1 Grobentwurf

1.1 Überblick

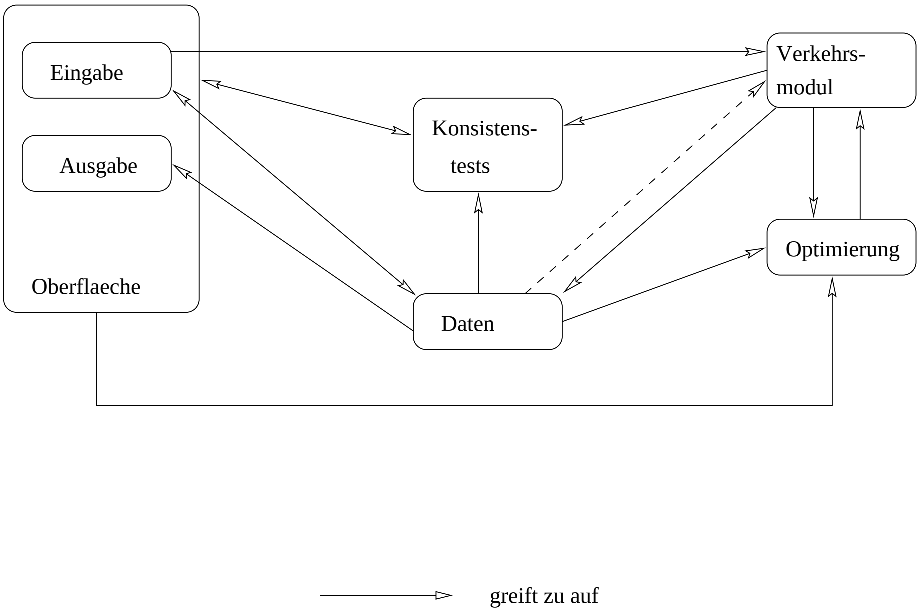


Abbildung 1: Module des Systems

Abbildung 1.1 zeigt die wesentlichen Funktionseinheiten des Systems und deren Beziehungen untereinander.

1.2 Benutzungsoberfläche

Die folgende Auflistung stellt die Menüstruktur des Systems dar, die dem Benutzer als Menüzeile mit Pull-down-Menüs angezeigt wird.

- Daten
  - Laden
  - Speichern
- Kunden
  - neu
  - ändern
  - löschen
- Touren
  - Touren
  - Untertouren

  - Fahrten
  - Optimierung
- Analyse
  - Auslastung(Mitarbeiter)
  - Einsatzzeiten(Fahrzeug)
  - Leerfahrten
  - Ausfallzeiten
    * Mitarbeiter
    * Fahrzeuge
  - Fahrzeugbesetzung
  - Ressourcenbindung
  - Kosistenztests
- Ausgabe
  - Mitarbeiterliste
  - Kundenliste
  - Fahrzeugliste
  - Tourplan
  - Untertourplan
  - Streckenplan
  - Fahrten
  - Dienstpläne
- Ressourcen
  - Mitarbeiter
  - Fahrzeuge
- Einstellungen
  - Fahrzeugtypen
  - Fahrzeugkonfigurationen
  - Mitarbeiterqualifikationen
  - Sitzhilfen
  - Essensarten
  - Feiertage
  - Streckenparameter

Daten Unter diesem Menüpunkt liegen die einzelnen Operationen, um Szenarien (siehe Entwurf Datenhaltung) zu laden und zu speichern.

Kunden Hier können Kunden neu eingegeben, geändert oder gelöscht werden. Will man einen Dienstwunsch eines Kunden ändern, löschen oder neu erstellen, so geschieht dies ebenfalls über die Kunden-ändern-Option.

Touren Dieses Menü untergliedert sich in vier Teile :

- Touren: Zum manuellen Eingeben neuer oder zum Ändern bereits bestehender Touren. Die Stationenreihenfolge kann jedoch nicht festgelegt werden, da diese erst für die Untertour festgelegt wird.
- Untertouren: Hier kann nur noch die Stationenreihenfolge aus den einzelnen Dienstwünschen der Untertour geändert werden. Weitergehende Änderungen müssen in der Tour vorgenommen werden.
- Fahrten: In diesem Menüpunkt lassen sich neue Mitarbeiter oder neue Fahrzeuge festlegen, die restlichen Rahmenbedingungen sind durch die Untertouren und die Touren festgelegt und müssen ggf. dort geändert werden.
- Optimierung: Die Optimierung erfolgt entweder für ausgewählte Touren oder für Dienstarten. Eine globale Optimierung erfolgt durch das mehrmalige Starten der Optimierungen für die einzelnen Dienstarten.

Analyse Für das System gibt es verschiedene Analyseansätze:

- Auslastung der Mitarbeiter und der Fahrzeuge. Für die Auslastung des Mitarbeiters wird die Sollarbeitszeit mit der tatsächlichen Arbeitszeit verglichen (z.B. durch Nachschauen in den Dienstplänen). Es ist dabei jedoch darauf zu achten, daß die Arbeitszeit des Mitarbeiters nicht zu sehr aufgesplittet wird, d.h. daß der Mitarbeiter nur zu kurzen Diensten, die über den ganzen Tag verteilt stattfinden, eingeteilt wird, ohne dadurch auf seine Sollarbeitszeit zu kommen. Dies würde nämlich zu einer Verzerrung der Mitarbeiterauslastung führen.
- Einsatzzeiten der einzelnen Fahrzeuge, wie lang ist das Fahrzeug pro Tag oder pro Woche unterwegs.
- Anzahl der Leerfahrten. Darstellung des Verhältnisses von Dienstfahrten, d.h. denjenigen Fahrten mit denen Geld verdient wird, zu den Fahrten, die für das DRK „nur Kosten“ verursachen.
- Ausfallzeiten. Diejenigen Zeiten, in denen Fahrzeuge und Mitarbeiter nicht für das DRK verfügbar bzw. einsetzbar sind. Bei Fahrzeugen z.B. durch Unfälle, Kundendiensttermine o.ä. Bei den Mitarbeitern werden im wesentlichen nur die Krankheiten als Ausfallzeiten gezählt.
- Fahrzeugbesetzung. Diese Maßzahl ist nur für diejenigen Dienste interessant, bei denen Personen transportiert werden. Sie gibt das Verhältnis vorhandener Sitzplätze (auch Rollstuhlplätze) zu den freien, nicht besetzten Plätzen an.
- Die Ressourcenbindung gibt an, zu wievielen Diensten ein Mitarbeiter oder ein Fahrzeug eingeteilt ist. Die Einteilung erfolgt nach Dienstarten getrennt. Diese Analysefunktion soll dem Planer einen groben Überblick

über den Ressourcenverbrauch der einzelnen Dienstarten ermöglichen, um z.B. bei der Aufnahme neuer Dienste den zusätzlichen Ressourcenbedarf überschlagen zu können.

Ausgabe Unter diesem Menüpunkt können alle für den Planer wichtigen Ausgaben entweder auf dem Bildschirm dargestellt oder auf dem Drucker ausgegeben werden. Dabei handelt es sich um folgende Punkte:

- Kundenliste, Mitarbeiterliste und Fahrzeugliste. Hier sind eine komplette Ausgabe und eine Ausgabe von Teillisten möglich.
- Tourplan. Ausgabe aller in der Tour festgelegten Eigenschaften (Zeitrahmen, Stationen, Kontaktpersonen ...)
- Untertourplan. Der Unterschied zum Tourplan liegt darin, daß die Stationen in eine Reihenfolge gebracht wurden, und daß aus dem Zeitrahmen konkretere Termine eingetragen wurden.
- Streckenplan. Zusammenfassung von Tour- und Untertourplan, die von den Mitarbeitern als Information mitgenommen werden kann.
- Fahrten. Endgültige Festlegung einer Untertour auf ein fixes Datum mit fixen Mitarbeiter und Fahrzeug.
- Dienstpläne. Es gibt zwei Arten von Dienstplänen:
  - einen für den jeweiligen Mitarbeiter, der anzeigt, wann dieser in einem festgelegten Zeitraum für welchen Dienst eingeteilt ist.
  - eine Gesamtübersicht, die dem Planer eine Zusammenfassung der Einzeldienstpläne bietet

Ressourcen Neueingabe, ändern und löschen von Mitarbeitern und Fahrzeugen.

Einstellungen Hier kann der Planer die im System frei definierbaren Bezeichnungen, wie z.B. Qualifikationen, Essensarten, Fahrzeugtyp ... , eingeben.

1.3 Datenhaltung

Java bietet die Möglichkeit der *Objekt-Serialisation*, mit der ein Objekt inklusive aller Attribute und Referenzen gespeichert werden kann, ohne daß der Programmierer sich um die Referenzen kümmern muß. Diese Möglichkeit werden wir nutzen, um alle Daten (Objekte) von TRO in einer Datei zu speichern.

Da neben dem aktuellen Datenbestand auch verschiedene Szenarien möglich sein sollen, stellt sich die Frage, wie die daraus resultierenden Datenbestände bei der Speicherung berücksichtigt werden können. In der Projektgruppe wurden zwei Möglichkeiten angesprochen:

1. Jede Abweichung vom Echtbestand stellt ein eigenes Szenario dar und wird in einer eigenen Datei gespeichert. Damit ist es gleichgültig, ob Stammdaten oder Tourdaten geändert werden, in jedem Fall muß ein neues Szenario angelegt werden (siehe Abbildung 2).

2. Es gibt 2 Arten von Szenarien:

   (a) **Tourszenarien:** Dabei werden bei gleichbleibenden Stammdaten verschiedene Tourkonstellationen ausprobiert. Da diese logisch zu den gleichen Stammdaten gehören, werden alle Tourszenarien samt den Stammdaten in einer Datei gespeichert.

   (b) **Allgemeine Szenarien:** Hierbei können auch Stammdaten geändert werden, um beispielsweise zu testen, wie sich eine Verringerung der Fahrzeugflotte auswirken würde. Diese allgemeinen Szenarien werden jeweils in einer eigenen Datei gespeichert und im folgenden kurz als **Szenarien** bezeichnet.

   (siehe Abbildung 3)

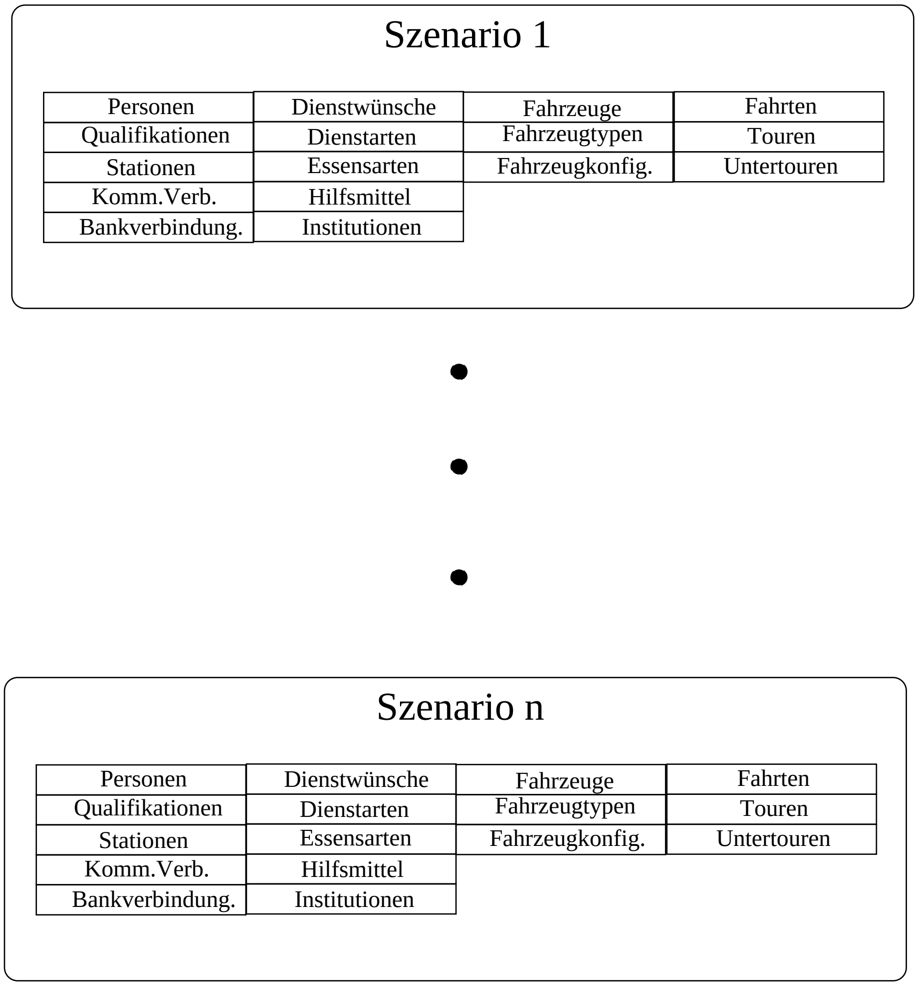


Abbildung 2: Aufbau aus Einzelszenarien

Die Projektgruppe entschied sich für die zweite Variante, da angenommen wurde, daß öfter Tourszenarien zu gleichen Stammdaten erstellt werden (als

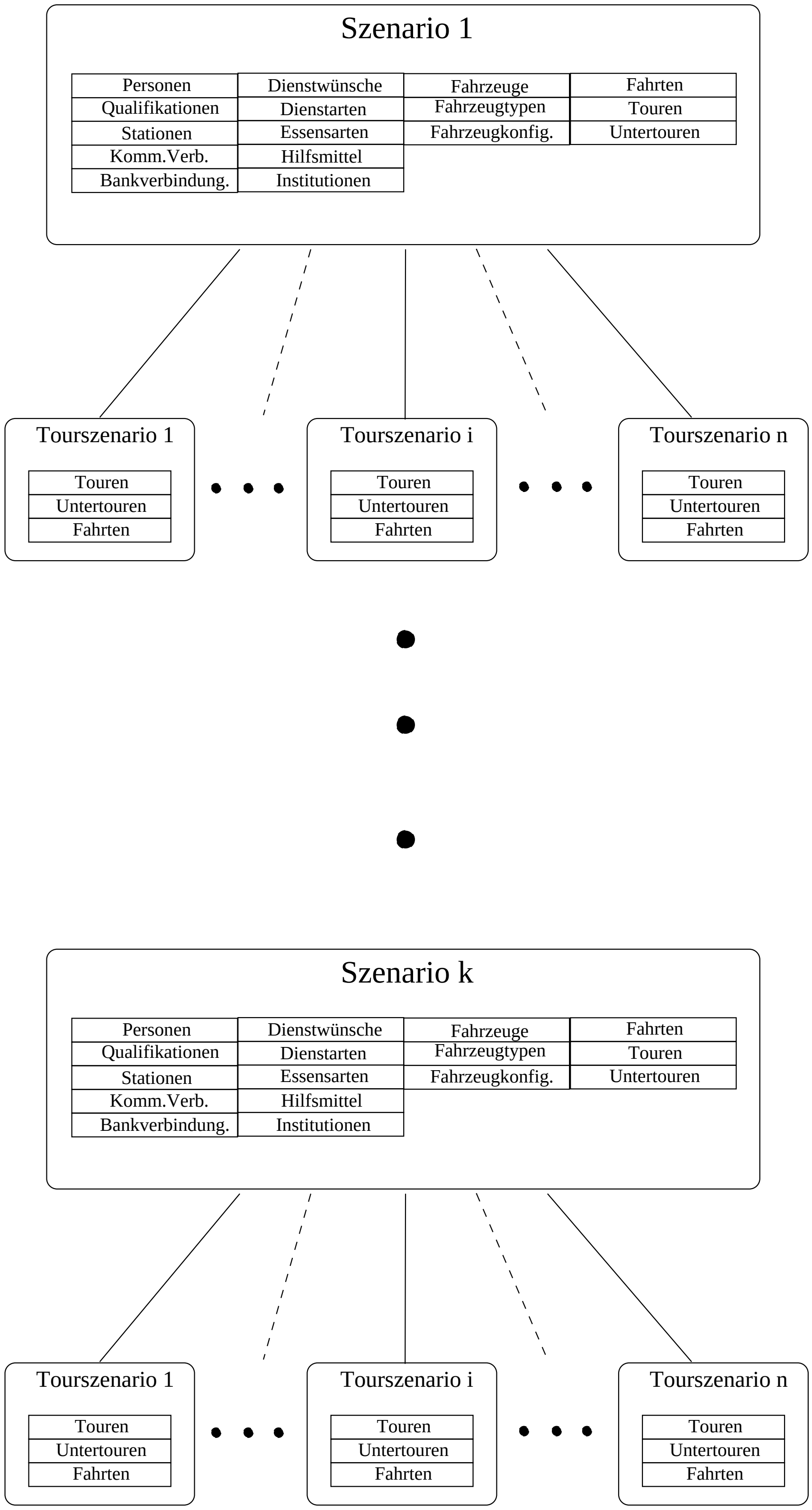


Abbildung 3: Aufbau mit Tourszenarien

manuelle Umsetzungs-/Optimierungsversuche). Das Problem bei der ersten Variante sind die Änderungen an den Stammdaten. Diese müssen in allen Szenarien nachgeführt werden, bei denen nur verschiedene Tourzusammenstellungen getestet werden, da diese Szenarien sonst nutzlos sind. Wenn aber jedes solche Szenario eine eigene Datei ist, ist eine automatische Nachführung praktisch unmöglich, da die Dateien auch auf beliebigen Wechselmedien gespeichert sein können. Bei der ersten Variante obliegt daher die Konsistenz der Szenarien dem Benutzer, der selbst entscheiden muß, bei welchen Szenarien Stammdaten nachgeführt werden müssen (z.B. wenn während der Tourumstellung ein Kunde anruft und ein neuer Dienstwunsch angelegt werden muß). Die zweite Variante gewährleistet dagegen, daß alle Tourszenarien auf einer gemeinsamen Stammdatenbasis arbeiten und damit garantiert vergleichbar sind, ohne daß der Benutzer wissen muß, welche Szenarien eigentlich die gleichen Stammdaten aufweisen müssen. Mit dieser Variante ist es daher auch leicht möglich, alternative Tourpläne parallel zu führen.

Um den Echtbestand von den übrigen Szenarien unterscheiden zu können, gibt es ein ausgezeichnetes **Masterszenario** und **Tourmasterszenario**. Für das Masterszenario gelten folgende Regeln:

- Es gibt genau ein Masterszenario, damit existiert also immer mindestens ein Szenario.
- Das Masterszenario kann nicht gelöscht werden, auch dann nicht, wenn andere Szenarien bestehen.
- Jedes bestehende Szenario kann zum Masterszenario gemacht werden.

Für das Tourmasterszenario gelten analoge Regeln.

1.3.1 Schnittstelle

Die Schnittstelle zur Datenhaltung kann vom Benutzer über das Menü aufgerufen werden und bietet folgende Funktionen:

ladeSzenario(Name) Löscht das im Speicher befindliche Szenario, lädt das angegebene in den Speicher und setzt dessen Veränderungsflag auf false. Wurden am alten Szenario Veränderungen vorgenommen, wird darauf hingewiesen und die Möglichkeit gegeben, die Änderungen zu speichern.

speichereSzenario(Name) Speichert das im Speicher befindliche Szenario unter dem angegebenen Namen und setzt das Veränderungsflag auf false. Stimmt der übergebene Name nicht mit dem des Szenarios überein (speichern als), wird der übergebene Name als neuer Szenarioname übernommen.

löscheSzenario(Name) Löscht das Szenario mit dem übergebenen Namen nach einer Sicherheitsabfrage von der Platte. Das Masterszenario darf nicht gelöscht werden!

neuesSzenario(Name) Legt ein leeres Szenario an.

werdeMaster(Name) Macht das angegebene Szenario nach einer Sicherheitsabfrage zum Masterszenario.

Um zu gewährleisten, daß es nur ein Masterszenario gibt, muß die Information, ob ein Szenario ein Masterszenario ist, außerhalb der Szenarien gespeichert werden (wie soll ein Mastermerkmal eines Szenarios auf Diskette gelöscht werden, wenn das System nicht weiß, auf welcher Diskette sich das Masterszenario befindet?). Diese Information kann beispielsweise in einer .ini-Datei abgelegt werden. Ferner wäre es wünschenswert, das oder die letzen x Masterszenarien für ein UNDO bereitzuhalten.

istMaster(Name) : boolean Gibt true zurück, wenn das übergebene Szenario ein Masterszenario ist. Andernfalls wird false zurückgegeben.

aktiviereTourszenario(Name) Aktiviert ein Tourszenario, das innerhalb des geladenen Szenarios zur Verfügung steht.

löscheTourszenario(Name) Löscht ein Tourszenario innerhalb des geladenen Szenarios nach einer Sicherheitsabfrage permanent. Das Tourmasterszenario darf nicht gelöscht werden.

neuesTourszenario(Name) Legt ein neues Tourszenario innerhalb des geladenen Szenarios an.

werdeTourMaster(Name) Macht das angegeben Tourszenario zum Master-Tourszenario.

istTourMaster(Name) : boolean Gibt true zurück, wenn das übergebene Tourszenario ein Master-Tourszenario ist. Andernfalls wird false zurückgegeben.

Da innerhalb eines Szenarios alle Tourszenarien bekannt sind, kann dies durch Setzen eines Flags im Tourszenario-Objekt geschehen.

1.4 Verkehrsmodul

Das Verkehrsmodul dient der Ermittlung von Entfernungen zwischen Anbindungspunkten. Ein Anbindungspunkt ist ein dem Verkehrstool bekannter Ort, der in der Nähe der gewünschten Stationen liegt. Um Speicherplatz zu sparen und die Anzahl der Anfragen an Map&Guide zu verringern, wird für beide Richtungen nur eine Entfernung (in Kilometern) und eine Fahrzeit (in Minuten) verwaltet. Es wird also davon ausgegangen, daß die Fahrzeiten und die Entfernungen etwa symmetrisch sind.

Im folgenden werden zunächst die vom Verkehrsmodul bereitgestellten Methoden und dann die verwendeten Klassen beschrieben.

sucheMoeglicheAnbindungsPunkte (StationsBeschreibung) : StationsBeschreibung []. Gibt eine Liste mit Beschreibungen von möglichen Anbindungspunkten für die angegebene Station zurück. In der Regel wird bei einer vollständig angegebenen Adresse nur eine Beschreibung zurückgegeben. Kann die angegebene Station nicht eingebunden werden, wird eine leere Liste zurückgegeben. Für die Beschreibungen können Anbindungspunkte erstellt werden.

erstelleAnbindungsPunkt (StationsBeschreibung) : AnbindungsPunkt Erstellt einen Anbindungspunkt für die angegebene Station. Die Beschreibung sollte eine der Beschreibungen sein, die von sucheMöglicheAnbindungsPunkte zurückgegeben wurde.

entferneAnbindungsPunkt (AnbindungsPunkt) . Muß immer aufgerufen werden, wenn ein Anbindungspunkt nicht mehr benötigt wird.

fahrdauer (AnbindungsPunkt, AnbindungsPunkt) : integer . Gibt die Fahrdauer in Minuten zwischen den beiden Anbindungspunkten an. Unter Umständen wird dazu Map&Guide gestartet.

entfernung (AnbindungsPunkt, AnbindungsPunkt) : integer . Gibt die Entfernung zwischen den beiden Anbindungspunkten in Kilometern zurück. Diese Entfernung ist weniger zur Optimierung gedacht, sondern vielmehr um entfernungsbezogene Größen zu berechnen (Kilometerpauschalen). Auch hierbei kann es notwendig sein, Map&Guide zu starten.

setzeWerte (AnbindungsPunkt, AnbindungsPunkt, integer, integer) . Setzt die Fahrdauer in Minuten und die Entfernung in Kilometern zwischen den angegebenen Anbindungspunkten auf die gegebenen Werte.

bereiteVor (AnbindungsPunkt[]) . Hiermit wird angekündigt, daß alle Fahrzeiten oder Entfernungen zwischen den Anbindungspunkten benötigt werden.

bereiteVor (Strecke[]) . Kündigt dem Verkehrsmodul an, daß Fahrzeiten oder Entfernungen der angegebenen Strecken benötigt werden.

initialisieren (String, String, String) . Muß vor den anderen Methoden aufgerufen werden (beim Starten des Systems). Die übergebenen Strings sind der Reihe nach der Name des Verzeichnisses, in dem Map&Guide installiert wurde, der Name des Verzeichnisses, in dem die Auftrags-Dateien abgelegt werden sollen, und der Name der Datei, in der die Tabellen gespeichert werden.

speichern () . Speichert die beiden Tabellen. Muß vor dem Beenden des Systems aufgerufen werden.

zeigeWegAn (AnbindungsPunkt[]) . Zeigt die ermittelte Route durch die angegebenen Anbindungspunkte in Map&Guide an.

1.4.1 AnbindungsPunkt

Diese Klasse repräsentiert Punkte auf der Strassenkarte von Map&Guide. Sie bietet nach außen keine Attribute oder Methoden an.

1.4.2 StationsBeschreibung

PLZ : String . Die Postleitzahl des Ortes.

Ortsname : String .

Strasse : String . Straßenname und Hausnummer.

1.4.3 Strecke

von : AnbindungsPunkt .

nach : AnbindungsPunkt .

1.5 Konsistenztests

Zwischen den Objekten der realen Welt, die das Transportoptimierungssystem auf Datenstrukturen abbildet, bestehen vielerlei Beziehungen, die nicht alle implizit durch die Datenmodellierung gewährleistet werden können. Um dennoch sicherzustellen, daß die Daten ein sinnvolles, in sich konsistentes Abbild der Realität sind, müssen explizite Testfunktionen implementiert werden. Diese werden entweder automatisch aufgerufen, um geplante Datenmanipulationen zu überprüfen, oder bei Bedarf, um z.B. die Vollständigkeit oder Korrektheit einer Menge von Benutzereingaben festzustellen.

1.5.1 Integritätsbedingungen

Hierunter fallen die „scharfen" Bedingungen, die für die logische Konsistenz der Daten immer gelten müssen. Werden solche Bedingungen verletzt, haben die Daten i.A. keine sinnvolle Entsprechung in der Realität mehr. Deshalb können die Integritätstests auch nicht vom Benutzer aufgerufen werden, sondern werden vor Datenmanipulationen aktiviert, um sicherzustellen, daß die Daten in einen konsistenten Zustand überführt werden.

Referentielle Integrität Verweise zwischen Objekten müssen eindeutig sein, d.h. das Ziel des Verweises muß existieren und eindeutig identifizierbar sein. Dafür müssen verschiedene Arten von Bedingungen überprüft werden.

- Alle Schlüsselattribute müssen eindeutige Werte haben (z.B. darf kein Kunde mit einer schon bestehenden Kundennummer eingefügt werden). Dies muß sowohl beim Einfügen eines Datensatzes als auch beim Ändern sichergestellt werden.

- Alle Fremdschlüsselattribute dürfen nur auf Objekte verweisen, die tatsächlich existieren. Dies muß ebenfalls beim Einfügen und Ändern der Objekte sichergestellt werden, die den Fremdschlüssel besitzen. Außerdem muß das Löschen von Objekten geprüft werden, auf die der Fremdschlüssel verweist.

  Hier gibt es drei Möglichkeiten zur Wahrung der Integrität (die je nach Rolle des Datenobjekts in der realen Welt passend gewählt werden müssen):

  1. Kaskadieren von Änderungen an Primärschlüsseln auf alle Objekte, die per Fremdschlüssel darauf verweisen (z.B. Ändern einer Kundennummer ⇒ Ändern der Kundennummer in allen Dienstwünschen und Touren, die auf diesen Kunden verweisen)
  2. *NULL*-setzen der Fremdschlüssel. Damit wird der Verweis gelöscht.
  3. Verbieten der Schlüsseländerung bzw. -entfernung.

- Gewisse Attribute müssen immer einen Wert haben (Verbot von *NULL*-Werten, z.B. muß jeder Dienstwunsch einem Kunden zuzuordnen sein). Dies muß bei Eingabe und Änderung der betreffenden Objekte beachtet werden.

Welche Attribute Schlüssel sind und welche immer einen Wert aufweisen müssen, ist bereits im Datenmodell (**??**) angegeben.

Statische Integritätsbedingungen *Statisch* sind alle Bedingungen, die sich auf momentane Werte beziehen. Dies sind sowohl Vorgaben für Wertebereiche (wie z.B. Minimal- und Maximalwerte) als auch einzuhaltende Beziehungen zwischen verschiedenen Werten (z.B. Startzeit $<$ Zielzeit).

- Vorlieben und Abneigungen eines Kunden müssen disjunkt sein.
- (Startdatum, Startzeit) $<$ (Zieldatum, Zielzeit) für alle Termine, Untertouren etc.
- Kein Kunde darf länger als eine definierte Maximalzeit im Fahrzeug verbringen

1.5.2 Vorgaben

Hiermit sind die Beziehungen oder Wertschranken gemeint, die nicht aus logischen oder physikalischen Gesetzmäßigkeiten folgen, sondern die Menschen von der zu modellierenden Realität fordern. Solche Vorgaben sollten normalerweise erfüllt werden, können aber im Einzelfall (durch explizite Vorgabe des Benutzers) außer Kraft gesetzt werden.

Diese Konsistenztest können deshalb auf zwei unterschiedliche Arten aktiviert werden:

Zum einen wird nach entsprechender Eingabe oder Änderung von Daten durch den Benutzer der passende Test durchgeführt, um auf (möglicherweise ungewollte) Inkonsistenzen in der Eingabe hinzuweisen. Der Benutzer kann jedoch bewußt trotzdem den inkonsistenten Zustand akzeptieren.

Zum anderen können diese Tests auch explizit über die Benutzungsoberfläche aufgerufen werden, um einen nachträglichen Überblick über solche bewußt akzeptierten Inkonsistenzen (und ggf. deren Folgen) zu bekommen.

Mitarbeiter

- Ist der Mitarbeiter immer nur für eine Untertour/Fahrt zur selben Zeit eingesetzt?
- Reichen die Qualifikationen für alle zugeteilten Dienste?
- Hält sich die Arbeitszeit innerhalb gewisser (definierter) Grenzen?
- Werden die Ausfallzeiten berücksichtigt?

Fahrzeuge

- Wird die maximale Fahrgastzahl auf allen Touren/Fahrten eingehalten?
- Wird immer nur eine Untertour/Fahrt zur selben Zeit verlangt?

- Werden die Ausfallzeiten berücksichtigt?

Kunden

- Finden unterschiedliche Dienstwünsche zu unterschiedlichen Zeiten statt? (Lieferdienste ausgenommen)

Dienstwünsche

- Werden alle Anforderungen (an Mitarbeiter, Fahrzeuge und Zeiten) durch die Untertour/Fahrt erfüllt?
- Sind alle Dienstwünsche eingeplant?

Fahrten

- Reichen die Mitarbeiter und Fahrzeuge aus, um aus allen Untertouren Fahrten zu bilden?

Diese Tests müssen für die langfristig geplanten Untertouren positiv ausfallen. Explizite Konsistenzeinschränkungen durch den Benutzer werden nur für Fahrten erlaubt (die ja einmalige Ereignisse darstellen).

1.6 Optimierung

Benutzt von Benutzungsoberfläche

Zugriff auf Datenobjekte und Verkehrsmodul

Folgende Optimierungsarten können explizit aufgerufen werden:

- Lokale Optimierung

  Parameter Dienstart, Menge von Dienstwünschen (mit dieser Dienstart)

  Funktion Die gegebenen Dienstwünsche zu Touren zusammenfassen

  Zugriff auf Daten (Abfrage der verfügbaren Mitarbeiter und Fahrzeuge), Verkehrsmodul (Fahrtdauern)

  Ergebnis Menge von Touren

- Inkrementelle Optimierung

  Parameter Dienstart, Menge von Touren, Menge von Dienstwünschen

  Funktion Die gegebenen Dienstwünsche auf die angegebenen und neu zu erstellende Touren verteilen

  Zugriff auf Daten (Abfrage der verfügbaren Mitarbeiter und Fahrzeuge), Verkehrsmodul (Fahrtdauern)

  Ergebnis Veränderte Menge von Touren

- Optimieren durch Umsetzen

  Parameter Dienstart, Menge von Touren

  Funktion Aufbrechen einzelner Touren und Versuch, deren Dienstwünsche in anderen Touren mit besserer Bewertung unterzubringen

Zugriff auf Daten (Abfrage der verfügbaren Mitarbeiter und Fahrzeuge), Verkehrsmodul (Fahrtdauern)

Ergebnis Gegebenenfalls veränderte Menge von Touren

- Touren wegoptimieren

  Parameter Dienstart, aufnehmende Tourmenge, aufzulösende Tourmenge

  Funktion Versuch, die angegebenen (aufzulösenden) Touren überflüssig zu machen, indem deren Dienstwünsche auf die aufnehmenden Touren verteilt werden

  Zugriff auf Verkehrsmodul (Fahrtdauern)

  Ergebnis Gegebenenfalls veränderte Menge von Touren

2 Feinentwurf

2.1 Klasse Person und zugehörige Klassen

2.1.1 Bank

- Attribute

  BLZ : String Acht Ziffern, die die Bankleitzahl darstellen.

  BankName: String Name der Bank mit obiger Bankleitzahl.

- Methoden

  leseBankname(BLZ : String) : String Gibt den Namen der Bank mit der übergebenen Bankleitzahl zurück.

2.1.2 Bankverbindung

- Attribute

  BLZ : String Acht Ziffern, die die Bankleitzzahl darstellen.

  KontoNr : String Zehn Ziffern, die eine Kontonummer bei obiger Bank darstellen.

  Einzug? : boolean Ist true, wenn eine Einzugsermächtigung vorliegt. Voreinstellung: wenn eine Bankverbindung bekannt ist: true. Ist keine Bankverbindung bekannt wird dieses Attribut nicht angezeigt!

2.1.3 Bezugsperson

Unterklasse von Person

- Attribute

  BezugsArt : String Gibt die Art des Bezuges zum Kunden an. Z.B. Familienverhältnis, Arzt, etc.

  Bemerkung : String Freier Text zum Vermerken zusätzlicher Informationen.

2.1.4 Institution

- Attribute

Name : String Name der Institution, z.B. Krankenkasse X, Sozialamt Y, Versicherung Z.

2.1.5 Kommunikationsverbindung

- Attribute

Art : String (Mußwert) Gibt die Art der Kommunikationsverbindung an (Telefon, Fax, eMail, etc.).

Eintrag : String (Mußwert) Zur Art passender Eintrag, z.B. Telefonnummer, eMail-Adresse, etc.

2.1.6 Kunde

Unterklasse von Person

- Attribute

KundenNr : String (Schlüssel) Eindeutige Nummer, welche den Kunden innerhalb des DRK identifiziert.

HausschluesselZahl : String Anzahl der Haus- bzw. Wohnungsschlüssel, die der Kunde dem DRK überlassen hat.

HausSchluesselText : String Beliebige Bemerkungen zu den Schlüsseln, beispielsweise vergebene Schlüsselnummern, für welche Tür, etc.

MaxFahrdauer : short Gibt an, wieviele Minuten ein Kunde maximal in einem Fahrzeug zubringen darf. Die globalen Obergrenzen (z.B. Schulfahrten max. 2h) bleiben davon unberührt.

Bemerkungen : String Diverse, zusätzliche Angaben zum Kunden. Diese Angaben werden bei der automatischen Planung nicht berücksichtigt, werden jedoch auf den Tourplänen gedruckt.

Bezugspersonen : (Bezug, Kundennummer, Sachbearbeiter)[] Die zum Kunden gehörenden Bezüge wie Hausartzt und Familienangehörige. Bei Institutionen werden noch die dortige Kundennummer und der zuständige Sachbearbeiter vermerkt.

Bevorzugt : Mitarbeiter[] Die vom Kunden bevorzugten Mitarbeiter, diese sollen i.d.R. dessen Dienstwünsche erfüllen.

Abgelehnt : Mitarbeiter[] Die vom Kunden abgelehnten Mitarbeiter, diese dürfen keine Dienste beim Kunden verrichten.

Hilfsmittel : Hilfsmittel[] Die vom Kunden benötigten Hilfsmittel wie Sitzkissen u.ä., die über die Attribute des Dienstwunsches hinausgehen. Diese Werte sind nur informativ und werden bei der automatischen Planung nicht berücksichtigt.

Fahrzeuge : Fahrzeug[] Fahrzeuge, die für den Kunden vom DRK freigegeben wurden. Ist dieser Vektor leer, dürfen alle Fahrzeuge bei dem Kunden eingesetzt werden.

Dienstwuensche : Dienstwunsch[] Die Dienstwünsche des Kunden.

- Methoden

  bevorzugt : Mitarbeiter[] Gibt, die bevorzugt gewünschten Mitarbeiter zurück.

  lehntAb : Mitarbeiter[] Gibt die abgelehnten Mitarbeiter zurück.

  lehntAb(Mitarbeiter[]) : boolean Gibt .T. zurück, wenn die übergebenen Mitarbeiter nicht eingesetzt werden dürfen.

  benoetigtFahrzeugTyp : Fahrzeugtyp[] Gibt eine Liste der Fahrzeugtypen zurück, in denen der Kunde mitfahren kann.

  erlaubt(Mitarbeiter[]|FahrzeugTyp[]) : boolean Gibt true zurück, wenn die übergebenen Mitarbeiter bzw. die übergebenen Fahrzeugtypen beim Kunden eingesetzt werden dürfen.

  bezugspersonen(Bezugsart) : Bezug[] Gibt alle Bezugspersonen der übergebenen Bezugsart zurück.

  hilfsmittel : Hilfsmittel[] Gibt alle vom Kunden benötigten Hilfsmittel zurück.

2.1.7 Mitarbeiter

Unterklasse von Person

- Attribute

  PersonalNr : String (Schlüssel) Personalnummer des Mitarbeiters beim DRK, die diesen eindeutig identifziert.

  DienstantrittsDatum : Date Datum, zu dem der Mitarbeiter seinen Dienst beim DRK antritt.

  EntlassungsDatum : Date Datum, zu dem der Mitarbeiter ausscheidet. Bei festangestellten Mitarbeitern ist dieser Eintrag in der Regel leer.

  VerfügbarkeitsZeiten : Termin Daten, an denen der Mitarbeiter eingesetzt werden kann.

  Bemerkungen : String Beliebiger Text, der den Einsatzplanern in der Mitarbeitermaske zur Information angezeigt wird.

  ErfuellteQualifikationen : Qualifikation[] Die vom Mitarbeiter erfüllten Qualifikationen.

- Methoden

  erfuelltQualifikationen : Qualifikation[] Gibt die vom Mitarbeiter erfüllten Qualifikationen zurück.

  erfuellt?(Qualifikation[]) : boolean Gibt true zurück, wenn die übergebenen Qualifikationen vom Mitarbeiter erfüllt werden.

2.1.8 Ort

- Attribute

  PLZ : String Fünfstellige Postleitzahl.

  OrtsName : String Name des Ortes, der zu obiger Postleitzahl gehört.

2.1.9 Person

- Attribute

  Name : String Nachname der Person.

  Vorname : String Vorname der Person.

  GeburtsDatum : Date Geburtsdatum der Person.

  Adressen : Station[] Die zur Person gehörenden Adressen.

  Kommunikationsverbindungen : Kommunikationsverbindung[] Die Kommunikationsverbindungen der Person.

  Bankverbindungen : Bankverbindung[] Die Bankverbindung der Person.

- Methoden

  adressen : Station[] Gibt die Adressen der Person zurück.

  kommunikationsverbindungen : Kommunikationsverbindungen[] Gibt die Kommunikationsverbindungen der Person zurück. (Werden Arten vorgegeben, könnte auch nach bestimmten Arten gefragt werden).

  bankverbindungen : Bankverbindung[] Gibt die Bankverbindungen der Person zurück.

2.1.10 Sachbearbeiter

Unterklasse von Person

- Attribute

  AngestelltBei : Instituition Arbeitgeber des Sachbearbeiters, beispielsweise Sozialamt oder Krankenkasse.

- Methoden

  angestelltBei : Institution Gibt den Arbeitgeber des Sachbearbeiters zurück.

2.1.11 Station

- Attribute

  Strasse : String Straßen- oder Platzname und -nummer, bzw. Postfach.

  PLZ : String Fünfstellige Postleitzahl.

  Anbindungspunkt : String Bezeichnung, unter der die Station im Verkehrstool vermerkt ist (i.d.R. die Adresse oder ein Punkt, der möglichst nahe an der Station liegt).

2.1.12 Qualifikation

- Attribute

Bezeichnung : String (Schlüssel) Eindeutiger (Kurz-)Bezeichner für die Qualifikation.

Beschreibung : String Nähere Beschreibung der Qualifikation.

Verrechnungswert : byte Kostenfaktor, der bei der Optimierung berücksichtigt wird, falls diese Qualifikation erfüllt, jedoch nicht gefordert wird. Dieser Wert wird automatisch berechnet, falls AutoWert true ist und sich Mitarbeiterdaten (bzgl. der Qualifikation) ändern.

AutoWert : boolean Ist AutoWert false, findet keine automatische Berechnung des Verrechnungswertes mehr statt. Wird der Verrechnungswert manuell geändert, wird AutoWert auf false gesetzt.

2.2 Klasse Dienstwunsch und zugehörige Klassen

2.2.1 Datumsspanne

- Attribute

Beginn: Calendar (Mußwert) Datum

Ende: Calendar Datum

2.2.2 Zeitspanne

- Attribute

frühestens: Calendar Uhrzeit

spätestens: Calendar Uhrzeit

Mindestens eines der Attribute muß einen Wert haben.

2.2.3 Zeitraum

- Attribute

Zeitrahmen: Datumsspanne (Mußwert)

Ausnahmezeiten: Vector of Datumsspanne

enthaeltDatum(Datum): boolean Liegt das gegebene Datum innerhalb des Zeitraums?

2.2.4 Termin

- Attribute

uebergeordneterDienstwunsch: Dienstwunsch (Mußwert)

Zeitraum: Zeitraum

Rhythmen: Vector of Rhythmus Verschiedene Rhythmen (z.B. für verschiedene Wochentage)

- Methoden

  enthaeltZeit(Zeit): boolean Liegt die gegebene Zeit innerhalb der angeforderten Zeiten?

  gleicherRhythmus(Termin): boolean Hat der Termin denselben Wiederholungsrhythmus wie ein anderer Termin?

2.2.5 Rhythmus

- Attribute

  Wochentage: int[] Werte = Wochentagskonstanten aus `java.util.Calendar`

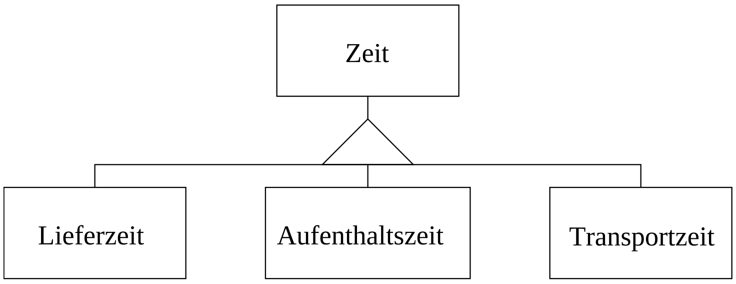
  NurWerktags: boolean

  MeldungAnFeiertagen: boolean

  Wochenrhythmus: int ≥ 1

  Uhrzeit: Zeit

2.2.6 Zeit: abstrakt



2.2.7 Lieferzeit: Zeit

- Attribute

  Lieferzeit: Zeitspanne (Mußwert)

2.2.8 Aufenthaltszeit: Zeit

- Attribute

  Beginn: Zeitspanne (Mußwert)

  Dauer: int Angabe in Minuten

2.2.9 Transportzeit: Zeit

- Attribute

  AbfahrtHin: Zeitspanne

  EinsteigezeitHin:int (Mußwert)

  AnkunftHin: Zeitspanne

  AussteigezeitHin:int (Mußwert)

AbfahrtRueck: Zeitspanne

EinsteigezeitRueck:int (Mußwert)

AnkunftRueck: Zeitspanne

AussteigezeitRueck:int (Mußwert)

Für Hin- und Rückfahrt muß entweder Abfahrtszeit oder Ankunftszeit gegeben sein.

2.2.10 Essensart

- Attribute

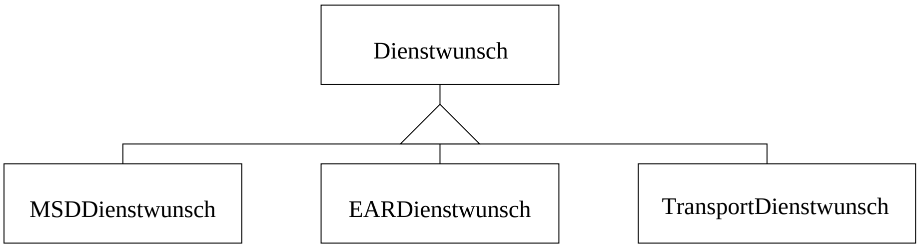
  Bezeichnung: String

  Beschreibung: String

  warm?: boolean

  Preis: int Angabe in Pfennig

2.2.11 Dienstwunsch: abstrakt



- Attribute

  Dienstart: int (Mußwert) Konstante $\in$ {MSD, EAR warm, EAR kalt, Schulfahrt, Dialysefahrt, Tagespflege, Individualfahrt}

  anfordernderKunde: Kunde (Mußwert)

  ZweiterMitarbeiter: boolean (Mußwert)

  Termin: Termin

  BezugspersonenAufTourplan: Vector of Bezugsperson Bezugspersonen, die auf dem Tourplan ausgedruckt werden sollen

  Zahler: Vector of Bezug (=Person oder Institution)

  Bemerkungen: String

- Methoden

  erfuelltAnspruch(Mitarbeiter, Mitarbeiter): boolean erfüllen die angegebenen Mitarbeiter die Ansprüche des Dienstwunsches bzw. des Kunden

  erfuelltAnspruch(Fahrzeug): boolean

  findetStattAm(Datum): boolean Soll der Dienst am gegebenen Tag stattfinden?

  erledigt(Datum): boolean Sind am angegebenen Tag alle Termine des Dienstwunsches abgearbeitet?

2.2.12 MSDDienstwunsch: Dienstwunsch

- Attribute

Dienstort: Station Wo soll der Dienst ausgeführt werden?

FaehrtKundeMit: boolean

benoetigterPlatz: int $\in$ {Rollstuhlplatz, Sitzplatz, Platz mit fester Sitzhilfe}

benoetigteSitzhilfe: Sitzhilfe

2.2.13 EARDienstwunsch: Dienstwunsch

- Attribute

Lieferadresse: Station

gewuenschteEssen: Vector of (Essensart $\times$ int) Wieviele Essen von welcher Sorte sollen geliefert werden?

2.2.14 TransportDienstwunsch: Dienstwunsch

- Attribute

Stationen: Vector of (Station $\times$ Dauer) Mindestens zwei Stationen. Die erste Station ist die Adresse, wo der Kunde abgeholt werden soll, die letzte die Zielstation, wo der Kunde hingebracht werden will.

Bei Individualfahrten können noch weitere Stationen angegeben werden, die unterwegs angefahren werden sollen.

benoetigterPlatz: int $\in$ {Rollstuhlplatz, Sitzplatz, Platz mit fester Sitzhilfe}

benoetigteSitzhilfe: Sitzhilfe

2.3 Klasse Fahrzeug und zugehörige Klassen

2.3.1 Klasse Fahrzeug

- Attribute

FahrzeugNr:String(Schlüssel,Mußwert) Eindeutige Bezeichnung für die DRK-interne Fahrzeugnummer.

KfzNr:String(Mußwert) Eindeutige Bezeichnung für das amtliche Kennzeichen.

Modell:String(Mußwert) Eindeutige Bezeichnung, die das Fahrzeugmodell beschreibt (z.B. mit Hersteller- und Modellname).

Anzahl_Schluessel:Byte Anzahl der mitgelieferten bzw. vorhandenen Schlüssel für das Fahrzeug.

Verfügbarkeit:Zeitraum(Mußwert) Termine, an denen das Fahrzeug eingesetzt werden kann. Die Klasse Termin verlangt einen Zeitrahmen der z.B. die geplante Nutzungsdauer des Fahrzeugs umfassen kann.

Anz_Sitzhilfen:Hashtable(Anz_Sitzhilfen_Art) (Mußwert) Anzahl der im Fahrzeug vorhandenen und nicht festinstallierten Sitzhilfen je Sitzhilfenart (z.B. Sitzkissen)

Naechste_HU:Date Datum, an dem das Fahrzeug das nächste mal zur Hauptuntersuchung muß.

Naechte_ASU:Date Datum, an dem das Fahrzeug das nächste Mal zur ASU muß.

Naechster_Kundendienst:Date Datum, an dem der nächste Kundendienst für das Fahrzeug fällig wird.

Aktuelle_Konfiguration:Fahrzeugkonfiguration(Mußwert) Die für das Fahrzeug momentan gültige Konfiguration.

passender_Typ:Fahrzeugtyp Der dem Fahrzeug zugeordnete Fahrzeugtyp.

- Methoden

Boolean ist_ASU_faellig(Zeitraum:Date) Funktion zum überprüfen, ob im angegebenen Zeitraum eine ASU für das Fahrzeug fällig ist.

Boolean ist_HU_faellig(Zeitraum:Date) Funktion zum überprüfen, ob im angegebenen Zeitraum eine Hauptuntersuchung für das Fahrzeug fällig ist.

Boolean ist_KD_faellig(Zeitraum:Date) Funktion zum überprüfen, ob im angegebenen Zeitraum ein Kundendienst für das Fahrzeug fällig ist.

Boolean ist_Fzg_durchgehend_verfuegbar(Zeitraum:Date) Funktion zum Überprüfen, ob das Fahrzeug für einen gegebenen durchgehend Zeitraum zur Verfügung steht.

2.3.2 Klasse Fahrzeugkonfiguration

- Attribute

Anzahl_Sitzplaetze:Byte(Mußwert) Anzahl der eingebauten Sitzplätze, einschließlich des Fahrersitzes und ohne Rollstuhlplätze. Kann nicht größer sein, als die maximale Sitzplatzzahl des zugeordneten Fahrzeugtyps (abzüglich der Rollstuhlplätze).

Anzahl_Rollstuhlplaetze:Byte(Mußwert) Anzahl der Plätze für Rollstühle, d.h. Plätze an denen kein Sitzplatz eingebaut ist.

Anzahl_fester_Sitzhilfen:Byte[1..Anz_Sitzhilfen_fest_Art] (Mußwert) Anzahl der festeingebauten Sitzhilfen, diese können im Gegensatz zu normalen Sitzhilfen nicht einfach vom Sitz entfernt werden.

Kuehlschienen:Boolean(Mußwert) Gibt an, ob in einem Fahrzeug Schienen für einen Kühlcontainer installiert sind.

- Methoden

Boolean kuehlmoeglichkeit() Liefert TRUE zurück, wenn das Fahrzeug Schienen für einen Kühlcontainer besitzt.

Byte verfuegbare_Plaetze() Gibt die im Fahrzeug verfügbaren Plätze zurück, die nicht durch festeingebaute Sitzhilfen belegt sind.

2.3.3 Klasse Fahrzeugtyp

- Attribute

  Bezeichnung:String(Schlüssel,Mußwert) Eindeutiger Kurzbezeichner des Fahrzeugtyps

  Beschreibung:String Nähere Beschreibung des Fahrzeugtyps.

  Max_Sitzplatzzahl:Byte(Mußwert) Maximale Anzahl der Sitzplätze im Fahrzeug, einschließlich des Fahrersitzes (lt. Fahrzeugschein).

  Kilometerpreis:Integer Preis, der je gefahrenen Kilometer mit diesem Typ abgerechnet wird.

  Moegliche_Konfigurationen:Vector of Fahrzeugkonfiguration Array, der die möglichen Konfiguration dieses Typs umfaßt.

2.4 Klasse Tour und zugehörige Klassen

2.4.1 Tour

- Attribute

  Tournummer : integer (Mußwert). Kein Schlüssel, da es alte Touren mit der gleichen Nummer geben kann. Es darf aber immer nur eine noch nicht abgelaufene Tour mit einer Tournummer geben.

  SollFahrzeug : Fahrzeug . Das Fahrzeug, das in der Regel diese Tour fährt. Sollte gesetzt sein, da sonst unter Umständen beim Bilden von Fahrten kein Fahrzeug mehr übrig ist.

  ersterSollMitarbeiter : Mitarbeiter . Der Mitarbeiter, der in der Regel diese Tour fährt. Auch dieser Wert sollte gesetzt sein (siehe SollFahrzeug).

  zweiterSollMitarbeiter : Mitarbeiter . Der zweite Mitarbeiter, der normalerweise diese Tour begleitet. Wenn ein zweiter Mitarbeiter benötigt wird, sollte dieser Wert gesetzt sein (siehe SollFahrzeug).

  erfülltWuensche : Vector (of Wunsch × boolean) (Mußwert). Der boolean-Wert gibt an, daß dieser Wunsch bei der Optimierung nicht aus dieser Tour genommen werden soll.

  Untertouren : Untertour[] (Mußwert). Liste der Untertouren dieser Tour, die auch leer sein kann.

- Methoden

  findetStattAn () : Termin . Liefert einen Termin zurück, in dem die Termine der Untertouren vereinigt sind.

  istAbgelaufen() : boolean . Ist true, wenn die Tour nicht mehr stattfindet, die Tour aber noch gespeichert wird.

erstelleFahrten(Zeitrahmen) : Fahrt[] . Erzeugt die Fahrten zur Tour im angegebenen Zeitrahmen. Eine Liste der erzeugten Fahrten wird zurückgegeben. In dieser Liste können Fahrten stehen, die nicht komplett sind, wo also noch ein Fahrzeug oder ein Mitarbeiter zugeordnet werden muß. Es werden alle erzeugten Fahrten zurückgegeben, um sie z.B. dem Benutzer zur Kontrolle anzeigen zu können.

2.4.2 Untertour

- Attribute

Untertournummer : integer (Mußwert). Wird extern als zweistellige Zeichenfolge dargestellt: 1..26=a..z, 27..702=aa..zz.

Tour : Tour (Mußwert). Verweist auf die zugehörige Tour.

An Wochentagen : boolean[Wochentag] (Mußwert). Gibt die Wochentage an, an denen diese Untertour gefahren wird.

Anfangsstation : Station . Gibt eine Station an, die immer als erste angefahren werden muß, z.B. die Küche bei einer Essen-Tour.

Endstation : Station . Gibt eine Station an, die unbedingt als letzte angefahren werden muß.

Stationen : Vector (of Station × Uhrzeit × Uhrzeit) (Mußwert). Die Reihenfolge, in der die Stationen angefahren werden, mit der jeweiligen Ankunfts- und Abfahrtsuhrzeit. Anfangs- und Endstation sind hier auch enthalten.

Fahrten Vector (of Fahrt) (Mußwert). Enthält die Fahrten, die zu dieser Untertour stattgefunden haben oder schon fest geplant sind.

- Methoden

Fahrten in Zeitrahmen (Zeitrahmen) : Fahrt[] . Gibt die Fahrten zurück, die im angegebenen Zeitrahmen stattgefunden haben bzw. schon fest geplant sind.

istAbgelaufen() : boolean (Mußwert). Siehe bei Tour.

erstelleFahrten(Zeitrahmen) : Fahrt[] . Erzeugt die Fahrten zur Tour im angegebenen Zeitrahmen. Eine Liste der erzeugten Fahrten wird zurückgegeben. In dieser Liste können Fahrten stehen, die nicht komplett sind, wo also noch ein Fahrzeug oder ein Mitarbeiter zugeordnet werden muß.

2.4.3 Fahrt

- Attribute

Untertour : Untertour . Dieser Eintrag kann auch NULL sein, wenn es sich um eine einmalige Individualfahrt handelt.

Startzeit : Calendar (Mußwert). Enthält Datum und Uhrzeit des Anfangszeitpunktes.

erfuellteWuensche : Vector (von Wunsch) (Mußwert). Wenn ein Kunde krank oder sonst irgendwie verhindert war, wird sein Wunsch hier nicht eingetragen oder wieder herausgelöscht.

Fahrzeug : Fahrzeug (Mußwert). Verweis auf das Fahrzeug, das wirklich gefahren ist.

ersterMitarbeiter : Mitarbeiter (Mußwert). Der Fahrer der Tour.

zweiterMitarbeiter : Mitarbeiter . Der Beifahrer der Tour, soweit vorhanden.

- Methoden

anzufahrendeStationen () : (Stationen × Uhrzeit)[] . Gibt die Stationen mit jeweiligem Ankunftszeitpunkt in der Reihenfolge zurück, in der sie angefahren werden sollen, egal ob Individualfahrt oder Tour.

planmäßig : boolean . Ist wahr, wenn soll-Fahrzeug und soll-Mitarbeiter die Fahrt übernehmen.

istKomplett : boolean . Ist wahr, wenn Fahrzeug und alle Mitarbeiter verfügbar sind, die Fahrt so also erfolgen kann.

2.5 Datenhaltung

Wie im Grobentwurf beschrieben, werden die Daten mittels Objekt-Serialisation in eine Datei geschrieben. Damit alle Objekte in eine Datei geschrieben werden, und um sicherzugehen, daß keines bei der Speicherung vergessen wird, wird zur Verwaltung ein Szenario-Objekt eingeführt, das alle anderen Stammdaten-Objekte, sowie die ebenfalls neu einzuführenden Tourszenario-Objekte referenziert. Es genügt dann, dieses Szenario-Objekt mittels der in Java vorgesehenen Methode *writeObject()* zu speichern.

Die Tourverwaltungsobjekte referenzieren jeweils alle Tour-, Untertour- und Fahrtobjekte eines Tourszenarios. Ein Tourszenario kann dann über dieses Tourszenario-Objekt aktiviert werden.

Es folgen nun die Beschreibung der Verwaltungsklassen, eine ausführliche Beschreibung der Methoden steht in Kapitel 1.3.1:

2.5.1 Szenario

- Attribute

Name : String (Schlüssel) Name des Szenarios.

Geaendert : Boolean Ist true, wenn der Datenbestand geändert wurde.

Personen : Person[] Referenz auf alle Personen des Szenarios.

Qualifikationen : Qualifikation[] Referenz auf alle Qualifikationen des Szenarios.

Stationen : Station[] Referenz auf alle Stationen des Szenarios.

Kommunikationsverbindungen : Kommunikationsverbindung[] Referenz auf alle Kommunikationsverbindungen des Szenarios.

Bankverbindungen : Bankverbindung[] Referenz auf alle Bankverbindungen des Szenarios.

Dienstwünsche : Dienstwunsch[] Referenz auf alle Dienstwünsche des Szenarios.

Dienstarten : Dienstart[] Referenz auf alle Dienstarten des Szenarios.

Essensarten : Essensart[] Referenz auf alle Essensarten des Szenarios.

Hilfsmittel : Hilfsmittel[] Referenz auf alle Hilfsmittel des Szenarios.

Bezugspersonen : Bezugspersonen[] Referenz auf alle Bezugspersonen des Szenarios.

Institutionen : Institution[] Referenz auf alle Institutionen des Szenarios.

Fahrzeuge : Fahrzeug[] Referenz auf alle Fahrzeuge des Szenarios.

Fahrzeugtypen : Fahrzeugtyp[] Referenz auf alle Fahrzeugtypen des Szenarios.

Fahrzeugkonfigurationen : Fahrzeugkonfiguration[] Referenz auf alle Fahrzeugkonfigurationen des Szenarios.

Tourszenarien : Tourszenario[] Referenz auf alle Tourszenarien des Szenarios.

- Methoden

ladeSzenario(Name) Lädt ein Szenario.

speichereSzenario(Name) Speichert ein Szenario.

loescheSzenario(Name) Löscht ein Szenario permanent.

neues Szenario(Name) Legt ein leeres Szenario an.

werdeMaster(Name) Macht ein Szenario zum Masterszenario.

istMaster(Name) : boolean Gibt true zurück, wenn es sich um ein Masterszenario handelt.

2.5.2 Tourszenario

- Attribute

Name : String (Schlüssel) Name des Tourszenarios.

IstMaster? : Boolean Ist true, wenn das Tourszenario ein Master-Tourszenario ist.

Touren : Tour[] Touren des Tourszenarios.

Untertouren : Untertour[] Untertouren des Tourszenarios.

Fahrten : Fahrt[] Fahrten des Tourszenarios.

- Methoden

aktiviereTourszenario(Name) Übergebenes Tourszenario wird das aktuelle Tourszenario, mit dem gearbeitet wird.

loescheTourszenario(Name) Löscht ein Tourszenario permanent.

neuesTourszenario(Name) Legt ein neues Tourszenario an.

werdeTourMaster(Name) Macht das angegeben Tourszenario zum MasterTourszenario.

istTourMaster(Name) : boolean Gibt true zurück, wenn das übergebene Tourszenario ein Master-Tourszenario ist. Andernfalls wird false zurückgegeben.

2.6 Vorgehensweise bei der Eingabe

2.6.1 Neuen Kunden eingeben

Über das Kundenmenü Kunde neu wählen. Es erscheint eine leere Kundenmaske, in die die in der Spezifikation festgelegten Eingabedaten eingegeben werden müssen. Zusätzlich zu den persönlichen Kundendaten müssen auch die Daten der Kontaktpersonen, der Rechnungsempfänger und die gewünschten Dienstwünsche eingegeben werden. Um den Dienstwunsch einzugeben, muß die Art des Wunsches aus einer Liste ausgewählt werden. Nach Auswahl der Dienstwunschart erscheint die passende Eingabemaske für den Dienstwunsch.

2.6.2 Kundendaten ändern

Bei der Auswahl zum Ändern eines Kunden erscheint eine Liste mit den dem System bekannten Kunden, aus denen der Benutzer den zu ändernden Kunden auswählt. Die Daten des Kunden erscheinen in der Kundenmaske und können dort geändert werden. Ebenso können hier Kontaktpersonen, Rechnungsempfänger und Dienstwünsche hinzugefügt, geändert und gelöscht werden. Beim Hinzufügen erscheint die entsprechende leere Eingabemaske, beim Ändern und Löschen erfolgt die Auswahl über eine Liste schon bestehender Kontaktpersonen, Rechnungsempfänger oder Dienstwünsche.

2.6.3 Kunden löschen

Es erscheint ebenfalls die Kundenliste, aus der der zu löschende Kunde ausgewählt wird.

2.6.4 Dienstwunsch hinzufügen/ändern

Siehe 2.6.2.

2.6.5 Manuelle Toureingabe

Durch Auswahl von Touren gelangt der Benutzer in eine Auswahlliste schon bestehender Touren, er kann hier entweder eine bestehende Tour ändern, löschen oder durch Auswahl von „<NEUE TOUR>“ eine neue Tour eingeben.

2.6.6 Manuelle Untertoureingabe

In der Untertour erfolgt die Festlegung der Stationenreihenfolge und der Zeiten der in der Tour spezifizierten Dienstwünsche, in der aus der Liste der möglichen Zeiten eine ausgewählt wird. Dazu muß aus der Liste der bestehenden Touren eine ausgewählt werden. Es erscheint eine Liste schon bestehender Untertouren, aus der eine zum Ändern ausgewählt oder eine neue Untertour eingegeben werden kann.

2.6.7 Fahrten ändern

Fahrten werden vom System implizit entweder durch das Erstellen der Dienstpläne oder durch das Abgleichen der Individualfahrten mit bestehenden Fahrten gebildet. Wurden in der zugehörigen Tour noch keine Fahrzeuge oder Mitarbeiter, die die Fahrt durchführen sollen, eingetragen, wird dies vom System gemeldet und die fehlenden Daten müssen von Hand eingetragen werden. Falls ein Mitarbeiter oder ein Fahrzeug ausfällt und ersetzt wird, sollte diese Änderung in den Fahrten vermerkt werden.

2.6.8 Optimierungen durchführen

Optimierungen in bestehenden Touren oder Untertouren können auf zwei Arten erfolgen: einerseits manuell, in diesem Falle greift der Benutzer von Hand in die Planung ein und führt eine Optimierung durch Änderungen in den Touren oder Untertouren (siehe 2.6.5 und 2.6.6) durch. Die zweite Möglichkeit besteht darin, die Optimierung automatisch durchzuführen. Diese Optimierung kann entweder für eine gesamte Dienstart erfolgen oder durch Auswahl einzelner Touren aus der Liste der bestehenden Touren. Sind im System noch keine Touren bzw. Untertouren erstellt worden, so können diese über diese Option erzeugt werden. Will der Benutzer alle Touren optimieren, muß er jede Dienstart der Reihe nach einzeln optimieren. Näheres zur Optimierung steht im Entwurf (1.6).

2.6.9 Ergebnisse der Analysefunktionen anzeigen

Je nach Analysefunktion kann der Benutzer wählen, ob er sich die Statistiken einzelner Mitarbeiter oder Fahrzeuge, bzw. die kompletten Statistiken anzeigen lassen oder ausdrucken will.

2.6.10 Ressourcen ändern

Je nach gewählter Ressource erscheint eine Liste der dem System bekannten Fahrzeuge oder Mitarbeiter, in der neue Ressourcen über die in der Spezifikation definierten Eingabemasken hinzugefügt, geändert oder gelöscht werden können.

2.6.11 Einstellungen

Unter diesem Menüpunkt kann der Benutzer in den für das System frei definierbaren Listen (Mitarbeiterqualifikationen, Essensarten, Feiertage, ...) Änderungen und Ergänzungen vornehmen.

2.7 Verkehrsmodul

In diesem Abschnitt wird die interne Arbeitsweise des Verkehrsmoduls beschrieben. Im einzelnen ist das die Art der Verwendung von Map&Guide, die verwendeten Tabellen und die Abwicklung der Vorbereitungs-Anfragen an Map&Guide.

2.7.1 Verwendung von Map&Guide

Damit Map&Guide die Batch-Aufträge des Transportoptimierungssystems findet, muß es im Batch-Modus sein. Dazu kann es entweder direkt in diesen Modus mit der Option `/b` gestartet werden, oder der Benutzer schickt Map&Guide mit der Tastenkombination Strg-B oder durch einen Klick auf den entsprechenden Schalter in den Batch-Modus.

Eine Abfrage des Zustands von Map&Guide ist nicht ohne weiteres möglich. Deshalb muß der Benutzer selbst dafür sorgen, daß Map&Guide im Batch-Modus läuft. Wenn es möglich ist zu prüfen, ob das Programm schon läuft, kann es als Erleichterung gegebenenfalls im Batch-Modus gestartet werden.

2.7.2 Verwaltete Daten

Das Verkehrsmodul hält zwei Tabellen: die eine Tabelle speichert die Anbindungspunkte mit all ihren Daten, die andere speichert die bereits von Map&Guide erhaltenen Entfernungen und die vom Benutzer überschriebenen Entfernungen.

Auf die Tabelle der Anbindungspunkte kann zum einen über den eindeutige Index, der in den Anbindungspunkten gespeichert wird, und zum anderen mittels einer Hash-Tabelle über die Map&Guide-Orte zugegriffen werden. Ein Eintrag in der Liste der Anbindungspunkte hat folgende Attribute:

AnzahlDerVerwendungen : integer . Gibt an, wie oft dieser Anbindungspunkt verwendet wird (ist z.B. > 1, wenn zwei Personen im gleichen Haus wohnen). Sobald dieser Wert 0 wird, wird der MapAndGuideOrt auf null gesetzt. Außerdem werden alle in der Entfernungs-Tabelle gespeicherten Einträge zu diesem Anbindungspunkt gelöscht. Der Eintrag selber darf aber nicht aus dem Vector entfernt werden, da sonst die Indexnummern der folgenden Einträge nicht mehr stimmen würden.

MapAndGuideOrt : string . Dieser String wird so an Map&Guide übergeben.

Die Entfernungstabelle wird als Hashtabelle implementiert. Auf die Einträge kann über die Kombination (von, nach) zugegriffen werden, wobei die Reihenfolge der Anbindungspunkte wegen der symmetrischen Tabelle keine Rolle spielt. Ein Eintrag hat folgende Attribute:

Fahrzeit : integer .

Entfernung : integer .

vonBenutzerGesetzt : boolean . Gibt an, ob der Eintrag vom Benutzer gesetzt wurde, um die Werte von Map&Guide zu überschreiben.

Vorerst werden Einträge in dieser Tabelle nur dann gelöscht, wenn einer der beiden Anbindungspunkte ungültig wird. Wenn der benötigte Speicher überhand nehmen sollte, kann auch eine vorzeitige Freigabe von länger nicht mehr benötigten Einträgen implementiert werden (Cache).

2.7.3 Ablauf von Vorbereitungs-Anfragen

Die Anfragen, bei denen eine Liste von Anbindungspunktpaaren angegeben wird, werden in der gleichen Reihenfolge an Map&Guide übergeben, nur daß Strecken, die schon bekannt sind, übersprungen werden.

Bei den Anfragen, wo alle Verbindungen zwischen den angegebenen Anbindungspunkten bereitgestellt werden sollen, kann der folgende Algorithmus verwendet werden:

```
Eingabe: Stationen (Menge der Stationen)
Ausgabe: -

s = |Stationen|;
haelfte = s / 2;
if ((haelfte*2) < s) ++haelfte; // aufrunden

position = 0;
fahreNach(position);
for(schrittweite=1; schrittweite<=haelfte; schrittweite++) {
  rest = s % schrittweite;
  versuche = ggT(s,schrittweite);

  if (rest+schrittweite == s) {
    continue;
  };

  for(lauf = 1; lauf <= versuche; lauf++) {
    anfang = position; // Anfangsort dieses Laufes
    do {
      if ((2 * schrittweite == s) && (next(position) == anfang))
        break;
      position = next(position);
      fahreNach(position);
    } while (position != anfang);
    if (lauf < versuche) {
      position = (position + 1) % s;
      fahreNach(position);
    }
  } // lauf
} // schrittweite
warte_auf_Ergebnis_von_MG;
lies_Ergebnis_und_speichere_es_in_Cache;
```

fahreNach(x) fügt die Station Stationen(x) als nächste in die Auftragsdatei für Map&Guide ein. Die Laufzeit in Anzahl der Aufrufe von `fahreNach` liegt in $\mathcal{O}(n^2)$.

Auch hier können die schon gespeicherten Strecken unter Umständen übersprungen werden. Ein Nachteil beider Verfahren ist, daß sie die schon gespeicherten Entfernungen nicht sinnvoll berücksichtigen.

Literatur

[Aig93] Aigner, Martin: *Diskrete Mathematik.* Friedr. Vieweg & Sohn, Postfach 58 29, D-65048 Wiesbaden, Deutschland, 1993.

[AL95] Appelrath, Hans-Jürgen und Jochen Ludewig: *Skriptum Informatik.* B. G. Teubner, Stuttgart, Dritte Auflage, 1995.

[AMO93] AHUJA, R. K., MAGNANTI und J. B. ORLIN: *Network flows: theory, algorithms, and applications.* Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 1993.

[Bal96] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik*, Band 1: Software-Entwicklung. Spektrum-Verlag, 1996.

[BFP+72] BLUM, M., R.W. FLOYD, V.R. PRATT, R.L. RIVEST und R.E. TARJAN: *Time bounds for selection.* J. Computer and System Sciences, 7:488–461, 1972.

[Boo94] BOOCH, G.: *Objektorientierte Analyse und Design.* Addison-Wesley, 1994.

[Buc94] BUCHHOLZ, FRIEDHELM: *Komplexität des Fahrgemeinschaften-Problems, Studienarbeit Nr. 1327.* Institut für Informatik der Universität Stuttgart, 1994.

[Buc95] BUCHHOLZ, FRIEDHELM: *Entwurf eines Systems zur Vermittlung von Fahrgemeinschaften, Diplomarbeit Nr. 1226.* Institut für Informatik der Universität Stuttgart, 1995.

[FB74] FINKEL, R.A. und J. L. BENTLEY: *Quad Trees - A Data Structure for Retrieval on Composite Keys.* Acta Informatica, 4:1–9, 1974.

[FH95] FREUDER, EUGENE C. und PAUL D. HUBBE: *A Disjunctive Decomposition Schema for Constraint Satisfaction*, Kapitel 17. In: [SvH95], 1995.

[Fla83] FLAJOLET, P.: *On the performance evaluation of extendible hashing and trie searching.* Acta Informatica, 20:345–369, 1983.

[FNPS79] FAGIN, R., J. NIEVERGELT, N. PIPPENGER und H.R. STRONG: *Extendible hashing – a fast access method for dynamic files.* ACM Trans. Database Systems, 4(3):315–344, 1979.

[FP86] FLAJOLET, P. und C. PUECH: *Partial match retrieval of multidimensional data.* Journal of the Association for Computing Machinery, 33(2):371–407, 1986.

[Gir87] GIRGENSOHN, ANDREAS: *ObjCon – Constraints in einer objektorientierten Sprache, Diplomarbeit Nr. 509.* Institut für Informatik der Universität Stuttgart, 1987.

[GJ78] GAREY, MICHAEL R. und DAVID S. JOHNSON: *Computers and Intractibility: A guide to the theory of NP-Completeness.* W. H. Freeman, San Francisko, 1978.

[Gle95] GLEICHER, MICHAEL: *Practical Issues in Graphical Constraints*, Kapitel 21. In: [SvH95], 1995.

[Grü90] GRÜN, BRIGIT: *Ein formularorientiertes Constraint-System, Studienarbeit Nr. 960.* Institut für Informatik der Universität Stuttgart, 1990.

[GT88] GOLDBERG, ANDREW V. und ROBERT E. TARJAN: *A New Approach to the Maximum-Flow Problem.* Journal of the Association for Computing Machinery, 35(4):921–940, Oktober 1988.

[Jon79] JONES, A.: *The Object Model: A Conceptual Tool for Structuring Software.* Springer Verlag, 1979.

[KW85] KRISHNARMUTHY, R. und K.-Y. WHANG: *Multilevel Grid Files – IVM Research Report*, 1985.

[Meh84] MEHLHORN, KURT: *Data Structures and Algorithms.* Springer-Verlag, 1984.

[NHS84] NIEVERGELT, J., H. HINTERBERGER und K. C. SEVICK: *The Grid File: An Adaptable, Symmetric Multikey File Structure.* ACM Transactions on Database Systems, 9(1):38–69, März 1984.

[OW93] OTTMANN, T. und P. WIDMAYER: *Algorithmen und Datenstrukturen.* BI Wissenschaftsverlag, 1993.

[Pro96] PROJEKTGRUPPE EVOLUTIONÄRE ALGORITHMEN: *Zwischenbericht – Bericht 1996/10.* Institut für Informatik der Universität Stuttgart, 1996.

[Pro97] PROJEKTGRUPPE FAHRGEMEINSCHAFTEN: *Zwischenbericht – Bericht 1997/10.* Institut für Informatik der Universität Stuttgart, 1997.

[PS95] PREPARATA, F.P. und M.I. SHAMOS: *Computational Geometry.* Springer-Verlag, 1995.

[Reg85] REGNIER, M.: *Analysis of grid file algorithms.* BIT, 25(2):335–357, 1985.

[Riv76] RIVEST, R.L.: *Partial-match retrivel algorithms.* SIAM Journal of Computing, 5(1):19–50, 1976.

[Sah74] SAHNI, SARTAJ: *Computationally related problems.* SIAM Journal of Computing, 3(4):262–279, dec 1974.

[San95] SANELLA, MICHAEL: *The SkyBlue Constraint Solver and Its Applications*, Kapitel 20. In: [SvH95], 1995.

[SB84] STEFIK, M. und D. BOBROW: *Object-Orientied Programming : Themes and Variations.* AI Magazine, 6(4):40–62, 1984.

[Sch97] SCHÄFFER, THOMAS: *Constraint Programming.* [Pro97].

[Sed91] SEDGEWICK, R.: *Algorithmen in C.* Addison-Wesley, 1991.

[SvH95] SARASWAT, VIJAY und PASCAL VAN HENTENRYCK: *Principles and Practice of Constraint Programming.* MIT Press, 1995.

[Tar91] TARJAN, ROBERT ENDRE: *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1991.

[Tri96] TRICK, MICHAEL A.: *Network Optimization.* http://mat.gsia.cmu.edu/OR/networks/networks.html, 1996.

[Wal92] WALTER, WOLFGANG: *Analysis 1.* Springer-Verlag, Zweite Auflage, 1992.