt werden. Einige Beispiele sollen kurz angesprochen werden:

- Position und Bezeichnung von Objekten, die in der Karte nicht eingezeichnet sind. Dadurch kann unzureichendes Kartenmaterial nachgebessert werden.
- Öffnungszeiten von Gebäuden. So muß der Anwender, nachdem er endlich sein Ziel erreicht hat, nicht enttäuscht vor verschlossener Türe stehen.
- Informationen über öffentliche Verkehrsmittel.
- Alle Daten, die vom Informationsmodul benötigt werden (vgl. 3.2.6).

Diese Datenquellen sind passiv und sie liefern lediglich Information, nehmen aber im laufenden Betrieb keine auf, ganz im Gegensatz zu den in den nächsten beiden Abschnitten behandelten Datenquellen.

3.2.8.3 Positionsbestimmung

Für die Berechnung einer geeigneten Route ohne explizit gegebenen Startpunkt und für die rechtzeitige Ausgabe von Kartenausschnitten und Ansagen für den Benutzer braucht man die jeweils aktuelle Position. Ich will hier kurz darauf eingehen, mit welchen Mitteln diese bestimmt werden kann. Wir werden sehen, daß dies eine mehr oder weniger aufwendige Angelegenheit sein kann.

Die Position des Benutzers ändert sich laufend, weshalb diese nicht mit den eher statischen Daten aus den weiter oben beschriebenen Quellen vergleichbar ist, die über längere Zeiträume unverändert bleiben. Deshalb schlage ich für die aktuelle Position eine aktive Datenquelle vor, die selbständig ihre Information an das Navigationsmodul weitergeben kann, ohne von ihm jedesmal immer erst eine Anfrage erhalten zu müssen. Dadurch wird auch die Arbeit des Navigationsmoduls sehr vereinfacht, da es immer auf dem neuesten Informationsstand ist. Man könnte

sich vorstellen, daß die Positionsberechnung in zeitlich oder örtlich äquidistanten Abständen die aktuelle Position liefert oder daß das Navigationsmodul bestimmte Punkte vormerken kann, bei deren Erreichen ein Meldung abgesetzt wird.

Leider wurde in diesem Testsystem keine Positionsbestimmung implementiert. Dies hätte den Komfort des Systems wesentlich erhöht.

Zur Bestimmung der aktuellen Position und Richtung gibt es u.a. folgende Möglichkeiten:

3.2.8.3.1 Global Positioning System Das GPS-System bietet über Satelliten die Bestimmung von Position (in Länge und Breite), Höhe, Geschwindigkeit und Bewegungsrichtung. Leider ist über die Details von GPS nicht viel bekannt, da es sich ursprünglich um eine Entwicklung des US-Militärs handelt. Die Genauigkeit reicht bis zu sieben Metern und läßt sich mit lokal ausgestrahlten Korrektursignalen weiter verbessern (*Differencial GPS*). Die Firma **Garmin** [4] bietet Empfänger an, die nicht viel größer als ein Handy sind und über einen RS232-Anschluß verfügen.

Der Empfang von GPS ist zwar im Freien recht unproblematisch, allerdings kann es auch in eng bebauten Gebieten vorkommen, daß nicht genügend Satelliten erreichbar sind. In Gebäuden ist eher davon auszugehen, daß kein Empfang möglich ist.

3.2.8.3.2 Kompaß Die durch GPS ermittelte Bewegungsrichtung ist für unsere Zwecke nicht ausreichend, denn sie ist natürlich nur für bewegte Objekte sinnvoll. Um dem Benutzer aber die Richtung, in der er gehen soll, mit Kommandos wie ‚links' oder ‚rechts' angeben zu könne, braucht man die aktuelle Blickrichtung des Benutzers. Man könnte sich zwar auch an der Richtung des letzten Straßenabschnittes orientieren, allerdings könnte sich der Benutzer auch gedreht haben, um sich umzusehen.

3.2.8.3.3 Schrittzähler Bei Navigationssystemen für PKW wird die zurückgelegte Strecke leicht über den Kilometerzähler erfaßt. Sieht man einmal davon ab, daß der Kontakt zwischen Rädern und Boden bei eisglatter Fahrbahn oder Bodenwellen nicht optimal ist, so bleibt im übrigen eine recht genaue Erfassung möglich, sofern die technischen Daten wie z.B. der Raddurchmesser bekannt sind.

Bei Fußgängern hingegen gestaltet sich die Sache ungleich schwieriger. ein zum Kilometerzähler vergleichbares Meßgerät wäre ein Schrittzähler, wie es sie in Sportgeschäften gibt. Allerdings sind die Messungen recht ungenau, wie ich aus eigener Erfahrung weiß. Denn die Schrittlänge bleibt selten über längere Zeit konstant, besonders beim lässigen Bummeln durch die Stadt. Und diese Schrittlänge muß erst einmal ermittelt werden und sie ist natürlich Benutzerabhängig. Ein Navigationssystem, das seine Position mittels Schrittzähler bestimmt, muß also erst auf den Benutzer abgestimmt werden. Geht man aber davon aus, daß dies geschehen ist, dann stellen Schrittzähler und Kompaß zur Ermittlung von zurückgelegter Wegstrecke und Bewegungsrichtung eine Möglichkeit dar, auch bei Ausfall des GPS die Position näherungsweise zu bestimmen. Die Bewegungsdaten müssen dazu nur hinreichend oft mit der Karte abgeglichen werden.

3.2.8.3.4 Gegenseitiger Abgleich Kommen mehrere der o.g. Möglichkeiten zur Positionsbestimmung zum Einsatz, so werden deren Angaben alsbald voneinander abweichen. Es muß also allgemein eine Lösung gefunden werden, sowohl die Angaben des Benutzers über seine aktuelle Position, als auch die Angaben aus den verschiedenen Meßmethoden miteinander in

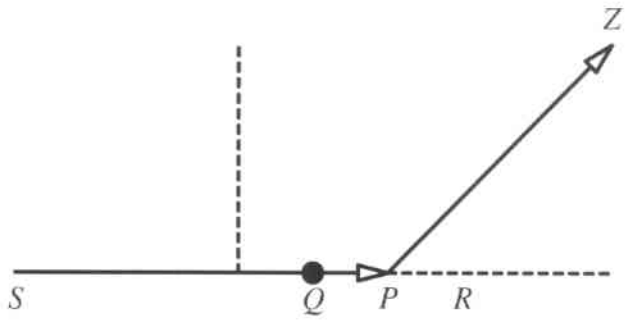


Abbildung 3.1: Abgleich der Positionsbestimmung mit der Karte

Verbindung zu bringt. Solche Methoden sind in **LingWear** ebenfalls nicht implementiert, da es ganz ohne Positionsbestimmung schwer ist, den Benutzer, der ja dann die einzige Quelle für die aktuelle Position ist, zu überprüfen. Dies könnte nur in einem längeren und ausgefeilten Dialog geschehen.

Das geeignete Mittel zur Synchronisation bildet die Landkarte. Dies soll an der Zusammenarbeit von Schrittzähler, Kompaß und Landkarte mittels Abbildung 3.1 erklärt werden.

Der Benutzer starte bei S und biege bei R nach Z ab. Die auf Schittzähler und Kompaß basierende Positionsbestimmung zeige aber als aktuelle Position den Punkt P an. Anhand der Karte kann das System feststellen, daß man bei P nicht abzweigen kann. Da Q und R gleich weit von P entfernt liegen, kann nur durch die vom Kompaß angezeigte Richtungsänderung festgestellt werden, daß der Benutzer bereits bei R ist. Der Schrittzähler wird nun korrigiert.

3.2.8.3.5 Einfluß auf die Navigation Der Einfluß einer genauen und hochverfügbaren Positionsbestimmung wird an den letzten Abschnitten deutlich. Je genauer die Positionsbestimmung arbeitet, desto komfortabler wird das Navigationsmodul. Es kann selbständig den Benutzer auf Fehler aufmerksam machen und ohne dessen Zutun zum richtigen Zeitpunkt Anweisungen und Erkäuterungen zu Sehenswürdigkeiten geben. Die Abschnitte, in die eine Route zu unterteilen ist, können deshalb länger ausfallen.

Natürlich sollte das Navigationsmodul aber auch eine Möglichkeit bieten, ohne die laufende Position auszukommen, falls diese doch einmal nicht verfügbar sein sollte. Auch wenn man die Position des Benutzers nur relativ messen kann, z.B. durch einen Schrittzähler, muß man dem Benutzer selbst die Verantwortung dafür geben, dem System seine genaue Position zumindest beim Systemstart mitzuteilen. Da in **LingWear** bis jetzt keinerlei Positionsbestimmung integriert ist, kann die Navigation nur erfolgen, indem das System einen Teil des Weges beschreibt und der Benutzer dann im Gegenzug dem System meldet, daß er der Beschreibung des Systems gefolgt ist und auf den nächsten Teil der Beschreibung wartet.

Fehlt jegliche Möglichkeit, die Bewegungen des Benutzers zu registrieren, so muß der Benutzer selbst dem System mitteilen, wann er einen Abschnitt der Route zurückgelegt hat. Diese Abschnitte müssen entsprechend kurz sein, denn das System kann nur an Haltepunkten auf wichtige Details der Route aufmerksam machen.

Anwender	**System**
Where can I eat something?	
	The nearest restaurant is called Oberländer Weinstube and can be found 550 meters to the west from here.
How can I get there?	
	I will give you guidance from here to Oberländer Weinstube. Use ‚continue' to get to the next instruction.

Abbildung 3.2: Beispiel zum Kontext bei Navigationsanfragen

3.2.8.4 Kontext

So wie die aktuelle Position immer als Bezugspunkt für die Navigation dient, so ist es auch wünschenswert, daß Eingaben des Benutzers immer im richtigen Kontext interpretiert werden. Dies ist nicht zu verwechseln mit dem globalen Zustand des Systems. Der Kontext beinhaltet vielmehr Informationen zu Dingen, auf die in der näheren zeitlichen oder räumlichen Umgebung Bezug genommen wurde.

Abbildung 3.2 zeigt einen Ausschnitt aus einem Dialog, in dem sich der Anwender erkundigt, wo er etwas essen könne. Das System sucht nun das nächste Restaurant in der Datenbank. Er nimmt den Vorschlag des Systems an und erfragt die Wegbeschreibung dorthin. Das System erkennt, daß sich die zweite Anfrage des Benutzers auf das gerade vorgeschlagene Restaurant bezieht und berechnet eine Route.

Die Kontextfunktionen sind noch nicht vollständig implementiert, das System erkennt nicht immer den Bezug zu vorher gestellten Anfragen bzw. deren Antworten.

Die in diesem und im vorherigen Abschnitt beschriebenen Datenquellen für Kontext und Positions- und Richtungsbestimmung müssen während der Laufzeit ständig aktualisiert werden, brauchen allerdings keine großen Datenmengen persistent zu halten. Dadurch unterscheiden sie sich von den übrigen beschriebenen Datenquellen.

3.2.9 Graphische Zusammenfassung

Abbildung 3.3 zeigt eine graphische Darstellung des auf den letzten Seiten entworfenen Systems. Dabei sind Module als Ellipsen und untergeordnete Komponenten und Datenquellen als Rechtecke dargestellt. Der zentrale Dialogmanager ist durch einen doppelten Rand gekennzeichnet. Optionale Komponenten und Verbindungen sind gestrichelt gezeichnet.

Die Beschriftung der Verbindungen zwischen den Komponenten soll Auskunft darüber geben, welche Art von Informationen ausgetauscht werden. Sie sind teilweise nur beispielhaft und könnten beliebig detailliert und erweitert werden. Darauf wurde aber aus Gründen der Übersichtlichkeit verzichtet.

Die Graphik ist in einigen Teilen stark vereinfacht. So kann das Übersetzungsmodul weiter in Teile zerlegt werden und wird eventuell auch z.B. den Parser der Spracheingabe nutzen. Die Zusatzmodule werden allerdings hier als *Black Box* betrachtet.

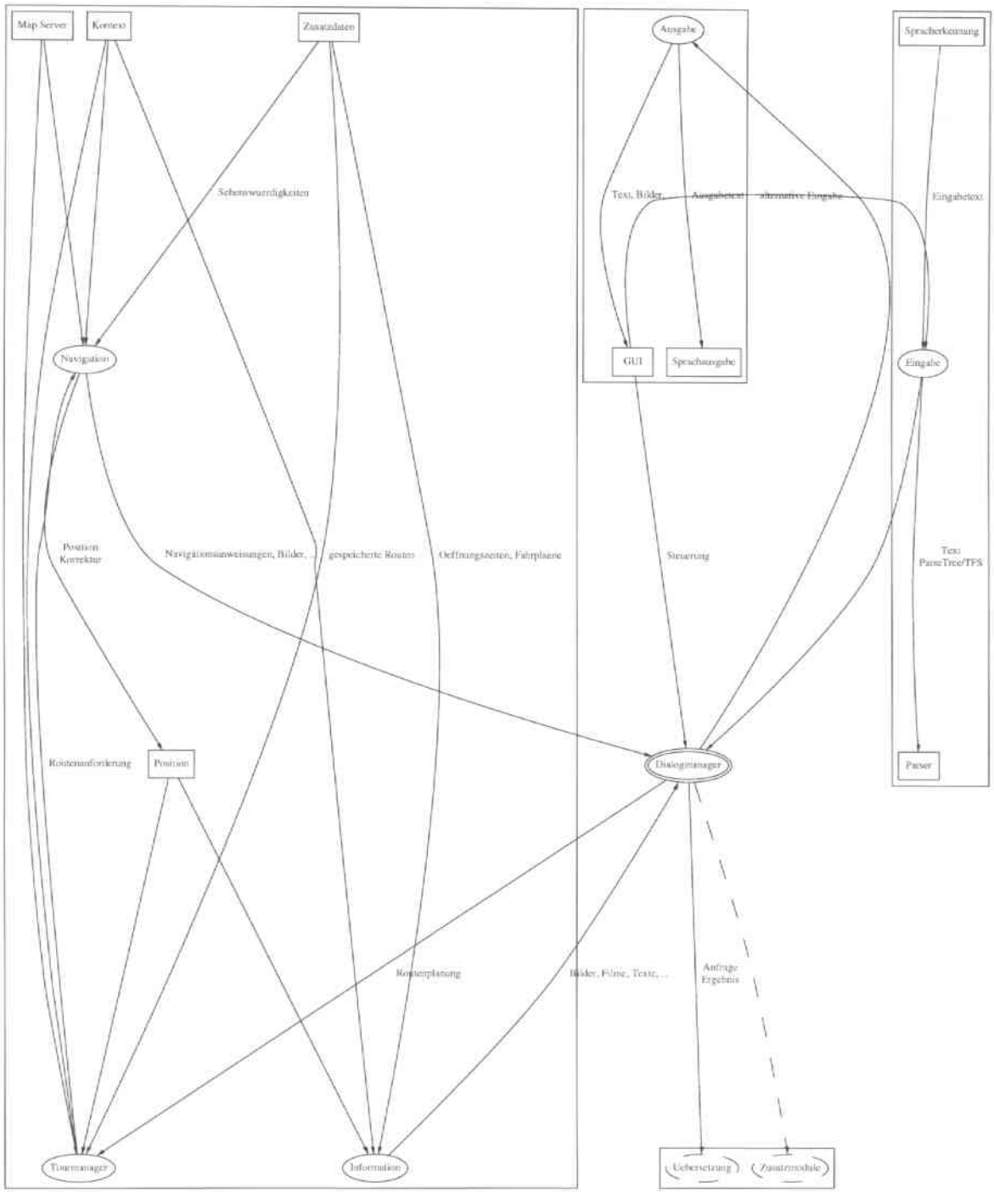


Abbildung 3.3: Logische Struktur eines Informations- und Navigationssystems für Touristen

3.3 Implementierung

3.3.1 Hardware und Betriebssystem

Wie schon in Abschnitt 2.2 gesagt, muß die Hardware im praktischen Einsatz leicht und klein sein. Beim Aufbau eines Testsystems wie im hier beschriebenen Fall liegen die Dinge allerdings

etwas anders. Hier zählt vorrangig die Verfügbarkeit, weshalb das System auf Standardhardware wie PCs oder Workstations implementiert wird. Man beachte, daß je nachdem, für welche Plattform man sich später jedoch entscheidet, sehr große Unterschiede in der Rechenleistung gegenüber der Entwicklungsplattform vorliegen können. Deshalb sind die hier vorgestellten Ansätze — gerade was den Bedarf an Speicher und Prozessorzeit angeht — nicht unbedingt auf ein späteres System übertragbar. Auf meiner PC liegt die Antwortzeit des Systems bei rund 15 Sekunden, allerdings auf einem *Pentium III* mit $800MHz$ unter *Linux 2.4.16* und $128MB$ RAM mit zusätzlichen $256MB$ Swap-Bereich. Ein nicht unerheblicher Teil dieser Zeit wird für die Spracherkennung einschließlich Parsing benötigt. Legt man nun z.B. einen Pocket-PC mit *Windows CE* zugrunde, dann wird klar, daß man hier schnell an Grenzen stößt, wenn man nicht für diese Geräte optimierte Softwarekomponenten verwendet.

3.3.2 Die Sprache Tcl/Tk

Ich will hier kurz auf **Tcl/Tk** eingehen, damit die später gezeigten Beispiele verständlich sind. Außerdem möchte ich darstellen, warum **Tcl/Tk** verwendet wurde.

Tcl/Tk, die *Tool Command Language* und ihr *Toolkit* sind unter *Unix* sehr verbreitet. **Tcl/Tk** ist eine interpretierte Sprache mit schwacher Typisierung, die ursprünglich als ein mächtiges Werkzeug für die Integration von kleineren Programmen diente. **Tcl/Tk** enthält deshalb Befehle für die Bearbeitung von Zeichenketten und Listen — einschließlich regulärer Ausdrücke. Im Grunde gibt es nur Variable für Felder und einfache Werte. Letztere sind Zeichenketten oder numerische Daten. Listen werden einfach als String dargestellt, die Elemente werden durch Whitespace getrennt. Obwohl dieses Typkonzept im Fehlerfall dem Programmierer Geduld und Konzentration abverlangt, ist es doch für den Anwendungsbereich der Sprache ideal, da man leicht Daten als Liste, Zeichenkette oder numerischen Wert interpretieren kann, so wie man es gerade braucht. Das Toolkit macht die Erstellung von graphischen Benutzungsoberflächen zum Kinderspiel. So kann man schnell aus den vielen kleinen Kommandozeilen-Werkzeugen, die *Unix* bietet, eine richtige Anwendung mit GUI bauen, in dem man geschickt verschiedene Programme kombiniert und mit **Tcl/Tk** einen Überbau darüber setzt. Ein weiteres Einsatzgebiet von **Tcl/Tk** ist die Integration in Anwendungsprogramme, um diese mit einer Skript- oder Makrosprache auszustatten. Zu diesem Zweck kann **Tcl/Tk** um eigene Funktionen z.B. in *C++* erweitert werden.

Die Erweiterbarkeit und die Einfachheit, mit der mehrere auf **Tcl/Tk** basierende Programme miteinander verbunden werde können, führte wohl dazu, daß die meisten eingesetzten Komponenten **Tcl/Tk** nutzten. Deshalb fiel die Entscheidung auch für das Navigationsmodul nicht schwer. Weitere Einzelheiten findet der Leser in Abschnitt 3.3.3 und 3.2. Wer selbst mehr zu **Tcl/Tk** erfahren möchte, wende sich an [3].

3.3.3 Komponenten von LingWear

Die aufgeführten Systemkomponenten könnten wohl jede für sich in eigenen Umgebungen implementiert werden. Allerdings stehen sich manche Komponenten sozusagen sehr nahe, da sie auf die selben Daten zugreifen müssen oder der Kommunikationsaufwand zwischen ihnen sehr hoch ist. Deshalb liegt es nahe, solche Komponenten zusammenzufassen und gemeinsam als Prozeß zu realisieren. Hier soll besprochen werden, wie die in 3.2 beschriebenen Komponenten nun tatsächlich zusammengefaßt wurden.

Abbildung 3.4: **LingWear** Display Manager nach dem Systemstart

Einige Details zu den eingesetzten Komponenten sowie weitere Literaturhinweise findet man in [10].

3.3.3.1 Ein- und Ausgabe

Die Ein- und Ausgabe wird in **LingWear** von mehreren Prozessen erledigt. Als erstes ist der Display Manager zu nennen, der — wie schon der Name sagt — das GUI des Systems verwaltet und in Abbildung 3.4 nach dem Systemstart gezeigt wird. Er veranlaßt auf Benutzerwunsch den Start der Spracherkennung. Der Benutzer kann über das GUI im Navigationsmodus auch die Beschreibung des nächsten Routenabschnitts anfordern und bekommt durch den Display Manager dann den Beschreibungstext sowie einen Kartenausschnitt auf den Bildschirm. Im Bild erkennt man gut die entsprechenden Buttons im GUI des Display Manager.

Die Sprachsynthese wird vom Display Manager über **Arti** versorgt. **Arti** nimmt einige Textersetzungen vor und leitet den Ausgabetext dann je nach Sprache an den passenden Synthesizer weiter.

Als Spracherkenner dient **Janus**. Er erkennt in **LingWear** ungefähr 5000 Wörter. Darin eingeschlossen ist bereits das Vokabular der Übersetzung.

Der erkannte Text wid von **NL**[1] an den auf dem Parser **SOUP** basierende **Miso** weitergeleitet, der daraus einen Parse Tree und schließlich eine *Typeed Feature Structure(TFS)* erstellt. Jede TFS hat einen Typ und davon abhängend verschiedene Merkmale, die entweder einfache Strings oder wieder eine TFS sein können. Die Typen sind in einer Typhierarchie organisiert, die flexibel je nach Anwendungsgebiet zusammengestellt werden können. In **LingWear** sind das z.B. Typen zum Bereich Navigation, Übersetzung und Steuerung des Systems (Umschalten

[1] Natural Language Processing

```
[ speechact
  PRED    pred_actionshowpathonly
  OBJECT1 [ obj_path
            DST [ obj_trainstation
                  QUANT quant_the
                ]
            SRC [ obj_castle
                  NAME "karlsruhe castle"
                ]
          ]
]
```

Abbildung 3.5: Beispiel einner TFS

zwischen Navigation und Übersetzung).

Als Parser kommt sowohl für die normalen Eingaben als auch für die Übersetzung **SOUP** zum Ensatz, der an der *CMU*[2] entwickelt wurde. Er verwendet modulare Grammatiken, die genau wie die typisierten Merkmalsstrukturen wegen ihrer Modularität leicht erweiterbar sind. So können je nach Betriebsmodus des Systems bestimmte Teile der Grammatik aktiviert oder abgeschaltet werden.

Abbildung 3.5 zeigt beispielhaft die TFS zur Anfrage

How can I get from Karlsruhe Castle to Karlsruhe Central Station?

Eine TFS hat gegenüber dem Parse Tree den Vorteil, daß für sie in **Tcl/Tk** eine Reihe von Hilfsmitteln existiert, mit denen man sie schnell und einfach auf ihren Typ und ihre Merkmale hin untersuchen kann.

3.3.3.2 Das Navigationsmodul Space

Das Modul **Space** faßt das Navigationsmodul, den Tourmanager, Zusatzdatenbanken und das Informationsmodul zusammen. Da diese Module, wie in Abbildung 3.3 gezeigt, viel miteinander kommunizieren müssen und im Wesentlichen auf die selben Daten zugreifen und einen gemeinsamen Kontext haben, ist es eine erhebliche Erleichterung, alles in einen Prozess zusammenzufassen. Somit übernimmt **Space** die in den Abschnitten 3.2.4, 3.2.5, 3.2.6, 3.2.8.2 und 3.2.8.4 beschriebenen Funktionen.

Abbildung 3.6 zeigt beispielhaft einige Anfragen, die von **Space** bearbeitet werden. Abbildung 3.7 zeigt **LingWear** im Navigationsmodus.

Der Tourmanager ist recht einfach gehalten. Er präsentiert dem Benutzer eine nach dessen Vorlieben geordnete Liste von Sehenswürdigkeiten in Abhängigkeit von der aktuellen Position. Man kann weder Routen individuell erstellen und dabei bestimmte Zwischenstationen einplanen, noch Routen zum späteren Gebrauch speichern. Abbildung 3.8 zeigt **LingWear** im Tourmodus.

Der Benutzer kann sich zu in der Datenbank abgelegten Objekten Detailinformation anzeigen und vorlesen lassen. Abbildung 3.9 zeigt dies. Die Zusatzdatenbank enthält momentan nur

[2] http://www.cmu.edu

- Where can I eat something?
- Where can I buy some cigarettes?
- How can I get to the station?
- How can I get THERE?
- Where is the next museum?
- What can I do here?
- Tell me more about the castle.

Abbildung 3.6: Von **Space** bearbeitete Anfragen

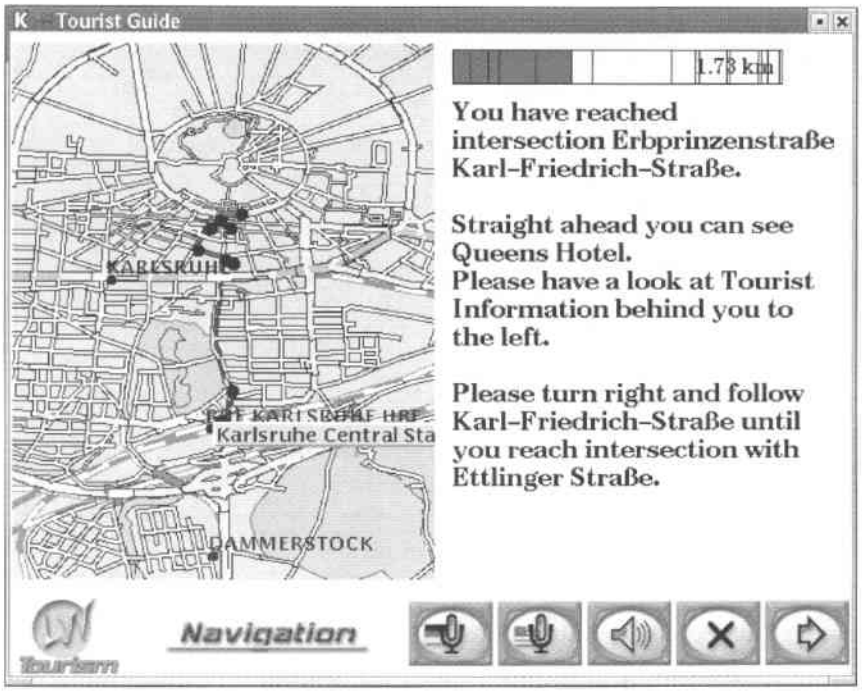


Abbildung 3.7: **LingWear** im Navigationsmodus

Sehenswürdigkeiten inklusive Öffnungszeiten von Museen oder anderen Gebäuden.Außerdem stehen in dieser Datenbank die Verweise auf die Informationstexte und Bilder.

Der Kontext einer Anfrage wird dadurch geschaffen, daß Angaben wie aktuelle Position oder das Ziel der Route generell allen Teilen von **Space** zur Verfügung stehen. Mit diesen Angaben und den in der Datenbank und der Karte abgelegten Informationen wird dann versucht, die Anfrage auszuwerten.

Die Landkarten werden vom Map Server **Rhea** verwaltet. Er liefert auch die Bilder mit den Kartenausschnitten für den Display Manager. Man kann auf ihnen zusätzliche Objekte einzeichnen und Beschriftungen anbringen. So werden Sehenswürdigkeiten und Start- und Zielpunkt der aktuellen Route markiert.

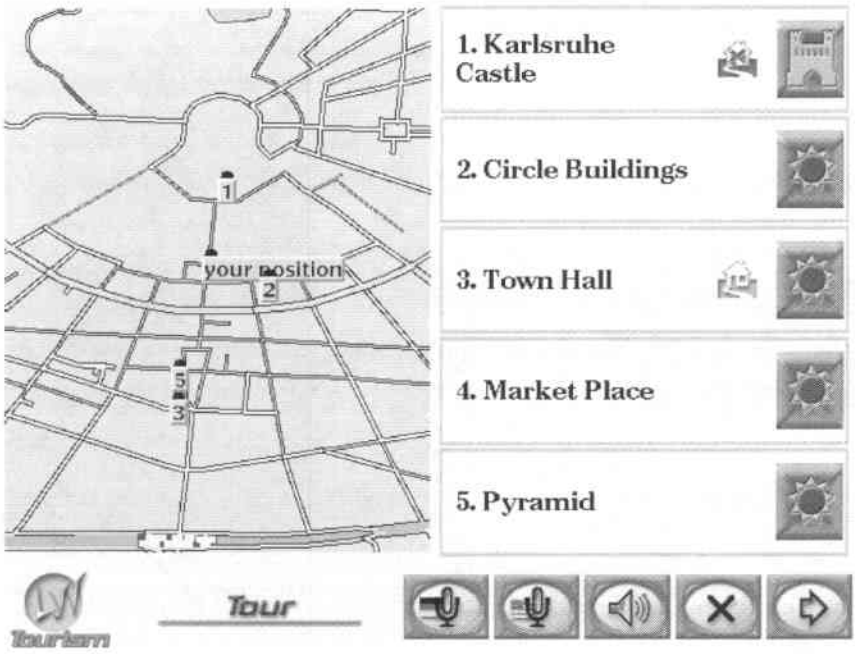


Abbildung 3.8: **LingWear** im Tourmodus

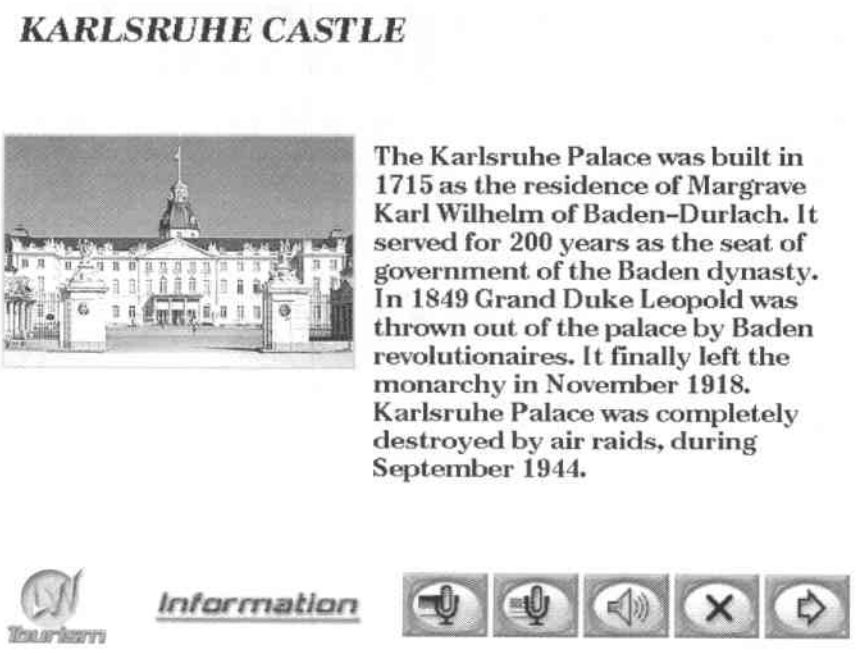


Abbildung 3.9: **LingWear** im Informationsmodus

3.3.3.3 Übersetzung

Die Übersetzung ist auf die Gebiete Reise/Tourismus und Arzt-Patienten-Dialog ausgelegt. Ein kann Tourist mit ihrer Hilfe Zimmer buchen, Postkarten kaufen und auch ohne Sprachproble-

Abbildung 3.10: **LingWear** Übersetzung — Reise/Touristik

Abbildung 3.11: **LingWear** Übersetzung — Arzt-Patienten-Dialog

me einen Arzt aufsuchen. Als Parser enthält sie ebenfalls **SOUP** und kann damit als *Black Box* betrachtet werden, die einen einfachen String als Eingabe erhält und diesen in den gewünsch-

ten Zielsprachen ausgibt. Intern arbeitet sie mit einer Zwischensprache (**Interlingua**), in die alle Eingaben übersetzt werden. Diese dient dann als Quelle für die Übersetzung in die Zielsprache. Dieser Ansatz gestattet es, das System um weitere Sprachen zu ergänzen, indem man genau zwei Übersetzungsprobleme löst: Nämlich von der einzubauenden Sprache nach **Interlingua** und umgekehrt. Die Zwischensprache muß die Semantik des zu übersetzenden Textes enthalten und darstellen. Deshalb ist sie domänenabhängig und man muß sie für jedes sachliche Teilgebiet erweitern. Momentan sind in **LingWear** Übersetzungen aus den Bereichen *Reise, Reisebuchung, Hotel* und *Arztbesuch* möglich. Abbildung 3.10 und 3.11 zeigen das System im Übersetzungsmodus.

3.3.3.4 Weitere Komponenten

LingWear enthält zusätzlich noch folgende Komponenten:

UserModel	legt Preferenzen von Benutzern ab
DataCollect	sammelt Daten über die Spracherkennung und Benutzer
History	Protokollierung und Simulation von Szenarien

3.3.3.5 Zentrale Steuerung

Die Forderung nach einer zentralen Steuerung des Systems durch einen Dialogmanager wird in **LingWear** etwas aufgeweicht. Der Display Manager übernimmt zwar teilweise die Weiterleitung der Nachrichten von **Space** an **Arti** und startet auch die Spracherkennung, allerdings liegt die Anwendungslogik großteils in **NL**. Jede Anfrage an das System passiert **NL** wird dort analysiert und an die richtigen Komponenten geleitet. Somit kommt **NL** die Rolle des Dialogmanagers zu.

Allerdings stimmen sich **NL** und der Display Manager eng miteinander über den momentanen Betriebsmodus ab, sodaß immer die passenden Grammatiken für den Parser, die richtigen Einstellungen für den Erkenner und das passende Modul des GUI aktiv sind.

3.3.3.6 Kommunikation

Die Mehrzahl der Komponenten kommuniziert nicht direkt miteinander. Es gibt einen zentralen **CommServer**, bei dem man sich über ein Socket unter einer freien ID anmelden kann. Danach können über diesen Server Nachrichten an andere Teilnehmer verschickt werden, wobei die IDs als Adressen benutzt werden. Allerdings werden weitaus häufiger nicht die IDs der Komponenten, sondern Namen von Kommunikationsgruppen gewählt. Diese entstehen, indem sich Komponenten unter mehreren IDs anmelden. So kann sich jede Komponente für die Nachrichten anmelden, die sie braucht. Fehlermeldungen innerhalb des Systems können so leicht an alle verteilt werden, ohne daß die Nachrichtenquelle die Empfänger einzeln kennen muß. Ich werde später im Abschnitt 5.2 zeigen, welches Format die über den **CommServer** verschickten Nachrichten haben.

Diese Art von Kommunikation hat mehrere Vorteile:

- Sie ist durch die Gruppenbildung flexibler als direkte Kommunikation.
- Jeder bekommt nur die für ihn interessanten Nachrichten und muß die anderen nicht analysieren.

- Die Kommuniktion läuft zentral an einem Punkt zusammen und kann dort bei Bedarf in jeder Form verändert werden.
- Es ist leicht möglich, an dieser Stelle eine Komponente zur Erstellung eines Protokolls anzubinden. Sie muß sich nur für alle Gruppen anmelden.
- Durch gezieltes Einspeisen von Nachrichten an dieser Stelle kann ein Szenario reproduziert werden.

Einige Komponenten kommunizieren direkt miteinander, ohne den **CommServer** zu benutzen. Dies sind insbesondere **Space** und der Map Server **Rhea**, sowie **NL** und **Miso** bzw. **NL NL** und die Übersetzung. In beiden Fälen läuft die Kommunikation über besondere Schnittstellen ab. Da der Parser aber nur von **NL** und der Map Server nur von **Space** benutzt werden, spielt dies für das Gesamtsystem keine Rolle.

3.3.3.7 Graphische Darstellung

Abbildung 3.12 stellt **LingWear** in seiner Implementierung graphisch dar. Kompnenten, die als *Black Box* betrachtet werden, werden als Rechtecke gezeichnet, die anderen als Ellipsen. Beziehungen zwischen den Komponenten werden durch Linien oder Pfeile repräsentiert. Dabei geben gestrichelte Verbindungen an, daß der **CommServer** zur Kommunikation benutzt wird. Durchgehende Linien stehen für direkte Kommunikation. **NL** und Display Manager sind durch einen doppelten Rand ausgezeichnet, um auf ihre besondere Rolle bei der Steuerung des Systems aufmerksam zu machen.

Da die Kommunikation über den **CommServer** in der Implementierung in allen Fällen über Kommunikationsgruppen abgewickelt wird — auch wenn diese dann nur aus genau einem Mitglied bestehen — müssen sich die durch gestrichelte Linien verbundene Kommunikationspartner nicht unbedingt kennen. So kommunizieren fast alle Komponenten mit **History**, ohne daß dies bei der Implementierung extra berücksichtigt worden wäre.

3.4 Bewertung

Die gewählte Implementierung macht das System sehr flexibel. Es kann problemlos durch neue Komponenten erweitert werden, was vor allem durch die Kommunikationsinfrastruktur des **CommServer** möglich wird. Allerdings ist der Nachrichtenfluß zwischen den Komponenten nicht unbedingt leicht zu durchschauen, da sich im Prinzip jede Komponente für jede Kommunikationsgruppe anmelden darf. Außerdem muß eine Komponente, die eine Nachricht an eine bestimmte Gruppe sendet, nicht selbst Mitglied in dieser Gruppe sein. Es bleibt nur, den Quellcode genau zu studieren oder sich die Log-Dateien des **CommServer** anzusehen. Es ist sogar möglich, mehrere Instanzen einer Komponente zu starten, was natürlich oft zu unerwartetem Systemverhalten führt.

Mein Entwurf aus Abschnitt 3.2 versucht, dies zu verhindern, indem Komponenten nur mit dem Dialogmanager kommunizieren. Dieser muß dann feststellen, ob die Nachricht auswirkungen auf andere Komponenten hat und diese dann informieren. Der Nachteil besteht dabei darin, daß eine Komponenten nur nach Anpassung des Dialogmanagers über zusätzliche Ereignisse im System informiert wird. Bei Erweiterungen muß der Dialogmanager immer mit angepaßt werden. Bei guter Programmiertechnik sollte dies aber auch nicht viel Arbeit erfordern.

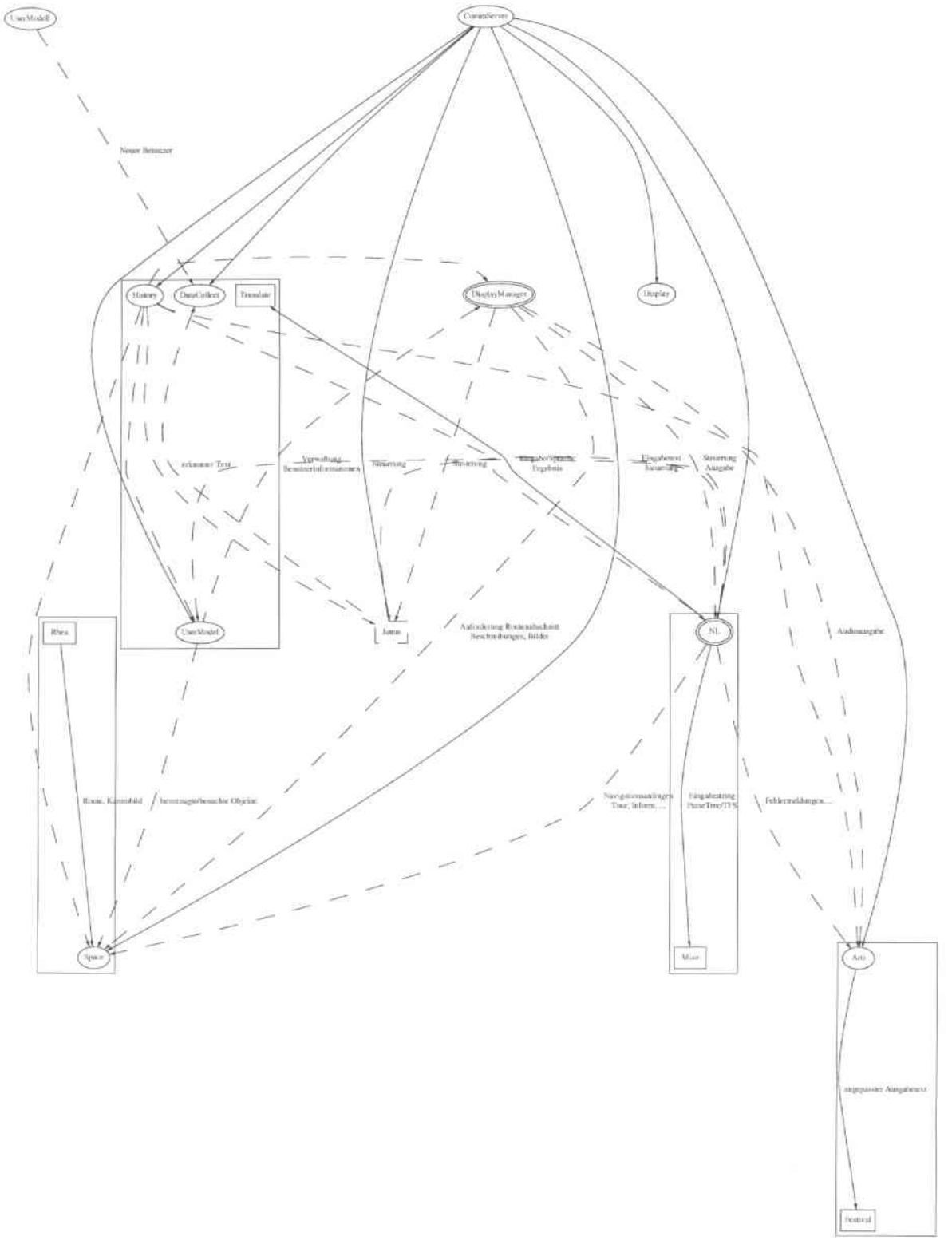


Abbildung 3.12: Implementierung von **LingWear**

Die eingesetzte Software ist nicht nur auf Unix-basierten Systemen lauffähig. Es gibt auch Implementierungen von **Tcl/Tk** für Windows-Systeme. Trotzdem ist es wünschenswert, zumindest bei den Datenquellen (vgl. 3.2.8) auf kommerzielle Produkte umzusteigen. So kann z.B. viel Arbeit eingespart werden, die zur Pflege des Kartenmaterials des Map Server und zum Eintragen weiterer Sehenswürdigkeiten, Restaurants, Hotels etc. eingesetzt werden muß.

4 Benutzerwünsche und Ansatzpunkte

Immer, wenn man ein neuartiges System entwirft, weiterentwickelt oder am Ende auf den Markt bringt, ist es sehr wichtig, die Wünsche und Anforderungen der (späteren) Benutzer mit einzubeziehen. Ich habe deshalb versucht, diese in einer kleinen Umfrage zu ermitteln.

Diese Umfrage ist nicht repräsentativ, da ich nur etwa zwanzig Personen befragt habe. Außerdem folgten nicht alle Gespräche demselben Muster, da ich mich nicht an ein festes Schema halten konnte. Trotzdem glaube ich, einige interessante Aspekte der Thematik darstellen zu können.

4.1 Vorgehensweise und Fragestellung

Ursprünglich hatte ich vor, einen Fragebogen zu erstellen und diesen von den Befragten bearbeiten zu lassen. Bald merkte ich aber, daß es sich bei den Fragestellungen, die mich interessierten, nicht um einfache ‚Ja-Nein-Fragen' handelte. Sie bedurften vielmehr längerer Antworten, oft mit zusätzlichen Erläuterungen und Rückfragen. Zudem war ich mit einem Problem konfrontiert, das bei neuen Produkten oder Systemen immer auftritt. Die Befragten konnten sich zunächst einmal unter einem Informations- und Navigationssystem für Touristen nicht viel vorstellen. Sie dachten sofort an Navigationssysteme für PKW und fragten dann zurecht, was man denn da noch zu entwickeln habe.

Also mußte ich meine Vorgehensweise umstellen. Nun benutzte ich meine Fragen als Grundlage für ein Gespräch — in den meisten Fällen gleich mit mehreren Teilnehmern — und versuchte durch eine Diskussion des Themas meine Antworten zu bekommen. Außerdem erklärte ich ihnen Stück für Stück **LingWear**. Waren die Befragten erst einmal über das Konzept des Systems informiert, wurden auch schnell Vorschläge und Ideen eingebracht.

Mich interessierten folgende Punkte:

- Alter, Ausbildung, Beruf, Umfeld der befragten Person.
- Wieviele Reisen, Art und Dauer der Reisen.
- Woher werden Informationen zum Reiseziel beschafft? Woher bekommt man Informationen vor Ort? Wie lange dauert die Planung?
- Das Vorgehen bei Ausflügen, insbesondere bei Stadterkundungen.
- Die Bereitschaft, einen Computer oder ein elektronisches Gerät unterwegs mitzunehmen. Welcher Mehrwert müßte damit verbunden sein? Wie groß und schwer dürfte es sein? Wie soll es bedient werden?
- Welche Funktionen soll ein Informations- und Navigationssystem bieten?

- Worin liegen die Probleme bei Wegbeschreibungen?
- Eine Wegbeschreibung des Befragten auf der Karte oder draußen.

4.2 Zielgruppen

Aus der Gruppe der **LingWear**-Benutzer will ich drei Teilmengen herausgreifen.

- Der Geschäftsreisende
  Er hat meist nur wenig privates Interesse an seinen Reisen, die eher kurz ausfallen. Dafür reist er häufig und in die verschiedensten Gegenden, ohne immer Einfluß auf die Wahl des Zieles zu haben. Zur Planung von Besichtigungen bleibt oft wenig Zeit und diese Aktivitäten sind mit einem vorgegebenen Programm abzustimmen.

- Der Wochenendtourist
  Der Wochenendtourist sucht sich seine Ziele selbst aus, ist allerdings bei seiner Wahl teilweise sehr spontan. Dies trifft vor allem auf jüngere Leute zu, die mal eben kurz einen Bekannten in einer anderen Stadt besuchen, oder einen Kurz-Trip machen, um vielleicht ein Konzert zu besuchen. Ziel der Reise sind hauptsächlich Freizeitaktivitäten.

- Der Urlauber
  Der Urlauber verbringt mehrere Tage entweder an einem festen Ort oder zieht im Laufe seiner Urlaubszeit von Ort zu Ort weiter. Er nimmt sich in der Regel mehr Zeit zur Planung als die anderen beiden Typen von Touristen und verbringt wesentlich mehr Zeit mit Informationsbeschaffung vor Ort. Seine Reise organisiert er wie die anderen beiden Teilgruppen selbst, oder er greift auf fertig zusammengestellte Angebote von Reiseveranstaltern oder auf Vorschläge aus Büchern zurück. Die Übergänge sind aber fließend.

Diese drei Teilgruppen sollen die Eckpunkte der folgenden Betrachtung sein. Eine so harte Trennung wird in der Realität zwar nie zu erkennen sein und eine einzelne Person tritt im laufe der Zeit natürlich in mehreren dieser Rollen auf. Aber an dieser Aufteilung lassen sich die Anforderungen an ein Informations- und Navigationssystem für Touristen gut darstellen.

4.3 Informationsbeschaffung und Planung

Im Allgemeinen lassen sich folgende Regeln aus den Gesprächen über **LingWear** ableiten.

- Die Planung der Geschäftsreisenden ist hauptsächlich auf die An- und Rückreise, die Unterbringung und eventuell die Bewältigung von Wegstrecken zwischen Hotel und z.B. Tagungs- oder Arbeitsräumen ausgerichtet. Bei größeren Konferenzen ist es manchmal der Fall, daß den Teilnehmern auch Informationen über den Zielort und Vorschläge für ein ansprechendes Abendprogramm wie Theaterbesuche angeboten werden. Außerdem ist das Tagesprogramm oft sehr straff und es bleibt nur wenig Zeit zur eigenen Verfügung. Daher tritt die Planung und Informationsbeschaffung in diesen Fällen eher in den Hintergrund. Informationsquellen sind beispielsweise Konferenzunterlagen und Begleitmaterial, welches oft erst vor Ort ausgegeben wird, oder im Vorfeld das Internet. Geplant wird oft nebenbei und kurzfristig.

- Der Urlauber ist im Gegenzug zum Geschäftsreisenden im Vorfeld viel mit der Planung und der Wahl des Reiseziels oft mehrere Monate beschäftigt. Dabei nutzt er als Informationsquellen Berichte aus erster Hand, Reiseführer, Dokumentationen und Berichte in Zeitschriften, im Fernsehen und in zunehmendem Maße auch das Internet. Vor allem Jüngere und Computererfahrene runden ihr Bild durch aktuelle Informationen aus dem Web ab. Dabei kommt der Nachteil von Reiseführern und Büchern ins Spiel, daß dieses Material langfristig angelegt ist und deshalb unmöglich aktuelle Ereignis oder Veranstaltungstips und Ausstellungen enthalten kann, genauso wenig wie den aktuellen Theater-Spielplan. Diese Bedürfnisse kann das Internet aber optimal befriedigen. Viele Städte bieten eigene Web-Seiten für diesen Zweck an.

- Der Wochenendtourist nimmt eine Stellung zwischen den o.g. Typen ein. Seine Spontanität hat zur Folge, daß die Planungsphase kürzer und weniger gründlich ausfällt als beim Urlauber, aber da er seine Reise doch zum großen Teil selbst bestimmt, plant er doch mehr und anders als der Geschäftsreisende. Hat er Bekannte vor Ort, kann es auch vorkommen, daß diese für ihn ein Reiseprogramm zusammenstellen und er sich um (fast) nichts zu kümmern hat. Nimmt er diese Gelegenheit wahr, wird er höchstens kleinere Änderungen und Anpassungen im Gespräch mit seinen ortskundigen Freunden wünschen. Eine Versuchsperson erzählte mir, daß in ihrem Umfeld oft in kleineren Gruppen ‚einfach zum Spaß' eine größere Stadt angesteuert wird. Dort läßt man sich dann mehr oder weniger treiben. Es findet also bei dieser Variante keine Informationsbeschaffung in nennenswertem Maße statt.

Unabhängig konnte festgestellt werden, daß der Umfang der Planung mit dem Alter der Reisenden und natürlich mit der größe der Reisegruppe zunimmt. Jüngere verlassen sich gerne auf Erfahrungen, die ältere Mitreisende im Laufe der Zeit gemacht haben.

Die Informationsbeschaffung vor Ort ist nicht zu unterschätzen. Nur am Zielort selbst ist es möglich, Zugang zu allen Angeboten zu erhalten. Selbst Reisebüros können nicht immer auf dem neuesten Stand sein.

4.4 Bauart und Bedienung

Bei meinen Umfragen machte ich die Erfahrung, daß die meisten Benutzer keinerlei Vorstellung davon hatten, wie die optimale Bedienung von **LingWear** aussehen könnte. Deshalb machte ich ihnen verschiedene Vorschläge wie Handschrifterkennung, Eingabe über eine mehrfachbelegte Tastatur (wie bei Handys), Sprachein- und Ausgabe, sowie allerlei Kombinationen dieser Schnittstellen. Die meisten Befragten bevorzugten dabei die Sprachsteuerung, ohne aber Erfahrung auf diesem Gebiet z.B. mit Diktiersystemen zu haben. Je erfahrener sie im Umgang mit Computern und Sprachsteuerung waren, desto mehr vertraten sie die Ansicht, daß eine Sprachausgabe auf die Dauer den Benutzer durch die monotone Aussprache eher stören würde und daß Spracheingabe in einer lauten Umgebung, wie sie im Straßenverkehr herrscht, nicht zuverlässig funktionieren könnte. Außerdem führten viele lange Trainingszeiten für die Spracheingabe an. Es gab auch Bedenken, ein System mit Sprachausgabe könnte in unpassenden Situationen den Benutzer mit langen und unverständlichen Ausgaben überfordern. Als Alternativen wurden virtuelle Tastaturen oder Handschrifterkennung vorgeschlagen.

Einheitlichkeit herrschte allerdings beim Thema *Bauart*, wie ich es hier nennen möchte. Die Geräte sollten nicht größer sein, als daß man sie bequem am Gürtel tragen kann. Andererseits

war jedem der Befragten klar, daß man ein System wie **LingWear** nicht auf einem **Palm Pilot** laufen lassen kann. Alle wünschten sich ein Display, über das das System bedient werden kann.

Ein Geschäftsreisender machte den Vorschlag, **LingWear** als Teil eines Navigationssystems für PKW zu implementieren, das am Zielort aus dem PKW herausgenommen werden kann und dann als Touristeninformationssystem dient. Er zog im mobilen Einsatz eine graphische Benutzungsschnittstelle vor, allerdings im Auto eine Sprachsteuerung, die sich auf wenige Befehle beschränkt und daher sehr einfach ausfallen kann.

Die Bereitschaft, sich mit einem System wie **LingWear** zumindest einmal zur Probe auseinanderzusetzen, ist sehr hoch. Allerdings betonen etwa ein Viertel der Befragten, daß sie auf ihren Ausflügen darauf bedacht sind, so wenig wie möglich mitzunehmen. Einer erklärte mir sogar, er empfinde sogar den Reiseführer in Buchform, den er zur Planung ausgiebig studiere, unterwegs als störend.

Es ist fraglich, ob sich Menschen, die nur gelegentlich verreisen, zum Kauf eines dedizierten Gerätes durchringen werden. Daher finde ich persönlich den Vorschlag sehr interessant, **LingWear** in eventuell abgespeckten Versionen anzubieten. Besitzer von Organizern oder PDAs könnten sich dann nur z.B. die Datenbanken mit Sehenswürdigkeiten auf ihren Geräten installieren, Besitzer von Pocket-PCs könnten eventuell die Sprachsteuerung und Fähigkeiten des Systems nutzen, die mehr Leistung erfordern. So können die Benutzer langsam an das Konzept herangeführt werden. Außerdem wünschten sich alle Befragten, **LingWear** am Zielort ausleihen zu können, anstatt es sich zu kaufen, da sich die meisten nicht sicher waren, ob ihnen ein solches System dauerhaft nutzen würden.

Die Benutzerführung sollte sehr offen gehalten sein und dem Anwender keine starre Vorgehensweise aufdrängen. Bedenkt man, daß zumindest in der Startphase nicht jeder Tourist selbst ein System wie **LingWear** besitzt und sich stattdessen ein Gerät am Zielort ausleiht, so liegt es auf der Hand, daß er sich nicht erst mit langen Einweisungen in das System aufhalten lassen will. Außerdem wird gerade die festgelegte Strukturierung der Informationen in Reiseführern, die teilweise Sehenswürdigkeiten thematisch ordnen, als störend empfunden, wenn die Ordnung nicht den eigenen Vorstellungen entspricht. Man sollte daher — beispielsweise durch Nennung eines Stichworts — frei und direkt auf alles zugreifen können, ohne sich an feste Zugriffspfade wie Menüs halten zu müssen.

4.5 Erwünschte Leistungsmerkmale

Je nach Benutzergruppe weichen die Anforderungen an das System etwas voneinander ab. Wer seine Reise im Vorfeld wenig oder überhaupt nicht plant, entscheidet am Zielort spontaner, es sei denn, der Benutzer kennt sich am Zielort schon aus. In diesem Fall wird er aber auch selten ein Navigationssystem benötigen, sondern eher Informationen über aktuelle Veranstaltungen.

Für Geschäftsreisende und Wochenendtouristen sind vor allem spontane Wahl eines Zieles und Informationsmöglichkeiten bezüglich Unterkunft, Verpflegung und Veranstaltungsangebot wichtig. Diese Informationen müssen immer auf dem neuesten Stand sein. Was nützt das aktuelle Theaterprogramm, wenn es sowieso keine Eintrittskarten mehr gibt. Aus dieser Überlegung heraus erwächst der Wunsch der Befragten, das System mit einer mobilen Internetverbindung auszustatten, denn nur so könne ohne viel Aufwand die Aktualität der Informationen gewährleistet werden. Das System kann dann auch Buchungen übernehmen. In diesem Fall spielt teilweise die Reichhaltigkeit des Angebotes eine untergeordnete Rolle. Wer als Geschäftsreisender zum ersten Mal in einer bestimmten Stadt ist und beispielsweise nach einem italienischen Re-

staurant sucht, wird sich auf den Vorschlag des Systems verlassen, da er die Qualität des vorgeschlagenen Restaurants sowieso nicht einschätzen kann. Ein Kenner wird allerdings enttäuscht sein, wenn sein Lieblingslokal nicht verzeichnet ist.

Touristen, die viel Zeit in die Organisation ihres Aufenthalts investieren, sich schon aus vielen Quellen informiert haben und vielleicht auch schon Erfahrungen aus früheren Aufenthalten am gleichen Ort einbringen können, möchten nicht durch das System eingeschränkt werden. So forderten alle Befragten für dieses Einsatz-Szenario die Möglichkeit, das System selbständig erweitern zu können. So sollten z.B. Sehenswürdigkeiten oder andere Objekte mit Prioritäten oder Notizen des Benutzers versehen werden können. Ebenso sollte der Datenbestand durch den Benutzer erweiterbar sein — zumindest vom heimischen PC aus. Für diesen Benutzerkreis ist es auch sehr wichtig, eigene Routen zusammenzustellen, bei denen nicht nur ein einzelnes Zielobjekt, sondern auch der Weg dorthin teilweise festgelegt werden kann. Damit bietet sich beispielsweise die Möglichkeit, mehrere Sehenswürdigkeiten in einer vom Benutzer festgelegten Reihenfolge zu besuchen und dabei seine eigenen Vorstellungen über die Wegewahl einzubringen. Für die Freunde von Stadtführung — unter den ungefähr zwanzig Befragten gab es davon aber nur drei — sollten es neben einer Liste der wichtigsten Sehenswürdigkeiten auch eine Auswahl von vordefinierten Routen geben, die diese Objekte in ansprechender Art und Weise ansteuern.

Das System sollte neben Landkarten, Beschreibungen zu Objekten und Hintergrundinformationen auch reichlich Bild- Video- und Tonmaterial bereithalten. Die Navigation sollte unbedingt auch öffentliche Verkehrsmittel mit einschließen. Vor allem die Gruppe der Urlauber möchte nicht auf ein enges Gebiet beschränkt sein, sondern auch Städte in der Umgebung mit in das System einbezogen wissen. Dies ergibt in letzter Konsequenz auch wieder die Vereinigung mit Navigationssystemen für PKW.

Die Möglichkeit, auch innerhalb von Gebäuden das Navigationssystem z.B .zu Führungen durch eine Ausstellung oder ein Museum zu nutzen, zog zwar kein Befragter von sich aus in Betracht, aber die meisten erkannten diese Anwendung als großen Mehrwert des Systems gegenüber herkömmlichen Informationsquellen.

Zusatzdienste wie die in **LingWear** integrierte Übersetzung wurden von den Befragten in keinem Fall vorgeschlagen. Nachdem ich sie allerdings auf diese Möglichkeit hinwies und noch auf die Möglichkeit aufmerksam machte, z.B. E-Mails oder Bilder oder Videos vom aktuellen Standort zu verschicken, wurde speziell die Übersetzung sehr begrüßt. Die anderen Vorschläge wurden als Spielerei abgetan.

4.6 Routenbeschreibungen

In der Frage, wie eine Route beschrieben werden sollte und wo die Probleme bei Wegbeschreibungen liegen, waren sich alle Befragten überraschend einig. Damit decken sich auch die in der Praxis gesammelten Erfahrungen.

Die Hauptprobleme sind danach veraltetes Kartenmaterial und erzwungene Routenänderungen, die z.B. durch Bauarbeiten und Sperrungen verursacht werden. Durch diese Umstände wird eine Beschreibung — und sei sie noch so genau — zumindest teilweise unbrauchbar und der Benutzer muß sich doch wieder selbst seinen Weg suchen. Dieses Problem kann nur durch ständige Updates gelöst werden, die möglichst automatisiert ablaufen sollten.

Als Orientierungspunkte und hilfreiche Angaben in Beschreibungen konnten im Allgemeinen nur Straßennamen, Entfernungen und Richtungsangaben ausgemacht werden. Das Abzäh-

len von Straßenkreuzungen oder Ampeln erschien den Befragten als nicht sinnvoll, da es oft unterschiedlich Ansichten darüber gibt, ob eine bestimmte Abzweigung denn nun als Kreuzung, oder eine Fußgängerampel als Ampel zu zählen ist oder nicht. Bei Beschreibungen für Fußgänger ist dieses Problem noch stärker ausgeprägt als bei Autofahrern, da für einen Fußgänger fast alles zugänglich ist und es für ihn weniger Beschränkungen wie z.B. Abbiege-Verbote gibt.

Richtungsangaben wurden immer relativ zur aktuellen Fortbewegungsrichtung angegeben. Erstaunlicherweise wurde am Startpunkt in den meisten Fällen nicht einmal die Ausgangsrichtung geklärt oder ein Überblick über die Route gegeben.

Die Frage, ob nicht auch beispielsweise Gebäude als Orientierung dienen könnten, wurde in den meisten Fällen verneint. Was fängt ein Benutzer mit dem Hinweis

„Gehen Sie bis zum Rathaus."

an, wenn er nicht weiß, wie das Rathaus aussieht? Diese Methode der Orientierung kann nur dann angewendet werden, wenn der Benutzer durch einen geeigneten Kartenausschnitt gut unterstützt wird und wenn es sich um herausragende, deutlich sichtbare Objekte handelt. Im Allgemeinen wurde davon aber von den Befragten abgeraten.

Die Befragten sollten, nachdem sie mir die Probleme geschildert hatten, die sie mit Wegbeschreibungen haben, nun selbst eine Beschreibung liefern. Bei rein kartenbasierten Beschreibungen wurde die Route in allen Fällen ausschließlich durch Richtungsangaben und Straßennamen beschrieben. Es wurden keinerlei anderen Orientierungspunkte verwendet oder Zusatzinformationen gegeben. Draußen, wenn wir dann tatsächlich eine Route abgingen, änderte sich dies. Die Versuchspersonen begannen nach einer kurzen Eingewöhnungszeit, auf alle möglichen Dinge aufmerksam zu machen, wie z.B. Straßenbahnhaltestellen, wichtigen Gebäude, Geschäfte usw. Zusätzlich wurden Straßennamen angesagt, relative Richtungs- und — ganz selten — Entfernungsangaben gemacht. Auf Sehenswürdigkeiten und andere interessante Gegebenheiten wurde immer dann hingewiesen, wenn sie in Sichtweite gerieten oder wenn wir an ihnen vorbeigingen. Die Pausen wurden mit Hintergrundinformationen zu den Sehenswürdigkeiten gefüllt.

4.7 Auswirkungen

Viele der in diesem Kapitel angesprochenen Aspekte liegen thematisch außerhalb meiner Hauptaufgabe, der Erzeugung von natürlichsprachlichen Wegbeschreibungen. Trotzdem habe ich sie aufgenommen, um zu zeigen, was von einem System wie **LingWear** erwartet wird.

Was mögliche Strategien zur natürlichsprachlichen Wegbeschreibung angeht, so muß ich leider feststellen, daß zwar die meisten Angaben, die in den Experimenten von den Versuchspersonen zur Beschreibung herangezogen wurden, aus der Karte und aus der datenbank ablesbar sind. Allerdings beruht ein Großteil der Lebendigkeit und Natürlichkeit der Beschreibungen darauf, daß zu einem bestimmten Zeitpunkt immer nur die momentan interessanten Angaben gemacht werden und nicht etwa schon mehrere Kreuzungen im Voraus auf Sehenswürdigkeiten hingewiesen wird, die man noch nicht sehen kann. Dies ist aber nur möglich, wenn das System über die Fähigkeit verfügt, die aktuelle Position genau und automatisch zu bestimmen. Außerdem sind weit mehr Angaben zur Topologie notwendig, als sie der Map Server **Rhea** mit seinem zweidimensionalen Ansatz liefern kann. So können keinerlei Sichtbarkeitsprüfungen durchgeführt werden, da die gegenseitige Überdeckung von Objekten nicht festgestellt werden kann.

Verzichtet man auf die automatische Positionsbestimmung und verlangt dafür vom Benutzer, daß er dem System über Befehle seinen Fortschritt auf der Route mitteilt, so kommt als Beschreibungsstrategie nur die Zerlegung der Route in Teilabschnitte in Frage. Diese Methode steht allerdings der von der Mehrzahl der Versuchspersonen angewandten Strategie entgehen. Als ich dies vorsichtig gegenüber den Versuchspersonen andeutete, war ihre Enttäuschung offensichtlich. Alle beteuerten einhellig, daß ein Navigationssystem ohne selbständige Positionsbestimmung für sie so gut wie wertlos sei.

5 Das Navigationsmodul Space im Detail

5.1 Allgemein

Nun werde ich das Modul **Space** genauer beschreiben, insbesondere den Navigationsanteil, den ich entworfen und implementiert habe. Dazu gehört die Schnittstelle des Moduls und seine Einbindung in das Gesamtsystem, die zur Bestimmung der Route, zur Anreicherung mit Zusatzinformation und zur Beschreibung eingesetzten Verfahren, sowie die grundlegenden Datenstrukturen.

Zu den Aufgaben von **Space** zählen neben der reinen Navigation noch die Information des Benutzers und die — allerdings nur sehr einfache — Routenplanung.

Space wurde so implementiert, daß die einzelnen Verarbeitungsschritte möglichst wenig voneinander abhängen. So kann beispielsweise die Suche nach Sehenswürdigkeiten (*POIs*) ganz abgeschaltet werden, ohne davon unabhängige Teile des Moduls zu stören. Außerdem wurde versucht, den Zugriff auf externe Komponenten wie z.B. den Map Server an wenigen Stellen zu konzentrieren. Alle Informationen zur Route werden deshalb in eigenen Datenstrukturen abgelegt. So kann in späteren Verarbeitungsschritten auf die Daten zugegriffen werden, ohne den Map Server erneut zu benutzen. Dies gelingt leider nicht immer, allerdings kann das Modul trotzdem leicht angepaßt werden, falls sich die Schnittstelle zum Map Server einmal ändern sollte.

Dieser Abschnitt geht teilweise sehr genau auf Details der Implementierung ein, die für den Leser vielleicht uninteressant erscheinen. Da diese Informationen für die Weiterentwicklung des Systems hilfreich sind, ist diese Ausführlichkeit jedoch beabsichtigt.

5.2 Einbindung in das System

Space benutzt nach außen zur Kommunikation den in Abschnitt 3.3.3.6 auf Seite 21 vorgestellten **CommServer**. Aus der Überlegung heraus, **Space** in mehrere Teile zu zerlegen, meldet sich das Modul für drei Kommunikationsgruppen an, nämlich *space* und *direct*. *direct* nimmt die Anforderung von verbalen Beschreibungen der einzelnen Routensegmente entgegen, *space* ist für die übrigen Anfragen an die Navigation und den Tourmanager und die präverbalen Beschreibungen zuständig. Die Implementierung unterscheidet aber die Empfängergruppen nicht, beide teilen sich eine einzige Prozedur zur Auswertung der Nachrichten.

Alle Nachrichten, die über den **CommServer** verschickt werden, orientieren sich einschließlich ihres Inhalts-Feldes am FIPA-Format [11]. Für dieses Nachrichtenformat wurden bei der Entwicklung des **CommServer** sehr flexible Funktionen zur Extraktion der einzelnen Nachrichtenparameter implementiert.

Bei der Verarbeitung der Nachrichten in **Space** spielen Absender und Empfängergruppe keine Rolle, nur der Typ der Nachricht ist wichtig.

1. `(do-path :tfs (%s))`
   - von **NL** an *space*
   - Startet die Bestimmung einer Route. Informationen zu Start und Ziel sind in Form einer TFS gegeben. Mehr dazu in Abschnitt 5.6.1.
2. `(start-path :rid %s :orig (%s) :dest (%s) :img %s :segmentLst (%s) :type (%s)`
   - von **Space** an den Display Manager
   - Reaktion auf `(do-path)`. Liefert dem Display Manager neben einem Kartenausschnitt mit der eingezeichneten Route Informationen, mit der er die Beschreibungen der einzelnen Abschnitte anfordern kann.
3. `(continue)`
   - von **NL** an den Display Manager
   - Veranlaßt den Display Manager, die nonverbale Beschreibung des nächsten Routenabschnittes bei **Space** anzufordern.
4. `(get-preverbal-route-element :rid %s)`
   - vom Display Manager an *space*
   - Reaktion auf `(continue)` auf `(start-path)`. Fordert die präverbale Beschreibung des ersten Abschnittes an.
5. `(get-preverbal-route-element :rid %s :preid %s)`
   - vom Display Manager an *space*
   - Wie oben, nur hier zur Anforderung beliebiger Abschitte. Der Display Manager gibt im einfachsten Fall als `:preid` den Wert an, den er bei der letzten nonverbalen Beschreibung erhalten hat. Damit ruft er die Beschreibungen sequenziell ab.
6. `(preverbal-route-element :rid %s :preid %s :preidN %s :route-elements (%s) :done %d)`
   - von **Space** an den Display Manager
   - Reaktion auf `(get-preverbal-route-element)`. Das Feld `:route-elements` enthält IDs der Knoten und Straßensegmente, Abstände usw. Diese Daten ermöglichen dem Display Manager, den nächsten Abschnitt abzurufen und den Fortschritt des Benutzers auf der Route zu ermiteln und darzustellen.
7. `(get-verbal-message :rid %s :preid %s :language %s)`
   - vom Display Manager an *direct*
   - Ruft die verbale Beschreibung des durch `:preid` gekennzeichneten Abschnittes ab.
8. `(verbal-message :rid %s :preid %s :text (%s) :language %s :img %s)`
   - von **Space** an den Display Manager
   - Reaktion auf `(get-verbal-message)`. Enthält die verbale Beschreibung und ein Bild des relevanten Kartenausschnitts.

Abbildung 5.1: Nachrichten für und von **Space** im Bereich Navigation

Die Abbildungen 5.1 und 5.2 fassen die Nachrichten zwischen **Space** und den übrigen Komponenten zusammen. Als Absender und Empfänger werden zur Vereinfachung nicht immer die wirklichen Namen angegeben, unter denen sich die Komponenten beim **CommServer** registrieren, sondern der Komponentenname. Nur bei **Space** werden die Gruppennamen *space* und *direct* verwendet. Die gezeigten Nachrichten sind nur Nachrichtentypen, also Muster für konkrete Nachrichten. Der Nachrichtentext enthält Platzhalter, wie man sie von den Funktionen **printf** von **C++** her kennt. Genau diese Platzhalter werden dann mittels **format** in **Tcl/Tk** durch Werte ersetzt.

Die Nachrichten sind so gruppiert, wie sie auch im Betrieb aufeinander folgen könnten. Die Nachricht `(continue)` wird zwar von **NL** zum Display Manager verschickt, ist aber eng mit **Space** und den übrigen dargestellten Nachrichten verbunden.

1. `(user-pref :visited %s :preferred %s :rejected %s)`
   - von **UserModel** an *space*
   - Teilt dem Tourmanager die Vorlieben des Benutzers für Sehenswürdigkeiten mit.
2. `(telloptions)`
   - von **NL** an *space*
   - Veranlaßt den Tourmanager, dem Benutzer abhängig von seiner Position mögliche Ziele vorzuschlagen.
3. `(options :text (%s) :img (%s))`
   - von **Space** an den Display Manager
   - Reaktion auf `(telloptions)`. Enthält einen beschreibenden Text und einen Kartenausschnitt, in dem die vorgeschlagenen Objekte eingezeichnet sind.
4. `(identify-object :tfs (%s))`
   - von **NL** an *space*
   - Identifizierung von Objekten als Folge von Anfragen der Art „What's that museum?"
5. `(preverbal-message :preverbal-message-types identify :name (%s) :type (%s) :info (%s))`
   - von **Space** an den Display Manager
   - Reaktion auf `(identify)`. Enthält Name, Typ und Kurzinformation zum erfragten Objekt.
6. `(information-on-object :name (%s))`
   - von**NL** an *space*
   - Fordert Informationstext zu einem Objekt an.
   - Beispielanfrage: „Tell me more about Karlsruhe Castle."
7. `(information-on-object :tfs (%s))`
   - von **NL** an *space*
   - Wie oben, nur für nicht durch einen Namen identifizierte Objekte.
   - Beispiel: „Tell me more about the bridge."
8. `(preverbal-message :preverbal-message-types description :name (%s) :type (%s) :info (%s))`
   - von **Space** an denDisplay Manager
   - Reaktion auf `(information-on-object)`. Liefert eine ausführliche textuelle Beschreibung des Objekts.
9. `(localize-object :tfs (%s))`
   - von **NL** an *space*
   - Lokalisierung von Objekten und Anzeige deren Position auf der Karte.
   - Beispiel: „Where is the Pyramid?"
10. `(preverbal-message :preverbal-message-types localization :name (%s) :xy (%s) :location (%s) :img %s)`
    - von **Space** an den Display Manager
    - Reaktion auf `(localize-object)`. Enthält Name, Koordinaten, Textuelle Positionsbeschreibung (Richtung, Abstand) und einen Verweis auf einen Kartenausschnitt als Bilddatei.

Abbildung 5.2: Sonstige Nachrichten für und von **Space**

Für den Zugriff auf den Map Server **Rhea** stehen eigene Routinen in **Tcl/Tk** bereit. Ursprünglich wurde dazu ein sog. **ArdWish** benutzt, eine Erweiterung von **Wish** dem graphischen

```
# connecting...
ard_addserver Rhea.manager rhea --port $par(rheaport) --host $par(rheahost)

...

# get closest intersections to origination and destination
ard_servercmd rhea getclosestintersection $p1 id1
ard_servercmd rhea getclosestintersection $p2 id2
# get path from $id1 to $id3
ard_servercmd rhea calcpathdescriptionL1 $id1 $id2 desc

...

# get street names of intersection with ID $v($i,id)
ard_servercmd rhea getintersectionnames $v($i,id) rhealiste

...
```

Abbildung 5.3: Zugriff von **Space** auf den Map Server **Rhea**

Interpreter von **Tcl/Tk**. Da diese Funktionen nun in reinem **Tcl/Tk** implementiert wurden, können die normalen Interpreter für **Tcl/Tk** zum Einsatz kommen. Abbildung 5.3 zeigt beispielhaft die Kommunikation zwischen **Rhea** und **Space** allerdings in kompakter Form. In **Space** sind die einzelnen Schritte stärker im Code verteilt.

5.3 Koordinaten und Richtungen

Rhea gibt alle Positionen in einem karthesischen zweidimensionalen Koordinatensystem an. Die Achsen sind gleich skaliert und eine Einheit im Koordinatensystem entspricht auf unseren Karten 0.084 Meter. Bei diesem Maßstab ist es nicht verwunderlich, daß die Koordinatenwerte rasch sehr groß werden. **Rhea** verwendet nur ganzzahlige Koordinaten. Dies ist schon in der Schnittstelle festgelegt. Dies reicht sicher für unsere Zwecke aus, man muß nur in **Tcl/Tk** bei der Implementierung darauf achten, ob man bei Berechnungen Ganzzahl oder Gleitpunktarithmetik verwendet.

Bei einfachen Additionen oder Subtraktionen kommt man eher selten an die Grenzen des Zahlenbereichs, da in **Tcl/Tk** Ganzzahlen im Bereich $[-2^{31}, 2^{31} - 1]i$ dargestellt werden können, also betragsmäßig über rund 2 Milliarden. Die Achsen sind damit, wenn man nur den nichtnegativen Bereich betrachtet, ungefähr $180000km$ lang. Man könnte also für alle Karten, die im Map Server gespeichert sind, ein einheitliches Koordinatensystem wählen und die Grenzen zwischen einzelnen Karten sogar transparent machen.

Bei Winkel-, Abstands- und Richtungsberechnungen kommt man schnell in Bereiche, in denen man z.B. beim Quadrieren von Werten den Zahlenbereich von **Tcl/Tk**-Ganzzahlen überschreitet. Außerdem verlangt die (normale) Division sowieso die Umwandlung in Gleitkommazahlen.

Ein Nachteil des zweidimensionalen Koordinatensystems ist die fehlende Unterstützung für die Navigation auf mehreren Ebenen. Will man z.B. das Innere von Gebäuden nachbilden, um den Benutzer durch ein Museum zu führen und ihm die Exponate zu erklären, muß man die Verteilung der Ausstellung auf mehrere Stockwerke entweder vollständig durch Zusatzdaten

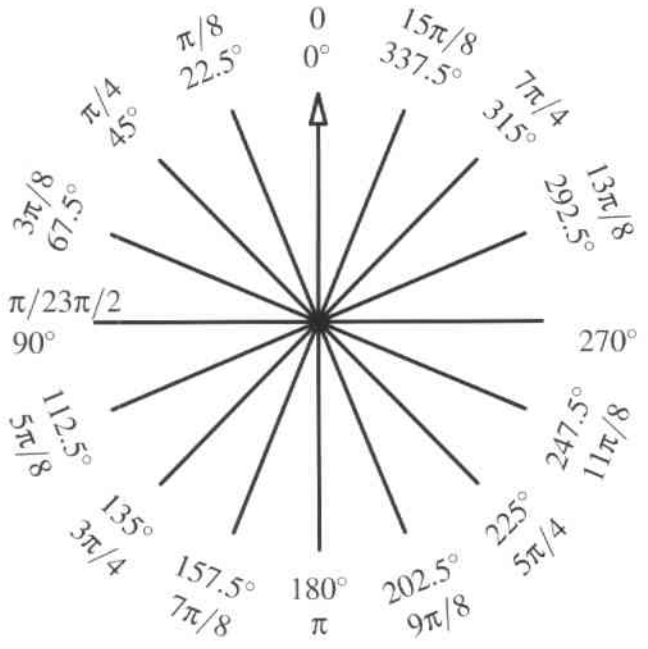


Abbildung 5.4: Winkel in **Space**

nachbilden und verwendet dann nur eine Karte des Gebäudegrundrisses, oder man wechselt zwischendurch die Karten, wenn der Benutzer des Systems das Stockwerk wechselt. Bei beiden Verfahren braucht man aber außerhalb des Map Server Funktionalität, die eigentlich mit der Topologie zu tun hat und deshalb in den Map Server gehört. **Space** beschränkt sich auf die Navigation in der Ebene.

In **Space** werden Winkel in einem System angegeben, das leicht gegenüber dem in der Mathematik üblichen abgewandelt ist. Das kommt daher, daß auf Landkarten die Nordrichtung eine besondere Rolle spielt. Nach ihr werden Karten im allgemeinen ausgerichtet. So zeigt bei **Rhea** dann auch die positive x-Achse nach Osten, die positive y-Achse nach Norden. Während in der Mathematik Winkel in Koordinatensystemen relativ zur positiven x-Achse angegeben werden, wird in der Navigation Norden auf 0° gesetzt (vgl. Abbildungen 5.4 und 5.5). **Space**arbeitet mit Winkeln im Bogenmaß, zur besseren Vorstellung für den Leser verwende ich aber hier meistens Angaben in Grad. Alle Winkel sind orientiert, d.h. es kommt bei der Winkelberechnung auf die Reihenfolge der beiden Vektoren an und Winkel liegen im Bereich $[0, 2\pi]$ anstatt $[0, \pi]$.

Richtungsangaben können in **Space** wie in Abbildung 5.5 gezeigt entweder absolut als Himmelsrichtung oder relativ zu einer Bezugsrichtung angegeben werden. Dabei wird der Vollkreis in beiden Fällen in acht Sektoren zu jeweils 22.5° unterteilt. Zur Bestimmung von Richtungen wird ein Richtungsvektor zwischen dem Standpunkt des Beobachters und dem betrachteten Objekt oder zwischen Start- und Endpunkt von Straßenabschnitten berechnet. Zwischen ihm und der (erwarteten) Blickrichtung des Beobachters wird dann ein Winkel im Bogenmaß zwischen 0 und 2π berechnet. Die Formeln dazu sind aus der linearen Algebra hinreichend bekannt.

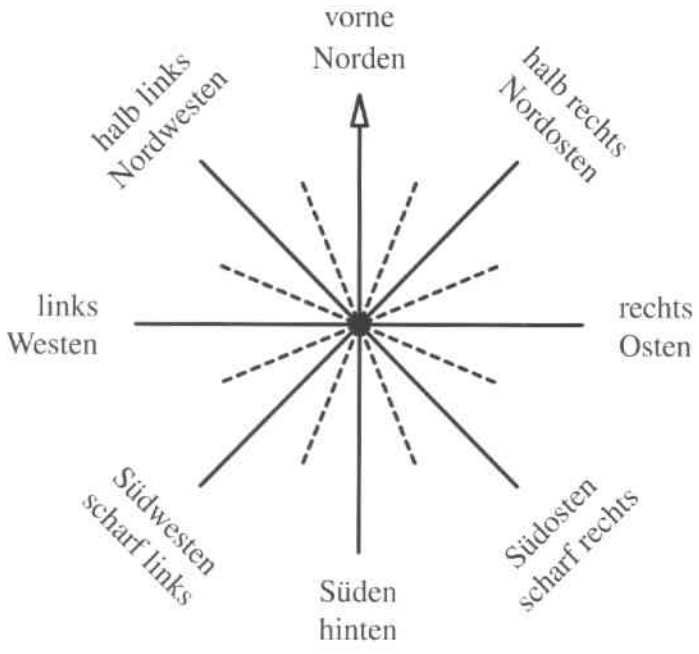


Abbildung 5.5: Richtungen in **Space**

Seien $\vec{r} \neq \vec{0}$ und $\vec{s} \neq \vec{0}$ zwei Richtungsvektoren.

$$\vec{r} := \begin{matrix} r_1 \\ r_2 \end{matrix}$$
$$\vec{s} := \begin{matrix} s_1 \\ s_2 \end{matrix}$$

Dann gilt für den Winkel α zwischen ihnen:

$$\cos(\alpha) = \frac{r_1 * s_1 + r_2 * s_2}{||\vec{r}|| * ||\vec{s}||}$$
$$\sin(\alpha) = \frac{r_1 * s_2 - r_2 * s_1}{||\vec{r}|| * ||\vec{s}||}$$

Und damit:

$$\alpha = \begin{matrix} \arccos(\alpha) & : & \sin(\alpha) >= 0 \\ 2\pi - \arccos(\alpha) & : & \sin(\alpha) < 0 \end{matrix}$$

Danach wird anhand der Abgrenzungen zwischen den acht Richtungsangaben, die in Bild 5.5 gestrichelt eingezeichnet sind, die Richtung ermittelt.

Es kann durchaus vorkommen, daß zwei Objekte zwar bei relativen Richtungsangaben in der gleichen Richtung liegen, aber unterschiedliche Himmelsrichtungen ermittelt werden. Dies kommt vor, wenn der Beobachter nicht genau in Richtung einer der acht Himmelsrichtungen blickt. Man kann wählen, ob man lieber relative oder absolute Richtungen in Beschreibungen verwenden will. Bei Beschreibungstexten kann man zwar beliebig zwischen beiden Arten

wechseln, da die Texte immer erst nach Anforderung erzeugt werden, aber die Wahl der Richtungsangaben wirkt sich wegen den unterschiedlichen Bezugsrichtungen auf die Zerlegung der Route in Teilabschnitte aus, wie wir in Abschnitt 5.6.5 sehen werden.

Relative Richtungsangaben sind viel genauer als Himmelsrichtungen. Betrachten wir noch einmal die beiden Abbildungen zu den Richtungsbezeichnungen. Stellen wir uns nun vor, wir kommen von Süd-Südosten (gehen also nach Nord-Nordwesten) und wenden uns dann nach Nord-Nordosten. Wir führen also eine Drehung um maximal $2 * 22.5° = 45°$ durch. Beide Richtungen liegen genau auf den Abgrenzungen der Nordrichtung, könnten also noch als Norden interpretiert werden. Es ergäbe sich also in Himmelsrichtungen ausgedrückt kein Richtungswechsel. Das liegt daran, daß die Bezugsrichtung bei Himmelsrichtungen immer fix bleibt. Bei relativen Richtungsangaben ist unsere Bezugsrichtung im Beispiel Nord-Nordwesten, also 22.5°, die neue Richtung ist Nord-Nordosten, also 337.5° oder $-22.5°$. Daraus ergibt sich ein Drehwinkel von $-45°$ und damit die Richtungsangabe ‚halb rechts'.

Richtungen werden intern in **Space** nur durch Richtungsvektoren und eventuell durch Bezugsrichtungen dargestellt. Es hat keinen Sinn, Richtungen durch Winkel darzustellen, da man ja im laufenden Betrieb durch Umschaltung von relativen auf absolute Richtungsangaben die Bezugsrichtungen ändert. Allerdings ist die Umschaltung durch den Benutzer von außen wie bei fast allen Parametern von **Space** bis jetzt noch nicht vorgesehen. Die Umsetzung in Textform wird im Abschnitt 5.7.3 erklärt.

5.4 Die Modellierung einer Route

Wie kann man ausgehend von den bisherigen Überlegungen nun eine Route repräsentieren? Die Antwort ist einfach und gilt in gleicher Weise intern für den Map Server **Rhea** und für **Space**.

Eine Route besteht aus einer Folge von Straßen- oder Wegabschnitten. Jeder Abschnitt ist anhand des Kartenmaterials nicht weiter in einzeln begehbare Teile zerlegbar. Es bietet sich an, eine Route daher als Graph zu modellieren, in dem die Staßenabschnitte die Kanten und deren Endpunkte die Knoten oder Ecken bilden. Im folgenden werde ich daher die Begriffe ‚Knoten', und ‚Kreuzung', sowie ‚Straße' und ‚Kante' gleichwertig verwenden.

Der Graph besteht nicht nur aus den Straßenabschnitten, die zur Route gehören. Vielmehr können weitere Straßenstücke eingebracht werden, die einen Endpunkt besitzen, der auf der Route liegt und so vom geplanten Weg des Benutzers in andere Richtungen abzweigen. Es ergibt sich in jedem Fall ein zusammenhängender Graph.

Die Kanten der Route sind dabei auszuzeichnen. Sie müssen im Gegensatz zu allen anderen Kanten tatsächlich beschritten werden, während andere Kanten nur als Zusatzinformation dienen. Jeder Knoten hat aus der Fortbewegungsrichtung des Anwenders betrachtet genau eine *ausgehende* und eine **eingehende** Kante. Die übrigen Kanten, die nicht zur eigentlichen Route gehören, können als ungerichtet angesehen werden.

Ecken und Kanten des Graphen werden nun mit den Daten über die modellierten Objekte markiert. So enthalten Knoten Angaben über ihre Position auf der Karte, ihren Namen, Informationen zu Sehenswürdigkeiten oder eine interne ID für den Map Server. Kanten werden mit der Position von Anfangs- und Endpunkt markiert und enthalten daneben ähnliche Daten wie die Knoten. Im folgenden Abschnitt geht es nun darum, wie dieser Graph im Hauptspeicher gehalten wird.

Index	Eintrag	Index	Eintrag
AssignPOIsToVertexFirst	1	WhatPOItypesNotToShow	{}
EdgePOIsDescMethod	2	buildvertexnamesfromstreetnames	1
MakeVPOIsToEPOIs	1	getmoreedges	1
POIOnEdgeMaxDistance	2	includedestinationpois	2
POIOnVertexMaxDistance	0	language	english
ProcessPOIs	1	lookupvertexnamesindb	1
ReassignPOIs	1	maxdistepoi	20
SayIntersectionNames	1	maxdistvpoi	50
ShowPOIsOnMap	1	rhearac	0.084
VPOISortMethod	1	stopatnamedvertex	0
VPOIsGroupByDir	1	stopdirchange	1
VertexDescMethod	2	umlaute	1
WhatPOItypesToShow	ALL	umlautmap	3
...	...	...	...

Abbildung 5.6: Einige Einträge im Feld **par**

5.5 Wichtige Datenstrukturen und der Modulzustand

In diesem Abschnitt sollen kurz die wichtigsten Datenstrukturen in **Space** beschrieben werden. Diese sind die sechs globalen Arrays **vertex**, **edge**, **vertexpoi**, **edgepoi**, **section** und **par**.

In **Tcl/Tk** werden Arrays oder Felder nicht wie in vielen anderen Programmiersprachen mit Ganzzahlen indiziert, sondern vielmehr mit beliebigen Zeichenketten. Sie entsprechen daher den Assoziativ-Arrays der Sprache **Perl**. Desweiteren sind in **Tcl/Tk** keine geschachtelten Arrays, d.h. Arrays als Elemente von Arrays, gestattet. Durch die Indizierung mit beliebigen Zeichenketten lassen sich aber sehr einfach hierarchische Indizes nachbilden, indem die Indexkomponenten durch ein frei wählbares Trennzeichen getrennt werden. Ich verwende hierzu durchgehend das Komma, da in manchen Programmiersprachen Indizes von mehrdimensionalen Feldern auch so geschrieben werden.

5.5.1 Das Feld par

Das Array **par** enthält, wie in Abbildung 5.6 dargestellt, eine Vielzahl von Parametern. Unter anderem sind dies Maximalabstände für die Suche nach Sehenswürdigkeiten (sog. *Points Of Interest*), die Länge einer Koordinateneinheit (**rheafac**), oder Einstellungen, die bestimmen, wie und welche Informationen in die Route aufgenommen werden (z.B. **ProcessPOIs**, **getmoreedges**, ...).

Der Leser möge an den Indexnamen auf die Bedeutung der Einträge schließen. Einige davon werden uns im Laufe dieses Kapitels noch begegnen.

Noch ein Wort zur Auswirkung von Änderungen in diesem Feld: Einige dieser Einstellungen werden nur während der Berechnung und Aufbereitung der Route ausgewertet, andere auch mehrmals im laufenden Betrieb. Da durch manche Einträge wichtige Entscheidungen bei der Aufbereitung oder Segmentierung der Route getroffen werden — z.B. Segmentierungskriterien, Maximalabstände von Sehenswürdigkeiten zur Route oder deren Auswahl — ist es nicht empfehlenswert, während einer laufenden Navigation zu ändern. Sie sollten vielmehr unmittelbar vor einer neuen Routenanfrage auf die gewünschten Werte gesetzt werden.

Index	Eintrag
num	9
0,0,name	Zähringerstraße
0,0,rx	-1249
0,0,ry	62
0,0,w	1.52119732727
0,0,x	8402345
0,0,y	49009018
0,edges	1
0,id	74
0,name	0
0,name2	intersection with Zähringerstraße
0,x	8403594
0,y	49008956
1,0,id	87
1,0,name	Zähringerstraße
1,0,rx	1249
1,0,ry	-62
1,0,w	4.66278998086
1,0,x	8403594
1,0,y	49008956
1,1,id	8
1,1,name	Lammstraße
1,1,rx	247
1,1,ry	634
1,1,w	5.91168523677
1,1,x	8402592
1,1,y	49009652
1,edges	2
1,id	11
1,name	intersection Zähringerstraße Lammstraße
1,name2	intersection with Lammstraße
1,x	8402345
1,y	49009018
...	...

Abbildung 5.7: Einige Einträge im Feld **vertex**

Leider wurde in der Schnittstelle von **Space** für die meisten Einstellungen keine Manipulationsmöglichkeit vorgesehen. Mann könnte **Space** aber leicht so erweitern, daß diese Änderungen über die Nachrichten des **UserModel** möglich sind, welches aber erst im Laufe der Implementierung von **Space** zu **LingWear** hinzukam.

5.5.2 Das Feld vertex

Im globalen Array **vertex** (Abbildung 5.7) werden alle Informationen gespeichert, die sich auf Knoten des in 5.4 besprochenen Graphen beziehen. Ich werde dieses Feld ausführlich beschreiben, da der Aufbau der anderen Felder analog erfolgt und ich mich dann dort kürzer fassen

Index	Eintrag	Index	Eintrag
num	8	...	...
0,id	87	7,id	840
0,name	Zähringerstraße	7,name	0
0,sx	8403594	7,sx	8404077
0,sy	49008956	7,sy	49011115
0,ex	8402345	7,ex	8404225
0,ey	49009018	7,ey	49012396
0,rx	-1249	7,rx	148
0,ry	62	7,ry	1281

Abbildung 5.8: Einige Einträge im Feld **edge**

kann. Es enthält im Eintrag **num** die Anzahl der Knoten, die wie alle numerierten Elemente der Implementierung mit 0 beginnen. Alle Einträge, die einem bestimmten Knoten i zugeordnet sind, haben demnach einen Index, der als erste Komponente i enthält. Jeder Knoten hat Einträge **x** und **y** für seine Position, sowie weitere Einträge für den Knotennamen und eine interne ID des Map Server.

Zu den Knoteninformationen gehören auch von Knoten ausgehende Straßenabschnitte, wobei im **vertex**-Array nicht zwischen den eingehenden und ausgehenden Kanten unterschieden wird, wie es das Abgehen der Route nahelegen würde. Die Informationen über die Straßen dienen hier nur der Beschreibung von Knoten.

Jeder Knoten hat also einen Eintrag **edges**, der Auskunft über die Anzahl an Kanten gibt. Jede Kante hat nun ihrerseits Einträge **name**, **x**, **y**, **rx**, **ry** und **w** für ihren Namen, x- und y-Koordinate des nicht mit dem Knoten identischen Endpunktes der Kante, einem Richtungsvektor $\frac{rx}{ry}$ von der Knotenposition zum anderen Kantenendpunkt und einem Winkel. Dieser Winkel wird für die Sortierung der Kanten bestimmt und wird dort näher besprochen.

Knoten an Kantenendpunkten, die nicht auf der Route liegen, werden nicht im **vertex**-Array aufgenommen. Es sind nur die Knoten enthalten, die auch tatsächlich beim Abgehen der Route besucht werden.

5.5.3 Das Feld edge

Das globale Array **edge** enthält nun noch einmal alle Kanten des Routengraphen, die die zu gehende Route und damit einen Pfad im Graphen vom Start- zum Zielknoten bilden. Zwar könnten diese auch aus dem Array **vertex** gewonnen werden, allerdings bringt ein zusätzliches Feld für diesen Zweck mehr Flexibilität. So kann das Feld **vertex** beliebig um weitere Straßen ergänzt und diese dort auch umsortiert werden.

Abbildung 5.8 zeigt einige Einträge des Feldes. Zu jeder Kante werden Name, ihre ID, ein in Durchlaufrichtung zeigender Richtungsvektor sowie Start- und Endpunkt vermerkt. Die Indizes sind in der schon vorgestellten Art und Weise strukturiert. Nun fehlt noch die Zuordnung von Kanten in **edge** zu den Knoten in **vertex**. Dabei verbindet die Kante mit der Nummer i die Knoten i und $i+1$. **section(num)** ist deshalb auch immer um genau 1 geringer als **vertex(num)**.

Index	Eintrag	Index	Eintrag
num	8	distance	6728.52944648
0,s	0	1,s	1
0,e	1	1,e	3
0,distance	1250.53788427	1,distance	1348.11659486
...	...	...	...
6,s	7	7,s	8
6,e	8	7,e	8
6,distance	1289.52122898	7,distance	0

Abbildung 5.9: Einige Einträge im Feld **section**

5.5.4 Das Feld section

Eine Navigation von Knoten zu Knoten des Graphen ist nicht immer wünschenswert. Vielmehr soll die Route in Gruppen von Kanten zerlegt werden, den sog. Sections oder auch Abschnitten. Das Wort ‚Abschnitt' wird je nach Kontext also für eine einzelne Kante (meist als Straßenabschnitt) oder aber für einen Pfad aus Kanten im Graphen verwendet. Diese Gruppierung von Kanten wird im Array **section** abgelegt, wo jedem Routenabschnitt ein Start- und ein Endknoten sowie seine Länge in **Rhea**-Einheiten zugeordnet wird. In Abbildung 5.9 besteht Abschnitt 0 allerdings genau aus einer Kante. Er besitzt den Startknoten 0 und den Endknoten 1. Normalerweise hätte das Feld **section** höchstens so viele Gruppen von Einträgen wie **edge**. Allerdings wird ihm als Markierung des Routenendes noch ein zusätzlicher Abschnitt hinzugefügt, im Bild der Abschnitt 8. Er hat die Länge 0 und die Einträge für Start- und Zielknoten enthalten den letzten Knoten im Array **vertex**, also das Ziel der Route.

5.5.5 Die Feld vertexpoi und edgepoi

Schließlich will der Benutzer auch noch über Sehenswürdigkeiten informiert werden, während er seinen Weg geht. Da die Behandlung von sog. *Points Of Interest* (*POIs*) möglichst unabhängig erfolgen soll, sind für die Aufnahme der POI-Daten zwei weitere Felder vorgesehen, nämlich **vertexpoi** und **edgepoi**. Der Leser hat nun wahrscheinlich genug Übung, um die Bedeutung der meisten Einträge in Abbildung 5.10 selbst herauszufinden. In ihm werden POIs festgehalten, die Knoten zugeordnet sind.

Das Feld **edgepoi** hat den gleichen Aufbau und enthält die POIs, Kanten zugeordnet werden. Zwischen **vertex** und **vertexpoi** und **edge** und **edgepoi** besteht also die Beziehung, daß das jeweils erste Feld Teile der Route enthält, denen die POIs aus dem zweiten Feld zugeordnet sind.

5.5.6 Die POI-Datenbank

Die POI-Datenbank verzeichnet Sehenswürdigkeiten samt deren Position, Öffnungszeiten, Typ, sowie Informationstexte und Bilder dazu. Sie ist momentan als Feld im Hauptspeicher implementiert, welches mit einigen Beispieleinträgen in Abbildung 5.11 zu sehen ist. Dieses Feld wird allerdings nie direkt ausgelesen oder beschrieben, sondern nur über eine Reihe von Funktionen. Daher kann die Implementierung jederzeit geändert werden, ohne daß sich die Schnittstelle ändert. Folgende Funktionen stehen zum Zugriff bereit:

Index	Eintrag	Index	Eintrag
num	9	0,1,ry	-421
0,num	3	0,1,w	2.58491723497
0,0,icon	sun.gif	0,1,x	8403332
0,0,importance	3	0,1,y	49008535
0,0,key	1	1,num	0
0,0,name	Market Place	...	...
0,0,rx	74	8,num	1
0,0,ry	226	8,0,icon	castle.gif
0,0,w	5.96675385155	8,0,importance	5
0,0,x	8403668	8,0,key	0
0,0,y	49009182	8,0,name	Karlsruhe Castle
0,1,icon	sun.gif	8,0,rx	75
0,1,importance	3	8,0,ry	104
0,1,key	2	8,0,w	5.77342751424
0,1,name	Town Hall	8,0,x	8404300
0,1,rx	-262	8,0,y	49012500

Abbildung 5.10: Einige Einträge im Feld **vertexpoi**

```
0 {names "Karlsruhe Castle" KARLSRUHE_CASTLE} {type castle building sight}
    {image Castle.jpg} {text Castle.txt} {importance 5}
    {open {month 1 12 {day 2 4 {time 10 17}} {day 5 7 {time 10 18}}}}
    {xy "8404300 49012500"}

1 {names "Market Place" MARKET_PLACE} {type square sight}
    {image Market_Place.jpg} {text Market_Place.txt}
    {importance 3} {xy "8403668 49009182"}

2 {names "Town Hall" TOWN_HALL} {type townhall building sight}
    {image Town_Hall.gif} {text Town_Hall.txt}
    {importance 3} {location "at the Market Place"}
    {open {month 1 12 {day 1 5 {time 10 18}}}} {xy "8403332 49008535"}
```

Abbildung 5.11: Einige Einträge im Feld **db**

- `DBreadFile file` zum Einlesen einer Datei in die Datenbank. Dort sind die Einträge zeilenweise aufgelistet und werden mit fortlaufenden Nummern als Index in das Feld **db** eingefügt. Existiert schon ein Eintrag mit gleichem Namen, wird er aktualisiert und ergänzt. Die Indexnummern bezeichne ich hier auch als *Schlüssel* oder *Key*.

- `DBgetEntry key fieldvar` füllt das Feld **fieldvar** mit den Daten des Eintrages mit Index oder Name **key**. Die Komponenten des Datensatzes können direkt aus dem Feld ausgelesen (z.B. `$fieldvar(type)`) oder durch Funktionsaufruf ermittelt werden.

- `DBcheckName name fieldvar` prüft, ob der Datensatz in **fieldvar** den Namen **name** hat.

- `DBgetName fieldvar` ergibt den ersten Namenseintrag des Datensatzes in **fieldvar**.

- `DBgetXY fieldvar` ergibt eine zweielementige Liste aus x- und y-Wert des Datensatzes.

- `DBsearchName name` liefert den Index des ersten Datensatzes mit passendem Namenseintrag. Mit diesem Wert kann dann eindeutig auf den Datensatz verwiesen werden.
- `DBsearchTypeXY type getName` liefert eine Liste mit Koordinaten und je nach **getName** Namen bzw. Schlüsseln von Datensätzen des passenden Typs.
- `DBfindNameXY x y` sucht nach einem Datensatz mit passenden Koordinaten und gibt den Namen des Objektes zurück.

Jeder Datensatz bekommt beim Einlesen aus den POI-Dateien eine fortlaufende Nummer, als *Key* oder *ID* bezeichnet. Da diese ID aber nicht in den Dateien gespeichert ist und von der Reihenfolge und Anzahl der Datensätze abhängt, ist nicht garantiert, daß ein POI in verschiedenen Sprachversionen der Datenbank unter derselben ID zu finden ist.

5.6 Bestimmung und Aufbereitung einer Route

Was geschieht nun genau, wenn ein Benutzer eine Wegbeschreibung durch eine Navigationsanfrage anfordert? Zunächst muß die Anfrage analysiert werden, danach wird die Route berechnet und mit Zusatzinformationen angereichert. Schließlich kann sich der Anwender Schritt für Schritt die Beschreibung abrufen und bekommt als Reaktion des Systems geeignete Kartenausschnitte und Anweisungen auf dem Bildschirm und über die Sprachausgabe.

5.6.1 Ermittlung von Start und Ziel

Ausgangspunkt ist eine Anfrage der Form *„How can I get from ... to ...“*, wie sie schon in Abbildung 3.6 auf Seite 57 gezeigt wurde. **Space** erhält eine solche Anfrage in Form einer *TFS*, wie sie in Abbildung 3.5 auf Seite 17 dargestellt ist. Daraus müssen nun Start- und Ziel der Route extrahiert werden.

Zunächst wird geprüft, ob ein Startpunkt in der TFS angegeben wurde. Wenn ja, wird der Name und/oder der Typ des Objekts extrahiert. Danach geschieht dasselbe mit der Zielangabe, die immer in der übergebenen TFS erwartet wird. Der Anwender kann sich auch auf ein Objekt aus einer früheren Anfrage beziehen, anstatt das Ziel explizit anzugeben. Ein Beispiel zu diesem Bereich, den ich als Kontext bezeichne, findet sich auf Seite 13. Diese Bezüge werden auch hier behandelt, wozu der Eintrag in **par(object)** ausgewertet wird.

Nun müssen Start und Ziel auf der Karte gefunden werden. Dazu wird zunächst die aktuelle Position abgerufen, meist das Ziel der letzten Routenanfrage. Damit und mit den Angaben über den Startpunkt aus der TFS wird nun geprüft, ob ein Objekt mit dem gesuchten Namen in der POI-Datenbank verzeichnet ist. Trifft dieses nicht zu, wird der Map Server durchsucht. Dabei kann es bei Straßennamen vorkommen, daß mehrere Objekte gefunden werden, da im Map Server eine Straße immer in Abschnitten abgelegt ist, die benachbarte Kreuzungen verbinden. Wir wählen bei mehreren Fundstellen im Map Server die nächstgelegene aus, was mit der aktuellen Position durch Abstandsberechnung leicht möglich ist.

Sind all diese Bemühungen nach der Suche des Startpunkts erfolglos, so müssen wir uns nun mit dem Typ des Startobjekts begnügen. Beispiele für Typen sind **sight**, **building**, **castle** oder **square**. Bei mehreren passenden Objekten wird wieder das nächstgelegene gewählt. Allerdings wird nur die POI-Datenbank durchsucht. Als letzte Alternative wählen wir als Startpunkt die aktuelle Position.

Damit gehen wir nun zur Suche des Zielpunktes über und verfahren genauso wie für den Startpunkt. Allerdings ist hier natürlich die aktuelle Position als letzte Möglichkeit sinnlos. Deshalb kann es im Gegensatz zur Startpunktsuche vorkommen, daß kein Ziel gefunden wird und die Anfrage mit einer Fehlermeldung beantwortet wird.

Danach stehen nun Start- und Zielobjekt fest. Da aber **Rhea** nicht im freien Gelände, sondern nur auf Straßen oder Wegen eine Route berechnet, müssen nun zu beiden Punkten die nächstgelegenen Straßenkreuzungen gesucht werden. Diese beiden Kreuzungspunkte dienen dann als Eingabe zur Routenberechnung.

5.6.2 Berechnung der Route

Ist die Verarbeitung der Navigationsanfrage bis hierher fortgeschritten, sind Start- und Zielpunkt der gesuchten Route bekannt. Nun geht es darum, eine passende Verbindung zwischen den beiden Punkten zu suchen. Hier können sich die verschiedensten Faktoren auswirken, von denen einige kurz angesprochen werden sollen.

Oft ist der schnellste Weg nicht immer der kürzeste. Nehmen wir einmal an, es gäbe zwischen Start- und Zielpunkt einen kurzen Fußweg und alternativ eine längere Verbindung durch öffentliche Verkehrsmittel. Wird nach Zeit entschieden, wird wohl jeder in diesem Fall gerne auf den Fußmarsch verzichten. Allerdings kann es dann sein, daß er die schönsten Sehenswürdigkeiten der Stadt umgeht. Also spielen gerade bei einer Route für Fußgänger gut gewählte Zwischenpunkte, die man unbedingt passieren möchte, eine Rolle. Vielleicht interessiert sich der Anwender nur für ganz bestimmte Gebäude oder Museen. Oder er hat einige Objekte schon besucht und will nicht unzählige Male daran vorbeilaufen, nur weil diese Objekte in der Datenbank des Systems als besonders sehenswert und wichtig vermerkt sind. Wer einen PKW zur Verfügung hat, will vielleicht lieber auf den öffentlichen Personennahverkehr verzichten. Andererseits kann es auch sinnvoll sein, das Auto stehen zu lassen, um Parkplatzproblemen in der Innenstadt auszuweichen.

Dies soll nur einen Eindruck davon vermitteln, welche Aspekte man bei der Berechnung einer Route berücksichtigen sollte, sofern man die notwendigen Informationen aus Datenbanken und Landkarten erhalten kann. All diese Überlegungen lassen sich letztendlich auf ein einziges Problem zurückführen, nämlich die Suche des *kürzesten Pfades* zwischen zwei Knoten in einem Graphen, in dem die Kanten mit Längenangaben markiert sind. Diese Längenangaben müssen nicht wirklich der Weglänge auf einer Landkarte entsprechen. Man kann vielmehr eine Kostenfunktion einfließen lassen, die bestimmte Kanten bevorzugt, andere so hoch bewertet, daß sie nicht für den kürzesten Pfad in Frage kommen.

LingWear setzt den Map Server **Rhea** dazu ein, den kürzesten Weg zwischen einem Startknoten und allen anderen Knoten auf der Karte zu berechnen. Es handelt sich hier wirklich um die geometrische Länge des zu gehenden Weges. Verfahren zur Berechnung von kürzesten Wegen in Graphen findet der interessierte Leser beispielsweise in [12] oder [13]. Sie besitzen meist eine Zeitkomplexität von $O(m * n)$ bei den sog. **LC-Algorithmen** oder $O(m * \log(n))$ beim Verfahren von **Dijkstra**, wobei n die Knoten- und m die Kantenanzahl ist. Insbesondere bei Graphen mit wenigen Knoten und vielen Kanten ist also **Dijkstra** schneller als **LC**.

Nachdem die Startkreuzung auf der Karte gesucht wurde und alle kürzesten Wege zu den anderen Objekten auf der Karte berechnet sind, wird von **Rhea** eine Pfadbeschreibung zur Zielkreuzung angefordert, die aus verschachtelten Listen besteht, die jeweils abwechselnd Daten eines Knotens und einer Kante enthalten. Abbildung 5.12 zeigt auszugsweise eine solche Struktur. Sie wird nun zerteilt und die Daten werden in die Felder **vertex** und **edge** gefüllt. Außerdem

```
nelist <
  nelist < 74 , 8403594 , 49008956 > ,
  nelist < 87 , "ZÄHRINGERSTRAßE" , 8111 > ,
  nelist < 11 , 8402345 , 49009018 > ,
  nelist < 8 , "LAMMSTRAßE" , 679 > ,
  nelist < 12 , 8402592 , 49009652 > ,
  nelist < 847 , "LAMMSTRAßE" , 667 > ,
  nelist < 726 , 8402852 , 49010267 > ,
  nelist < 862 , "ZIRKEL" , 1115 > ,
  nelist < 729 , 8403962 , 49010153 > ,
  nelist < 860 , "KARL-FRIEDRICH-STRAßE" , 13530 > ,
  nelist < 732 , 8404010 , 49010726 > ,
  nelist < 854 , "SCHLOßPLATZ" , 26273 > ,
  nelist < 724 , 8404548 , 49010725 > ,

...                    ...                    ...

>
```

Abbildung 5.12: Rohdaten für eine Route von **Rhea**

werden die restlichen Felder initialisiert. Nun kann dieser von seinem Informationsgehalt minimale Graph mit weiteren Informationen angereichert werden.

5.6.3 Anreicherung durch Zusatzinformationen

5.6.3.1 Weitere Straßenabschnitte

Wie man an der Ausgabe von **Rhea** in Bild 5.12 sieht, werden zunächst nur die Straßenabschnitte auf dem Weg vom Start zum Ziel bereitgestellt. Oft ist es aber wünschenswert, möglichst viele zusätzliche Straßenabschnitte zu finden, die einen Endpunkt besitzen, der sich mit einem Knoten der Route deckt. Aus diesen Straßenabschnitten können dann später Namen für anderweitig unbenannte Kreuzungen entstehen. Dies ist in unserem Fall sehr wichtig, da **Rhea** von sich aus nur die Namen der Straßen in die Ausgabe aufnimmt, Kreuzungen bleiben jedoch unbenannt.

Die Arbeit mit **Rhea** gestaltet sich als etwas umständlich. Zunächst kann über die Knoten-ID eine Beschreibung der Kreuzung angefordert werden, die ihre Position und die Namen der Straßenabschnitte enthält. Die Position einer der Endpunkte jedes Abschnittes ist ja bekannt, nämlich der aktuelle Knoten. um nun aber den anderen Endpunkt zu ermitteln, wird zunächst eine Liste aller Straßenabschnitte mit einem bestimmten Namen angefordert, die auch die Koordinaten enthält. Danach wird für jeden Abschnitt geprüft, ob einer seiner beiden Endpunkte mit der Knotenposition identisch ist. Wenn ja, hat man tatsächlich einen weiteren Abschnitt gefunden, der am aktuellen Knoten endet und kennt nun seinen Namen und seine Koordinaten.

Die Aufnahme von weiteren Straßenabschnitten in den Routengraphen ist optional und kann durch den Eintrag **par(getmoreedges)** aktiviert oder deaktiviert werden.

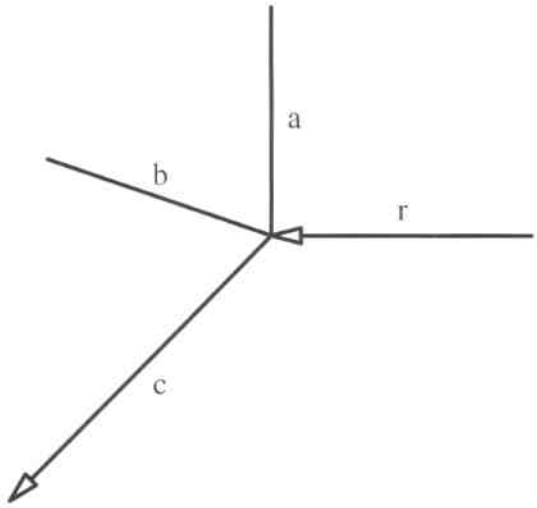


Abbildung 5.13: Beispiel einer Kreuzung

5.6.3.2 Sortierung

Nun, da alle mit der Route direkt verbundenen Straßenabschnitte auf der Karte gefunden sind, kann man sich überlegen, in welcher Reihenfolge man diese für jeden Knoten im Array **vertex** ablegt. Wie schon in Abschnitt 5.5.2 gesagt, dienen die Straßeneinträge in **vertex** lediglich der Routenbeschreibung und nicht der Navigation. So können sie beliebig angeordnet werden.

Bei der Beschreibung einer Straßenkreuzung ist eine geordnete Ausgabe der Straßen sinnvoll, da der Betrachter nicht wiederholt von links nach rechts wechseln will. Ich habe mich hier für eine Ordnung im Gegenuhrzeigersinn entschieden. Es stellte sich aber heraus, daß bei der verbalen Beschreibung einer Route höchst selten die Straßen einer Kreuzung aufgezählt werden, weshalb diese auch in der später beschriebenen Struktur der verbalen Beschreibung, die in Abschnitt 5.7.4 beschrieben wird, nicht auftauchen. Die Sortierung ist daher nicht zwingend notwendig, kam aber bei einer früheren Version der Knotenbeschreibung zum Tragen.

Durchläuft der Benutzer die Kreuzung in Abbildung 5.13 in Pfeilrichtung, so ergibt sich aus seiner Perspektive im Gegenuhrzeigersinn die Sortierung ‚c, r, a, b'. Dabei wird als Bezugsrichtung, der Richtungsvektor der eingehenden Kante benutzt, der so zu orientieren ist, daß er auf den Knoten zeigt. Dann berechnet man die Winkel der Straßen bezüglich dieser Richtung und sortiert schließlich aufsteigend nach diesem Winkel.

Beim Startknoten der Route gibt es keine eingehende Kante. Man könnte zwar als Bezugsrichtung die ausgehende Kante nehmen, allerdings weiß man zu Beginn nicht, in welche der Benutzer des Systems tatsächlich blickt. Es ist also besser, in diesem Fall Himmelsrichtungen zu verwenden. Es ergibt sich dann die Sortierung ‚a, b, c, r'.

Noch ein Wort zur Implementierung der Sortierroutinen. Wie auch die Sortierung der POIs ist diese Routine so implementiert, daß sie nicht geändert werden muß, wenn man weitere Daten in das **vertex**-Feld hinzufügt. Auch optionale Einträge sind kein Problem, da die Implementierung nur Annahmen über die Struktur der Indizes, nicht aber über die tatsächlich eingetragenen Werte macht. Dies wird durch die Nutzung regulärer Ausdrücke erreicht, indem die Vertau-

```
# Parameter:
# v   Vertex-Array
# n   Nummer des aktuellen Knotens.
# Hier werden die Winkel berechnet, aus Platzgründen weggelassen.
# Sortierung:
for {set i [expr $v($n,edges)-1]} {$i>0} {incr i -1} {
   for {set j 0} {$j<$i} {incr j} {
   set k [expr $j+1]
   if {$v($n,$j,w)>$v($n,$k,w)} {
      # j und k tauschen.
      # Array zu Liste
      set vertexlist [array get v]
      regsub -all "(^| )($n,)($k)(,| )" $vertexlist "\\1\\2swapxxx\\4" vertexlist
      # j nach k
      regsub -all "(^| )($n,)($j)(,| )" $vertexlist "\\1\\2$k\\4" vertexlist
      # swapxxx nach j
      regsub -all "(^| )($n,)(swapxxx)(,| )" $vertexlist "\\1\\2$j\\4" vertexlist
      # Liste zu Array
      array set v $vertexlist
   }
   # Ende Tausch
   }
   # Ende for j
}
# Ende for i
```

Abbildung 5.14: Auszug aus der Routine **sortedges**.

schung der Daten zweier Straßenabschnitte nur über eine Umbenennung der Indizes geschieht.

Abbildung 5.14 zeigt den Vertauschungsschritt in **Tcl/Tk**. Zunächst wandelt man das zu sortierende Feld in eine Liste um, da der Befehl **regsub** nur auf Zeichenketten arbeitet. Sollen z.B. am Knoten 0 Kanten 5 und 6 vertauscht werden, so sucht man nach allen Vorkommen von Indizes der Form $0,6,k$ und benennt sie nach $0,swapxxx,k$ um. Danach werden alle Indizes der Form $0,5,k$ nach $0,6,k$ und dann $0,swapxxx,k$ nach $0,5,k$ umgewandelt. k steht hier für einen beliebigen Unterindex, der natürlich selbst auch wieder strukturiert sein darf. Nach Rückwandlung der Liste in ein Feld ist der typische Dreieckstausch beendet.

5.6.3.3 Knotennamen

Da **Rhea** von sich aus keine Knotennamen liefert, müssen wir diese anderweitig gewinnen. Dazu bietet sich als erstes die POI-Datenbank an. Aus ihr lassen sich durch Angabe der Position die Namen von Objekten ermitteln. Der Nachteil dieser Methode liegt darin, daß neben Plätzen, die sich ja als Knotennamen eignen, auch andere Objekte wie z.B. Gebäude in der Datenbank verzeichnet sind. Diese sollten allerdings nicht zur Benennung von Knoten, also Staßenkreuzungen benutzt werden. Deshalb werden bei der Suche nach geeigneten Namen nur Objekte betrachtet, die genau an der Knotenposition sind.

Eine andere Methode zur Bildung von Knotennamen nutzt die Namen der Straßenabschnitte. Zur Erläuterung betrachten wir noch einmal Bild 5.13.

Zunächst wird geprüft, ob es eine eingehende oder eine ausgehende Kante gibt. Wenn ja, werden ihre Namen verwendet, falls beide Namen definiert und unterschiedlich sind. In unserem Beispiel ist dies der Fall und wir erhalten als Knotennamen ‚intersection **r c**‘.

Ist die Bildung eines Knotennamen an dieser Stelle noch nicht möglich, werden nun die übrigen Straßenabschnitte, deren Suche in Abschnitt 5.6.3.1 beschrieben wurde, herangezogen. Findet sich hierbei ein Straßenname, der noch nicht zur Bildung des Knotennamens beiträgt, wird dieser nun berücksichtigt. Die Suche bricht ab, wenn entweder zwei unterschiedliche Straßennamen gefunden wurden oder alle Straßenabschnitte erfolglos durchsucht wurden.

Wäre Beispielsweise Straße **r** unbeschriftet, so würde der erste Suchschritt nur **c** liefern, da **c** ausgehende Kante ist. Im zweiten Schritt ergibt sich als Knotenname dann beispielsweise ‚intersection **c a**'. Dies hängt von der Sortierung der Straßenabschnitte ab.

Als letzter Ausweg wird versucht, eine Knotenbezeichnung aus nur einem Straßennamen wie z.B. ‚intersection with Waldhornstraße' zu bilden. Dabei wird die Bezeichnung einer eventuell vorhandenen eingehenden Kante nicht verwendet. Dies wäre nämlich dem Benutzer nicht hilfreich. Bewegt er sich beispielsweise auf **a** und erhält die Anweisung

„Please follow this way until you reach the intersection with **a**."

oder die Beschreibung

„You are at the intersection with **a**."

so kann er damit wenig anfangen, da ja jede Kreuzung auf **a** diese Bedingung erfüllt.

Parallel dazu wird noch zusätzlich zur normalen Knotenbezeichnung eine weitere Bezeichnung konstruiert, die den Namen der eingehenden Kante nicht enthält. Während die normale Bezeichnung für die Beschreibung des Knotens benutzt wird, an dem sich der Benutzer gerade befindet, wird die zweite Bezeichnung für den nächsten Knoten verwendet. Dies geschieht genau aus oben genanntem Grund.

Da beide Methoden zur Erzeugung von Knotenbezeichnungen nicht immer sinnvolle Ergebnisse liefern können und die Bildung von Bezeichnungen aus Straßennamen vielleicht als störend empfunden wird, sind die Suche nach Namen in der Datenbank und die eben beschriebene Namenserzeugung unabhängig voneinander durch **par(buildvertexnamesfromstreetnames)** und **par(lookupvertexnamesindb)** abschaltbar.

5.6.4 Sehenswürdigkeiten

Obwohl Sehenswürdigkeiten auch eine Art Zusatzinformation darstellen, wird ihnen ein eigener Abschnitt gewidmet, da ihre Suche und Zuordnung etwas komplizierter ist.

5.6.4.1 Suche

Zunächst wird um den gesamten Routengraphen im Koordinatensystem eine sog. *BoundingBox* gelegt. Dazu werden aus allen Koordinaten der Knoten und allen Kantenendpunkten die minimalen und maximalen x- und y-Werte ermittelt. Diese Box muß nun so vergrößert werden, daß auch POIs gefunden werden, die etwas Abstand zur Route haben. **par(maxdistepoi)** und **par(maxdistvpoi)** enthalten Maximalabstände von POIs zu Kanten bzw. Knoten in Metern. Nach Umrechnung in **Rhea**-Koordinaten wird die Box nun nach allen Seiten um das Maximum dieser beiden Werte vergrößert. Nun werden alle POIs gesucht, die in dieser Box liegen und einem Typ aus **par(WhatPOItypesToShow)** angehören. Nach Entfernung von POIs mit Typen aus **par(WhatPOItypesNotToShow)** steht nun eine Liste mit Schlüsseln und Koordinaten bereit, die auf Datensätze in der POI-Datenbank verweisen. Die POI-Typ-Einträge in **par** können durch das **UserModel** beeinflußt werden.

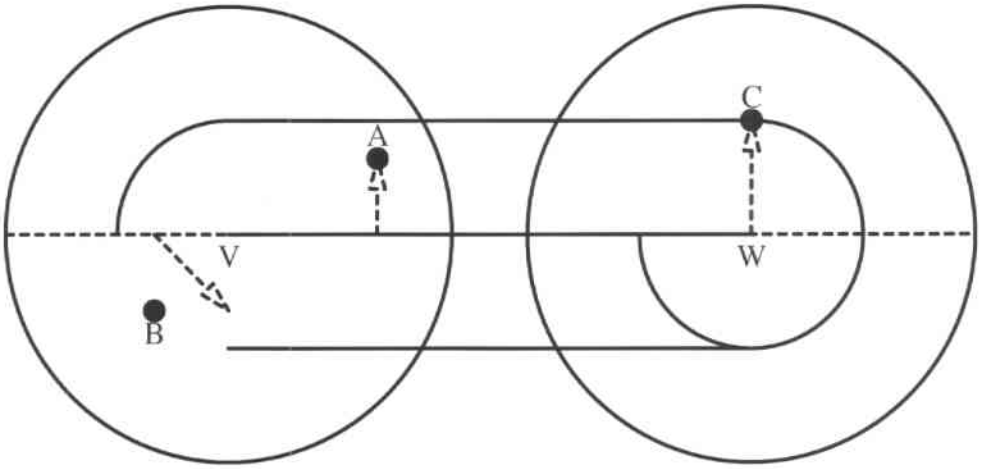


Abbildung 5.15: Beispiel zur POI-Suche

Da als Start- und Endpunkt im Routengraphen nur die nächstgelegenen Kreuzungen zum vom Benutzer gewünschten Start und Ziel sind, kann es vorkommen, daß POIs rund um das gewünschte Ziel oder sogar das Zielobjekt selbst nicht in der Box um den Routengraphen enthalten sind. Deshalb können auf Wunsch alle POIs rund um das Zielobjekt mit aufgenommen werden, oder nur die, die mit den gewünschten POI-Typen verträglich sind. Dies wird im Eintrag **par(includedestinationpois)** festgelegt. Nach dieser einfachen Vorauswahl findet ein komplexer Zuordnungsprozeß der POIs zu Knoten und Kanten statt.

Der Suchraum um einen Knoten besteht aus einem Kreis mit Radius **par(maxdistvpoi)**. Der Suchraum um eine Kante wird durch zwei parallele Linien entlang der Kante im Abstand **par(maxdistepoi)** gebildet, wobei dieser Bereich an jedem Kantenende durch einen Halbkreis abgeschlossen wird. Durch den Abschluß mit Halbkreisen werden Risse im Suchraum am Übergang zwischen zwei Kanten verhindert. In Bild 5.15 sind zwar die Kreise um die Knoten größer als der Maximalabstand zu Kanten, aber das muß nicht so sein. Darum muß auch der Suchraum um die Kanten für sich alleine gut geformt sein. Es wird auch anhand der Abbildung klar, warum man um Knoten einen eigenen Suchraum legen muß. Die zwei Kreise um **V** und **W** überlappen nämlich nicht. Zwar könnte man sie entsprechend groß wählen, würde dann aber bei längeren Kanten wieder die im Bild dargestellte Situation erreichen oder bei zu großen Maximalabständen auch POIs einfangen, die viel zu weit von der Route entfernt sind, als daß sie vom Benutzer von dort aus Beachtung finden.

Man beachte, daß POIs nur den Kanten des Graphen zugeordnet werden, die auch wirklich zur Route gehören, die der Benutzer also wirklich geht. Die zusätzlichen Kanten, die durch die Aufnahme von weiteren Straßenabschnitten in den Graphen entstehen, sind dafür nicht geeignet.

Bei der Zuordnung von POIs zu Knoten und Kanten wird der Abstand zwischen einem POI zu einer Kante benötigt. Zu dessen Berechnung gehe ich folgendermaßen vor: Man denke sich die Kante als Gerade verlängert und berechne nun den Lotfußpunkt einer Geraden durch den POI auf die Kantengerade. Liegt dieser zwischen den Kantenendpunkten wie bei POI **A**, so wird als Abstand zwischen POI und Kante wie üblich die Länge der Lotgeraden zwischen POI und Lotfußpunkt verwendet. Sonst wird der Abstand zwischen dem POI und dem nähergelegenen

Endpunkt als Abstand zwischen POI und Kante gewertet. Dies ist bei POI **B** der Fall.

Dieses Verhalten spiegelt recht gut die Situation eines Benutzers wieder, der sich dem POI nähert. Im ersten Fall passiert er den POI *auf* der Kante und damit auch den Lotfußpunkt. Im zweiten Fall kommt er dem POI nur an einem Kantenendpunkt nahe.

Nun muß für jeden in der Vorauswahl gefundenen POI geprüft werden, ob er einem Knoten oder einer Kante zugeordnet werden kann. Betrachten wir nochmals Abbildung 5.15.

Folgende Einträge in **par** beeinflussen die Zuordnung:

- **par(AssignPOIsToVertexFirst)** legt fest, ob beim Versuch, einen POI zuzuordnen, zunächst Knoten oder Kanten überprüft werden sollen.
- **par(ReassignPOIs)** legt fest, ob und wann eine Neuzuordnung von POIs stattfinden soll. Mögliche Einstellungen sind:
  - Nie.
  - Nur wenn der POI dem neuen Objekt näher liegt als dem alten.
  - Immer.
- **par(maxdistvpoi)** enthält den Maximalabstand zwischen einem Knoten und einem ihm zugeordneten POI.
- **par(maxdistepoi)** enthält den Maximalabstand zwischen einer Kante und einem ihm zugeordneten POI.

Die Zuordnung läuft dann wie folgt ab:

1. Wenn zunächst Zuordnung zu Kanten, dann weiter mit Punkt 4. Sonst hier weiter.
2. POI dem ersten Knoten zuweisen, zu dem er minimalen Abstand hat.
3. Wenn Neuzuordnung erlaubt, dann die erste Kante ermitteln, zu der der POI minimalen Abstand hat. Danach den POI entweder immer dieser Kante zuordnen, oder den Abstand zwischen POI und Knoten aus 2 mit dem Abstand zwischen POI und gerade bestimmter Kante vergleichen und dann dem Objekt mit geringerem Abstand zuordnen. Die Zuordnung ist hier abgeschlossen. Weiter mit dem nächsten POI.
4. POI der ersten Kante zuweisen, zu der er minimalen Abstand hat.
5. Wenn Neuzuordnung erlaubt, dann den ersten Knoten ermitteln, zu dem der POI minimalen Abstand hat. Danach den POI entweder immer diesem Knoten zuordnen, oder den Abstand zwischen POI und Kante aus 4 mit dem Abstand zwischen POI und gerade bestimmtem Knoten vergleichen und dann dem Objekt mit geringeren Abstand zuordnen. Die Zuordnung ist hier abgeschlossen. Weiter mit dem nächsten POI.

Ich empfehle, den Maximalabstand zu Knoten größer zu wählen als den zu Kanten. Außerdem sollte man zuerst versuchen, einen POI einem Knoten zuzuordnen und die Neuzuordnung zu einer Kante nur dann zuzulassen, wenn der POI dieser näher liegt. Dies hat damit zu tun, daß **Space** den Benutzer wegen des fehlenden GPS von Knoten zu Knoten führt und deshalb Kanten-POIs schlechter unterstützt, wie wir später sehen werden. Außerdem ist es sinnvoll,

POIs um Knoten herum mit größerem Abstand zu suchen als um Kanten. Gerade auf größeren Plätzen sind auch weiter entfernte POIs interessant.

Bei diesen Einstellungen und Abständen wie in Bild 5.15 wird POI **A** zunächst **V** zugeordnet, wechselt aber bei der Neuzuordnung zur Kante über. POI **B** hingegen bleibt **V** zugeordnet, da er zum Knoten wie auch zur Kante gleichen Abstand hat. Das liegt daran, daß hier, wie schon gesagt, zur Abstandsberechnung der Kantenendpunkt verwendet wird.

Die POI-Suche ist Dank der vielen Variationsmöglichkeiten der Abstände und der Zuordnungseinstellungen sehr flexibel. Man könnte sich z.B. vorstellen, Kanten überhaupt keine POIs zuzuweisen. Man könnte die POIs zwar über den Suchraum der Kanten sozusagen einfangen, sie dann aber immer an den nächstgelegenen Knoten übergeben.

Zum Schluß sei noch darauf hingewiesen, daß die Zuordnung von POIs zu Knoten oder Kanten nichts über die Sichtbarkeit der POIs aussagt. Es kann durchaus vorkommen, daß ein POI zwar vom nächstgelegenen Knoten durch andere Objekte verdeckt wird, von einer anderen Stelle aus aber gut zu sehen ist. Da aber aus der Datenbank und dem Map Server keinerlei Informationen über solche Verdeckungen abrufbar sind, kann dies von **Space** auch nicht berücksichtigt werden.

5.6.4.2 Sortierung

Auch die POIs können wie die Straßenabschnitte eines Knotens sortiert werden. Die Sortierung soll auch hier das Verhalten des Betrachters nachbilden, der an einem bestimmten Knoten steht, sich die dem Knoten zugeordneten POIs ansieht und dann den Knoten über die nächste ausgehende Kante verläßt und auf dem Weg zum nächsten Knoten die Kanten-POIs passiert.

Beschäftigen wir uns zunächst mit den Knoten-POIs aus Bild 5.16. Es gibt mehrere Möglichkeiten, sie zu sortieren. Die Sortiermethoden nutzen alle die **w**-Komponenten des Feldes **vertexpoi**. Zunächst wird darin der Winkel zwischen dem Richtungsvektor des POI und der Bezugsrichtung abgelegt, wobei als Bezugsrichtung wieder wahlweise die Richtung der eingehenden Kante in den Knoten oder die Nordrichtung eingesetzt wird. Danach werden diese Winkel in Abhängigkeit von der gewählten Sortiermethode modifiziert bzw. ersetzt. Folgende Sortiermethoden wurden implementiert und können durch **par(VPOISortMethod)** gewählt werden:

- Im Uhrzeigersinn oder Gegenuhrzeigersinn. Die Sortierreihenfolgen im Beispiel sind dann ‚B, D, C, A' bzw. ‚A, C, D, B'. Bei Sortierung im Gegenuhrzeigersinn bleiben die Winkel unverändert, bei Sortierung im Uhrzeigersinn gilt: $w := 2\pi - w$

- Nach zunehmendem Winkel zur Blickrichtung, getrennt nach links und rechts. Es ergeben sich die Reihenfolgen ‚A, C, B, D' und ‚B, D, A, C'. Die Modifikationen sind entweder $w >= \pi \longrightarrow w := 4\pi - w$ oder $w >= \pi \longrightarrow w := 2\pi - w$ und $w < \pi \longrightarrow w := 2\pi + w$.

- Nach zunehmendem Winkel, aber ohne Trennung zwischen links und rechts. Es ergibt sich die Reihenfolge ‚A, B, C, D'. **w** wird gemäß $w := -|\pi - w|$ verändert.

- Nach zunehmendem Abstand zum Knoten. Das Ergebnis im Beispiel ist ‚D, C, B, A'. Die Ersetzungsvorschrift lautet: $w :=$ Abstand

- Nach Priorität. Die POIs werden absteigend nach ihren **importance**Einträgen im Feld **vertexpoi** sortiert. $w := -$importance

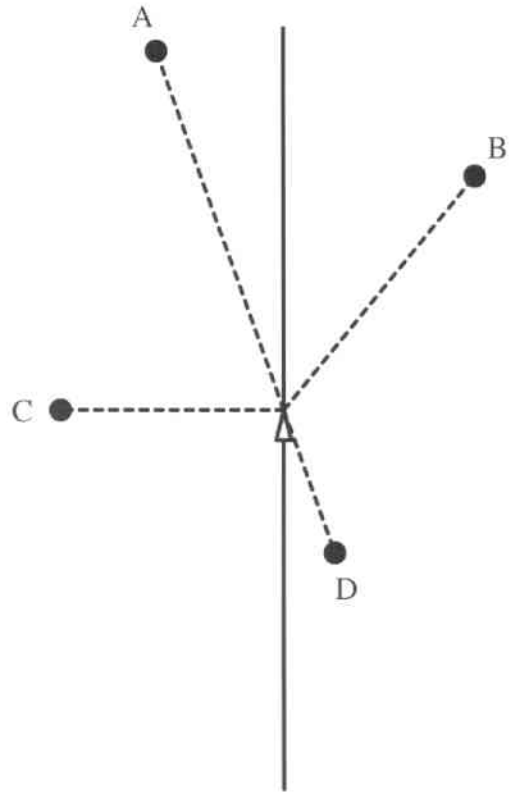


Abbildung 5.16: Sortierung von Knoten-POIs

- Zufällig. Jedem POI wird eine Pseudozufallszahl zugeordnet und die POIs werden dann nach diesen Zahlen sortiert. $w :=$ Zufallszahl

- Unveränderte Reihenfolge. Die POIs werden nicht sortiert. Die Anordnung im Feld **vertexpoi** hängt also nur von der Reihenfolge ab, in der sie einem Knoten zugeordnet wurden. Praktisch dürfte es allerdings kaum möglich sein, durch geschickte Anordnung der POIs in der Datenbank eine bestimmte Ordnung im **vertexpoi**-Feld aufzuprägen, da die Zuordnung schwer vorauszusehen ist.

Zusätzlich zu diesen Optionen kann durch **par(POIOnVertexMaxDistance)** festgelegt werden, bis zu welchem Abstand ein POI noch als *direkt am Knoten positioniert* gilt. Davon betroffene POIs bekommen bei allen Sortiermethoden außer der zufälligen Anordnung und der Sortierung nach Abstand höchste Priorität, indem sein Winkel auf -999 gesetzt wird, und stehen daher zu Beginn. Außerdem besteht die Möglichkeit, die Sortiermethode für jeden Knoten zufällig auszuwählen. So wird die Routenbeschreibung indeterministisch, was sie etwas natürlicher macht.

Für die Sortierung der Kanten-POIs habe ich nur eine einzige Methode implementiert, da sie mir aus eigener Erfahrung und nach Gesprächen mit potentiellen Benutzern, als mit hoher Wahrscheinlichkeit die einzig sinnvolle Ordnung der POIs erscheint. Sie werden in der Reihenfolge sortiert, in der der Benutzer an ihnen vorbeigeht. Auch in Routenbeschreibungen in Reiseführern oder bei Stadtführungen durch Ortskundige wird man oft in dieser Art auf Sehens-

würdigkeiten hingewiesen, die man auf dem Weg zwischen zwei Haltepunkten erreicht.

Bei der Sortierung helfen uns wieder die Lotfußpunkte der POIs auf der Kante, der sie zugewiesen sind. Betrachten wir noch einmal Bild 5.15 auf Seite 51.

Sei P ein Kanten-POI mit Ortsvektor $\vec{p} := \frac{p_1}{p_2}$. Die Kante, der er zugeordnet ist, habe die Endpunkte V und W mit den Ortsvektoren $\vec{v}$ und $\vec{w}$, $\vec{v} \neq \vec{w}$. Zunächst wird der Lotfußpunkt L von P auf der Geraden durch V und W berechnet. Dieser Punkt kann durch die beiden Ortsvektoren der Endpunkte und ein $t \in \mathbb{R}$ eindeutig dargestellt werden.

$$\vec{l} = \vec{v} + t * (\vec{w} - \vec{v})$$

t läßt sich nun aus der Bedingung berechnen, daß die Verbindung zwischen dem Lotfußpunkt L und dem POI P senkrecht zur Kante verlaufen muß.

$$\left\langle \left(\vec{p} - \vec{l}\right), (\vec{w} - \vec{v}) \right\rangle = 0$$

Für $t = 0$ ergibt sich $L = V$, für $t = 1$ dagegen $L = W$. Ist $t < 0$ oder $t > 1$, so liegt L nicht mehr auf der Kante. In Bild 5.15 ergäbe sich z.B. für **A** ein negatives t, für **B** wäre t im Bereich $[0,1]$ und für **C** ist $t = 1$, da der Lotfußpunkt von **C** W ist. t ist also ein Maß dafür, wann ein Benutzer, der die Kante von V nach W durchläuft, an einem POI vorbeikommt.

Zu jedem Kanten-POI wird nun t berechnet und in das Feld **edgepoi** eingetragen. Danach werden die POIs einer Kante aufsteigend nach t sortiert.

5.6.5 Aufteilung der Route

Nun sind alle Informationen zusammengetragen, die zu einer Beschreibung der Route in **LingWear** beitragen. Man könnte jetzt sofort mit der Beschreibung der Route beginnen. Allerdings steht bis jetzt noch nicht fest, wie die Beschreibung sinnvoll zerlegt werden kann. Sicher ist es nicht angebracht, dem Benutzer am Startpunkt schon mit allen Sehenswürdigkeiten, Straßennamen und Richtungen vertraut zu machen. Dies würde ihn überfordern. Andererseits ist es wenig sinnvoll, ihm immer nur den Weg zwischen benachbarten Knoten zu zeigen, da diese nicht immer logischen Abschnitten der Route entsprechen. Wer will schon bei jeder Straßenkreuzung anhalten, wenn er sowieso geradeaus weitergehen könnte und es dort auch nichts zu sehen gibt.

Die Aufteilung der Route in logische Abschnitte — Segmente — ist im Feld **section** abgelegt, das in Abschnitt 5.5.4 beschrieben wurde, und geht nach folgendem Schema vor sich:

1. Wähle den Startpunkt als aktuellen Knoten.
2. Der aktuelle Knoten ist der Startknoten eines neuen Segments.
3. Gehe Knoten für Knoten in Richtung Zielpunkt, bis eine der folgenden Bedingungen eintritt:
   - Die Segmentbildung ist ausgeschaltet. Dies wird durch die globale Variable **bigsections** gesteuert.
   - Die eingehende Kante hat einen anderen Namen als die ausgehende Kante. An diesem Punkt ändert sich der Straßenname, was dem Benutzer mitgeteilt werden sollte. Dieses Kriterium läßt sich durch **par(stopdirchange)** zu- und abschalten.

- Der aktuelle Knoten besitzt Knoten-POIs und der Benutzer möchte in diesem Fall anhalten.
- Die Richtungsangaben für die eingehende und die ausgehende Kante unterscheiden sich *(optional)*. Dabei wird die globale Variable **dirflag** berücksichtigt, die festlegt, ob bevorzugt absolute oder relative Richtungsangaben verwendet werden sollen (vgl. Abschnitt 5.3).
- Der Knoten hat einen Namen. Dieses Kriterium ist abschaltbar und man kann auch bestimmen, ob ein aus Straßennamen gebildeter Kreuzungsname als Knotenbezeichnung gilt oder nicht. Dazu dient der Eintrag **par(stopatnamedvertex)**.

4. Der aktuelle Knoten ist Endknoten des Segments.
5. Gibt es noch weitere Knoten auf dem Weg zum Routenziel, dann kehre zu Schritt 2 zurück.
6. Füge ein weiteres Segment hinzu, das nur den Zielknoten enthält und das Routenende markiert.
7. Berechne die Segmentlängen und trage alles in das Feld **section** ein.

Durch die Kombination der Segmentierungskriterien ist die Aufteilung der Route in Abschnitte sehr flexibel und kann an die Wünsche des Benutzers angepaßt werden. Die meisten einstellbaren Werten in **Space** können aber bis jetzt noch nicht über die Schnittstelle von Außen manipuliert werden.

Segmentbildung und POI-Zuordnung beeinflussen sich gegenseitig. Einerseits kann festgelegt werden, daß bei allen Knoten, denen POIs zugeordnet sind, ein neues Segment begonnen werden soll. Andererseits ist es wenig sinnvoll, Knoten, die keine Segmentgrenze bilden, überhaupt erst POIs zuzuordnen, da der Benutzer an diesen Knoten nicht vom System zum Anhalten aufgefordert wird.

Die Lösung dieser zyklischen Abhängigkeit der beiden Verarbeitungsschritte läßt sich durch eine Nachträgliche Umwandlung von Knoten-POIs in Kanten-POIs erreichen, die durch **par(MakeVPOIsToEPOIs)** gesteuert wird. Alle POIs, die inneren Knoten von Segmenten zugeordnet sind, werden nun zwangsweise der Kante zugeordnet, deren sie am nächsten sind. Man könnte zwar auch die Zuordnung zu Knoten beibehalten, müßte dann aber bei der Beschreibung von POIs entlang eines Segments auf beide POI-Arten eingehen. Durch die Zuordnung zu Kanten erreicht mal also eine Vereinheitlichung.

Die Segmentierung der Route ist nicht immer optimal. Problematisch sind z.B. kurze Straßenabschnitte von nur wenigen Metern Länge, die benötigt werden, um komplexe Staßenkreuzungen darzustellen, bei denen sich nicht alle Straßen in einem Punkt treffen. Sie können dazu führen, daß sehr kurze Segmente entstehen.

5.7 Beschreibung

In diesem Abschnitt soll nun geklärt werden, wie die Route, die nun berechnet und mit allerlei Zusatzinformation angereichert wurde, dem Benutzer beschrieben wird. Die Beschreibung einer Route zerfällt in **LingWear** in zwei Teile, nämlich einen *nonverbalen* und einen *verbalen* Teil. Der nonverbale Teil besteht aus Angaben, die in erster Linie für den Display Manager

gedacht sind und von diesem interpretiert werden. So kann er z.B. aus diesen Angaben die aktuelle Entfernung zum Ziel bestimmen und dies im GUI anzeigen. Dieser Beschreibungsteil wird im Quellcode auch manchmal als *präverbale* Beschreibung bezeichnet. Dies rührt daher, daß zu Beginn geplant war, die verbale Beschreibung eines Routensegments in einer eigenen Komponente mit dem Namen **Direct** zu implementieren. Der Display Manager sollte mit den Daten aus der von **Space** erhaltenen *präverbalen Beschreibung* die *verbale Beschreibung* von dieser Komponente anfordern können. Da nun aber die verbale Beschreibung in **Space** selbst implementiert wurde, ist die Trennung der beiden Beschreibungsarten nicht mehr dringend notwendig. Deshalb fällt die nonverbale oder präverbale Beschreibung eher spartanisch aus.

Zunächst gehe ich noch einmal genau auf den Ablauf einer Navigationsanfrage ein. Dabei wird auch die *nonverbale* oder *präverbale* Beschreibung abgehandelt. Der verbalen Beschreibung eines Routensegments widme ich einen eigenen Abschnitt. Zum Ende folgen einige Gedanken zur Sprachanpassung und ein Beispiel.

5.7.1 Ablauf einer Anfrage und nonverbale Ausgabe

Wie wird nun die Routenbeschreibung über die in Abbildung 5.1 auf Seite 34 gezeigten Nachrichten von **Space** an den Display Manager übergeben?

Zunächst erhält **Space** von **NL** in einer `(do-path)`-Nachricht Angaben über Start und Ziel der Route. Nachdem die Positionen von Start- und Zielpunkt bestimmt, die Route berechnet, diese mit Zusatzinformationen und Angaben zu Sehenswürdigkeiten angereichert und schließlich in Segmente zerlegt wurde, sendet **Space** an den Display Manager eine `(start-path)`-Nachricht, wie beispielsweise folgende:

```
 (start-path :rid 1 :orig Pyramid :dest (Karlsruhe Castle)
:img /home/demo/CSTAR\_ARENA/map.gif
:segmentLst (565 105 113 93 48 45 51 108 0)
:type (castle building sight))
```

Sie enthält eine Routen-ID **rid**, die Namen von Start und Ziel, den Dateinamen eines Bildes, das einen passenden Kartenausschnitt zeigt, sowie eine Segmentliste und ein Typ-Feld, das die Typ-Einträge des Zielobjekts aus der Datenbank enthält.

Das von **Space** mit Hilfe von **Rhea** erzeugte Bild zeigt die gesamte Route, Start- und Zielpunkt, sowie alle nach dem Verfahren aus Abschnitt 5.6.4.1 ermittelten POIs. Diese sind allerdings momentan nur als Punkt auf der Karte dargestellt, da sich bei vielen POIs auf engem Raum die Beschriftungen gegenseitig überdecken. Es besteht außerdem die Möglichkeit, für jeden POI ein Symbol auf der Karte anzuzeigen, das seinem Typ entspricht. Allerdings sind diese Symbole auch sehr groß. Nachdem die Route auf der Karte eingezeichnet ist, wird zunächst die BoundingBox des Kartenausschnitts von **Rhea** erfragt. Diese wird nun nach allen Seiten um das Maximum aus **par(MaxDistVPOI)** und **par(MaxDistEPOI)** vergrößert. Danach wird die endgültige BoundingBox aus der vergrößerten Box von **Rhea** und der POI-BoundingBox aus Abschnitt 5.6.4.1 berechnet und an **Rhea** übermittelt. Danach werden die POIs auf der Karte eingezeichnet.

Die Segmentliste enthält als ersten Eintrag die Länge der gesamten Route und als weitere Einträge die einzelnen Segmentlängen in Metern. Mit diesen Angaben kann der Display Manager die schon zurückgelegte und noch verbleibende Wegstrecke graphisch durch einen Fortschrittsbalken darstellen.

Der Display Manager fordert nun — angestoßen durch `(continue)`-Nachrichten von **NL** — wiederholt von **Space** Daten über das nächste Routensegment an. Dazu nutzt er die Nachricht `(get-preverbal-route-element)`. Sie enthält die in `(start-path)` von **Space** ausgegebene Routen-ID **rid** und außer beim ersten Segment noch die **preid**, die die letzte Nachricht von **Space** als Parameter enthielt. Damit weiß **Space**, auf welchen Teil der Route sich die Anforderung bezieht.

Der Display Manager ist momentan so programmiert, daß er sich genau an obiges Schema hält. Er könnte aber theoretisch auch andere **preid**-Werte übermitteln, um Segmente zu überspringen oder zu wiederholen. Dazu müßte er aber Kenntnis über die Erzeugung der **preid** haben, **Space** wäre in dieser Hinsicht nicht mehr unabhängig. Die **preid** ist z.Zt. zwar nur die Nummer des Segments, auf das sie sich bezieht, könnte aber auch in Zukunft weitere Informationen enthalten.

i Die Antwort auf `(get-preverbal-route-element)` ist eine Nachricht vom Typ `(preverbal-route-element)`. Sie enthält neben der Routen-ID und einer `preid` Angaben zum Routensegment, das im nächsten Schritt verbal beschrieben werden soll. Momentan sind dies abwechselnd die IDs der Knoten und Kanten, aus denen das Segment besteht. Der Parameter `preidN` enthält die Segmentanzahl und ist in der aktuellen Implementierung gleich der Anzahl verschiedener `preid`-Werte, die **Space** erzeugt. In späteren Versionen von **Space** in denen die `preid` nicht nur die Segmentnummer repräsentiert, könnte dies aber nicht mehr gelten.

Die `(preverbal-route-element)`-Nachricht für das erste Segment der Route, die in diesem Abschnitt als Beispiel dient, lautet:

```
(preverbal-route-element :rid 1 :preid 0 :preidN 8
:route-elements (74 87 11) :done 0)
```

Nun fordert der Display Manager endlich die verbale Beschreibung des Routensegments an. Die dazu verwendete Nachricht ist `(get-preverbal-route-element)` sehr ähnlich. Sie enthält im Unterschied dazu aber immer eine `preid`, da sie sich auf ein voriges `preverbal-route-element` bezieht.

```
(get-verbal-message :rid 1 :preid 0 :language ENG)
```

Space hat in der Zeit zwischen `(preverbal-route-element)` und `(get-verbal-message)` schon ein Bild erzeugt, das das aktuelle Routensegment auf der Karte hervorgehoben anzeigt. Jetzt wird der Beschreibungstext in mehreren Schritten zusammengesetzt, die in Abschnitt 5.7.4 besprochen werden sollen. Text und Bild werden nun in eine Nachricht verpackt und an den Display Manager geschickt.

```
(verbal-message :rid 1 :preid 0  :text (...)
:language ENG :img /home/demo/CSTAR_ARENA/map.gif)
```

Damit ist die Beschreibung eines Routensegments beendet. Der Display Manager kann nun das nächste segment anfordern oder der Benutzer kann jederzeit eine neue Route bestimmen lassen.

```
XformatUF {Hallo, @name@.} {name} {Stefan}
```

„Hallo, Stefan."

```
XformatUFR {{Ich bin @name@, ein @was@.}
            {Ich bin das @was@ @name@}} \
   {name was} {LingWear Navigationssystem}
```

„Ich bin LingWear, ein Navigationssystem."
Oder:
„Ich bin das Navigationssystem LingWear."

Abbildung 5.17: Die Anwendung von **Xformat**

5.7.2 Texterzeugung und Variation der Ausgabe

Zur Erzeugung der Segmentbeschreibungen ist es zunächst einmal notwendig, Informationen über Kreuzungen oder einzelne Sehenswürdigkeiten in möglichst natürlichsprachliche Sätze zu formen. Diese können dann zu einer Segmentbeschreibung aneinandergereiht werden.

Zur Bildung von Sätzen oder Satzteilen stehen im globalen Feld **text** vordefinierte Muster zur Verfügung, in die z.B. Straßennamen oder Richtungsangaben eingesetzt werden können. Ich werden diese Muster auflisten, wenn ich ihre Verwendung behandle. Zunächst will ich auf die Einsetzung der Informationen in diese Muster eingehen.

Ich habe eine Reihe von Formatierungsfunktionen implementiert, die als Eingabe im wesentlichen ein Muster, eine Liste von Platzhaltern und eine Liste von Werten erhalten und danach alle Platzhalter im Eingabemuster durch ihre Werte ersetzen. Die Platzhalter sind im Eingabemuster speziell gekennzeichnet. Es ist sogar denkbar, als Wert eines Parameters eine Zeichenkette zu übergeben, die ihrerseits wieder Platzhalter enthält, die dann durch andere Werte ersetzt werden. Von dieser mehrstufigen Ersetzung innerhalb eines Musters wird aber in **LingWear** z.Zt. kein Gebrauch gemacht. Es gibt mehrere Varianten dieser Formatierungsfunktionen. Die wichtigsten sind **XformatUF** und **XformatUFR**.

XformatUF erwartet ein Muster, eine Liste von Platzhaltern, sowie eine Liste von Werten. Nach Ersetzung der Platzhalter durch ihre Werte wird das Ergebnis noch gefiltert, damit keine Sonderzeichen wie geschweifte Klammern mehr darin auftauchen. Dies kann bei **Tcl/Tk** durchaus vorkommen, da sie zur Trennung von Listenelementen benutzt werden. Die Werte werden als Zeichenketten behandelt. In anderen Versionen der **Xformat**-Funktionen kann man für jeden Platzhalter angeben, welches Format sein Wert haben soll. So kann man bei der Ersetzung beispielsweise festlegen, wie Dezimalzahlen formatiert werden sollen. Als Format kann jeder Wert angegeben werden, der auch in der **format**-Funktion von **Tcl/Tk** gültig ist.

Zwei optionale Argumente steuern die Behandlung der Großschreibung. Es können entweder alle Wörter im Ergebnis groß geschrieben, oder das Ergebnis ohne Änderungen übernommen werden. Außerdem läßt sich angeben, daß lediglich der Erste Buchstabe im Ausgabestring groß geschrieben wird.

XformatUFR arbeitet wie **XformatUF**, nur übernimmt sie nicht ein einzelnes Muster, sondern eine Musterliste. Aus ihr wird per Pseudozufallszahlen-Generator ein Muster ausgewählt. Abbildung 5.17 zeigt einige Beispiele der Anwendung von **Xformat**.

Bei der Beschreibung von POIs kommt es oft vor, daß mehrere POIs zu Aufzählungen zu-

Straßenrichtungen	Navigationsanweisungen	Lage von POIs
to the north-east	go north-east	in the north-east
to the south-east	go south-east	in the south-east
...	...	...
straight ahead	go ahead	ahead
to the left	turn left	on your left
backward	go back the way you came	on your back
to the right	turn right	on your right

Abbildung 5.18: Einige Richtungsbezeichnungen

sammengefaßt werden sollen. Es wurde deshalb eine Funktion **ComposePOIList** implementiert, die aus einer als Eingabe erhaltenen Liste eine einzige Zeichenkette zusammensetzt. Die Listenelemente werden dabei durch Elemente im Eintrag **text(POIListSep)** getrennt. Für die Verbindung der letzten beiden Elemente der Liste kommt der Eintrag **text(POIListLast)** zum Einsatz. Beide Einträge in **text** sind Listen, aus denen zufällig ein Element ausgewählt wird.

`ComposePOIList {A B C}` ergibt beispielsweise ‚A, B and C'.

ComposePOIList enthält keinen POI-spezifischen Code und kann daher überall zur Erzeugung von Aufzählungen verwendet werden.

5.7.3 Richtungsangaben

Wie schon in 5.3 auf Seite 36 gesagt, werden Richtungen in **Space** über Winkel zwischen einem Richtungsvektor und der Bezugsrichtung **Norden** oder der Richtung eines anderen Vektors bestimmt. Wie werden nun aber aus Winkeln verbale Angaben? Dazu muß beachtet werden, daß es nicht ausreicht, jedem Winkelbereich starr einen Text zuzuordnen. Dieser würde nicht in allen Situationen passen. In Navigationsanweisungen tauchen Richtungsangaben wie ‚Gehen Sie nach links...' oder ‚Kehren Sie um...' auf. Bei Ortsbeschreibungen lautet die verbale Repräsentation für dieselben Richtungen jedoch ‚Links von Ihnen...' und ‚Hinter Ihnen...'.

Deshalb sind sowohl die Winkelgrenzen zwischen den Richtungen als auch die textuelle Repräsentation in einem globalen Feld **richtung** abgelegt. Bei der Bestimmung einer Richtung wird zunächst der Richtungsvektor vom Ausgangspunkt zum betrachteten Objekt berechnet und dessen Winkel zur Bezugsrichtung berechnet. Betrachten sie sich hierzu noch einmal die Abbildungen 5.4 und 5.5. Danach wird geprüft, in welchen Richtungsbereich dieser Winkel fällt und die Liste der Richtungsbezeichnungen abgerufen. Aus dieser Liste wird nun ein passendes Element gewählt. Die Richtungsbezeichnungen sind sowohl für absolute als auch für relative Angaben gespeichert. Bild 5.18 zeigt einige Bezeichnungen für absolute und relative Richtungen. Man sieht, daß Einträge für Ortsbeschreibungen, die Richtung von Straßenabschnitten und für Navigationsanweisungen vorgesehen sind.

Space ist so eingestellt, daß wenn möglich relative Richtungsangaben verwendet werden. Nur beim ersten Knoten der Route, also beim Startpunkt, ist die Bezugsrichtung undefiniert, da wir nicht wissen, in welche Richtung der Benutzer wahrscheinlich schaut. Deshalb werden an dieser Stelle — und nur dort — Himmelsrichtungen eingesetzt, wie leicht am Beispiel auf Seite 66 zu prüfen ist.

Index	Eintrag
...	...
intersection	intersection
intersectionwith	intersection with
theintersectionwith	the intersection with
endreached	You are at the end of your route!
follow	Please %s and follow %s.
followto	Please %s and follow %s until you reach %s.
VertexDescriptionName	This place is called @name@.
	You have reached @name@.
	This is @name@.
VDesc1POIDir	@dir@ you can see @name@.
	Please have a look at @name@ @dir@.
DescEPOIs1	On your way you will pass @poilist@
DescEPOIs2	You will see @poilist@.
	On your way you will see @poilist@.
	Please have a look at @poilist@.
...	...

Abbildung 5.19: Einige Muster zur Erzeugung von verbalen Beschreibungen im Feld **text**

5.7.4 Struktur der verbalen Ausgabe

Es folgt eine Beschreibung des Aufbaus der verbalen Ausgaben, die **Space** als Beschreibung eines Routensegments in den **verbal-message**-Nachrichten an den Display Manager schickt. Ein Beispiel findet sich am Ende dieses Kapitels.

Zur Erzeugung der Beschreibung wird auf die in den letzten beiden Abschnitten beschriebenen Hilfsmittel zurückgegriffen. Die Muster für die **Xformat**-Funktionen liegen im globalen Feld **text**, das in Abbildung 5.19 auszugsweise dargestellt ist.

Die Beschreibung eines Segments besteht aus drei Teilen:

1. Beschreibung des aktuellen Knotens,
2. Navigationsanweisung für den Benutzer bis zum Segment-Ende
3. und Beschreibung der Sehenswürdigkeiten entlang des Segments.

Sie werden in dieser Reihenfolge einfach hintereinander ausgegeben.

5.7.4.1 Beschreibung des aktuellen Knotens

Als erstes wird geprüft, ob der aktuelle Knoten einen Namen hat und ob dieser sich für die Ausgabe eignet. Zwei Gründe sprechen dagegen, eine Knotenbezeichnung anzusagen (vgl. Abschnitt 5.6.3.3 auf Seite 49):

- Der Knotenname ist ein Kreuzungsname und das System ist durch **par(SayIntersectionNames)** so eingestellt, daß solche künstlich erzeugten Namen nicht angesagt werden sollen.

- Der Name ist identisch mit der Bezeichnung eines POIs, der diesem Knoten zugeordnet ist. Dies kann vorkommen, wenn zur Gewinnung von Knotennamen auch die POI-Datenbank hinzugezogen wurde. In diesem Fall wird der Knotenname nicht angesagt, da er später sowieso als POI wiederkehrt.

Danach werden eventuell vorhandene Knoten-POIs ausgegeben. Dazu werden diese zu Serien gruppiert, wobei alle POIs einer Serie dieselbe Richtungsbezeichnung aufweisen. Es wird wieder zwischen relativen und absoluten Bezeichnungen unterschieden, als Bezugsrichtung für relative Angaben dient die eingehende Kante des Knoten.

Nun kommt die Vorsortierung der POIs (vgl. Abschnitt 5.6.4.2 auf Seite 53) ins Spiel. Sie legt fest, in welcher Reihenfolge die POIs ausgegeben werden. Man beginnt eine neue POI-Serie mit dem ersten noch nicht beschriebenen POI des Knoten und fügt dieser Serie so lange weitere POIs entsprechend der Sortierung hinzu, bis man einen POI erreicht, dessen Richtungsbezeichnung von der der Serie abweicht. Alternativ kann auch durch **par(VPOIsGroupByDir)** festgelegt werden, daß immer alle POIs mit gleicher Richtungsbezeichnung zu Serien zusammengefaßt werden, auch wenn sie nicht in der knoteninternen Anordnung durch die Sortierung aufeinander folgen. Dies kann z.B. bei Sortierung nach zunehmendem Abstand zum Knoten sinnvoll sein, da zwar einerseits die Knoten nach zunehmendem Abstand sortiert ausgegeben werden sollen, aber andererseits auch POIs, die in gleicher Richtung liegen, zusammengefaßt werden können.

Beim Aufbau der Serien ist noch zu beachten, daß POIs, die den Abstand **par(POIOnVertexMaxDistance)** zum Knoten unterschreiten, als ‚direkt am Knoten' behandelt werden und deshalb keine Richtungsangabe erhalten.

Hat man nach diesem Schema nun eine Serie zusammengestellt, werden die Namen der POIs mit **ComposePOIList**) zu einer Aufzählung zusammengefaßt. Dieser kann nun durch die Art und Weise, wie die POI-Serie gebildet wurde, entweder genau eine Richtungsbezeichnung zugeordnet werden, oder die POIs befinden sich alle so dicht am Knoten, daß sie ohne Richtungsangabe ausgegeben werden. Weiterhin wird beachtet, ob die Serie nur einen oder mehrere POIs enthält.

Für diese vier Gruppen von POI-Serien stehen in **text(VDesc1POIHere)**, **text(VDesc1POIDir)**, **text(VDescPOIsHere)** und **text(VDescPOIsDir)** entsprechende Muster für **Xformat** bereit, in die die (eventuell einelementige) POI-Aufzählung und ggf. eine Richtungsbezeichnung eingesetzt wird.

Wie schon erwähnt, richtet sich die Auswahl der Richtungsbezeichnungen an der Einstellung **dirflag**, was die Wahl zwischen relativen und absoluten Richtungsbezeichnungen gestattet. Beim ersten Knoten einer Route werden Himmelsrichtungen verwendet. Der Leser betrachte auch noch einmal Abschnitt 5.7.3, insbesondere Abbildung 5.18 auf Seite 60.

Die Ausgaben zu den einzelnen POI-Serien bilden jeweils vollständige Sätze, die aneinandergereiht werden., bis alle POIs einer Serie zugeordnet und ausgegeben wurden.

5.7.4.2 Navigationsanweisung

Wenn der Name der eingehenden Kante eines Knoten nicht mit dem der ausgehenden Kante übereinstimmt, so wird an diesem Knoten unbedingt eine Segmentgrenze gesetzt. Daher haben entweder alle Kanten eines Routensegments dieselbe Bezeichnung oder sie sind alle unbenannt. Daher muß man bei Navigationsanweisungen, die den Benutzer ein Segment näher an sein Ziel führen, maximal einen Straßennamen verwenden. Wenn der Knoten am Segment-Ende benannt

ist, dann soll auch dieser Knotenname enthalten sein. Dabei ist zu prüfen, ob es eine alternative Bezeichnung für diesen Knoten gibt, der den Namen der eingehenden Kante nicht enthält (vgl. 5.6.3.3, Seite 5.6.3.3). Danach ist noch die Richtungsbezeichnung für den als erstes zu gehenden Straßenabschnitt — also die ausgehende Kante des aktuellen Knotens — zu bestimmen. Danach werden je nach Verfügbarkeit diese Angaben in eines der Muster **text(follow)** oder **(followto)** eingesetzt. Diese werden im Gegensatz zu den übrigen Mustern nicht mit **Xformat**, sondern mit der **Tcl/Tk**-eigenen Funktion **format** ausgefüllt.

5.7.4.3 Kanten-POIs

Hätte **LingWear** GPS-Unterstützung, so könnte man POIs immer dann ansagen, wenn der Benutzer unmittelbar daran vorbeigeht. Wir müssen uns aber darauf beschränken, den Benutzer von Knoten zu Knoten zu führen und ihm dort jeweils die notwendigen Informationen und Hinweise für das nächste Routensegment zu geben. Trotzdem ist zu beachten, daß sich die Angaben zu einem, einer Kante des Routengraphen zugeordneten POI, auf den Augenblick beziehen sollen, in dem der Benutzer des Systems den POI passiert. Daraus folgt, daß als Richtungsangaben nur ‚links' und ‚rechts' sinnvoll sind, denn der Benutzer ist ja in diesem Moment auf gleicher Höhe mit dem POI. Auf absolute Richtungsangaben bei Kanten-POIs wurde verzichtet, da durch die Bewegungsrichtung des Benutzers immer eine Bezugsrichtung gegeben ist.

Sehen wir uns noch einmal Bild 5.15 auf Seite 51 an. Zur Bestimmung der Richtung, in der der POI beobachtet wird, reicht es aus, den (orientierten) Winkel zwischen den Strecken ‚Anfangspunkt der Kante und Lotfußpunkt des POI' und ‚Anfangspunkt der Kante und POI' zu berechnen. Ist er bis auf kleinere durch Rundungsfehler verursachte Abweichungen 90°, so liegt der POI links der Kante, bei $-90° \equiv 270°$ rechts. Dies gilt aber nur dann, wenn der Lotfußpunkt des POI wirklich auf der Kante selbst liegt. Anderenfalls wird die Richtung des POI so bestimmt, als wäre er dem Endpunkt der Kante zugeordnet, dem er näher liegt. Als Bezugsrichtung wird die eingehende Kante dieses Knotens verwendet.

Die Kanten-POIs eines Segments — einige davon gehörten vorher eventuell zu Knoten innerhalb des Segments und wurden nach der Segmentierung einer Kante zugeordnet — wurde ja so sortiert, daß sie innerhalb einer Kante in der Reihenfolge auftauchen, in der sie der Benutzer passiert. Alle Kanten-POIs eines Segments werden nun in drei Gruppen aufgeteilt, ähnlich den Serien bei den Knoten-POIs:

- POIs, die links der Strecke liegen,
- POIs auf der rechten Seite
- und POIs, die den Abstand **par(POIOnEdgeMaxDistance)** zu ihrer Kante unterschreiten und damit als *direkt auf der Kante liegend* betrachtet werden.

Man hätte auch Serien zusammenstellen können, die primär die Sortierreihenfolge und erst danach die Richtung der POIs berücksichtigen, wie es auch bei Knoten-POIs der Fall ist. Aber da es in diesem Fall nur drei mögliche Richtungsangaben gibt, würden bei Gleichverteilung der POIs diese Serien höchstwahrscheinlich kürzer. Außerdem wäre ein wiederholter Wechsel der Richtung bei der Ausgabe der POIs möglich. Dies stört den Benutzer meiner Meinung nach aber mehr als die Abweichung in der POI-Reihenfolge, wenn man sie nach Richtungen gruppiert.

Es besteht auch die Möglichkeit, die POIs ganz ohne Richtung anzusagen. Sie werden dann einfach alle in der Reihenfolge aufgezählt, in der sie nach der Sortierung im Feld **edgepoi** eingetragen sind.

par(EdgePOIsDescMethod) legt die Beschreibungsmethode fest.

Aus den drei POI-Gruppen werden nun Aufzählungen gemacht und bei den Gruppen für ‚links' und ‚rechts' die Richtungsbezeichnung angehängt. Es entstehen beispielsweise die beiden Aufzählungen

‚A, B and C on your left'

und

‚D on your right'

Die POIs, die direkt auf der Kante liegen, werden dann mit einem zufällig gewählten Muster aus **text(DescEPOIs1)** ausgegeben. Die beiden anderen beiden Gruppen werden mit **ComposePOIList** zu einer zweielementigen Aufzählung verbunden. Das Ergebnis könnte so aussehen:

‚A, B and C on the left and D on the right'

Dieser Satzteil wird nun mit **text(DescEPOIs2)** zu einem vollständigen Satz geformt. Nachdem beide ausgefüllten Textmuster aneinandergereiht sind, ist die Beschreibung beendet.

text(DescEPOIs1) und **text(DescEPOIs2)** unterscheiden sich nur darin, daß ersteres Muster enthalten kann, die auch für Ausgaben ohne Richtungsangabe geeignet sind.

5.7.5 Umlaute und Mehrsprachigkeit

Sicher ist Ihnen schon aufgefallen, daß manche Einträge in den vorgestellten globalen Feldern — insbesondere Straßennamen — seltsam geschrieben sind. Anstelle von ß enthalten sie beispielsweise s̃. Dies soll jetzt erklärt werden.

Da einige Komponenten von **LingWear** wie z.B. die Übersetzung und der Display Manager auch mit japanischen Textausgaben umgehen müssen, kommt es zu Problemen bei den verwendeten Zeichensätzen. Im deutschen Zeichensatz liegen nämlich die Umlaute an Positionen, an denen bei der japanischen Sprachversion von **Wish** bestimmte Kanji-Zeichen liegen. Da in einem Ausgabestring auch japanischer und deutscher Text gemischt vorkommen kann, müssen die Umlaute speziell kodiert werden.

Auch **Rhea** verhält sich höchst merkwürdig, was Umlaute und ß angeht. So muß man bei der Suche nach Straßennamen die Umlaute in der Form ã, o oder **ũ** eingeben, im Suchergebnis werden sie aber wieder normal dargestellt.

Aus diesen Gründen werden in **Space** Umlaute immer wie folgt dargestellt:

ä ã Ä Ã
ö õ Ö Õ
ü ũ Ü Ũ
ß s̃

Durch **par(umlaute)** und **par(umlautmap)** kann festgelegt werden, wie Umlaute nach außen an den Display Manager weitergegeben werden sollen und wie sie in Beschriftungen auf der Karte kodiert sein sollen. Die Voreinstellung legt fest, daß Umlaute auf der Karte normal dargestellt werden, aber in der ˜-Form nach außen gegeben werden. Der Display Manager muß dann je nach dort eingestelltem Zeichensatz die Umkodierung der Umlaute vornehmen, bevor er die Ausgabe auf dem Bildschirm anzeigen kann.

Die Sprachausgabe-Komponente tut dies auch, allerdings kommt bei ihr noch ein Problem dazu. Sie hat dafür zu sorgen, daß vor allem Straßennamen oder Bezeichnungen von POIs in der Zielsprache einigermaßen korrekt ausgesprochen werden. Wer sich schon einmal einen

deutschen Text durch eine englische Sprachsynthese angehört hat, der weiß, wovon ich rede. Deshalb werden, bevor der Text endgültig an die Synthese übergeben wird, abhängig von der Zielsprache noch einige Ersetzungen durchgeführt. Als Beispiel will ich neben der geeigneten Ersetzung von Umlauten und anderen Sonderzeichen, die die Synthese irritieren, noch das Beispiel ‚Schloßplatz' anführen. Damit die Sprachsynthese-Software **Festival** dies richtig ausspricht, wird ‚Schloßplatz' durch ‚Schloss-Plutz' ersetzt.

Noch abschließend ein Wort zur Mehrsprachigkeit. Da alle Teile der Ausgabe, die nicht unmittelbar aus der POI-Datenbank oder **Rhea** stammen, im globalen Feld **text** abgelegt und nicht fest in den Programm-Code eingebaut sind, kann man **Space** relativ leicht um weitere Sprachen ergänzen, indem man einfach die Einträge im Feld **text** geeignet ersetzt. Dazu kommen noch die Richtungsbezeichnungen in **richtung**, wie sie auf Seite 60 gezeigt wurden. In dieser Form wurde ein Satz deutscher Muster bereits erstellt. Diese Vorgehensweise birgt aber einige Probleme in sich.

Die Einträge in der POI-Datenbank sind sprachabhängig. Beispielsweise wird der ‚Marktplatz' im Englischen natürlich als ‚market place' ausgegeben. Zwar wird die POI-Datenbank beim Programmstart in der passenden Sprachversion geladen, nicht aber im laufenden Betrieb. Außerdem werden die Namen der POIs ja in die globalen Felder übernommen, in denen die Route abgelegt ist. Beim Umschalten der Sprache müßten diese aktualisiert werden, oder man müßte die Namen überall durch IDs ersetzen, über die dann auf die Namen zugegriffen wird. Dann müßte allerdings garantiert sein, daß jeder Eintrag in allen unterstützten Sprachen vorliegt.

Ein zweites Problem liegt in der Zielsprache selbst. Im Englischen ist die Erzeugung von natürlichsprachlichen Texten sehr viel einfacher als im Deutschen, da dort z.B. bei Straßennamen keine Artikel vorangestellt werden müssen und nicht so viele Anpassungen wie im Deutschen notwendig sind. Man vergleiche beispielsweise nur einmal die folgenden englischen Sätze

- „You have reached *market place*."
- „This is *market place*."
- „Please follow *Zirkel*."
- „Please follow **Kronenstraße**."

mit den deutschen Übersetzungen

- „Sie haben den **Marktplatz** erreicht."
- „Dies ist der *Marktplatz*."
- „Bitte folgen Sie *dem Zirkel*."
- „Bitte folgen Sie *der Kronenstraße*."

Um diese Anpassungen gut und effizient durchführen zu können, sind Methoden notwendig, wie sie z.B. auch bei der Erzeugung von korrekten Sätzen bei der Übersetzung von Text gebraucht werden.

Space ist im Punkt Mehrsprachigkeit auf jeden Fall noch ausbaufähig. Allerdings gehörte die Unterstützung mehrerer Sprachen auch eher zu den untergeordneten Zielen. Die Umschaltung der Sprache im laufenden Betrieb ist aus o.g. Gründen nicht empfehlenswert, aber der

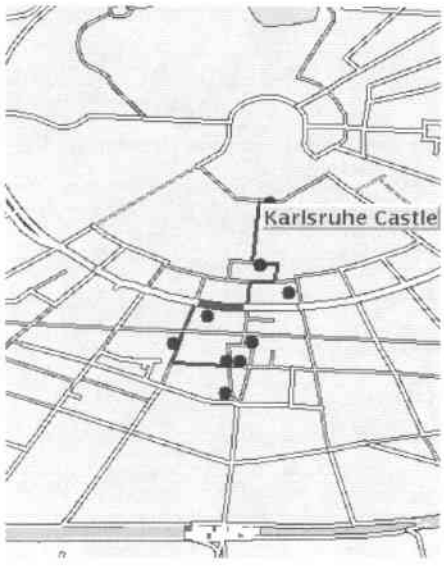
Wechsel der Sprache in Verbindung mit einem Neustart der Komponente ist problemlos möglich, solange die Möglichkeiten der Texterzeugung durch die **Xformat**-Muster für die jeweilige Sprache ausreichend sind.

5.7.6 Beispiel

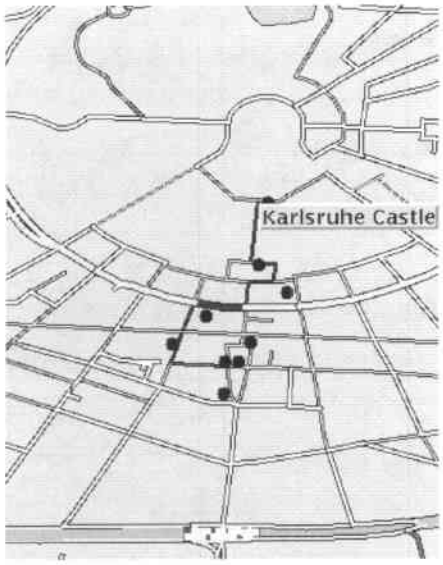
Zum Abschluß meiner Ausführungen soll nun ein vollständiges Beispiel einer Navigationsanfrage folgen. Es lieferte schon im gesamten Kapitel die Daten in den Feld-Abbildungen. Die Einstellungen von **Space** können Bild 5.6 auf Seite 40 entnommen werden.

Segmentgrenzen wurden bei Richtungsänderungen gesetzt. Knoten mit Sehenswürdigkeiten wurden nicht als eigenes Kriterium für die Segmentierung gewählt, deshalb gibt es auch einen Kanten-POI. Das Beispiel zeigt sowohl mehrere Knoten-POIs als auch einen Kanten-POI, nämlich das Kaufhaus ‚Karstadt'. Außerdem wird deutlich, daß ohne den Einsatz von GPS die Benennung von Straßen und Kreuzungen sehr wichtig ist. Die letzten Straßenabschnitte sind z.B. unbenannt.

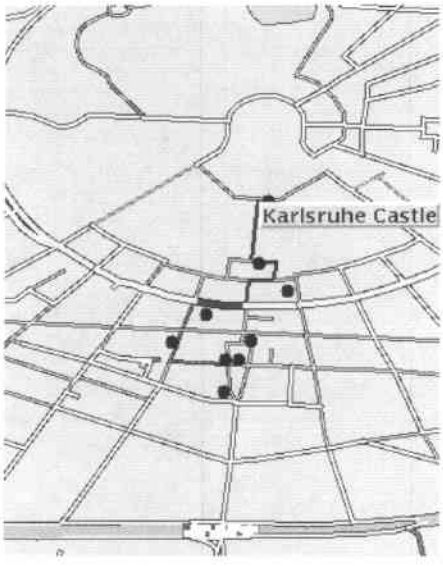
normalqHow can I get from Pyramid to Karlsruhe Castle?



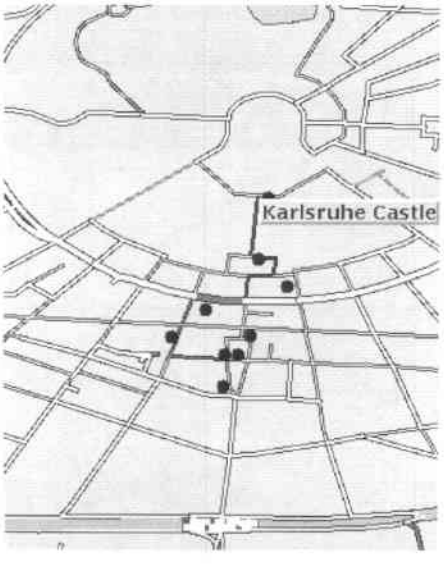
I will give you guidance from Pyramid to Karlsruhe Castle. Use continue to get to the next instruction.



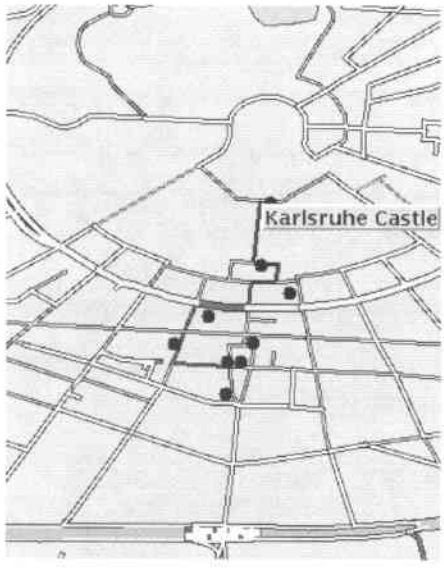
Please have a look at Pyramid in the north-west. You can see Town Hall in the south-west. In the north you can see Market Place.
Please go west and follow Zähringerstraße until you reach the intersection with Lammstraße.



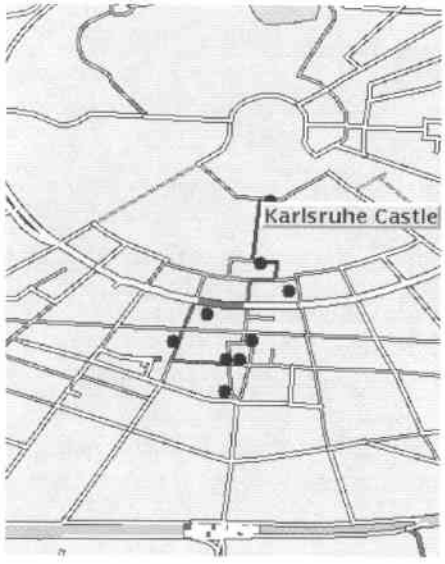
You have reached intersection Zähringerstraße Lammstraße.
Please turn right and follow Lammstraße until you reach the intersection with Zirkel.
On your way you will see Karstadt Department Store on your left.



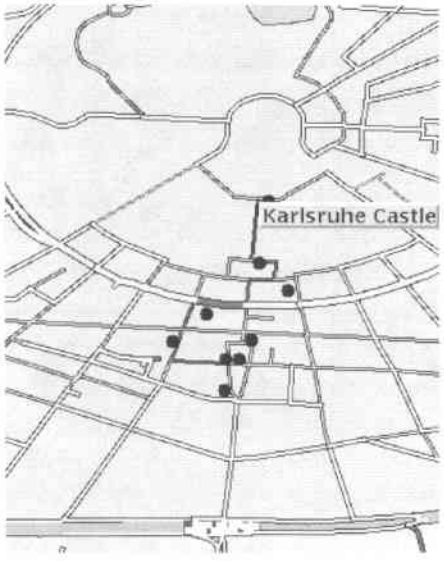
This place is called intersection Lammstraße Zirkel.
Please have a look at Ritterhof behind you on the right.
Please turn right and follow Zirkel until you reach the intersection with Karl-Friedrich-Straße.



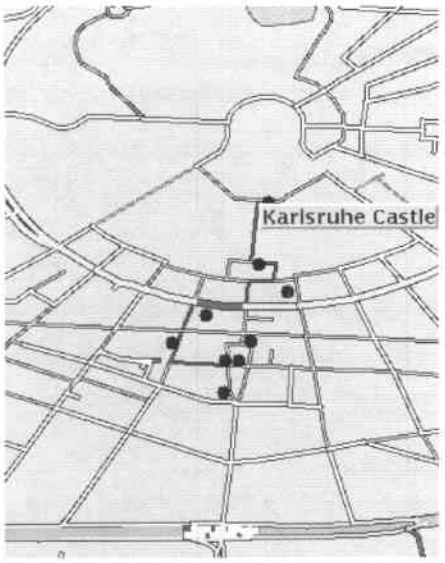
You have reached intersection Zirkel Karl-Friedrich-Straße.
Please have a look at Marktapotheke on your right.
Please turn left and follow Karl-Friedrich-Straße until you reach the intersection with Schloßplatz.



You have reached intersection Karl-Friedrich-Straße Schloßplatz.
Please turn right and follow Schloßplatz.



In front of you on the right you can see Cafe Feller.
Please make a sharp left and follow this way.



In front of you on the right you can see Schloßplatz.
Please bear right and follow this way.

In front of you on the right you can see Karlsruhe Castle.
You are at the end of your route!

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, wie man für das Touristeninformationssystem **LingWear** natürlichsprachliche Wegbeschreibungen erzeugen kann. Dabei wurde zunächst die Struktur eines solchen Informations- und Navigationssystems allgemein dargestellt und dann auf die Architektur von **LingWear** eingegangen. Anschließend wurden die Erwartungen und Wünsche der Benutzer und darauf resultierende Ansatzpunkte im Rahmen der Möglichkeiten der in **LingWear** eingebundenen Komponenten besprochen. Schließlich stellte ich vor, wie aus den Daten des Map Server und der POI-Datenbank eine Beschreibung erzeugt werden kann und was dabei zu beachten ist.

Das System wird ständig weiterentwickelt. Es müssen nicht nur weitere Gebäude und andere Objekte in die Datenbank aufgenommen werden, sondern der Wortschatz des Systems erfordert viel Pflege. Ein großes Problem stellen dabei Eigennamen dar. Sie werden oft von Anderssprachigen falsch ausgesprochen und müssen daher für jede Sprache eigens in die Datenbanken des Erkenners eingebracht werden. Analog dazu muß auch bei der Sprachausgabe eine Sprachanpassung erfolgen, damit der Benutzer die Namen so hört, wie er sie auch selbst aussprechen würde. Andernfalls fällt das Verstehen schwer.

Geplant sind weiterhin die Erweiterung des Systems um verbesserte Eingabemethoden wie Handschrifterkennung und multimodale Schnittstellen, die es auch gestatten sollen, während der Laufzeit z.B. Kommentare und Anmerkungen zu Objekten hinzuzufügen, die dem Benutzer wichtig sind. Eine Anbindung ans Internet könnte bei der automatischen Anpassung des Systems an andere Städte helfen. So könnte sich das System mit zur Lokalität passenden Angeboten im WWW verbinden und daraus das Informationsmodul, die POI-Datenbank, die Spracherkenner und den Parser aktualisieren.

LingWear ist momentan gut als Demonstrations- und Testsystem geeignet, beinhaltet es doch eigens für diesen Zweck Module zur Protokollierung aller Nachrichten und der Ergebnisse von Erkenner und Parser.

Für die Vermarktung solcher Systeme bieten sich gleich mehrere Alternativen an. Städte könnten damit werben und einige Systeme zur Ausleihe anbieten. Privatpersonen, die oft auf Reisen sind, könnten sich selbst ein solches Gerät zulegen, das sie dann ihren Wünschen entsprechend um Daten zu den Reisezielen erweitern können. Vorstellbar ist auch, daß z.B. Automobilhersteller solche Systeme als Teil oder Ergänzung zum Navigationssystem im Auto anbieten und auch spezielle Updateangebote bereithalten. Da sich Internet im Auto wahrscheinlich über kurz oder lang durchsetzen wird, könnte das System dann sogar auf dem Weg zum Zielort vorbereitet und mit Informationen aufgetankt werden.

Literatur und Web-Links

[1] Homepage des **ISL**. http://isl.ira.uka.de

[2] Veröffentlichungen des **ISL**. http://isl.ira.uka.de/nojs/publi_all.html

[3] **Tcl/Tk** Homepage. http://www.scriptics.com

[4] **Garmin**, Hersteller von GPS-Geräten. http://www.garmin.de

[5] **Palm, Inc.**, Hersteller des **Palm Pilot**. http://www.palm.com

[6] **Stefan Jäger**: *NPen++: An On-line Handwriting Recognition System.* 7th International Workshop on Frontiers in Handwriting Recognition, Amsterdam 2000, pages 249–260. http://isl.ira.uka.de/papers/multimodal/IWFHR/IWFHR_stephen1.pdf

[7] **Tanja Schultz, Alex Waibel**: *EXPERIMENTS TOWARDS A MULTI-LANGUAGE LVCSR INTERFACE.* Second International Conference on Multimodal Interfaces (ICMI99), Hong Kong. http://isl.ira.uka.de/papers/speech/ICMI99/ICMI99-tanja.pdf.gz

[8] **Handykey Corporation**, Hersteller des **Twiddler**. http://www.handykey.com

[9] *Java Speech API Markup Language*. http://java.sun.com/products/java-media/speech/

[10] **Christian Fügen, Martin Westphal, Mike Schneider, Tanja Schultz, Alex Waibel**: *LingWear: A Mobil Tourist Information System* Proceedings of the First International Conference on Human Language Technology Conference (HLT 2001), San Diego, march 2001. http://isl.ira.uka.de/papers/speech/HLT2001/hlt2001_fuegen.pdf

[11] **FIPA**: *Foundation for Intelligent Physical Agents.* http://www.fipa.org

[12] **Thomas Ottmann, Peter Widmayer**: *Algorithmen und Datenstrukturen.* 2. vollständige überarbeitete und erweiterte Auflage, BI-Wissenschaftsverlag Mannheim, Leipzig, Wien, Zürich, 1993, S619–631. ISBN 3-411-16602-9.

[13] **Klaus Neumann, Martin Morlock**: *Operations Research.* Hanser-Verlag München, WienHanser, 1993, S203–226. ISBN 3-446-15771-9.

Abbildungsverzeichnis