Modularisierte Routenplanung mit der donavio-Umgebung

Jörg Roth

Fakultät Informatik
Ohm-Hochschule Nürnberg
Kesslerplatz 12
90489 Nürnberg
Joerg.Roth@Ohm-Hochschule.de

Abstract: Dieser Beitrag stellt die *donavio*-Umgebung vor, die ein vollständiges Gerüst für alle Funktionen von Anwendungen zur Routenplanung anbietet. Die Umgebung unterstützt den Einsatz verschiedener Verteilungsarchitekturen, z.B. der Zugriff auf einen Routing-Service über eine Web-Service-Schnittstelle oder die Offline-Navigation auf einem Smart-Phone. Insbesondere die Algorithmen der Routenplanung können modifiziert werden, so sind verschiedene Heuristiken einsetzbar. Die Konfigurierbarkeit geht hinunter bis auf niedrige Ebenen des Datenzugriffs – so können die Laufzeitstrukturen des eingesetzten A*-Verfahrens bezüglich Zeit- und Speicherbedarf abgestimmt werden. Auch für den Zugriff auf die umfangreichen Straßennetz-Daten können verschiedene Varianten eingesetzt werden, optimiert beispielsweise entweder für die Speicherung auf dem Smart-Phone oder auf einem Server.

1 Einleitung

Die Suche nach optimalen Wegen in Straßennetzen ist ein gut erforschtes Gebiet. Die meisten Arbeiten behandeln jedoch im Schwerpunkt die Graphenalgorithmen, meist Variationen des A*-Algorithmus, um möglichst effizient einen Weg auf einem topologischen Straßennetz zu finden. Für den realen Einsatz sind jedoch weitere Funktionen notwendig, z.B. die Entgegennahme von Routing-Aufträgen oder die Generierung von Abbiegekommandos. Bezüglich dieser Funktionen gibt es eine Reihe von Freiheitsgraden. Als Beispiele:

- Wo wird welche Funktion ausgeführt, insbesondere in einer Client-Server-Umgebung. Soll z.B. eine Offline-Navigation auf dem Smart-Phone ausgeführt werden, oder soll die Routenplanung außerhalb des Endgerätes über eine Web-Service-Schnittstelle aufgerufen werden.

- Wie werden die umfangreichen Daten des Straßennetzes gespeichert z.B. als SQL-Datenbank, im Laufzeitspeicher oder auf dem Flash-Speicher eines mobilen Gerätes. Insbesondere ist zu klären, wie man geometrische Abfragen durch einen räumlichen Index beschleunigt.

- Für Algorithmen der Routenplanung (insbesondere im Zusammenhang mit A*) gibt es verschiedene Varianten der Laufzeitoptimierung. Beispiele: Schätzung mit overdo, vorberechnete ALT-Potenziale, ignorieren langsamer Straßen in der Routenmitte.

- Auch für die Generierung der Abbiegekommandos sind verschiedene Ansätze denkbar. So gibt es unterschiedliche Verfahren zur Erkennung, ob man eine Straße verlässt (Namen, Kurvengeometrien, Vorfahrtregeln). Abbiegevorgänge können unter Einbeziehung der exakten Straßengeometrien oder Straßennamen genauer formuliert werden.

In der aktuellen Situation sind die eigentlichen Graphenalgorithmen wissenschaftlich zwar gut untersucht, die genauen Ausgestaltungen verschiedener Varianten, die insbesondere in kommerziellen Routenplanern zum Einsatz kommen, sind aber oft ein Geschäftsgeheimnis. Ziel unserer Arbeit ist daher, eine Umgebung bereitzustellen, die neben der Routenplanung alle Aspekte einer Navigationsanwendung abdeckt. Unterschiedliche Ansätze zu bestimmten Teilbereichen sollen leicht integrierbar sein und getestet werden können.

Es gibt einer Reihe von frei verfügbaren Routenplanungsdiensten, z.B. Open Route Service [NZ08] oder die Routenplanung von Google Maps – hierbei sind allerdings die Interna der Dienste nicht modular austauschbar. Außerdem sind verschiedene Architekturvarianten wie z.B. die Offline-Navigation auf Smart-Phones, nicht möglich.

2 Die donavio-Umgebung

2.1 Einleitung

Vielfach entscheidet man sich bei einer speziellen Softwareumgebung für eine bestimmte Variante einer Teillösung und legt damit den gesamten Entwurf fest. Veränderungen sind dann nur noch mit viel Aufwand nachträglich integrierbar. Ziel der *donavio*-Umgebung ist es, eine modulare Plattform anzubieten, mit der verschiedene Varianten integriert und ausprobiert werden können. Insbesondere sollen unterschiedliche Architekturen (z.B. Client-Server, Offline-Navigation auf Smart-Phones) getestet werden können. Erreicht wird die Modularität über eine Komponentenarchitektur mit festen Schnittstellen:

- Alle auswechselbaren Module werden über objektorientierte Schnittstellen aufgerufen. Diese Vorgehensweise entspricht dem Standardmuster, wenn man unterschiedliche algorithmische Varianten in einem Rahmenwerk kapseln möchte.

- Größere Komponenten (z.B. die Suche nach Straßen über den Namen) haben *zusätzlich* eine Web-Service-Schnittstelle. Hierdurch ist eine flexible Verteilung zwischen Client und Server möglich.

- Einige Zugriffsfunktionen sind so zeitkritisch, dass eine Kapselung über eine Objektschnittstelle zu viel Aufwand zur Laufzeit erzeugen würde. Ein Beispiel ist die Organisation der Knotenlisten innerhalb von A*. Diese kann über feste Arrays signifikant beschleunigt werden. Prinzipiell könnte der Array-Zugriff zwar über Objektschnittstellen gekapselt werden, hiermit würde man aber den Laufzeitvorteil aufgeben. Als Lösung werden zeitkritische Module über Macro-Preprocessing integriert.

Die donavio-Umgebung ist in Java realisiert und läuft auf Desktop-Systemen (Stand-alone oder über Web-Service-Schnittstelle) und als Android-App (Offline-Navigation oder als Web-Service-Client). Zur Beschreibung der Umgebung kann man zwei Anwendungsfälle unterscheiden:

1. *Bereitstellung*:

   Navigationsdaten und Code werden für eine bestimmte Export-Konfiguration oder ein bestimmtes Szenario generiert und für die Benutzung bereitgestellt. Die Daten können beispielsweise auf die Art der Fortbewegung angepasst werden. Der Code kann beispielsweise für eine bestimmte Einsatzumgebung exportiert werden. Da die Daten für die eingesetzten Algorithmen optimiert exportiert werden, müssen Code und Daten zusammenpassen.

2. *Benutzung zur Laufzeit*:

   Der generierte Code wird mit den exportierten Daten verwendet. Die Ausführung kann in bestimmten Grenzen durch eine Laufzeit-Konfiguration beeinflusst werden. Beispielsweise kann ein bestimmtes Verkehrsmittel aus der Liste von denjenigen Verkehrsmitteln ausgewählt werden, die von den exportierten Daten unterstützt werden. Die Grenzen der Laufzeit-Konfiguration ergeben sich daraus, wie Daten und Code unter Punkt 1 bereitgestellt wurden.

Der Rest des Kapitels 2 befasst sich mit der Bereitstellung. Kapitel 3 wird die Laufzeitfunktionen darstellen. Kapitel 4 schließlich zeigt, wie die Modularisierung softwareseitig unterstützt wird.

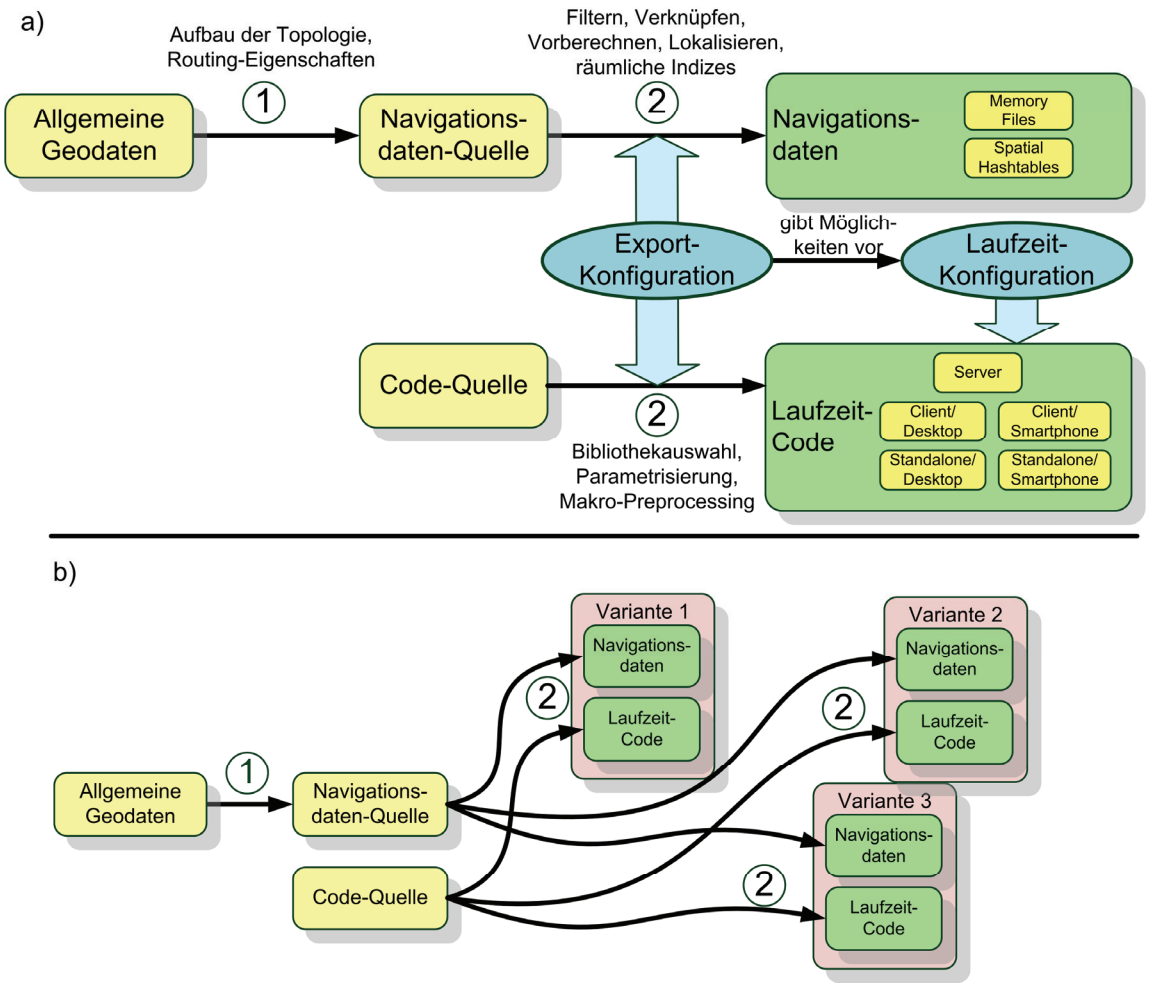

Abbildung 1: a) Phasen der Bereitstellung; b) Erzeugen von Varianten

Abb. 1a) zeigt die Bereitstellung von Code und Daten. Es gibt zwei Phasen:

(1) Aus einem allgemeinen Datenbestand wird eine Navigationsdaten-Quelle abgeleitet. Der Navigationsdatenbestand ist allgemein, beinhaltet beispielsweise alle möglichen Objekte, nicht nur Straßen. Als wichtige Festlegung beim Export wird eine regionale Einschränkung der Daten gemacht, indem z.B. nur alle Einträge aus Bayern zugrunde gelegt werden. Details dieser Übernahme sind in [Ro10b] zu finden.

(2) Aus der Navigationsdaten-Quelle werden für eine bestimmte *Export-Konfiguration* Navigationsdaten abgeleitet. Eine bestimme Export-Konfiguration kann dabei die Einschränkung auf bestimmte Straßen sein (z.B. nur mit dem Fahrrad befahrbar). Zusätzlich werden diverse Vorarbeiten auf den Daten durchgeführt. Parallel zu den Daten wird Laufzeit-Code abgeleitet.

Als Export-Format für die Navigationsdaten werden im Moment *Memory Files* und *Spatial Hashtables* unterstützt (siehe später).

Abb. 1b) illustriert, wie verschiedene Varianten erzeugt werden können. Die Navigationsdaten- und Code-Quelle sind so allgemein, dass verschiedene Laufzeit-Varianten erzeugt werden können. Durch unterschiedliche Export-Konfigurationen von Schritt (2) entstehen so unterschiedliche Paare von Laufzeit-Code und Navigationsdaten. Wichtig ist hierbei, dass diese zueinander passen müssen.

2.1 Übernahme aus dem allgemeinen Geodatenbestand

Die Navigationsdaten werden aus dem allgemeinen Datenbestand von Open Street Map (OSM) [OSM, Ro12a] abgeleitet. OSM-Daten unterstützen nicht vordergründig die Routenplanung, d.h. entsprechende Informationen liegen verstreut im Datenbestand vor. In einem ersten Schritt werden daher alle relevanten Daten herausgerechnet:

- Anhand der Straßengeometrien wird das Wegenetz, also ein Graph aus Knoten und Kanten berechnet. Kreuzende Straßen werden in OSM nur dadurch ausgezeichnet, dass sie einen gemeinsamen Geometriepunkt besitzen. Daher kann die Topologie nur über aufwändige Überkreuzungstests abgeleitet werden.

- Die resultierende Topologie enthält noch keine Sackgassen, da die Sackgassen-Enden formal keine Kreuzung sind. Topologisch sind Sackgassen zwar nicht interessant, später möchte man aber Start oder Ziel in eine Sackgasse legen können. Daher werden entsprechende Knoten und Kanten künstlich integriert.

- Straßen werden in diesem Schritt klassifiziert. Diese Klassifikation wird später verwendet, um festzustellen, von welchem Verkehrsmittel eine Straße verwendet werden kann und wenn ja, mit welcher Durchschnittsgeschwindigkeit. Darüber hinaus werden Geschwindigkeitsbeschränkungen und Einbahnstraßenregelungen erfasst.

- Auch Informationen über die Benennungen von Straßen werden in diesem Schritt ermittelt. Das ist nicht trivial, da OSM viele Varianten für die Namensgebung unterstützt.

- Die Länge von Straßenabschnitten von Kreuzung zu Kreuzung wird berechnet. Die Durchschnittsgeschwindigkeit oder Fahrzeit können an dieser Stelle noch nicht ermittelt werden, da diese von der Export-Konfiguration (insb. dem Fortbewegungsmittel) abhängen.

Die Ausgabe dieses Schritts wird in einer SQL-Datenbank [Ro10a, Ro12c] gespeichert. Tabelle 1 zeigt ein typisches Mengengerüst für Straßendaten in Deutschland.

Tabelle 1: Mengengerüst der Straßendaten für Deutschland (Stand Juni 2012)

	Alle Straßen in Mio.	**Befahrbare Straßen in Mio.**
Kreuzungen	17,7	4,7
Straßenabschnitte (Verbindungen zwischen Kreuzungen)	23,3	10,9
Komplette Straßen (inkl. Namen)	6,1	2,3

2.2 Generieren von Laufzeit-Daten und -Code

Gemäß der Export-Konfiguration wird aus den Navigationsdaten für ein spezielles Anwendungsszenario eine Datensammlung erzeugt. Die Export-Konfiguration kann folgendes festlegen:

- Mit welchem Fahrzeug (oder mit welchen Fahrzeugen) soll die Routenplanung später erfolgen? Hiermit wird gefiltert, welche Straßentypen überhaupt exportiert werden müssen. Es werden auch nur noch Kreuzungsobjekte zwischen relevanten Straßen exportiert. Zusätzlich werden Geschwindigkeitsprofile eingetragen. Für jeden Straßenabschnitt von Kreuzung zu Kreuzung wird insbesondere berechnet, welche Zeit zum Durchfahren benötigt wird.

- Sollen Abbiegekosten berücksichtigt werden? Ist das der Fall, müssen Kreuzungskosten vorberechnet werden, indem z.B. alle möglichen Abbiegewinkel berechnet werden.

- Bestimmte Heuristiken der Wegeplanung mit A* benötigen eine Vorberechnung, z.B. bei der Verwendung so genannter ALT-Potenziale (siehe später).

- Zum schnellen späteren Zugriff müssen zahlreiche Indizes, insb. räumliche Indizes berechnet werden [Ro09, Ro11b]. Beim Export in eine Smart-Phone-Umgebung kommen beispielsweise so genannte Spatial Hash-tables zum Einsatz [Ro12b]. Damit diese effizient arbeiten, müssen alle Einträge zunächst so sortiert werden, dass räumlich nahe Einträge auch im Speicher nah beieinander liegen (Lokalisierung).

Passend zu den Daten wird Laufzeit-Code exportiert. Hierzu liegen die entsprechenden Komponenten schon in verschiedenen Export-Bibliotheken (z.B. Android, Server) vor. Zusätzlich steuert die Export-Konfiguration die Einbindung bestimmter Klassen, z.B. für den Zugriffsmechanismus auf die Laufzeit-Daten. Beim Start des Laufzeit-Codes wird überprüft, ob der Code zu den Laufzeitdaten passt. Gegebenfalls wird der Hochlauf mit einem Fehlercode beendet.

3 Laufzeitfunktionen von donavio

3.1 Die Basiskomponenten von donavio

Neben der reinen Wegeplanung gibt es im Rahmen einer Navigationsanwendung viele Funktionen. Eine Übersicht über alle relevanten Funktionen zeigt Abb. 2.

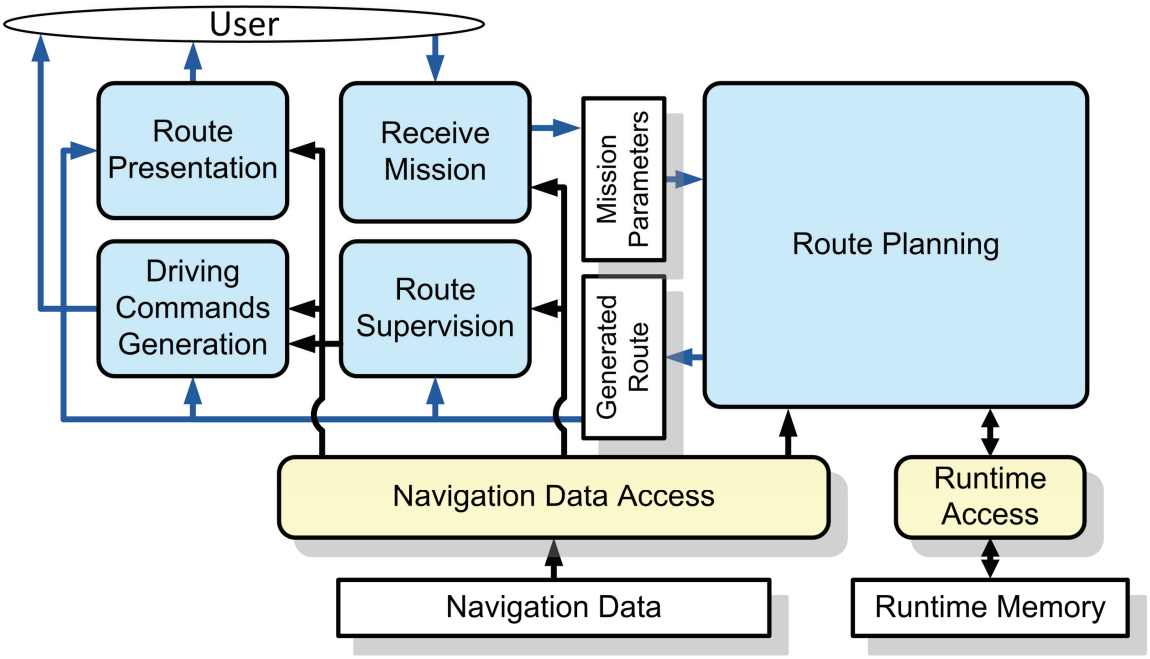

Abbildung 2: Übersicht über donavio-Laufzeitfunktionen

Der Zugriff auf die Navigationsdaten kann über verschiedene Verfahren erfolgen, die in den Access-Schichten gekapselt werden. Die funktionalen Komponenten sind:

- *Receive Mission*: Ein Navigationsauftrag muss entgegengenommen werden. Das erfordert die Suche nach Start- und Endpunkt über symbolische Namen, Adressen, Karten oder geographische Positionen.

- *Route Planning*: Ein Weg soll gemäß verschiedener Optimalitätskriterien und Fortbewegungsprofilen berechnet werden. Die Berechnung muss angemessen schnell erfolgen.

- *Route Supervision*: Schließlich muss die Fahrt überwacht werden. Es muss ermittelt werden, wann Abbiegekommandos präsentiert werden und wann eine Neuberechnung der Route veranlasst werden muss.

- *Driving Commands Generation*: Der gefundene Weg muss auf Abbiegekommandos (z.B. "In 100 Metern rechts abbiegen") abgebildet werden (Abb. 3 links oben). Das ist keineswegs trivial, da die Wahrnehmung markanter Punkte durch Menschen sich signifikant vom topologischen Straßennetz für die Routenplanung unterscheidet.

- *Route Presentation*: Zusätzlich soll der Weg meistens auf einer Karte dargestellt werden (Abb. 3 rechts unten). Das umfasst das Darstellen eines Kartenhintergrundes (u.U. gedreht) und die übersichtliche Darstellung der abzufahrenden Route. Für die aktuelle Realisierung dieser Komponenten verwenden wir die selbstentwickelte *dorenda*-Umgebung [Ro11a], die gegenüber Google Maps einige Vorteile hat.

459m fahren
am Ende der 'Obere Schanzstraße' biegen Sie links ab auf die 'Zollstraße'
971m fahren
biegen Sie links ab auf die 'Hochbergerstrasse'
801m fahren
verlassen Sie die 'Hochbergerstrasse' und biegen Sie links ab
616m fahren
fahren Sie auf die Autobahn 'A2'
291km fahren
verlassen Sie die Autobahn
733m fahren
fahren Sie auf die Autobahn 'A 5'
149km fahren
am Ende der 'A 5' fahren Sie gerade aus auf die 'A 7'
523km fahren
verlassen Sie die Autobahn
331m fahren
biegen Sie rechts ab auf die 'B 199'
894m fahren

Abbildung 3: Beispiel einer donavio-Ausgabe

Zu den fünf Komponenten aus Abb. 2 stellt donavio jeweils eine standardisierte Schnittstelle zur Verfügung. Damit sind diese Komponenten modular auswechselbar. neben einer Aufrufschnittstelle über ein Class Interface gibt es eine Web-Service-Schnittstelle. Damit können einzelne Komponenten in einer Client-Server-Umgebung ausgelagert werden. Die Ausnahme hierzu bildet die Komponente *Route Supervision*. Da diese relativ zeitkritisch ist, muss sie auf dem jeweiligen Endgerät ausgeführt werden.

3.2 Die donavio-Zugriffsmodule

Neben der algorithmischen Realisierung der fünf Basiskomponenten hat der Zugriff auf die Daten und die Verwaltung der Laufzeitdaten einen großen Einfluss auf das Laufzeitverhalten. Eine Design-Entscheidung von donavio war, die Zugriffsmodule vollständig von den Basiskomponenten zu trennen. Es gibt zwei relevante Module: *Navigation Data Access* und *Runtime Access*.

Navigation Data Access

Dieses Modul bietet Zugriffsfunktionen auf Kreuzungen, Straßenabschnitte und komplette Straßen. Kreuzungen und Straßenabschnitte beschreiben dabei die Topologie für die Routenplanung. Die kompletten Straßen beinhalten die Straßengeometrie für die Darstellung auf der Karte und symbolische Informationen für die Suche. Das Zugriffsmodul unterstützt den Zugriff per Objekt-ID sowie die räumliche und symbolische Suche. Zurzeit werden zwei Realisierungen für dieses Modul angeboten:

- *Memory Files*: Beim Programmstart werden alle Daten in den Objekt-Speicher geladen. Hierzu wurde vorher eine Datei generiert, die alle Objektstrukturen enthält. Einmal geladen, kann der Zugriff auf alle Daten mit maximaler Geschwindigkeit erfolgen, allerdings ist der Speicherbedarf zur Laufzeit sehr groß. Diese Variante eignet sich für Server, die mit entsprechendem Hauptspeicher ausgestattet sind.

- *Spatial Hashtables*: Hier wird nur der jeweils benötigte Bereich der Tabellen aus einem Sekundärspeicher in den Hauptspeicher eingelagert. Das Verfahren ähnelt dem virtuellen Speicher, allerdings wird die räumliche Lage der Einträge berücksichtigt. Damit beim Einlagern direkt viele Einträge in der näheren Umgebung vorliegen, werden die Tabellen vorher *lokalisiert* [Ro12b]. Dieses Zugriffsmodul ist für Smart-Phones geeignet, die über verhältnismäßig wenig Hauptspeicher verfügen.

Ein drittes Modul, das den Zugriff über SQL auf eine Datenbank unterstützt, wurde zwar implementiert, wird aber aufgrund der sehr schlechten Performance nicht eingesetzt.

Runtime Access

Einen großen Einfluss auf die Laufzeit und den Speicherbedarf hat die Verwaltung der Strukturen, die A* während des Ablaufs benötigt. Hierbei handelt es sich um den Knotenstatus (*OPEN* oder *CLOSED*) sowie um die Knotenwerte *f* und *g* [HNR68]. Prinzipiell könnten diese Informationen in dem topologischen Netzwerk gespeichert werden. Beim Zugriff über Spatial Hashtables liegt das topologische Netzwerk jedoch auf dem externen Speicher, kommt damit als Speicherort für die Verwaltungsstrukturen von A* aus Zeitgründen nicht in Frage. Daher werden die Verwaltungsstrukturen separate abgelegt und zu jedem Eintrag wird eine Referenz auf den Knoteneintrag in der Topologie verwaltet.

Bei einem typischen Suchlauf werden einige 10 000 bis mehrere Million Knoten besucht, dabei hat die Liste der offenen Knoten meist nicht mehr als 10 000 Einträge. Die donavio-Umgebung unterstützt derzeit folgende Zugriffsverfahren:

- *Arrays*: Für alle A*-Eigenschaften werden Arrays von der Größe der gesamten Knotenliste angelegt. Diese Arrays sind in der Regel viel zu groß, da nur ein Teil der Knoten durch A* besucht wird. Es wird aber dadurch ein sehr schneller Zugriff ermöglicht. Darüber hinaus bleibt der Speicherbedarf zur Laufzeit konstant, d.h. es geht keine Zeit für die Garbage Collection verloren. Diese Variante ist vor allem für Server geeignet.

- *DynArray*: Es wird ein einzelnes Objekt-Array von der Größe der gesamten Knotenliste angelegt. Die Referenzen in diesem Array sind zuerst alle unbelegt. Erst wenn ein Knoten durch A* besucht wurde, wird

ein Objekt-Eintrag angelegt. Der Speicherbedarf ist deutlich niedriger als bei *Arrays*, allerdings entsteht ein zusätzlicher Aufwand durch das Anlegen und Freigeben der Objekte.

- *Hashtable*: Hier werden Hashtable-Einträge mit der Knotenreferenz als *key* und einem Objekt aller Verwaltungsinformationen als *value* gespeichert. Es entsteht ein Suchaufwand zur Laufzeit, da ein Knoten erst in der Hashtable gesucht werden muss. Der Speicherbedarf kann jedoch nie größer als die Liste der besuchten Knoten werden. Diese Variante ist selbst auf Endgeräten mit wenig Hauptspeicher lauffähig.

Abb. 4 illustriert noch einmal die verschiedenen Varianten.

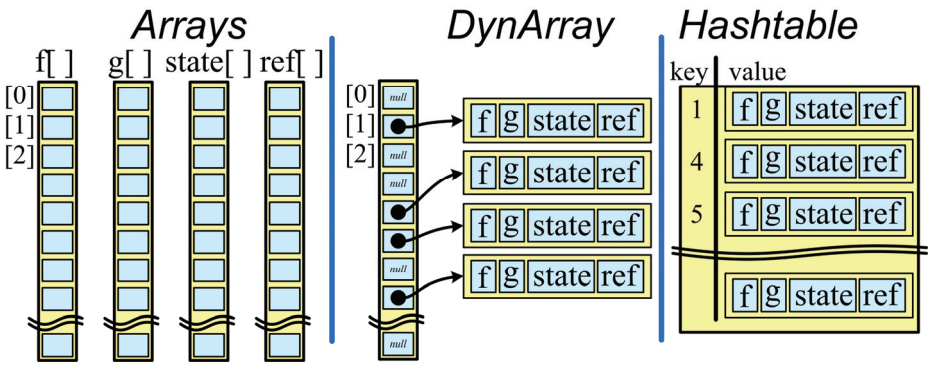

Abbildung 4: Zugriffsverfahren für die A*-Laufzeitstrukturen

3.3 Die donavio-Routenplanung

Die meisten Möglichkeiten für verschiedene Konfigurationen bietet die Komponente *Route Planning*. Daher ist diese in Abb. 5 weiter aufgeschlüsselt. Die donavio-Umgebung unterstützt derzeit verschiedene Module, die den A*-Algorithmus modifizieren oder erweitern. Insbesondere die Heuristic-Module haben im Zusammenhang mit der Laufzeit eine besondere Rolle. Ziel ist, durch geeignete Verfahren den Suchaufwand so zu reduzieren, dass die Routenplanung selbst auf leistungsschwachen Plattformen nur einige Sekunden benötigt. Dabei nimmt man suboptimale Routen in Kauf.

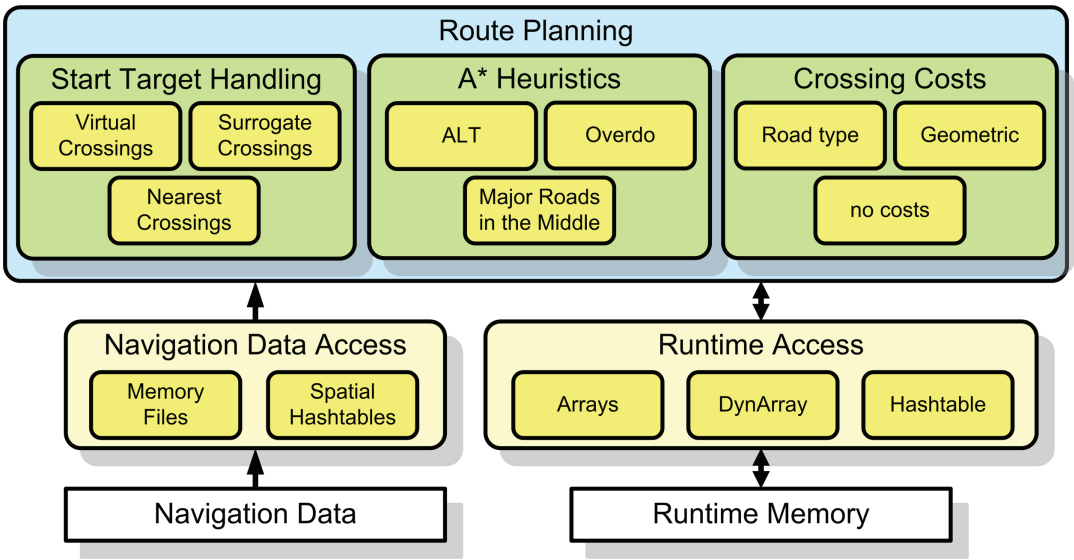

Abbildung 5: Übersicht über Funktionen der Routenplanung

Für die inneren Komponenten gibt es jeweils mehrere Möglichkeiten. Im Rahmen der Code-Bereitstellung müssen, je nach Export-Konfiguration, nicht alle Komponenten bereitgestellt werden. So kann eine konkrete Variante nur eine einzelne Sub-Komponente von A* Heuristics enthalten. Wird in der Laufzeit-Konfiguration (Abb. 1a) eine nicht existente Variante gewählt, gibt es zur Laufzeit einen Fehler.

A* Heuristics

In der Literatur werden verschiedene Verfahren zur Optimierung von A* im Kontext der Wegeplanung vorgeschlagen. Zurzeit sind in donavio die folgenden Verfahren integriert:

- *overdo*: es wird eine nicht-optimistische Schätzung verwendet, indem die optimistische (Luftlinien-) Schätzung mit einem konstanten Faktor multipliziert wird. Hiermit wird die Schätzung in der Regel realistischer, im Grenzfall aber pessimistisch. Daher sind suboptimale Resultate möglich.

- *ALT* (*A* search, landmarks, and triangle inequality*): Zu einer kleinen Anzahl ausgewählter Landmarks werden die exakten Wege-Kosten für allen Knoten von und zu diesen Landmarks berechnet, die so genannten ALT-Potenziale [GH05]. Für die Gesamtkosten eines Knotens zu einem bestimmten Ziel ist die maximale Differenz der Potenzialwerte eine sehr gute optimistische Schätzung. Diese Variante erfordert, dass bei der Bereitstellung der Daten die ALT-Potenziale vorberechnet und exportiert werden.

- *Major Roads in the Middle*: A* verfolgt bei Überschreitung einer Entfernung vom Start oder Ziel nur noch große Straßen [SP05]. Das entspricht der typischen Erfahrung bei langen Fahrten: im "mittleren" Segment einer Route verwendet man typischerweise nur Autobahnen oder Landstraßen. Optimale Routen, die in der Routenmitte kleine Straßen verwenden, werden so aber nicht erkannt.

Weitere Module sind denkbar und können bei Bedarf integriert werden.

Start Target Handling

Oft wird ein Problem vernachlässigt: A* kann zwar eine optimale Route von Knoten zu Knoten (also von Kreuzung zu Kreuzung) berechnen – typischerweise liegen Start und Ziel aber nicht auf einer Kreuzung. Es werden derzeit folgende Varianten zur Behandlung des Problems unterstützt:

- *Nearest Crossings*: Start und Ziel werden auf die nächste *erreichbare* Kreuzung abgebildet. Dieses Verfahren löst das Problem nicht eigentlich, wird daher nur als letzte Möglichkeit angeboten.

- *Virtual Crossings*: Start und Ziel werden als zusätzliche Knoten in das Straßennetz eingefügt. Zusätzlich müssen diese Knoten über neue Kanten mit den anteiligen Kosten mit dem Straßennetz verbunden werden. Diese Variante kann nur eingesetzt werden, wenn das Zugriffsmodul auf die Navigationsdaten eine Erweiterung der Topologie erlaubt. Spatial Hashtables erlauben dies nicht, da die Daten optimiert exportiert wurden und nicht verändert werden können.

- *Surrogate Crossings*: Start und Ziel werden zunächst jeweils auf die nächste *erreichbare* Kreuzung abgebildet (genannt *Surrogate* Crossing). A* plant dann von Surrogate zu Surrogate. Am Ende wird eine Gesamtroute durch *Start*→*Surrogate*, *Surrogate*→*Surrogate*, *Surrogate*→*Ziel* zusammengesetzt. Hierbei müssen ggfs. doppelte Wege abgezogen werden. Es entstehen zahlreiche Sonderfälle, wenn beispielsweise Start und Ziel auf Einbahnstraßen liegen oder gar auf derselben Straße. Darüber hinaus können suboptimale Wege entstehen, wenn eine theoretisch optimale Route nicht beide Surrogate Crossings enthält.

Das Verfahren Surrogate Crossings ist noch einmal in Abb. 6 illustriert.

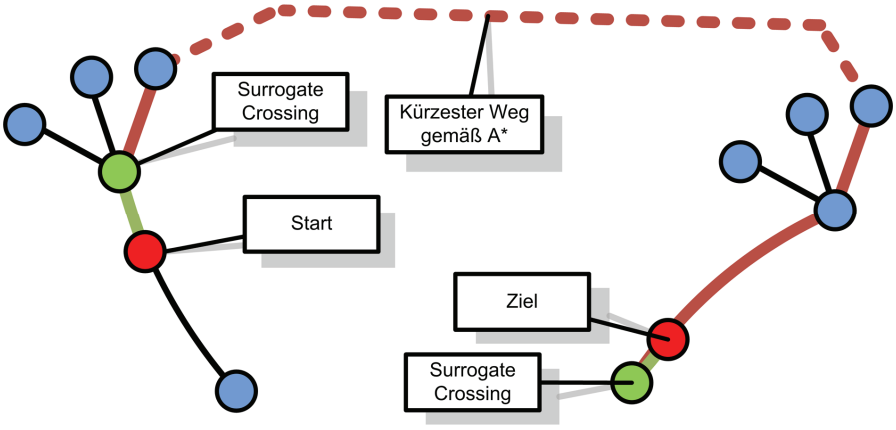

Abbildung 6: Das Surrogate-Crossing-Verfahren

Crossing Costs

In der Realität kosten Abbiegevorgänge Zeit. Eine Wegesuche in Graphen berücksichtigt aber erst einmal nur Kantenkosten. Wünschenswert sind daher Verfahren, die auch die Kosten von Abbiegungen berücksichtigt. Es werden derzeit folgende Varianten unterstützt:

- *no costs*: Die Zeit für das Abbiegen wird nicht berücksichtigt.
- *Road Type*: Es werden Kosten geschätzt, die daraus resultieren, dass man eine Straße verlässt und auf eine andere Straße fährt. Die Kosten berücksichtigen dabei nur den Übergang der Straßentypen (z.B. Autobahn→Ausfahrt).
- *Geometric*: Zusätzlich zu den geschätzten Kosten durch den Übergang zwischen Straßen wird der Abbiegewinkel berücksichtigt. Für scharfes Abbiegen wird daher eine längere Fahrzeit erwartet. Dieses Modul erfordert, dass bei der Bereitstellung der Navigationsdaten die Winkel berechnet werden, unter der eine Straße auf eine Kreuzung trifft.

Auch hier sind weitere Module denkbar. So könnte ermittelt werden, ob eine Kreuzung eine Ampel besitzt, oder ob man über eine Vorfahrt-Achten-Straße einmündet.

4 Die softwareseitige Unterstützung der Modularisierung

Ziel der Modularisierung ist, dass verschiedene Varianten ausprobiert werden können und dass verschiedene Zerlegungen zwischen Client und Server eingesetzt werden können. Daher erfüllen alle Basismodule (Abb. 2) eine objektorientierte Aufrufschnittstelle und haben (bis auf das Modul *Route Supervision*) zusätzlich eine Web-Service-Schnittstelle.

Die inneren Module von *Route Planning* sowie die Zugriffsmodule werden nur lokal aufgerufen, deshalb wurden keine Web-Service-Schnittstellen vorgesehen. Man kann hier zwei Arten von Modulen unterschieden:

1. *Start Target Handling* und *Crossing Costs*: Diese binden sich über eine klassische objektorientierte Aufrufschnittstelle in die Routenplanung ein. In den Laufzeit-Code werden alle Klassen integriert, die gemäß der Export-Konfiguration später notwendig sind. Wurde in der Export-Konfiguration beispielsweise die Vorberechnung von Kreuzungswinkeln deaktiviert, wird für das Modul *Geometric Crossing Costs* kein Code exportiert. Die eigentliche Auswahl zur Laufzeit wird über den Aufruf der Routenplanung definiert.

2. *Zugriffmodule* und *A* Heuristics*: Diese Module zeichnen sich durch folgende Eigenschaften aus:

- Sie werden extrem häufig aufgerufen (viele Millionen Mal), der eigentlich ausgeführte Code ist demgegenüber aber sehr klein.

- Eine Modellierung über eine Objektschnittstelle würde einen erheblichen Laufzeitaufwand produzieren. Letztlich kann man zwar jede Funktionalität über ein System von Klassen darstellen, der Code muss aber entsprechend zerlegt werden. Darüber hinaus produzieren die Parameterübergabe, Rückgabewerte und der Objektaufruf selbst einen gewissen Aufwand.

Da als Zielplattformen auch solche mit geringerer Prozessorleistung in Frage kommen sollen, werden diese Module über *Macro-Preprocessing* eingebunden, d.h. konkret:

- Es gibt Code, der außerhalb einer Klasse als Code-Fragment textuell vorliegt.

- In der Routenplanung werden diese Code-Fragmente je nach Modul eingeblendet. Die Parameterübergabe findet zur Compile-Zeit statt. Da Code kopiert wird, gibt es formal auch keinen Aufruf zur Laufzeit.

Als Beispiel: Die Variante *Arrays* des Moduls *Runtime Access* greift auf die Knoten-Eigenschaften über Felder zu, z.B. mit **`g[index]`** auf die bisherigen Kosten vom Start zum Knoten. Selbstverständlich könnte dieser Aufruf über eine Objektschnittstelle als Methode

```
public float getG(int index) { return g[index]; }
```

modelliert werden. Damit würde aber für jeden Aufruf zusätzlicher Code ausgeführt werden. Über Macro-Preprocessing wird der Array-Zugriff zur Compile-Zeit an die entsprechenden Stellen kopiert. Noch größer ist der Vorteil bei der Runtime Access-Variante *Hashtable*. Hier zerfällt der Zugriff auf den Knotenstatus in zwei Teile:

- Über **`astar_state=astar_hash.get(index)`** beschafft man sich die Referenz auf das Knotenobjekt.

- Über **`astar_state.g`**, **`astar_state.f`** etc. greift man auf die Eigenschaften zu.

Bei einer klassischen objektorientierten Zerlegung würde man innerhalb der Methoden **`getG`**, **`getF`** das Knotenobjekt **`astar_state`** immer wieder aus der Hashtable lesen. Wenn man die Reihenfolge der Zugriffe nicht kennt, könnte das Zugriffsobjekt sonst nicht gewährleisten, dass man schon vorher das richtige Statusobjekt ausgelesen hat. Stehen hinreichende Berechnungsressourcen zur Verfügung, nimmt man solche Mehrfachzugriffe in Kauf, wenn man dadurch eine robuste Verwendung gewährleisten kann. In zeitkritischen Umgebungen kann man sich unnütze Zugriffe nicht erlauben. Daher wird über Macro-Preprocessing die richtige und redundanzfreie Verwendung in den Code eingebettet.

Die donavio-Plattform ist komplett in Java entwickelt und Java kennt von Hause aus kein Macro-Preprocessing. Es kommt im Java-Design bewusst nicht vor und die Verwendung von Macros wird kontrovers diskutiert. Das Argument dagegen ist, dass man praktisch alles, was als Macros denkbar ist, auch über ein Klassensystem modellieren kann. Darüber hinaus kann man durch falschen Gebrauch von Macros die Wartung von Code signifikant erschweren. Die Entscheidung fiel dennoch zugunsten der Einbindung durch Macros aus, da der resultierende Laufzeit-Code nur noch den Code der gewählten Variante enthält, damit sehr schlank und effizient ist.

Zurzeit kann Code für die Java Standard Edition sowie für die Android-Umgebung generiert werden. Abb. 7 zeigt den Prototyp einer Android-Navigationsanwendung.

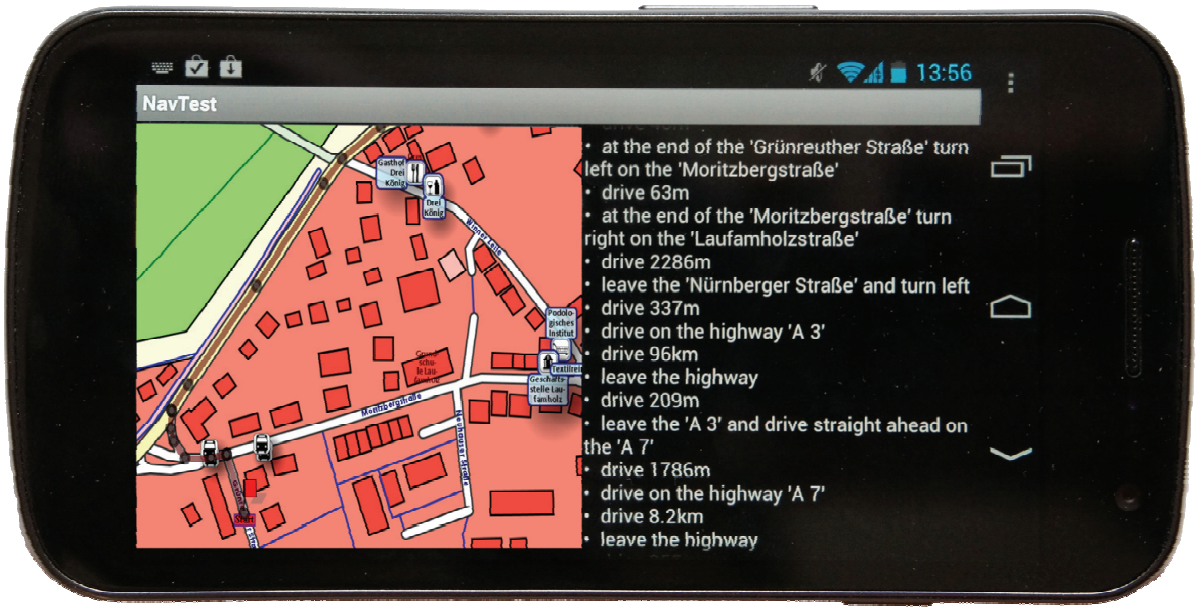

Abbildung 7: Android-Prototyp basierend auf donavio

5 Zusammenfassung

Die donavio-Umgebung stellt eine vollständige und leistungsfähige Plattform für die Generierung und Ausführung von Routen-Planungsanwendungen dar. Hierbei sind verschiedene Architekturen und Einsatzumgebungen denkbar.

Durch den modularen Aufbau können auf einfache Weise neue Verfahren getestet werden. Auch der Einsatz in Lehrveranstaltungen bietet sich an. Bisher gab es das Problem, dass man z.B. Varianten von A* in Lehrveranstaltungen realisieren konnte, diese waren aber nicht isoliert lauffähig, da die Zugriff auf Topologiedaten und die Präsentation der Routen eine Lehrveranstaltung überfrachtet hätten. Mit donavio können bestimmte Komponenten in Praktika realisiert und in der gesamten Navigationsanwendung zum Einsatz gebracht werden.

Literaturverzeichnis

[GH05] Goldberg, A. V.; Harrelson, C.: 2005, Computing the Shortest Path: A* Search Meets Graph Theory. In Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, 2005, 156–165

[HNR68] Hart P. E., Nilsson N. J., Raphael B. 1968, A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics SSC4 (2), 100–107

[NZ08] Neis, P.; Zipf, A.: LBS_2.0 - Realisierung von Location Based Services mit user-generated, collaborative erhobenen freien Geodata, in: J. Roth (Hrsg.): 5. GI/ITG KuVS Fachgespräch Ortsbezogene Anwendungen und Dienste, 4.-5. September 2008, Nürnberg. Schriftenreihe der Georg-Simon-Ohm-Hochschule Nürnberg Nr. 42, 111-115

[OSM] Open Street Map, http://www.openstreetmap.org/

[Ro09] Roth, J.: The Extended Split Index to Efficiently Store and Retrieve Spatial Data With Standard Databases, IADIS International Conference Applied Computing 2009, Rom (Italien), 19.-21. Nov. 2009, Vol. I, 2009, 85-92

[Ro10a] Roth, J.: Die HomeRun-Plattform für ortsbezogene Dienste außerhalb des Massenmarktes, in Zipf A., Lanig S., Bauer M. (Hrsg.) 6. GI/ITG KuVS Fachgespräch "Ortsbezogene Anwendungen und Dienste", Heidelberger Geographische Bausteine Heft 18, 2010, 1-9

[Ro10b] Roth, J.: Übernahme von Geodatenbeständen aus Open Street Map und Bereitstellung einer effizienten Zugriffsmöglichkeit für ortsbezogene Dienste, Praxis der Informationsverarbeitung und Kommunikation (PIK), Vol. 13, Heft 4, 2010, 268-277

[Ro11a] Roth, J.: Der Einsatz von OpenStreetMap-Daten in der akademischen Informatik-Ausbildung, in Küpper A., Roth J. (Hrsg.): 7. GI/ITG KuVS Fachgespräch "Ortsbezogene Anwendungen und Dienste", Logos-Verlag, 2011, 65-72

[Ro11b] Roth, J.: Moving Geo Databases to Smart Phones – An Approach for Offline Location-based Applications, Innovative Internet Computing Systems (I2CS), Berlin, 15.-17. Juni 2011, GI Lecture Notes in Informatics, Vol. P-186, 2011, 228-238

[Ro12a] Roth J. 2012, Managing Geo Data – Usage of Open Street Map for Own Services and Applications, Tutorial on the TPMC 2012, Lisbon, available at http://www.wireless-earth.org/homerun.html

[Ro12b] Roth, J.: A Spatial Hashtable Optimized for Mobile Storage on Smart Phones, 9. GI/ITG KuVS Fachgespräch "Ortsbezogene Anwendungen und Dienste", 2012

[Ro12c] Roth, J.: Geo-Tabellen im HomeRun-Projekt, Internes technisches Papier, Ohm-Hochschule Nürnberg, 2012

[SP05] Sanders, P.; Schultes, D.: Highway hierarchies hasten exact shortest path queries, in Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS 3669, Springer, Heidelberg, 2005, 568–579