Zwischenbericht der
Projektgruppe
Transportoptimierung

Bericht Nr. 1998/06

Universität
Stuttgart

Zwischenbericht der Projektgruppe Transportoptimierung

Jörg Fleischmann
Lars Hermes
Tobias Spribille
Frank Wagner

Betreuung
Prof. Dr. Volker Claus
Dipl.-Inform. Friedhelm Buchholz
Dipl.-Inform. Stefan Lewandowski

Abteilung Formale Konzepte
Fakultät Informatik
Universität Stuttgart

4. Mai 1998

Prof. Dr. Volker Claus
Abteilung Formale Konzepte
Institut für Informatik
Universität Stuttgart

Breitwiesenstr. 20-22
D-70565 Stuttgart

Telefon:

0711-7816-300 (Prof. Dr. V. Claus)
0711-7816-301 (Sekretariat)
0711-7816-330 (FAX)

E-Mail: claus@informatik.uni-stuttgart.de

Inhaltsverzeichnis

Kapitel 1

Vorwort

1.1 Die Projektgruppe im Informatikstudium

Eine Projektgruppe ist eine besondere Form der Lehrveranstaltung im Rahmen des Informatikstudiums. Sie vereinigt Seminar bzw. Hauptseminar (2 SWS), Fachpraktikum (4 SWS) und Studienarbeit (10 SWS) in einer thematisch durchgehenden Lehrveranstaltung über den Zeitraum eines Jahres (hierfür sind 16 SWS angesetzt). Während dieser Zeit arbeitet eine Gruppe von (etwa fünf) Studenten von der Erarbeitung theoretischer Grundlagen bis zur Implementierung eines lauffähigen Programms gemeinsam an einem zentralen Thema.

Dadurch können Ausbildungsziele verfolgt werden, die sich kleineren, in sich abgeschlossenen Lehrveranstaltungen von Natur aus nicht oder nur begrenzt erschließen:

- Arbeiten im Team
- Analyse von Problemen, Strukturierung von Lösungen und gemeinsamer Entwurf geeigneter Systeme
- Selbständige Erarbeitung von Lösungsvorschlägen und deren Vorstellung und Verteidigung in einer Gruppe
- Übernahme von Verantwortung für die Lösung von Teilaufgaben und die Erstellung von Modulen
- Mitwirkung an einer umfassenden Dokumentation
- Erstellen eines Software-Produktes, das ein Einzelner innerhalb des vorgegebenen Zeitraumes unmöglich bewältigen kann. Hierbei sollen sämtliche Phasen eines Software-Lifecycles – von der Planung bis zur Wartung – durchlaufen werden
- Projekt-Planung und Kosten/Nutzen-Analyse
- Einsatz von Werkzeugen
- Persönlichkeitsbildung (Übernahme von Verantwortung, Selbstvertrauen, Verläßlichkeit, Rücksichtnahme, Durchsetzungsfähigkeit usw.)

Das Konzept der Projektgruppe wird bereits seit Jahren an anderen Universitäten wie z.B. in Oldenburg und Dortmund erprobt und durchgeführt. Dort sind Projektgruppen zum Teil schon Pflichtveranstaltungen im Rahmen des Informatikstudiums. An der Universität Stuttgart gibt es seit 1994 Projektgruppen im Fach Informatik. Im neu eingerichteten Studiengang Software-Technik ist die Teilnahme verbindlich vorgeschrieben.

Durch die eher kleine Teilnehmerzahl von 4 Studenten pro Projektgruppe (die ursprünglich für bis zu zehn Studierende vorgesehen war) können in der Praxis leider nicht alle Projektziele realisiert werden. Als ungünstig für die Teilnehmer hat sich auch erwiesen, daß die Arbeit in vollem Umfang in der vorlesungsfreien Zeit weitergeführt werden soll, was zwangsläufig zu Kollisionen mit Prüfungsvorbereitung und Urlaub führt.

Der Ablauf der Projektgruppe ist in verschiedene Phasen unterteilt:

Seminarphase Zu Beginn des Projekts werden verschiedene, für die folgende Arbeit grundlegende Themen erarbeitet. Literaturrecherche sowie Aufarbeitung und Präsentation des Materials sind wesentliche Arbeitsschritte, die hierbei gelernt werden sollen.

Anforderungsanalyse Die Projektgruppe analysiert die Aufgabenstellung durch Diskussion mit dem Kunden sowie Erarbeitung verschiedener Möglichkeiten der Modellierung. Ziel dieser Phase ist es, ein möglichst genaues Bild der zu modellierenden Objekte aus der realen Welt zu bekommen, sowie alle Begriffe festzulegen und zu standardisieren. Damit kann im folgenden eine Abbildung auf Verfahren und Strukturen der Informatik erfolgen.

Spezifikation Die jetzt genau bekannte Aufgabenstellung und Anforderung des Kunden wird modelliert. Dabei werden bekannte Problemlösungen aus der Literatur genauso gesammelt wie eigene Ideen. Abwägungen zwischen unterschiedlichen Ansätzen in Hinblick auf die gewünschten Eigenschaften des Ergebnisses (Flexibilität, Schnelligkeit, Erweiterbarkeit, etc.) müssen hier getroffen werden. Dabei soll eine möglichst genaue, in Teilen auch formale, Beschreibung des zu programmierenden Systems entstehen.

Entwurf Aufbauend auf die Spezifikation werden hier die einzelnen Komponenten des Programms bis ins Detail beschrieben. Die verschiedenen Schnittstellen sowohl zum Benutzer als auch innerhalb des Programms werden genau festgelegt, ebenso die zu verwendenden Datenstrukturen und Algorithmen. Der Entwurf soll eine möglichst eindeutige Beschreibung des Programms darstellen und somit direkt als Vorlage für die Implementierung dienen. Alle wesentlichen Design-Entscheidungen, die bisher noch nicht getroffen werden, fallen im Entwurf.

Implementierung Nach Einarbeitung in eine Programmiersprache und zugehörige Entwicklungsumgebungen werden die Vorgaben von Spezifikation und Entwurf konkret in Code umgesetzt.

Test Das lauffähige Programm wird bereits während der Codierung mit möglichst realistischen Daten getestet, um Programmfehler zu entdecken und die Handhabbarkeit zu prüfen.

Dokumentation/Präsentation Die gesamte Arbeit der Projektgruppe wird fortlaufend dokumentiert. Der Stand nach dem Entwurf („Zwischenbericht“) sowie der Überblick über das gesamte Projekt („Endbericht“) werden gedruckt und ähnlich einer Studienarbeit veröffentlicht. Das Ergebnis der einjährigen Arbeit wird außerdem in einer Präsentationsveranstaltung der interessierten Öffentlichkeit vorgeführt.

1.2 Die Aufgabenstellung

Neben den allgemeinen Lehrinhalten im Rahmen des Informatikstudiums hat diese Projektgruppe ein konkretes Ziel: Es soll ein Programm zur Verwaltung und Optimierung von sozialen Fahrdiensten entwickelt werden, das dann beim DRK in Stuttgart eingesetzt wird.

Das Programm soll alle für das DRK in Zusammenhang mit seinen Fahrdiensten wichtigen Daten erfassen und verwalten. Die momentan manuell durchgeführte Planung soll in Zukunft computergestützt ablaufen können, was den Planern beim DRK einen besseren Überblick über die Dienste und Einsatz von Mitarbeitern und Fahrzeugen liefern soll. Außerdem ist vorgesehen, die Optimierung von Fahrdiensten auch automatisch auszuführen, um aufwendige Handarbeit einzusparen oder gar bessere Ergebnisse zu bekommen.

Zunächst wird das Programm also denjenigen Ausschnitt der realen Welt modellieren, der soziale Fahrdienste (und zwar ganz speziell aus Sicht des DRK Stuttgart) betrifft. Dabei geht die Anforderungsanalyse vom Ist-Zustand beim DRK aus, so daß das Programm letztlich in der Lage ist, diesen möglichst exakt wiederzugeben.

Dennoch sollen einzelne Komponenten des Programms unabhängig austausch- und erweiterbar sein. Durch Einsatz objektorientierter Analyse- und Designmethoden von der Spezifikation bis zur Implementierung wird ein modularer Programmaufbau erreicht, der die einzelnen Komponenten weitgehend unabhängig voneinander und damit austauschbar macht.

Kapitel 2

Seminarvorträge

2.1 Constraint Programming (Tobias Spribille)

2.1.1 Einführung

Ein Computer ist von seiner Grundstruktur her für eine imperative Vorgehensweise bei der Problemlösung geschaffen. Dies schlägt sich auch in den meisten gängigen Programmiersprachen und bekannten Algorithmen nieder: Um ein Problem zu lösen, wird eine Folge von konkreten Handlungsanweisungen angegeben, die aus einem Startwert die zugehörige Lösung berechnen. Alle Eigenschaften der Lösung sind implizit in den einzelnen Teilen der Rechenvorschrift enthalten. Deshalb ist einem Programm oft nicht leicht anzusehen, was es überhaupt macht. Ebenso aufwendig kann es sein, zu einem Problem einen Algorithmus zu finden, der die Lösung berechnet, deren Eigenschaften oft recht einfach anzugeben sind.

Von eben diesen Eigenschaften der Lösung geht das deklarative Programmierparadigma aus, das in Sprachen wie z.B. PROLOG zum Einsatz kommt. Der Programmierer beschreibt in einer formalen Syntax die Eigenschaften, die die gesuchte Lösung haben soll, und das System ermittelt diese daraus.

```
int fib(int x)
{
  int f1 = 0, f2 = 1;
  while (x-- >= 1)
  {
    int temp = f1 + f2;
    f1 = f2;
    f2 = temp;
  }
  return f2;
}
```

```
fib(0,0).
fib(1,1).
fib(N,X) :- fib(N-1,Y),
            fib(N-2,Z),
            X is Y+Z.
```

Abbildung 2.1: Imperative und deklarative Berechnung der Fibonacci-Zahlen

Die Bezeichnung Constraint Programming steht für solche deklarativen Problemlösungsmethoden, bei denen für eine Menge von Variablen Werte gefunden werden müssen, die einer Menge von Bedingungen (den constraints) genügen.

Zu deklarativ spezifizierten Problemen gibt es im Unterschied zu den imperativen Verfahren nicht immer eine eindeutige Lösung. Für praktische Problemstellungen wie die Transportoptimierung ist es aber ausreichend, irgendeine mögliche Lösung zu finden, die alle Anforderungen erfüllt.

Ein PROLOG-Interpreter findet die Lösung mit den angegebenen Eigenschaften durch eine Tiefensuche im exponentiell großen Lösungsraum. Dies kann bei großen Problemen sehr lange dauern. Deshalb werden für Constraint Programming spezielle Lösungsalgorithmen benutzt, die durch geschickte Wahl der Suchrichtung und/oder Einsatz von Heuristiken möglichst früh auf eine Lösung stoßen sollen oder die Anzahl der zu betrachtenden Möglichkeiten reduzieren.

2.1.2 Grundlagen

Definition 2.1.1 Ein *Constraint Satisfaction Problem* (oder kurz *CSP*) ist ein Tripel $(\mathcal{V}, \mathcal{W}, \mathcal{C})$, bestehend aus:

- Einer Menge von Variablen $\mathcal{V} = \{V_1, \ldots, V_n\}$ mit Wertebereich $\mathcal{W}$.
- Einer Menge von Constraints $\mathcal{C} = \{C_1, \ldots, C_r\}$. Ein Constraint ist eine Beziehung zwischen Variablenwerten:

$$\begin{aligned} C_j : \mathcal{P}(\mathcal{V}) \times (\mathcal{V} \to \mathcal{W}) &\to \mathbb{B} \\ \{\sigma(V_{i_1}), \ldots, \sigma(V_{i_k})\} &\mapsto \{\text{true, false}\} \end{aligned}$$

Gesucht ist eine Variablenbelegung $\sigma \in (\mathcal{V} \to \mathcal{W})$, die alle Constraints erfüllt:

$$\forall j \in \{1, \ldots, r\} : \quad C_j(\{V_{i_1}, \ldots, V_{i_k}\}, \sigma) = \text{true}$$

Definition 2.1.2 Ein *hierarchisches CSP* definiert zusätzlich eine Hierarchie auf den Constraints:

$$H : C \to \{1, \ldots, h_{\max}\}$$

Constraints, die höher in der Hierarchie stehen, sind wichtiger als tiefere, und ihre Erfüllung hat Priorität vor allen anderen Constraints.

Es gibt verschiedene Arten von CSPs:

- ein *unterbestimmtes (underconstrained) CSP* hat mehrere Lösungen, da die Constraints nicht alle Variablen eindeutig bestimmen. Da in vielen Anwendungen nicht der Benutzer entscheiden kann, welche Lösung gewählt werden soll, könnte hier mit Heuristiken versucht werden, dessen Intention zu treffen.
- ein *überbestimmtes (overconstrained) CSP* enthält mehr Constraints, als zur eindeutigen Bestimmung der Variablenwerte notwendig wären. Dies ist auf zwei verschiedene Arten möglich:
  - Einige Constraints sind redundant, d.h. es kann eine Lösung gefunden werden, die allen Constraints genügt (aber bereits mit einer Teilmenge auch gefunden worden wäre)

  - Es entsteht ein Konflikt, da nicht alle Constraints gleichzeitig erfüllt werden können. Über eine Hierarchie kann von vornherein festgelegt werden, welche Constraints in einem solchen Fall wichtiger sind, und welche am ehesten wegfallen dürfen. Eine Möglichkeit der Konfliktlösung ist auch, die Lösung mit dem minimalen Fehler zu wählen.

- Probleme mit dynamisch sich ändernden Constraints lassen sich mit *inkrementellen* Verfahren lösen, die die Constraints in ihrer Entstehungsreihenfolge einzuplanen versuchen. Bei einer Änderung muß meist nur ein kleiner Teil der Werte neu berechnet werden, so daß sich diese Verfahren besonders für interaktive Anwendungen eignen.

2.1.2.1 Vorteile von Constraint Programming

Aus den bisher vorgestellten Eigenschaften lassen sich bereits einige Vorteile von Constraint Programming gegenüber expliziten Lösungsverfahren für ein Problem erkennen:

- Man muß nur die Eigenschaften einer Lösung beschreiben, nicht den Weg dorthin

- Bedingungen können sich dynamisch ändern (in Form von zusätzlichen Constraints)

- lokale Zusammenhänge können ein globales Verhalten beschreiben

2.1.3 Anwendungsbeispiele

2.1.3.1 Robotersteuerung

Einfache Roboter wie den Kran in Bild 2.2 links lassen sich noch recht einfach durch explizite Angabe der Bewegung für die einzelnen Teile steuern. Mit zunehmender Anzahl der Freiheitsgrade werden die Zusammenhänge jedoch so komplex, daß dies nicht mehr ohne weiteres möglich ist. Für die Roboterhand in Bild 2.2 rechts müssen zahlreiche mechanische Beziehungen zwischen den sieben Teilen berücksichtigt werden. Diese können als Constraints formuliert werden, aus denen dann ein geeigneter Lösungsalgorithmus die Bewegung der einzelnen Teile ermittelt.

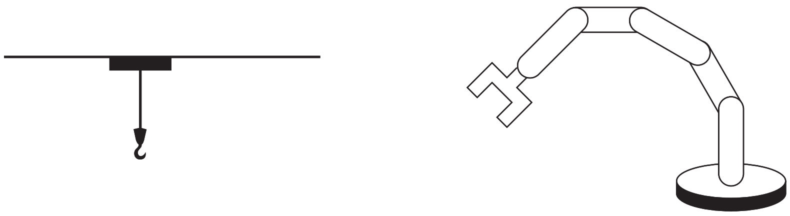


Abbildung 2.2: Roboter mit wenigen und vielen Freiheitsgraden

Hier ist gut erkennbar, wie sich lokale Constraints (Bedingungen an den Gelenken, die nur die Lage zweier benachbarter Glieder beschreiben) auf das globale Verhalten des Roboters auswirken.

2.1.3.2 Selbsterklärende Simulation

Ein Simulationssystem, dessen Wertebeziehungen durch Constraints realisiert sind, kann nicht nur gültige Werte berechnen, sondern ist überdies in der Lage, die Zusammenhänge zwischen diesen Werten zu erklären. So kann dem Benutzer auf Anfrage mit beliebigem Detaillierungsgrad mitgeteilt werden, warum eine Variable einen gewissen Wert hat. Im folgenden Beispiel könnte erklärt werden, warum die Fahrt erst um 7.30 Uhr beginnen kann.

Beispiel aus der Transportoptimierung Betrachtet werden soll ein Teilzusammenhang aus der Behindertenbeförderung: Zwei Behinderte sollen morgens von einem Fahrzeug nacheinander zu Hause abgeholt werden. Die dabei zu beachtenden zeitlichen Zusammenhänge könnten folgendermaßen als hierarchisches CSP mit drei Variablen und vier Constraints formuliert werden:

V_1 : Weckzeit für den Behinderten, der zuerst abgeholt wird

V_2 : Abholzeit für den ersten Behinderten

V_3 : Abholzeit für den zweiten Behinderten

C_1 : $V_1 \geq 6,5$ (die Eltern des Behinderten, die ihn betreuen, wollen nicht so früh aufstehen), Stärke 1

C_2 : $V_2 - V_1 \geq 1$ (bis der Behinderte fertig zum Abholen ist, dauert es eine Stunde), Stärke 3

C_3 : $V_3 - V_2 \geq \frac{1}{2}$ (die Fahrt von der Wohnung des ersten Behinderten zum zweiten dauert eine halbe Stunde), Stärke 4

C_4 : $V_3 \leq 7,5$ (die Eltern des zweiten Behinderten wollen früh zur Arbeit, damit sie früh Feierabend haben), Stärke 1

2.1.3.3 Intelligente Computergraphik

In der Computergraphik müssen oft geometrische Zusammenhänge zwischen Objekten erfüllt werden. Viele Graphikprogramme bieten deshalb Funktionen an, die z.B. das Zeichnen von parallelen Linien erlauben. Dadurch ist der Zusammenhang aber nur unmittelbar nach dem Einfügen der Linien gegeben. Wird eine Linie gedreht, ändern sich auch die geometrischen Zusammenhänge.

Constraints zur Überwachung solcher Beziehungen würden die zweite Linie mitdrehen und so die Parallelität aufrechterhalten. Dadurch können lokale interaktive Änderungen einen globalen Effekt haben. Ein Objekt hat kein festes Verhalten mehr, dieses ergibt sich erst aus der Kombination von Objekt und Constraints. Durch die Angabe von Constraints „programmiert“ der Benutzer sozusagen das System [Gle95].

2.1.4 Constraints als Vorgaben

2.1.4.1 Einführung

In Abschnitt 2.1.2 wurde ein Constraint als eine Beziehung zwischen Variablen definiert. Ein Programm, das zu einer Constraintmenge eine Variablenbelegung findet, also ein CSP löst, heißt *Constraint Solver*. Ein Beispiel für einen solchen Lösealgorithmus soll im folgenden betrachtet werden.

2.1.4.2 Disjunktive Zerlegung

Definition In [FH95] wird ein grundlegendes Verfahren zur Lösung von Constraint Satisfaction Problems angewandt, das nach dem Verfahren *Teile und Herrsche* arbeitet. Die Hauptrolle spielt dabei die Zerlegung des Problems in Teilprobleme, die jedes für sich auf Lösungen untersucht werden. Damit ist diese Methode für Parallelisierung sehr gut geeignet.
Das Verfahren geht folgendermaßen vor:

```
Lege das Ausgangsproblem auf den Stack
while Stack ≠ ∅
   Nimm ein Problem P vom Stack
   if alle Variablen in P sind instantiiert
     return(P)
   else
     Zerlege P in Teilprobleme {P_i}
     Lege alle nichtleeren P_i auf den Stack
  endif
endwhile
return(∅)
```

Zunächst sind alle Variablen uninstantiiert. Sie werden *instantiiert*, indem bei der Zerlegung Werte oder Teile der (endlichen) Wertemengen dafür festgelegt werden. Ein Teilproblem heißt *leer*, wenn der Wertebereich einer instantiierten Variablen leer ist, wenn es also keine möglichen Werte mehr gab, die mit den anderen instantiierten Variablen konsistent sind. Dabei heißen zwei Werte für zwei Variablen *konsistent*, wenn es keinen Constraint gibt, der diese Wertekombination verbietet.

Definition 2.1.3 Eine *disjunktive Zerlegung* ist eine Zerlegung eines Problems in Teilprobleme mit folgenden Eigenschaften:

1. Konsistenz: Die Werte instantiierter Variablen sind zueinander konsistent.
2. Reduktion: Jedes der P_i hat weniger mögliche Wertekombinationen oder mehr instantiierte Werte als P.
3. Semi-Vollständigkeit: Wenn es zu P eine Lösung gibt, dann auch zu mindestens einem der P_i.

Satz 2.1.1 Eine disjunktive Zerlegung garantiert, daß das oben vorgestellte Zerlegungsverfahren für lösbare Probleme eine der Lösungen und sonst die leere Menge zurückgibt.

Beweis 2.1.1 Da die Wertebereiche der Variablen endlich sind, folgt aus der Reduktionseigenschaft, daß irgendwann alle Teilprobleme entweder leer sind oder nur instantiierte Variablen besitzen.

Nun betrachtet man den Baum, der entsteht, wenn die Teilprobleme nach der Zerlegung als Knoten unter das Problem gehängt werden, von dem sie ausgehen. Dann folgt aus der Semi-Vollständigkeit, daß auf jeder Ebene ein Knoten existiert, der eine Lösung des Wurzelproblems enthält. Wenn man diesen Pfad von der Wurzel aus verfolgt, kommt man nach endlich vielen Schritten an ein

Blatt, das alle Variablen instantiiert hat und somit eine Lösung für das Ausgangsproblem darstellt.

Weitere wünschenswerte Eigenschaften könnten sein:

1. Vollständigkeit: Jede Lösung von P ist auch Lösung mindestens eines der P_i.
2. Redundanzfreiheit: Jede mögliche Wertekombination für P existiert in höchstens einem der P_i.
3. Reduzierbarkeit: Die Summe der Größen (möglichen Wertekombinationen) der P_i kann kleiner sein als die Größe von P.

Diese Eigenschaften gelten unabhängig von der Reihenfolge, in der die Probleme und Teilprobleme betrachtet werden. Zum einen kann die Reihenfolge variiert werden, in der die Teilprobleme auf den Stack gelegt werden, zum anderen muß die Menge der noch zu bearbeitenden Probleme gar nicht unbedingt als Stack organisiert sein. Diese Entscheidung, in welcher Reihenfolge die Teilprobleme erledigt werden, ist ein guter Ansatzpunkt für heuristische Verfahren.

Beispiel aus der Transportoptimierung Um verschiedene Arten solcher Zerlegungsstrategien vergleichen zu können, soll das Verfahren auf folgendes Beispiel angewandt werden:

Ein Fahrdienst besteht aus vier Touren pro Tag, zu denen drei verschiedene Fahrzeuge eingesetzt werden können. Da jeder Bus nach der Tour gereinigt oder für die nächste Fahrt ausgerüstet werden muß (z.B. mit speziellen Sitzen), können zwei aufeinanderfolgende Fahrten nicht mit dem selben Fahrzeug gemacht werden („aufeinanderfolgend“ ist hier zyklisch gemeint, z.B. wenn die Autos rund um die Uhr im Schichtbetrieb eingesetzt werden).

Das Problem, die Fahrzeuge den Touren zuzuordnen, besteht also aus den Variablen $T_1, \ldots, T_4$ für die Touren, mit jeweils der Wertemenge $\{a, b, c\}$, entsprechend den drei Fahrzeugen. Die Bedingungen ergeben vier Constraints: $C_i: \quad T_i \neq T_{i \oplus 1}, \quad i = 1, \ldots, 4$

Backtracking Diese einfachste Form der Suche im Zerlegungsbaum definiert die Zerlegung in zwei Teilprobleme:

- P_1: Sei V_1 die erste nicht instantiierte Variable, $v_1 \in \mathrm{dom}(V_1)$. Wenn v_1 konsistent mit den bisher instantiierten Variablen ist, instantiiere V_1 mit $\mathrm{dom}(V_1) = \{v_1\}$, sonst mit $\mathrm{dom}(V_1) = \emptyset$.
- P_2: $\mathrm{dom}(V_1) = \mathrm{dom}(V_1) \setminus \{v_1\}$.

Ein Ausschnitt des Baumes ist in Abbildung 2.3 zu sehen. Jeder Knoten steht für ein Teilproblem, die vier Zeilen jedes Knotens geben die Wertemengen der Variablen $T_1 \ldots T_4$ an. Die Werte instantiierter Variablen sind unterstrichen.

Forward Checking Da mit der Instantiierung einer Variablen sich bereits die Wertebereiche anderer Variablen verkleinern (weil Constraints manche Kombinationen von Werten mit den instantiierten Werten ausschließen), ist es sinnvoll, den Baum zu verkleinern, indem nur die jeweils konsistenten Werte weiterbetrachtet werden. Dies führt zu folgender Zerlegung:

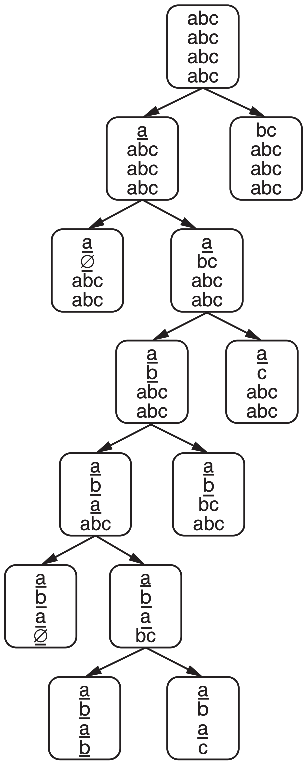


Abbildung 2.3: Backtracking

- P_1: $\mathrm{dom}(V_1) = \{v_1\}$.
  Für alle anderen uninstantiierten Variablen W:
  $\mathrm{dom}(W) = \{w \in \mathrm{dom}(W) \mid w \text{ ist konsistent zu } v_1\}$
- P_2: $\mathrm{dom}(V_1) = \mathrm{dom}(V_1) \setminus \{v_1\}$.

Ein Ausschnitt des Baumes ist in Abbildung 2.4 zu sehen.

Backtracking mit kartesischem Produkt Hier können Variablen mit ganzen Wertemengen instantiiert werden. Jedes Element aus dem kartesischen Produkt dieser Wertemengen ist eine Lösung des Problems. Damit müssen sehr viel weniger Einzelfälle unterschieden werden (z.B. in Abbildung 2.4 die Zerlegung der Wertemenge $\{bc\}$ für T_4).

Sei $\{S_i\}$ die Partition von $\mathrm{dom}(V_1)$ in disjunkte Mengen S_i. Dabei sind zwei Werte in der gleichen Teilmenge, wenn sie mit denselben Werten der bereits instantiierten Variablen konsistent sind.

Zu jeder Teilmenge S_i ergibt sich das Teilproblem P_i:

- P_i: $\mathrm{dom}(V_1) = S_i$
  Für alle bereits instantiierten Variablen W:
  $\mathrm{dom}(W) = \{w \in \mathrm{dom}(W) \mid w \text{ ist konsistent zu den Werten in } S_i\}$

Ein Ausschnitt des Baumes ist in Abbildung 2.5 zu sehen.

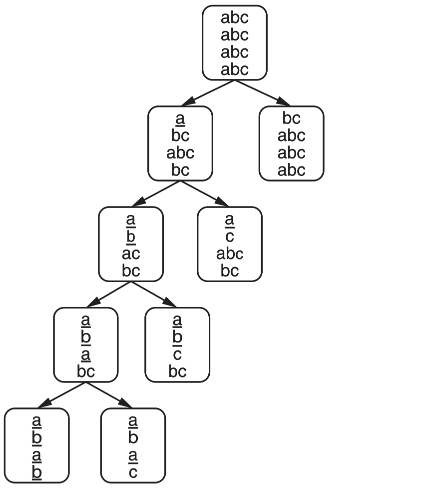


Abbildung 2.4: Forward Checking

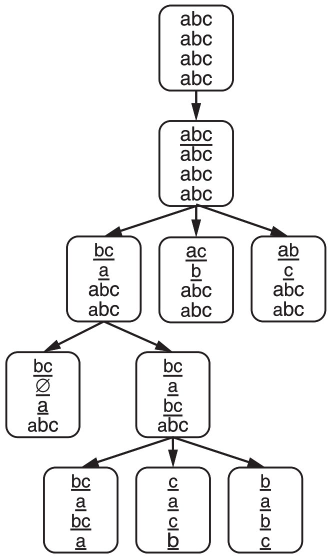


Abbildung 2.5: Backtracking mit kartesischem Produkt

Komplexität

Satz 2.1.2 Der Stack für die Disjunktive Zerlegung wird $O(t \cdot |\mathcal{V}|)$ Knoten groß. Dabei ist $t \leq |\mathcal{W}|$ die Anzahl der Teilprobleme, die bei einem Zerlegungsschritt maximal enstehen.

Beweis 2.1.2 Auf jeder Ebene des Zerlegungsbaumes können $t - 1$ noch unbehandelte Knoten stehen. Da bei jedem Zerlegungsschritt eine Variable instantiiert wird, können nur $|\mathcal{V}|$ solche Ebenen entstehen.

Satz 2.1.3 Die Größe des Zerlegungsbaumes ist in $O(|\mathcal{W}|^{|\mathcal{V}|})$.

Beweis 2.1.3 Betrachte exemplarisch Backtracking mit kartesischem Produkt: Im schlechtesten Fall gibt es genausoviele Teilprobleme wie Variablenwerte. Auf jeder Ebene wird eine Variable instantiiert, es kann also $|\mathcal{V}|$ Ebenen geben, deren Knoten je $|\mathcal{W}|$ Unterknoten haben.

2.1.5 Constraints als Beziehungen

2.1.5.1 Einführung

Mit einem deklarativen Programm kann man einfach nur Beziehungen überprüfen (z.B. wird das PROLOG-Programm von Seite 11 auf die Eingabe „`fib(7,13).`“ mit „`Yes`“ antworten, da 13 die siebte Fibonacci-Zahl ist).

Der hauptsächliche Sinn und Zweck der deklarativen Programmierung besteht jedoch darin, daß der Rechner die Lösung selbst bestimmt, also auf Eingabe von „`fib(7,X).`“ mit „`X=13`“ antwortet.

Ebenso sind Constraints nicht nur Bedingungen, die eine potentielle Lösung auf ihre Korrektheit prüfen, sondern können auch selbständig aus gegebenen

Daten Werte entsprechend ihrer Bedingung berechnen: Ein Constraint, der eine Summenbeziehung ausdrückt, könnte wie in Abbildung 2.6 links dargestellt werden. Die Beziehung $a + b = c$ kann auf drei Arten sichergestellt werden: Sind a und b gegeben, kann man $c := a + b$ berechnen. Genausogut ließe sich aber $a := c - b$ oder $b := c - a$ berechnen, d.h. aus zwei gegebenen Variablen kann automatisch die dritte berechnet werden, so daß der Constraint erfüllt ist. Ein Constraint hat also keine fest definierten Ein- und Ausgabevariablen, sondern eine Reihe von *Methoden*, die angeben, wie aus einem Teil der Variablen ein anderer berechnet werden kann.

Constraints mit Methoden können zu *Constraint-Netzen* zusammengesetzt werden. In Abbildung 2.6 rechts ist ein Netz aus zwei Constraints abgebildet, das einen Zusammenhang aus der Transportoptimierung veranschaulicht: Aus dem Anfahrtszeitpunkt beim DRK ergibt sich durch Addieren der Fahrtzeit (Konstante dt_1) die Ankunftszeit beim Behinderten. Mit einer weiteren Fahrtzeit (dt_2) erhält man die Ankunftszeit an der Schule. Wie oben erläutert kann hier sowohl aus dem Abfahrtszeitpunkt beim DRK die Ankunftszeit an der Schule berechnet werden als auch umgekehrt.

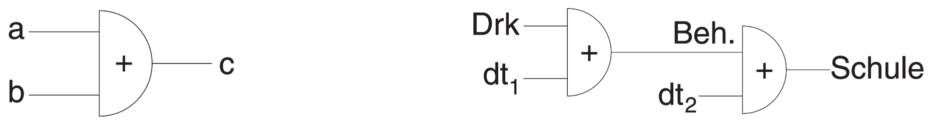


Abbildung 2.6: Constraint als Wertebeziehung

2.1.5.2 Constraint-Netze am Beispiel des Solvers „SkyBlue“

In [San95] wird der an der University of Washington entwickelte inkrementell Constraint Solver „SkyBlue“ vorgestellt, der mit Netzen aus hierarchischen Constraints arbeitet. Obwohl das allgemeine CSP NP-hart ist (siehe Abschnitt 2.1.7), gehen die Entwickler des Algorithmus’ davon aus, daß in praktisch vorkommenden Problemstellungen das Einfügen/Entfernen eines Constraints in linearer Zeit möglich ist.

Grundlegende Vorgehensweise Um eine Menge von Constraints zu erfüllen, wird zu jedem Constraint eine Methode gewählt. Aus den Variablen und den Constraints als Knoten sowie den gewählten Methoden als Kanten entsteht der *Methodengraph.* Kantenrichtungen geben an, welcher Constraint welche Variable setzen darf. Haben mehrere Constraints dieselbe Variable in ihrem Ausgabebereich, entsteht ein Konflikt.

Enthält der Graph keine Zyklen und keine Konflikte, kann eine den Constraints entsprechende Variablenbelegung ermittelt werden, indem die gewählten Methoden in topologischer Reihenfolge ausgeführt werden. Zyklen werden hier zu speziellen Knoten zusammengefaßt, die zunächst wie normale Knoten behandelt werden, und wenn die zugehörige Methode aufgerufen wird, von einem speziellen Auflöse-Algorithmus behandelt werden.

„SkyBlue“ benutzt Constraint-Hierarchien, um optimale Methodengraphen zu erzeugen. Ein Methodengraph heißt hier *optimal*, wenn kein unerfüllter Constraint erfüllt werden könnte, ohne daß durch die veränderte Methodenwahl ein Konflikt mit Constraints gleicher oder höherer Stärke auftritt.

Der SkyBlue-Algorithmus Ausgehend von leerem Methodengraph werden einzelne Constraints hinzugefügt/entfernt. In jedem Schritt wird der neu entstandene Graph optimiert und die gewählten Constraint-Methoden werden ausgeführt, damit die Variablenwerte die jeweiligen Constraints erfüllen.

Wird ein neuer Constraint hinzugefügt bzw. ein bestehender entfernt, versucht der Algorithmus, die nicht erfüllten Constraints durch die Konstruktion von *Methoden-Ketten* zu erfüllen. Können in einem Methodengraphen keine zusätzlichen Constraints durch Methoden-Ketten erfüllt werden, ist dieser optimal.

Methoden-Ketten Um einen Constraint zu erfüllen (den *Ausgangsconstraint* der Kette), wählt „SkyBlue" eine Methode dieses Constraints. Steht diese Methode in Konflikt mit anderen, bereits erfüllten Constraints, werden diese zunächst aufgehoben. Gleichstarke oder stärkere Constraints müssen wieder erfüllt werden, indem für sie andere Methoden gewählt werden, die nicht mit dem Ausgangsconstraint kollidieren. Diese neu gewählten Methoden können wiederum in Konflikt zu anderen Constraints stehen, für die dann neue Methoden gewählt werden, so daß eine Kette von Constraints mit neu gewählten Methoden entsteht.

Dies geht im Detail folgendermaßen: Anfangs besteht die Kette nur aus dem Anfangsconstraint (AC). Constraints, die dazu in Konflikt stehen, werden aufgehoben und in die Kette eingereiht. Dabei werden Constraints, die gleich oder stärker als der AC sind, wieder erfüllt, indem für sie andere Methoden gewählt werden, die mit keinem Constraint der Kette kollidieren. Hierbei können durch Konflikte weitere Constraints die Kette verlängern. Konnten auf diese Weise alle stärkeren Constraints in der Kette erfüllt werden, ist der AC erfüllbar und der Methodengraph kann entsprechend angepaßt werden.

Sollte beim Verfolgen der Kette für einen Constraint keine Methode wählbar sein, die mit der gesamten Kette kollisionsfrei ist, erfolgt ein Backtracking-Schritt: Der Erfüllungsvorgang wird schrittweise zurückgenommen, bis für einen Constraint eine andere Methode gewählt werden kann, mit der dann weiterverzweigt wird. Sollte keine konfliktfreie Kette möglich sein, kann der Ausgangsconstraint nicht erfüllt werden, der Methodengraph bleibt unverändert.

In Abbildung 2.7(a) ist der Methodengraph zum Beispiel aus Abschnitt 2.1.3.2 zu sehen. Der starke Constraint C_2 ist nicht erfüllt (dargestellt durch die gestrichelten Linien) und soll eingeplant werden. Dies kann geschehen, indem C_3 zunächst aufgehoben wird. Damit kann C_2 die Variable V_2 bestimmen (V_1 ist hierfür die Eingabe), siehe Abbildung 2.7(b). Da aber C_3 stärker ist, muß dieser seinerseits wieder erfüllt werden. Dies kann geschehen, indem C_4 außer Kraft gesetzt wird. Da es für C_4 keine alternative Methode gibt, und C_2 stärker ist als C_4, ist in Abbildung 2.7(c) ein optimaler Methodengraph erreicht.

Ebenso wäre es im ersten Schritt möglich gewesen, C_1 aufzuheben und zum optimalen Graphen 2.7(d) zu gelangen. In beiden Fällen ist ein Constraint der Stärke 1 nicht erfüllt, im Vergleich zu 3 im Ausgangsgraphen (a).

Worst case: exponentielle Laufzeit Da das allgemeine CSP NP-hart ist (siehe Abschnitt 2.1.7), liegt die Vermutung nahe, daß auch das hier vorgestellte

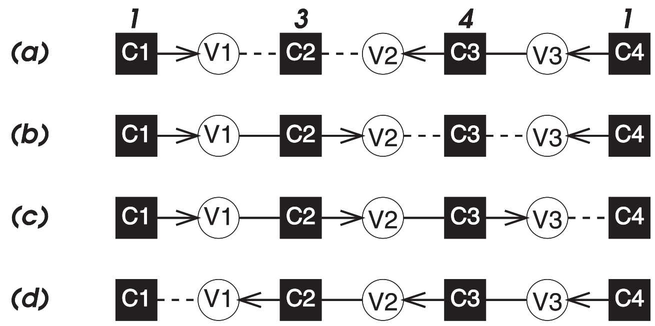


Abbildung 2.7: Optimierung eines Methodengraphs

Verfahren im worst case exponentielle Laufzeit hat. In Abbildung 2.8 ist ein Beispiel hierfür angegeben:

Um den Constraint C_1 zu erfüllen, muß für C_2 eine andere Methode gewählt werden. Da C_2 bis C_4 aus je zwei Variablen die Werte der anderen beiden ermitteln können, gibt es hierfür verschiedene Möglichkeiten. Eine Möglichkeit ist hier mit b bezeichnet. Für C_3 könnte jetzt ebenfalls b gewählt werden, bei C_4 scheitert allerdings sowohl b als auch die Alternative c (in der ein Pfeil auf die andere Variable von C_5 zeigt) am Konflikt mit dem stärkeren C_5 (Abbildung 2.8(b)).

Im Backtrackingschritt wird für C_3 die Möglichkeit c geprüft, was ebenfalls scheitert, egal welche Methode für C_4 gewählt wird (Abbildung 2.8(c)). Erneutes Backtracking probiert die Methode c für C_2 und nacheinander b und c für C_3. Dies scheitert analog (Abbildung 2.8(d) und (e)).

Es wurden also für C_2 zwei Methoden (b und c) probiert, für jeden dieser Fälle zwei Methoden für C_3 und dafür wiederum jeweils zwei Methoden für C_4. Es mußten insgesamt also 2^3 mögliche Methodenkombinationen geprüft werden, bevor feststand, daß C_1 nicht erfüllt werden kann. Es ist leicht zu sehen, daß C_2 bis C_4 durch n Constraints gleicher Art ersetzt werden können, wodurch die Suche 2^n Schritte dauern würde.

Möglichkeiten zur Steigerung der Effizienz Größere Methodengraphen haben deutlich mehr Konflikte zur Folge, was sich negativ auf die Laufzeit auswirkt. Als Maßnahmen zur Einschränkung der jeweils zu betrachtenden Constraints kommen hier in Frage:

1. **Schrittweises Propagieren in Flußrichtung:** Analog zur Weitergabe von Werten innerhalb eines Constraintnetzes kann nach dem Hinzufügen eines Constraints von den Variablen ausgegangen werden, die dadurch neu bestimmt wurden. Nur diejenigen Variablen, Methoden und Constraints, die in Pfeilrichtung erreichbar sind, müssen neu betrachtet werden.

2. **Unerfüllte Constraints nach Stärke abarbeiten:** Konnte ein Constraint durch Konstruktion einer Methodenkette erfüllt werden, so wurden notwendigerweise alle stärkeren Constraints auch erfüllt (siehe Arbeitsweise des Algorithmus in 2.1.5.2). Unerfüllt bleiben damit nur Constraints, die schwächer als der Ausgangsconstraint der Kette sind.

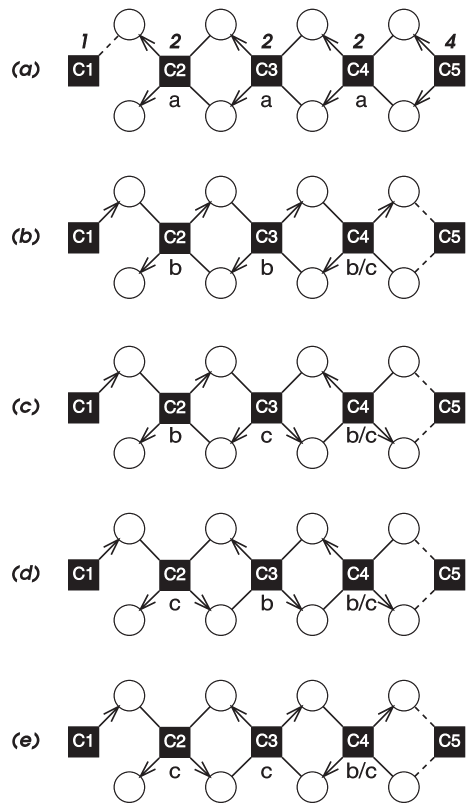


Abbildung 2.8: Beispiel für den Worst case

Versucht man also, die unerfüllten Constraints in der fallenden Reihenfolge ihrer Stärke zu erfüllen, muß jeder dieser Constraints nur einmal bearbeitet werden. Insbesondere kann abgebrochen werden, sobald an einer Stelle keine Methodenkette mehr möglich ist, da dann für schwächere Constraints erst recht keine Kette erzeugt werden kann (die ja den momentanen, stärkeren Constraint mit erfüllen müßte).

3. **Änderungsstärken für Variablen berechnen:** Soll durch einen neu eingefügten Constraint C_1 ein anderer Constraint C_2 aufgehoben werden (da dieser dieselbe Variable in seinem Ausgabebereich hatte), setzt dies voraus, daß in der entstehenden Methodenkette sowohl für C_2 als auch für folgende abhängige Constraints, die stärker als C_1 sind, alternative konfliktfreie Methoden möglich sind. Wenn bereits im voraus bekannt wäre, ob dies für eine bestimmte Stärke von C_1 möglich ist, könnte der aufwendige erfolglose Versuch, eine Kette zu konstruieren, entfallen.

   Dies wird durch das Ermitteln und Speichern von *Änderungsstärken* zu jeder Variable möglich. Die Änderungsstärke ist die Stärke des schwächsten Constraints, der aufgehoben werden müßte, damit die Variable nicht mehr durch Constraints bestimmt ist, und somit durch einen neu hinzukommenden Constraint bestimmt werden kann.

```
function Änderungsstärke(constraint C, variable v)
  min := Stärke(C)
  Für alle Methoden m zum Constraint C
    v_m := Ausgabevariable von m
    if (Stärke(v_m) < min) min := Stärke(v_m)
  end für alle
  return min
end function
```

Damit kann direkt ermittelt werden, ob das Aufheben eines Constraints zum Zwecke der Bestimmung einer Variablen durch einen neuen Constraint Erfolg hat: Ist die Änderungsstärke der Variablen kleiner als die Stärke des neuen Constraint, kann die Änderung gewinnbringend durchgeführt werden, sonst ist sie nicht möglich.

2.1.6 Einfluß des Benutzers

In den vorherigen Abschnitten wurde gezeigt, wie Constraint Solver versuchen, die Effizienz der Lösungssuche durch geeignete Reduzierung von Wertemengen oder zu betrachtenden Variablen zu steigern. Dabei ist es wichtig, sich nicht nur auf die Fähigkeiten des Solvers zu verlassen. Durch geschickte Wahl des Datenmodells kann die Zahl der Constraints gering gehalten werden, indem manche Bedingungen als *implizite Constraints* bereits durch das Modell sichergestellt werden:

- Soll der Wert einer Variable nicht verändert werden, so braucht dafür kein neuer Constraint eingefügt zu werden, sondern man entfernt diese Variable aus der Variablenmenge des Solvers, dem *working set*.

- Solche unveränderlichen Variablen müssen unabhängig von anderen Variablen sein. Dies läßt sich oft durch Wahl der passenden numerischen Repräsentation für ein Datenobjekt erreichen. In Abbildung 2.9 sollte die Beschreibung der Schwungstange durch Anfangspunkt, Winkel und Länge (statt durch Anfangs- und Endpunkt) erfolgen. Winkel und Länge als Konstanten können aus dem working set entfernt werden.

- Wertegleichheit läßt sich durch *merging* erreichen: Variablen, die gleich sein sollen, werden zusammengefaßt, so daß alle Constraints, die diesen Wert benutzen, auf dieselbe Variable zugreifen. Die einheitliche Größe der verschiedenen Räder in Abbildung 2.9 könnte somit durch einen einzigen Wert repräsentiert werden (statt einem pro Rad mit zugehörigem Gleichheitsconstraint).

Implizite Constraints sind im Unterschied zu explizit numerisch angegebenen Constraints exakt. Dies kann von Vorteil sein, hat aber auch den Nachteil, daß der Solver keine Wertetoleranzen zur Beschleunigung der Suche nach einer nahezu optimalen Lösung ausnutzen kann.

2.1.7 Komplexität

Die vorgestellten Constraint-Solving-Verfahren versuchen durch eine geschickte Steuerung der Lösungssuche die Laufzeit zu verringern. Verschiedene Heuristische Verfahren kommen zum Einsatz oder könnten an passenden Stellen noch

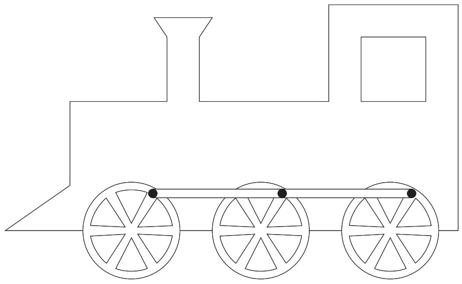


Abbildung 2.9: Beispiel für implizite Constraints

eingebaut werden. Dies ist für eine akzeptable Laufzeit vermutlich unumgänglich, denn es gilt:

Satz 2.1.4 CSP ist NP-hart.

Beweis 2.1.4 3SAT $\leq_p$ CSP
Eingabe für 3SAT ist eine logische Formel in konjunktiver Normalform mit höchstens drei Variablen pro Klausel. Diese Formel läßt sich direkt auf ein CSP abbilden: Die Variablen des CSP sind genau die logischen Variablen aus 3SAT. Jede Klausel ergibt einen Constraint, der prüft, ob die Klausel erfüllt ist. Dieses CSP hat genau dann eine Lösung, wenn die Formel erfüllbar ist.

2.2 Objektorientierte Spezifikation nach der Booch-Methode (Lars Hermes)

2.2.1 Einleitung

Heutige (industriell) erstellte Software-Systeme haben eine derartige Komplexität, daß es für einen einzelnen Entwickler nicht mehr möglich ist, diese zu überblicken und zu verstehen, geschweige denn, ein solches System alleine zu entwickeln. Man betrachte z.B. ein Software-System, das die Abläufe in einem Kraftwerk steuert.
Wie kann dieses Problem nun gelöst werden? Ganz einfach, indem das System in kleinere, handhabbare Teile zerlegt wird und auf diese Teile Entwicklungsteams angesetzt werden. Wie kann diese Zerlegung nun erfolgen? Da gibt es zwei unterschiedliche Ansätze:

- Algorithmische Zerlegung
- Objektorientierte Zerlegung

2.2.1.1 Algorithmische Zerlegung

Die algorithmische Zerlegung betrachtet die Elemente Datenstrukturen und Algorithmen als getrennte Einheiten. Sie setzt ihre Schwerpunkte auf die Konzentration der Anordnung der Ereignisse, d.h., daß die Daten durch Prozeduren manipuliert werden, und auf die Zusammenstellung von Modulen, die jedes für sich einen wichtigen Abschnitt im Gesamtablauf des Systems darstellen.

2.2.1.2 Objektorientierte Zerlegung

Die objektorientierte Zerlegung betrachtet die Datenstrukturen und Algorithmen als zusammengehörende Elemente. Genauer: die Daten sind im Objekt eingeschlossen und können nur durch die Methoden des Objekts, die die Schnittstelle nach außen darstellen, manipuliert werden. Welche Schwerpunkte hat die objektorientierte Zerlegung nun? Zum einen werden Objekte verwendet, die Aktionen auslösen oder von Aktionen anderer Objekte betroffen sind. Zum anderen wird das zu zerlegende Problem als Menge autonomer Bestandteile (= Objekte) betrachtet.

2.2.2 Das Objektmodell

2.2.2.1 Überblick

Objekte Das Objektmodell beinhaltet die Gesamtheit der Elemente des objektorientierten Entwurfs. Objektorientiert ist als Schlagwort heute in aller Munde. Trotzdem gibt es keine einheitliche Definition, da es sich aus verschiedenen Quellen ableitet. Das Konzept des Objekts bildet jedoch das Fundament der Objektorientierung. Informell läßt sich das Objekt als greifbare Einheit definieren, die ein wohldefiniertes Verhalten zeigt. Stefig und Bobrow gehen in ihrer Definition noch weiter und bezeichnen Objekte als „Einheiten, die die Eigenschaften von Prozeduren und Daten kombinieren; sie führen Programmschritte durch und speichern einen lokalen Status“[SB84]. Tiefer geht die Definition von Jones: „ … Objekte besitzen eine gewisse Integrität, die nicht verletzt werden sollte .... Ein Objekt kann nur seinen Status verändern, sein Verhalten aufzeigen, manipuliert werden oder in einer Beziehung zu anderen Objekten stehen, immer in genau der Art und Weise, die dem Objekt entspricht. Anders ausgedrückt, es gibt invariante Eigenschaften, die ein Objekt und sein Verhalten charakterisieren."[Jon79]

OOP

Definition 2.2.1 Objektorientierte Programmierung ist eine Implementierungsmethode, bei der Programme als kooperierende Ansammlung von Objekten angeordnet sind. Jedes dieser Objekte stellt eine Instanz einer Klasse dar, und alle Klassen sind Elemente einer Klassenhierarchie, die durch Vererbungsbeziehungen gekennzeichnet ist.[Boo94]

Aus dieser Definition lassen sich drei wichtige Bestandteile ableiten:

1. Grundbausteine sind Objekte und nicht Algorithmen

2. Jedes Objekt ist eine Instanz einer Klasse

3. Klassen stehen miteinander in Vererbungsbeziehung

OOD

Definition 2.2.2 Objektorientiertes Design ist eine Designmethode, die den Prozeß der objektorientierten Zerlegung beinhaltet, sowie eine Notation für die Beschreibung der logischen und physikalischen, wie auch statischen und dynamischen Modelle des betrachteten Systems.[Boo94]

Wichtig hierbei ist, daß objektorientiertes Design

1. zu einer objektorientierten Zerlegung führt und

2. unterschiedliche Notationen benutzt werden, um neben den statischen und dynamischen Aspekten des Systems auch die verschiedenen Modelle des logischen (Klassen- und Objektstruktur) und physikalischen (Modul- und Prozeßarchitektur) Designs eines Systems zu beschreiben.

OOA

Definition 2.2.3 Die objektorientierte Analyse ist eine Analysemethode, die die Anforderungen aus der Perspektive der Klassen und Objekte, die sich im Vokabular des Problembereichs finden, betrachtet.[Boo94]

Die OOA legt größten Wert auf die Erzeugung von Modellen der realen Welt, und zwar mit Hilfe einer objektorientierten Sicht der Welt.

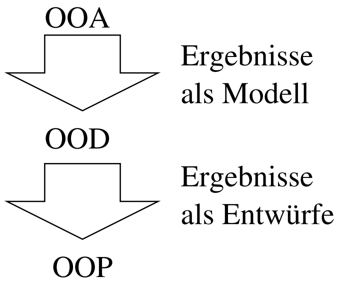


Abbildung 2.10: Zusammenhang OOA-OOD-OOP

2.2.2.2 Elemente des Objektmodells

Die vier Hauptelemente des Objektmodells sind:

- Abstraktion
- Kapselung
- Modularität
- Hierarchie

Wenn ein *Hauptelement* fehlt, handelt es sich nicht mehr um Objektorientierung Weiterhin gibt es drei weniger wichtige Elemente:

- Typisierung
- Nebenläufigkeit
- Persistenz

Diese Elemente sind zwar nützlich, aber kein unentbehrlicher Bestandteil des Objektmodells.

Abstraktion Bei der Abstraktion handelt es sich um eine Verallgemeinerung, bei der man sich auf die wesentlichen Details eines Objektes konzentriert und die weniger wichtigen ignoriert. Unter wichtigen Details versteht man diejenigen Charakteristika, die die Eigenschaften eines Objektes ausmachen – die Einteilung in wichtige und unwichtige Details hängt selbstverständlich von der Perspektive des Betrachters ab. Die Abstraktion macht das Verständnis der Zusammenhänge eines komplexen Objektes einfacher.

Kapselung Bei der Kapselung handelt es sich um eine Methode, die Details der Objekt-Implementierung zu verbergen, man bezeichnet sie deshalb auch als „information hiding“. Die Kapselung verbietet also einen Blick in die Interna (das sind i.a. die Daten oder spezielle Methoden) eines Objektes. Auf diese verborgenen Elemente kann von außen nur über die Schnittstelle des Objektes zugegriffen werden; dabei handelt es sich um die Methoden, die ein Objekt zur Verfügung stellt.

Modularität Die Modularität packt Abstraktionen in eigenständige Einheiten. Diese Einheiten sollten funktional zusammengefaßt werden.

Hierarchie Die Abstraktionen werden mittels der Modularität in funktionale Einheiten gepackt. Diese Einheiten bilden eine Hierarchie. Mit anderen Worten: Die Abstraktionen werden von oben nach unten immer detaillierter.

Typisierung Strenge Typisierung verhindert die Vermischung von Abstraktionen. Die Typisierung hilft bei der Einschränkung von Abstraktionen, d.h. sie stellt eine Art Überwachungsmechanismus dar, der kontrolliert, daß die Abstraktionen nur in dem für sie vorgesehenen Aktionsrahmen agieren.

Nebenläufigkeit Mit der Nebenläufigkeit wird es verschiedenen Objekten erlaubt, gleichzeitig in einem System zu agieren. Dies ist natürlich nur möglich, solange die parallel ablaufenden Objekte nicht voneinander abhängen.

Persistenz Die Persistenz ist ein Mechanismus, der das „Überleben“ eines Objektes durch Raum und Zeit ermöglicht. Wobei mit Raum gemeint ist, daß das Objekt seinen Speicherplatz z.B. vom Hauptspeicher auf Diskette wechseln kann, ohne seine Informationen zu verlieren. Über die Zeit hinweg bedeutet, daß das Objekt seinen Erzeuger überleben kann, d.h. daß ein Objekt weiterexistieren kann, obwohl sein Aufrufer bereits nicht mehr existiert.

2.2.3 Objekte und Klassen

2.2.3.1 Objekte

Was sind Objekte? Jedes Objekt besitzt einen Status, ein Verhalten und eine Identität:

Status Unter Status versteht man die unveränderlichen Eigenschaften eines Objekts plus deren dynamischen Werte. Betrachtet man z.B. das Objekt Aufzug: Dessen unveränderliche Eigenschaften sind, daß er sich nur in vertikaler, nicht aber in horizontaler Richtung bewegen kann, seine dynamischen Werte können z.B. das aktuelle Stockwerk sein und das nächste, an dem er halten soll.

Verhalten Das Verhalten bezeichnet die Art und Weise, wie das Objekt durch Statusänderungen oder durch Nachrichtenübergabe agiert bzw. reagiert. Beim Beispiel des Aufzugs wäre dessen Verhalten eine Bewegung nach oben bzw. nach unten, das Verhalten eines Objekts hat also Einfluß auf die dynamischen Werte seines Status'.

Identität Die Identität eines Objekts trägt zu seiner Einzigartigkeit bei. Wie man später sehen wird, handelt es sich bei Objekten um Instanzen von Klassen. Diese Klassen geben ihren Instanzen ihren Status und ihr Verhalten vor. Durch die Identität werden Objekte von derselben Klasse voneinander unterscheidbar. Beim Aufzug wäre das z.B. der Aufzug des Eiffelturms oder der Aufzug in der Fakultät Informatik. Beide haben denselben Status und dasselbe Verhalten, aber jeder eine eigene Identität.

Zusammenarbeit zwischen Objekten Zwischen Objekten gibt es zwei Arten von Zusammenarbeit:

- Links
  Einfache Zusammenarbeit zwischen Objekten. Es handelt sich hierbei um eine gleichberechtigte Client-/Supplier-Beziehung.

- Aggregation
  Diese gibt eine Ganzes-/Teil-Hierarchie zwischen Objekten an, d.h. ein Aggregat-Objekt umschließt ein Attribut-Objekt, das Bestandteil des Aggregats ist. Diese Bestandteil-Beziehung kann (muß aber nicht) physikalischer Natur sein. Z.B. bilden bei einem Objekt Aktionär die Aktien, die er besitzt, das Attribut, diese sind jedoch nicht physikalischer Bestandteil des Aktionärs. Anders sieht es bei einem Objekt Flugzeug aus, dessen Tragfächen einen physikalischen Bestandteil bilden.

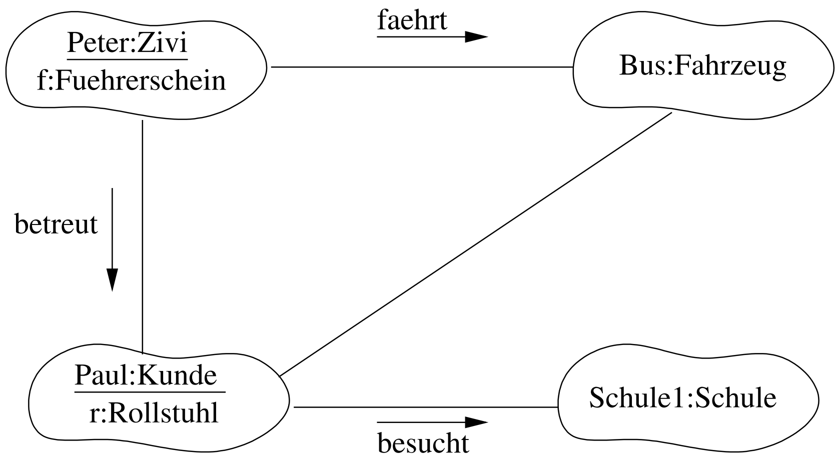


Abbildung 2.11: Beispiel für die Zusammenarbeit zwischen Objekten (bei den Objekten Zivildienstleistender (Zivi) und Kunde handelt es sich um Aggregate)

2.2.3.2 Klassen

Was ist eine Klasse? Eine Klasse gibt eine gemeinsame Struktur und ein gemeinsames Verhalten für deren Instanzen (= Objekte) vor. Eine Klasse kann im Prinzip als Schablone für das von den Objekten realisierte Verhalten und deren Eigenschaften gesehen werden.

Zusammenarbeit zwischen Klassen Klassen können auf verschiedene Arten in Beziehung miteinander stehen:

- Assoziation
- Vererbung
- Aggregation
- Verwendung
- Instantiierung
- Metaklasse

Assoziation Einzelne Klassen stehen in Verbindung miteinander. Diese Abhängigkeiten lassen sich durch Rollen darstellen, die die Klassen in ihrer Beziehung miteinander übernehmen. Weiter kann der Assoziation zweier Klassen eine Kardinalität zugeordnet werden. Die Kardinalität zeigt, daß für die Anzahl von Instanzen der einen Klasse eine oder mehrere Instanzen der anderen Klasse vorhanden sein können.

Vererbung Klassen können voneinander abgeleitet werden, d.h. die abgeleitete Klasse (= Subklasse) „erbt" die Attribute und Methoden von der Vorgängerklasse (= Superklasse). Es gibt zwei Arten von Vererbung : die Einfachvererbung, bei der jede Subklasse von genau einer Superklasse erben kann, oder die Mehrfachvererbung, bei der jede Subklasse mehrere Superklassen haben kann.

Aggregation Für die Aggregation von Klassen kann eine direkte Parallele zur Aggregation von Objekten gezogen werden, d.h. die Aggregation von Klassen gibt, genauso wie die Aggregation zwischen Objekten, eine Ganzes-/Teilhierarchie zwischen Klassen an. Im Unterschied zu der Assoziation, die keine Richtung besitzt, besitzt die Aggregation eine Richtungsangabe. Aggregate dürfen nicht zyklisch sein.

Verwendung Bei der Verwendung handelt es sich um eine Verfeinerung der Assoziation. Man spricht von einer Client-/Supplierbeziehung, d.h. eine Klasse bietet Dienste an (Supplier), die eine andere Klasse benutzt (Client).

Instantiierung Eine parametrisierte Klasse stellt eine Klassenfamilie dar, deren Verhalten unabhängig von ihrem formalen Parameter (= abstrakter Datentyp) definiert ist. Durch die Instantiierung einer Klasse wird deren formaler Parameter durch einen aktuellen Parameter (= realer Datentyp) ersetzt. Bei der Instantiierung handelt es sich somit um die Bildung einer konkreten Klasse (Vergleichbar mit dem Konzept der Templates in C++).

Metaklasse Eine Metaklasse bezeichnet die Klasse einer Klasse. Von einer Metaklasse dürfen keine Instanzen gebildet werden. Hauptsächlicher Verwendungszweck von Metaklassen: Zur Verfügung Stellen von Klasseninstanzvariablen (Dieses Konzept ist vergleichbar mit den statischen Elementen in C++).

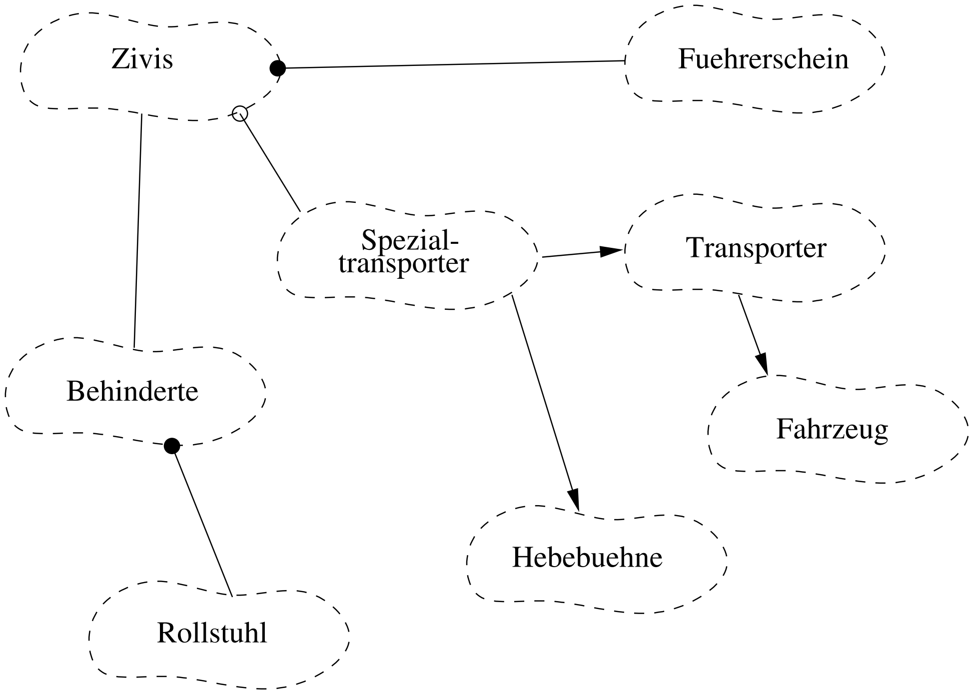


Abbildung 2.12: Beispiel 1

In Beispiel 1 stehen die Klassen Zivis und Behinderte in Beziehung. Die Klassen Zivis und Führerschein bilden eine Aggregation, ebenso wie die Klassen Behinderte und Rollstuhl. Die Klasse Zivis verwendet die Klasse Spezialtransporter. Spezialtransporter erbt von den Klassen Transporter und Hebebühne, die Klasse Transporter erbt wiederum von der Klasse Fahrzeug.

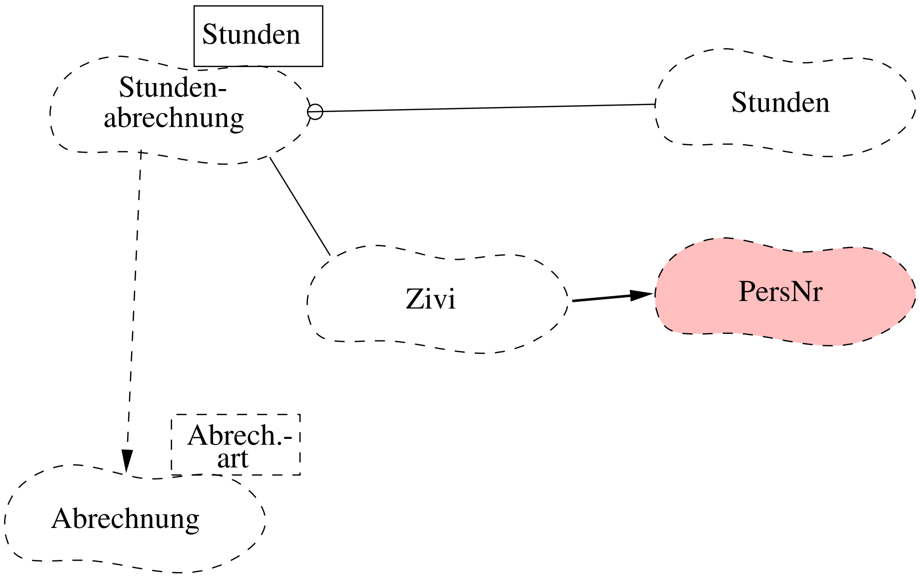


Abbildung 2.13: Beispiel 2

In Beispiel 2 wird die Klasse Stundenabrechnung von der Klasse Abrechnung instantiiert, in dem sie die Klasse Stunden verwendet. Die Metaklasse der Klasse Zivi bietet die Klasseninstanzvariable PersNr. an, die im System nur einmalig vorkommen darf.

2.2.3.3 Klassen und Objekte in Analyse und Design

Für den Entwickler stellen sich während der Analyse und in der frühen Designphase zwei grundlegende Fragen:

- Finden der Klassen und Objekte, die das Vokabular des Problembereichs bilden. Die gefundenen Klassen und Objekte nennt man auch die Schlüsselabstraktionen des Systems.

- Gruppierung der einzelnen Objekte mit anschließendem Aufstellen einer Struktur, in der diese Objektgruppen zusammenarbeiten und so dem von den Problemanforderungen erwünschten Verhalten gerecht werden. In diesem Schritt werden also die Mechanismen gefunden, die zur Problemlösung beitragen.

2.2.4 Klassifizierung

2.2.4.1 Probleme bei der Klassifizierung

Welche Möglichkeiten gibt es, um die Schlüsselabstraktionen und die Mechanismen eines Problems zu finden? Oder einfacher ausgedrückt, wie findet man die passenden Klassen bzw. Objekte, und wie arbeiten diese zusammen? Um diese Frage zu klären, sollte man sich zuerst Gedanken darüber machen, welche Probleme bei dem Finden von Klassen auftreten können. Unterhalten sich mehrere

Personen über ein Problem, von dem sie versuchen, dessen Schlüsselabstraktionen zu finden, wird jeder einzelne auf seine eigene Lösung kommen. Es müssen also Strategien angewendet werden, um das Finden der Schlüsselabstraktionen und Mechanismen im Entwicklungsteam zu steuern.

2.2.4.2 Analyseansätze

Bei der Analyse gibt es keine deterministische Vorgehensweise. Das Entdecken geeigneter Klassen und Objekte zur Modellierung des Problems während der Analysephase und das Finden geeigneter Abstraktionen und Mechanismen zur Realisierung des gewünschten Verhaltens hängen weitgehend von der Erfahrung der Entwickler ab. Die vorgestellten Analyseansätze sollen beispielhaft skizzieren, wie geeignete Klassen und Objekte gefunden werden können.

- klassischer Ansatz
- Verhaltensanalyse
- Bereichsanalyse
- Gebrauchsfall-Analyse
- informelle sprachliche Beschreibung
- CRC-Karten
- (Strukturanalyse)

Klassischer Ansatz Mit diesem Ansatz werden Klassen abgeleitet, indem die real existierenden Objekte und Verantwortlichkeiten des zu modellierenden Systems direkt übernommen werden.

- reale Dinge (z.B. Fahrzeuge, Touren ...)
- Rollen (z.B Pfleger, Fahrer ...)
- Ereignisse (z.B. Transport ...)
- Interaktionen
- Organisatorische Einheiten
- ...

Verhaltensanalyse Bei diesem Ansatz bildet man Klassen derart, daß Objekte die ein ähnliches Verhalten aufzeigen, in einer Klasse zusammengefaßt werden. Es wird das dynamische Verhalten betrachtet, d.h. welchen Zweck erfüllen die Objekte innnerhalb des Systems? Durch diesen Ansatz könnte man auf folgende Klassenkandidaten kommen:

- Zivis, die Schulfahrten erledigen
- Zivis, die Essen ausfahren
- Planer, die Touren zusammenstellen
- Fahrzeuge, die Rollstuhlfahrer transportieren
- ...

Bereichsanalyse Die Bereichsanalyse bildet Klassen, deren Objekte sich einem gemeinsamen Bereich zuordnen lassen. Unter Bereich werden verwandte Bestandteile innerhalb des zu modellierenden Systems verstanden. Diese Methode eignet sich i.a. dazu, neue Schlüsselabstraktionen zu finden, wenn während der Analyse ein Stillstand auftritt, indem Bereichsexperten hinzugezogen werden, d.h. Fachleute, die mit dem zu modellierenden Problem tagtäglich zu tun haben. Durch die Bereichsanalyse könnten folgende Klassen gefunden werden:

- Dialysepatienten
- Schüler einer bestimmten Schule
- Schulwege zu einer bestimmten Schule
- Transporter für Rollstuhlfahrer
- Zivis mit spezieller Ausbildung im Pflegedienst
- ...

Gebrauchsfall-Analyse Unter Gebrauchsfall versteht man ein Muster für die jeweilige Anwendung in Teilbereichen des Problems. Die Gebrauchsfälle müssen jedoch wieder durch die anderen Ansätze auf einzelne Klassenkandidaten untersucht werden. Ein Beispiel für einen Gebrauchsfall wäre z.B.: „Ein Zivi startet morgens am DRK, holt unterwegs eine bestimmte Anzahl von Schülern ab, in dem er einer vorgegebenen Tour folgt, bringt diese zur Schule und fährt zurück zum DRK.“

Informelle sprachliche Beschreibung Bei dieser Analysemethode beschreibt man das Problem, indem ein in Umgangssprache verfaßter Text erstellt wird, der die Anforderungen genau beschreibt. Kandidaten für Klassen sind hierbei die Substantive innerhalb des Textes und die Verben könnten potentielle Methoden darstellen. Diese Methode ist mit Vorsicht zu genießen, da umgangssprachliche Beschreibung ein höchst ungenaues Mittel zur Beschreibung eines Problems ist. Desweiteren muß man auch darauf achten, daß Verben substantiviert werden können. Ein Vorteil dieser Methode ist, daß der Entwickler gezwungen wird, im Vokabular des Problembereichs zu denken.

CRC-Karten Diese Methode ist keine Analysemethode im eigentlichen Sinne. Sondern vielmehr eine Möglichkeit, die anderen Analyseansätze visuell zu unterstützen. CRC steht für <u>C</u>lass – <u>R</u>esponsibilities – <u>C</u>ollaborators, also für Klasse – Verantwortlichkeit – Beteiligung. Wie kann man die anderen Ansätze damit unterstützen? CRC-Karten sind im Prinzip Karteikarten, bei denen oben der Klassenname steht, auf der linken Seite die Verantwortlichkeiten (was macht die Klasse?) und auf der rechten Seite die Beteiligungen (welche Klassen sind noch betroffen?). Zur besseren Übersicht können die Karten farblich unterschiedlich sein und an eine Pinnwand geheftet werden (zusätzlich können diejenigen Klassen mit Linien verbunden werden, die zusammenarbeiten).

Strukturanalyse Die Strukturanalyse ist nur bedingt für die objektorientierte Spezifikation einsetzbar. Da sie ihre Ursprünge in der strukturierten Analyse bzw. strukturierten Programmierung hat. Hier wird das Modell des Systems mittels Datenflußdiagrammen erstellt. Was könnte nun eine Klasse sein?

- externe Entitäten
- Datenelemente
- Steuerungselemente
- ...

Abschließend läßt sich sagen, daß keine dieser Analysemethoden für sich alleine ausreichen wird, um ein vollständiges Modell und die ersten Designansätze für ein System zu liefern. Eine Kombination sollte jedoch eine gute Grundlage liefern.

2.2.4.3 Zentrale Fragen der Analyse

Bevor man sich jedoch darauf stürzt, irgendwelche Klassen zu finden, die das System modellieren könnten, ist es nötig, sich zu überlegen, welche Ziele mit der Analyse verfolgt werden sollten. Es sind hierbei zwei wichtige Punkte zu nennen:

- Was ist das gewünschte Verhalten des Systems, d.h. was soll das System leisten, was wird von ihm erwartet?
- Was sind die Rollen und Verantwortlichkeiten der Objekte, die dieses Verhalten realisieren? Diese Frage erfordert natürlich, daß schon eine gewisse Vorstellung der Objekte vorhanden ist.

2.2.4.4 Zentrale Fragen des Designs

Die Aufgabe des Designs besteht darin, die in der Analyse gefundenen Schlüsselabstraktionen und Klassen sinnvoll im System unterzubringen und gegebenenfalls zu erweitern. Dabei sollte zuerst überlegt werden, wie das Design auszusehen hat und was damit bezweckt werden soll.

- Welche Klassen existieren und in welcher Beziehung stehen diese Klassen? Diese Frage läßt sich nicht eindeutig dem Design zurechnen, da auch schon während der Analyse Klassen entdeckt bzw. erfunden werden. Dieses Problem, daß Analyse und Design nicht sauber zu trennen sind, wird jedoch noch häufiger auftreten.
- Welche Mechanismen werden verwendet, um die Zusammenarbeit der Objekte zu steuern?
- Wo sollen die einzelnen Klassen und Objekte deklariert werden?
- Welchem Prozessor sollte ein Prozeß zugeordnet werden, und wie sollte für einen bestimmten Prozessor die zeitliche Reihenfolge der verschiedenen Prozesse festgelegt werden?

2.2.5 Notation

2.2.5.1 Klassendiagramme

Wozu dienen die Klassendiagramme? Sie dienen der Darstellung der Existenzen von Klassen und deren Beziehungen miteinander. Das Klassendiagramm erzeugt eine logische Sicht des zu entwickelnden Systems. In der Analysephase hilft das Klassendiagramm, die Klassen und deren Verantwortlichkeiten darzustellen. Während des Designs zeigt das Klassendiagramm die Struktur des Systems.

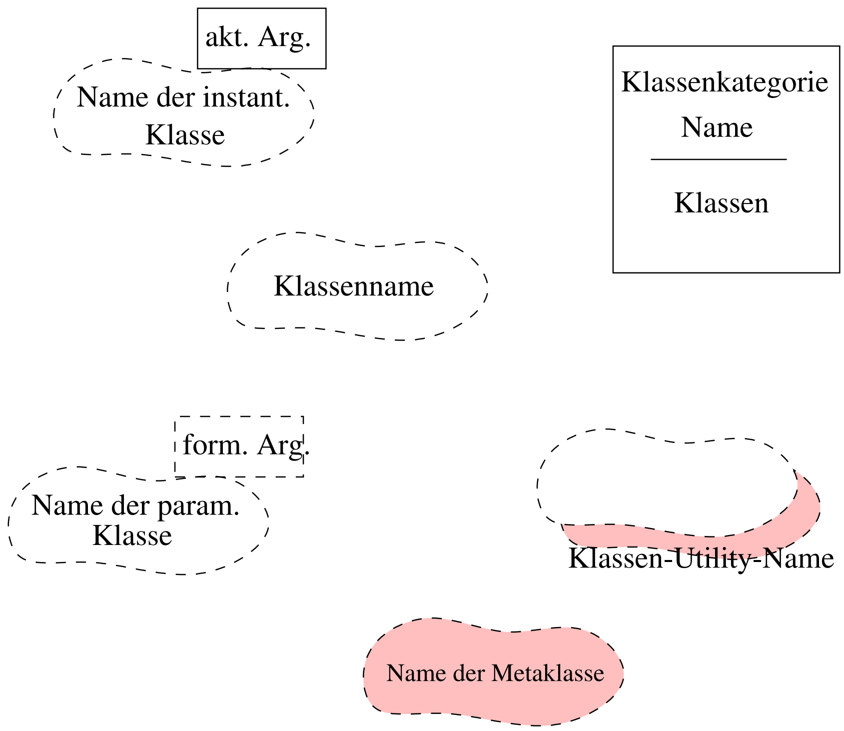


Abbildung 2.14: Klassen-Icons

Klassen-Icons Bis auf Klassenkategorien und Klassenutilities sind die Bedeutungen der einzelnen Icons bereits bei der Zusammenarbeit von Klassen erklärt worden. Ein Klassenutility bietet globale Attribute und Methoden an, auf die alle Klassen zugreifen können. Die Klassenkategorien helfen bei der Strukturierung der Klassendiagramme. In einer Klassenkategorie werden diejenigen Klassen zusammen gefaßt, die im Klassendiagramm eine funktionale Einheit bilden, d.h. Klassenkategorien bilden Klassenbibliotheken. Im Bsp. verwendet die Klassenbiblithek „Fahrdienste des DRK“ zum Beispiel die GUI-Bibliothek, um die graphische Oberfläche darzustellen.

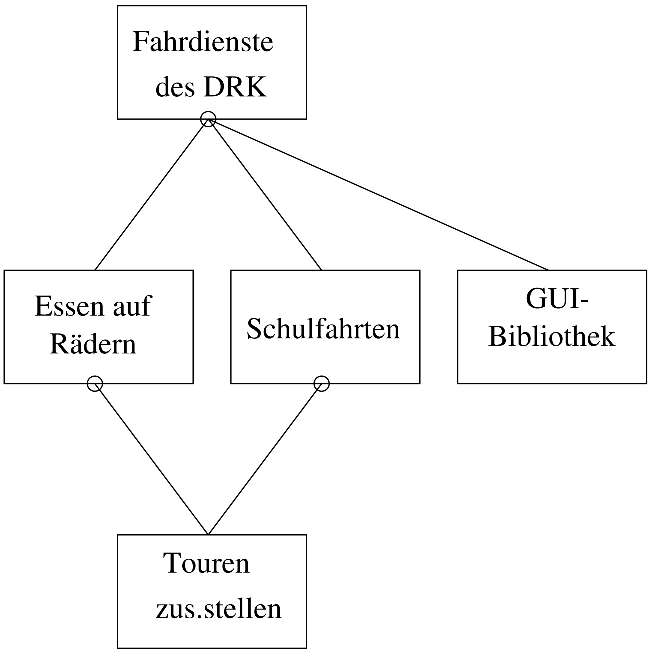


Abbildung 2.15: Beispiel für ein Klassenkategorie-Diagramm

Klassenbeziehungen Arten von Klassenbeziehungen :

- Assoziation:
  Einfache Beziehungen zwischen Klassen untereinander. Eine Klasse kann auch eine Assotiation zu sich selbst besitzen (= reflexive Assoziation).
- Vererbung ("is-a"-Beziehung):
  Der Pfeil zeigt auf die Oberklasse. Vererbungsbeziehungen dürfen keine Zyklen enthalten.
- Eigentum (Aggregation oder „has“-Beziehung):
  Der ausgefüllte Kreis am Ende der Assoziation gibt das Aggregat an. Die Klasse am anderen Ende gibt den Teil an, dessen Instanzen im Aggregat-Objekt enthalten sind.
- Verwendung (Client-/Supplier-Beziehung):
  Der Kreis am Ende der Assoziation gibt den Client an, der auf irgendeine Art und Weise vom Supplier abhängig ist, um bestimmte Dienstleistungen zu realisieren (z.B. Operationen der Clientklasse rufen Operationen der Supplierklasse auf).
- Instantiierung:
  Die instantiierte Klasse (mit aktuellem Parameter) zeigt auf die parametrisierte Klasse (mit formalem Parameter).
- Metaklasse (Klasse einer Klasse):
  Metaklassen werden verwendet, um Klasseninstanzvariablen und -

operationen zur Verfügung zu stellen, können also selbst keine Instanzen besitzen. Metaklassen können aber von anderen Klassen erben.

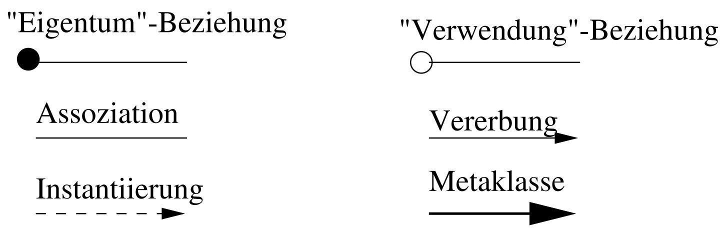


Abbildung 2.16: Klassenbeziehungen

Kennzeichnungen der Beziehungen Beziehungen können beschrieben werden, um die Zusammenarbeit der Klassen deutlich zu machen:

- Rolle:
  Sie gibt den Zweck oder den Umfang der Beziehung einer Klasse zu einer anderen an. (z.B. Fahrer, Mitarbeiter, Planer, ...)
- Schlüssel:
  Ein Attribut, dessen Wert ein einzelnes Zielobjekt eindeutig identifiziert. (z.B. Fahrzeugnummer, Mitarbeitername, Kunden-ID, ...)
- Einschränkung:
  Durch die Einschränkung wird eine sematische Bedingung einer Klasse oder einer Beziehung beschrieben, die erfüllt sein muß, solange sich das System in einem stabilen Zustand befinden soll (z.B. TÜV am Fahrzeug darf nicht abgelaufen sein).
- Kardinalität:
  Sie gibt an, wieviele Instanzen der einen Klasse in der Instanz der jeweils anderen Klasse vorhanden sein dürfen.
- Attributklasse:
  Die Attributklassen beschreiben die Eigenschaften der Assoziationen. D.h. jede Assoziation muß eine Instanz der Attributklasse sein.

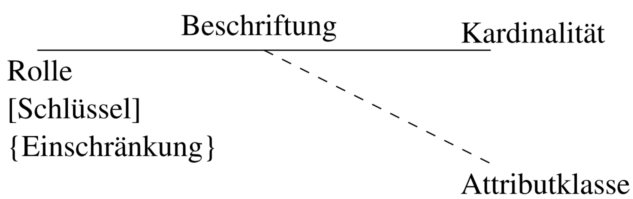


Abbildung 2.17: Kennzeichnungen der Beziehungen

Kennzeichnungen für das Enthaltensein Das (physikalische) Enthaltensein wird dargestellt als Anmerkung am Ende einer „Eigentum“-Beziehung. Es können zwei Arten unterschieden werden:

- über den Wert:
  Gibt das (physikalische) Beinhalten des Wertes des Bestandteils an.
- über eine Referenz:
  Gibt das (physikalische) Beinhalten über einen Zeiger oder eine Referenz auf den Bestandteil an.

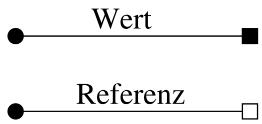


Abbildung 2.18: Kennzeichnungen für das Enthaltensein

Eigenschaften In einigen objektorientierten Sprachen sind die Beziehungen von ihrer Sematik her gesehen fundamental verschieden, so daß spezielle Symbole eingeführt werden müssen:

- abstrakte Klasse:
  Von abstrakten Klassen können keine Instanzen erzeugt werden, sie dienen in der Regel dazu, ein Klassengerüst zur Verfügung zu stellen. D.h. die abstrakte Klasse gibt die Schnittstellen der Methoden vor, deren Implementierung in der Verantwortung der Unterklasse liegt.
- static:
  Die Bezeichnung eines Klassenelementobjekts oder einer -funktion (statische Klassen finden z.B. in der Programmiersparche C++ Verwendung; Smalltalk oder CLOS verwenden dagegen Metaklassen).
- virtual:
  Diese Eigenschaft wird benutzt, um eine gemeinsam verwendete Basisklasse oder eine polymorphe Operation zu bezeichnen.
- friend:
  Eine friend-Klasse erhält von einer anderen Klasse Rechte, um auf deren nicht- public-Teile zugreifen zu dürfen.

Exportsteuerung Die meisten OO-Sprachen bieten eine klare Trennung zwischen Schnittstelle und Implementierung einer Klasse. Mit Hilfe der Exportsteuerung kann der Zugriff auf die Schnittstelle genau spezifiziert werden.

- public:
  Die Schnittstelle ist für alle Clients erreichbar.
- protected:
  Nur Unterklassen, Friend und die Klasse selbst dürfen auf die Schnittstelle zugreifen.

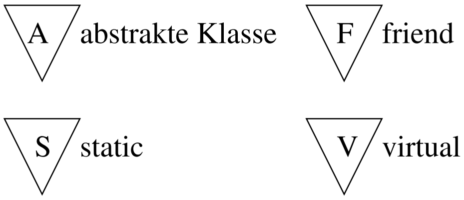


Abbildung 2.19: Eigenschaften

- private:
  Auf die Schnittstelle dürfen nur die Klasse selbst oder ein Friend zugreifen.
- implementation:
  Nur die Implementierung hat Zugriff (siehe abstrakte Klassen).

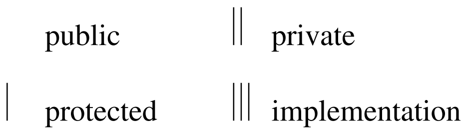


Abbildung 2.20: Exportsteuerung

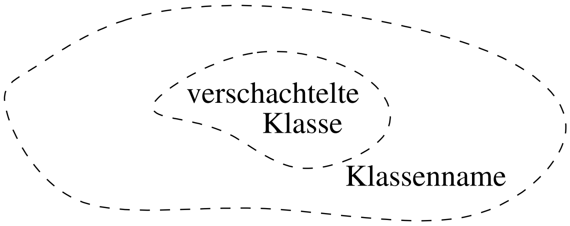


Abbildung 2.21: geschachtelte Klassen

Geschachtelte Klassen Klassen können ineinander geschachtelt werden. Dies soll bei der Kontrolle über den Namensraum helfen, da Instanzen von eingeschlossenen Klassen nur in der umgebenden Klasse existieren können.

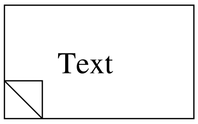


Abbildung 2.22: Notizen

Notizen Notizen können entweder an Klassenkategorien oder an Klassen direkt angehängt werden und dienen zur näheren Beschreibung bzw. Orientierung.

Klassenspezifikation Die graphische Sicht der Klassen und deren Zusammenarbeit alleine helfen nicht bei der Systementwicklung. Klassen und später deren Operationen müssen näher spezifiziert werden. Wie könnte eine solche Klassenspezifikation nun aussehen? Hier ein Beispiel (die ersten sechs Einträge sollten unbedingt enthalten sein):

```
Name:                 Bezeichner
Definition:           Text
Verantwortlichkeiten: Text
Attribute:            Liste der Attribute
Operationen:          Liste der Operationen
Einschränkungen:      Liste der Einschränkungen
Status Maschine:      Referenz auf Statusmaschine (s.u.)
Exportsteuerung:      public | protected | private |
                      implementation
Kardinalität:         Ausdruck
Parameter:            Liste formaler und generischer Parameter
Persistenz:           transient | persistent
Nebenläufigkeit:      sequentiell | überwacht | synchron | aktiv
Speicherkomplexität:  Ausdruck
```

Operationsspezifikation Eine Klasse beinhaltet verschiedene Operationen, die jede einzeln spezifiziert werden muß. Das folgende Beispiel zeigt, wie eine solche Spezifikation aussehen könnte (die ersten vier Einträge sollten wieder unbedingt enthalten sein):

```
Name:                Bezeichner
Definition:          Text
Rückgabeklasse:      Referenz auf die Klasse
Argumente:           Liste formaler Argumente
Exportsteuerung:     public | protected | private |
                     implementation
Protokoll:           Text
Vorbedingungen:      Text | Referenz auf Quellcode |
                     Referenz auf Objektdiagramm
Semantik:            Text | Referenz auf Quellcode |
                     Referenz auf Objektdiagramm
```

```
Nachbedingungen:     Text | Referenz auf Quellcode |
                     Referenz auf Objektdiagramm
Ausnahmen:           Liste der Ausnahmen
Nebenläufigkeit:     sequentiell | überwacht | synchron
Speicherkomplexität: Ausdruck
Zeitkomplexität:     Ausdruck
```

2.2.5.2 Zustandsdiagramm

Das Zustandsdiagramm zeigt die Existenz von Klassen und, viel wichtiger, wie sie ihr Verhalten ändern. Mit Hilfe von Zustandsdiagrammen läßt sich das dynamische Verhalten von einzelnen Klassen modellieren.

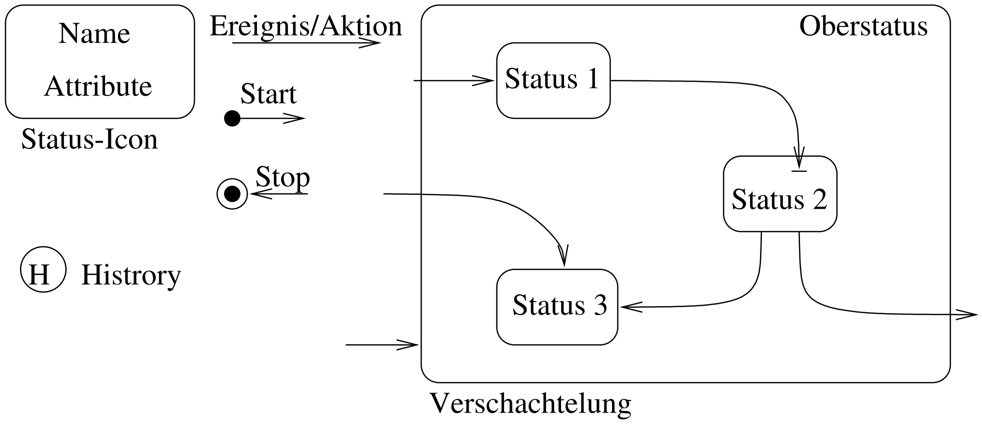


Abbildung 2.23: Elemente des Zustandsdiagramms

Befindet sich ein Querstrich an der Pfeilspitze einer Aktion oder eines Ereignisses, handelt es sich bei dem Status, auf den der Pfeil zeigt, um einen verschachtelten Status, d.h. wird eine Hierarchieebene tiefer gegangen, so öffnet sich der Status zu einem neuen Oberstatus. Das History-Icon gibt an, daß der Vorgängerzustand gespeichert wird und nach Abarbeitung der aktuellen Zustandsfolge in den Vorgängerzustand zurückgekehrt wird.

Das Beispiel zum Zustandsdiagramm in Abbildung 2.24 zeigt exemplarisch an der Klasse Zivi, in welchen Zuständen sich diese Klasse befinden kann, bzw. wie ein Zivi modelliert werden kann.

2.2.5.3 Objektdiagramme

Das Objektdiagramm stellt die Existenz von Objekten und deren Beziehungen bzw. deren Interaktionen miteinander dar. Mit Hilfe von Objektdiagrammen können Szenarien durchgespielt werden, um zu testen, ob das System das gewünschte Verhalten modelliert.

Im Beispiel in Abbildung 2.26 wird mit Hilfe eines Objektdiagramms modelliert, wie ein Zivi einen Schüler zur Schule bringt.

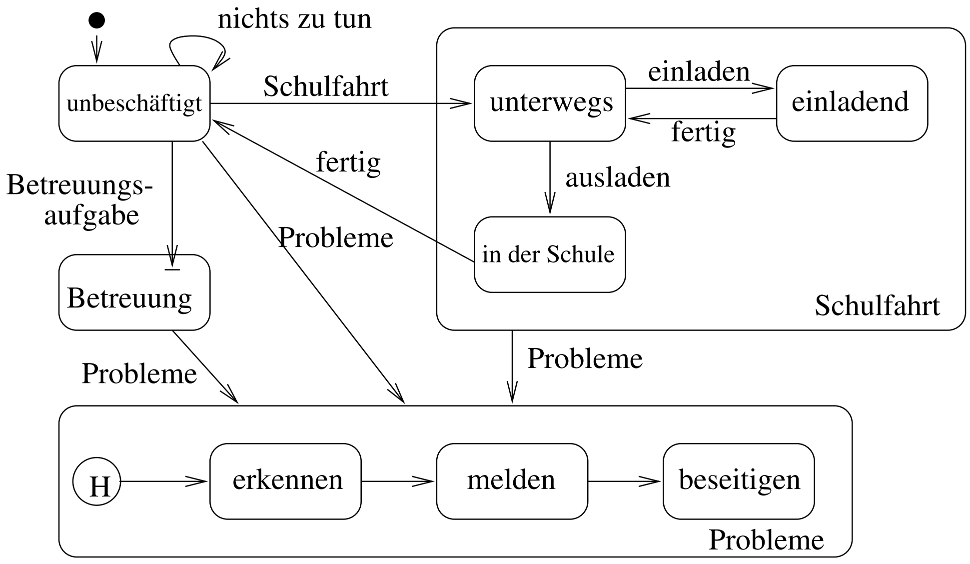


Abbildung 2.24: Beispiel für Zustandsdiagramme für die Klasse Zivis

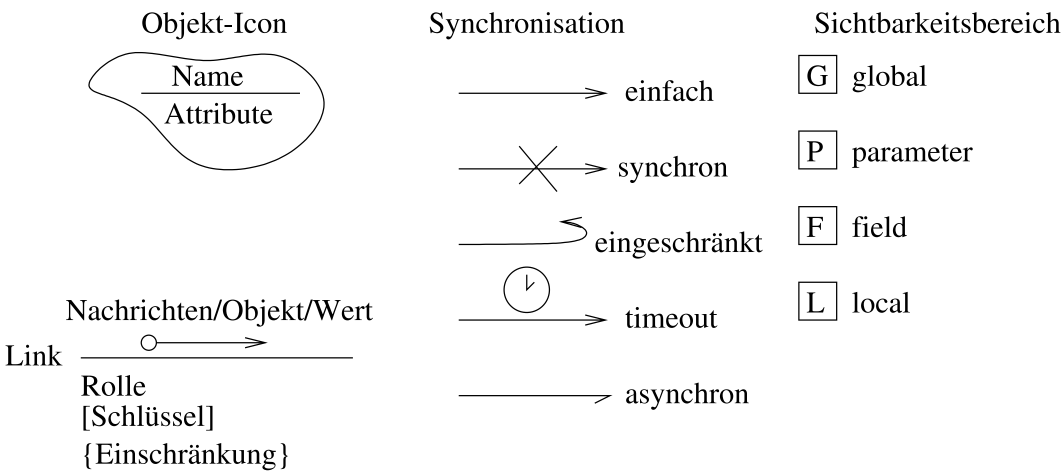


Abbildung 2.25: Elemente des Objektdiagramms

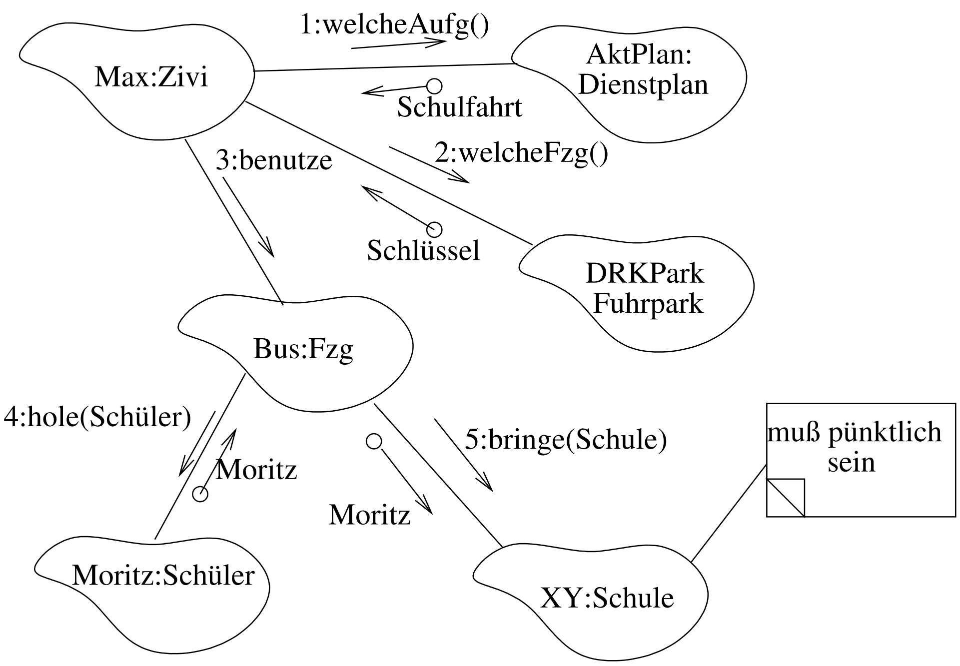


Abbildung 2.26: Beispiel für ein Objektdiagramm

2.2.5.4 Interaktionsdiagramm

Das Interaktionsdiagramm ist eine andere Darstellungsform des Objektdiagramms. Es stellt die Interaktionen zwischen Objekten in zeitlichem Ablauf dar.

Das Beispiel in Abbildung 2.27 ist direkt aus dem obigen Beispiel zum Objektdiagramm abgeleitet.

2.2.5.5 Moduldiagramm

Anders als die vorhergehenden Diagramme bietet das Moduldiagramm keine logische Sicht auf das zu entwickelnde System, sondern eine physikalische. Das Moduldiagramm modelliert die Aufteilung in einzelne Module, d.h. die Hierarchiestufen der Implementierung. Innerhalb der Module befinden sich die Klassen, die durch die Module in funktionale Einheiten gegliedert sind.

Mit Hilfe der Untersysteme lassen sich einzelne Modulbäume hierarchisch anordnen. Ob eine strikte Trennung zwischen Modulspezifikation und Modulrumpf besteht, ist abhängig von der Programmiersprache.

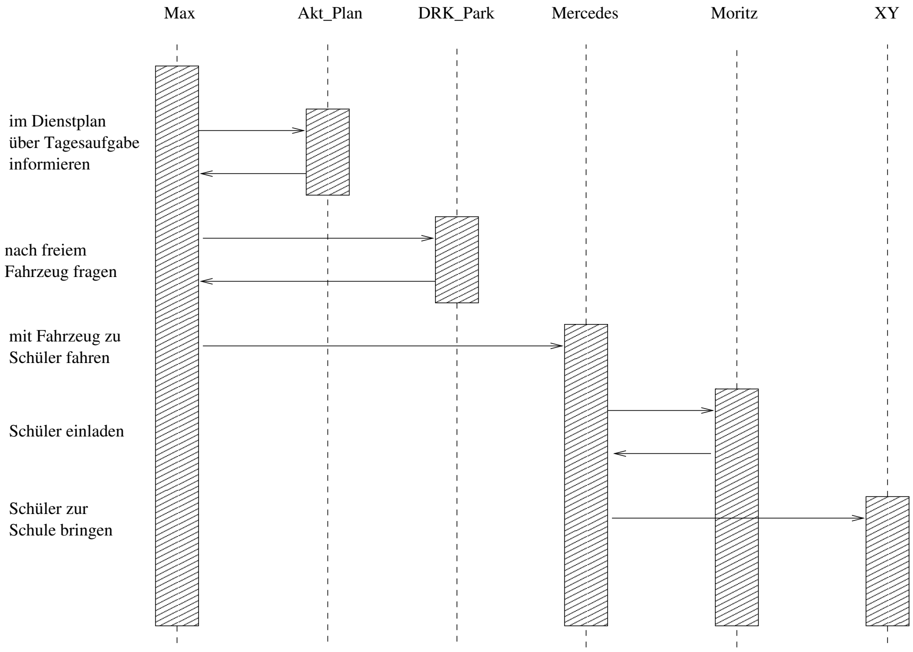


Abbildung 2.27: Beispiel für ein Interaktionsdiagramm

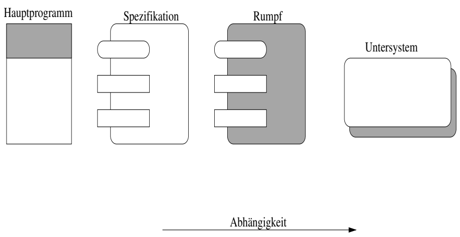


Abbildung 2.28: Elemente des Moduldiagramms

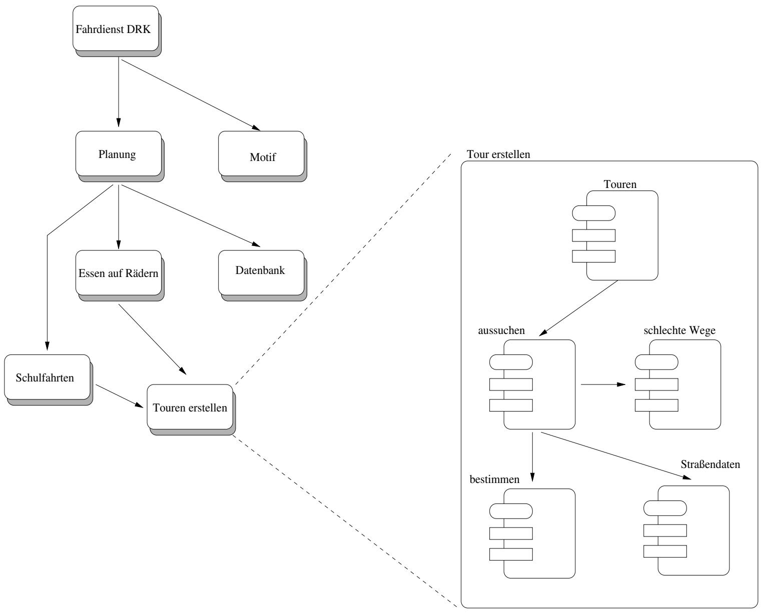


Abbildung 2.29: Beispiel für ein Moduldiagramm

2.2.5.6 Prozeßdiagramm

Das Prozeßdiagramm visualisiert die Aufteilung der Systemarchitektur auf einzelne Hardwarekomponenten. Folglich ist die Erstellung eines Prozeßdiagrammes nur dann erforderlich, wenn bei der Systementwicklung auf verteilte Systeme bzw. Mehrprozessorsysteme zugegriffen werden soll.

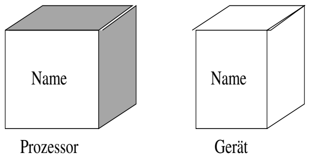


Abbildung 2.30: Elemente des Prozeßdiagramms

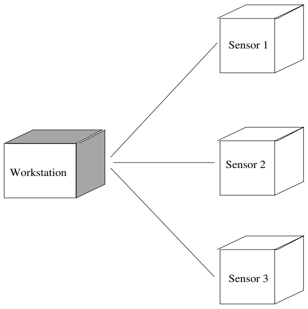


Abbildung 2.31: Beispiel für ein Prozeßdiagramm

2.2.6 Prozeß

2.2.6.1 Was macht gute Software aus?

Wenn man anfängt, ein Software-System zu entwickeln, sollte man sich Gedanken machen, was gute Software ausmacht:

- Abstraktionen sollten wohldefiniert und in sich geschlossen sein
- klar definierte Schnittstellen, da andere Mitglieder des Entwicklungsteams u.U. bei der Entwicklung ihrer Klassen auf diese Schnittstellen zugreifen
- klare Trennung zwischen Schnittstelle und Implementierung, dies macht spätere Änderungen einfacher bzw. überhaupt möglich. Stellt der Entwickler während der Codierung fest, daß es für ein bestimmtes Problem einen besseren Algorithmus gibt, so kann er diesen implementieren, ohne die Schnittstelle ändern zu müssen. Eine Änderung der Schnittstelle könnte dagegen auch andere Klassen beeinträchtigen.
- eine einfache Architektur, die das Erweitern und Ändern des Systems vereinfacht

Desweiteren sollten vor der Systementwicklung von vorneherein schon einige Architekturentscheidungen getroffen werden. Hierbei unterscheidet man:

- Strategische Entscheidungen:
  - haben umfassende Auswirkungen auf die Architektur und die Organisation
  - stellen Mechanismen und Paradigmen für die Fehlerbehandlung, die Schnittstellen, die Speicherverwaltung usw. zur Verfügung
- Taktische Entscheidungen:
  haben nur lokalen Einfluß auf einzelne Systemkomponenten:
  - Details der Schnittstelle
  - Wahl eines speziellen Algorithmus
  - Signatur einer Methode

Auch ist es wichtig, den Entwicklungsprozeß zu steuern, um ein systematisches Vorgehen zu erhalten. Ein vollständig rationaler Entwicklungsprozeß ist nicht möglich. Dieser läßt sich jedoch mit Hilfe des Makro- und des Mikroentwicklungsprozesses simulieren.

2.2.6.2 Mikroentwicklungsprozeß

Der Mikroentwicklungsprozeß wird durch eine Folge von Aktivitäten und Szenarien des Makroentwicklungsprozesses gesteuert. Er beinhaltet die täglichen Arbeiten der Entwickler.

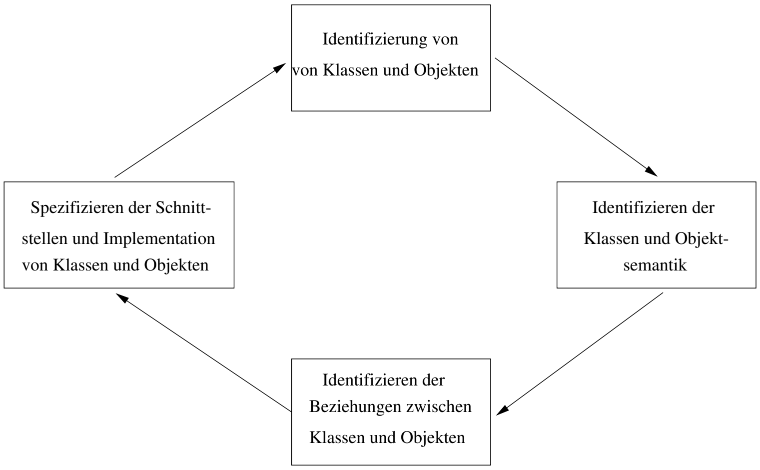


Abbildung 2.32: Mikroentwicklungsprozeß

Schritt 1 Die Identifizierung von Klassen und Objekten auf einem bestimmten Abstraktionsniveau. Die primären Aktionen dieses Schrittes sind die Entdeckung und Erfindung der benötigten Klassen.

Schritt 2 Identifizierung der Semantik der in Schritt 1 gefundenen Klassen und Objekte, d.h. herausstellen, für was die Klassen und Objekte zuständig sind und wie sie diese Zuständigkeiten realisieren (= Drehbuchkonzept). In diesem Schritt wird jede Klasse und jedes Objekt isoliert für sich betrachtet.

Schritt 3 Identifizierung der Beziehungen zwischen den oben gefundenen Klassen und Objekten. In diesem Schritt sind die Assoziationen zwischen den Klassen und Objekten, bzw. eine Verfeinerung schon bestehender Assoziationen zu spezifizieren.

Schritt 4 Implementierung der Klassen und Objekte. Die primäre Aktivität in diesem Schritt ist die Auswahl der zu den Klassen und Objekten bzw. zu ihrem spezifizierten Verhalten passenden Datenstrukturen und Algorithmen.

Der Mikroentwicklungsprozeß ist ein Kreislauf, denn während der Abarbeitung der einzelnen Schritte können neue Klassen und Objekte benötigt werden, für die das gesamte Schema wieder durchlaufen werden muß.

2.2.6.3 Makroentwicklungsprozeß

Der Makroentwicklungsprozeß dient als steuerndes Gerüst für den Mikroentwicklungsprozeß. Er definiert den Entwicklungszyklus des Produktes. Aus ihm können eine Vielzahl meßbarer Ergebnisse und Aktivitäten für die Risikoverwaltung hervorgehen.

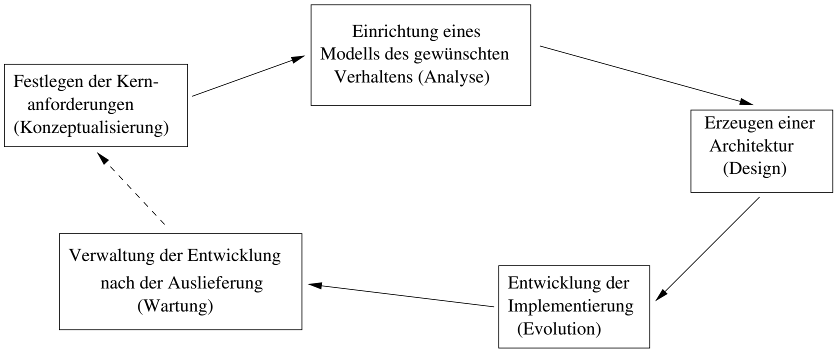


Abbildung 2.33: Makroentwicklungsprozeß

Schritt 1 Festlegung der Grundanforderungen. Diese Grundanforderungen können gleichzeitig dazu benutzt werden, um die Gültigkeit des Konzeptes zu beweisen. In diesem Schritt sollte mit verschiedenen Lösungsansätzen experimentiert werden.

Schritt 2 Sind die Grundanforderungen an das zu erzeugende System festgelegt, muß mit diesen Anforderungen ein Modell erstellt werden. Dieses kann z.B. mit Hilfe der Bereichsanalyse oder über eine Szenarioplanung erfolgen.

Schritt 3 Schaffen einer Architektur für das System. Festlegen der Vorgehensweisen, d.h. architektonische Planung, taktische Entscheidungen, Planung der einzelnen Versionen (auch im Hinblick auf die Weiterentwicklung).

Schritt 4 Mit den ersten drei Schritten sollte der schwierigste Teil der Entwicklung erledigt sein. Während der Implementierungsphase geht es „nur noch“ darum, das Design des Systems in Code zu fassen.

Schritt 5 In dieser Phase sollten Fehler entfernt werden, die nach der Auslieferung aufgetreten sind. Mit Weiterentwicklung hat dieser Schritt nichts zu tun. Wird beschlossen, das System weiterzuentwickeln, muß wieder bei Schritt 1 angefangen werden.

2.3 Flüsse in Netzwerken (Frank Wagner)

2.3.1 Einleitung

Überall begegnen uns Netzwerke: Das Straßen- und das Schienennetz, Flüsse, Hochspannungsleitungen, Pipelines, Telekommunikationsnetze und viele andere mehr. Oft interessieren den Betreiber oder den Benutzer in irgend einer Weise optimale Methoden, sein Gut – Autos, LKWs, Züge, Schiffe, Waren, Öl, Elektronen, Nachrichten – über dieses Netzwerk zu transportieren. Abstrakte Modelle für einige dabei auftretende Probleme sind die Fluß-Probleme [AMO93, Tar91]. Ihre Anwendung ist aber nicht auf Probleme beschränkt, bei denen wirklich ein Netzwerk vorhanden ist. Auch Aufgaben aus Bereichen, wie z.B. Projekt- und Produktionsplanung lassen sich mit diesen Ansätzen bearbeiten. Im folgenden werden einige dieser Modelle, hauptsächlich das Minimum Cost Flow Problem (MCFP) und das Maximum Flow Problem (MFP), vorgestellt. Zum Abschluß werden einige Möglichkeiten gezeigt, das MCFP in unserem Problem, den Mobilen Diensten des DRK, anzuwenden.

2.3.2 Definitionen

ungerichteter Graph (undirected graph): Ein ungerichteter Graph $G = (N, A)$ besteht aus einer endlichen, nichtleeren Menge N, den *Knoten* (Ecken, nodes, vertices), und einer Menge $A \subseteq \{\{i,j\} \mid i,j \in N\}$ ungeordneter Paare von Knoten, den *Kanten* (arcs, edges). Abbildung 2.34(a) zeigt eine graphische Repräsentation des Graphen $G_1 = (N, A)$ mit $N = \{1,2,3,4,5,6\}$ und $A = \{\{1,2\},\{1,3\},\{1,5\},\{2,4\},\{3,4\},\{3,5\},\{4,6\},\{5,6\}\}$.

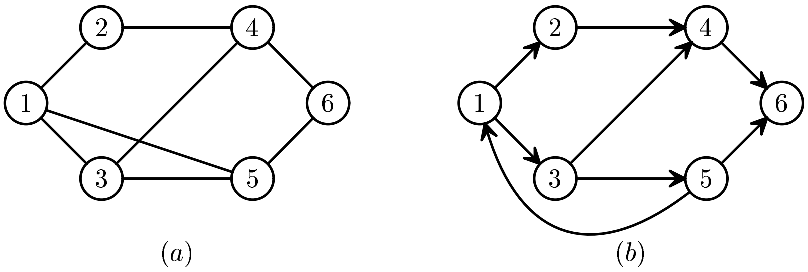


Abbildung 2.34: Repräsentationen der Graphen G_1 (a) und G_2 (b)

gerichteter Graph (directed graph, digraph): Ein gerichteter Graph $G = (N, A)$ besteht wie der ungerichtete Graph aus einer endlichen, nichtleeren Menge N von Knoten, die Kantenmenge $A \subseteq N \times N$ besteht jedoch aus geordneten Paare (i, j) von Knoten. Knoten i nennt man *Anfangsknoten* (head), Knoten j ist der *Endknoten* (tail). Abbildung 2.34(b) zeigt eine gerichtete Variante des Graphen aus Abbildung 2.34(a) G_2 mit $A = \{(1,2),(1,3),(2,4),(3,4),(3,5),(4,6),(5,1),(5,6)\}$.

Mehrfachkanten und Schleifen (multiarcs and loops): *Mehrfachkanten* sind zwei oder mehr Kanten, die die gleichen Anfangs- und Endknoten haben.

Eine *Schleife* ist eine Kante, bei der der Anfangsknoten auch der Endknoten ist. Abbildung 2.35 zeigt Beispiele dieser beiden Kantentypen.

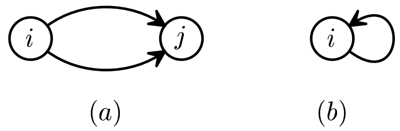


Abbildung 2.35: (a) Mehrfachkante und (b) Schleife

Weg (path): Ein Weg $w = (i_1, \ldots, i_r)$ der Länge r in einem gerichteten Graphen $G = (N, A)$ ist eine schleifenfreie Folge von Knoten. Die Richtung, in der die Kanten durchlaufen werden, spielt keine Rolle.

$$w = (i_1, i_2, \ldots, i_r) \in N^r \tag{2.3.1}$$
$$\forall 1 \leq k \leq r-1 : ((i_k, i_{k+1}) \in A) \vee ((i_{k+1}, i_k) \in A) \tag{2.3.2}$$
$$\forall x, y \in \{1, \ldots, r\} : x \neq y \Rightarrow i_x \neq i_y \tag{2.3.3}$$

$2-4-3-1$ im Graphen aus Abbildung 2.34(b) ist ein Weg.

gerichteter Weg (directed path): Ein gerichteter Weg ist eine Weg, bei dem die Kanten nur in Vorwärtsrichtung benutzt werden. Bedingung (2.3.2) wird also ersetzt durch

$$\forall 1 \leq k \leq r-1 : ((i_k, i_{k+1}) \in A) \tag{2.3.4}$$

Jeder Knoten außer dem Anfangsknoten hat einen eindeutigen Vorgänger. $3-5-1-2-4-6$ ist der längste gerichtete Weg im Graphen aus Abbildung 2.34(b).

Zyklus (cycle): Ein Zyklus ist ein Weg $i_1 - i_2 - \ldots - i_r$ zusammen mit der Kante (i_r, i_1) oder der Kante (i_1, i_r). Ein Zyklus in G_2 ist zum Beispiel $4-6-5-3-4$.

gerichteter Zyklus (directed cycle): Ein gerichteter Zyklus ist ein gerichteter Weg $i_1 - i_2 - \ldots - i_r$ zusammen mit der Kante (i_r, i_1). $1-3-5-1$ ist ein gerichteter Zyklus in G_2. Ein Graph ist *azyklisch*, wenn er keine gerichteten Zyklen enthält.

Schnitt (cut): Ein Schnitt ist eine Partitionierung der Knotenmenge N in zwei Teile S und $\bar{S} = N - S$. Jeder Schnitt legt die Menge der Kanten fest, die die beiden Partitionen verbinden:

$$(S, \bar{S}) = \{(i,j) \in A \mid i \in S \wedge j \in \bar{S}\} \tag{2.3.5}$$
$$[S, \bar{S}] = (S, \bar{S}) \cup (\bar{S}, S) \tag{2.3.6}$$

s-t-Schnitt (s-t cut): Ein s-t-Schnitt ist ein beliebiger Schnitt mit der Eigenschaft, daß $s \in S$ und $t \in \bar{S}$ gilt.

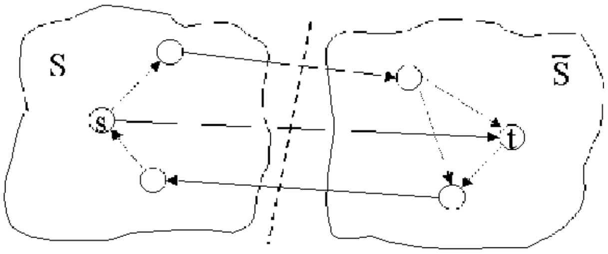


Abbildung 2.36: Ein s-t-Schnitt

Bipartiter Graph (bipartite graph): Ein Graph $G = (N_1 \cup N_2, A)$ ist ein bipartiter Graph, wenn alle Kanten einen Knoten aus der einen Menge mit einem aus der anderen Menge verbinden, d.h.

$$\forall (i,j) \in A : (i \in N_1 \wedge j \in N_2) \vee (i \in N_2 \wedge j \in N_1) \tag{2.3.7}$$

2.3.3 Das Minimum Cost Flow Problem

2.3.3.1 Das Problem

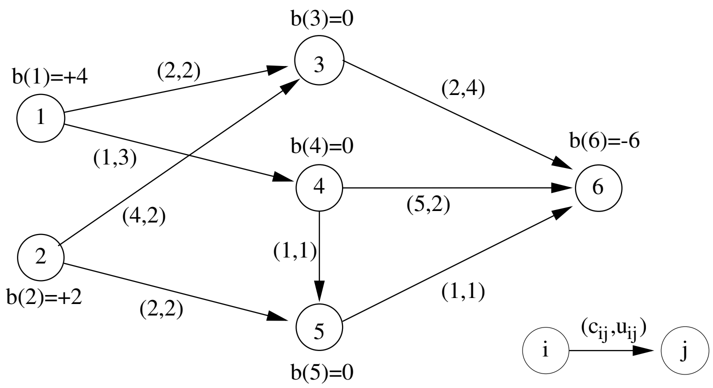


Abbildung 2.37: Ein Fluß-Netzwerk

Das Minimum Cost Flow Problem (MCFP) ist ein Modell für Anwendungen, bei denen eine Möglichkeit gesucht wird, ein Gut mit minimalen Kosten von Orten mit einem Überschuß, über eventuell vorhandene Durchgangsorte, zu Orten mit einem Bedarf für dieses Gut zu bringen.

Gegeben ist ein Netzwerk $\mathcal{N} = (N, A, c, u, b)$ mit

$G = (N, A)$	gerichteter Graph
$c : A \to \mathbb{N}$	Kostenfaktor
$u : A \to \mathbb{N}$	Kapazitätsgrenze
$b : N \to \mathbb{Z}$	Angebot-Bedarf.

Gesucht ist ein Fluß $x : A \to \mathbb{R}^+$, bei dem die Gesamtkosten

$$z(x) = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.3.8}$$

minimal sind und die Randbedingungen

$$\forall i \in N : \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = b(i) \tag{2.3.9}$$

$$\forall (i,j) \in A : \quad 0 \le x_{ij} \le u_{ij} \tag{2.3.10}$$

eingehalten werden. Anstatt $x((i,j))$ schreiben wir x_{ij}. Die Gleichgewichtsbedingung (mass balance constraint) (2.3.9) stellt sicher, daß an keinem Knoten Fluß verloren geht. Gleichung (2.3.10) sorgt dafür, daß die Kapazitätsschranken eingehalten werden (flow bound constraint).

2.3.3.2 Erweiterungen

In diesem Abschnitt werden einige Umformungen gezeigt, wie scheinbare Erweiterungen des MCFP behandelt werden können.

Positive rationale Daten (Kosten, Kapazitätsgrenzen, Angebot, Bedarf) können behandelt werden, indem man den Hauptnenner sucht und alle Werte damit multipliziert. Nun kann man mit ganzzahligen Daten rechnen und muß am Ende nur den Hauptnenner wieder herausrechnen.

Ungerichtete Graphen werden in einen gerichteten Graphen überführt, indem die Kanten durch Hin- und Rückkante mit gleichen Kosten und Kapazitätsgrenzen ersetzt werden. In einer minimalen Lösung wird nur Fluß über eine der beiden Kanten fließen.

Mehrfachkanten sind ein rein notationelles Problem, da man einzelne Kanten nicht mehr durch Quell- und Zielort identifizieren kann. Durch Definition der Kanten als $A \subseteq N \times N \times \mathbb{N}$, wobei die dritte Komponente eine fortlaufende Nummer ist, kann dieses Problem behoben werden. Eine andere Möglichkeit ist es, Hilfsknoten in die Mehrfachkanten einzufügen.

Ganzzahlige Kostenfaktoren $c : A \to \mathbb{Z}$, also auch negative Kostenfaktoren, können durch die lokale Transformation aus Abbildung 2.38 mit dem ursprünglichen Modell behandelt werden. Der Fluß über die Kante (i,j) mit $c_{ij} < 0$ wird auf das Maximum, die Kapazitätsgrenze u_{ij}, gesetzt, womit die Kosten minimal sind. Es kann nun aber passieren, daß so viel gar nicht über die Kante fließen kann, weil z.B an Knoten i weniger Fluß zur Verfügung steht. Dieser zu viel geschickte Fluß wird über die neu eingefügte Kante (j,i) zu nun positiven Kosten wieder zurückgeschickt. Der durch diese Umformung an den Gesamtkosten $z(x)$ entstandene konstante Fehler von $c_{ij}u_{ij}$ muß am Ende wieder ausgeglichen werden.

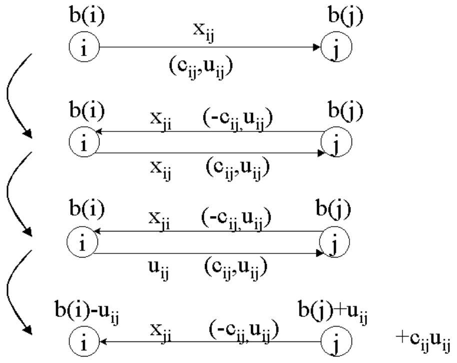


Abbildung 2.38: Beseitigung einer Kante (i, j) mit negativen Kosten $c_{ij} < 0$

Untergrenzen $l_{ij} \leq x_{ij}$ mit $l : A \to \mathcal{NAT}$ für den Fluß über die Kanten können ebenfalls durch eine lokale Transformation wieder in das MCFP überführt werden. Die Schritte sind in Abbildung 2.39 zu sehen. Die Kante (i, j) mit $l_{ij} \neq 0$ wird aufgespalten in eine Kante mit $u_{ij1} = l_{ij}$, die den Fluß bis zur Untergrenze aufnimmt und die Untergrenze l_{ij} bekommt, und eine zweite Kante mit $u_{ij2} = u_{ij} - l_{ij}$ für den darüber hinausgehenden Fluß, die die Untergrenze 0 hat. Der Fluß über die erste Kante muß nun genau l_{ij} sein, ist also von keiner Variablen mehr abhängig. Die Kante kann deshalb aus dem Graphen entfernt werden. Den dadurch entstandenen Fehler an den Gesamtkosten von $c_{ij}l_{ij}$ muß man sich wieder merken.

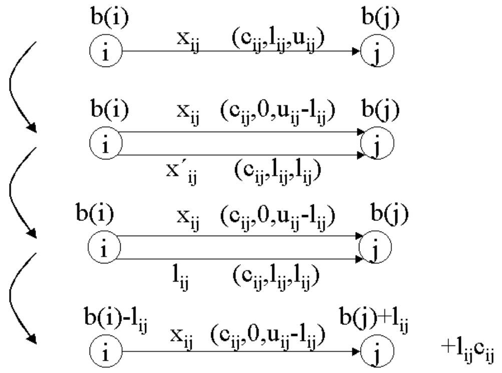


Abbildung 2.39: Behandlung der Untergrenze u_{ij} der Kante (i, j)

Angebot $\neq$ Bedarf ist im Modell nicht erlaubt, da sonst (2.3.9) nicht erfüllt ist. Durch Hinzufügen eines Hilfsknotens, der den überflüssigen bzw. fehlenden Fluß ausgleicht und Kanten von allen Knoten mit $b(i) > 0$ bzw. zu allen Knoten mit $b(i) < 0$ hat, kann diese Differenz ausgeglichen werden.

2.3.3.3 Ein Algorithmus

```
Eingabe: Netzwerk G=(N,A,c,u,b)
Ausgabe: minimaler Fluss x
algorithm succesive_shortest_path;
  v := 0; forall {i∈N|b(i)>0} do v += b(i); od;
  N := N ∪ {s,t}; b(s) := v; b(t) := -v;
  forall {i∈N|b(i)>0} do
    A := A ∪ {(s,i)};
    u(s,i) := b(i); c(s,i) := 0;
    b(i) := 0;
  od;
  forall {i∈N|b(i)<0} do
    A := A ∪ {(i,t)};
    u(i,t) := -b(i); c(i,t) := 0;
    b(i) := 0;
  od;
  x := 0;          -- Fluss
  while v>0 do
    W := kürzester zunehmender Weg von s nach t;
    if (W=∅) then exit("Keine Lösung");
    δ := min({v} ∪
          {u(i,j) − x(i,j) | (i,j) ∈ W ∧ (i,j) ∈ A} ∪
          {x(j,i) | (i,j) ∈ W ∧ (j,i) ∈ A});
    erhöhe den Fluss auf dem Weg W um δ;
    v -= δ;
  od;
end;
```

Der Algorithmus formt zunächst das Netzwerk um. Eine neue Quelle s und eine neue Senke t werden hinzugefügt. Das Angebot $b(i)$ an einem Knoten i mit $b(i) > 0$ wird nun durch eine Kante von der Quelle s nach i mit Kapazität $b(i)$ und Kostenfaktor 0 simuliert. Entsprechend wird mit den Senken im ursprünglichen Netzwerk verfahren. Jetzt gibt es nur noch eine Quelle und eine Senke, am gesuchten Fluß hat sich aber nichts geändert. Die eigentliche Suche geschieht dann in der while-Schleife. In jedem Durchlauf wird ein kürzester zunehmender Weg von s nach t gesucht. Ein zunehmender Weg ist ein Weg von s nach t, auf dem der Fluß erhöht werden kann. Die Kanten auf diesem Weg haben also entweder noch freie Kapazitäten ($x_{ij} < u_{ij}$) oder in entgegengesetzter Richtung herrscht ein Fluß ($x_{ji} > 0$). Auf diesem Weg wird der Fluß so stark wie möglich erhöht. Da in jedem Durchlauf v um mindestens eins erniedrigt wird, terminiert der Algorithmus. Da die Gesamtkosten $z(x)$ nie geringer werden, wird auch ein minimaler Fluß gefunden.

Abbildung 2.40 zeigt einen möglichen Ablauf des Algorithmus bei dem Netzwerk aus Abbildung 2.37 ausführt. Bei mehreren gleich teuren Möglichkeiten wurde zufällig eine davon gewählt.

Theorem 2.3.1 (Integrality) Wenn es eine Lösung des MCFP gibt, dann gibt es auch eine ganzzahlige Lösung.

Schritt	w	δ	$\Delta z(x)$
1	1-4-5-6	1	3
2	1-3-6	2	8
3	2-3-6	2	12
4	2-5-4-6	1	6

Abbildung 2.40: Schritte des Algorithmus

Beweis 2.3.1 Dies sieht man per Induktion über die Durchläufe des Algorithmus. Der Algorithmus beginnt mit einem ganzzahligen Fluß (0). Unter der Annahme, daß das Angebot v und die Kapazitätsgrenzen u alle ganzzahlig sind, erhöht der Algorithmus den Fluß um ein ganzzahliges δ, weshalb der Fluß auch nachher noch ganzzahlig ist. □

2.3.4 Spezialfälle

Es gibt viele Anwendungen für das Minimum Cost Flow Problem. Einige davon sind Kürzeste-Wege-Probleme (sowohl Single Target als auch Multiple Target), das Maximum-Fluß-Problem, das Zirkulations-Problem, das Transport-Problem und das Matching-Problem. In diesem Abschnitt werden drei davon näher vorgestellt.

2.3.4.1 Single Target Shortest Path

Bei diesem Problem wird ein kürzester Weg von einem Startort zu einem Zielort gesucht. Gegeben sind ein Netzwerk $\mathcal{N} = (N, A, c)$ und zwei Knoten s, t mit

$G = (N, A)$	gerichteter Graph
$c : A \to \mathbb{N}$	Kosten bzw. Längen der Kanten
$s, t \in N$	Start- und Zielort

Gesucht ist ein gerichteter Weg w von s nach t mit minimaler Länge

$$l(w) = \sum_{(i,j)\in w} c_{ij}$$

Ein beliebiger Algorithmus, der das MCFP löst, kann auch einen kürzesten Weg zwischen zwei Orten s und t in einem Netzwerk $\mathcal{N} = (N, A, c)$ suchen. Dazu definiert man das Netzwerk $\mathcal{N}' = (N, A, c, u, b)$ mit:

$$\forall (i,j) \in A : u_{ij} = 1 \tag{2.3.11}$$

$$\forall i \in N : b(i) = \begin{cases} +1, \text{ falls } i = s \\ -1, \text{ falls } i = t \\ 0 \text{ sonst} \end{cases} \tag{2.3.12}$$

Der Algorithmus von oben wird eine Flußeinheit auf einem kürzesten zunehmenden Weg, was bei einem Fluß von 0 einem kürzesten Weg entspricht, vom Startort s zum Zielort t schicken. Danach wird v Null und der Algorithmus endet nach einem Durchlauf.

2.3.4.2 Maximum Flow Problem

Beim Maximum Flow Problem (MFP) wird eine Möglichkeit gesucht, so viel (Dauer-)Fluß wie möglich von einer Quelle s zu einer Senke t zu schicken. Kosten spielen hierbei keine Rolle.

Gegeben sind ein Netzwerk $\mathcal{N} = (N, A, u)$ und zwei Knoten s, t mit

$G = (N, A)$	gerichteter Graph
$\forall (i, j) \in A : (j, i) \in A$	
$u : A \to \mathbb{N}$	Kapazität
$s, t \in N$	Quelle/Ziel

Gesucht ist ein Fluß $x : A \to \mathbb{R}^+$ mit maximalem Betrag v, für den gilt:

$$\forall i \in N : \sum_{\{j \mid (i,j) \in A\}} x_{ij} - \sum_{\{j \mid (j,i) \in A\}} x_{ji} = \begin{cases} v & \text{falls } i = s \\ -v & \text{falls } i = t \\ 0 & \text{sonst} \end{cases} \qquad (2.3.13)$$

$$\forall (i,j) \in A : \quad 0 \leq x_{ij} \leq u_{ij} \qquad (2.3.14)$$

Dabei darf kein gerichteter Weg mit unendlicher Kapazität von s nach t existieren, da sonst das Problem keine endliche Lösung hat, und ein Algorithmus unter Umständen nicht terminiert.

Eine Möglichkeit zu prüfen, ob ein gegebener Fluß x maximal ist, liefert das Max-Flow-Min-Cut-Theorem.

Theorem 2.3.2 (Max-Flow-Min-Cut) Die folgenden drei Aussagen sind äquivalent:

1. x ist ein maximaler Fluß
2. Zu einem Fluß x gibt es keinen zunehmenden Weg von s nach t
3. es gibt einen Fluß x mit $v = u[S, \bar{S}]$ für einen s-t-Schnitt $[S, \bar{S}]$

(1)⇒(2): Annahme: Es gibt doch einen zunehmenden Weg von s nach t. Dann kann der Fluß von s nach t auf diesem Weg erhöht werden, weshalb er vorher nicht maximal gewesen sein konnte.

(2)⇒(3): Setze $S := \{i \in N \mid i = s$ oder es gibt einen zunehmenden Weg von s nach $i\}$, $\bar{S} := N - S$. Da es keinen zunehmenden Weg von s nach t gibt, liegen die beiden Knoten in verschiedenen Partitionen.

(3)⇒(1): Ein Fluß kann nicht größer sein als ein Schnitt, da:

$$\begin{aligned} v &= \sum_{i \in S} \left[\sum_{\{j \mid (i,j) \in A\}} x_{ij} - \sum_{\{j \mid (j,i) \in A\}} x_{ji} \right] \\ &= \sum_{(i,j) \in (S, \bar{S})} x_{ij} - \sum_{(i,j) \in (\bar{S}, S)} x_{ij} \\ &\leq \sum_{(i,j) \in (S, \bar{S})} x_{ij} \\ &\leq \sum_{(i,j) \in (S, \bar{S})} u_{ij} \\ &= u[S, \bar{S}] \end{aligned} \qquad \square$$

Das Maximum Flow Problem läßt sich folgendermaßen als MCFP formulieren: Erweitere das Netzwerk $\mathcal{N} = (N, A, u)$ zu einem Netzwerk $\mathcal{N}' = (N, A, c, u, b)$ mit

$$\forall i \in N: \quad b(i) = \begin{cases} |A| \cdot U & \text{falls i=s} \\ -|A| \cdot U & \text{falls i=t} \\ 0 & \text{sonst} \end{cases} \tag{2.3.15}$$

$$\forall (i,j) \in A: \quad c_{ij} = 0 \text{ und } l_{ij} = 0 \tag{2.3.16}$$

und füge eine neue Kante (s, t) mit

$$c_{st} = 1 \tag{2.3.17}$$

$$u_{st} = \infty \tag{2.3.18}$$

$$l_{st} = 0 \tag{2.3.19}$$

ein. U bezeichnet die maximale Kapazität aller Kanten.

Die Lösung des MCFP wird so viel Fluß wie möglich durch das ursprüngliche Netzwerk schicken, da dort keine Kosten entstehen. Der Rest wird dann erst über die teure Kante (s, t) geschickt.

Es gibt jedoch effizientere Algorithmen, um einen maximalen Fluß zu suchen. Zum Beispiel genügt es, im angegebenen MCFP-Algorithmus irgendwelche zunehmenden Wege zu suchen, also nicht unbedingt kürzeste. Eine andere Möglichkeit ist die Preflow-Push-Methode nach Karzanov. Geschickt implementiert erreicht man eine Laufzeit von $\mathcal{O}(nm \cdot log(n^2/m))$.

2.3.4.3 Transportation Problem

Beim Transportation Problem wird eine Möglichkeit gesucht, ein Gut so billig wie möglich von den Erzeugern zu den Verbrauchern zu transportieren. Dies ist ein Spezialfall des MCFP, wobei die Knotenmenge N in die Menge N_1 der Erzeuger und die Menge N_2 der Verbraucher partitioniert ist, und nur Kanten von Erzeugern zu Verbrauchern existieren. Es gilt also:

$$\forall i \in N_1 \quad : \quad b(i) > 0 \tag{2.3.20}$$

$$\forall i \in N_2 \quad : \quad b(i) < 0 \tag{2.3.21}$$

$$\forall (i,j) \in A \quad : \quad i \in N_1 \land j \in N_2 \tag{2.3.22}$$

2.3.5 Verallgemeinerungen

2.3.5.1 Convex Cost Flow Problem

Bisher waren die an einer Kante entstehenden Kosten immer linear abhängig vom Fluß über die Kante $(c_{ij}x_{ij})$. Um mehr Probleme bearbeiten zu können, werden nun konvexe Funktionen zugelassen. Eine Funktion f heißt *konvex*, wenn es zu jedem Punkt a aus dem Inneren ihres Definitionsbereiches eine Funktion $\tau(x) = \alpha + \beta x \quad (\alpha, \beta \in \mathbb{R})$ mit den Eigenschaften $\tau(a) = f(a)$ und $\tau(x) \leq f(x)$ gibt [Wal92]. Die Kosten einer Kante bezeichnet man mit $C_{ij}(x_{ij})$. Gegeben ist nun also ein Netzwerk $\mathcal{N} = (N, A, C, u, b)$ mit:

$G = (N, A)$	gerichteter Graph
$C : A \to (\mathbb{R}^+ \to \mathbb{R})$	konvexe Kostenfunktionen
$u : A \to \mathbb{N}$	Kapazitätsgrenze
$b : N \to \mathbb{Z}$	Angebot-Bedarf

Gesucht ist ein Fluß $x : A \to \mathbb{N}$, bei dem

$$z(x) = \sum_{(i,j)\in A} C_{ij}(x_{ij}) \tag{2.3.23}$$

minimal ist, und die Randbedingungen

$$\forall i \in N : \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} x_{ji} = b(i) \tag{2.3.24}$$

$$\forall (i,j) \in A : \quad 0 \leq x_{ij} \leq u_{ij} \tag{2.3.25}$$

$$\tag{2.3.26}$$

eingehalten werden.

Die Beschränkung auf ganzzahlige Lösungen ist hier zwar eine Einschränkung, durch Skalierung mit einem angemessenen Faktor kann jedoch eine beliebig hohe Genauigkeit erreicht werden. Durch die Beschränkung ist es möglich, die konvexe Funktion linear zu approximieren. Um die dadurch unter Umständen stark zunehmende Zahl der Kanten einzuschränken, kann der MCFP-Algorithmus so modifiziert werden, daß diese Kanten erst nach und nach eingefügt werden.

2.3.5.2 Multicommodity Flow Problem

In der Praxis treten nicht nur Fälle auf, in denen ein Gut unabhängig von anderen Gütern über ein Netzwerk geschickt werden soll. Auch für verschiedene Güter, die sich nicht unbedingt physikalisch unterscheiden, sondern etwa „nur“ durch Quell- und Zielort, werden optimale Flüsse gesucht.

Eine Möglichkeit, solche Anwendungen zu modellieren, ist das Multicommodity Flow Problem. Gegeben ist ein Netzwerk $\mathcal{N} = (N, A, K, c, u, v, b)$ mit

$G = (N, A)$	gerichteter Graph
K	endliche, nichtleere Menge der Güter
$c : A \times K \to \mathbb{N}$	Kostenfaktor
$u : A \to \mathbb{N}$	Kapazitätsgrenze
$v : A \times K \to \mathbb{N}$	Kapazitätsgrenze pro Gut
$b : N \times K \to \mathbb{Z}$	Angebot-Bedarf

Gesucht ist ein Fluß $x : A \times K \to \mathbb{R}^+$ mit

$$z(x) = \sum_{k\in K} \left(\sum_{(i,j)\in A} c_{ijk} x_{ijk} \right) \tag{2.3.27}$$

minimal und den Randbedingungen

$$\forall (i,j) \in A : \sum_{k \in K} x_{ijk} \leq u_{ij} \tag{2.3.28}$$

$$\forall (i,j) \in A, k \in K : \quad 0 \leq x_{ijk} \leq u_{ijk} \tag{2.3.29}$$

$$\forall k \in K, i \in N : \sum_{\{j|(i,j)\in A\}} x_{ijk} - \sum_{\{j|(j,i)\in A\}} x_{jik} = b_k(i) \tag{2.3.30}$$

Die Güter nehmen sich nun wegen (2.3.28) gegenseitig den Platz auf den Kanten weg.

2.3.5.3 Generalized Flow Problem

Eine andere Möglichkeit, das Modell des MCFP zu erweitern, ist es, Flußvermehrung und -verminderung an den Kanten zuzulassen.

Gegeben ist ein Netzwerk $\mathcal{N} = (N, A, c, u, b, \mu)$ mit

$G = (N, A)$	gerichteter Graph
$c : A \to \mathbb{N}$	Kostenfaktor
$u : A \to \mathbb{N}$	Kapazitätsgrenze
$b : N \to \mathbb{Z}$	Angebot-Bedarf
$\mu : A \to \mathbb{Q}$	Output/Input

Jede Kante hat nun einen Faktor μ_{ij}, der das Verhältnis von die Kante verlassendem Fluß zu eintretendem Fluß angibt. Wenn also a Einheiten in die Kante hineingehen, kommen am Ende $a\mu_{ij}$ heraus. Dabei entstehen die Kosten pro eintretender Flußeinheit. Gesucht ist nun ein ganzzahliger Fluß $x : A \to \mathbb{N}^+$ mit

$$z(x) = \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.3.31}$$

minimal und den Randbedingungen

$$\forall i \in N : \sum_{\{j|(i,j)\in A\}} x_{ij} - \sum_{\{j|(j,i)\in A\}} \mu_{ji} x_{ji} = b(i) \tag{2.3.32}$$

$$\forall (i,j) \in A : \quad 0 \leq x_{ij} \leq u_{ij} \tag{2.3.33}$$

2.3.6 Das Integer Generalized Flow Problem ist NP-hart

Um zu zeigen, daß das Integer Generalized Flow Problem (IGFP) NP-hart ist, reduziere ich es auf das 3-SAT Problem. 3-SAT ist das Problem, zu entscheiden, ob eine gegebene Formel in konjuktiver Normalform mit je drei Literalen pro Klausel erfüllbar ist. Dazu wird zu einer gegebenen Formel $F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ mit $C_i = \{C_{i1}, C_{i2}, C_{i3}\}$ und den Variablen $X_1, X_2, \ldots, X_n$ das folgende Netzwerk $\mathcal{N} = (N, A, c, u, b, \mu)$ konstruiert:

$$
\begin{array}{lcll}
X & = & \{X_i \mid 1 \leq i \leq n\} & \\
X^T & = & \{X_i^T \mid 1 \leq i \leq n\} & \\
X^F & = & \{X_i^F \mid 1 \leq i \leq n\} & \\
C & = & \{\{C_{i1}, C_{i2}, C_{i3}\} \mid 1 \leq i \leq m \wedge C_{ix} \in X^T \cup X^F\} & \\
N & = & \{s, t, F\} \cup X \cup X^T \cup X^F \cup C & \\
\multicolumn{3}{l}{\forall i \in N : b(i) = \begin{cases} n, & \text{falls } i = s \\ -m, & \text{falls } i = t \\ -nm + m, & \text{falls } i = F \\ 0, & \text{sonst} \end{cases}} & \\
A_1 & = & \{(s, X_i) \mid 1 \leq i \leq n\}, & \forall (i,j) \in A_1 : u_{ij} = 1 \wedge \mu_{ij} = 1 \\
A_2 & = & \{(X_i, X_i^T) \mid 1 \leq i \leq n\}, & \forall (i,j) \in A_2 : u_{ij} = \infty \wedge \mu_{ij} = m \\
A_3 & = & \{(X_i, X_i^F) \mid 1 \leq i \leq n\}, & \forall (i,j) \in A_3 : u_{ij} = \infty \wedge \mu_{ij} = m \\
A_4 & = & \{(x, C_i) \mid 1 \leq i \leq m \wedge x \in C_i\}, & \forall (i,j) \in A_4 : u_{ij} = \infty \wedge \mu_{ij} = 1 \\
A_5 & = & \{(C_i, t) \mid 1 \leq i \leq m\}, & \forall (i,j) \in A_5 : u_{ij} = 1 \wedge \mu_{ij} = 1 \\
A_6 & = & \{(C_i, F) \mid 1 \leq i \leq m\}, & \forall (i,j) \in A_6 : u_{ij} = 1 \wedge \mu_{ij} = 1 \\
A & = & A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6, & \forall (i,j) \in A : c_{ij} = 0
\end{array}
$$

Abbildung 2.41 zeigt das entstehende Netzwerk für die Formel $F = (a \vee b \vee c) \wedge (a \vee \neg b \vee \neg c)$. Wenn ein Wert nicht angegeben ist, gelten die folgenden Defaults: $c = 0, u = \infty, b = 0, \mu = 1$. Die dicken Linien stellen die Kanten dar, deren Fluß bei der Belegung X_1=False, X_2=True und X_3=False größer Null ist.

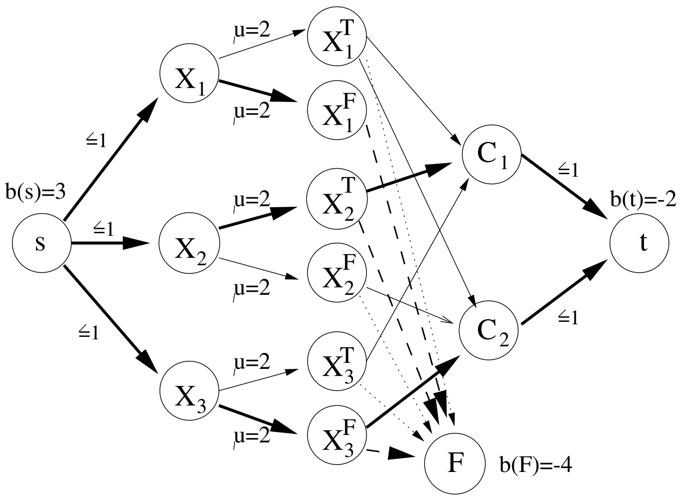


Abbildung 2.41: 3-SAT$\leq_p$IGFP

Theorem 2.3.3 Das konstruierte IGFP hat genau dann eine Lösung, wenn die Formel F erfüllbar ist.

Beweis 2.3.2 $\Rightarrow$: Es gibt einen (minimalen) Fluß x. Da alle n Kanten in A_1 die Kapazitätsgrenze 1 haben und $b(s) = n$ ist, ist der Fluß über jede dieser Kanten 1. Da der Fluß ganzzahlig sein muß, kann er sich nach den Knoten X_i nicht aufspalten. Deshalb kann man eine Belegung $\mathcal{A}$ der Variablen folgendermaßen definieren:

$$\forall (X_i, X_i^T) \in A_2 : \quad (x_{ij} > 0) \Rightarrow \mathcal{A}(X_i) := \text{True} \tag{2.3.34}$$

$$\forall (X_i, X_i^F) \in A_3 : \quad (x_{ij} > 0) \Rightarrow \mathcal{A}(X_i) := \text{False} \tag{2.3.35}$$

Da auch alle Kanten in A_6 einen Fluß von eins haben müssen und wegen der Konstruktion der Kanten in A_4 ist die Formel mit dieser Belegung erfüllt.

$\Leftarrow$: Es gibt eine Belegung, die F erfüllt. Es kann also für alle $i \in \{1, \dots, n\}$ ein gültiger Fluß von s zu entweder X_i^T, wenn $X_i = True$, oder zu X_i^F fließen. Wegen der Konstruktion von A_4 gibt es mindestens einen Fluß zu allen C_i. Der Fluß zum Knoten F ist unkritisch und dient nur dem Ausgleich von Angebot und Bedarf.

2.3.7 Anwendung

Zum Abschluß stelle ich drei Möglichkeiten vor, das MCFP für die Transportoptimierung anzuwenden. Die erste Möglichkeit aus Abbildung 2.42 erlaubt die Zusammenstellung von kürzesten Routen. An den Kanten stehen jeweils die Kosten für die Fahrt entlang dieser Kante. Die Kunden P_1, P_2 und P_3, die alle vom ihrem Wohnort nach S wollen, werden zu möglichst günstigen Touren zusammengefasst. Die vorhandenen Sitzplätze in den Fahrzeugen werden jedoch nicht berücksichtigt und es können sehr lange Touren entstehen.

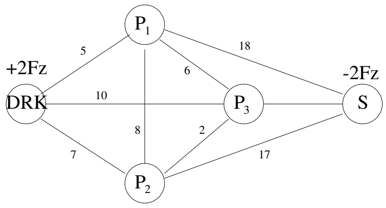


Abbildung 2.42: Routenplanung

In der zweiten Variante werden die Personen auf die Fahrzeuge verteilt. Hier werden die Kapazitäten der Fahrzeuge und die besonderen Anforderungen der Fahrgäste berücksichtigt. In dieser Variante ist es aber nicht möglich, kurze Touren zu suchen.

Um die berücksichtigten Aspekte beider Varianten zu kombinieren, kann man einen Knoten als Belegung eines Sitzplatzes mit einer Person betrachten. Dadurch entsteht aber ein nahezu vollständiger Graph mit sehr vielen Knoten (|Personen| * |Sitzplätze| + ϵ), was diese Variante fast unverwendbar macht.

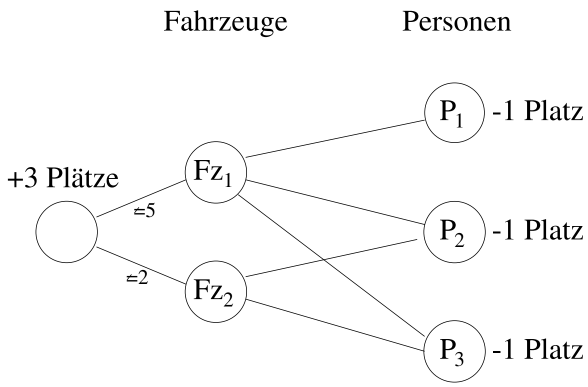


Abbildung 2.43: Fahrzeugbelegung

2.4 Nachbarschaftssuche im $\mathbb{R}^d$ (Jörg Fleischmann)

2.4.0.1 Einführung

Häufig müssen viele (konkurrierende) Wünsche mehrerer Kunden möglichst gut befriedigt werden. Wir können die Kundenwünsche als Schlüssel, die Teilwünsche als dessen Attribute modellieren. Um nun Kunden mit ähnlichen Anforderungen zusammenzufassen (z.B. ähnlicher Abfahrts- und Zielort, sowie ähnliche Abfahrts- und Ankunftszeit) werden wir die Attribute, $(x_1, x_2, \dots, x_d)$ als Koordinaten und die Schlüssel x als Punkte im d-dimensionalen Kartesischen Raum $\mathbb{R}^d$ auffassen. In diesem Raum können wir dann geometrische Algorithmen anwenden, um nahe beieinanderliegende Punkte (=ähnliche Kundenwünsche) zu bestimmen.

Im Rahmen des Vortrages werden wir uns weitgehend im $\mathbb{R}^2$ bewegen, da die Probleme im Zweidimensionalen einfacher und anschaulicher zu erläutern sind, eine Erweiterung auf d Dimensionen aber meist keine Probleme bereitet.

2.4.1 Vorbemerkungen

Zunächst sollen einige Begriffe, auf die später öfter zurückgegriffen wird, definiert werden.

Definition 2.4.1 Ein **Rechteck** $\mathcal{R}$ ist Teil einer Ebene und wird durch das kartesische Produkt $(x_1, x_2) \times (y_1, y_2)$ eines x-Intervalls (x_1, x_2) und eines y-Intervalls (y_1, y_2) definiert. Die Variablen x_1 und y_1 dürfen dabei auch den Wert $-\infty$, die Variablen x_2 und y_2 den Wert ∞ annehmen. Rechtecke dürfen an den (einzelnen) Seiten offen oder geschlossen sein.

Analog wird im $\mathbb{R}^d$ eine **Region** $\mathcal{R}$ als kartesisches Produkt aus d Intervallen definiert.

Desweiteren bezeichnen wir mit

- $U_i, 0 \leq i < d$ eine geordnete Menge (von Schlüsseln)
- $U = U_0 \times U_1 \times \cdots \times U_{d-1}$ unser Universum
- $x = (x_0, \dots, x_{d-1}) \in U$ ein Element, auch Punkt, Datensatz oder d-Tupel genannt
- der Komponente x_i die i-Koordinate

Definition 2.4.2 Mit **Level** wird die jeweilige Baumebene bezeichnet. Die Wurzel hat das Level 0, bei jeder weiteren Ebene erhöht sich das Level um 1. Es gilt also:

1. Level(root) = 0
2. Level(v) = Level(u) + 1; u ist Vaterknoten von v

Operationen: Auf einer gegebenen Punktmenge $S \subseteq \mathbb{R}^d$ sollen folgende Operationen effizient durchgeführt werden:

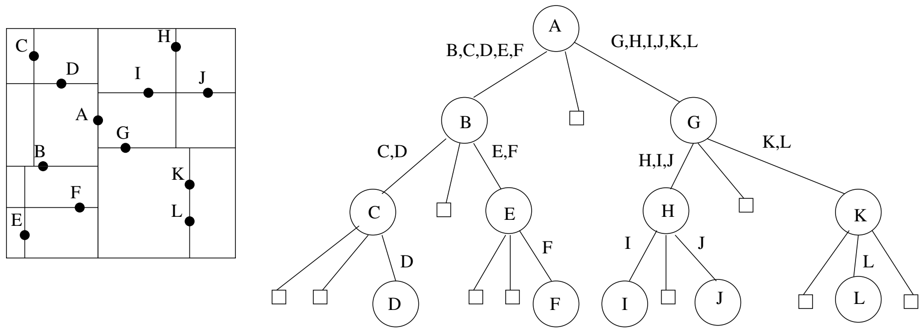


Abbildung 2.44: Aufbau eines dd-Trees

- Einfügen(p)
- Löschen(p)
- (exakte) Suche(p)
- Bereichsanfrage(D) : $\{q|q \in D \cap S\}$

Dabei sei $p \in S$ ein Punkt und D ein Rechteck bzw. eine Region.

2.4.2 Baumstrukturen

Bäume sind bekanntermaßen, zumindest im eindimensionalen Raum, effiziente Suchstrukturen. Es liegt also nahe, auch Punkte des mehrdimensionalen Raums in Bäumen abzulegen. Dies geht sogar ohne Änderung der bekannten Suchbäume: Bilde aus den d Attributen/Koordinaten einen einzigen *Superschlüssel*. Dieses Vorgehen ist ausreichend, solange man sich auf Operationen beschränkt, die auch im eindimensionalen Raum möglich sind (Suchen, Einfügen und Löschen). Um auch die Bereichssuche unterstützen zu können, bedarf es jedoch einer Erweiterung der bekannten Suchbäume. Zwei Erweiterungsmöglichkeiten sollen hier besprochen werden: *Quadranten-* und *dd-Bäume*.

2.4.2.1 dd-Bäume – d-dimensionale Bäume

Die dd-Bäume sind die natürliche Erweiterung der eindimensionalen binären Suchbäume. Die Teilung der Punktemenge erfolgt dabei entlang einer (mit jeder Baumebene zyklisch wechselnden) Koordinate. Damit der Baum möglichst ausgeglichen wird, legt man die Schnittlinie durch den Punkt, der den Median bezüglich der Schnittkoordinate darstellt (s. Abbildung 2.44).

Im Gegensatz zu den bekannten binären Suchbäumen verwenden wir einen trinären Baum, der folgendermaßen definiert wird:

Definition 2.4.3 [Meh84] Sei $S \subseteq U_0 \times \cdots \times U_{d-1}, |S| = n$. Ein d**d-Baum** für S, beginnend mit der i-ten Koordinate, wird dann wie folgt definiert:

1. Wenn $d = n = 1$ ist, besteht er aus einem einzigen Blatt, das mit dem einzigen Element $x \in S$ bezeichnet wird.
2. Andernfalls besteht er aus

- einer Wurzel, die mit dem Element $d_i \in U_i$ bezeichnet wird und
- drei Teilbäumen

  $T_<$ ein dd-Baum beginnend bei Koordinate $i+1 \bmod d$ für folgende Teilmenge von S:
  $S_< = \{x \in S|\ x = (x_0, \dots, x_{d-1}) \text{ und } x_i < d_i\}$

  $T_=$ ein $(d-1)$-dimensionaler Baum, beginnend bei Koordinate $i - 1 \bmod d$ für die Menge

  $$\begin{aligned} S_= = \{(&x_0, \dots, x_{i-1}, x_{i+1}, \dots, x_{d-1})| \\ &x = (x_0, \dots, x_{i-1}, d_i, x_{i+1}, \dots, x_{d-1}) \in S\} \end{aligned}$$

  $T_>$ ein dd-Baum beginnend bei Koordinate $i+1 \bmod d$ für folgende Teilmenge von S:
  $S_> = \{x \in S|\ x = (x_0, \dots, x_{d-1}) \text{ und } x_i > d_i\}$

Ist w ein Vaterknoten zum Knoten v und $l(w)$ eine Schnittlinie senkrecht zur Koordinate $i \in \{0, 1\}$, dann assoziieren wir (für $d = 2$) mit jedem Knoten v:

- einen Punkt $P(v)$, wobei P_i die i-te Koordinate von $P(v)$ bezeichnet
- eine Schnittlinie $l(v)$, die $P(v)$ schneidet und parallel zur x-, oder y-Achse verläuft
- ein Rechteck $\mathcal{R}(v)$, welches vom Knotentyp ($<, =, >$, Wurzel) abhängt:

  Wurzel: $\mathcal{R}(v) := S$
  <: $\mathcal{R}(v) := \{x \in \mathcal{R}(w)|\ x_i < (l(w), \mathcal{R}(w))\}$
  =: $\mathcal{R}(v) := \{x \in \mathcal{R}(w) \backslash \{w\}|\ x_i = (l(w), \mathcal{R}(w))\}$
  >: $\mathcal{R}(v) := \{x \in \mathcal{R}(w)|\ x_i > (l(w), \mathcal{R}(w))\}$

Die Definition 2.4.3 kann direkt in einen Algorithmus zum Aufbau eines dd-Baumes aus einer Menge $S \subseteq \mathbb{R}^d$ ab Baumhöhe l umgesetzt werden (siehe Algorithmus 1).

Aufwand zum Aufbau eines dd-Baumes Wir wollen nur den Aufwand für einen idealen Binärbaum (Definition 2.4.4) ohne =-Zeiger[1] betrachten, da dies leichter zu zeigen ist und das Zeitverhalten in der gleichen Größenordnung liegt (an jedem =-Zeiger hängt nur noch ein $(d-1)$d-Baum, bzw. bei $d = 2$ nur eine sortierte Liste (Quicksort nach [OW93]: $O(n \log n)$).

Definition 2.4.4 Ein dd-Baum T heißt **ideal**, wenn gilt: Keiner der beiden Teilbäume $T_<, T_>$ von T hat mehr als die Hälfte der Knoten von T, d.h. es gilt: $|T_<| \leq |T| \ / \ 2 \quad \wedge \quad |T_>| \leq |T| \ / \ 2$.

Der Zeitaufwand zum Aufbau eines dd-(Teil-)Baumes mit n Knoten sei $T(n)$. Interessant sind nur die Zeiten, die für die aufgerufenen Funktionen benötigt werden, da alle anderen Wertzuweisungen offensichtlich in konstanter Zeit erfolgen:

- median: nach [BFP+72] $O(n)$

[1] d.h. $\forall p_i, p_j \in S : p_i \neq p_j$

Algorithmus 1 build_ddTree: build a dd-tree from a given set S

Eingabe: set $S \subseteq \mathbb{R}^d$
level l {0 ist Level für Wurzel des Gesamtbaumes}
card d {Anzahl der Dimensionen}
card i {aktuelle Dimension}
Ausgabe: node *RootS* {Wurzel des dd-Baumes zur Menge S}

node: n
if $S \neq \emptyset$ **then**
  $n.P \leftarrow$ median(i, S);
  n.level $\leftarrow l$;
  n.< $\leftarrow$ build_ddtree($\{x \in S | \ x = (x_0, \ldots, x_{d-1}) \ \wedge \ x_i < n.P_i\}, \ l+1, \ d, \ i+1$ mod d);
  if $d = 1$ **then**
    n.= $\leftarrow$ quickSort($l(n.P) \cap S$);
  else
    n.= $\leftarrow$ build_ddTree($\{(x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{d-1}) | \ x = (x_0, \ldots, x_{i-1}, d_i, x_{i+1}, \ldots, x_{d-1}) \in S - n.P\}, \ l+1, \ d-1, \ i+1$ mod d);
  end if
  n.> $\leftarrow$ build_ddTree($\{x \in S | \ x = (x_0, \ldots, x_{d-1}) \ \wedge \ x_i > n.P_i\}, \ l+1, \ d, \ i+1$ mod d);
else
  n $\leftarrow$ NIL;
end if

- build_ddTree (wird zweimal mit jeweils $n/2$ Punkten aufgerufen): $2T(n/2)$

Damit beträgt der Zeitaufwand für einen Teilbaum mit n Knoten: $T(n) = O(n) + 2T(n/2)$. Da beim Binärbaum auf dem i-ten Level 2^i Knoten untergebracht werden können, erhalten wir für n Knoten eine Baumhöhe von $\log n$. Wir erhalten also:

$$\begin{aligned}
T(n) &= O(n) + 2T(n/2) \\
&= O(n) + 2O(n/2) + 4T(n/4) \\
&\vdots \\
&= \underbrace{O(n) + 2O(n/2) + 4O(n/4) + \cdots + 2^{\log n} O(n/2^{\log n})}_{\log n \text{ Summanden}} \\
&= \underbrace{O(n) + O(n) + O(n) + \cdots + O(n)}_{\log n\text{-Mal}} \\
&= \log n O(n) \\
&= O(n \log n)
\end{aligned}$$

Werden auch Knoten mit $=$-Zeigern berücksichtigt, werden am Rekursionsende (Zeile 10) die verbleibenden Punkte mittels Quicksort sortiert. Dies nimmt jedoch keinen Einfluß auf die Laufzeitabschätzung, da selbst der Worst Case (alle n Punkte müssen sortiert werden), wie schon erwähnt in $O(n \log n)$ erledigt werden kann.

Operationen auf *d*d-Bäumen Bei der **exakten Suche** muß der dd-Baum einfach per Tiefensuche von der Wurzel beginnend durchlaufen werden, bis der gesuchte Punkt gefunden, oder ein Blatt erreicht wird. Da die Suche immer nur in einem Teilbaum fortgesetzt werden muß, liegt der Worst Case (das Durchlaufen aller Level) in $O(\log n)$.

Zum **Einfügen** eines Punktes p muß zunächst eine exakte Suche nach p durchgeführt werden. Da wir keine gleichen Punkte betrachten, endet die Suche in jedem Fall (nach $\log n$ Schritten) bei einem Blatt, welches der Vaterknoten von p wird. Die Knoteninitialisierung und das Einfügen der Zeiger ist in konstanter Zeit machbar. Dies ergibt einen Gesamtaufwand von $O(\log n)$.

Beim **Löschen** von p kommt zur exakten Suche noch der Aufwand für den Neuaufbau des gesamten Unterbaumes, der am gelöschten Knoten hing dazu: $O(\log n + |\mathcal{R}(p)| \log |\mathcal{R}(p)|)$
Bei den Operationen Einfügen und Löschen ist zu beachten, daß das Ergebnis i.d.R. kein idealer Baum mehr ist! Diesen erhält man i.d.R. nur bei komplettem Neuaufbau, da zur Medianbestimmung alle Punkte betrachtet werden müssen. Man sollte also Veränderungen nur bis zu einem bestimmten Grad zulassen und dann einen kompletten Neuaufbau erzwingen.

Bereichsanfragen werden durch dd-Bäume besonders gut unterstützt: man muß die Bereichssuche immer nur bei einem von zwei Söhnen fortsetzen, sofern der Anfragebereich D nicht von der jeweiligen Schnittline $l(v)$ geschnitten wird (siehe Abbildung 2.45). Ist $D \cap l(v) \neq \emptyset$, muß geprüft werden, ob $P(v)$ in D liegt, außerdem muß die Bereichssuche in beiden Söhnen fortgesetzt werden.

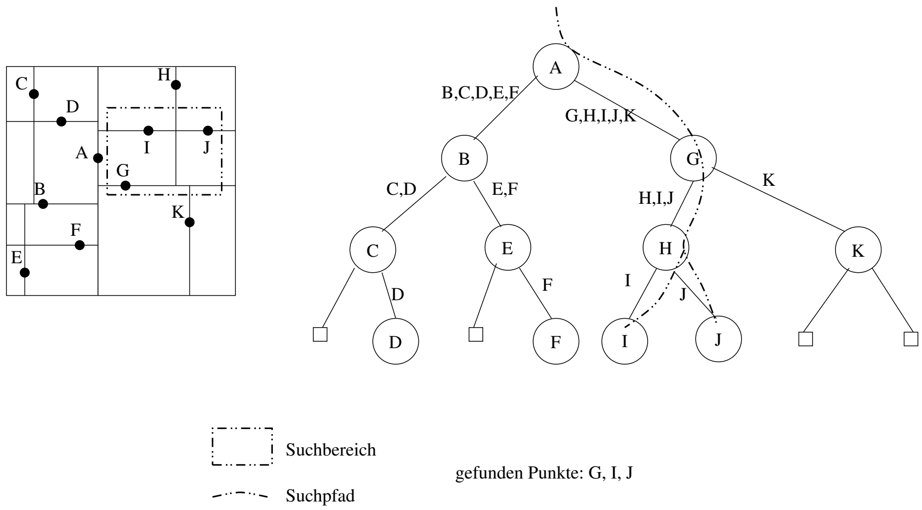


Abbildung 2.45: Bereichssuche in dd-Bäumen

Algorithmus 2 Search: Bereichssuche nach [PS95]

Eingabe: node: v {Wurzel des Suchbaumes zur Menge $S(v)$}
region: D {Suchbereich $D \subseteq \mathbb{R}^2$}
Ausgabe: set: U {Antwortmenge $U = D \cap S$}

U $\leftarrow \emptyset$
if $v.CuttingDirection$ = vertical **then**
  $[l, r] \leftarrow [x_1, x_2]$;
else
  $[l, r] \leftarrow [y_1, y_2]$;
end if
if $l \leq v.CuttingLine \leq r$ **then**
  if $P(v) \in D$ **then**
    $U \leftarrow U \cup P(v)$;
  end if
end if
if $v \neq$ leaf **then**
  if $l < v.CuttingLine$ **then**
    Search(LSon[v], D);
  end if
  if $v.CuttingLine < r$ **then**
    Search(RSon[v], D);
  end if
end if

Aufwand für die Bereichssuche in dd-Bäumen

Theorem 2.4.1 Sei T ein idealer 2d-Baum für eine n-elementige Punktemenge $S \subseteq \mathbb{R}^2$. Dann gilt für den Zeitaufwand der Bereichssuche $T(n)$: $T(n) \in O(\sqrt{n})$.

Beweis 2.4.1 [Buc95] Sei D der Suchbereich, T der Suchbaum zur Punktemenge S und T' der Teilbaum von T, der alle Knoten enthält, die während der Bereichssuche besucht werden, d.h. $v \in T' \Leftrightarrow R(v) \cap D \neq \emptyset$.

Den Worst Case erhalten wir, wenn keiner der besuchten Knoten zur Antwortmenge gehört (solche Knoten nennen wir *u*nproduktive Knoten). Sei nun Y die Menge der möglichen unproduktiven Knoten:

$$Y = \{v \in T' |\ R(v) \cap D \neq \emptyset \ \wedge \ R(v) - D \neq \emptyset\}$$

Weiter unterscheiden wir nach der Anzahl der Dimensionen (dabei sei L der Rand der angefragten Region D):

$$\begin{aligned} W = \{v|\ v \in Y,\ R(v) \cap L \neq \emptyset,\ R(v) \text{ ist zweidimensional}\} \\ \Rightarrow \text{Pfad nach } w \in W \text{ enthält keine } =\text{-Zeiger} \end{aligned}$$

$$\begin{aligned} Z = \{v|\ v \in Y,\ R(v) \cap L \neq \emptyset,\ R(v) \text{ ist eindimensional}\} \\ \Rightarrow \text{Pfad nach } z \in Z \text{enthält mindestens einen } =\text{-Zeiger} \end{aligned}$$

Nun zählen wir die Elemente aus W, dabei sei W_k die Teilmenge von W, die alle Knoten aus dem Level k enthält. Dann gilt:

$\forall v \in W_k$: auf Level $k+2$ gibt es maximal 2 Enkel von v, d.h.
$|W_{k+2}| \leq 2|W_k|$,
für die Wurzel gilt: $|W_0| \leq |W_1| \leq 2$
$\Rightarrow |W_k| \leq 2 \cdot |W_{k-2}| \leq \cdots \leq 2^{k/2}|W_0| \leq 2 \cdot 2^{l/2}$.

Als nächstes bestimmen wir die Anzahl der Knoten $z \in Z$, welche Nachfolger von Knoten $w \in W$ sein müssen: sei z Nachfolger von $w \in W_k$, dann gilt: $S(z) \subseteq S(w) \Rightarrow |S(z)| \leq |S(w)| \leq n/2^k$. Damit erhalten wir für den Worst Case die maximale Zeit $t_{\max}$:

$$\begin{aligned} t_{\max} = |W| + |Z| \ &\leq \ \sum_{k=0}^{\log n} \underbrace{2 \cdot 2^{k/2}}_{W_k} \log n/2^k \\ &= 2 \sum_{k=0}^{\log n} 2^{k/2}(\log n - k) \\ &= 2\sqrt{n} \sum_{k=0}^{\log n} 2^{\frac{k-\log n}{2}}(\log n - k) \\ &= 2\sqrt{n} \sum_{k=0}^{\log n} (1/\sqrt{2})^k k \quad \in O(2\sqrt{n}) \end{aligned}$$

2.4.2.2 Quadrantenbäume – Quad Trees

Quadrantenbäume sind *Bäume vierter Ordnung*, d.h. jeder innere Knoten hat Söhne, die folgendermaßen aufgebaut werden (s. Abbildung 2.46):

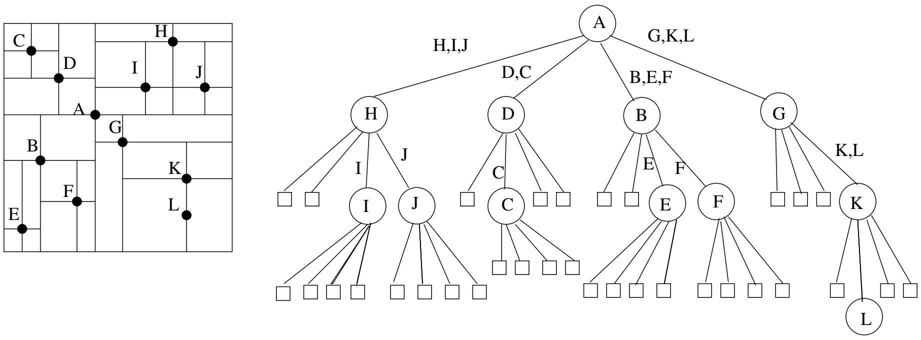


Abbildung 2.46: Aufbau eines Quad-Trees

1. Nehme einen beliebigen Punkt p der Ebene und speichere ihn in der Wurzel w.
2. Lege ein Koordinatenkreuz durch p; dieses teilt die Ebene in vier Quadranten.
3. Jeder Sohn von w ist Wurzel eines Teilbaums, der einen Quad Tree bezüglich eines Quadranten repräsentiert.
4. Iteriere dieses Verfahren über alle entstehenden Quadranten, bis keine Punkte mehr vorhanden sind.

Nicht besprochen wurde, was mit Punkten geschieht, die direkt auf einer der Quadrantenlinien liegen. Für diese gilt folgende Vereinbarung: Quadranten eins und drei sind geschlossen, die Quadranten zwei und vier sind dagegen offen. Punkte, die auf einer Quadrantenlinie liegen, fallen damit eindeutig entweder in den ersten oder dritten Quadranten.

Ein weiterer Punkt, der außeracht gelassen wurde, ist das Auftreten von gleichen Datensätzen bzw. Punkten. Dies ist beispielsweise dadurch zu lösen, daß ein weiterer Zeiger vom Knoten auf eine Liste von Datensätzen mit gleichen Schlüsseln zeigt.

Operationen auf Quadrantenbäumen Die Operationen **Suchen** und **Einfügen** laufen analog zu denen der dd-Bäume. Ein Problem stellt wieder das **Löschen** dar, falls der zu entfernende Punkt nicht nur Blätter als Söhne hat. Am Einfachsten ist in diesem Fall, alle durch das Entfernen abgehängten Knoten wieder einzeln einzufügen.

Die **Bereichssuche** arbeitet folgendermaßen: Beginne bei der Wurzel und prüfe, ob der dort gespeicherte Punkt im Anfragebereich liegt. Setze die Bereichssuche bei all den Söhnen fort, deren zugehöriger Quadrant einen nichtleeren Durchschnitt mit dem gegebenen Bereich hat.

Aufwand bei Quadratenbäumen Bentley ermittelte durch Simulation, daß das Einfügen von n Knoten in zufälliger Reihenfolge in $O(n \log n)$ liegt ([FB74]). Für die Suche werden damit durchschnittlich $\log n$ Knoten besucht. Den *Worst Case* erhält man, wenn die Nachfolgeknoten jeweils an den untersten Knoten angehängt werden, d.h. wenn der Baum zu einer Liste entartet. Dann müssen für das n-malige Einfügen bzw. Suchen maximal $\frac{n(n-1)}{2}$ Knoten besucht werden und der Aufwand liegt damit in $O(n^2)$.

Der quadratische Worst Case kann durch eine Medianstrategie vermieden werden (verwende als nächsten Ursprung immer den Median bezüglich einer Koordinate). Dadurch verlieren die Quad Trees jedoch ihre Dynamik (Einfügen / Löschen nur noch durch Neuaufbau des Baumes). Außerdem erzielt man mit den idealen Quad Trees auch zeitlich keine Verbesserungen gegenüber den dd-Trees.

2.4.3 Das Gitterverfahren – mehrdimensionales Hashing

Eine völlig andere Möglichkeit, den Raum einzuteilen, stellt das Gridfile dar.

Die Idee des Gitterverfahrens ist die Zerlegung der zu durchsuchenden Fläche mit Hilfe eines künstlich darübergelegten Gitters. Zu jeder Teilfläche wird eine Liste mit allen darin enthaltenen Punkten angelegt. Dadurch wird die zunächst unübersichtliche Punktmenge in überschaubare Einheiten unterteilt.

Allen bekannt dürfte diese Methode durch Landkarten sein: man schaut in einer Liste, in welchem Planquadrat der gesuchte Ort liegt und kann einen Ort/Punkt dann in diesem kleinen Kartenausschnitt suchen, anstatt die gesamte Karte abzusuchen.

Für die Brauchbarkeit des Gitterverfahrens ist die Wahl der Gittergröße entscheidend: ist sie zu groß, findet man wieder nichts, ist sie dagegen zu klein, sind die Gitterzellen zu dünn besetzt ($\rightarrow$ hoher Verwaltungsaufwand und es ist schwieriger die Zellen zu finden, als die zugehörigen Punkte). Zudem sollte die Anzahl der Gitterzellen ein konstanter Bruchteil der Gesamtzahl der Punkte sein, da dann zu erwarten ist, daß die Anzahl der Punkte in jedem Quadrat ungefähr gleich ist (gilt nur bei gleichverteilten Punkten, bei Landkarten ist dies nicht der Fall (Ballungsgebiete, Wälder)). Möchte man n Punkte pro Gitterquadrat haben, wähle für die Größe der Gitterquadrate $size$ die ganze Zahl, die dem maximalen Abstand vom Ursprung max dividiert durch $\sqrt{|P|/n}$ am nächsten kommt $\rightarrow$ ca. $|P|/n$ Gitterquadrate.

Satz 2.4.1 [Sed91]: Das Gitterverfahren für die Bereichssuche ist im durchschnittlichen Fall linear bezüglich der Anzahl der Punkte im Bereich und im ungünstigsten Fall linear bezüglich der Gesamtzahl der Punkte.

Damit ist das Gitterverfahren gut geeignet, falls die Punkte gleichmäßig verteilt sind, mit zunehmender Punkthäufung wird es jedoch schlechter.

2.4.3.1 Das Gitterverfahren formal betrachtet

Wir bewegen uns in einem d-dimensionalen Raum aus Schlüsseln und versuchen diesen, in Anlehnung an das bekannte eindimensionale Hashing, in mehrdimensionale Rechtecke einzuteilen, die gerade das Produkt eindimensionaler Intervalle sind; dort wird dann das eindimensionale Hashing angewandt.

Grundbegriffe (siehe auch Abbildung 2.47):

d**:** Dimension

D**:** Menge der Dimensionen $D = \{1, \ldots, d\}$.

I**:** Teilmenge von D.

S_i: Universum der $i-ten$ Schlüsselkomponente $1 \leq i \leq d$.

S: Universum aller möglichen d-dimensionalen Schlüssel $S = S_1 \times \cdots \times S_d$.

Gitterzelle: einzelner Teilraum T des gesamten Datenraums S: $T \subseteq S$.

Datenblock: Ein Datenblock B enthält maximal b Punkte aus T, dabei kann b beliebig gewählt werden, ist dann jedoch fest. Jedem Teilraum wird genau ein Datenblock zugewiesen, derselbe Datenblock kann auch mehreren Teilräumen zugeordnet sein.

Blockregion: Alle Gitterzellen, die dem selben Datenblock zugeordnet sind.

Scale: Einteilung des Datenraums für jede Dimension.

Directory: Adreßtabelle.

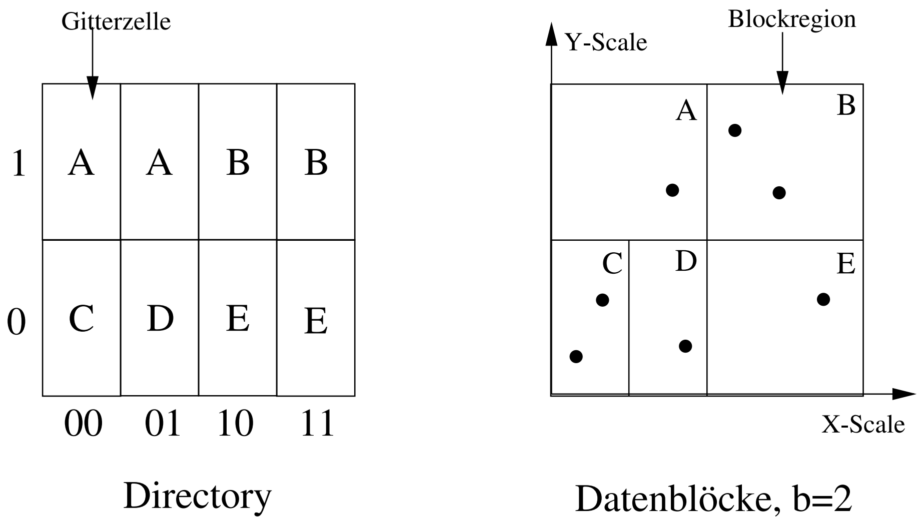


Abbildung 2.47: Gridfile

Die Schlüssel werden als Punkte im d-dimensionalen Datenraum abgebildet. Für jede Dimension gibt es eine Hashfunktion, die die Zuordnung vornimmt. Im 2-dimensionalen Fall gibt es also zwei Hashfunktionen, die die x- bzw. y-Position des Punktes in der Ebene bestimmen, der den gehashten Schlüssel repräsentiert.

Wie oben angegeben darf jeder Block eine frei wählbare (dann feste) Anzahl von Punkten aufnehmen. Im Beispiel aus Abbildung 2.47 wurde $b = 2$ gewählt. Dies führt dazu, daß man mehrere Gitterzellen zu Blockregionen/Datenblöcken zusammenfassen kann (so daß jeder Datenblock möglichst $b = 2$ Punkte enthält).

Jetzt gibt es mehrere Verweise von der Adreßtabelle auf einen Datenblock (z.B. [1,10] und [1,11] verweisen beide auf den Block B, [1,00] und [1,01] können nicht zusammengefaßt werden, da der zweite Schlüssel zur Unterscheidung der Gitterzellen C und D benötigt wird). Dieses Problem läßt sich im mehrdimensionalen Fall – im Gegensatz zum eindimensionalen – leicht beheben: hier genügen wenige Hashadressen um viele Adreßtabelleneinträge zu verwalten, da die Anzahl der Adreßtabelleneinträge das Produkt der Anzahl der Hashadressen in den verschiedenen Dimensionen ist und nicht, wie im eindimensionalen Fall, deren Summe. Für das Beispiel aus Abbildung 2.47 ergibt sich damit die Adreßtabelle in Abbildung 2.48. Damit können Hashadressen in allen Dimensionen explizit

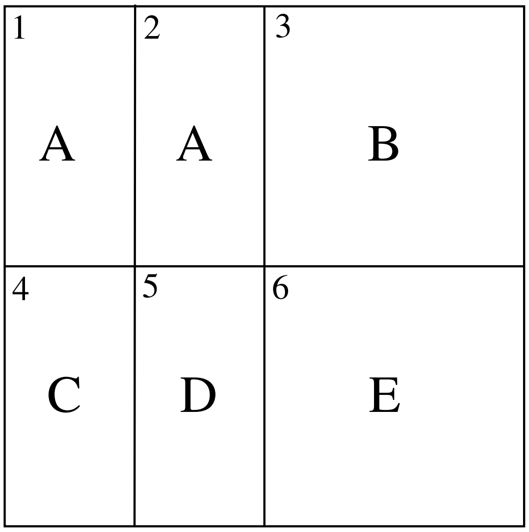


Abbildung 2.48: Gridfile

verwaltet werden und dies geht für realistische Anwendungsfälle problemlos im Hauptspeicher. Eine Adreßtabellenverdopplung wie bei einer einzigen Dimension entfällt damit. Genauere Größenanalysen der Adreßtabelle findet man in [Reg85].

Operationen auf dem Grid-File Wir wollen hier die Operationen *Suchen, Bereichsanfrage, Einfügen* und *Entfernen* besprechen. Bei den Operationen Suchen und Bereichsanfrage kann man vollständige [2] und partielle [3] Operationen unterscheiden.

Suche(p) Dabei ist p ein Punkt im zweidimensionalen Raum: $p = (x, y)$. Die Suche kann nach folgender Algorithmus-Skizze aus [OW93] erfolgen:

1. Bestimme anhand der X-Scales die Spalte s der Directory-Matrix, in die p fällt und anhand der Y-Scales die Zeile z der Directory-Matrix in die p fällt.

2. Bestimme die Externspeicheradresse a_1 des Directory-Elements in Zeile z und Spalte s.

3. Lies den Directory-Block dir mit Adresse a_1 in den Hauptspeicher.

4. Bestimme die Externspeicheradresse a_2 des Datenblocks zu derjenigen Gitterzelle in dir, in die (x, y) fällt.

[2] alle Dimensionen werden angegeben
[3] nur ein Teil der Schlüsselattribute wird angegeben

5. Lies den Datenblock *dat* mit Adresse a_2 in den Hauptspeicher

6. Durchsuche *dat* nach (x, y) und berichte das Ergebnis.

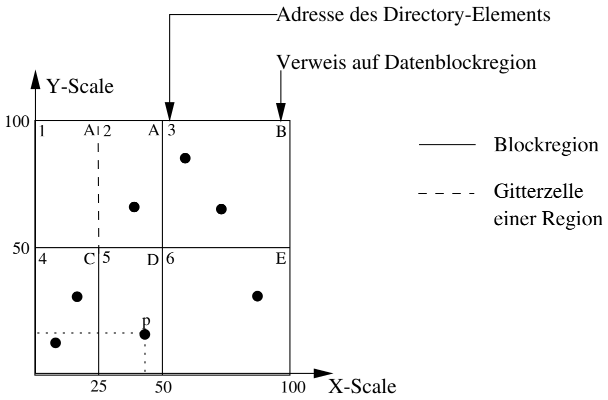


Abbildung 2.49: Gridfile

Wird beispielsweise der Punkt $p = (43, 12)$ gesucht, kommt man über die zweite Zeile und die zweite Spalte der Directory-Matrix zum Directory-Element mit der Adresse 5, welches wiederum auf den Datenblock D verweist. In diesem ist schließlich der gesuchte Punkt enthalten und die Suche endet erfolgreich (siehe Abbildung 2.49).

Falls man die Scales im Hauptspeicher verwaltet [4], findet lediglich in den Schritten 3 und 5 des Algorithmus ein Externzugriff statt, dies nennt man das *Zwei-Zugriffs-Prinzip*.

Bereichsanfrage(x,y)

1. Suche(linken unteren Punkt des Suchbereichs).

2. Überprüfe für alle Punkte im gefunden Datenblock, ob sie im Anfragebereich liegen.

3. Setze Suche nach rechts und nach oben über benachbarte Zeilen und Spalten und den daraus berechenbaren Directoryelemente fort.

Diese Vorgehensweise beinhaltet gleichzeitig eine Bereichssuche auf der Directory-Matrix, die i. a. wegen ihrer Größe auf externem Speicher verwaltet wird. Daraus resultiert, daß deren Datenstruktur dieselben Operationen unterstützen muß wie die ursprüngliche Datenstruktur. Daher ist es vernünftig, Gitterzellen und Datenpunkte in einem Gridfile zu organisieren [KW85], für welches man im Regelfall nicht mehr als zwei Ebenen benötigt.

Die Effizienz einer Bereichsanfrage ist also nach unten beschränkt durch die Effizienz der exakten Suche; mit größer werdenden Anfragebereichen steigt in der Tendenz auch die Anzahl der als Antwort gefundenen Datensätze und die Anzahl der benötigten Externzugriffe. Die Effizienz von Bereichsanfragen mit

[4] dies ist wie schon erwähnt für realistische Anwendungen möglich

großen Anfragebereichen ist eng an die Speicherplatzausnutzung gekoppelt, weil Externzugriffe, die wenig zur Antwort beitragen, nur für Gitterzellen und Datenblockregionen am Rand des Anfragebereichs ausgeführt werden müssen. Eine genaue Analyse [FP86] ergibt, daß im Mittel $O(n^{1-|I|/d})$ Externzugriffe für die partielle Suche nach $|I|$ von d Schlüsseln in einem Gridfile mit n Datensätzen ausreichen. Diese Effizienz wird für optimal gehalten [Riv76].

Einfügen Auch beim Einfügen wird zuerst eine exakte Suche durchgeführt, um den Datenblock zu ermitteln, in den der neue Datensatz eingefügt werden muß. Ist im gefunden Datenblock noch Platz für einen weiteren Datensatz, kann dieser einfach eingefügt werden und der Block wird auf den Externspeicher geschrieben. Ist der Block jedoch schon voll belegt, muß er geteilt werden. Dazu wird ein *Datenblocksplit* durchgeführt, d.h. der Block wird entlang einer Koordinatenachse in der Mitte zerschnitten. Die Datensätze werden gemäß der beiden neuen Datenblockregionen auf die beiden neuen Datenblöcke aufgeteilt und die neue Situation im Directory vermerkt. Das Directory muß als Matrix organisiert bleiben, weswegen die Splitlinie in allen von der Splitdimension verschiedenen Dimensionen den gesamten Datenraum durchtrennt. Als Sonderfall kann das Problem auftreten, daß durch den Split die Datensätze nicht wirklich verteilt werden, das heißt, daß einer der neu geschaffenen Blöcke leer bleibt. In diesem Fall wird der Blocksplit rekursiv für den noch immer übervollen Block fortgesetzt.

Für das Spliten gibt es folgende Regeln, von denen die erste eindeutig zutreffende angewendet wird:

1. an der längsten Seite,

2. gemäß einer vorhandenen Einteilung in Gitterzellen oder

3. in derjenigen Dimension, in der die kleinste Anzahl von Teilungen vermerkt ist.

Erhält man durch obige Regeln keine Splitdimension, so kann entlang einer beliebigen Dimension geschnitten werden (z.B. abwechselnd nach X und Y). Da keine bestimmte Dimension bevorzugt wird, erhält man in der Regel Blockregionen, deren Verhältnis von Länge zu Breite nahe bei 1 liegt. Durch die Existenz des Directorys, in dem auch leere Blöcke vermerkt sind, kann man sich das Speicheren dieser leeren Blöcke ersparen.

Da durch das Teilen der Blöcke meist auch die Anzahl der zu verwaltenden Directoryverweise steigt, kann es beim Directory ebenfalls zum Überlauf kommen. Diesen Überlauf behebt man ebenso mit Teilung nach obiger Strategie. Allerdings muß hier nach der Teilung noch die Einteilung der beiden neuen Blöcke in Gitterzellen überprüft werden, da diese in Folge der Teilung günstiger werden kann.

Löschen Abermals wird anfangs eine exakte Suche ausgeführt. Dadurch erhält man den Datenblock, in dem der zu löschende Datensatz steht, kann ihn entfernen und den Block wieder zurückschreiben. Weitere Aktionen sind hier nicht zwingend erforderlich. Allerdings wird ein durch Einträge auf leere Blöcke aufgeblähtes Directory nicht gerade schneller auf Suchanfragen reagieren. Deswegen kann man, analog zum Teilen der Blöcke, nun wenn möglich, zwei

Blöcke verschmelzen. Dies ist auch sinnvoll, um die Speicherausnutzung nicht unter ein bestimmtes Mindestmaß sinken zu lassen. Allerdings kann damit bei dynamischen Vorgängen das Problem auftreten, daß Blöcke nach dem Einfügen getrennt, dann durch Löschen wieder verschmolzen werden, nur um bei der nächsten Einfügeoperation wieder geteilt zu werden.

Wie kann man nun diesem Problem begegnen? Man nützt einfach aus, daß das Verschmelzen kein notwendiger Vorgang ist, indem man zwei Blöcke nur dann verschmilzt, wenn die Speicherplatzausnutzung eines Datenblocks nach Entfernen eines Datensatzes unter eine vorgegebene Schranke fällt. Dieses vergeudet nicht zuviel Speicher für eigentlich überflüssige Directoryeinträge, die Suche im Directory bleibt effizient, und die langsamen Externzugriffe, die nach jeder Verschmelzung notwendig sind, werden minimiert (es können mehrere Datensätze in einem Block gelöscht werden, bevor er verschmolzen wird, außerdem kann er auch wieder gefüllt werden, wobei man sich sogar noch die relativ aufwendige Teilung des Datenblocks erspart). Eine erfahrungsgemäß gute Schranke unterhalb der eine Verschmelzung erwogen werden sollte, liegt bei etwa 30%. Nur wenn diese Schranke unterschritten wird, schaut man nach, ob ein Partner für eine Verschmelzung existiert. Ob ein solcher Partner gefunden werden kann, hängt von der gewählten Verschmelzungsstrategie und der Gitterzelleneinteilung ab. Von allen gefundenen potentiellen Verschmelzungspartnern wird derjenige mit der schwächsten Füllung ermittelt. Eine Verschmelzung wird durchgeführt, falls nach der Verschmelzung eine vorgegebene Obergrenze – typischerweise ca. 70% – bezüglich der Füllung nicht überschritten wird.

Verschmelzungsstrategien Durch die Verschmelzungsstrategie wird festgestellt welche Regionen überhaupt für eine Verschmelzung in Frage kommen. Eine Bedingung, die unabhängig von der gewählten Strategie immer erfüllt werden muß, ist die, daß die entstehende Blockregion wieder rechteckig ist.

Nachbarstrategie

Die Nachbarstrategie läßt alle Verschmelzungen zu, bei denen ein rechteckiger Bereich entsteht. Damit sind alle vier in Abbildung 2.50 dargestellten Möglichkeiten erlaubt. Dies Nachbarschaftsstrategie ist sehr einfach und nutzt den Speicherplatz recht gut aus. Allerdings führt diese Strategie in einigen Fällen zu Verklemmungen, d.h. durch eine gültige Verschmelzung werden weitere Verschmelzungen beim Löschen weiterer Datensätze unmöglich und man erhält viele leere Blöcke. Abbildung 2.51 zeigt ein Beispiel für solch eine Verklemmung. Durch die Verklemmung kann also die Speicherplatzausnutzung beliebig abfallen, was natürlich in jedem Fall zu vermeiden ist. Man muß also Prüfungen einführen, um Verklemmungen beim Verschmelzen zu unterbinden. Daß diese Aufgabe nicht immer einfach und effizient zu lösen ist, sollte klar sein.

Bruderstrategie (buddy merge)

Die Bruderstrategie erlaubt die Verschmelzung nur, wenn die Blöcke aus einer Teilung hervorgegangen sein könnten. Dies bedeutet, daß die Verschmelzung eine (mögliche) Teilung rückgängig macht. Damit sind nur die zwei unter „Bruderstrategie“ zusammengefaßten Verschmelzungsmöglichkeiten erlaubt. Allgemein hat ein Block in jeder Dimension höchstens einen Bruder, jedoch bis zu zwei Nachbarn. Im zweidimensionalen Fall kann man also maximal zwischen zwei Brüdern wählen, bei der Nachbarstrategie hätte man schon maximal vier

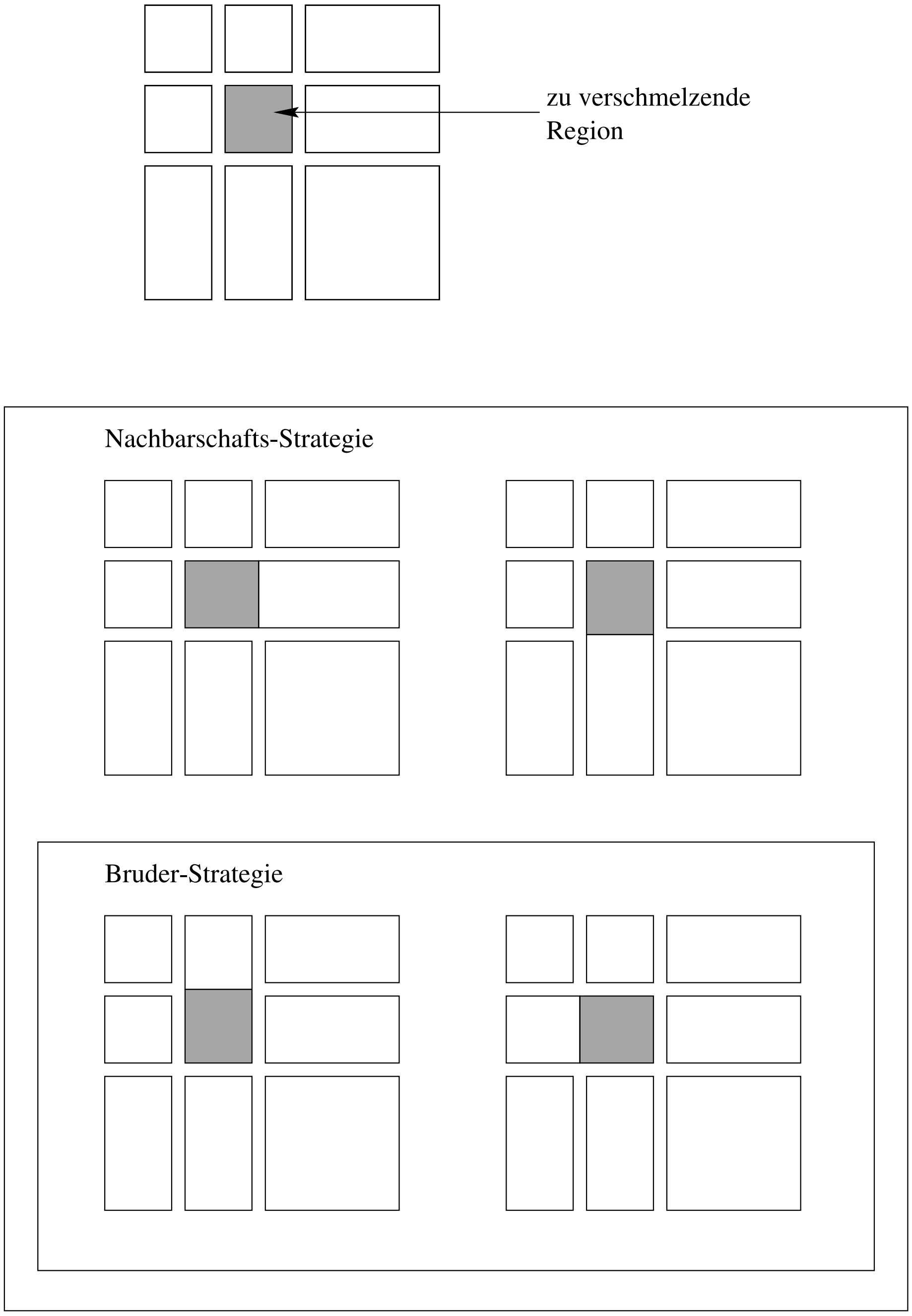


Abbildung 2.50: Verschmelzungsstrategien

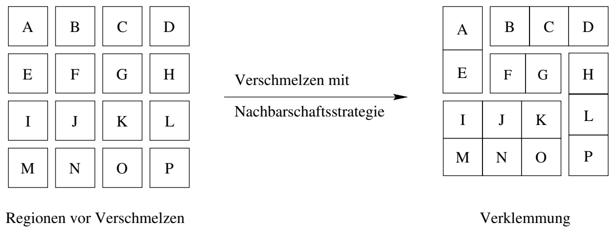


Abbildung 2.51: Verklemmung im $\mathbb{R}^2$

Nachbarn zur Auswahl. Da auch bei der Bruderstrategie nur rechteckige Bereiche entstehen können, stellt die Menge der Verschmelzungsmöglichkeiten, wie in Abbildung 2.50 dargestellt, eine Teilmenge der Verschmelzungsmöglichkeiten dar, die sich aus der Nachbarschaftsstrategie ergeben.

Die Bruderstrategie kann im zweidimensionalen Fall nicht verklemmen, aber schon bei drei Dimensionen ist auch hier wieder eine Verklemmung möglich, wie in Abbildung 2.52 zu sehen ist. Da eine Region in jeder Dimension einen Bruder

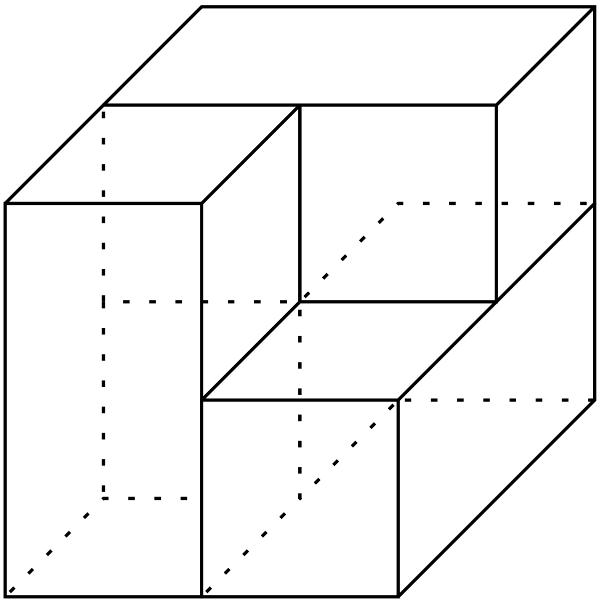


Abbildung 2.52: Verklemmung im $\mathbb{R}^3$

haben kann, ist es manchmal sinnvoll, unmittelbar nach der Aufteilung einer Region in zwei neue Regionen die Möglichkeit der Verschmelzung, gewissermaßen mit dem anderen Bruder zu überprüfen.

Verhalten von Gridfiles Eine Analyse des durchschnittlichen Verhaltens des Gridfiles ist sehr schwierig. Mit Simulationen wurde gezeigt, daß die durchschnittliche Auslastung von Datenblöcken meist bei etwa 70 % (ungefähr ln 2) liegt. Dieser Wert ist typisch für Strukturen, die mit rekursivem Halbieren arbeiten [FNPS79] und [Reg85]. Bei einer Datenblockkapazität b wächst das Directory des Gridfiles bei n gleichverteilten Datensätzen mit $O(n^{1+1/b})$ ebenso, wie eindimensionales erweiterbares Hashing. Bei einer ungünstigen Verteilung der Datensätze, im zweidimensionalen Fall etwa entlang einer Diagonalen, wächst das Directory sogar mit $O(n^d)$ für ein d-dimensionales Gridfile. Trotz dieses relativ schlechten Worst Case ist das Gridfile eine für viele Anwendungen geeignete mehrdimensionale Datenstruktur. Eine ausführliche Darstellung von Simulationen findet man in [NHS84].

2.4.4 Das Closest Pair Problem – CPP

In diesem Kapitel sollen zwei Variationen des Closest Pair Problems als Vertreter der Distanzprobleme vorgestellt werden. Auch hierbei werden wir uns wieder auf die *euklidische Ebene* beschränken. Wir werden es also mit einem zweidimensionalen Raum $\mathbb{R}^2$, den Punkten $p_1 = (x_1, y_1)$ und $p_2 = (x_2, y_2)$, sowie der *Distanzfunktion* $d(p_1, p_2) := \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ zu tun haben. Dies bedeutet, daß wir die Distanz zwischen zwei Punkten als Länge ihrer geradlinigen Verbindung betrachten (in unserem Fall könnte man auch von Luftlinie sprechen).

Problem: *dichtestes Punktpaar (Closest Pair)*

gegeben: Eine Menge P von n Punkten in der Ebene.

gesucht: Ein Paar (p_1, p_2) von Punkten aus P mit minimaler Distanz.

Eine Variation zum CPP soll den Bezug zu unserem Projekt verdeutlichen: Ein MSD-Kunde sucht ein Betreuungsangebot eines gewissen Typs (z.B. Wohnung putzen) zu einer grob festgelegten Zeit (z.B. Dienstag oder Mittwoch Nachmittag). Er hat Idealvorstellungen in vielerlei Hinsicht (Dauer, wer soll kommen, wieviele sollen kommen, Rhythmus), die er insgesamt so gut es geht realisiert haben möchte. Das DRK wird versuchen, aus der Grundmenge der verfügbaren Betreuungsmöglichkeiten ein möglichst gut passendes Angebot zu machen (*best match*). Man kann nun die Attribute einer Betreuung als Koordinaten in einem mehrdimensionalen Koordinatensystem auffassen und die Gewichtung der Attribute in einer Distanzfunktion zum Ausdruck bringen. Der Kunde sucht dann nach einer Betreuung mit möglichst geringer Distanz zu seiner Idealvorstellung.

Für den 2-dimensionalen Fall formulieren wir das Problem wie folgt:

Problem: *Suche nächsten Nachbarn* $nearestneighborsearch, bestmatch$

gegeben: Eine Menge P von n Punkten in der Ebene.

gesucht: Eine Datenstruktur und Algorithmen, die

1. P in der durch die Datenstruktur vorgeschriebenen Form speichern (preprocessing),
2. zu einem gegebenen , neuen Punkt q (Anfragepunkt, query point, i.d.R nicht aus P) einen Punkt aus P finden, der nächster Nachbar von q ist.

2.4.4.1 Berechnung des CPPs

Würde man naiv vorgehen, könnte man für jedes Punktpaar die Distanz berechnen und dann das Minimum der Distanzen ausfindig machen. Bei n Punkten gibt es $n(n-1)/2$ Punktpaare, womit uns das naive Verfahren $\Theta(n^2)$ Schritte kostet.

Da eine quadratische Laufzeit für reale Probleme nicht akzeptabel ist, müssen wir versuchen einen besseren Wert zu erreichen. Betrachten wir zunächst ein Verfahren, daß im Eindimensionalen gut funktioniert: Divide-and-Conquer. Das Verfahren soll zunächst im eindimensionalen Fall vorgestellt werde:

- sortiere alle Punkte
- teile (in der Mitte)
- suche in jedem Teil das ClosestPair
- Ergebnis ist das Minimum von Abstand((CPP(LinkeTeilmenge)), Abstand(CPP(RechteTeilmenge)) und Abstand(Maximum(LinkeTeilmenge), Minimum(RechteTeilmenge)).

Der letzte Abstandsvergleich sichert denn Fall ab, daß die Mengenteilung gerade durch das Closest Pair ging. Bei mehreren Dimensionen ist nun das Problem, daß die Punkte keiner totalen Ordnung unterliegen. Damit wird obiges Vorgehen unmöglich, da gerade die durch Trennung geteilten Closest Pairs nicht festzustellen sind (was ist das größte bzw. kleinste Element in der Ebene?). Wir könnten versuchen eine totale Ordnung herzustellen, in dem wir alle Punkte auf eine Koordinatenachse projizieren. Dies scheitert allerdings daran, daß wir entscheidende Informationen verlieren und so ein Closest Pair ein Farthest Pair werden kann. So sind in Abbildung 2.53 die Punkte P_1 und P_2 das Closest Pair. Nach der Projektion auf die y-Achse haben sie jedoch den maximalen Abstand.

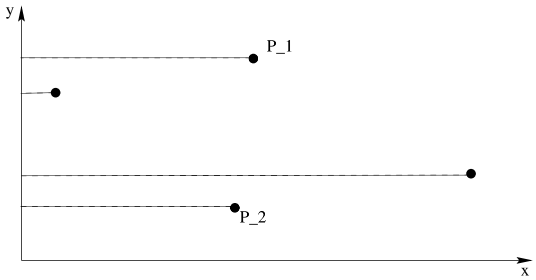


Abbildung 2.53: CPP $\rightarrow$ Farthest Pair

Wir müssen uns für das Zusammenfügen der Teilmengen also etwas anderes einfallen lassen. Doch zuerst zum *rekursiven Abstieg*: Teile die Mengen, bis sie nur noch zwei Punkte enthalten, dann hat man das CP der Teilmenge. Wir teilen die Menge mit einer vertikalen Schnittlinie l (Median(X)). Und lösen das Problem rekursiv für beide Teilmengen S_1 und S_2 (s. Abbildung 2.54). δ_1 ist Abstand(ClosestPair(S_1)). Analog ist δ_2 der Abstand des Nächsten Paares in S_2.

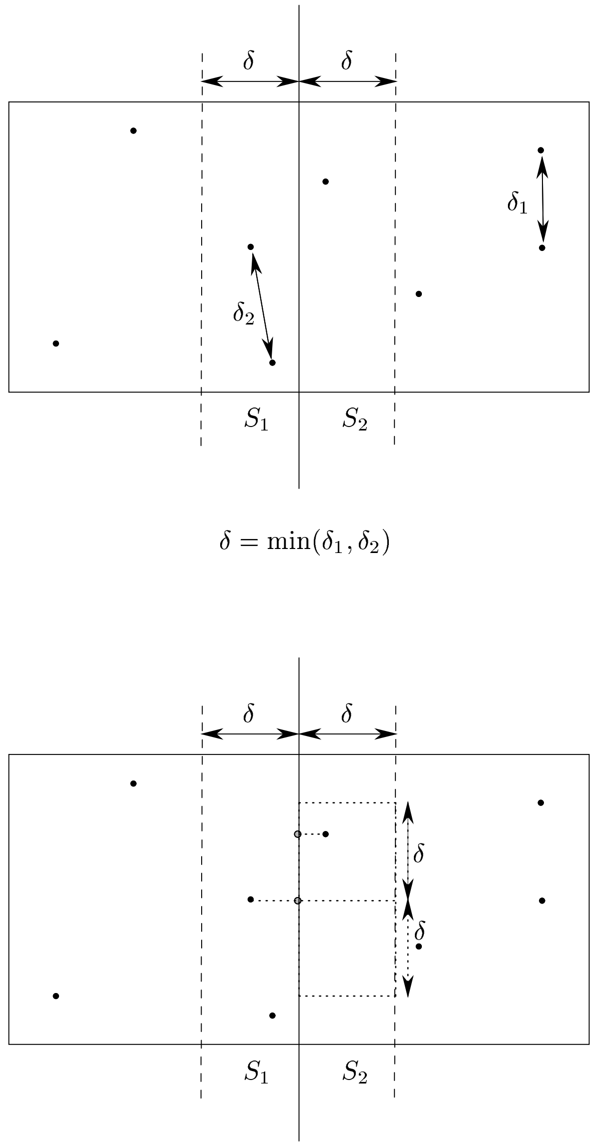


Abbildung 2.54: Merge beim CPP

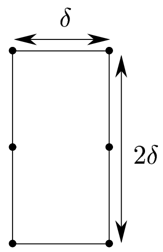


Abbildung 2.55: In einem $\delta \times 2\delta$-Rechteck können maximal sechs Punkte liegen.

Was jetzt noch fehlt ist die Prüfung, ob das Nächste Paar gerade durch die Schnittlinie getrennt wurde (d.h. in jeder Teilmenge ein Punkt des Closest Pairs liegt). Diesen Fall behandeln wir im *rekursiven Aufstieg* folgendermaßen: Sei $\delta = \min(\delta_1, \delta_2)$, (p, q), $p \in S_1, q \in S_2$ das CP, dann liegen sowohl p, als auch q offensichtlich innerhalb des Abstands δ von l (s. Abbildung 2.54). Wenn wir dies ausnützen, können wir uns darauf beschränken, das Closest Pair in einer δ-Umgebung von l zu suchen. Damit haben wir jedoch immer noch $n^2/4$ Abstandsvergleiche durchzuführen. Diesen Aufwand kann man weiter verringern, da nicht alle möglichen Punktpaare (p, q) der δ-Umgebung überprüft werden müssen. Wir wählen ein beliebiges $p \in S_1$ und müssen dazu nun nicht alle möglichen $q \in S_2$ testen, sondern nur diejenigen, die in der $\delta \times 2\delta$-Rechteck R liegen (s. Abbildung 2.55). Weiterhin können wir ausnützen, daß keine zwei Punkte in S_2 näher als δ beieinander liegen. Damit können in R maximal 6 Punkte liegen und damit müssen wir für alle Punkte $p \in S_1$ nur 6 und nicht $n/2$ Punkte aus S_2 testen. Damit erhalten wir für den Trennungsfall maximal $6 \cdot n/2 = 3n$ Abstandsvergleiche. Dies reduziert das Problem zwar schon erheblich, wir haben aber immer noch keinen $O(n \log n)$-Algorithmus, da wir nicht wissen welche 6 Punkte wir betrachten müssen. Um dies herauszufinden, projizieren wir nun p und alle Punkte aus S_2 auf l. Die 6 zu testenden Punkte liegen nun alle innerhalb des Abstands δ von der Projektion des Punktes p. Wenn alle Punkte nach der y-Koordinate sortiert vorliegen, kann für alle Punkte aus S_1 der mögliche Nächste Nachbar in einem Durchlauf der sortierten Liste gefunden werden. Dieses Vorgehen wird durch Algorithmus 3 implementiert.

Algorithmus 3 CPP nach [PS95]

Eingabe: $S \subseteq \mathbb{R}^2$ {zu durchsuchende Punktmenge, nach y-Koordinate sortiert}
Ausgabe: $(p_1, p_2) \in S$ {das Closest Pair}
δ {die minimale Distanz zwischen dem CPP}
1: Teile S mit Gerade l, welche durch den vertikalen Median von S verläuft, in 2 Teilmengen S_1 und S_2
2: $\delta_i \leftarrow \text{ClosestPair}(S_i)$ für $i \in \{1, 2\}$
3: $\delta \leftarrow \min(\delta_1, \delta_2)$
4: $P_1 \leftarrow \{p \mid l_x - \delta \leq p_x \leq l_x\}$
5: $P_2 \leftarrow \{p \mid l_x < p_x \leq l_x + \delta_x\}$
6: $\forall p \in P_i, i \in \{1, 2\}$: projiziere p_i auf l {die Projektion heiße $l(p_i)$}
7: $\forall p_1 \in P_1$: berechne minimale Distanz δ_p zu allen Punkten $p_2 \in P_2$, für die gilt: die Projektion von p_2 liegt im Intervall $[l(p_1) - \delta,\ l(p_2) + \delta]$
8: $\delta \leftarrow \min(\delta, \delta_p)$

2.4.4.2 Aufwandsabschätzung

Analog zu der Abschätzung des Aufbauaufwands eines idealen 2d-trees, wird zunächst der Aufwand für jede Rekursionsstufe angegeben:

- in Zeilen 1 und 7: Median-Berechnung und Merge $O(n)$
- in Zeilen 3 und 8: Minimum von zwei Zahlen O(konst.)
- in Zeile 2: Rekursion $2O(n/2)$ s.u.

- in Zeilen 4+5+6: wenn Menge vorsortiert (benötigt $O(n \log n)$: nur auslesen $O(n)$

Wegen der Teilung beim Median liegen in beiden Teilmengen jeweils weniger als die Hälfte der Punkte: $|T_i| \leq |T|/2;\ i = 1, 2.$

Der Aufwand kann analog dem des dd-Baum-Aufbaus abgeschätzt werden: Der Aufwand einer Rekursionsstufe (mit n Punkten) beträgt $A(n) = O(n) + 2A(n/2)$. Mit $\log n$ Rekursionsstufen ergibt sich daraus ein Gesamtaufwand von $O(n \log n)$.

Kapitel 3

Anforderungsanalyse

3.1 Einleitung

Das Ziel des Projektes Transportoptimierung ist die Erstellung eines Systems, das den Planer eines Fuhrparks bei seiner Arbeit unterstützt. Kunde ist der Fahrdienstleiter der mobilen Dienste beim Kreisverband Stuttgart des Deutschen Roten Kreuzes (DRK). Die Unterstützung besteht aus der

- Verwaltung der benötigten Daten
- Unterstützung bei der manuellen Routenplanung
- automatische Teilumlegung und Routenplanung
- Durchführung von Konsistenztests
- Ermittlung von Analysedaten

Der Schwerpunkt bei der Entwicklung liegt bei der Erstellung eines verwendbaren Systems. Das nachträgliche Verbessern der Algorithmen wird durch eine definierte Schnittstelle und durch modularen Aufbau unterstützt.

Als zukünftige Erweiterungsmöglichkeit ist an ein Modul zur Abrechnung gedacht, das automatisch für die erledigten Dienste Rechnungen erstellt.

3.2 Umgebung

Das System soll am Ende auf einem PC unter Windows95 laufen. Für die Ermittlung von Fahrzeiten zwischen zwei Orten, ohne die die automatische Planung unmöglich ist, wird ein noch nicht festgelegtes Verkehrstool verwendet. Die Entwicklung findet im wesentlichen auf einer SUN-SPARC-Station an der Fakultät Informatik statt. Zusätzlich ist noch ein PC, der dem Ziel-System ähnelt, vorhanden. Bezüglich der Programmiersprache und zu verwendender Programme wurden vom Kunden keine Anforderungen gestellt.

3.3 Aufbau des Systems

In diesem Abschnitt wird der grobe Aufbau des Systems kurz beschrieben. Die Gliederung muß jedoch nicht mit dem Aufbau des endgültigen Systems übereinstimmen.

3.3.1 Benutzungsschnittstelle

Das Programm soll dem Anwender eine graphische Benutzungsoberfläche anbieten, die eine Windows-übliche Bedienung erlaubt. Dem Benutzer werden Masken zur Eingabe, Abfrage und Veränderung der Daten angeboten. Zusätzlich zu diesen Masken wird die aktuelle Planung auch grafisch dargestellt. Mit Hilfe der aus dieser Darstellung gewonnenen Erkenntnisse können dann manuelle Modifikationen an den Tour- und Dienstplänen vorgenommen werden.

Um die Planungen auch den Mitarbeitern zugänglich zu machen, werden zwei Arten von Plänen auf dem Drucker ausgegeben: zum einen sind das die Dienstpläne, die die Dienste der Mitarbeiter enthalten, und zum anderen die Tourpläne, die die Stationen einer Tour auflisten.

3.3.2 Datenverwaltung

Die vom System zu verwaltenden Daten betreffen im wesentlichen die zu verplanenden Ressourcen Mitarbeiter und Fahrzeuge, die Kunden mit ihren Wünschen und die geplanten Touren. Für diese Daten wird zunächst ein Datenmodell erstellt, das die zu jedem Objekt gehörenden Attribute und die Beziehungen zwischen den Objekten beschreibt.

Obwohl die Daten vertraulich sind, muß dieses Problem innerhalb dieses Projektes nicht berücksichtigt werden, da der Auftraggeber der einzige Mensch mit Zugang zu dem System ist. Auch die Datensicherheit, d.h. der Schutz vor Zerstörung der Daten, wird hier nicht betrachtet.

3.3.3 Konsistenztests

Für die Daten werden Konsistenzbedingungen ermittelt und implementiert. Beispiele für solche Bedingungen sind

- höchstens so viele Passagiere wie Sitzplätze
- mindestens ein Fahrer und ein Fahrzeug pro Tour
- Jeder Mitarbeiter und jedes Fahrzeug kann zu einem Zeitpunkt höchstens eine Tour fahren
- verplante Mitarbeiter und Fahrzeuge sind auch vorhanden
- Ein Wunsch kann durch maximal eine Tour erledigt werden
- Ein Kunde steigt aus dem Fahrzeug aus, in das er eingestiegen ist

Es soll aber möglich sein, über mehrere inkonsistente Zwischenzustände wieder zu einem konsistenten Zustand zu gelangen, um Veränderungen am Plan nicht unnötig kompliziert oder gar unmöglich zu machen.

3.3.4 Automatische Tourplanung

Die automatische Tourplanung könnte mit genetischen Algorithmen oder mit Matching Verfahren geschehen. Zur Entwicklung können unter anderem das Programm GENOM [Pro96] für genetische Algorithmen, die Algorithmen- und Datenstrukturen-Bibliothek LEDA [Meh84] und Ergebnisse der Projektgruppe Fahrgemeinschaften [Pro97] verwendet werden. Dem Planer werden mehrere Algorithmen für unterschiedliche Aufgabenstellungen angeboten.

3.3.5 Ermittlung von Analysedaten

Aus den Daten über die geleisteten Einsätze werden Analysedaten abgeleitet. Dabei handelt es sich zum Beispiel um

- Anzahl der Personen pro Sitzplatz und Tour
- das Verhältnis von Einsatzzeit zu Gesamtzeit von Fahrzeugen und Mitarbeitern
- Anzahl und Länge der Leerfahrten
- gefahrene km und Zeit je Dienst
- Anzahl der Dienste pro Mitarbeiter

3.3.6 Anbindung an ein Verkehrstool

Die Anbindung an ein Verkehrstool wird benötigt, um die zeitliche Entfernung zwischen Orten zu ermitteln. Die Orte werden dabei durch Straßenname und Hausnummer angegeben. Dem Planer muß eine Möglichkeit gegeben werden, diese Zeiten an die realen Fahrtzeiten anzupassen.

3.4 Ist-Zustand beim Kunden

Als Grundlage der Anforderungsanalyse dienten zwei Sitzungen mit dem Kunden, Herrn Schroff. Dabei bekam die Projektgruppe einen Überblick über den status quo der Dienste des DRK in Bad Cannstatt sowie die Vorgehensweisen bei der (momentan) manuellen Planung.

3.4.1 Allgemeines

Das Zentrum Mobile Dienste des DRK Stuttgart beschäftigt circa 200 Zivildienstleistende (kurz Zivis) und etwa zehn Fachpflegekräfte, im Einsatz beim Kunden, sowie einige Verwaltungsangestellte, die mit der Organisation der Dienste beschäftigt sind. Für die Fahrten zum Kunden bzw. den Transport von Kunden stehen 70 Fahrzeuge zur Verfügung (verschiedene PKW, Kleinbusse und Rollstuhlbusse).

Die meisten Dienste werden von den Kunden bzw. deren Angehörigen oder anderen betreuenden Personen telephonisch bestellt. Die personenbezogenen Daten sowie die Dienstwünsche werden zunächst auf Papier erfaßt. Teile davon werden z.B. zur Rechnungsstellung oder zum Ausdruck von Tourplänen auch elektronisch gespeichert. Lediglich die Schulfahrdienste werden einmal jährlich von den Schulen zentral an verschiedene Fahrdienstanbieter vergeben.

	Schulfahrdienst	Behindertenfahrdienst			Essen auf Rädern		MSD, Pflegedienst
		Dialyse	Tagespflege	Individualfahrten	kalt	warm	
Touren	25	4	3	3000/Jahr	5	1	
Personen	140	8	20	1-2/Fahrt	175	30	240
Fahrtrhythmus	täglich an Schultagen	Mo-Mi-Fr	täglich	—	wöchentlich (Werktags)	täglich	täglich
Planung	jährlich	bei Veränderung		täglich	bei Veränderung		wöchentlich
Finanzierung	gefahrene km	km	pauschal	km	über Essenspreis		pauschal
Zu berücksichtigende Bedingungen							
Fahrzeugart	×	×	×	×	×		
Zeiten	×	×	×	×	×	×	×
Wohnort	×	×	×	×	×	×	×
Fahrtdauer	×	×	×				
Begleitperson	×	×	×	×			×
Mitarbeiterbezug	×		×		×		×

Tabelle 3.1: Übersicht über die mobilen Dienste des DRK

3.4.2 Dienste

Eine grobe Übersicht über wichtige Daten der verschiedenen Dienste bietet Tabelle 3.1. Eine detaillierte Besprechung der einzelnen Dienste geschieht in den folgenden Abschnitten.

3.4.2.1 Schulfahrdienst

An allen Schultagen werden behinderte Kinder an ihrem Wohnort abgeholt und in die Schule gefahren. Aus den körperlichen Behinderungen der Kinder ergeben sich Anforderungen an die Art des Fahrzeuges, das zur Beförderung eingesetzt wird. Es kommen PKW und Kleinbusse mit und ohne Rollstuhlplätze und Rampe bzw. Hebebühne zum Einsatz.

Mehrere Kinder werden mit einem Fahrzeug auf einer Tour zusammen befördert. Diese sollte möglichst immer von demselben Mitarbeiter gefahren werden, damit die Kinder eine ihnen bekannte Bezugsperson haben. Bei bestimmten Kindern (z.B. Epileptikern) muß zusätzlich zum Fahrer ein zweiter Mitarbeiter mitfahren, der sich gegebenenfalls um die Fahrgäste kümmern kann, ohne daß der Fahrer vom Verkehr abgelenkt wird. Nach Möglichkeit werden jedoch sowieso alle Touren mit zwei Mitarbeitern besetzt.

Die zu befördernden Schüler werden einmal jährlich zwischen verschiedenen Fahrdienstanbietern aufgeteilt. Das DRK macht somit seine Schulfahrpläne für das gesamte Schuljahr, neu zugezogene Kinder werden durch lokale Änderungen in die bestehenden Touren integriert. Die Anfangszeiten sind für jede Schule fest vorgegeben (aber nicht nach Klassen unterschieden), so daß sich für jeden Wochentag ein fester Tourplan ergibt, der in der Regel immer gleich abgefahren wird.

Allgemein werden die Touren so geplant, daß die Kinder nicht unnötig lange unterwegs sind (absolute Obergrenze sind zwei Stunden). Die Strecken von einem Kind zum nächsten werden zunächst mit Hilfe eines Routenplanungsprogramms abgeschätzt und ggf. nach der ersten Fahrt korrigiert. Zeiten für das Einsteigen der Kinder werden als Pauschalwerte mit eingerechnet. Daneben werden diverse persönliche Anforderungen hinsichtlich der Abholzeit berücksichtigt, die teilweise nur im persönlichen Gespräch der Fahrer mit den Eltern vereinbart wurden und nirgendwo festgehalten sind. Deshalb sollte jede Tour mindestens zwei Fahrern bekannt sein, die sich im Krankheitsfall oder während des Urlaubs gegenseitig vertreten können.

Zu jeder Schultour gibt es implizit eine Rückfahrt am Nachmittag, die die Kinder an der Schule abholt und i.d.R. in umgekehrter Reihenfolge der Abholung wieder nach Hause bringt. In Einzelfällen werden die Kinder an eine andere als die morgendliche Abholadresse gebracht, dies geschieht aber nur im Rahmen der Tour, falls der Umweg nicht allzu groß wird.

3.4.2.2 Individualfahrdienst

Hierunter fallen alle Fahrdienste, bei denen Kunden zu Hause abgeholt und zu einem Ort ihrer Wahl gebracht werden. Dabei müssen vor allem zwei Arten von Fahrten unterschieden werden:

- Regelmäßige Fahrten, z.B. von Patienten zur Dialyse oder von alten Menschen in eine Tagespflegeeinrichtung. Diese werden in Touren zusammengefaßt und über längere Zeiträume geplant.

  Es gibt einige feste Zielorte, zu denen jeweils mehrere Kunden gefahren werden. Die Uhrzeiten bleiben immer etwa gleich, besonders bei der Dialyse müssen diese recht genau eingehalten werden. Manche Kunden müssen nur auf der Hin- oder Rückfahrt mitgenommen werden, so daß sich die Zusammenstellung der Touren ändern kann.

  Die Fahrten zur Tagespflege werden parallel zu den Schulfahrten ausgeführt, meist kommen hier (aus Kostengründen) PKW zum Einsatz. Dialysefahrten mit Rollstuhlbussen werden in Verbindung mit dem Schulfahrdienst geplant und vor oder nach der Schultour mit demselben Fahrzeug gefahren.

- Individualfahrten, die einzeln bestellt werden und individuell verschieden sind (z.B. einzelne Fahrten zum Arzt oder auch in die Disco). In der Regel kommt hier ein Rollstuhlbus zum Einsatz (da körperlich mobile Kunden von Taxiunternehmen befördert werden), oft ist ein zweiter Mitarbeiter nötig, um den Fahrgast zu tragen.

  Je nach Dauer und Entfernung bleibt der Fahrer bis zur Rückfahrt am Zielort oder fährt zurück und steht in der Zwischenzeit für andere Aufgaben zur Verfügung.

3.4.2.3 Essen auf Rädern

Die Lieferung von Essen in die Wohnung des Kunden wird auf zwei verschiedene Arten ausgeführt:

- Kaltlieferung: An einem festen Tag der Woche wird tiefgefrorenes Essen für die ganze Woche ausgefahren, das die Kunden sich dann warmmachen können. Hierfür ist ein spezieller Kleinbus erforderlich, der Schienen für den Kühlcontainer eingebaut hat. Die Essenstouren sollten immer zur gleichen Uhrzeit gefahren werden, damit die Kunden wissen, wann sie zu Hause sein müssen. Auch hier sollte auf den Mitarbeiterbezug geachtet werden, es fährt also i.d.R. derselbe Mitarbeiter regelmäßig „seine“ Essenstour.

- Warmlieferung: Warmes Essen wird täglich in kleinen Isolierbehältern ausgefahren. Hierfür kann jedes Fahrzeug benutzt werden. Die Uhrzeit ist fest vorgegeben: Frühestens um 10:45 Uhr darf die Tour beginnen, damit das Essen um 12:30 Uhr (spätester Auslieferungszeitpunkt) noch ausreichend warm ist. Auch hier werden die Kunden in Touren zusammengefaßt, wobei eine hohe Fluktuation häufige Planungsänderungen nach sich zieht.

Von Kunden, die wegen ihrer Schwerhörigkeit die Klingel nicht hören oder wegen ihrer eingeschränkten Mobilität nicht selbst die Tür öffnen können, besitzt das DRK einen Haus- und Wohnungsschlüssel. Damit können die Mitarbeiter ohne Aufwand direkt in die Wohnung des Kunden gelangen, und dort das Essen abgeben.

3.4.2.4 Mobile soziale Dienste (MSD) / Ambulanter Pflegedienst (APD)

Für die verschiedenen sozialen Dienstleistungen vor Ort werden im Allgemeinen andere Mitarbeiter eingesetzt als für die Fahrdienste. Diese erledigen verschiedene Aufgaben am Wohnort des Kunden: Von Alltagshilfen wie Kehrwoche machen und Einkaufen gehen bis hin zu pflegerischer Versorgung bettlägriger Patienten. Hier bestimmt zunächst die Qualifikation eines Mitarbeiters, für welche Dienste er eingesetzt werden kann. Darüberhinaus ist hier der persönliche Bezug von Mitarbeiter zum Kunden wichtig, so daß ein Mitarbeiter regelmäßig zu denselben Kunden fährt.

Der Rhythmus dieser Dienste geht von täglich (für z.B. Pflege) bis hin zu mehrwöchentlich (z.B. Kehrwoche). Wie auch beim Essensfahrdienst müssen die Mitarbeiter für bestimmte Kunden den Wohnungsschlüssel mitnehmen, da diese nicht selbst die Tür öffnen können. Dies muß vor allem am Wochenende, wo die Dienststelle des DRK nicht besetzt ist, im voraus geplant werden.

3.4.3 Ressourcen

Bei der Planung von Touren, die die verschiedenen Dienstwünsche der Kunden des DRK erfüllen sollen, müssen im Wesentlichen zwei Arten von Ressourcen berücksichtigt werden, die hinsichtlich Eignung und Verfügbarkeit nicht beliebig verplant werden können:

- Mitarbeiter: Dies sind hauptsächlich Zivildienstleistende, die nur eine gewisse Zeit im Dienst des DRK stehen, aber auch festangestellte Fachpflegekräfte. Die Zivildienstleistenden werden i.A. entweder dem Fahrdienst oder den mobilen sozialen Diensten zugeordnet. Es gibt verschiedene Qualifikationen, die ein Mitarbeiter für gewisse Dienste haben muß, es kann also nicht jeder Mitarbeiter zu jedem Kunden geschickt werden.

  Urlaub und Fehlzeiten werden momentan nicht eingeplant, hier werden als Vertretung z.B. Beifahrer von Touren abgezogen, um den Fahrer einer anderen Tour zu vertreten.

- Fahrzeuge: Hier gibt es die drei Grobklassen PKW, Kleinbus und Rollstuhlbus (eine feinere Unterteilung muß für gewisse Dienste gemacht werden, ist aber momentan noch nicht schematisch realisiert). Je nach Art der Behinderung von Fahrgästen kommen nur gewisse Fahrzeuge für diese in Frage.

  Darüberhinaus kann jedes Fahrzeug mit verschiedenen Arten von Sitzen (Sitzkissen, Sitzschale, Individualsitz etc.) ausgestattet werden, so daß es nicht mehr beliebig für andere Fahrten nutzbar ist.

  Ausfallzeiten der Fahrzeuge für Inspektion oder TÜV werden nach Möglichkeit so geplant, daß sie in die Schulferien fallen, wo ausreichend Ersatzfahrzeuge zur Verfügung stehen, da keine Schultouren gefahren werden. Darüberhinaus fällt jedes Fahrzeug durchschnittlich etwa einmal pro Jahr unvorhergesehen aus, z.B. wegen eines Unfalls.

3.4.4 Planung

Die manuelle Planung der Fahrten, wie sie beim DRK Stuttgart momentan gemacht wird, unterteilt die Dienste in drei große Blöcke:

1. Zunächst werden die Schultouren geplant, da diese die meisten Fahrgäste umfassen, die Fahrten zur Tagespflege, die zeitlich mit den Schultouren zusammenfallen, sowie die Dialysefahrten, die medizinisch unabdingbar sind.
2. Essen auf Rädern, mobile soziale Dienste und ambulanter Pflegedienst werden zunächst getrennt als Touren geplant. Dann wird versucht, diese zeitlich so auf den Bereich 1 zu verteilen, daß Fahrzeuge und Mitarbeiter keine unnötigen Leerfahrten machen, sondern sich möglichst ein Dienst an den nächsten anschließt.
3. Die restlichen Touren, die nicht in Bereich 1 integriert werden konnten, sowie die sonstigen Individualfahrten werden in einem dritten Bereich gesammelt und gegebenenfalls untereinander optimiert.

Allgemein wird versucht, die Auslastung von Fahrzeugen und Mitarbeitern dadurch zu optimieren, daß anstelle einer Rückfahrt vom Ziel-/Einsatzort mit leerem Fahrzeug gleich eine weitere Tour mit diesem Fahrzeug unternommen wird. Dabei dürfen allerdings keine Dienste gemischt werden: Während einer Schultour darf z.B. kein Essen mitgenommen und ausgeliefert werden, wohl aber im Anschluß, wenn die Kinder bereits in der Schule ausgeladen wurden.

3.5 Erfahrungsbericht

Am 11. Dezember 1997 fuhren die Mitglieder der Projektgruppe jeweils bei einer Schultour und bei einer Essenstour mit. Dabei wurden folgende Erfahrungen gemacht, die die Anforderungen, die im Gespräch mit Herrn Schroff ermittelt wurden, ergänzen :

- Schul- und Essensfahrten haben feste Zeiten, dazwischen können jedoch Individualfahrten stattfinden.
- Von seiten der Zivildienstleistenden wurde der Wunsch nach einen festen Dienstplan geäußert , der über einen längeren Zeitraum gültig sein soll, da die Mitarbeiter oft erst morgens für die Touren eingeteilt werden, und es den Mitarbeitern dadurch schwerfällt, andere Aktivitäten einzuplanen (insbesondere im Bezug auf Wochenenddienste).
- Es existieren keine Pläne, wann sich welches Fahrzeug wo befindet bzw. befinden soll.
- Während der Essenstour mußten die Zivildienstleistenden des öfteren mit einem Hausschlüssel die Haustür öffnen, um das Essen entweder vor der Wohnungstür zu deponieren oder es direkt in die Wohnung des Kunden zu bringen.
- Die Beförderung von Schülern beschränkt sich nicht nur auf das Bringen und das Abholen zu/von der Schule, vielmehr stellt das DRK auch bei Klassenfahrten Fahrzeuge und Fahrer.

- Rollstühle werden nicht zwingend mitgenommen. Für diejenigen Schüler, deren Rollstuhl nicht mit zur Schule genommen wird, steht an der Schule ein passender Rollstuhl bereit.

- Individualfahrten können auch von Taxifahrten im eigentlichen Sinne abweichen, da sie auch längere Zeiträume einnehmen können (z.B. Diskobesuche, Konzertbesuche ...).

3.6 Notwendige Eingabedaten

Aus der Analyse des Ist-Zustandes beim DRK sowie weiteren Gesprächen mit dem Leiter des Fahrdienstes ergab sich eine Menge von Daten, die momentan vom DRK zu den verschiedenen Objekten der Planung (das sind sowohl Kunden und deren Dienstwünsche, als auch Mitarbeiter) erfaßt werden. Diese sind für eine vollständige Beschreibung der jeweiligen Objekte notwendig.

3.6.1 Kunden erfassen

Persönliche Daten Um einen Kunden zu erfassen, müssen seine persönlichen Daten bekannt sein. Hierzu zählen: Vorname, Nachname, Geburtsdatum, Adresse mit Straße, Hausnummer, Postleitzahl und Wohnort und falls es für den Dienst relevant sein sollte, ob ein Hausschlüssel beim DRK hinterlegt wurde.

Rechnungsempfänger Jedem Dienst muß ein Rechnungsempfänger zugeordnet werden. Hierbei kann es sich um Privatpersonen oder um Instutitionen (z.B Krankenkasse oder Sozialamt) handeln. Für die Erfassung der Daten sind wichtig: Name, Anschrift, ggf. Versicherungsnummer, Bankverbindung und die Zuordnung zu dem zu zahlenden Dienst.

Anforderungen Hier wird erfaßt welche Hilfsmittel der Kunde besitzt bzw. zur Durchführung des Dienstes benötigt. Unter Hilfsmittel versteht man i.a. Geräte, die den Kunden unterstützen bzw. helfen (z.B. Rollstuhl, Sitzkissen oder Einstiegshilfen). Weiter werden dem Kunden bestimmte Fahrzeugklassen zugeordnet, in denen er problemlos mitfahren kann.

Persönliche Abneigungen Eine Liste derjenigen Mitarbeiter, die aus irgendwelchen persönlichen Gründen vom Kunden nicht erwünscht sind. Aus der Menge aller Mitarbeiter kann hier eine beliebige Teilmenge ausgewählt werden. Diese Liste wird als Sperrliste gehandhabt, d.h. bei der Planung wird darauf geachtet, daß diese Mitarbeiter auf keinen Fall bei diesem Kunden eingesetzt werden.

Persönliche Vorlieben Eine Liste der Mitarbeiter, die der Kunde besonders bevorzugt. Unabhängig vom generellen Mitarbeiterbezug (ein Kunde wird i.d.R. immer von demselben Mitarbeiter betreut/gefahren/beliefert) können hier besondere Vorlieben erfaßt werden. Bei der Planung wird nach Möglichkeit einer der angegebenen Mitarbeiter zu diesem Kunden geschickt.

Angehörige und Kontaktpersonen Eine Liste von Personen, die kontaktiert werden sollen/können, wenn es während des Dienstes zu Zwischenfällen kommt. Das Verhältnis, in dem die Personen zum Kunden stehen, wird ebenfalls erfaßt (z.B. Mutter, Ehemann oder Hausarzt).

Bemerkungen Beliebiger Text, der für die Planung i.a. keine Rolle spielt, aber in den Tour- oder Dienstplan wörtlich übernommen werden kann. Hier können z.B. besondere Eigenschaften des Kunden eingetragen werden, auf die der Mitarbeiter zu achten hat.

Dienste Jedem Kunden können ein oder mehrere Dienste zugeordnet werden.

3.6.2 Dienst anfordern

3.6.2.1 MSD / Pflegedienst

Um für einen Kunden einen solchen Dienst anzufordern, sind folgende Daten notwendig:

Mitarbeiter, Qualifikation Zunächst muß erfaßt werden, ob zu diesem Dienst bei diesem Kunden ein zweiter Mitarbeiter zwingend erforderlich ist (z.B. um behinderte Kunden zu tragen). Die planmäßige Beteiligung von mehr als zwei Mitarbeitern an einem Dienst ist nicht vorgesehen.

An die notwendigen Mitarbeiter können dann verschiedene Forderungen hinsichtlich ihrer Qualifikation gestellt werden. Dazu werden benötigte Qualifikationen aus einer (vom Benutzer frei definier- und änderbaren) Liste gewählt. Diese können bei der Planung mit den tatsächlichen Qualifikationen der Mitarbeiter abgeglichen werden.

Persönliche Abneigungen und Vorlieben Diese sollten zum einen zur Information bei der Anforderung eines Dienstes angezeigt werden, zum anderen auch hier änderbar sein, so daß deswegen nicht extra zur Kundenerfassung verzweigt werden muß (siehe 3.6.1).

Termine Hier wird erfaßt, für wann, für wie oft, und für wie lange der Kunde den Dienst bestellen möchte. Jede Dienstanforderung kann beliebig viele Termine enthalten. Auf die Strukturierung dieser Termine wird in 3.6.2.5 eingegangen.

Bemerkungen Beliebiger Text, der zwar für die Planung keine Rolle spielt, aber in den Tourplan wörtlich übernommen wird und dort dem Mitarbeiter noch zusätzliche Details erklären kann.

3.6.2.2 Schulfahrt

Für die Anforderung einer Schulfahrt für ein behindertes Kind müssen folgende Informationen vorhanden sein:

Start- und Zielort Standardmäßig wird hier die Adresse des Kindes und die Adresse der Schule eingetragen, in Einzelfällen kann dies aber abweichen. Deshalb sollte beides frei wählbar sein.

Die Angabe, ob eine Rückfahrt mit denselben Orten gewünscht ist, kann aus den verschiedenen Terminen entnommen werden, wo ggf. eine Zeit für die Rückfahrt eingetragen wird.

Fahrzeugkonfiguration Genauso wie an Mitarbeiter, können auch an das eingesetzte Fahrzeug verschiedene Bedingungen gestellt werden. Vorgegeben werden kann hier eine Menge von benötigten Ausstattungsmerkmalen (wie z.B. eingebauter Kindersitz).

Mitarbeiter, Qualifikation Analog zum MSD (3.6.2.1) können hier notwendige Qualifikationen der eingesetzten Mitarbeiter festgehalten werden (z.B. muß bei anfallsgefährdeten Kindern ein zweiter Mitarbeiter mitfahren, der das Kind bei einem Anfall versorgen kann).

Persönliche Abneigungen und Vorlieben Analog zum MSD (3.6.2.1)

Termine siehe 3.6.2.5

Bemerkungen Auch hier können beliebige zusätzliche Bemerkungen angegeben werden, die für die Mitarbeiter im Tourenplan abgedruckt werden.

3.6.2.3 Individualfahrten

Für Individualfahrten, bei denen Kunden wie mit dem Taxi zu einem Ort ihrer Wahl gefahren werden, müssen dieselben Daten wie für die Schulfahrten erfaßt werden. Darüberhinaus kann verlangt werden, daß der/die Mitarbeiter zwischen Hin- und Rückfahrt mit dem Kunden am Zielort bleiben (z.B. weil er dort weitere Hilfe benötigt). Wird dies nicht gefordert, können die Mitarbeiter in der Zwischenzeit für beliebige andere Dienste eingesetzt werden, bevor sie die Rückfahrt erledigen.

3.6.2.4 Essen auf Rädern

Für die Lieferung von Essen in die Wohnung von Kunden (egal, ob warm oder kalt) müssen folgende Daten erfaßt werden:

Persönliche Abneigungen und Vorlieben Analog zum MSD (3.6.2.1)

Termine siehe 3.6.2.5

Essensart Für Kunden, die eine spezielle Diät einhalten müssen, kann hier eine spezielle Essensart ausgewählt werden (z.B. für Diabetiker geeignet oder vegetarisch). Außerdem wird angegeben, ob das Essen warm oder kalt geliefert wird.

Bemerkungen Auch hier können beliebige zusätzliche Bemerkungen anegegeben werden, die für die Mitarbeiter im Tourenplan abgedruckt werden.

3.6.2.5 Termine

Ein Termin enthält alle notwendigen Daten, um einmalige und regelmäßige Ereignisse der Arten Aufenthalt, Transport und Lieferung (s.u.) zeitlich eindeutig festzulegen. Die erforderlichen Angaben für einen Termin lassen sich in drei Gruppen unterteilen:

Zeitraum Eine Menge von Zeitspannen, innerhalb derer (ggf. eingeschränkt durch die Häufigkeit) das Ereignis stattfindet. Der Zeitraum wird durch zwei Angaben charakterisiert:

Zeitrahmen Ein umschließender Zeitraum (z.B. Schuljahr)

Ausnahmen Zeiten innerhalb des Zeitrahmens, in denen das Ereignis nicht stattfinden soll (z.B. Schulferien)

Beides kann sowohl ein längerer Zeitraum (wie Schuljahr und Ferien) sein, als auch eine Menge einzelner angegebener Tage (um ein Ereignis an exakt diesen Tagen zu beschreiben).

Häufigkeit Für regelmäßige Ereignisse, die sich innerhalb des angegebenen Zeitraums periodisch wiederholen, kann hier ein Wiederholungsrhythmus angegeben werden:

einmalig Genau einmal am angegebenen Tag

täglich Jeden Tag innerhalb des angegebenen Zeitraums

regelmäßig Hier können diejenigen Wochentage gewählt werden, an denen das Ereignis stattfinden soll. Zusätzlich kann angegeben werden, ob es nur an Werktagen stattfinden soll oder auch an Feiertagen, die auf einen Werktag fallen. Neben wöchentlicher Wiederholung kann ein beliebiger n-Wochen-Rhythmus angegeben werden.

Zeit Hier wird die Uhrzeit erfaßt, an denen das Ereignis an den angegebenen Tagen stattfindet. Abhängig von der Art des Ereignisses (und damit von der Art des Dienstes, für den ein Termin festgelegt wird) müssen verschiedene Information bereitgestellt werden:

Aufenthalt Hierunter fallen Dienste, bei denen Mitarbeiter zu einem Kunden fahren und dort eine gewisse Zeit verbringen, um für den Kunden oder mit ihm verschiedene Tätigkeiten zu verrichten. Nötige Angaben sind eine Zeit für den Beginn des Dienstes (oder eine Zeitspanne, um eine gewisse Toleranz zu erlauben), sowie die Dauer.

Transport Fahrten, bei denen der Kunde an einem Ort abgeholt und zu einem anderen Ort befördert wird, brauchen folgende Angaben: Einen Zeitpunkt für den Start und für die Ankunft (optional kann auch hier über eine Zeitspanne der frühestmögliche/späteste Abhol- bzw. Ankunftstermin angegeben werden). Ist eine Rückfahrt erwünscht, müssen dafür dieselben Daten erfaßt werden.

Lieferung Für eine Lieferung, bei der dem Kunden eine Ware in die Wohnung gebracht wird, muß die Zeit(spanne) für die Ankunft beim Kunden festgehalten werden.

3.6.3 Touren eingeben

Die verschiedenen Merkmale einer Tour können entweder manuell vorgegeben werden oder als Ergebnis einer automatischen Planung in geeigneter Weise auf dem Bildschirm und/oder Drucker präsentiert werden:

Tour-Nummer Jeder Tour ist eine eindeutige Nummer zugeordnet, die sich aus einer Zahl und einem Buchstaben zusammensetzt.

Mitarbeiter Konkrete Namen (+ evtl. Nummern) max. zweier Mitarbeiter, die die Tour im Regelfall fahren. Zur Information werden nach Auswahl eines Mitarbeiters dessen Qualifikationen angezeigt. Dadurch kann bei der manuellen Toureingabe durch optischen Abgleich mit den verlangten Qualifikationen einfach ein passender Mitarbeiter gefunden werden.

Fahrzeug Interne Fahrzeugnummer des Fahrzeugs, das im Regelfall für diese Tour eingesetzt wird. Damit ist implizit auch die Fahrzeugkonfiguration vorgegeben.

Wochentage Die Tage, an denen diese Tour regelmäßig stattfindet

Startort und -zeit Adresse für den Startort (i.d.R. wird dies die Zentrale des DRK Stuttgart sein), sowie die Abfahrtszeit.

Stationen Die verschiedenen Stationen der Tour mit Ankunftszeit und Abfahrtszeit. Die Aufenthaltsdauer ergibt sich aus der Dauer des Dienstes (im Falle von MSD) und einer Aufenthaltszeit (z.B. für das Einladen in einen Rollstuhlbus), die explizit angegeben werden kann. Die Fahrtzeiten werden zunächst mit dem zugrundeliegenden Verkehrstool ermittelt, sollen aber von Hand korrigiert werden können.

Aus der Liste mit Stationen können die benötigten Qualifiaktionen der Mitarbeiter als Vereinigungsmenge aller von Einzelkunden benötigten Qualifikationen ermittelt werden. Ebenso ergeben sich die persönlichen Abneigungen (hinsichtlich einzelner Mitarbeiter) der Tourkunden als Vereinigungsmenge der einzelnen Abneigungen, sowie die Vorlieben als Schnittmenge der einzelnen Vorlieben. Diese implizit in der Stationenliste enthaltenen Informationen werden hier zur einfachen Konsistenzprüfung für den Benutzer angezeigt.

Bei der Bildschirmmaske für die Tour sollten Möglichkeiten vorgesehen sein, um entweder die ganze Tour oder einzelne Stationen für die automatische Planung zu sperren, d.h. die gesperrten Komponenten bleiben fest mit dieser Tour verbunden und dürfen nirgendwo sonst eingeplant werden.

3.7 Szenarien

Szenarien sind Anwendungsbeispiele, die ausgehend vom Ist-Zustand einen möglichen Arbeitsablauf beim Benutzer beschreiben. Hierbei soll nicht das zu erstellende System beschrieben werden, sondern verschiedene denkbare Situationen durchgespielt werden, anhand derer später die Spezifiaktion und der Entwurf geprüft werden können.

3.7.1 Schulfahrten

3.7.1.1 Einleitung

Das Szenario stellt beispielhaft den Ablauf des Schulfahrdienstes des DRK Bad Cannstatt dar. Auf Szenarien der beiden anderen Fahrdienste (Dialysefahrten und Tagespflegefahrten) wurde verzichtet, da sich deren Ablauf nicht grundsätzlich von dem der Schulfahrten unterscheidet.

3.7.1.2 Szenario 1

Für dieses Szenario gilt die Annahme, daß die Schultouren schon fest eingeteilt sind.
Die Zivildienstleistenden Gustav Häberle und Karl Tremmle sind für die Schultour 21 eingeteilt. Auf dieser Tour müssen die beiden vier Schüler abholen, also können sie den sechs-sitzigen Ford Transit benutzen. Da einer der Schüler, Thomas Müller, ab und zu einen epileptischen Anfall bekommen kann, muß der Beifahrer Karl Tremmle im Umgang mit anfallsgefährdeten Kindern geschult sein. Dieses Wissen eignete sich Karl Tremmle zu Beginn seiner Dienstzeit auf einem Lehrgang an. Um die Schüler pünktlich um 8:30 Uhr an der Behindertenschule abliefern zu können, müssen die beiden Zivis um 7:45 Uhr mit ihrer Tour beginnen. Die Reihenfolge, in der sie die Kinder abholen wird durch den Tourplan bestimmt, der die einzelnen Stationen aufführt. Diese Reihenfolge sollte nicht geändert werden, da Susanne Schlecht keine langen Autofahrten verträgt und somit erst am Ende der Tour zusteigen kann. Als erstes auf der Liste steht Max Maier, er wohnt der Dienststelle am nächsten. Die zweite Station ist Karin Klein, die heute jedoch krank ist und erst wieder in der nächsten Woche abgeholt werden soll. Thomas Müller sitzt im Rollstuhl, den er aber nicht mit zur Schule nimmt, also müssen Gustav Häberle und Karl Tremmle diesen zu zweit in den Kleinbus heben. Zuletzt wird Susanne Schlecht abgeholt. An der Schule angekommen, müssen die beiden Zivis dafür sorgen, daß die Schüler in ihr Klassenzimmer gebracht, daß Thomas Müller, in den für ihn bereitgestellten Rollstuhl gesetzt wird und daß dem Klassenlehrer von Karin Klein über deren Krankheit Bescheid gesagt wird. Da alle Fahrzeuge der Fahrdienste den Schulhof nicht vor 8:35 Uhr, nachdem sich alle Schüler im Klassenzimmer befinden, verlassen dürfen und Gustav Häberle und Karl Tremmle anschließend keinen Dienst haben, gehen die beiden in der Schulkantine noch einen Kaffee trinken.Die Rückfahrt am Nachmittag von der Schule nach Hause ist im wesentlichen von der Hinfahrt nicht zu unterscheiden.

3.7.1.3 Szenario 2

Sofie Schüchtern ist am Anfang des Jahres mit ihren Eltern nach Stuttgart gezogen. Da sie taubstumm ist muß sie eine Sonderschule besuchen, da ihr Wohnort in den Zuständigkeitsbereich des DRK Bad Cannststatt fällt muß sie in eine schon bestehende Schultour aufgenommen werden. Da sich doeser Dienstwunsch nicht so einfach erledigen läßt beschließt der zuständige Planer sie für den Anfang von einem einzelnen Zivildienstleistenden in einem Pkw abholen und zur Schule bringen zu lassen. Ebenso auf der Rückfahrt von der Schule nach Hause. Nach einigem hin und her und durch mühevolles Umsetzen einzelner Schüler in andere Touren kann Sofie Schüchtern in Zukunft regulär mit dem Kleinbus von Tour 11 abgeholt und zur Schule gebracht werden.

3.7.2 MSD / Pflegedienst

3.7.2.1 Dienstanforderung

Bei der Verwaltung des DRK gehen mehrere Anrufe ein, die Mobile soziale Dienste bzw. ambulante Pflegedienste anfordern:

1. Herr Häberle ist aufgrund seines Alters nicht mehr mobil genug, um selbst einkaufen zu gehen und seine Kehrwoche zu erledigen. Deshalb möchte er diese Aufgaben vom DRK erledigen lassen. Nachdem die allgemeinen personenbezogenen Daten von H. Häberle erfaßt wurden, notiert der DRK-Mitarbeiter seine genauen Dienstwünsche:

   Jeden Mittwoch nachmittag soll ein DRK-Mitarbeiter zu H. Häberle kommen, den Einkaufszettel abholen, und die notwendigen Besorgungen erledigen. Die Kehrwoche muß alle drei Wochen am Freitag nachmittag gemacht werden. Fällt einer dieser Tage auf einen Feiertag, so soll der Dienst stattdessen am Werktag davor stattfinden.

   Der Mitarbieter beim DRK gibt eine Dienstanforderung MSD mit zwei Terminen ein: Mittwochs, Ankunft beim Kunden zwischen 14 und 17 Uhr, Dauer eine Stunde, sowie Freitags zwischen 15 und 19 Uhr, Dauer 30 Minuten. Bei beiden Terminen wird angemerkt, daß sie nur werktags stattfinden, aber gegebenenfalls an anderen Tagen erledigt werden müssen. Für den Dienst ist ein Mitarbeiter ausreichend.

2. Für Frau Sparwasser bestellt ihre Tochter eine Hilfe, um die alte Frau abends ins Bett zu bringen. Da diese auch mit Essen auf Rädern vom DRK versorgt wird, sind ihre persönlichen Daten bereits bekannt.

   Um Frau Sparwasser ins Bett zu bringen, muß sie von zwei Mitarbeitern getragen werden. Da sie Diabetikerin ist, benötigt sie abends eine Insulinspritze. Dies soll jeden Werktag vom DRK gemacht werden, da ihr Schwiegersohn zu spät nach Hause kommt. Während seines Urlaubs (vom 28.3. bis 13.4. sowie vom 1.8. bis 20.8.) ist er zu Hause, das DRK muß also niemanden schicken. Da sich Frau Sparwasser bereits an den Zivi gewöhnt hat, der ihr das Essen bringt, wäre es ihr recht, wenn dieser auch abends zu ihr kommt.

   Der Mitarbeiter des DRK fordert an: Einen ambulanten Pflegedienst, der Montags bis Freitags (aber nur an Werktagen) stattfindet, mit Ausnahme der Zeiten vom 28.3. bis 13.4. sowie vom 1.8. bis 20.8. Die Ankunft der Mitarbeiter bei Frau Sparwasser sollte zwischen 18 und 19 Uhr liegen, die Aufenthaltszeit wird 30 Minuten betragen. Es müssen zwei Mitarbeiter für diesen Dienst eingesetzt werden, die beide Lasten heben dürfen. Ein Mitarbeiter muß pflegerisch geschult sein, um die Spritze geben zu können. Nach Möglichkeit sollte der Zivi Rainer Kluge, der das Essen für Frau Sparwasser bringt, bei diesem Dienst dabei sein.

3.7.2.2 Dienstablauf

Der Zivildienstleistende Rainer Kluge und der Pfleger David Helfgott fahren mit einem PKW am Mittwoch nachmittag um 16:30 Uhr beim DRK zu ihrer regelmäßigen Tour los. Um 17 Uhr sind sie bei Herrn Häberle und erledigen dessen Einkäufe. Kurz vor 18 Uhr ist dieser Dienst erledigt und die Fahrt geht weiter zu Frau Sparwasser. Dort treffen die DRK-Mitarbeiter um 18:20 Uhr ein, tragen sie ins Bett und geben ihr die Insulinspritze. Gegen 18:45 Uhr beginnt die Rückfahrt zur DRK-Zentrale, wo um 19 Uhr noch das Fahrzeug abgestellt wird, bevor die Mitarbeiter nach Hause gehen.

3.7.3 Essen auf Rädern

3.7.3.1 Allgemeines

Beim Essen auf Rädern sind im wesentlichen Warm- und Kaltlieferungen zu unterscheiden. Warmlieferungen werden an den vom Kunden gewünschten Tagen zwischen 10:45 Uhr und 12:30 Uhr ausgeliefert. Der Kunde hat auf die Uhrzeit im Allgemeinen keinen Einfluß. Bei den Kaltlieferungen wird dem Kunden der Wochentag und die Uhrzeit, zu der er Essenslieferungen erhalten wird, mitgeteilt. Auch bei Kaltlieferungen hat der Kunde in der Regel keinen Einfluß auf die Vorgaben des DRK.

3.7.3.2 neue Kunden einfügen

Szenario 1 Herr Helmut Hohl, ruft beim DRK Stuttgart an und teilt mit, daß er montags, mittwochs und freitags Diätessen beziehen möchte. Außerdem soll die Lieferung soll nur an Werktagen erfolgen. Auf Nachfrage teilt er mit, daß er in der Süß-Straße 99 in Leckerhausen wohnt. Desweiteren teilt er mit, daß wegen des bissigen Hundes das Essen nur in den Topf hinter dem Gartentor zu legen ist.

Da Herr Hohl bisher kein DRK-Kunde ist, werden seine Stammdaten aufgenommen, der Essenswunsch wird an die zuständige Küche übermittelt und Herr Hohl wird in einen bestehenden Essenstourenplan aufgenommen.

Szenario 2 Herr Ikebana bestellt beim DRK per eMail Normalkost, welche er täglich erhalten möchte. Ferner wünscht er, daß diese auf einem Teller appetitlich angerichtet wird und daß das Geschirr vom Vortag gespült wird.

Herr Ikebana ist schon Kunde, so muß nur sein Essenswunsch an die entsprechende Küche weitergeleitet werden. Da bei ihm noch ein kurzer Dienst zu verrichten ist, wird Ikebana an das Ende einer bestehenden Essenstour angehängt. Der Dienst wird als ein Essen und eine MSD-Stunde verrechnet.

Szenario 3 Die Altenhilfe Stuttgart bestellt Essen für Herrn Kirk, welcher in seinem hohen Alter nicht mehr ganz bei Kräften ist und keine Verwandten mehr hat. Er benötigt alle 2 Wochen Montag bis Sonntag püriertes Essen, das ihm eingegeben werden muß. Zuvor muß er wegen seines hohen Gewichts von zwei Personen vom Bett an den Tisch gebracht werden. Nach dem Essen soll Herr Kirk noch eine Stunde betreut werden (Unterhaltung, evtl. Wohnung saubermachen) und danach wieder ins Bett gebracht werden. Da Kirk nicht selbst aufstehen kann, bekommt das DRK einen Hausschlüssel von seiner Wohnung.

Da Herr Kirk Neukunde ist, werden zunächst seine Stammdaten aufgenommen. Dabei wird notiert, daß das DRK den Hausschlüssel besitzt, und die Altenhilfe als Bezugsperson erfaßt. Weiterhin wird vermerkt, daß das Sozialamt Stuttgart die Rechnungen von Herrn Kirk bezahlt. Innerhalb der Tourplanung wird der Dienst bei Herrn Kirk als MSD-Dienst mit zwei Mitarbeitern eingeplant, zu dem ein Warmessen und der Hausschlüssel mitzubringen sind.

Szenario 4 Da die Frau von Heinz Becker erkrankt ist, sind bis auf weiteres (bis es von Herrn Becker wieder abbestellt wird) drei warme Normalkostessen (für die ganze Familie) zu liefern. Da Herr Becker nicht gerne von seinem gewohnten Tagesrhythmus abweicht, besteht er auf eine Lieferung gegen 12:00 Uhr, und er jedes Jahr eine größere Summe an das DRK spendet, wird versucht, ihn so in einen Tourenplan zu integrieren, daß seine Wunschzeit ermöglicht wird, auch wenn dafür ein Umweg in Kauf genommen werden muß.

3.7.3.3 Änderungen

Szenario 5 – Essensart Ernie sind die Zähne ausgegangen, deshalb bekommt er statt normaler Tiefkühlkost jetzt nur noch püriertes Diabetikeressen.

Diese Änderung wird der Küche mitgeteilt, eine Änderung in den Essenstouren resultiert aus dem Änderungswunsch jedoch nicht.

Szenario 6 – Ausfall Herr Hohl fährt übermorgen für 17 Tage an den Wolfgangsee. In dieser Zeit soll natürlich kein Essen geliefert werden.

Die Küche kann in diesen 17 Tagen ein Essen weniger vorbereiten und Herr Hohl wird vorübergehend aus dem Essenstourenplan gestrichen.

Szenario 7 – Ende Karl Ramseier ist tot ... Ab sofort keine Essen mehr!!!

Die Küche wird benachrichtigt, daß ab sofort ein Essen weniger benötigt wird. Die Station wird aus der Tour entfernt. Gegebenenfalls kann die Tour neu optimiert werden (falls z.B. die Anzahl der Kunden auf der Tour unter ein Minimum fällt, könnten diese auf andere Touren verteilt werden).

Szenario 8 – Unfall Nach der Auslieferung des zweiten Warmessens nimmt ein anderes Fahrzeug dem Zivi die Vorfahrt und es kommt zu einem Unfall. Der Zivi berichtet der Zentrale über Funk, daß er einen Unfall hatte. Des weiteren teilt er der Zentrale mit, daß er auf die Polizei warten muß und bisher erst zwei Essen ausgeliefert hat.

Der Einsatzleiter des DRK trägt daraufhin seiner Sekretärin auf, alle Kunden ausstehender Lieferungen anzurufen, um ihnen die Verzögerung mitzuteilen. Daraufhin versucht er, möglichst schnell einen Ersatzzivi mit Fahrzeug zu finden, der die Essenstour zu Ende fahren kann.

3.7.4 Individualfahrten

3.7.4.1 Benötigte Daten

Die benötigten Daten können über die „Standardeingabemaske“ erfaßt werden. Die Taxi- oder Individualfahrten werden im Gegensatz zu den regelmäßigen Fahrdiensten (wie z.B. Schulfahrtdienst, Dialysefahrten ...) als einmalige Dienstleistung angesehen.

3.7.4.2 Ablauf

Bei der Bestellung einer Individualfahrt müssen folgende Einschränkungen beachtet werden:

- ist ein passendes Fahrzeug zur erforderlichen Zeit vorhanden?

- ist ein Zivi mit den erforderlichen Qualifikationen zum benötigten Zeitpunkt frei?

Die Einhaltung dieser Beschränkungen sollte im wesentlichen kein Problem sein, da die Fahrten einen gewissen Vorlauf zwischen Bestellung und Durchführung haben sollten.

3.7.4.3 Fahrzeugbelegungspläne

Das DRK besitzt Fahrzeugbelegungspläne, in denen in graphischer Form festgehalten wird, wann welches Fahrzeug im Einsatz ist. Diese Fahrzeugbelegungspläne spielen vor allem in der manuellen Planung der Individualfahrten eine wesentliche Rolle, da sie dem Planer einen genauen Überblick geben und er somit flexibel auf die Bestellungen von Individualfahrten reagieren kann, da er sofort sieht, ob diese Fahrt durchgeführt werden kann oder abgelehnt werden muß.

3.7.4.4 Szenario 1

Frau Meier hat um 8:00 Uhr einen Arzttermin mit voraussichtlicher Dauer von 30min, da Frau Meier nur eine Spritze bekommt. Die Fahrzeit beträgt (einfach mit Einstieg) 45min. Frau Meier muß beim Treppensteigen geholfen werden, da sie auf Krücken geht, es sind jedoch keine weiteren Qualifikationen des Zivis erforderlich, ebenso kann Frau Meier in einem normalen Pkw befördert werden. Dieser Fahrtwunsch kann erfüllt werden, da Frau Meier kein besonderes Fahrzeug benötigt.

3.7.4.5 Szenario 2

Hans Müller ist querschnittsgelähmt und hat einen Klapprollstuhl. Er will zu einer Beerdigung (Beginn 14:00 Uhr) und danach zur Trauerfeier im Gemeindehaus, das nicht weit vom Friedhof entfernt ist. Von dort will er dann gegen 16:00 Uhr abgeholt werden. Die Fahrzeit beträgt (einfach, mit einsteigen): 20min.
Die Fahrt kann erfüllt werden: Die Hinfahrt kann entweder mit einem PKW oder einem Kleinbus durchgeführt werden. Falls ein Kleinbus eingesetzt wird, werden jedoch zwei Mitarbeiter zum Einsteigen benötigt. Weitere Qualifikationen sind nicht erforderlich. Für den Rücktransport können die oben genannten Fahrzeugklassen wieder eingesetzt werden, jedoch gelten weiterhin die Einschränkungen, daß zwei Mitarbeiter dabei sein müssen und der Klapprollstuhl in den Kofferraum der PKW paßt.

3.7.4.6 Szenario 3

Voraussetzungen wie in Szenario 2, jedoch besitzt Herr Müller diesmal einen Elektrorollstuhl. Die Fahrten können durchgeführt werden, wenn es Herrn Müller nichts ausmacht, etwas früher am Friedhof anzukommen und erst nach 16:30 Uhr abgeholt zu werden.

3.7.4.7 Szenario 4

Thomas möchte nachmittags seine Oma besuchen, besitzt aber einen Spezialrollstuhl und ist stark anfallsgefährdet. Dieser Fahrtwunsch muß abgelehnt werden, da an diesem Nachmittag alle Rollstuhlbusse mit Hebebühne ausgebucht sind und die Mitarbeiter mit der entsprechenden Qualifikation entweder im Einsatz oder krank sind.

3.7.5 Pläne

3.7.5.1 Dienstpläne

Der Mitarbeiter Zacharias Zivi kommt am Montag morgen in die Dienststelle und möchte nun etwas tun. Er nimmt sich seinen Dienstplan vor und sieht darauf, daß er zunächst die Tour 21a alleine mit Fahrzeug S-RK 110 und kurz vor 11 die Tour 33a fahren soll. Um 12:40 muß er dann Herrn Müller von der Wertstr. 12 zum Weg 12 im gleichen Fahrzeug befördern.

3.7.5.2 Tourpläne

Der Mitarbeiter Zacharias Zivi hat auf seinem Dienstplan gesehen, daß er Tour 21a mit Fahrzeug S-RK 110 alleine fahren soll. Er sieht nach, ob für diese Tour ein neuer Tourplan vorhanden ist. Ist dies nicht der Fall, geht er direkt in das angegebene Fahrzeug, wo der immer noch gültige Plan liegt. Auf dem Tourplan sieht er, wann er welche Orte anfahren muß, welche Kunden er dort jeweils ein- und ausladen muß und was er dabei jeweils zu beachten hat. Dazu gehört zum Beispiel die Art des benötigten Sitzes für einen Schüler oder ob der Essenskunde ein Vegetarier ist.

3.7.5.3 Planung

Der Planer hat einen neuen Auftrag an der Hand und möchte nachsehen, ob zu diesem Zeitpunkt ein Fahrzeug und Mitarbeiter zur Verfügung stehen. Dazu werden ihm Gantt-Diagramme angeboten. Diese stellen die Einsatzzeiten der gewählten Ressourcen als Balken dar.

3.7.5.4 Auslastung

Der Planer möchte sehen, wie stark seine Mitarbeiter und seine Fahrzeuge ausgelastet sind. Das kann er auch mit den Gantt-Diagrammen, die im vorigen Abschnitt schon erwähnt wurden, machen. Je mehr Einsätze und damit auch Balken es zu einer Ressource gibt, desto größer ist der Anteil der dunklen Flächen im Gantt-Diagramm.

Kapitel 4

Spezifikation

4.1 Gesamtsystem

4.1.1 Umgebung des Produktes

Bei dem zu entwickelnden System handelt es sich um ein fensterorientiertes Softwaresystem, daß sowohl unter dem Betriebssystem SUN SOLARIS als auch unter MS Windows95 lauffähig ist. Neben der Bildschirmausgabe werden auch gedruckte Listen und Pläne angeboten, außerdem werden Daten auf Festplatte gespeichert. Die Betriebssystemunabhängigkeit dieser Komponenten wird durch die Verwendung der Programmiersprache JAVA erreicht.

Ferner wird zur vollen Leistungsfähigkeit des Systems das Programm „Map&Guide" benötigt, welches die Berechnung von Entfernungen und kürzesten Wegen zwischen beliebigen Adressen im Raum Stuttgart übernimmt. Der Datenaustausch zwischen beiden Produkten findet über die zu „Map&Guide" gehörige Batch-Schnittstelle statt. Für den effizienten Zugriff werden die Ergebnisse von „Map&Guide" in einer Tabelle hinterlegt, welche auch von Hand gepflegt werden kann, falls „Map&Guide", das nur auf Windows läuft, nicht verfügbar ist. Außerdem wird durch die Anbindung mit der Tabelle die Austauschbarkeit des Verkehrstools gewährleistet.

4.1.2 Warum Java?

Wichtigstes Kriterium bei der Wahl der zu verwendenden Programmiersprache war die Verfügbarkeit sowohl für die UNIX-Systeme an der Universität als auch für den Windows95-PC beim DRK. Darüberhinaus durfte die Anschaffung eines Compilers sowie einer passenden Entwicklungsumgebung nur mit geringen Kosten verbunden sein.

In der engeren Wahl standen:

- C++ mit der Klassenbibliothek wxWindows (die eine Quellcodekompatible Programmierung der Benutzeroberfläche erlaubt)
- Das Smalltalk-Entwicklungssystem VisualWorks
- Java (momentan in der Version JDK 1.1)

Die Kandidaten wurden anhand verschiedener Kriterien charakterisiert und verglichen:

Portabilität C++/wxWindows läßt sich als Quellcode auf UNIX und Windows compilieren. Smalltalk und Java werden in einen hardwareunabhängigen Bytecode übersetzt.

Kosten Die Compiler für C++ und Java sind kostenlos erhältlich, für VisualWorks sind einige Lizenzen am Institut vorhanden.

Entwicklungsumgebung VisualWorks ist eine integrierte Entwicklungsumgebung, die insbesondere auch einen passenden GUI-Builder enthält. Für wxWindows gibt es einen GUI-Builder als Freeware. Für Java gibt es eine Vielzahl von Entwicklungsumgebungen, davon auch einige Freeware-Programme.

C++-Anbindung Da möglicherweise Algorithmen zum Verwalten und Durchsuchen von Graphen aus der in C++ geschriebenen Bibliothek LEDA verwendet werden sollen, wäre eine Einbindungsmöglichkeit wünschenswert.

In C++ geht dies natürlich. Sowohl Java als auch Smalltalk bieten eine Schnittstelle, um Funktionen aus C-Bibliotheken aufzurufen. Möglicherweise gibt es auch entsprechende in Java geschrieben Bibliotheken.

Bibliotheken für Standardfunktionen Damit die Projektgruppe das Rad nicht zweimal erfinden muß, sollten für Standardfunktionen (z.B. Verwaltung von Listen, vielleicht auch Zeichnen von Graphen) möglichst kostenlose Bibliotheken existieren.

VisualWorks enthält bereits eine sehr umfangreiche Klassenbibliothek. Weitergehende Bibliotheken dürften kostenlos kaum zu bekommen sein. Zu C++ existiert die Bibliothek LEDA, die einen Teil dieser Funktionen abdeckt. Die meisten frei verfügbaren Bibliotheken oder Problemlösungen gibt es vermutlich für das vor allem im Internet stark verbreitete Java.

Die Entscheidung der Projektgruppe fiel auf die Sprache Java. Dies ist eine leicht zu erlernende objekt-hybride Programmiersprache. Sie kommt im Gegensatz zu C++ ohne Zeiger aus, was eine Reihe schwierig zu entdeckender Fehler von vornherein ausschließt. Die Programmierung einer graphischen Benutzungsoberfläche ist Bestandteil der Sprachdefinition. Da seit Erfindung dieser Sprache ein regelrechter Java-Boom herrscht, existiert im Internet eine große Zahl frei verfügbarer Bibliotheken für Java, die oft benötigte Funktionen bereits fertig zur Verfügung stellen.

Das Dokumentieren wird unterstützt durch spezielle Java-Kommentare im Quellcode, aus denen automatisch ein Teil der Dokumentation erzeugt werden kann. Außerdem besitzen mehrere Mitglieder der Projektgruppe bereits Erfahrungen in der Java-Programmierung, so daß weniger Einarbeitungsaufwand erforderlich ist und Probleme bei der Programmierung bereits innerhalb der Projektgruppe behoben werden können.

4.1.3 Verwendete Tools

Das System wird, wie bereits in den Kapiteln 4.1.1 und 4.1.2 erwähnt, unter Verwendung des „Java Development Kit„ JDK 1.1 auf SUN SOLARIS entwickelt.

Zusätzlich werden noch folgende Tools im Projektverlauf bzw. im laufenden System eingesetzt:

CVS Zur Versionsverwaltung aller Dokumente einschließlich der Quellcodes. Zuständig für CVS-Fragen ist Frank Wagner.

ObjectTeam ist ein CASE-Tool. Es wird zum Entwurf des Datenmodells und der Klassen eingesetzt. Für die Verwendung von ObjectTeam ist Jörg Fleischmann zuständig.

MS-Project wird zur Planung und Beobachtung des Projektverlaufs benutzt. Da die Aufsicht über den Projektverlauf bei Lars Hermes liegt, fällt auch MS-Project in seine Zuständigkeit.

Map&Guide ist ein Verkehrstool mit einer Batch-Schnittstelle. Wir verwenden „Map&Guide“ wie in 4.1.1 erwähnt im wesentlichen zur Entfernungsbestimmung zwischen Adressen. Für die Anbindung von „Map&Guide“ an das System ist Frank Wagner verantwortlich.

4.1.4 Funktionalität

Das System gliedert sich grob in die vier Bereiche Datenverwaltung, Planung, Optimierung und Analyse.

4.1.4.1 Datenverwaltung

Mit dem System können die Stammdaten für Kunden, Mitarbeiter und Fahrzeuge verwaltet werden. Dazu ist es möglich benötigte Daten wie Adressen, Bankverbindungen, Kommunikationsverbindungen, Fahrzeugausstattungen etc. einzugeben und getrennt nach Klassen zu speichern. Zur Datenspeicherung wird auf eine Datenbank verzichtet, da es im Rahmen des Projektes Transportoptimierung zu aufwendig wäre, eine Datenbankanbindung an das System zu schreiben. Die Schnittstelle an die Datenhaltung soll jedoch so offen gehalten werden, daß eine spätere Anbindung an eine Datenbank möglich ist, um die Daten persistent zu speichern, falls sich die Gegebenheiten ändern. Neben den Stammdaten beinhaltet die Datenverwaltung auch Dienstwünsche, geplante (Unter-)Touren, sowie eine Historie der erledigten Fahrten zur späteren Abrechnung, oder für Analysezwecke.

Die Datenerfassung wird in Kapitel 4.7 beschrieben.

4.1.4.2 Planung

Die Planung von Touren, Untertouren, Fahrten und Dienstplänen kann sowohl manuell als auch automatisch erfolgen, wobei die automatische Planung an jeder Stelle durch manuelle Änderungen an spezielle Bedürfnisse angepaßt werden kann.

Die automatische Planung verläuft in mehreren Schritten:

1. Aus den Dienstwünschen werden Touren zusammengestellt.

2. Touren werden zu Untertouren konkretisiert.

3. Aus Untertouren werden konkret stattfindende Fahrten abgeleitet.

4. Aus Fahrten werden Dienstpläne erstellt.

Dabei ist zu beachten, daß eine komplette Planung selten durchgeführt wird. In der Regel werden Dienstwünsche oder (Unter-)Touren in das bestehende Tourensystem eingefügt. Dies bedeutet, daß die verschiedenen Planungsschritte auch einzeln angestoßen werden können. Ferner ist es möglich ganze (Unter-)Touren oder einzelne Dienstwünsche von der weiteren Optimierung auszunehmen. Ausgenommene Teile bleiben also unbeschadet nach der Optimierungen bestehen.

Wozu dient die Unterscheidung von Tour, Untertour und Fahrt? Eine Tour faßt zunächst Dienstwünsche zusammen, die grundsätzlich miteinander bzw. nacheinander erledigt werden können, sagt jedoch noch nichts über die Reihenfolge der Dienstwünsche (und damit der Stationen) oder die Abfahrtszeit aus. Dies ist nötig, da Kunden unabhängig von der Uhrzeit oder ob es sich um ein Hin- oder Rückfahrt handelt, immer in gleichen Gruppen befördert werden sollen. Außerdem wissen die Mitarbeiter anhand der Tournummer schon, welche Kunden sie abholen müssen.

Mit der Untertour wird dann die Dienstwunsch- bzw. Stationenreihenfolge und die Uhrzeit festgelegt, zu der eine Tourinstanz gefahren werden soll. Damit ist es also möglich, Hin- und Rückfahrten zu gestalten. Außerdem kann mit einer Untertour dargestellt werden, daß die Gruppe nicht unbedingt vollständig befördert werden muß, fehlt eine Person (regelmäßig), ändert das nur etwas an der Untertour, die Personengruppe der Tour bleibt jedoch bestehen.

Die Fahrt wird schließlich benötigt, um konkret gefahrenen Tourinstanzen darzustellen. Bei den Fahrten kann auch, da sie ein festes Datum haben, überprüft werden, ob die zugeordneten Mitarbeiter und das zugordnete Fahrzeug überhaupt verfügbar sind, und ob alle Kunden deren Dienstwünsche zur Erledigung anstehen die Ausführung wünschen. Ist beispielsweise Kunde X im Urlaub wird dessen Dienstwunsch nicht in der Fahrt berücksichtigt, ohne daß dies weitere Auswirkungen auf die bestehende (Unter-)Touren hat.

4.1.4.3 Optimierung

Bei dem System gibt es zwei Optimierungsansätze: die *lokale Optimierung* 4.5 zur Optimierung innerhalb einer Dienstart und die *inkrementelle Optimierung* 4.6 zum Einfügen neuer Dienstwünsche und (Unter-)Touren in ein bestehendes Tourensystem, sowie zur Optimierung einer gegebenen Menge von Touren. Auf eine *globale Optimierung* wurde verzichtet, da diese durch gezieltes Anwenden der lokalen Optimierung auf alle Dienstarten erreicht werden kann. Dem Anwender bleibt damit die Möglichkeit offen, jeweils neu festzulegen, welche Dienstart bei seiner aktuellen globalen Optimierung welche Priorität haben soll.

4.1.4.4 Analyse und Datenausgabe

Die Analyse und Datenausgabe dient als Grundlage zur Dienst- und Infrastrukturplanung, sowie als Überblick über den aktuellen Planungsstand.

Die Datenausgabe wird Listen zu den Stammdaten, sowie den Druck von Dienstpländen beinhalten. Zudem ermitteln die Analysefunktionen des Systems die Auslastung von Mitarbeitern und Fahrzeugen, so daß freie oder fehlende Kapazitäten aufgedeckt werden können. Eine genaue Beschreibung der Analysefunktionen können dem Kapitel 4.4.2 entnommen werden.

4.1.5 Entwicklungsphilosophie

Bei der Entwicklung des Systems wird hierarchisch vorgegangen. Dies bedeutet, daß wir einzelne Module in der Reihenfolge ihrer Wichtigkeit immer so implementieren, daß das System lauffähig ist. Dadurch können die einzelnen Module schon frühzeitig getestet werden, außerdem haben wir immer ein einsatzfähiges System, dessen Funktionalität mit der Zeit zunimmt. Wir werden zunächst die Dateneingabe (incl. manueller Tourplanung) und Datenverwaltung implementieren, anschließend Konstistenztests für die manuelle Eingabe hinzufügen und am Ende das System durch eine automatische Planung abrunden.

Durch den modularen Aufbau kann das System auch leicht an neue oder veränderte Gegebenheiten angepaßt werden. Als mögliche Erweiterungen wären ein Abrechnungsmodul zur automatischen Rechnungsstellung erledigter Dienstwünsche, sowie ein Personalplanungsmodul zur automatischen stundengenauen Abrechnung der Arbeitszeiten vorstellbar.

4.2 Datenmodell

Da wir zur Programmierung die objektorientierte Programmiersprache JAVA verwenden, liegt es nahe, auch zur Erstellung des Datenmodells eine objektorientierte Methode heranzuziehen. Wir verwenden daher die *objektorientierte Analyse (OOA)*, die beispielsweise in [Bal96] beschrieben wird, und benutzen dazu das CASE-Tool ObjectTeam.

Da das Datenmodell schon unsere Klassen beinhaltet, stellt dieses Kapitel nur eine Momentaufnahme des aktuellen Standes dar. Die vollständige Spezifikation findet erst in der Entwurfsphase statt, in der noch die Datentypen, Wertebereiche, etc. festgelegt werden.

4.2.1 Klassendiagramme

Dieses Kapitel enthält die Klassendiagramme des Datenmodells. Diese Klassen dienen als Grundlage für die anschließende Entwurfsphase, werden dort jedoch nicht unbedingt eins-zu-eins in Java-Klassen umgesetzt.

Abbildung 4.1 stellt eine Kurzübersicht über die OMT-Notation dar, soweit sie in den nachfolgenden Diagrammen verwendet wurde. An dieser Stelle möchten wir uns bei Stefan Krauß aus der Abteilung Software-Engineering der Universität Stuttgart bedanken, der uns für diese Referenz seine in mühevoller Kleinarbeit erstellten xfig-Bilder zur Verfügung gestellt hat.

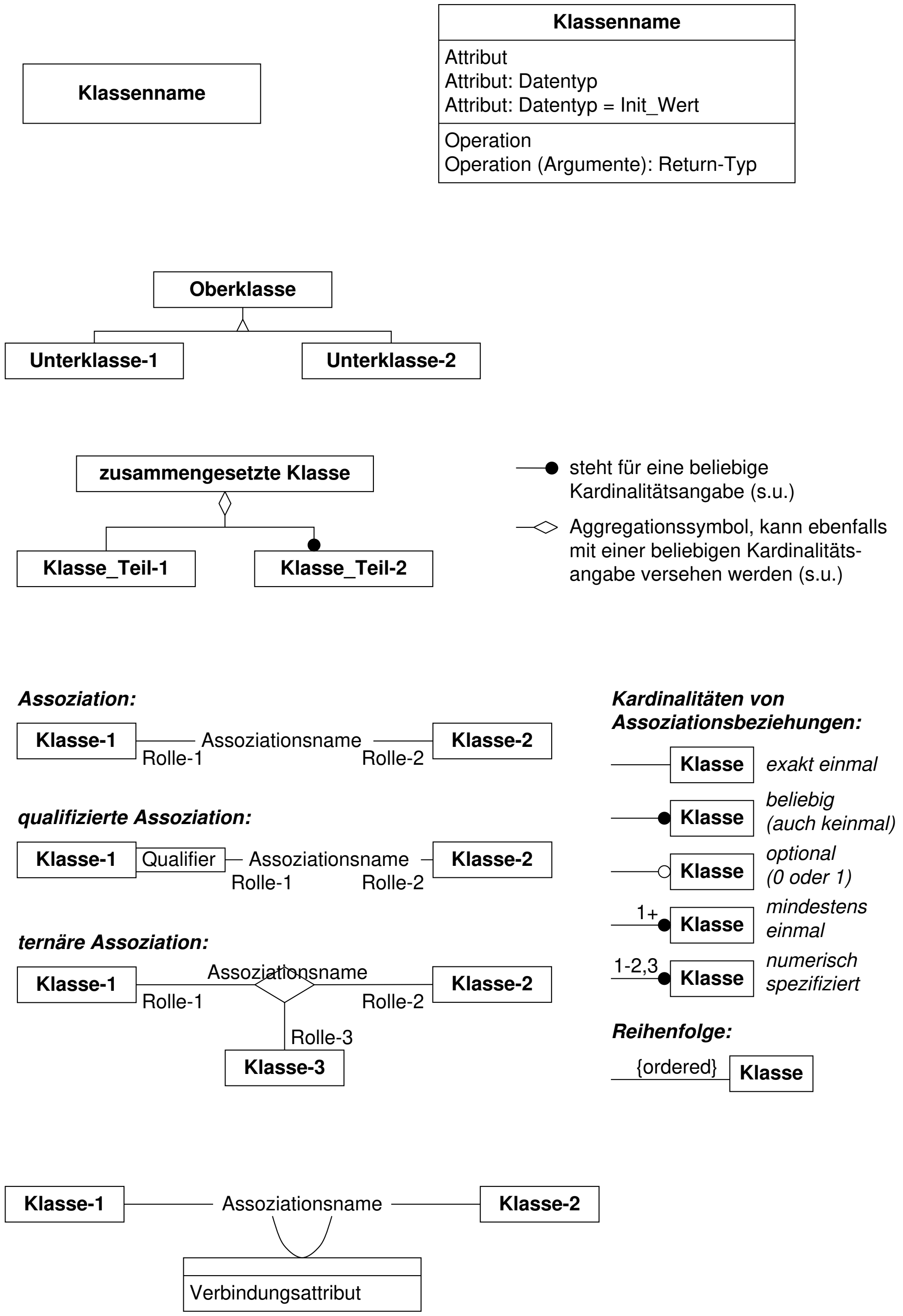


Abbildung 4.1: die OMT-Notation

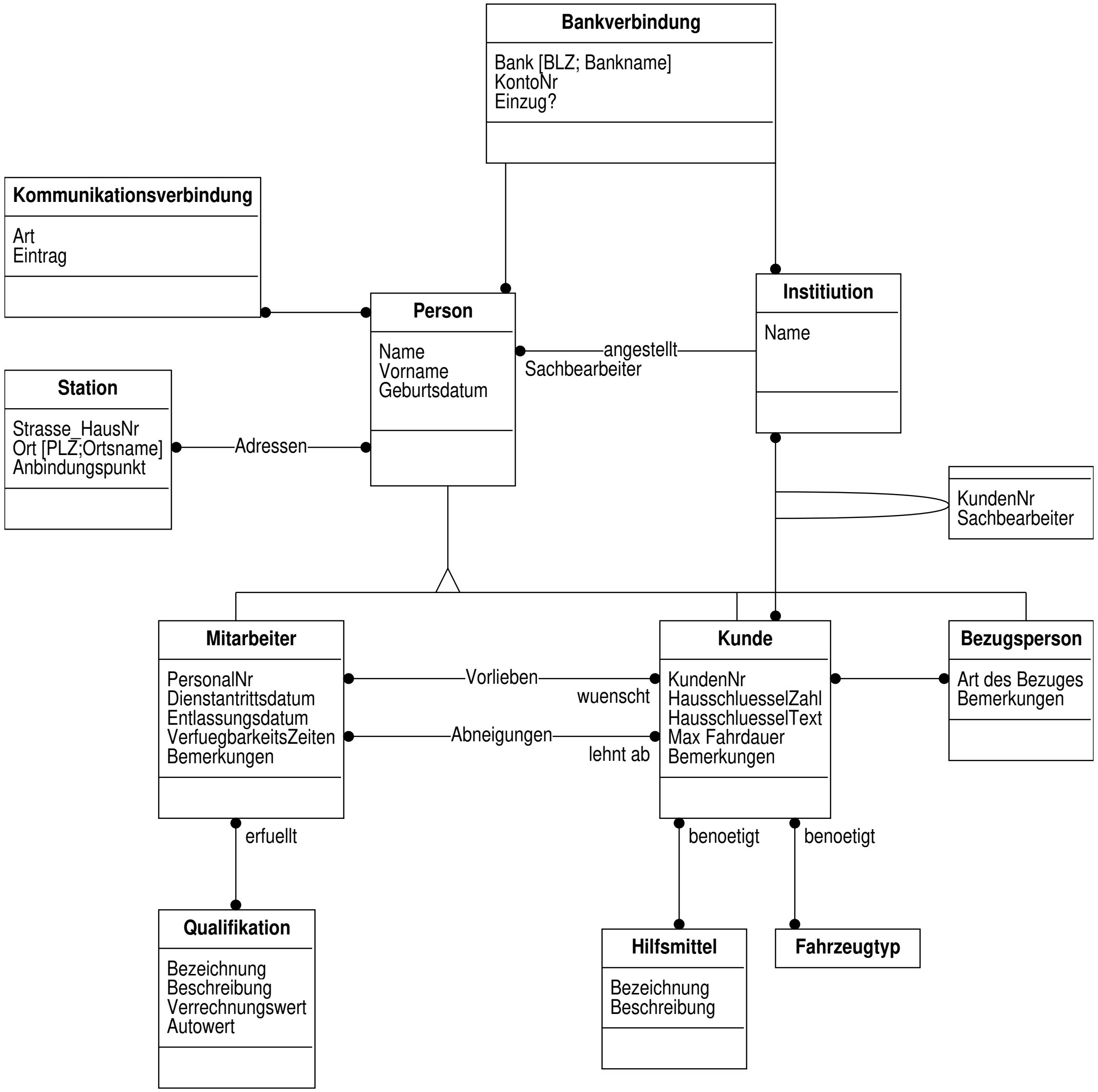


Abbildung 4.2: Personen

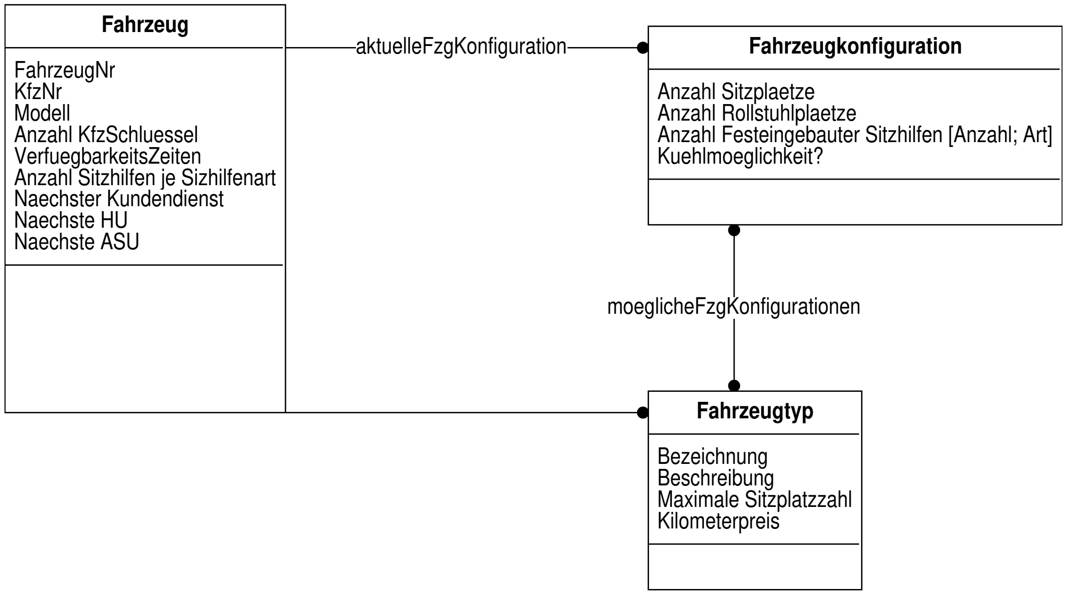


Abbildung 4.3: Fahrzeuge

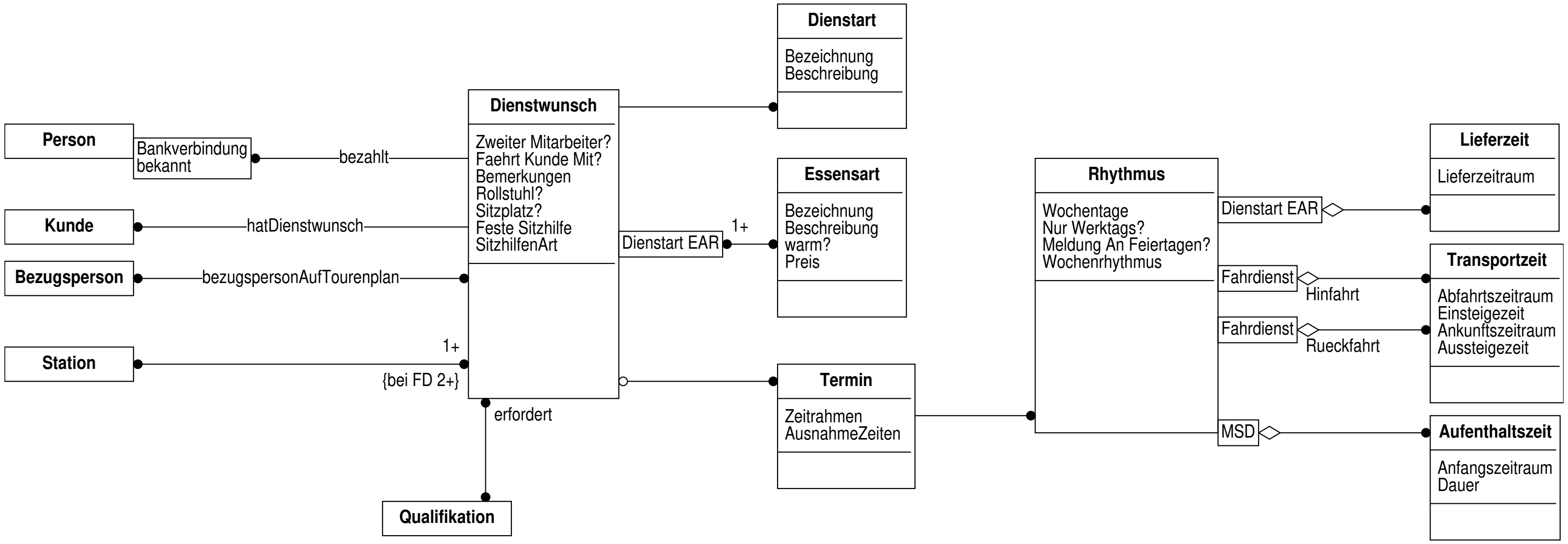


Abbildung 4.4: Dienstwünsche

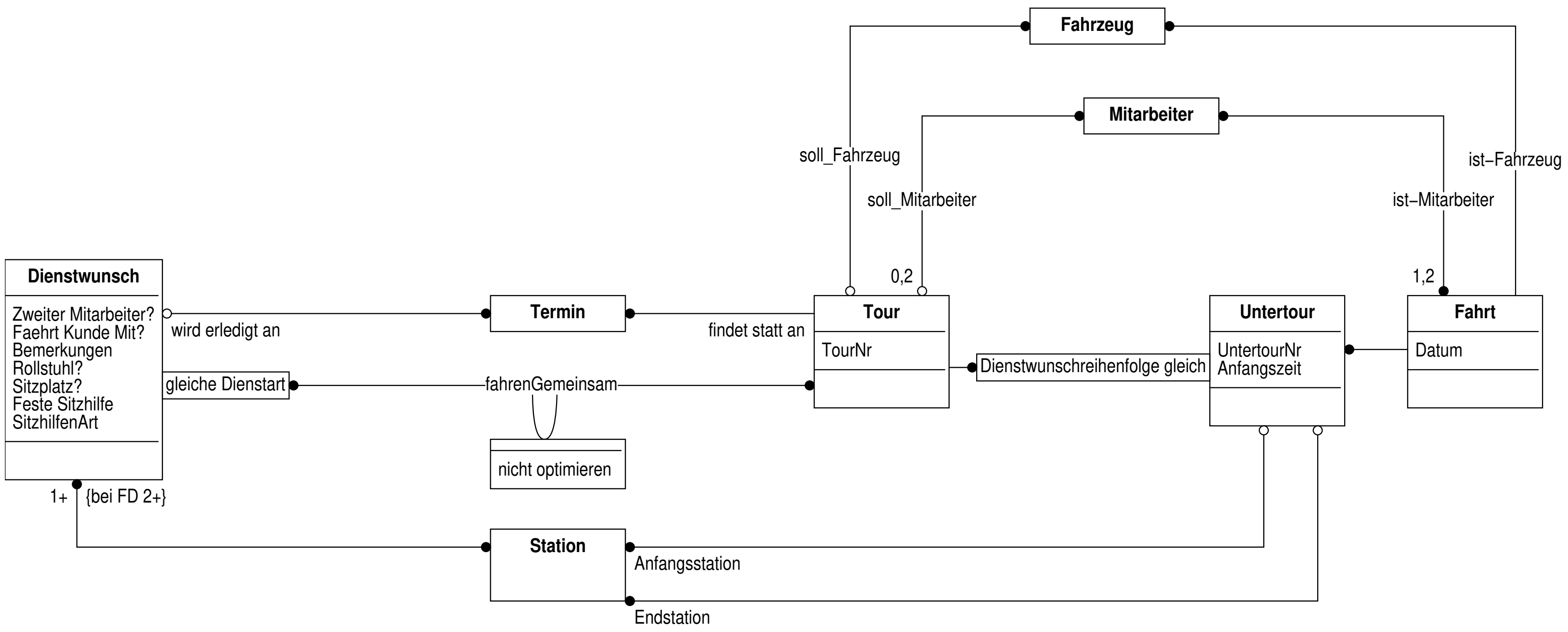


Abbildung 4.5: Touren

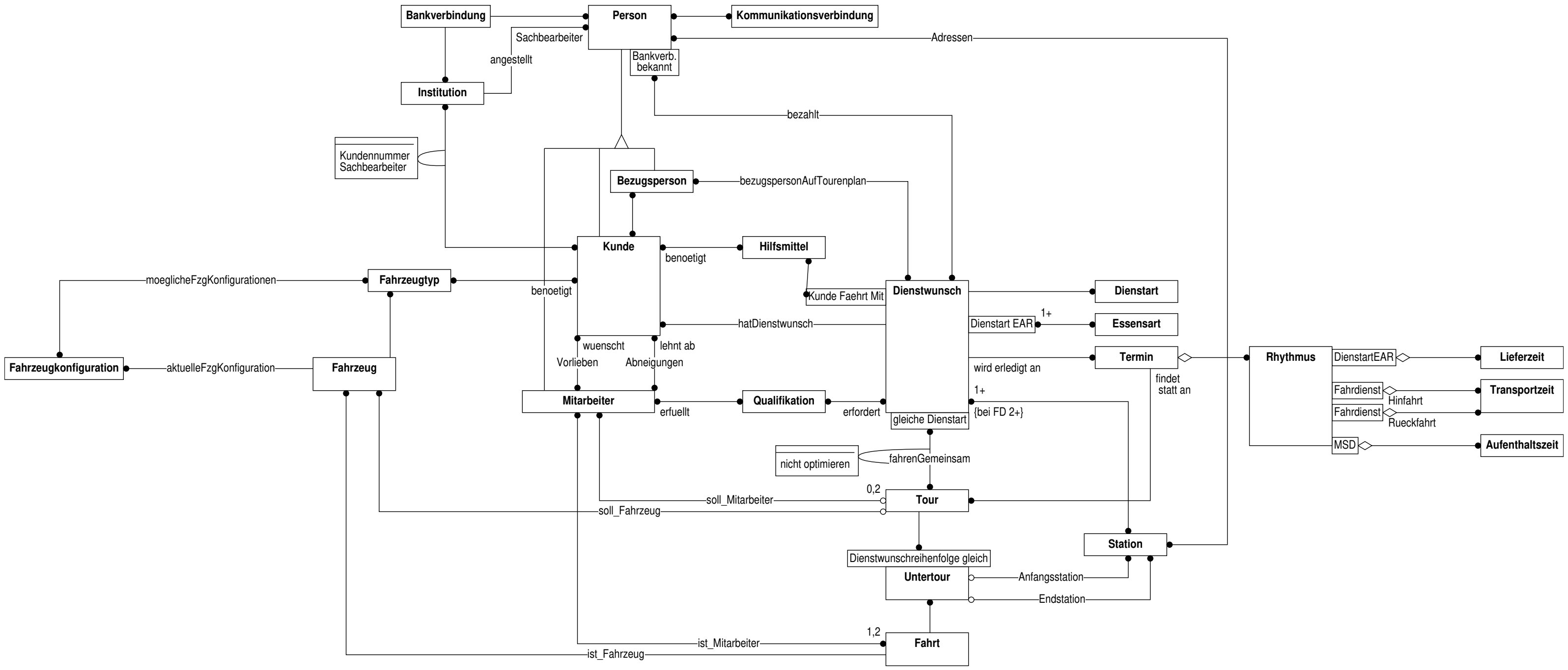


Abbildung 4.6: Gesamtübersicht

4.2.2 Klassen-Beschreibungen

Aufenthaltszeit		
Zeit bei Diensten mit Anfang und Aufenthaltsdauer		
Attribute	Spezifikation	
Anfangszeitraum	Beschreibung:	Zeitintervall innerhalb dessen der zugeordnete Dienstwunsch begonnen werden soll.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Muß innerhalb eines Tages liegen!
	Voreinstellung:	
Dauer	Beschreibung:	Dauer des Aufenthalts, z.B. beim Kunden, d.h. die für den zugeordneten Dienstwunsch benötigte Zeit, ohne An- und Abfahrtszeit.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	

Bankverbindung		
Bankverbindungen von Personen bzw. Institutionen.		
Attribute	Spezifikation	
BLZ	Beschreibung:	Bankleitzahl
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Bankname	Beschreibung:	Bankname zu obiger BLZ
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Fortsetzung der Klasse **Bankverbindung**		
Attribute	Spezifikation	
KontoNr	Beschreibung:	Kontonummer der Kontoverbindung bei obiger Bank
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Einzug?	Beschreibung:	Liegt eine Einzugsermächtigung vor?
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Bezugsperson		
Bezugspersonen haben einen bestimmten Bezug zu Kunden des DRK und dienen diesem als Ansprechpartner. Bezugspersonen sind z.B. Eltern, Kinder, Hausärzte.		
Attribute	Spezifikation	
Art des Bezuges	Beschreibung:	Kurzbeschreibung des Bezuges, z.B. Familienverhältnis oder Arzt
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Bemerkung	Beschreibung:	Freier Text zum Vermerken zusätzlicher Informationen
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Dienstart		
Die Dienstart gibt an, um welchen, der vom DRK angebotenen, Dienste es sich handelt. Dienstarten sind z.B. MSD, EAR und Schulfahrdienst.		
Attribute	Spezifikation	
Bezeichnung	Beschreibung:	Eindeutiger (Kurz-)Bezeichner einer Dienstart
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Beschreibung	Beschreibung:	Nähere Beschreibung der Dienstart.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Dienstwunsch		
Der Dienstwunsch repräsentiert einen Kundenauftrag.		
Attribute	Spezifikation	
Zweiter Mitarbeiter?	Beschreibung:	Angabe, ob ein zusätzlicher Mitarbeiter benötigt wird.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	FALSE
Faehrt Kunde Mit?	Beschreibung:	Angabe ob der Kunde bei Ausführung des Dienstwunsches mitfahren möchte. Wird nur beim MSD benötigt um zu wissen, ob auch an das Fahrzeug Anforderungen gestellt werden.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	bei MSD und EAR: FALSE, bei Fahrdiensten: TRUE

Fortsetzung der Klasse **Dienstwunsch**		
Attribute	Spezifikation	
Bemerkungen	Beschreibung:	Beliebiger Text, der auf dem Tourplan gedruckt werden soll.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Rollstuhl?	Beschreibung:	Ist true, wenn für den Dienstwunsch ein Rollstuhl benötigt wird.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nur true sein, wenn „Sitzplatz? ’ünd „FesteSitzhilfe?“ false sind.
	Voreinstellung:	FALSE
Sitzplatz?	Beschreibung:	Ist true, wenn ein Sitzplatz (mit oder ohne Sitzkissen) für die Ausführung des Dienstwunsches benötigt wird.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nur true sein, wenn „Rollstuhl?“ und “FesteSitzhilfe?“ false sind.
	Voreinstellung:	TRUE
FesteSitzhilfe?	Beschreibung:	Ist true, wenn eine festeingebaute Sitzhilfe benötigt wird.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nur true sein, wenn „Rollstuhl?“ und „Sitzplatz?“ false sind.
	Voreinstellung:	FALSE
Sitzhilfenart	Beschreibung:	Gibt an, welche Sitzhilfe benötigt wird (Sitzkissen, Spezialsitz).

Fortsetzung der Klasse **Dienstwunsch**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nur, wenn "FesteSitzhilfe?" true ist.
	Restriktionen:	ist "Rollstuhl?" true, darf hier nichts stehen.
	Voreinstellung:	

Essensart		
Repräsentiert die unterschiedlichen Essensarten, beispielsweise Normalkost, Diätkost, vegetarisch oder püriert.		
Attribute	Spezifikation	
Bezeichnung	Beschreibung:	Eindeutige (Kurz-)Bezeichnung für die Essensart, z.B. vegetarisch, pürierte Normalkost, etc.
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Beschreibung	Beschreibung:	Nähere Erläuterung, welches Essen mit der Bezeichnung gemeint ist.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Warm?	Beschreibung:	Gibt an, ob es sich um ein warmes oder tiefgefrorenes Essen handelt.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	TRUE
Preis	Beschreibung:	Der Preis, der für ein Essen dieser Art bezahlt werden muß.

↷

Fortsetzung der Klasse **Essensart**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	

Fahrt		
Die Fahrt modelliert eine tatsächlich stattfindende Untertour und unterscheidet sich von dieser im wesentlichen durch ein festes Datum, an dem sie ausgeführt wird. Eine Fahrt ist also etwas einmaliges.		
Attribute	Spezifikation	
Datum	Beschreibung:	Ein festes Datum, an dem die Fahrt stattfindet.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	

Fahrzeug		
Repräsentiert ein vorhandenes Fahrzeug.		
Attribute	Spezifikation	
FahrzeugNr	Beschreibung:	DRK-interne Fahrzeugnummer
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
KfzNr	Beschreibung:	amtliche Zulassungsnummer
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Modell	Beschreibung:	Herstellername und Modell-Bezeichnung des Herstellers
	Typ:	

↷

Fortsetzung der Klasse **Fahrzeug**		
Attribute	Spezifikation	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Anzahl Kfz-Schluessel	Beschreibung:	Anzahl der mitgelieferten bzw. vorhandenen Schlüssel für das Fahrzeug
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Verfügbarkeitszeiten	Beschreibung:	Termine, an denen das Fahrzeug eingesetzt werden kann. In der Regel ein offener Zeitrahmen mit Ausnahmen, wenn das Fahrzeug beispielsweise in der Werkstatt ist.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Anzahl Sitzhilfen Je Sitzhilfenart	Beschreibung:	Anzahl der im Fahrzeug vorhandenen und nicht festinstallierten Sitzhilfen je Sitzhilfenart (z.B. Kindersitze)
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	0
Naechster Kundendienst	Beschreibung:	Termin, an dem das Fahrzeug das nächste Mal zum Kundendienst muß
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein

Fortsetzung der Klasse **Fahrzeug**		
Attribute	Spezifikation	
	Restriktionen:	
	Voreinstellung:	
Naechste HU	Beschreibung:	Datum, an dem das Fahrzeug das nächste Mal zur Hauptuntersuchung muß
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Naechste ASU	Beschreibung:	Datum, an dem das Fahrzeug das nächste Mal zur Abgassonderuntersuchung muß
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Fahrzeugkonfiguration		
Konfigurationsmöglichkeit eines Fahrzeugs		
Attribute	Spezifikation	
Anzahl Sitzplaetze	Beschreibung:	Anzahl der eingebauten Sitzplätze, einschließlich des Fahrersitzes und ohne Rollstuhlplätze
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nicht größer sein, als die maximale Sitzplatzzahl des zugeordneten Fahrzeugs abzüglich der Anzahl Rollstuhlplätze. Mindestens ein Sitzplatz muß exisitiern (für den Fahrer).
	Voreinstellung:	
Anzahl Rollstuhlplaetze	Beschreibung:	Anzahl der Plätze für Rollstühle, d.h. Plätze an denen kein eingebauter Sitzplatz ist.

Fortsetzung der Klasse **Fahrzeugkonfiguration**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nicht größer sein, als die maximale Sitzplatzzahl des zugeordneten Fahrzeugs abzüglich der Anzahl Sitzplätze
	Voreinstellung:	
Anzahl Festeingebauter Sitzhilfen	Beschreibung:	Anzahl und Art der festeingebauten Sitzhilfen. Diese können im Gegensatz zu normalen Sitzhilfen nicht einfach vom Sitz entfernt werden
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	0
Kuehlmoeglichkeit?	Beschreibung:	Gibt an, ob das Fahrzeug eine Kühlvorrichtung für tiefgefrorenes Essen (z.B. Schienen für Kühlbox) besitzt.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	FALSE

Fahrzeugtyp		
Fahrzeuge werden in Typen eingeteilt, die für das DRK relevant sind. Beispiele: Rollstuhlbus oder hoher Bus.		
Attribute	Spezifikation	
Bezeichnung	Beschreibung:	eindeutige (Kurz-)Bezeichnung des Fahrzeugtyps
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Beschreibung	Beschreibung:	nähere Beschreibung des Fahrzeugtyps

Fortsetzung der Klasse **Fahrzeugtyp**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Maximale Sitzplatzzahl	Beschreibung:	Maximale Anzahl der Sitzplätze incl. des Fahrersitzes (lt. KFZ-Schein).
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Kilometerpreis	Beschreibung:	Preis, der je Kilometer mit einem Fahrzeug diesen Typs abgerechnet wird.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	

Hilfsmittel

Hilfsmittel die ein Kunde benötigt. Beispielsweise Rollstuhl oder Sitzkissen. Hilfsmittel werden bei der Zuordnung der Fahrzeuge beachtet, falls der Kunde in einem Fahrzeug mitfährt.

Attribute	Spezifikation	
Bezeichnung	Beschreibung:	Eindeutiger Bezeichner für das Hilfsmittel, z.B. Krücken, Rollstuhl, Kindersitz, Sitzschale, etc.
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Beschreibung	Beschreibung:	nähere Beschreibung für das Hilfsmittel, z.B. besonders großer Rollstuhl, Spezialsitz für Kunde X

Fortsetzung der Klasse **Hilfsmittel**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Institution		
Private und öffentliche Institutionen wie Krankenkassen, Sozilaämter u.ä.		
Attribute	Spezifikation	
Name	Beschreibung:	Name der Institution
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	

Kommunikationsverbindung		
Mögliche (technische) Kommunikationsverbindungen einer Person. Diese können frei gewählt werden, z.B. Telefon, Fax oder eMail.		
Attribute	Spezifikation	
Art	Beschreibung:	Art der Kommunikationsverbindung, z.B. Telefon, Fax, eMail, Handy
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Eintrag	Beschreibung:	Zur Art passender Eintrag, z.B. Telefonnummer, eMail-Adresse
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	muß zur Kommunikationsart passen
	Voreinstellung:	

Kunde		
Modelliert einen Kunden des DRK.		
Attribute	Spezifikation	
KundenNr	Beschreibung:	Eindeutige Nummer, welche den Kunden innerhalb des DRK identifiziert
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Hausschlüsselzahl	Beschreibung:	Anzahl der Kunden-Hausschlüssel, die beim DRK vorhanden sind.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Hausschlüsseltext	Beschreibung:	Zur freien Verwendung. Hier kann z.B. eine Schlüsselnummer, oder Türbezeichnung stehen.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
MaxFahrdauer	Beschreibung:	Gibt an, wie lange ein Kunde maximal in einem Fahrzeug zubringen darf. Global existierende Beschränkungen für die Fahrdauer (z.B. Schulfahrten 2h) müssen hier nicht eingegeben werden; diese werden automatisch berücksichtigt.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Fortsetzung der Klasse **Kunde**		
Attribute	Spezifikation	
Bemerkung	Beschreibung:	Hier können diverse zusätzliche Angaben zum Kunden gemacht werden. Bemerkungen werden bei der Planung nicht berücksichtigt, erscheinen jedoch auf den Tourplänen.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Lieferzeit		
Die Uhrzeit bei Lieferterminen (z.B. EAR)		
Attribute	Spezifikation	
Lieferzeitraum	Beschreibung:	Zeitrahmen, innerhalb dessen die Lieferung erfolgen soll.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Muß innerhalb eines Tages liegen!
	Voreinstellung:	

Mitarbeiter		
Modelliert einen Mitarbeiter des DRK.		
Attribute	Spezifikation	
PersonalNr	Beschreibung:	Eindeutige Nummer, die den Mitarbeiter innerhalb des DRK identifiziert
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	Muß eindeutig sein
	Voreinstellung:	
Dienstantrittsdatum	Beschreibung:	Datum, an dem der Mitarbeiter seinen Dienst antritt.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja

↷

Fortsetzung der Klasse **Mitarbeiter**		
Attribute	Spezifikation	
	Restriktionen:	
	Voreinstellung:	
Entlassungsdatum	Beschreibung:	Datum, zu dem der Mitarbeiter offiziell ausscheidet, d.h. ab dem er kein Entgeld mehr bekommt. Bei festangestellten Mitarbeitern ist dieser Eintrag i.d.R. leer.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Verfuegbarkeitszeiten	Beschreibung:	Zeitintervalle, an denen der Mitarbeiter zur Verfügung steht.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Bemerkung	Beschreibung:	Beliebiger Text, der den Einsatzplanern in der Mitarbeitermaske zur Information angezeigt wird, der jedoch sonst nirgends auftaucht.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Person		
Enthält die Daten, die allen Personen innerhalb des Systems gemeinsam sind und stellt damit deren Superklasse dar.		
Attribute	Spezifikation	
Name	Beschreibung:	Nachname der Person
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	

Fortsetzung der Klasse **Person**		
Attribute	Spezifikation	
	Voreinstellung:	
Vorname	Beschreibung:	Vorname der Person
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Geburtsdatum	Beschreibung:	Geburtsdatum der Person
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Qualifikation		
Qualifikationen modellieren, was ein Mitarbeiter kann, bzw. welchen Anforderungen an Ihn gestellt werden, z.B. ob er spritzen kann oder ob er Lasten heben darf.		
Attribute	Spezifikation	
Bezeichnung	Beschreibung:	Eindeutiger (Kurz-)Bezeichner für die Qualifikation
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	
Beschreibung	Beschreibung:	Nähere Beschreibung der Qualifikation
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Verrechnungswert	Beschreibung:	Kostenfaktor, der bei der Optimierung berücksichtigt wird, falls diese Qualifikation erfüllt, jedoch nicht gefordert wird. Der Verrechnungswert wird in der Regel automatisch berechnet, wenn sich Mitarbeiter oder Qualifikationen ändern. Die automatische Berechnung erfolgt jedoch nur, wenn Autowert false ist.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	
Autowert	Beschreibung:	Dieser Wert gibt an, ob der Verrechnungswert automatisch berechnet werden darf: Ist er true, darf die automatische Berrechnung erfolgen, ansonsten behält der Verrechnungswert seinen Eintrag.
		↷

Fortsetzung der Klasse **Qualifikation**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	Wird ein Verrechnungswert vom Benutzer angegeben, wird der Autowert auf false gesetzt, ansonsten ist die Voreinstellung true..

Rhythmus		
der Rhythmus in dem ein Termin stattfinden soll		
Attribute	Spezifikation	
Wochentage	Beschreibung:	Gibt an, an welchen Wochentagen der zugeordnete Dienstwunsch ausgeführt werden soll. Mehrfachnennung (z.B. montags, mittwochs und freitags) ist möglich.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
NurWerktags?	Beschreibung:	Hier wird vermerkt, ob der zugeordnete Dienstwunsch nur an Werktagen oder auch an Feiertagen ausgeführt werden soll.
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	TRUE
MeldungAn-Feiertagen?	Beschreibung:	Soll ein zugeordneter Dienstwunsch an Feiertagen nicht durchgeführt werden, wird der Benutzer an Feiertagen darauf aufmerksam gemacht, daß der Dienstwunsch nicht erfüllt wird. Dies gibt ihm die Möglichkeit mit dem Kunden einen Ersatztermin zu vereinbaren.

Fortsetzung der Klasse **Rhythmus**		
Attribute	Spezifikation	
	Typ:	BOOLEAN
	Bereich:	TRUE, FALSE
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Kann nur TRUE sein, wenn NurWerktags? TRUE ist!
	Voreinstellung:	Bei Fahrdiensten: FALSE, sonst TRUE.
Wochenrhythmus	Beschreibung:	Gibt an alle wieviel Wochen der zugeordnete Dienstwunsch ausgeführt werden soll
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	FALSE

Station
Eine Station ist im wesentlichen eine Adresse. Außerdem enthält sie für die Verarbeitung mit dem Verkehrsmodul einen Anbindungspunkt an dessen Straßensystem.

Attribute	Spezifikation	
Strasse	Beschreibung:	Straßen- oder Platzname, bzw. Postfach incl. der jeweiligen Nummer
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Ort	Beschreibung:	PLZ und Name des Ortes, in dem sich die Station befindet
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	
Anbindungspunkt	Beschreibung:	Für die Verarbeitung mit dem Verkehrsmodul muß eine Anbindung an dessen Straßensystem vorhanden sein. Dies ist in der Regel die Adresse der Station. Ist diese jedoch im Verkehrsmodul nicht identifizierbar, kann hier auch ein anderer Anbindungspunkt angegeben werden, der dem Verkehrsmodul bekannt ist (und möglichst nahe an der Stationsadresse liegt).
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	Die Stationsadresse im Format des Verkehrsmoduls

Termin		
Der Termin legt fest, wann zugeordnete Dienstwünsche ausgeführt werden.		
Attribute	Spezifikation	
Zeitrahmen	Beschreibung:	Das Datumsintervall, in dem der zugeordnete Dienstwunsch ausgeführt werden soll. Ist nicht bekannt, wie lange ein Dienstwunsch erfolgen soll, ist das Intervallende offen.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	Intervallanfang $<$ Intervallende, falls Intervallende vorhanden
	Voreinstellung:	
Ausnahmezeiten	Beschreibung:	Datumsintervalle innerhalb des Zeitrahmens, in denen der zugeordnete Dienstwunsche nicht erfolgen soll, z.B. Schulferien, Kunde im Urlaub
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Tour		
Eine Tour faßt Dienstwünsche gleicher Dienstart zusammen, die später als „ein Block“ gefahren werden.		
Attribute	Spezifikation	
TourNr	Beschreibung:	Nummer, die die Tour eindeutig identifziert
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	muß eindeutig sein
	Voreinstellung:	

Transportzeit		
Zeit bei Fahrdiensten		
Attribute	Spezifikation	
Abfahrtszeitraum	Beschreibung:	Uhrzeitintervall, in dem der Transport beginnen soll.

Fortsetzung der Klasse **Transportzeit**		
Attribute	Spezifikation	
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	Nur, wenn kein Ankunftszeitraum vorhanden ist.
	Restriktionen:	Muß innerhalb eines Tages liegen!
	Voreinstellung:	
Einstiegszeit	Beschreibung:	Aufenthaltszeit am Einstiegsort, z.B. bis ein Kind vom 3. Stock bis zum Auto getragen wird. Wird kein Wert eingetragen, wird der Pauschalwert aus den Voreinstellungen übernommen.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	
	Restriktionen:	
	Voreinstellung:	
Ankunftszeitraum	Beschreibung:	Uhrzeitintervall, in dem das Ziel erreicht werden soll.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	Nur, wenn kein Abfahrtszeitraum vorhanden ist.
	Restriktionen:	Muß innerhalb eines Tages liegen!
	Voreinstellung:	
Aussteigezeit	Beschreibung:	Aufenhaltszeit am Aussteigeort, z.B wenn ein Kind in einen Rollstuhl gesetz und in die Klasse gefahren werden muß. Wird kein Wert eingetragen, wird der Pauschalwert aus den Voreinstellungen übernommen.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	nein
	Restriktionen:	
	Voreinstellung:	

Fortsetzung der Klasse **Transportzeit**		
Attribute	Spezifikation	

Untertour		
Untertouren entstehen aus Touren durch eine festgelegte Dienstwunschreihenfolge sowie eine feste Anfangszeit.		
Attribute	Spezifikation	
UntertourNr	Beschreibung:	Setzt sich aus der Tournummer und einem alphanumerischen Teil zusammen. Die Untertournummer identifiziert eine Untertour eindeutig.
	Typ:	
	Bereich:	
	Schlüssel:	ja
	Muß-Wert:	ja
	Restriktionen:	Muß eindeutig sein.
	Voreinstellung:	
Anfangszeit	Beschreibung:	Die feste Uhrzeit, zu der die Tour beginnt.
	Typ:	
	Bereich:	
	Schlüssel:	nein
	Muß-Wert:	ja
	Restriktionen:	
	Voreinstellung:	

4.2.3 Tourkonzept

In diesem Kapitel soll beschrieben werden, wozu die Unterscheidung von Touren, Untertouren und Fahrten dient. Außerdem sollen die Analogien zum derzeitigen Vorgehen beim DRK aufgezeigt werden.

Die Dienstpläne ergeben sich direkt aus den Fahrten. Sie stellen also nur eine andere Sicht auf die Daten und damit keine eigenständigen Objekte dar, weshalb sie im Datenmodell auch nicht berücksichtigt wurden.

4.2.3.1 Tour

Eine Tour faßt zunächst Dienstwünsche zusammen, die grundsätzlich miteinander bzw. nacheinander erledigt werden können (die also insbesondere von der gleichen Dienstart sind und mit einem Fahrzeug zu bewältigen sind). Damit entspricht die Tour der DRK-Tour, wie sie durch die derzeitigen Tourpläne beim DRK repräsentiert wird.

Die Assoziation **fahrenGemeinsam** nimmt gerade die oben angegebene Partitionierung der Dienstwünsche in Touren vor. Zudem enthält sie das Attribut *nicht optimieren* als Flag mit der Semantik: die Dienstwünsche der Tour, bei denen das Flag gesetzt ist, werden bei der automatischen Touroptimierung nicht verändert (bleiben also in jedem Fall in der Tour). Dies gewährleistet die DRK-Anforderung, daß Kunden in jedem Fall in einer bestimmten Tour mitfahren sollen, z.B. damit bestimmte Kunden auch dann zusammenfahren können, wenn dies keine optimale Tour ergibt.

Die Assoziationen **sollFahrzeug** und **sollMitarbeiter** bieten die Möglichkeit, einer Tour maximal zwei Mitarbeiter und ein Fahrzeug zuzuordnen. Diese Sollwerte werden nach Möglichkeit bei der Fahrtenbildung berücksichtigt. Mit diesen Assoziationen wird also vorallem der Forderung des Mitarbeiterbezuges und der Ressourcenplanung nachgekommen. Um bei der Planung größtmögliche Flexibilität zu gewährleisten, ist die Zuweisung der Sollwerte optional. Damit werden jedoch die Analysefunktionen und Konsistenzprüfungen eingeschränkt, da nicht bekannt ist, welche Ressourcen für die Tour verplant werden.

Zu beachten ist, daß bei Touren nichts über die Reihenfolge der Dienstwünsche (und damit der Stationen) oder die Abfahrtszeit ausgesagt wird. Dies ist notwendig, da Kunden, unabhängig von den Rahmenbedingungen wie Uhrzeit oder ob es sich um eine Hin- bzw. Rückfahrt handelt, immer in gleichen Personengruppen befördert werden sollen. Außerdem wissen die Mitarbeiter anhand der eindeutigen Tournummer schon, welche Kunden sie abholen müssen.

4.2.3.2 Untertour

Mit der Untertour wird die Dienstwunsch- bzw. Stationenreihenfolge und der Rhythmus (Wochentag, Wochenrhythmus und Uhrzeit), zu der eine Tourinstanz gefahren werden soll, festgelegt. Damit ist es also beispielsweise möglich, Hin- und Rückfahrten zu gestalten. Außerdem kann mit einer Untertour dargestellt werden, daß die zu befördernde Personengruppe nicht unbedingt vollständig befördert werden muß (fehlt eine Person regelmäßig, ändert das nur etwas an der Untertour, die Personengruppe der Tour bleibt jedoch bestehen).

Auch die Untertouren existieren beim DRK bereits heute. Sie werden jedoch nicht explizit als solche aufgeführt, sondern finden sich als Variationen der Tour auf dem DRK-Tourplan wieder: im Kopf steht die Information zu unterschiedlichen Startzeiten, die Stationenreihenfolge bei Rückfahrten wird einfach gedanklich umgekehrt und regelmäßig ausfallende Personen werden als Bemerkungen bei den jeweiligen Stationen vermerkt.

4.2.3.3 Fahrt

Die Fahrten repräsentieren konkret gefahrene Untertouren und unterscheiden sich von diesen im wesentlichen durch ein festes Datum.

Da bekannt ist an welchem Datum eine Fahrt stattfindet, kann anhand der Verfügbarkeitszeiten der Mitarbeiter und Fahrzeuge überprüft werden, ob die zugeordneten Mitarbeiter und das zugeordnete Fahrzeug überhaupt verfügbar sind. Auch bei den Dienstwünschen der Untertour kann anhand ihrer Termine geprüft werden, ob sie ausgeführt werden müssen. Ist beispielsweise Kunde X im Urlaub, wird dessen Dienstwunsch nicht in der Fahrt berücksichtigt, ohne daß dies weitere Auswirkungen auf die bestehenden (Unter-)Touren hat.

Auch die TRO-Fahrten existieren schon heute beim DRK: Abweichungen von bestehenden Touren, beispielsweise wegen Mitarbeiter- oder Fahrzeugausfall, werden den betroffenen (evtl. neu eingeteilten) Mitarbeitern als ergänzende Information zum Tourenplan schriftlich oder mündlich mitgeteilt.

4.3 Verkehrsmodul

Das Verkehrsmodul bildet die Schnittstelle zwischen dem Transportoptimierungssystem und einem Verkehrstool (einer digitalen Straßenkarte). Es bietet dem Rest des Systems eine Möglichkeit, Fahrzeiten zwischen zwei Stationen zu ermitteln, die als Kombination aus Postleitzahl, Ortsname, Straßenname und Hausnummer gegeben sind. Das Verkehrstool wird dabei komplett eingekapselt, wodurch es auch einfach auszutauschen ist.

4.3.1 Das Verkehrstool: Map&Guide

Als Verkehrstool wurde Map&Guide der CAS Software GmbH in der Version 4.1 gewählt. Standardmäßig hat es nur eine interaktive Benutzerschnittstelle. Mit dem Zusatzmodul „batch-control" bietet es aber auch eine Möglichkeit, von anderen Programmen aus Anfragen zu stellen. Dazu legt man eine Auftragsdatei mit einer Liste der Stationen an, die in einer Tour angefahren werden sollen. Map&Guide sieht in regelmäßigem, einstellbaren Abstand in einem ebenfalls einstellbaren Verzeichnis nach solchen Auftragsdateien. Hat es eine gefunden, liest es sie ein und kann zunächst die Reihenfolge der Stationen optimieren, was hier aber nicht verwendet wird. Für die Tour wird dann ein kürzester Weg gesucht, wobei das Geschwindigkeitsprofil des Fahrzeugs und die Gewichtung von räumlicher Entfernung und Fahrzeit eingestelt werden kann. Der ermittelte Weg kann dann auf einem Drucker oder in Map&Guide als Liste oder in einer Karte ausgegeben werden. Für die Weiterverarbeitung gibt es die Möglichkeit, die Liste der Stationen mit Zeiten und Entfernungen, wahlweise auch mit Wegbeschreibung, in einer Ergebnisdatei zu speichern.

Im Benutzerhandbuch wird noch ein mögliches Problem angesprochen: auf dem Rechner, auf dem die batch-Aufträge abgearbeitet werden, ist darüberhinaus fast kein Arbeiten mehr möglich.

4.3.2 Aufbau des Verkehrsmoduls

Die Beschreibung des Verkehrstools Map&Guide macht schon deutlich, daß eine Anfrage lange dauern kann: zuerst muß Map&Guide die neue Auftragsdatei entdecken, dann muß es den Weg suchen und speichern und schließlich muß das Verkehrsmodul die Ergebnisdatei finden und auswerten. Daher sollte die Zahl der Anfragen an Map&Guide möglichst gering gehalten werden.

Der erste Ansatz, um Anfragen an das Verkehrstool zu vermeiden, ist die Speicherung bereits ermittelter Entfernungen in einer Tabelle, wo sie auch über mehrere Aufrufe des Transportoptimierungssystems hinweg erhalten bleiben. Die Größe dieser Tabelle kann der Benutzer einschränken, da sie sehr groß werden kann (bis zu $|\text{Stationen}|^2 \approx |\text{Kunden}|^2$ Einträge). Ist diese Grenze erreicht, werden alte Einträge von neuen überschrieben.

Der zweite Ansatz ist die Ankündigung großer Mengen von Anfragen. Das Verkehrsmodul kann dann alle angekündigten Entfernungen mit einer Anfrage an das Verkehrstool ermitteln und speichern. Bei den folgenden Anfragen nach einzelnen Entfernungen kann dann auf die gespeicherten Werte zurückgegriffen werden. Es sind aber trotzdem einzelne Anfragen ohne vorherige Ankündigung möglich, deren Bearbeitung dann aber lange dauern kann.

Zur Ankündigung von Anfragen wurden zwei Ansätze diskutiert: ein eher „naiver“ und ein „intelligenterer“.

Bei der „naiven“ Variante wird dem Verkehrsmodul entweder eine Liste von zu verbindenden Stationspaaren übergeben, oder eine Liste von Stationen, zwischen denen alle Entfernungen ermittelt werden sollen. Der Aufrufer, also z.B. der gerade anlaufende Optimierungsalgorithmus, muß dann zunächst eine Menge von benötigten Entfernungen bestimmen. Wenn das nicht ohne weiteres möglich ist, kann auch eine Obermenge angegeben werden.

Beim „intelligenteren“ Ansatz wird vorgesehen, daß das Verkehrsmodul auch auf die restlichen Daten des Systems zugreift. Es sind dann auch Ankündigungen der Art „alle Entfernungen zwischen Stationen von Wünschen der Dienstart Schuhlfahrdienst“ oder „alle Entfernungen zwischen Stationen von Wünschen am Montag Morgen“ möglich.

Der „naive“ Ansatz hat den Vorteil, daß das Verkehrsmodul einfacher und leichter wartbar wird, da es vollkommen unabhängig vom restlichen System ist. Der „intelligente“ Ansatz hat den Vorteil, daß die einzelnen Ankündigungen, die von mehreren Stellen des restlichen Systems kommen, einfacher werden. Jedoch werden dabei in der Regel auch viele nicht benötigte Entfernungen angekündigt.

4.3.3 Korrektur von Entfernungen

Eine Anforderung des Kunden ist, daß er die gelieferten Entfernungen der Realität anpassen kann. Deshalb verwaltet das Verkehrsmodul eine Liste mit Änderungen, in der immer als erstes nach einer Entfernung gesucht wird. Grundsätzlich sollte der Benutzer aber zuerst nachsehen, weshalb eine Entfernung nicht stimmt. Führt zum Beispiel der Weg zwischen zwei Kunden, den das Verkehrstool als kürzesten ermittelt, durch eine gesperrte Straße, so hat es wenig Sinn, nur die Entfernung zwischen diesen beiden Kunden anzupassen. Besser wäre es dann, die fehlerhafte Straße im Verkehrstool zu korrigieren. Dazu wird bei Map&Guide aber das Zusatzmodul „road editor plus“ benötigt, das uns jedoch nicht zur Verfügung steht.

4.4 Datenausgabe

In diesem Abschnitt werden mögliche Ausgaben von TRO vorgestellt. Diese werden schrittweise, je nach vorhandener Zeit entsprechend ihrer Priorität[1] implementiert.

4.4.1 Dienstpläne

Dienstpläne zeigen die Zuordnung von Fahrten zu Mitarbeitern bzw. zu Fahrzeugen und dienen sowohl als Übersicht für die DRK-Planer, als auch für die persönliche Zeitplanung der Zivis. Die unterschiedlichen Ansprüche werden durch drei unterschiedliche Dienstpläne berücksichtigt:

- Ein Mitarbeiter / eine Woche (A)
- Alle Mitarbeiter / eine Woche (A)

[1] es gibt die Prioritäten A,B und C, wobei A die höchste darstellt. Die jeweiligen Prioritäten werden in Klammern am Ende des jeweiligen Eintrags angegeben.

- Alle Fahrzeuge / eine Woche (A)

Die Dienstpläne könnten in textueller Form als Tabelle oder als Ganttcharts dargeboten werden. Ein Beispiel für ein Ganttchart ist Abb. 4.7 .

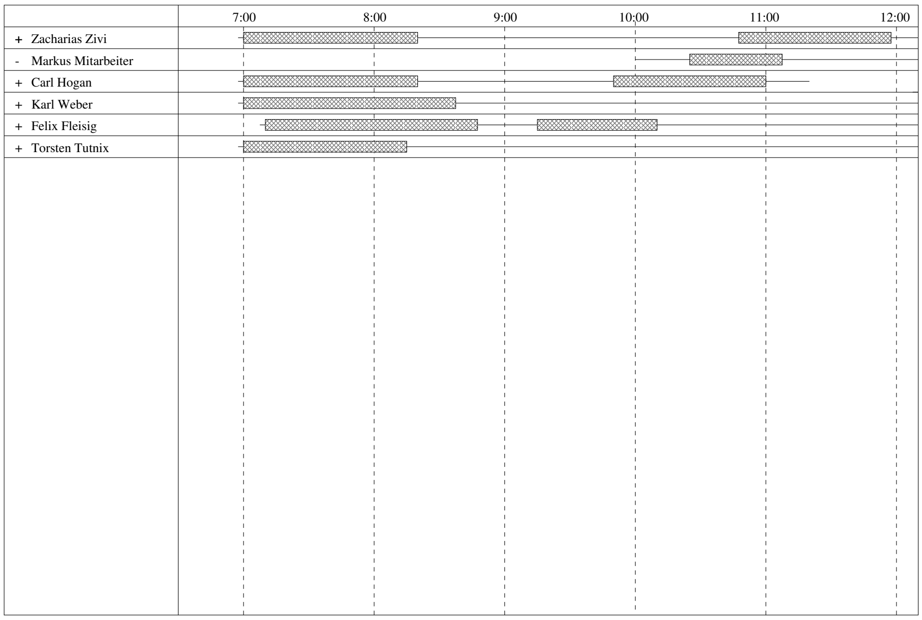


Abbildung 4.7: Beispiel für einen Dienstplan als Ganttchart

4.4.2 Analysedaten

Analysedaten dienen als Grundlage zur Infrastrukturplanung des DRK. Mit deren Hilfe sollen Fragen wie „werden mehr oder weniger Fahrzeuge benötig?“, oder „werden mehr oder weniger Mitarbeiter benötigt?“ beantwortet werden.

Folgende Analysedaten werden jeweils für Auslastung [2] und Ausfall angeboten:

- ein Mitarbeiter / Zeitraum (A)
- alle Mitarbeiter / Zeitraum (A)
- ein Fahrzeug / Zeitraum (A)
- alle Fahrzeuge / Zeitraum (A)
- alle Fahrzeuge einer Fahrzeugklasse / Zeitraum (A)

Die Auslastung bedeutet dabei: $\frac{\text{Einsatzzeit}}{\text{verfügbare Zeit}}$. Bei Fahrzeugen bzw. Fahrzeugklassen kann die Auslastung alternativ auch auf die Besetzung bezogen werden:

[2] Damit Aussagen über eine Auslastung sinnvoll sind, muß es einen Referenz- bzw. Maximalwert geben. D.h., daß für Mitarbeiter und Fahrzeuge eine Soll-Arbeitszeit für einen Tag und evtl. auch für eine Woche (da es beim DRK die 5- und die 7-tage Woche gibt) eingegeben werden muß

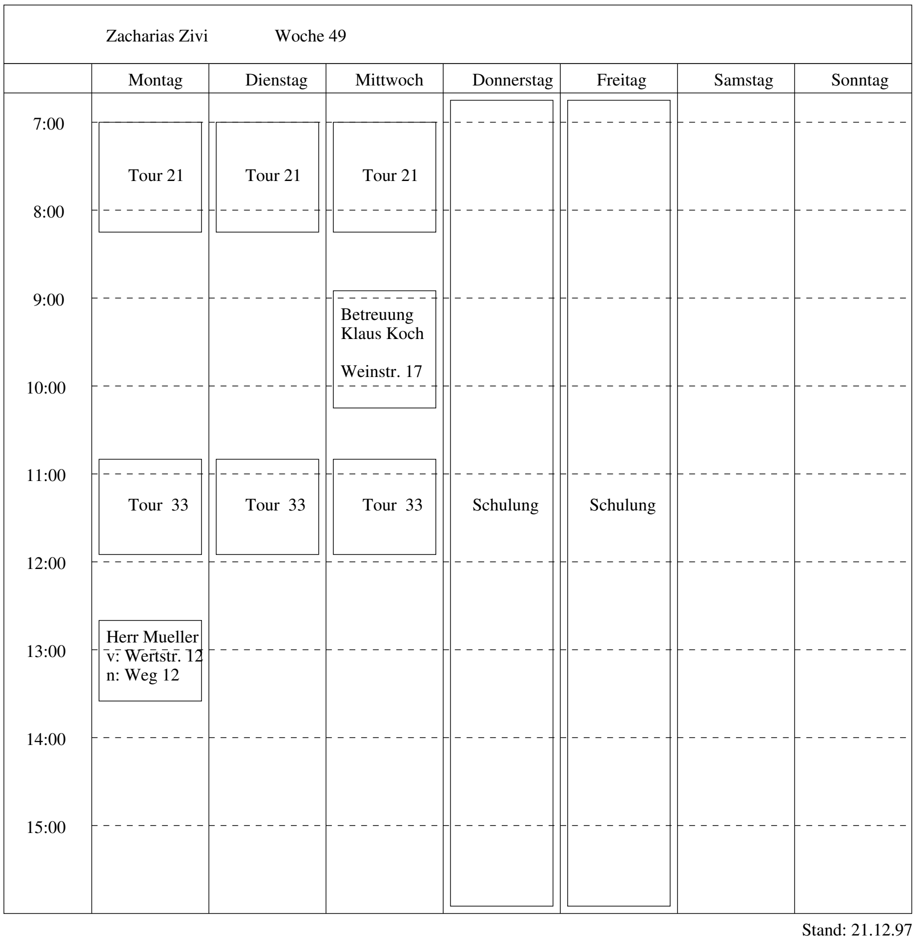


Abbildung 4.8: Beispiel für einen Dienstplan als Tabelle

$\frac{\text{belegte Plätze}}{\text{vorhandene Plätze}}$. Der Zeitraum ist vom Anwender einzugeben und kann beispielsweise ein Wochentag, eine Woche, ein Quartal oder ein Jahr sein. Ist der Zeitraum länger als ein Tag, kann er zustäzlich angeben, ob er die Auslastung für jeden Tag in diesem Zeitraum haben möchte, oder ob ein Durchschnitt der Auslastung über alle Tage gebildet werden soll. Desweiteren kann auch nach einem bestimmten Wochentag über einen Zeitraum gefragt werden, z.B. alle Dienstage im letzten Jahr.

Zusätzlich werden folgende Analysedaten angeboten:

- Kosten / Dienst (A)
- Kosten / Dienstwünsche (A)
- Kosten / Kunden (A)
- Anzahl Dienstwünsche je Dienstart (B)
- Anzahl Dienstwünsche / Anzahl Mitarbeiter (B)

- Anzahl Dienstwünsche / Anzahl Fahrzeuge (B)
- Einnahmen / Dienst (B)
- Einnahmen / Dienstwünsche (B)
- Einnahmen / Kunden (B)
- Anzahl, Zeit und Km der Leerfahrten (C)
- Anzahl, Zeit und Km je Dienstart (C)
- Durchschnittlich benötigte Zeit für die Erledigung eines Dienstwunsches (C)

4.4.3 Listen

Zum Gesamtüberblick über den aktuellen Bestand werden folgende Listen angeboten:

- Kunden
  - sortiert nach
    - Nachname, Vorname (B)
    - Kundennummer (B)
  - gefiltert nach Terminen ihrer Dienstwünsche (C)
- Mitarbeiter
  - sortiert nach
    - Nachname, Vorname (B)
    - Personalnummer (B)
  - gefiltert nach Qualifikation (auch mehrfach, z.B. alle Fahrer die Lasten heben können) (B)
  - nach Terminen, an denen sie verplant sind (aus Fahrten bzw. Untertouren) (C)
- Fahrzeuge
  - sortiert nach
    - Fahrzeug-Nummer (B)
    - nach Fahrzeug-Typ (B)
    - mögliche/aktuelle Konfigurationen (C)
  - nach Terminen, an denen sie verplant sind (aus Fahrten bzw. Untertouren) (C)

Als Ausgabeform der Analysedaten sind Balken- und Tortendiagramme vorgesehen.

Desweiteren werden für die Küchen Listen mit den benötigten Essen erstellt.

4.4.4 Anfragen

Anfragen sollen die DRK-Planer vor allem bei der Frage unterstützen, ob und wann ein bestimmter Dienst noch angenommen werden kann, bzw. wo es noch freie Kapazitäten gibt. Folgende Anfragen sind vorstellbar:

- Kann ein bestimmter Dienstwunsch mit/ohne Tourumstellungen erfüllt werden? (A)
- Ist zum Zeitpunkt (Datum + Uhrzeit) ein Mitarbeiter und/oder ein Fahrzeug verfügbar? (A)
- Wieviele Schultouren können mit/ohne Tourumstellung noch angenommen weden? (A)
- Rechnen sich die Km-Pauschalen? (B)

4.4.5 Konsistenztests

Konsistenztest sollen Unstimmigkeiten bei Touren, Untertouren und Fahrten aufdecken. Vorstellbar wären etwa:

- Gibt es unerfüllte Dienstwünsche? (A)
- Werden Zeitgrenzen eingehalten? (A)
- Ist kein Fahrzeug überfüllt? (A)
- Stimmen die Mitarbeiterqualifikationen und Fahrzeugtypen der Touren/Fahrten mit den Anforderungen der Dienstwünsche überein? (A)
- Hat jede Fahrt einen Fahrer? (A)
- Sind eingeplante Mitarbeiter und Fahrzeuge einsatzfähig? (A)
- Fahren Mitarbeiter bzw. Fahrzeuge mehrere Touren gleichzeitig? (A)
- Steigen alle Kunden, die in ein Fahrzeug einsteigen, auch aus dem gleichen wieder aus (und umgekehrt)?[3] (A)

4.5 Lokale Optimierung

„Lokale" Optimierung bedeutet hier, daß die Optimierung nach Dienstarten getrennt vorgeht und immer nur Dienstwünsche einer Dienstart kombiniert, um möglichst optimale Touren und Untertouren zu bilden. Dabei müssen je nach Dienstart unterschiedlich viele Randbedingungen berücksichtigt werden:

	Orte	Qualifikation der Mitarbeiter	Fahrzeug-Konfiguration	Zeit
EAR	1			(×)
MSD	1	×	(×)	×
Schulfahrten / Dialysefahrten / Tagespflegefahrten	1	×	×	(×)
Individualfahrten	2	×	×	×

[3]Wenn Stationenfolge im Dienstwunsch nicht korrekt ist

Daraus ergibt sich eine Hierarchie hinsichtlich der Anzahl der zu berücksichtigenden Parameter:

1. EAR: Wohnort des Kunden. In seltenen Fällen auch Vorgaben für den Lieferzeitpunkt.

2. MSD: Wohnort des Kunden + Anforderungen an die Qualifikation der Mitarbeiter + Vorgaben zur Uhrzeit.

3. Schulfahrten, Dialysefahrten und Fahrten zur Tagespflege: Hier werden viele Kunden zu wenigen Zielen gefahren, so daß die Kunden nach dem Ziel gruppiert werden können. Dadurch muß die Optimierung lediglich die unterschiedlichen Startpunkte berücksichtigen. Hinzu kommen Anforderung an die Qualifikation der Mitarbeiter sowie an die Ausstattung des Fahrzeugs. Außer der Ankunftszeit ist nur mit wenigen zeitlichen Anforderungen zu rechnen.

4. Individualfahrten: Hier gibt jeder Kunde individuell Start- und Zielort vor (sowie Start- bzw. Zielzeit). Anforderungen an Mitarbeiter und Fahrzeuge existieren genauso wie bei den anderen Fahrdiensten.

4.5.1 Ein- und Ausgabe der Optimierung

Gegeben Eine Menge von Kunden mit ihren Dienstwünschen, bestehend aus Start- und Zielort, sowie Anforderungen an Zeiten, Qualifikationen der Mitarbeiter und Konfigurationen der Fahrzeuge. Außerdem eine Menge von Mitarbeitern mit bestimmten Qualifikationen sowie eine Menge von Fahrzeugen mit verschiedenen Konfigurationen.

Gesucht Eine Menge von Untertouren, auf denen alle Kunden gemäß ihren Wünschen mit den vorhandenen Mitarbeitern und Fahrzeugen befördert werden können. Gibt es mehrere Lösungen, sollte die beste im Sinne einer noch zu definierenden Bewertung gewählt werden.

4.5.2 *Was* soll optimiert werden?

1. Anzahl der eingesetzten Fahrzeuge/Mitarbeiter möglichst gering

2. Summe der Einsatzzeiten möglichst klein

3. Bei Fahrten zur Tagespflege, Essen auf Rädern sowie MSD/APD, die pauschal bezahlt werden (und nicht nach gefahrenen Kilometern) sollte auch die Anzahl der gefahrenen Kilometer möglichst klein sein.

4.5.3 Effizienz

Ein intuitiver Lösungsansatz könnte etwa so aussehen:

Bestimme eine Qualitätsfunktion bzw. Kostenfunktion, bezüglich derer verschiedene Lösungen (= Mengen von Untertouren) verglichen werden können. Diese könnte beispielsweise als

$$K(T) = g_1 * \text{Anzahl Fahrzeuge} + g_2 * \text{Anzahl Mitarbeiter}$$

definiert werden. Die Lösung des Problems bestünde nun darin, $K(T)$ zu minimieren, d.h. unter allen möglichen Untertouraufteilungen diejenige zu finden, für die diese Funktion einen minimalen Wert annimmt.

Da es sich bei der Optimierung von Fahrdiensten um eine NP-vollständiges Problem handelt [Buc94] (und wir davon ausgehen, daß P≠NP ist), ist ein solch allgemeines Verfahren nicht effizient lösbar (die Laufzeit der Suche steigt exponentiell zur Anzahl der Dienstwünsche, die eingeplant werden müssen). Deshalb werden im folgenden Heuristiken zum Einsatz kommen, die weder immer eine optimale Lösung finden noch überhaupt garantieren können, daß eine existierende Lösung gefunden wird, von denen man sich aber nach mäßiger Laufzeit eine zumindest akzeptable Lösung verspricht.

4.5.4 Maße für verschiedene Parameter

Sei W eine Menge von Dienstwünschen, die in Untertouren eingeteilt werden sollen. Seien $T_1, \ldots, T_i \subseteq W$ Mengen von Dienstwünschen, die im Verlaufe der Planung bereits zu den Untertouren 1 bis $i-1$ kombiniert wurden. Untertour i ist die momentan in Planung befindliche, der noch weitere Dienstwünsche hinzugefügt werden können. Sei weiter $R = W \setminus \bigcup_{k=1}^{i} T_k$ die Menge der restlichen Dienstwünsche, die noch nicht in Untertouren eingeteilt wurden.

Der Dienstwunsch, der als nächster an Untertour T_i angefügt wird, heiße D.

Nähe von Orten Für die Auswahl eines Dienstwunsches, der örtlich am besten zur Untertour T_i paßt, sind verschiedene Metriken denkbar:

1. D ist derjenige Dienstwunsch, der zum Ort eines beliebigen Wunsches aus T_i den minimalen Abstand hat (im Vergleich zu allen anderen Dienstwünsche aus R).
2. Die Summe der Abstände von D zu allen $D_j \in T_i$ ist minimal.
3. Die Streckenlänge von $T_i \cup \{D\}$ ist minimal. Dabei kann die Streckenlänge einer Untertour aus gegebenen Stationen approximiert werden, indem der minimale spannende Baum dieser Stationen so umlaufen wird, daß jede Station nur einmal auf diesem Weg liegt.

Anforderungen an Mitarbeiter/Fahrzeuge Ein Maß für die Stärke von Anforderungen an Mitarbeiter ist die Anzahl der Mitarbeiter, die diese Anforderungen erfüllen können. Der anspruchsvollste Dienst bzw. Dienstwunsch ist also der, den am wenigsten Mitarbeiter ausführen können. Analog lassen sich Dienstwünsche nach ihren Ansprüchen an das eingesetzte Fahrzeug ordnen.

Reihenfolge von Stationen Da die eingesetzten Fahrzeuge jeweils nur eine kleine Anzahl von Kunden transportieren können (zwischen zwei und acht), läßt sich eine hinsichtlich Fahrtdauer bzw. -strecke gute Stationenfolge recht einfach und effizient bestimmen.

4.5.5 *Wie* könnte eine gute Lösung gefunden werden?

4.5.5.1 Matching-Verfahren

Für das Problem der Bildung von allgemeinen Fahrgemeinschaften ließen sich durch Maximale Matchings auf Personengraphen gute Lösungen finden, indem die Knoten (=Personen) im Graphen genau dann verbunden werden, wenn zwei Personen zusammen fahren können [Pro97]. Ein solcher Ansatz ist für die Transporte von Personen mit besonderen Bedürfnissen (Alte, Kranke, Behinderte) nicht sinnvoll:

- Die Aussage „Person X und Y können zusammen befördert bzw. betreut werden", liefert keine wertvolle Information. Ein hochqualifizierter Mitarbeiter kann mit einem optimal ausgerüsteten Fahrzeug (z.B. Rollstuhlbus mit Hebebühne, der zudem noch eine Sitzbank mit verschiedenen Spezialsitzen eingebaut hat) beliebige Dienstwünsche befriedigen. Das heißt, je zwei beliebige Kunden „matchen" auf diese Art, wobei dadurch möglicherweise für die restlichen Kunden nicht mehr genügend qualifizierte Mitarbeiter und Fahrzeuge zur Verfügung stehen.

  Um die Aussage „X kann mit Y" fahren korrekt zu modellieren, wäre ein gerichteter Graph notwendig. Denn ein gehbehinderter Kunde, den man stützen muß, der aber in einem PKW fahren kann, kann gut auf der Sitzbank eines Rollstuhlbusses mitfahren. Umgekehrt ist es nur in gewissen Fällen möglich, einen Rollstuhlfahrer in einem PKW zu transportieren.

- Ein Matching auf einem Graphen, der jeder Person mögliche Fahrzeuge zuordnet, wird dem quantitativen Aspekt nicht gerecht. Ein Maximales Matching würde Paare bilden, so daß zwar viele Kunden einem passenden Fahrzeug zugeordnet würden, aber eben jedem Fahrzeug nur ein Kunde. Da gerade die Anzahl der eingesetzten Fahrzeuge durch Bildung von Untertouren minimiert werden soll, führt dieser Ansatz zu nichts.

- Verbindet man Dienstwünsche, die nacheinander ausgeführt werden könnten, in einem Graphen durch Kanten, bildet ein Matching-Verfahren zwar korrekte, aber zu kleine Untertouren (alle Dienste sollen 45 Minuten dauern, die Zwischenstrecken sind jeweils in 15 Minuten zurückzulegen):

Kunde	Dienst-beginn
A	16 Uhr
B	17 Uhr
C	17 Uhr
D	18 Uhr
E	18 Uhr
F	19 Uhr

  Ein Maximales Matching wäre z.B. {AB, CD, EF}, eine Zusammenfassung zu zwei Untertouren {ABDF, CE} wäre hier allerdings zu bevorzugen.

4.5.5.2 EAR: Greedy-Ansatz nach Wohnortnähe

Für den Dienst „Essen auf Rädern" wird angenommen, daß die meisten Kunden für jeden Tag Essen bekommen möchten (zumindest für jeden Werktag). Der Tourplan wird deshalb für die ganze Woche erstellt, so daß sich für jeden Tag die gleiche Untertour ergibt, einzelne Ausnahmen werden nach Bedarf berücksichtigt, ohne jedoch die Tour an sich umzugestalten.

Lieferungen von kaltem/tiefgefrorenem und warmem Essen werden getrennt geplant.

Algorithmus Beginne mit einem beliebigen Kunden. Füge solange den jeweils nächsten Kunden zur Untertour hinzu, bis entweder alle Kunden eingeplant sind oder die Ladekapazität des Fahrzeugs bzw. der späteste erlaubte Lieferzeitpunkt überschritten ist. Im letzteren Fall werden analog weitere Untertouren erzeugt.

Ausgabe Partitionierung der Kunden. Zwei Kunden sind in derselben Partition, wenn sie mit demselben Fahrzeug beliefert werden. Wird als Maß für den „nächsten Kunden" der geometrische/zeitliche Abstand genommen, ergibt sich die Reihenfolge der Stationen aus der Reihenfolge, in der die Kunden zur Untertour hinzugefügt wurden. In anderen Fällen (z.B. Abstandssumme) muß diese Stationenfolge ggf. noch ermittelt werden.

Eigenschaften

Vorteile Die Summe der Fahrtzeiten in allen Untertouren ist recht gering.

Nachteile Vorgaben für die Lieferzeit werden nicht berücksichtigt. Dies könnte jedoch dadurch erreicht werden, daß ein Kunde, der zeitlich nicht paßt, bei der Suche nach dem nächsten Kunden übersprungen wird.

Laufzeit Um den nächsten Kunden zu ermitteln, müssen alle restlichen verglichen werden. Es entsteht also der Aufwand $\mathcal{O}(w^2)$ (w sei die Anzahl der Dienstwünsche).

4.5.5.3 MSD/APD: Auswahl nach Beginn der Dienste

Da Dienste dieser Art im allgemeinen eine Zeit vorgegeben haben, wann sie durchgeführt werden sollen, liegt es nahe, Untertouren nach diesem Kriterium zusammenzustellen.

Algorithmus Beginne mit dem Dienst, der am frühesten begonnen werden kann. Setze die Untertourzeit auf dessen frühestmögliche Anfangszeit plus dessen Dauer. Wähle passende Mitarbeiter und ein passendes Fahrzeug.

Wiederhole folgende Schritte, solange wie möglich:

- Bestimme alle Dienste, die zur ermittelten Untertourzeit (= nach dem letzten Dienst) möglich wären. Wähle darunter einen aus, der eine möglichst kleine zeitliche Lücke zum vorigen läßt und möglichst gleiche Anforderungen an Mitarbeiter und Fahrzeug hat. Bei mehreren gleichwertigen wird der am nächsten gelegene gewählt.

- passe gegebenenfalls Mitarbeiter und Fahrzeug so an, daß diese die Anforderungen aller Kunden auf der Untertour erfüllen

- Addiere zur Untertourzeit die Fahrzeit vom letzten Dienstort zum gewählten, sowie die Dienstdauer. (Alle Dienste, die zu dieser Untertourzeit bereits begonnen haben müßten, können hier weggelegt werden und werden bei der nächsten Untertour wieder berücksichtigt)

Damit ist eine Untertour ermittelt und mit den restlichen Dienstwünschen wird das Verfahren wiederholt, bis alle Kunden eingeplant sind.

Eine Verbesserung des Verfahrens läßt sich durch Parallelisierung der Untertourplanung erreichen: Wenn ein Dienst an keine Untertour angehängt werden kann, wird eine Paralleluntertour eröffnet, die mit anderen Mitarbeitern und mit einem anderen Fahrzeug gleichzeitig zu den bisherigen Untertouren ausgeführt werden soll. Im folgenden wird jeder Dienstwunsch an die Untertour angehängt, an die er hinsichtlich Entfernung der Stationen und Pausen (=„Löchern") zwischen den Diensten am besten paßt.

Ausgabe Eine Menge von Untertouren.

Eigenschaften

Vorteile Die Untertouren werden zeitlich zusammenhängend. Es entstehen keine unnötig großen Löcher zwischen einzelnen Diensten einer Untertour.

Nachteile Persönliche Vorlieben und Abneigungen der Kunden werden nicht berücksichtigt. Abneigungen könnten berücksichtigt werden, indem der jeweils nächste Dienst so gewählt wird, daß er zu den momentan gewählten Mitarbeitern paßt. Da diese sich aber im Laufe des Algorithmus ändern können, wird hier u.U. unnötig eingeschränkt und dadurch fallen mögliche Lösungen weg.

Anforderungen an die Qualifikation der Mitarbeiter werden kaum berücksichtigt. Dadurch können die Kunden mit hohen Ansprüchen so stark verteilt werden, daß nicht ausreichend qualifizierte Mitarbeiter für alle Untertouren zur Verfügung stehen.

Laufzeit Da nach jedem eingeplanten Dienstwunsch alle verbleibenden bezüglich Beginn und Fahrzeit zwischen den Stationen verglichen werden müssen, entsteht hier Aufwand $\mathcal{O}(w^2)$.

Positiv-Beispiel Sechs Dienste ohne besondere Qualifikationsanforderungen sind zu erledigen (z.B. Einkaufen, Kehrwoche). Alle Dienste dauern 45 Minuten, die Zwischenstrecken sind jeweils in 15 Minuten zurückzulegen:

Kunde	Dienst-beginn
A	16 Uhr
B	17 Uhr
C	17 Uhr
D	18 Uhr
E	18 Uhr
F	19 Uhr

Es werden zwei Untertouren zusammengestellt: ABDF und CE.

Negativ-Beispiel Ein Fachpfleger und ein Zivi stehen zur Verfügung, um folgende Dienstwünsche zu erfüllen (Fahrtzeiten werden hier vernachlässigt):

Kunde	Beginn	Tätigkeit	Dauer
A	13 Uhr	Einkaufen	1 h
B	14 Uhr	Wohnung putzen	1 h
C	15 Uhr	Kehrwoche	1 h
D	15:30 Uhr	Krankenpflege	2 h
E	18 Uhr	Einkaufen	1 h
F	18:30 Uhr	Spritzen	10 min

Die Dienstwünsche A, B und C werden zunächst dem Zivi zugeordnet. Für den Dienst D wird eine zweite Untertour eröffnet, die der Pfleger übernimmt. Da Dienst E zeitlich für letzteren günstiger liegt, führt er E nach D aus. Dadurch steht aber für Dienst F, der zeitlich mit E überlappt, kein qualifizierter Mitarbeiter zur Verfügung.

4.5.5.4 MSD: Auswahl nach Anforderung an Mitarbeiter

Hierfür müssen alle Dienstwünsche nach Anspruch an die Qualifikation der Mitarbeiter geordnet werden (siehe 4.5.4).

Algorithmus Es wird der anspruchsvollste noch nicht verplante Dienstwunsch gewählt (bei mehreren Möglichkeiten wird nach örtlicher Nähe zu einem bereits verplanten Dienst vorgegangen). Dieser wird in die begonnene Untertour integriert, indem er an der zeitlich passenden Stelle eingefügt wird. Gegebenenfalls werden hierzu die anderen Dienste im Rahmen der zeitlichen Toleranzen verschoben, um Platz zu machen. Sollte ein Dienstwunsch nicht in die Lücken der bisher gewählten passen, wird er zur Seite gelegt und der nächste probiert.

Dies wird wiederholt, bis sich kein Dienstwunsch mehr in die bis dahin ermittelte Untertour einreihen läßt. Mit den zur Seite gelegten sowie den bisher noch nicht betrachteten Dienstwünschen werden weitere Untertouren auf dieselbe Art zusammengestellt.

Auf ähnliche Weise könnten persönliche Vorlieben und Abneigungen von Kunden berücksichtigt werden: Zu einem Mitarbeiter werden zunächst Kunden gewählt, die gerne von diesem Mitarbeiter betreut werden wollen. Dann solche, die möglichst viele der Qualifikationen benötigen, die dieser Mitarbeiter hat.

Ausgabe Eine Menge von Untertouren.

Eigenschaften

Vorteile Auch hohe Anforderungen an die Qualifikation der Mitarbeiter können erfüllt werden (falls dies mit dem vorhandenen Personal überhaupt möglich ist).

Nachteile Es können Untertouren entstehen, die sehr weite Wege beinhalten. Die Nähe der Kunden spielt eine sehr untergeordnete Rolle. Außerdem werden die später entstehenden Untertouren immer schlechter (größere Lücken, weniger Dienste) und möglicherweise unnötig zahlreich (und daurch evtl. mit der vorhandenen Menge an Fahrzeugen nicht mehr fahrbar).

Laufzeit Im schlimmsten Fall müssen alle unverplanten Dienstwünsche mit der Untertour abgeglichen werden, bis dann jeweils der letzte dazu paßt. Es kann also zu $\mathcal{O}(w^2)$ Einfügeversuchen kommen.

Positiv-Beispiel Das Negativ-Beispiel aus 4.5.5.3 würde mit diesem Verfahren korrekt gelöst, da der Fachpfleger hier zunächst die Dienste D und F zugeordnet bekäme. Dadurch entstünde für den Zivi automatisch die Untertour ABCE, was auch die einzige Möglichkeit ist (und somit trotz großer Lücken das Optimum).

Negativ-Beispiel Zwei Zivis mit pflegerischer Grundausbildung stehen zur Verfügung, um folgende Dienstwünsche zu erfüllen (Fahrtzeiten werden wieder vernachlässigt):

Kunde	Beginn	Tätigkeit	Dauer
A	17 Uhr	Grundpflege	1 h
B	17:30 Uhr	Einkaufen	1 h
C	18 Uhr	Kehrwoche	1 h
D	18:30 Uhr	Grundpflege	1 h

Zivi 1 bekommt zunächst die beiden anspruchsvollsten Dienste zugeteilt (A und D). Dienstwunsch B und C können aber weder in die Untertour von Zivi 1 integriert werden, noch kann Zivi 2 sie alleine erledigen. Die korrekte Lösung (AC und BD) muß in diesem Fall gerade Dienste mit gleichen Anforderungen trennen, da sich sonst die Zeitvorgaben nicht erfüllen lassen.

4.5.5.5 Fahrdienste: Greedy-Ansatz über Anforderung an Fahrzeug

Die Kunden werden hinsichtlich ihrer Ansprüche an das Fahrzeug in verschiedene Klassen eingeteilt, z.B. 1. Rollstuhlfahrer; 2. Kunden, die besondere Sitze benötigen; 3. übrige Kunden. Ein konkretes Maß für die Stärke der Ansprüche wäre die Anzahl der Fahrzeuge, die diese Ansprüche erüllen können: Je weniger Fahrzeuge für einen Kunden in Frage kommen, desto höher sind seine Ansprüche in diesem Punkt.

Algorithmus Beginne mit der Kundenklasse, die die „höchsten Ansprüche“ hat. Im Beispiel wären das die Rollstuhlfahrer, da diese nur in einem Rollstuhlbus transportiert werden können, während z.B. ein Kind mit speziellem Kindersitz sowohl in einem Kleinbus als auch in einem Rollstuhlbus mit Sitzbank mitgenommen werden kann.

Diese Klassen werden nacheinander auf Fahrzeuge verteilt, wie im folgenden exemplarisch für die Rollstuhlfahrer angegeben:

1. Reserviere zunächst für die Rollstuhlfahrer eine ausreichende Menge Rollstuhlbusse

2. Verteile die Rollstuhlfahrer darauf:

   Beginnend mit einem beliebigen Kunden, wird immer derjenige Kunde als nächster dem Fahrzeug zugeordnet, der zu den Kunden, die schon im Fahrzeug sitzen, am nächsten wohnt. Ist ein Fahrzeug voll besetzt, wird mit einem neuen Fahrzeug und einem beliebigen übrigen Kunden fortgesetzt.

3. Fülle ggf. mit Kunden der nächsten Klasse auf, die am nächsten zu den bisher im Fahrzeug eingeplanten Kunden wohnen.

Ausgabe Eine Menge von „Fahrgemeinschaften“. Die Reihenfolge, in der die Kunden abgeholt werden, muß noch festgelegt werden. Dies kann auch Fahrzeugübergreifend innerhalb der verschiedenen Klassen geschehen.

Eigenschaften

Vorteile Anforderungen an das Fahrzeug werden gut berücksichtigt. Besondere Fahrzeuge (z.B. Rollstuhlbusse), die nur in geringer Anzahl vorhanden sind, werden zunächst an diejenigen Kunden verteilt, die diese auf jeden Fall benötigen.

Nachteile Anforderungen an die Qualifikationen der Mitarbeiter werden nicht berücksichtigt. Sollten diese so hoch sein, daß nicht ausreichend viele qualifizierte Mitarbeiter für die berechneten Untertouren zur Verfügung stehen, muß versucht werden, durch inkrementelle Verfahren (Umsetzen von Kunden in andere Fahrzeuge) Kunden mit ähnlichen Anforderungen zusammenzubringen.

Anforderungen an Zeiten werden nicht berücksichtigt. Diese können teilweise beim Festlegen der Reihenfolge der Stationen in jeder Untertour erfüllt werden.

Laufzeit Die möglichen Fahrzeuge für einen Kunden können bereits bei dessen Aufnahme in das System berechnet werden, und bleiben dann gültig, solange an den Kundendaten oder Fahrzeugen nichts geändert wird. Die anfängliche Einteilung in Klassen ist dann eine normale Sortieroperation mit Aufwand $\mathcal{O}(w \log(w))$.

Während der Aufteilung müssen bei jedem neu zu besetzenden Fahrzeugplatz alle noch nicht eingeplanten Kunden hinsichtlich der Nähe der Startorte mit den bereits im Fahrzeug verplanten verglichen werden. Aufwand also $\mathcal{O}(w^2)$.

Positiv-Beispiel Sechs Rollstuhlfahrer und sechs Gehbehinderte sollen befördert werden. Dazu stehen folgende Fahrzeuge zur Verfügung: Zwei Rollstuhlbusse (mit entweder vier Rollstuhlplätzen oder zwei Rollstuhlplätzen und drei Sitzplätzen), sowie drei PKW mit je drei Sitzplätzen. Da keine besonders qualifizierten Mitarbeiter benötigt werden, können diese mit dem obigen Algorithmus folgendermaßen aufgeteilt werden:

Rollstuhlbus 1	4 Rollstuhlfahrer
Rollstuhlbus 2	2 Rollstuhlfahrer und 3 Gehbehinderte
PKW	3 Gehbehinderte

Eine sinnvolle Abholreihenfolge muß für jedes Fahrzeug noch ermittelt werden, was sich aber mit den in der Praxis vorkommenden Fahrzeuggrößen recht effizient machen läßt.

Negativ-Beispiel Die sechs Rollstuhlfahrer und sechs Gehbehinderten aus dem vorigen Beispiel wohnen diesmal so verteilt, daß jeweils zwei Rollstuhlfahrer und zwei Gehbehinderte in einem Ort wohnen. Es stehen drei Rollstuhlbusse zur Verfügung.

Die Rollstuhlfahrer würden auf zwei Rollstuhlbusse verteilt und die Untertouren unnötig lang, da pro Teilort ein Rollstuhlbus mit Sitzbank geschickt werden könnte. Allerdings benötigt der betrachtete Algorithmus nur zwei Rollstuhlbusse und einen PKW gegenüber drei Rollstuhlbussen für die Streckenoptimierte Lösung.

4.5.5.6 Fahrdienste: Adaptiver Ansatz über Wohnortnähe

Hier werden die Kunden nach örtlicher Nähe ihres Startorts zu „Fahrgemeinschaften“ zusammengefaßt. Fahrzeuge und Mitarbeiter werden passend gewählt, soweit möglich.

Algorithmus Die folgenden Schritte werden wiederholt, bis alle Kunden untergebracht sind:

1. Die Kunden, die im momentan betrachteten Fahrzeug zusammen eingeplant werden, heißen *momentane Besetzung*. Die Menge der *möglichen Kunden* ergibt sich aus den noch nicht eingeplanten Kunden durch Entfernung derjenigen Kunden, die nicht mit der momentanen Besetzung zusammen fahren können, weil für die Summe der Anforderungen kein Fahrzeug oder kein passender Mitarbeiter gefunden werden kann.

2. Wähle unter den möglichen Kunden denjenigen, der zur momentanen Besetzung am nächsten wohnt. Füge diesen zur momentanen Besetzung hinzu.

3. Passe ggf. das Fahrzeug und die Mitarbeiter an, so daß alle Anforderungen der momentanen Besetzung dadurch erfüllt sind.

   Ist das Fahrzeug durch den neu hinzugekommen Kunden voll, wähle ein neues, sowie zwei Mitarbeiter. Diese sollten jeweils möglichst wenig Anforderungen erfüllen. Dadurch ist gewährleistet, daß Fahrzeug und Mitarbeiter, die am Ende der Untertour zugeordnet werden, nicht unnötig mehr als die geforderten Eigenschaften aufweisen.

Ausgabe Eine Menge von Fahrzeugbesetzungen. Die Reihenfolge jeder Untertour muß auch hier noch festgelegt werden, da die Reihenfolge, in der sie zusammengestellt wurde, nicht optimal sein muß.

Eigenschaften

Vorteile Die entstehenden Untertouren haben sehr kurze Wege zwischen den Stationen.

Nachteile Müssen sehr viele Anforderungen an Mitarbeiter und Fahrzeuge berücksichtigt werden, wird dieses Verfahren vermutlich keine Lösung liefern, da Kunden mit ähnlichen Ansprüchen zu stark auf verschiedene Fahrzeuge verteilt sind.

Anforderungen an Zeiten werden nicht berücksichtigt. Diese können teilweise beim Festlegen der Reihenfolge der Stationen in jeder Untertour erfüllt werden.

Laufzeit Bei jedem neu zu besetzenden Fahrzeugplatz müssen alle noch nicht eingeplanten Kunden hinsichtlich der Nähe der Startorte mit der momentanen Besetzung verglichen werden. Aufwand also $\mathcal{O}(w^2)$.

Positiv-Beispiel Eine Gruppe von behinderten Kindern aus verschiedenen Orten soll in die Schule gebracht werden. Manche Kinder brauchen einen Kindersitz, andere können normal transportiert werden, benötigen aber neben dem Fahrer eine Aufsichtsperson.

Es werden Untertouren geplant, auf denen jeweils sechs nahe beieinander wohnende Kinder mit einem Kleinbus abgeholt werden. Auf jeder Untertour ist ein zweiter Mitarbeiter dabei.

Negativ-Beispiel Sechs Rollstuhlfahrer und sechs Gehbehinderte sollen transportiert werden. Die Rollstuhlfahrer wohnen allerdings ziemlich weit entfernt voneinander, die Gehbehinderten jeweils in der Nähe eines Rollstuhlfahrers.

Es werden einige Fahrgemeinschaften gebildet, in denen jeweils ein bis zwei Rollstuhlfahrer mit drei Gehbehinderten kombiniert werden. Dabei werden aber schnell alle verfügbaren Rollstuhlbusse verplant, so daß nicht alle Rollstuhlfahrer untergebracht werden.

4.5.5.7 Individualfahrten

Hier werden aus Abrechnungsgründen keine Dienstwünsche zu Touren zusammengefaßt. Deshalb kann hier auch keine Touroptimierung stattfinden.

4.5.6 Generischer Ansatz

In den obigen Optimierungsideen wurden für die Planung einzelner Dienstarten unterschiedliche Vorgehensweisen bei der Zusammenstellung von Touren bzw. Untertouren gewählt. Dabei wurden meist nur einige Parameter der Dienstwünsche berücksichtigt. Aus den all diesen Heuristiken gemeinsamen Punkten läßt sich ein allgemeiner Ansatz ableiten, der versucht, alle Parameter zu berücksichtigen.

Im Mittelpunkt steht dabei eine Gütefunktion, die bewertet, wie gut ein Dienstwunsch zu einer Tour paßt.

4.5.6.1 Touren

Da einige wichtige Bewertungsgrößen nur aus Untertouren ermittelt werden können (z.B. setzt die Ermittlung und Bewertung von Fahrtzeiten eine konkrete Reihenfolge der Stationen voraus), ist die Grundlage der Bewertung die Gütefunktion auf Untertouren (Abschnitt 4.5.6.5). Daraus kann dann eine Gütefunktion für Touren abgeleitet werden, die angibt, wie gut ein Dienstwunsch D zu einer Tour T (einer Menge von Untertouren) paßt:

$$G(T,D) = \frac{1}{|T|} \cdot \sum_{U \in T} \gamma(U,D) \quad \text{mit} \quad \gamma(U,D) = \begin{cases} 0 & D \notin U \\ G(U,D) & D \in U \end{cases}$$

4.5.6.2 Untertouren

Jede Dienstart wird einzeln geplant. Dabei wird wochentagweise vorgegangen. Es werden regelmäßige Untertouren erstellt, die i.a. jede Woche gleich gefahren werden. Einzelne Ausnahmen von der Regelmäßigkeit (z.B. Dienste, die an einigen speziellen Tagen nicht ausgeführt werden sollen) werden hier nicht berücksichtigt, sondern erst bei der Bildung von Fahrten.

4.5.6.3 Einfügen von Dienstwünschen

Um mögliche Kombinationen von Untertour und Dienstwunsch zu bewerten, muß jedesmal ein Dienstwunsch in die Untertour eingefügt werden (in der Gütefunktion mit $U \cup D$ bezeichnet). Da hier zum einen die zeitlichen Vorgaben der Dienstwünsche berücksichtigt werden müssen, zum anderen aber die Fahrtstrecke zwischen den einzelnen Stationen möglichst optimal sein soll, müssen hierfür noch Heuristiken entwickelt werden, die diese Aufgabe möglichst effizient lösen.

4.5.6.4 Definitionen

Untertour Eine Untertour ist hier ein Tupel von Dienstwünschen

MA Die Menge aller Mitarbeiter

FZ Die Menge aller Fahrzeuge

QF Die Menge aller möglichen Mitarbeiter-Qualifikationen

DW Die Menge aller Dienstwünsche

$\mathbf{M_U}$ Der oder die Mitarbeiter, die der Untertour U zugeordnet sind

$\mathbf{F_U}$ Das Fahrzeug, das der Untertour U zugeordnet ist

4.5.6.5 Gütefunktion

Den Kern des generischen Algorithmus' bildet eine Gütefunktion, die ein Maß dafür angibt, wie gut ein neuer Dienstwunsch D zu einer in der Planung befindlichen Untertour U paßt. Diese Funktion bekommt eine Untertour und einen neuen Dienstwunsch als Eingabe und liefert eine Zahl zwischen 0 und 1 als Ergebnis. Je besser der Dienstwunsch zur Untertour paßt, desto höher die Bewertung.

Alle Teilfunktionen der Gütefunktion haben ebenfalls den Wertebereich $[0;1]$. Liefert eine Teilfunktion den Wert 0, muß sich auch für die gesamte Gütefunktion der Wert 0 ergeben, da der Wert 0 bedeutet, daß eine wesentliche Bedingung nicht erfüllt ist.

$$G(U,D) = \begin{cases} 0 & : \ \exists g_X = 0 \\ \begin{array}{l} (g_t * \Delta t_F(U,D) + g_M * M(D, M_U) + \\ g_F * F(D, F_U) + g_P * \Delta P(U \cup D)) \\ * \frac{1}{\sum g_X} \end{array} & : \ sonst \end{cases}$$

Dabei ist

g_X Gewichtungsfaktoren für die Teilfunktionen. Diese können vom Benutzer vorgegeben und verändert werden, um verschiedenen Bewertungsaspekten mehr/weniger Bedeutung zukommen zu lassen.

$\Delta t_F(U, D)$ Bewertung der zusätzlichen Fahrtdauer, die durch Hinzufügen des Dienstwunsches D zur Untertour U entsteht.

Um dies normieren zu können, muß eine Obergrenze für die Gesamtdauer einer Untertour festgelegt werden. Diese sei $t_{\max}$.

Die gesamte Fahrtdauer der Untertour U wird dann folgendermaßen bewertet:

$$t_F(U) = 1 - \frac{1}{t_{\max}} \cdot \sum_{i=1}^{|U|-1} t_F(D_i, D_{i+1})$$

Dabei bezeichnet $t_F(D_i, D_{i+1})$ die Fahrtzeit zwischen den Stationen D_i und D_{i+1}.

Damit ergibt sich die Bewertung der durch den Dienstwunsch D verursachten zusätzlichen Fahrtdauer als

$$\Delta t_F(U, D) = t_F(U \cup D) - t_F(U)$$

Berechnungszeit: $\mathcal{O}(|U \cup D|)$

$M(D, M_U)$ Bewertung der der Untertour U zugeordneten Mitarbeiter im Verhältnis zu den Anforderungen des neuen Dienstwunsches D.

Hierzu wird definiert:

$M \models Q \quad (M \in MA, Q \in QF) \iff$ Der Mitarbeiter M hat die Qualifikation Q.

$M \models D \quad (M \in MA, D \in DW) \iff$ Der Mitarbeiter M hat alle Qualifikationen, die der Dienstwunsch D fordert.

$D \leftrightarrow M \quad (M \in MA, D \in DW) \iff$ Der Mitarbeiter M ist bei der Person, die den Dienstwunsch D geäußert hat, nicht erwünscht.

$D \heartsuit M \quad (M \in MA, D \in DW) \iff$ Der Mitarbeiter M gehört zu den bevorzugten Mitarbeitern der Person, die den Dienstwunsch D geäußert hat.

$w_Q = 1 - \frac{|\{M \in MA | M \models Q\}|}{|MA|}$ ist der Wert einer Qualifikation $Q \in QF$. Er wird bei der Änderung von Mitarbeiter-Qualifikationen aktualisiert und gespeichert, muß also nicht bei jeder Anfrage neu berechnet werden.

$\Delta Q(D, M)$: Menge der „überzähligen" Qualifikationen, die Mitarbeiter M hat, die aber vom Dienstwunsch D nicht gefordert sind.

Damit können zwei Teilbewertungen berechnet werden:

$M_Q(D, M) = 1 - \frac{\sum_{Q \in \Delta Q(D,M)} w_Q}{\sum_{Q \in QF} w_Q}$

$M_V(D, M) = \begin{cases} 1 & D \heartsuit M \\ 0 & \text{sonst} \end{cases}$

Die Bewertung der Mitarbeiter ist damit

$$M(D, M) = \begin{cases} 0 & D \leftrightarrow M \vee M \not\models D \\ \alpha \cdot M_Q(D, M) + (1 - \alpha) \cdot M_V(D, M) & \text{sonst} \end{cases}$$

Durch den Faktor α kann festgelegt werden, wie stark die Vorlieben der Kunden im Verhältnis zur Eignung eines Mitarbeiters (und dem Interesse des DRK, keine hochqualifizierten Mitarbeiter für einfache Dienste zu „verschwenden“) berücksichtigt werden. Ein sinnvoller Wert könnte bei 0,8 liegen, da die durch Anforderungen an das Fahrzeug gegebenen Sachzwänge in der Praxis sicher stärker berücksichtigt werden müssen als Vorlieben der Kunden.

Berechnungszeit: $\mathcal{O}(|QF|)$

$F(D, F_U)$ Bewertung des der Untertour U zugeordneten Fahrzeugs im Verhältnis zu den Anforderungen des neuen Dienstwunsches D.

$F \models D \quad (F \in FZ, D \in DW) \iff$ Das Fahrzeug F kann die Anforderungen des Dienstwunsches D erfüllen.

$F < F' \quad (F, F' \in FZ) \iff |\{D \mid F \models D\}| < |\{D \mid F' \models D\}|$ (F kann weniger Dienstwünsche erfüllen als F')

$w_F \in [0; 1] \quad (F \in FZ)$: Wert des Fahrzeugs F, dieser wird vom Benutzer für die verschiedenen Fahrzeugtypen vorgegeben.

$$F(D, F) = \begin{cases} 0 & F \not\models D \\ 1 - w_F & \text{sonst} \end{cases}$$

Berechnungszeit: $\mathcal{O}(1)$

$\Delta P(U \cup D)$ Bewertung der Pausen, die in der Untertour U zusätzlich entstehen, wenn D aufgenommen wird.

Die Pausen der Untertour U werden folgendermaßen bewertet:

$$P(U) = 1 - \frac{1}{t_{\max}} \cdot \sum_{i=1}^{|T|-1} (\Delta t(D_i, D_{i+1}) - t_F(D_i, D_{i+1}))$$

Dabei ist $\Delta t(D_i, D_{i+1})$ die Zeit zwischen Erledigung des Dienstwunsches D_i und des darauffolgenden Dienstwunsches D_{i+1}, und $t_F(D_i, D_{i+1})$ die Fahrtzeit zwischen D_i und D_{i+1}.

Damit ergibt sich die Bewertung der durch den Dienstwunsch D verursachten zusätzlichen Pausen als

$$\Delta P(U \cup D) = \frac{1 + P(U \cup D) - P(U)}{2}$$

Berechnungszeit: $\mathcal{O}(|U \cup D|)$

Zwei weitere Gütefunktionen bewerten, wie gut eine Menge von Mitarbeitern (maximal zwei) bzw. ein Fahrzeug für eine Untertour geeignet ist:

$G_M(U)$ Bewertung der Mitarbeiter. Setzt sich zusammen aus den persönlichen Vorlieben und Abneigungen der Kunden auf der Untertour, sowie deren Anforderungen an die Qualifikation der Mitarbeiter.

$M_Q(U, M) = \frac{1}{|U|} \cdot \sum_{D \in U} M_Q(D, M)$

($M_Q(U,M) = 1 - \prod_{D\in U} \frac{|\{M'\in MA \mid M'<M \wedge M' \models D\}|}{|MA|}$ ist vermutlich exakter, aber aufwendiger zu berechnen)

$M_V(U,M) = \frac{1}{|U|} \cdot \sum_{D\in U} M_V(D,M)$

$$G_M(U) = \begin{cases} 0 & \exists D \in U : D \leftrightarrow M \vee M \not\models D \\ \alpha \cdot M_Q(U,M) + (1-\alpha)\cdot M_V(U,M) & \text{sonst} \end{cases}$$

α ist der Gewichtungsfaktor, der schon bei $M(D, M_U)$ definiert wurde.

Berechnungszeit: $\mathcal{O}(|U| \cdot |QF|)$

$G_F(U)$ Bewertung des Fahrzeugs. Berücksichtigt die Fahrzeugklassen, mit denen ein Kunde transportiert werden darf, sowie dessen sonstige Anforderungen an die Konfiguration.

$$G_F(U) = \begin{cases} 0 & \exists D \in U : F \not\models D \\ \frac{1}{|U|} \cdot \sum_{D\in U} F(D,F) & \text{sonst} \end{cases}$$

($G_F(U) \;=\; \begin{cases} 0 & \exists D \in U : F \not\models D \\ 1 - \prod_{D\in U} \frac{|\{F'\in FZ \mid F'<F \wedge F' \models D\}|}{|FZ|} & \text{sonst} \end{cases}$ wäre hier wohl exakter, aber aufwendig zu berechnen)

Berechnungszeit: $\mathcal{O}(|U|)$

4.5.6.6 Lineare Vorgehensweise

Der Algorithmus arbeitet nach dem Greedy-Prinzip und stellt eine Tour nach der anderen zusammen.

Sei W die Menge der einzuplanenden Dienstwünsche sowie T die gerade entstehende Tour.

while $W \neq \emptyset$
  Wähle beliebigen Dienstwunsch $D \in W$ (oBdA den anspruchsvollsten)
  Wähle F_T und M_T optimal bezüglich $G_F(T)$ bzw. $G_M(T)$.
  Kunde-gefunden := true
  $T := \{D\}; W := W \setminus \{D\}$
  while (Fahrzeug nicht voll) und (Tour nicht zu lang) und (Kunde-gefunden)
    Sortiere W bezüglich der Gütefunktion $G(T, D_i)$ für alle $D_i \in W$
    Kunde-gefunden := false
    for D_k := (bester Dienstwunsch)
    to (schlechtester Dienstwunsch $\geq$ Minimalschranke)
      if $\exists$ Mitarbeiter $M'_T \in$ MA und Fahrzeug $F'_T \in$ FZ,
        die die Tour $T \cup D_k$ ausführen können
      then
        $T := T \cup D_k; \quad W := W \setminus \{D_k\}$
        $M_T := M'_T; \quad F_T := F'_T$
        Kunde-gefunden := true
      end if
    end for

end while
Speichere Tour T
end while

Aufwandsabschätzung Sei w die Anzahl der Dienstwünsche, t die Anzahl der entstehenden Touren. Die äußere While-Schleife wird für jede Tour einmal ausgeführt: $\mathcal{O}(t)$. Die innere While-Schleife wird höchstens so oft ausgeführt, bis das Fahrzeug voll ist (im Falle von Fahrdiensten) oder die Tour insgesamt so lang wird, daß die Mitarbeiter ihr Arbeitszeitsoll erfüllt haben. Dies ist also durch eine Konstante abschätzbar.

Innerhalb der inneren While-Schleife muß für jeden Dienstwunsch die Güte berechnet werden: $\mathcal{O}(w)$.

Die innere For-Schleife wird im schlimmsten Fall $\mathcal{O}(w)$ mal ausgeführt.

Insgesamt wird also die Gütefunktion $\mathcal{O}(t \cdot w)$ mal aufgerufen, die Suche nach passenden Mitarbeitern und Fahrzeugen muß bis zu $\mathcal{O}(t \cdot w^2)$ mal durchgeführt werden.

4.5.6.7 Parallele Vorgehensweise

Ein anderer Ansatz stellt nicht eine Tour nach der anderen zusammen, sondern nimmt Dienste, die nicht zu einer Tour passen, als Ausgangspunkt für eine neue Tour. Jeder Dienstwunsch wird derjenigen Tour zugeordnet, zu der er bezüglich der Qualitätsfunktion am besten paßt.

Wähle beliebigen Dienstwunsch $D \in W$ (oBdA den anspruchsvollsten)
$T_1 := \{D\}; W := W \setminus \{D\}; t := 1$
Wähle M_{T_1} und F_{T_1} optimal bezüglich $G_M(T_1)$ bzw. $G_F(T_1)$.
while $W \neq \emptyset$
  forall $D_i \in W$
    forall $j = 1 \ldots t$
      berechne $G(T_j, D_i)$
    end forall
    $G_{\text{opt}}(D_i) := \min_j G(T_j, D_i)$
  end forall
  Wähle $D := \max_i D_i$ bezüglich G_{opt}.
  Prüfe Touren nach absteigendem $G(T_j, D)$
    if $T_j \cup D$ möglich (bzgl. MA und Fz) und $G(T_j, D) \geq$ Minimalschranke
    then
      $T_j := T_j \cup D$; Passe M_{T_j} und F_{T_j} an
    else
      $t := t + 1; T_t := \{D\}$
      Wähle M_{T_t} und F_{T_t} optimal bezüglich $G_M(T_t)$ bzw. $G_F(T_t)$.
    end if
  $W := W \setminus \{D\}$
end while

Aufwandsabschätzung Sei wieder w die Anzahl der Dienstwünsche, t die Anzahl der entstehenden Touren. Die äußere While-Schleife wird hier für jeden

Dienstwunsch einmal ausgeführt: $\mathcal{O}(w)$.

Die Gütefunktion muß für jede Kombination aus Dienstwunsch und Tour berechnet werden: $\mathcal{O}(t \cdot w)$.

Insgesamt finden also $\mathcal{O}(t \cdot w^2)$ Aufrufe der Gütefunktion statt, die Suche nach passenden Mitarbeitern und Fahrzeugen muß bis zu $\mathcal{O}(t \cdot w)$ mal durchgeführt werden.

4.5.6.8 Variante der parallelen Vorgehensweise

Eine Laufzeit von $\mathcal{O}(t \cdot w^2)$ Aufrufen der Gütefunktion kann bei realen Werten von etwa $w = 140$ und $t = 40$ bereits recht groß werden ($40 \cdot 140^2 = 784000$). Deshalb könnte die folgende Variante des obigen Algorithmus in der Praxis wertvoller sein. Dabei werden die Dienstwünsche nach Anfangszeit sortiert und dann in dieser Reihenfolge eingeplant.

Die Gewinne bei der Laufzeit werden allerdings mit vermutlich schlechteren Ergebnissen erkauft. Da die Dienstwünsche zunächst nur hinsichtlich ihrer Anfangszeit gewählt werden, werden hier möglicherweise Dienstwünsche zu Touren kombiniert, deren Bewertung nur knapp die Minimalschranke übersteigt. Im vorigen Ansatz wurde dagegen immer der bestpassende Dienstwunsch gewählt. Dies kann möglicherweise durch Anwendung eines inkrementellen Optimierungsverfahrens auf die berechneten Touren ausgeglichen werden.

Sortiere Dienstwünsche nach Anfangszeit
$t := 0$
while $W \neq \emptyset$
  $D :=$ frühester Dienstwunsch $\in W$
  Wähle $T_{\text{opt}} = \min_i T_i$ bezüglich $G(T_i, D)$
  if $T_{\text{opt}} \neq \emptyset \wedge G(T_{\text{opt}}, D) \geq$ Minimalschranke **then**
    $T_{\text{opt}} := T_{\text{opt}} \cup D$
    Passe $M_{T_{\text{opt}}}$ und $F_{T_{\text{opt}}}$ an
  else (keine passende Tour gefunden)
    $t := t + 1; T_t := \{D\}; W := W \setminus \{D\}$
    Wähle M_{T_t} und F_{T_t} optimal bezüglich $G_M(T_t)$ bzw. $G_F(T_t)$.
  end if
end while

Aufwandsabschätzung Die While-Schleife wird für jeden Dienstwunsch ausgeführt, die Gütefunktion muß hier aber für jeden Dienstwunsch höchstens t mal ausgeführt werden.

Es finden also $\mathcal{O}(t \cdot w)$ Aufrufe der Gütefunktion statt.

4.5.6.9 Einsatzbeispiele

Durch passende Gewichtung der Teilfunktionen bei der Bewertung der Untertouren können mit diesem generischen Algorithmus die den oben vorgeschlagenen speziellen Heuristiken zugrundeliegenden Ideen verwirklicht werden.

- Setzt man $g_W = 1$ und $g_M = g_F = g_A = g_P = 0$, wird immer derjenige Dienstwunsch zu einer Tour dazugenommen, der die gesamte Fahrtdauer (und damit auch den Weg) am wenigsten verlängert. Dies entspricht dem Ansatz aus 4.5.5.2

- Die starke Berücksichtigung der Anforderungen an das Fahrzeug in 4.5.5.5 kann durch ein entsprechend hohes Gewicht g_F erreicht werden. Somit könnte dieser Ansatz durch $g_W = g_F = 1$ und $g_M = g_A = g_P = 0$ simuliert werden.

4.6 Inkrementelle Optimierung

Die bisherigen Algorithmen gingen immer davon aus, daß eine Dienstart komplett automatisch optimiert werden soll, daß also noch keine Dienstwünsche in bereits vorhandenen Touren zusammengefaßt sind.

Der inkrementelle Ansatz geht hingegen von einer bestehenden Tourenmenge aus, in die weitere Dienstwünsche integriert werden sollen. Es können hiermit also neue Dienstwünsche einfach in das Tourensystem integriert werden, ohne eine komplette Neuplanung durchführen zu müssen. Eine weitere Anwendungsmöglichkeit besteht darin, durch nachträgliche Umgruppierung einzelner Dienstwünsche ein automatisch erzeugtes Tourensystem noch weiter zu verbessern.

4.6.1 Generischer Algorithmus zur Erweiterung von Touren

Da der in 4.5.6.8 vorgestellte Algorithmus letztlich nichts anderes tut, als Dienstwünsche auf Touren zu verteilen, kann er mit minimalen Änderungen so erweitert werden, daß damit eine Menge von Dienstwünschen auf eine Menge von Touren verteilt wird.
Eingabe:

- Eine Menge von Touren $\mathcal{T} = \{T_1, \dots, T_n\}$ einer Dienstart (ist diese leer, verhält sich der Algorithmus genau wie der in 4.5.6.8)
- Eine Menge W von Dienstwünschen derselben Diensart wie die Touren
- Eine Minimalgüte $g_{\min}$. Ergibt die Bewertung, wie gut ein Dienstwunsch D zu einer Tour T paßt, weniger als $g_{\min}$, darf der Dienstwunsch nicht in diese Tour integriert werden. In diesem Falle wird eine neue Tour begonnen, die nur diesen Dienstwunsch enthält.

Ausgabe: Erweiterte Tourenmenge $\mathcal{T}$

Sortiere Dienstwünsche nach Anfangszeit
$t := n$
while $W \neq \emptyset$
  $D :=$ erster Dienstwunsch $\in W$
  Wähle $T_{\text{opt}} = \min_i T_i$ bezüglich $G(T_i, D)$
  if $T_{\text{opt}} \neq \emptyset \wedge G(T_{\text{opt}}, D) \geq g_{\min}$ **then**
    $T_{\text{opt}} := T_{\text{opt}} \cup D$
    Passe $M_{T_{\text{opt}}}$ und $F_{T_{\text{opt}}}$ an
  else (keine passende Tour gefunden)
    $t := t + 1; T_t := \{D\}; W := W \setminus \{D\};$

Wähle M_{T_t} und F_{T_t} optimal bezüglich $G_M(T_t)$ bzw. $G_F(T_t)$.
$\mathcal{T} := \mathcal{T} \cup \{T_t\}$
end if
end while

4.6.2 Einfügen von Dienstwünschen in ein bestehendes Tourensystem

Neue Dienstwünsche können nun einfach in das bestehende Tourensystem integriert werden: Für jede Dienstart wird der obige Algorithmus aufgerufen, als Parameter werden alle bestehenden Touren dieser Dienstart sowie die neu einzufügenden Dienstwünsche übergeben.

Um noch bessere Touren zu erreichen, könnte man die Reihenfolge verändern, in der die Dienstwünsche in die Touren eingefügt werden. Anstatt nach der Anfangszeit könnten diese nach z.B. Anforderungen an Mitarbeiter sortiert werden (siehe 4.5.4). Denkbar wäre auch, den Algorithmus mit allen möglichen bzw. einer bestimmten Anzahl zufälliger Permutationen der Dienstwunschmenge aufzurufen. Man erhält dann mehrere Tourenmengen und kann davon die beste wählen (wie diese bestimmt werden kann, wird in 4.6.4 erläutert).

4.6.3 Inkrementelles Verbessern durch Umgruppieren von Dienstwünschen

Ein Versuch, eine Menge von Touren zu verbessern, besteht darin, diese Touren aufzulösen und die Dienstwünsche wie in 4.6.2 auf die restlichen Touren zu verteilen.

Sei $\mathcal{T}$ eine Menge von Touren, sowie $T_1, \ldots, T_i \in \mathcal{T}$ die Touren, die verteilt werden sollen. Dies wird durch folgenden Aufruf des Algorithmus' aus 4.6.1 realisiert:

$$\mathcal{T}' = \text{Optimiere}(\mathcal{T} \setminus \{T_1, \ldots, T_i\}, \{D \mid D \in \bigcup_{k=1}^{i} T_k\}, g_{\min})$$

Ist nun die Bewertung $G(\mathcal{T}') > G(\mathcal{T})$, wurde mit $\mathcal{T}'$ tatsächlich eine bessere Tourenmenge gefunden, die dann an Stelle von $\mathcal{T}$ benutzt werden kann.

4.6.4 Bewertung Tourenmengen

Um entscheiden zu können, ob sich eine Tourenmenge durch Umordnung von Dienstwünschen verbessert hat, muß eine Bewertungsfunktion vorliegen, mit der Mengen von Touren verglichen werden können.

4.6.4.1 Bewertung einzelner Touren

Zunächst einmal müssen einzelne Touren bewertet werden können. Dies geht ganz analog zu 4.5.6.1 durch Mittelwertbildung über die Untertourgüten:

$$G(T) = \frac{1}{|T|} \cdot \sum_{U \in T} \gamma(U) \quad \text{mit} \quad \gamma(U) = \begin{cases} 0 & D \notin U \\ G(U) & D \in U \end{cases}$$

4.6.4.2 Bewertung einzelner Untertouren

Da in 4.5.6.5 bereits eine Gütefunktion angegeben wurde, die angibt, wie gut ein Dienstwunsch zu einer Untertour paßt, läßt sich die Bewertung einer fertigen Untertour leicht aus deren Komponenten ableiten:

$$G(U) = (g_M * G_M(U) + g_F * G_F(U) + g_A * A(U) + g_P * P(U)) \cdot \frac{1}{\sum g_X}$$

Dabei ist

g_X Gewichtungsfaktoren für die einzelnen Teilfunktionen

$G_M(T)$ Bewertung des/der Mitarbeiter, die der Untertour U zugeordnet sind (wie in 4.5.6.5 definiert).

Berechnungszeit: $\mathcal{O}(|U| \cdot |QF|)$

$G_F(U)$ Bewertung des Fahrzeugs, das der Untertour U zugeordnet ist (wie in 4.5.6.5 definiert).

Berechnungszeit: $\mathcal{O}(|U|)$

$A(U, F_U)$ Bewertung der Auslastung des Fahrzeugs im Verhältnis zu seiner Kapazität:

$$A(U, F) = \frac{|U|}{K(F)}$$

Dabei ist K(F) die Kapazität des Fahrzeugs (=maximal mögliche Fahrgastzahl ohne Mitarbeiter).

Berechnungszeit: $\mathcal{O}(1)$

$P(U)$ Bewertung der Pausen zwischen den Stationen einer Untertour (wie in 4.5.6.5 definiert).

Berechnungszeit: $\mathcal{O}(|U|)$

4.6.4.3 Bewertung einer Tourmenge

Sei $\mathcal{T} = \{T_1 \ldots, T_n\}$ eine Menge von Touren. Diese soll hinsichtlich der Kriterien aus 4.5.2 sowie der Güte der einzelnen Touren bewertet werden. Der Einfluß der verschiedenen, sich teilweise widersprechenden Kriterien kann durch Gewichtungsfaktoren angepaßt werden.

$$G(\mathcal{T}) = (\alpha \cdot \frac{1}{n} + \beta \cdot \sum_{i=1}^{n} t(T_i) + \gamma \cdot \sum_{i=1}^{n} G(T_i)) \cdot \frac{1}{\alpha + \beta + \gamma}$$

Dabei ist

α Gewichtung der Touranzahl

β Gewichtung der Bewertung der Tourdauer (=Einsatzzeit von Mitarbeitern und Fahrzeugen). Diese hängt von den unterschiedlichen Dauern der Untertouren der Tour T ab: $t(T) = \frac{1}{|T|} \cdot \sum_{U \in T} t(U)$

γ Gewichtung der Einzeltourgüten

Um die Optimierungskriterien gemäß 4.5.2 zu berücksichtigen, könnten bei 10 Dienstwünschen Werte von etwa $\alpha = 20, \beta = 3, \gamma = 2$ benutzt werden.

4.7 Die Benutzungsoberfläche

Bereits unmittelbar nach Abschluß der Anforderungsanalyse wurde parallel zur Spezifikation ein Prototyp der Benutzungsoberfläche entwickelt. Dieser wurde dem Kunden am 13.2.98 vorgeführt und stellt auf anschauliche Art die Zusammenhänge im Datenmodell sowie die geplanten Funktionen dar. Insbesondere kann aus den Bildschirmmasken genau entnommen werden, welche Daten der Anwender eingeben muß und was das Programm ausgeben und berechnen soll. Der Prototyp erhebt jedoch keinen Anspruch auf Vollständigkeit, da spätere Veränderungen am Datenmodell noch nicht in den Prototyp mit aufgenommen wurden.

4.7.1 Konsistenztests

Konsistenztests werden durchgeführt, wenn die Eingabe einer Maske abgeschlossen wurde. Welche Konsistenztests durchgeführt werden, ist bei den einzelnen Masken selbst beschrieben. (Zu Konsistenztest siehe auch Abschnitt 3.3.3 in der Anforderungsanalyse.)

4.7.2 Hauptmenu

Die wesentlichen Funktionen können über ein Menu aufgerufen werden, das aus verschiedenen, thematisch geordneten Untermenus besteht:

Daten Schnittstelle zum persistenten Speicher. Hier können Kunden, Touren, Ressourcen und Einstellungen gespeichert und wieder geladen werden.

Kunden Pflege des Kundenbestands, Erfassung von neuen Kunden und deren Dienstwünschen.

Touren Ansicht und Eingabe der Touren und Untertouren, automatische Berechnung neuer Touren und Untertouren

Datenausgabe Ausgabe von Plänen und Übersichtsdiagrammen auf Bildschirm oder Drucker

Analyse Diverse Funktionen um Fahrzeug- und Ressourcenbelegung und -auslastung zu ermitteln und darzustellen.

Ressourcen Verwaltung der Mitarbeiter und Fahrzeuge

Einstellungen Definition verschiedener grundlegender Daten: Mögliche Mitarbeiterqualifikationen, Fahrzeugtypen und -konfigurationen, Arten von Sitzhilfen und angebotene Essensarten. Außerdem können hier Parameter für die Bewertung von Touren bei der automatischen Planung sowie für die Berechnung von Fahrtzeiten und -kosten aus den Streckendaten definiert werden.

4.7.3 Dienste anfordern

In den Masken gibt es Ein- und Ausgabefelder. Während der Benutzer Eintragungen in den Eingabefeldern vornimmt, werden in den Ausgabefeldern Informationen aus den Stammdaten angezeigt, die u.U. aus den Eingabefeldern berechnet werden.

Zur Anforderung eines Dienstes kann es erforderlich sein, daß eine oder mehrere Kontaktpersonen (siehe 4.7.11) angegeben werden, die später auf den Tourplänen erscheinen sollen.

4.7.3.1 Essen auf Rädern

Die zugehörige Eingabemaske zeigt Abbildung 4.9.

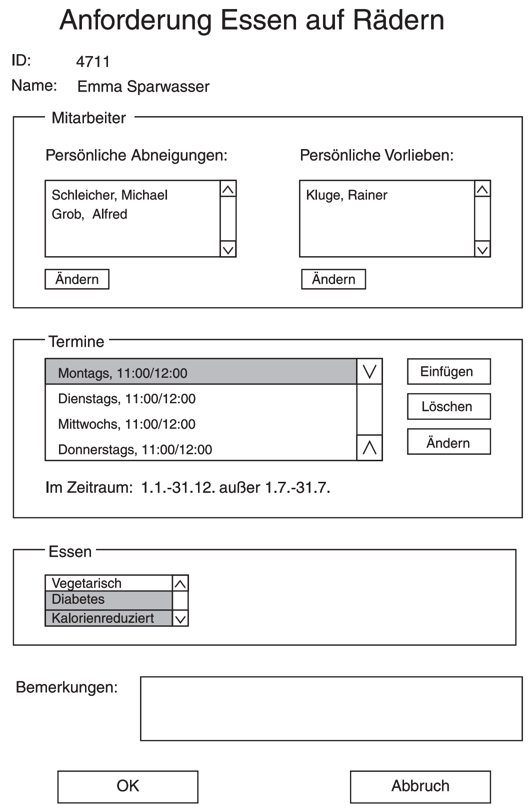


Abbildung 4.9: Dienstwunsch Essen auf Rädern

Eingaben des Benutzers

- Eine Menge von Terminen, an denen der Dienst stattfinden soll (siehe 4.7.3.5).
- Die Essensart. Diese kann aus einer Liste mit vorgegeben Essensarten gewählt werden, die z.B. verschiedene Diätessen enthält
- Bemerkungen. Hier kann ein beliebiger Text eingegeben werden, der dann im Tourplan erscheint und dem Mitarbeiter zusätzliche Informationen geben kann.

Ausgaben des Programms

- Kundennummer und -name. Für wen wird hier ein Dienst bestellt?

- Die persönlichen Zu- und Abneigungen des Kunden. (Diese Angaben sind auch änderbar)

Änderungen und Ergänzungen

- Auswahl, ob es sich um warmes oder kaltes Essen handelt.
- Es wird nicht die Art eines Essens bestimmt (durch Auswahl verschiedener kombinierbarer Eigenschaften), sondern Anzahl und Art der unterschiedlichen Essen (aus einer Auswahlliste mit lieferbaren Essensarten)

4.7.3.2 MSD/APD

Die zugehörige Eingabemaske zeigt Abbildung 4.10.

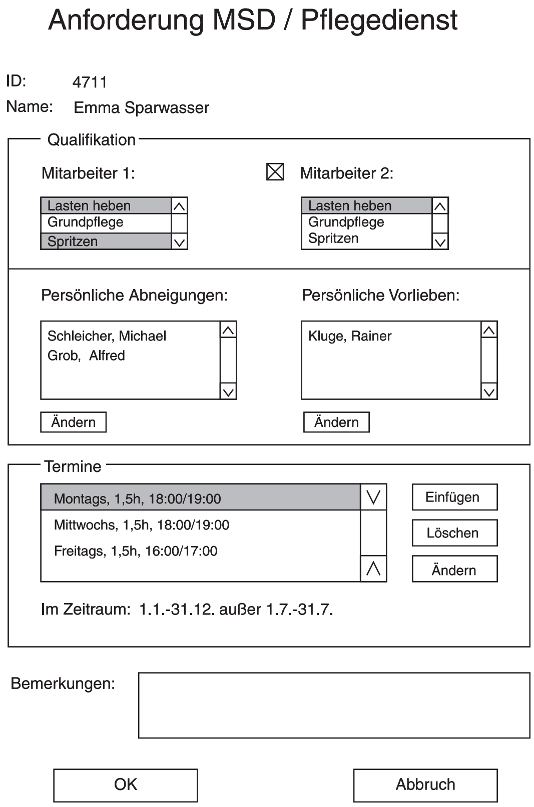


Abbildung 4.10: Dienstwunsch MSD/APD

Eingaben des Benutzers

- Ist ein zweiter Mitarbeiter für den Dienst nötig?
- Die erforderlichen Qualifikationen der Mitarbeiter für diesen Dienst
- Eine Menge von Terminen

- Bemerkungen

Ausgaben des Programms

- Kundennummer und -name
- Persönliche Zu- und Abneigungen des Kunden

Bemerkungen Bei der Auswahl der Qualifikationen der einzelnen Mitarbeiter kann nicht angegeben werden, daß einer der beiden Mitarbeiter die benötigte Qualifikation besitzen muß, sondern die Qualifikation hängt fest an dem Mitarbeiter, für den sie angegeben wurde. Diese Lösung schränkt zwar die Allgemeinheit ein, ist aber leichter zu handhaben.

4.7.3.3 Individualfahrt

Die zugehörige Eingabemaske zeigt Abbildung 4.11.

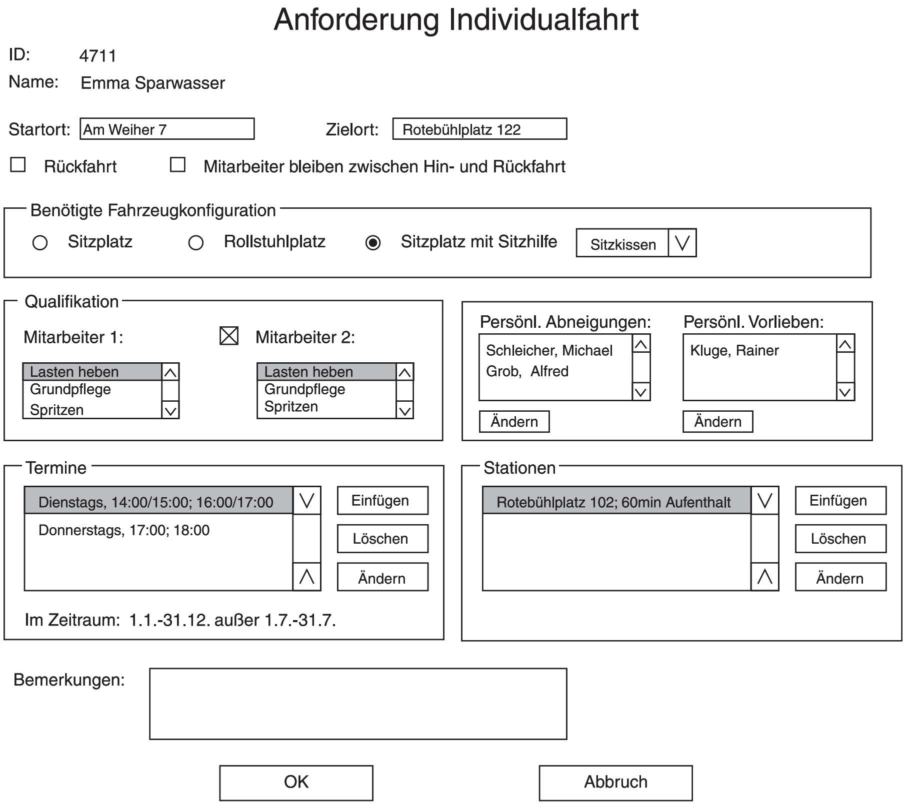


Abbildung 4.11: Dienstwunsch Individualfahrt

Eingaben des Benutzers

- Start- und Zielort der Fahrt
- Soll auch eine Rückfahrt vom DRK durchgeführt werden?

- Müssen die Mitarbeiter zwischen Hin- und Rückfahrt bleiben, z.B. um dem Kunden am Ziel bei seinen Tätigkeiten zu helfen?
- Ansprüche an die Konfiguration des Fahrzeugs (siehe 4.7.7.1), z.B. Rollstuhlplatz benötigt.
- Die erforderlichen Qualifikationen der Mitarbeiter für diesen Dienstwunsch
- Eine Menge von Terminen
- Eine Menge von Zwischenstationen, die auf der Fahrt angefahren werden sollen. Diese werden durch Adresse und Aufenthaltsdauer bestimmt.
- Bemerkungen

Ausgaben des Programms

- Kundennummer und -name
- Persönliche Zu- und Abneigungen des Kunden

Änderungen und Ergänzungen

- Die Angabe, ob eine Rückfahrt erwünscht ist, wurde herausgenommen. Diese Entscheidung stellt keine Einschränkung dar und dient zur einfacheren Handhabung des Systems. Wird trotzdem eine Rückfahrt gewünscht, so läßt sich diese durch Anhängen der Zieladresse an die Stationenliste modellieren.
- Die Angabe, ob die Mitarbeiter zwischen Hin- und Rückfahrt bleiben, wurde zur einfacheren Bedienung in die Eingabemaske für die Termine verschoben.

4.7.3.4 Schulfahrt, Dialysefahrt, Tagespflege

Für diese Dienste werden die selben Informationen eingegeben und angezeigt, die bereits für Individualfahrten im vorigen Abschnitt beschrieben wurden. Nur die Liste mit den Zwischenstationen kann hier entfallen.

4.7.3.5 Termine

Termine beschreiben, an welchen Tagen, zu welcher Zeit und in welchem Rhythmus ein Dienst ausgeführt wird. Die Eingabemaske für einen Fahrdienst zeigt Abbildung 4.12.

Eingaben des Benutzers

Zeitraum Eine Menge von Zeitspannen, innerhalb derer (ggf. eingeschränkt durch die Häufigkeit) der Dienst stattfindet. Der Zeitraum wird durch zwei Angaben charakterisiert:

Zeitrahmen Ein umschließender Zeitraum (z.B. Schuljahr)

Ausnahmen Zeiten innerhalb des Zeitrahmens, in denen der Dienst nicht stattfinden soll (z.B. Schulferien)

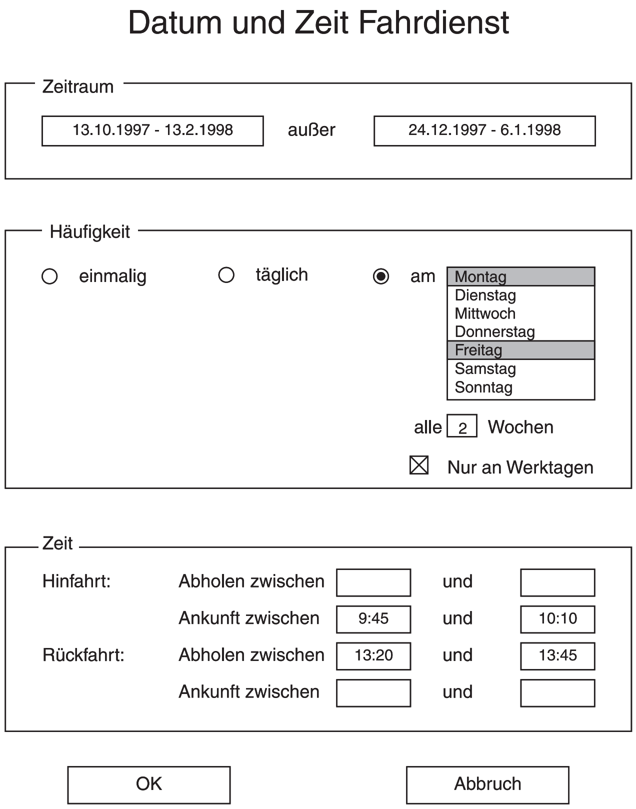


Abbildung 4.12: Termin für einen Fahrdienst

Beides kann sowohl ein längerer Zeitraum (wie Schuljahr und Ferien) sein, als auch eine Menge einzeln angegebener Tage (um einen Dienst an exakt diesen Tagen zu beschreiben).

Häufigkeit Für regelmäßige Dienste, die sich innerhalb des angegebenen Zeitraums periodisch wiederholen, kann hier ein Wiederholungsrhythmus angegeben werden:

einmalig Genau einmal (hier sollte der Zeitraum nur einen Tag umfassen)

täglich Jeden Tag innerhalb des angegebenen Zeitraums

regelmäßig Hier können diejenigen Wochentage gewählt werden, an denen der Dienst stattfinden soll. Zusätzlich kann angegeben werden, ob er nur an Werktagen stattfinden soll oder auch an Feiertagen, die auf einen Werktag fallen. Neben wöchentlicher Wiederholung kann ein beliebiger n-Wochen-Rhythmus angegeben werden.

Zeit Hier werden die Uhrzeiten erfaßt, an denen der Dienst an den angegebenen Tagen stattfindet. Abhängig von der Art des Dienstes müssen verschiedenen Information eingegeben werden:

Aufenthalt Hierunter fallen Dienste, bei denen Mitarbeiter zu einem Kunden fahren und dort eine gewisse Zeit verbringen, um für den

Kunden oder mit ihm verschiedene Tätigkeiten zu verrichten (z.B. MSD/APD). Nötige Angaben sind eine Zeit für den Beginn des Dienstes (optional kann auch über eine Zeitspanne der frühestmögliche/späteste Abhol- bzw. Ankunftstermin angegeben werden), sowie die Dauer.

Transport Fahrten, bei denen der Kunde an einem Ort abgeholt und zu einem anderen Ort befördert wird, brauchen folgende Angaben: Einen Zeitpunkt für den Start und für die Ankunft (optional kann auch hier eine Zeitspanne angegeben werden). Ist eine Rückfahrt erwünscht, müssen dafür dieselben Daten erfaßt werden.

Lieferung Für eine Lieferung, bei der dem Kunden eine Ware in die Wohnung gebracht wird (z.B. Essen auf Rädern), muß die Zeit(spanne) für die Ankunft beim Kunden festgehalten werden.

Änderunge und Ergänzungen

- Wie in 4.7.3.3 schon erläutert, wurde die Angabe, ob die Mitarbeiter zwischen Hin- und Rückfahrt bleiben, in diese Maske verschoben.
- Der Benutzer kann angeben, ob das System eine Warnmeldung ausgeben soll, wenn der Dienstwunsch auf einen Feiertag fällt (damit entschieden werden kann, ob der Dienst verlegt werden soll). Diese Warnung wird aktiviert, wenn aus Untertouren Fahrten gebildet werden (z.B. beim Erstellen der Dienstpläne). Dazu wird aber eine Eingabemöglichkeit für Feiertage benötigt.

4.7.4 Obertour eingeben

Die zugehörige Eingabemaske zeigt Abbildung 4.13.

Eingaben des Benutzers

- Die Tournummer, falls die vom System erzeugte nicht erwünscht ist
- Die Art des Dienstes
- Der oder die Mitarbeiter, die normalerweise die Fahrten fahren, die aus dieser Tour erzeugt werden.
- Das Fahrzeug, mit dem die Touren in der Regel gefahren werden
- Eine Menge von Dienstwünschen. Die Reihenfolge der Stationen spielt hier keine Rolle.

Außerdem kann der Benutzer die gesamte Tour oder einzelne Stationen für die Optimierung sperren.

Ausgaben des Programms

- Die Tournummer. Dies ist eine Vorgabe, die der Benutzer ändern kann.
- Die Qualifikation der gewählten Mitarbeiter
- Anforderungen der auf der Tour bedienten Kunden bezüglich der Mitarbeiter. Damit kann der Benutzer bereits bei der Eingabe prüfen, ob die vorgesehenen Mitarbeiter zu den Kunden passen.

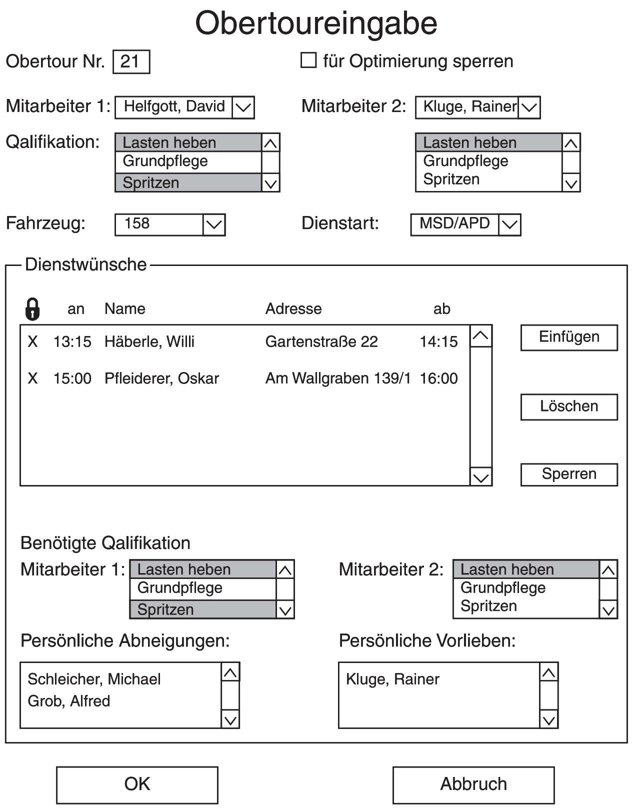


Abbildung 4.13: Eingabe einer Obertour

Konsistenztest Es wird geprüft, ob die Qualifikationen der eingegebenen Mitarbeiter mit den von den Kunden geforderten Qualifikationen übereinstimmen. Ebenso wird das Fahrzeug mit den Anforderungen der Dienstwünsche abgeglichen.

4.7.5 Untertour eingeben

Die zugehörige Eingabemaske zeigt Abbildung 4.14.

Eingaben des Benutzers

- Die Untertournummer, falls die vom System erzeugte nicht erwünscht ist
- Die Wochentage, an denen die Tour gefahren wird
- Die Abfahrtszeit und den Abfahrtsort
- Die Reihenfolge der Stationen der Tour
- Ob das Fahrzeug vor der nächsten Tour wieder zur Dienststelle zurück muß

Außerdem kann der Benutzer die gesamte Untertour oder einzelne Dienstwünsche für die Optimierung sperren.

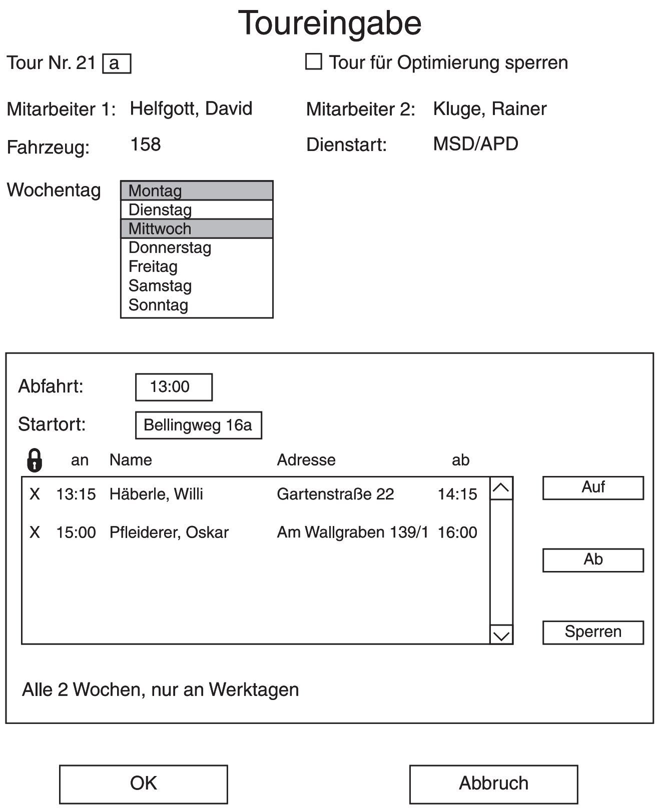


Abbildung 4.14: Eingabe einer Untertour

Ausgaben des Programms

- Die Tournummer. Dies ist eine Vorgabe, die der Benutzer ändern kann.
- Die Art des Dienstes
- Der oder die Mitarbeiter, die die Tour fahren sollen
- Das Fahrzeug, mit dem die Tour gefahren werden soll
- Die Dienstwüsche, die in der Tour definiert wurden.
- Der aus den Dienstwünschen ermittelte Wochenrhythmus und ggf. die Einschränkung auf Werktage.

4.7.6 Mitarbeiter eingeben

Die zugehörige Eingabemaske zeigt Abbildung 4.15.

Eingaben des Benutzers

- Persönliche Daten des Mitarbeiters: Name, Adresse, Geburtsdatum, Telephonnummer
- Eintrittsdatum

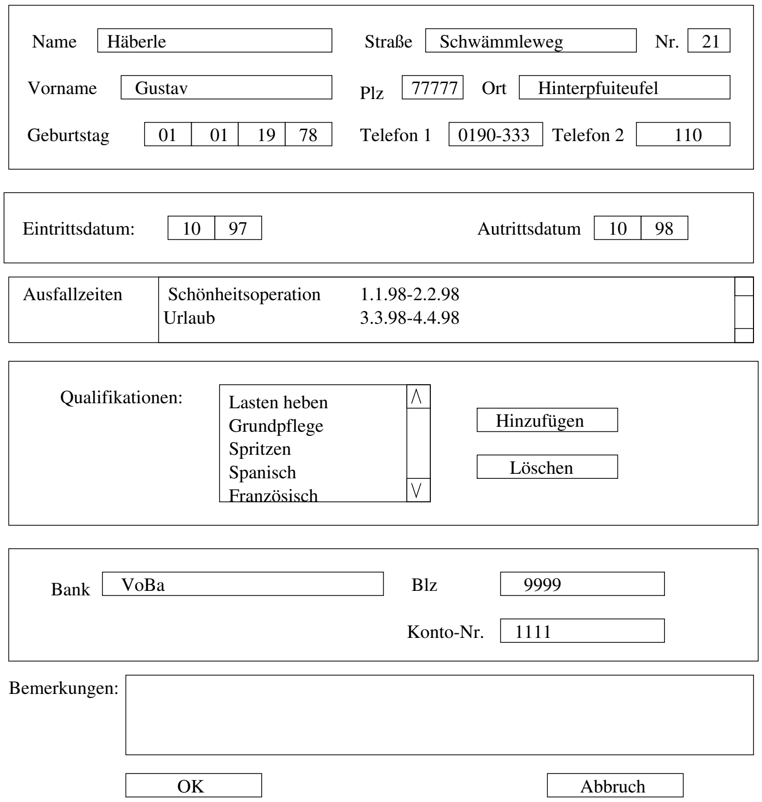


Abbildung 4.15: Mitarbeiter eingeben

- Austrittsdatum
- Ausfallzeiten: Eine Liste mit Zeiträumen, an denen der Mitarbeiter nicht zu Diensten eingeteilt werden kann.
- Qualifikationen: Aus einer Liste mit Qualifikationen, die für die Dienstplanung eine Rolle spielen, kann hier gewählt werden, welche Qualifikationen ein Mitarbeiter hat
- Die Bankverbindung des Mitarbeiters
- Bemerkung

4.7.7 Fahrzeug eingeben

Die zugehörige Eingabemaske zeigt Abbildung 4.16.

Eingaben des Benutzers

- Modell
- amtliches Kennzeichen
- interne Fahrzeugnummer
- Anzahl der Kfz-Schlüssel, die das DRK besitzt

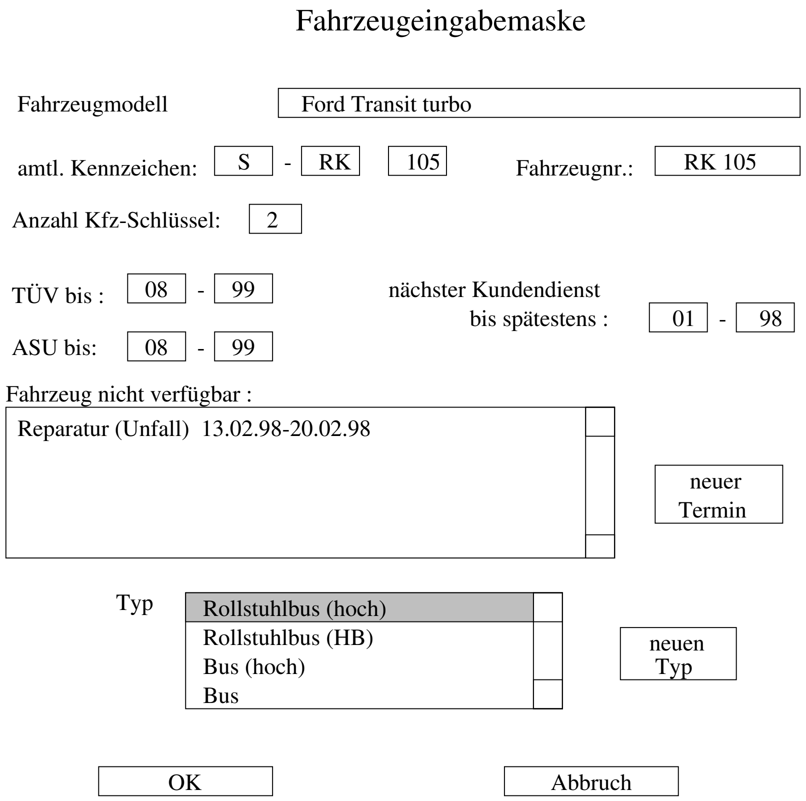


Abbildung 4.16: Fahrzeug eingeben

- Termine, wann das Fahrzeug in die Werkstatt muß (und damit nicht für die mobilen Dienste zur Verfügung steht): HU, ASU, Kundendienst
- Fahrzeugtyp. Aus einer Liste, die eine für die Planung wichtige Einteilung der Fahrzeuge in verschiedene Typen liefert, muß ein Typ gewählt werden. Diese Typenliste kann ggf. auch geändert bzw. erweitert werden. (Siehe 4.7.8)
- Aktuelle Konfiguration. Eine der für den Fahrzeugtyp möglichen Konfigurationen (siehe 4.7.7.1). Darüberhinaus die Anzahl der verschiedenen Sitzhilfen, die sich momentan im Fahrzeug befinden.

4.7.7.1 Fahrzeugkonfiguration

Eine Konfiguration beschreibt eine von mehreren Umbaumöglichkeiten eines Fahrzeugs. So können z.B. durch Ausbau einer Sitzbank zusätzliche Rollstuhlplätze geschaffen werden.

Eingaben des Benutzers

- Anzahl der Sitzplätze

- Anzahl der Sitzplätze mit fester Sitzhilfe (z.B. eingebauter Kindersitz)
- Anzahl der Rollstuhlplätze
- Sind Schienen für einen Kühlcontainer vorhanden (in dem tiefgekühltes Essen ausgeliefert wird)?

4.7.8 Fahrzeugtyp eingeben

Der Fahrzeugtyp beschreibt die Zusammenfassung mehrerer Fahrzeugmodelle und deren Konfigurartion unter einem einheitlichen Namen. Die Konfigurationen einzelner Modelle unter der gleichen Typbezeichnung müssen die gleichen sein.

Eingaben des Benutzers

- Typbezeichnung: Eindeutiger Bezeichner, der den Fahrzeugtyp von allen anderen unterscheidet.
- Beschreibung: Beliebiger Text, der die Typbezeichnung ergänzen kann.
- Kilometerpreis: Der Preis mit einem Fahrzeug dieses Typs gefahrenen Kilometers.

4.7.9 Kunden eingeben

Die zugehörige Maske zeigt Abbildung 4.17

Eingaben des Benutzers

- Persönliche Daten: Name, Vorname, Geburtsdatum, Adresse, Telefon
- Schlüssel: Anzahl der Hausschlüssel, zusätzlich kann optional ein Bezeichner für den Hausschlüssel vergeben werden (z.B. die Kundennummer).
- Rechnungsempfänger : siehe 4.7.10
- Rollstuhl: Angabe, ob der Kunde einen Rollstuhl besitzt.
- Hilfsmittel: ist aus einer Liste auszuwählen. Es kann auch festgelegt werden, ob dieses benötigte Hilfsmittel vom DRK gestellt werden muß, oder ob es der Kunde selbst mitbringt. Diese Liste kann auch erweitert und geändert werden.
- Fahrtdauer: Maximale Zeit, die der Kunde im Fahrzeug verbringen soll. Unabhängig davon existiert eine allgemeine Obergrenze (beim DRK momentan zwei Stunden), die für alle Kunden gilt und auf keinen Fall überschritten werden darf.
- Passender Fahrzeugtyp: Auswahl und Freigabe aus der Liste der definierten Fahrzeugtypen, mit denen der Kunde transportiert werden kann. (Siehe 4.7.8)
- Kontaktpersonen: siehe 4.7.11
- Bemerkungen
- persönliche Vorlieben und Abneigungen: neu, ändern und löschen

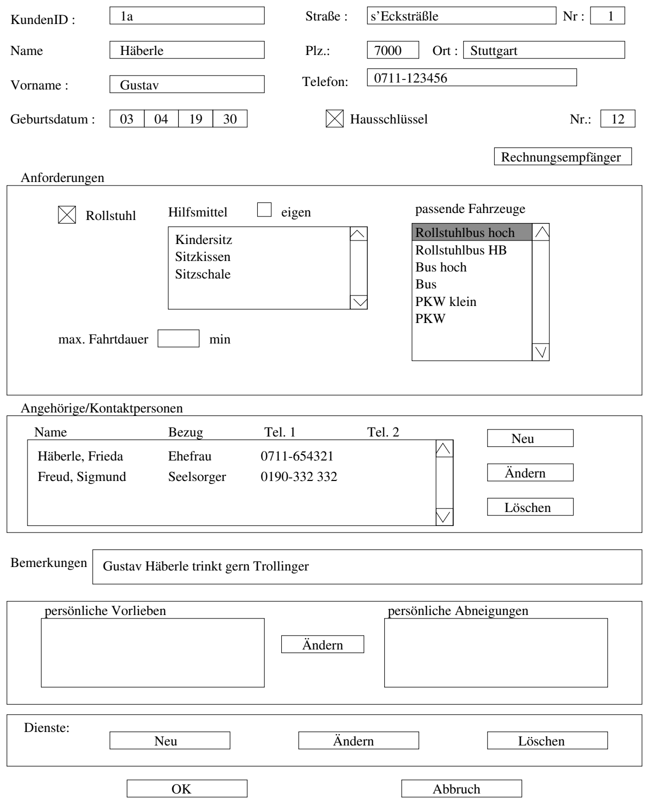


Abbildung 4.17: Kunden eingeben

- Dienstwünsche: neu, ändern und löschen

Ausgaben des Programmes

- Kundennummer: Wird vom Programm vorgegeben, kann aber vom Benutzer durch eine andere Kundennummmer ersetzt werden.
- Listen der schon bekannten Rechnungsempfänger, Kontaktpersonen und Dienstwünsche.

4.7.10 Rechnungsempfänger eingeben

Unter Rechnungsempfängeren versteht man diejenigen Personen oder Institutionen, die für die Bezahlung eines oder mehrerer Dienstwünsche aufkommen. Der Kunde selbst kann dabei ebenfalls als Rechnungsempfänger auftreten. Ebenso kann der Dienstwunsch eines Kunden mehrere Rechnungsempfänger haben.

Eingaben des Benutzers

- Name des Rechnungsempfängers
- Adresse und Telefon

- Ansprechpartner: z.B. bei Institutionen der Name des zuständigen Sachbearbeiters
- Bankverbindung: Name der Bank, BLZ und Kontonummer
- Dienstwunsch: Auswahl aus einer Liste, die die vom Kunden geäußerten Dienstwünsche umfaßt
- Bemerkung

4.7.11 Kontaktperson eingeben

Bezugspersonen sind Personen, die bei unvorhergesehenen Ereignissen benachrichtigt werden sollen, z.B. Eltern, Hausarzt, ...

Eingaben des Benutzers

- Name
- Adresse und Telefon
- Art des Bezugs: Bezeichnung, in welchem Verhältnis die Bezugsperson zum Kunden steht (z.B. Mutter oder Hausarzt)
- Bemerkung

Kapitel 5

Entwurf

5.1 Grobentwurf

5.1.1 Überblick

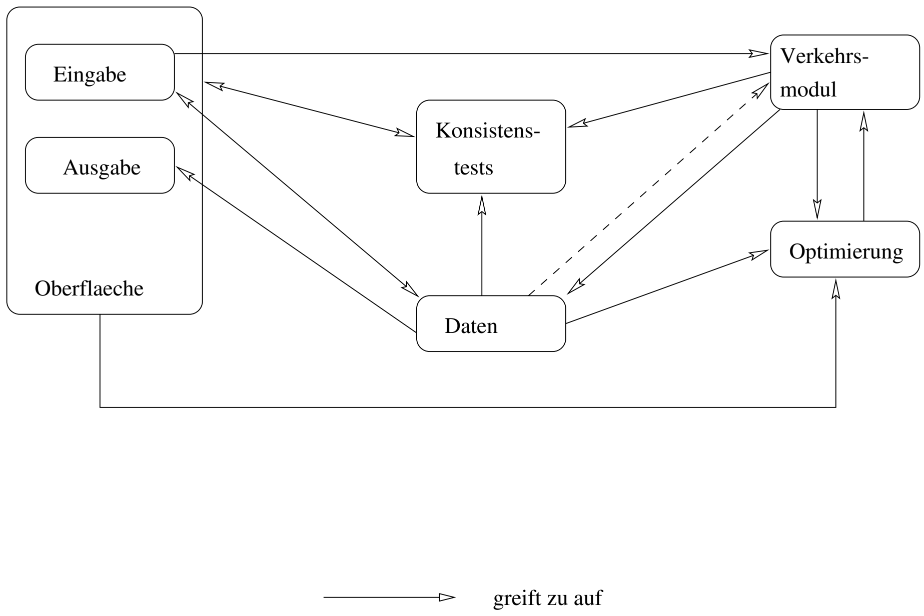


Abbildung 5.1: Module des Systems

Abbildung 5.1.1 zeigt die wesentlichen Funktionseinheiten des Systems und deren Beziehungen untereinander.

5.1.2 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.

5.1.3 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 5.2).

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 5.3)

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

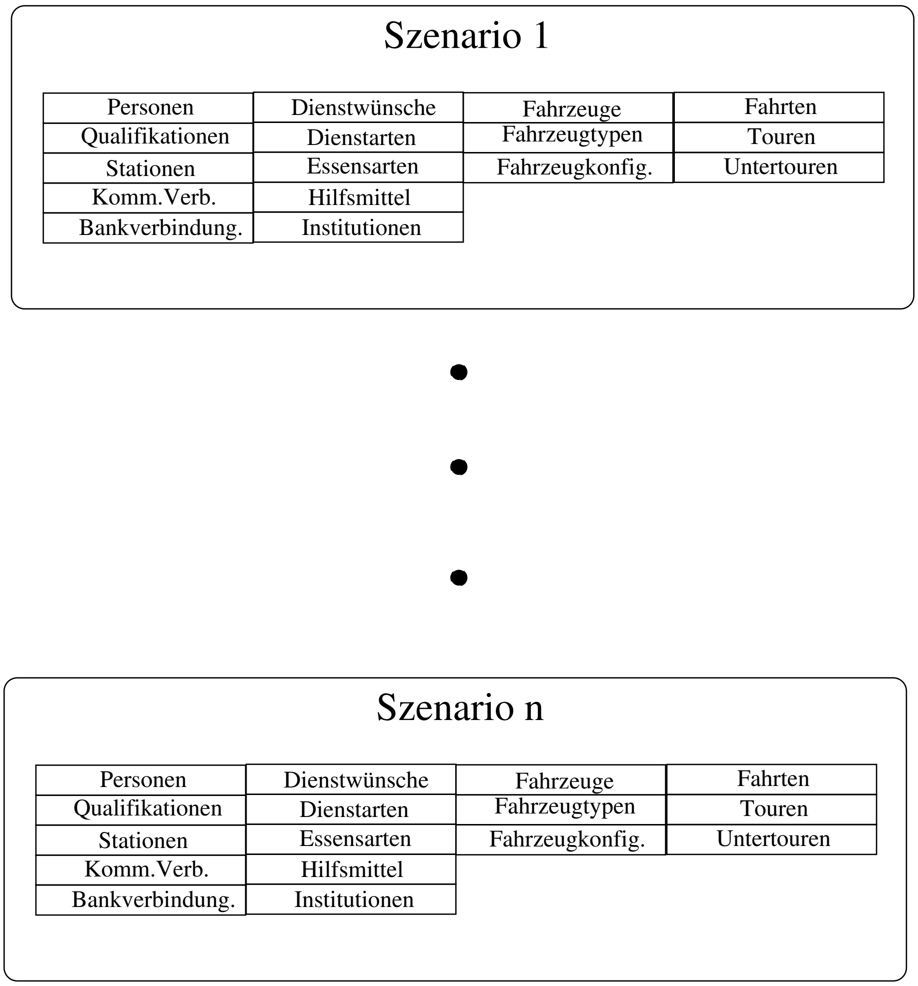


Abbildung 5.2: Aufbau aus Einzelszenarien

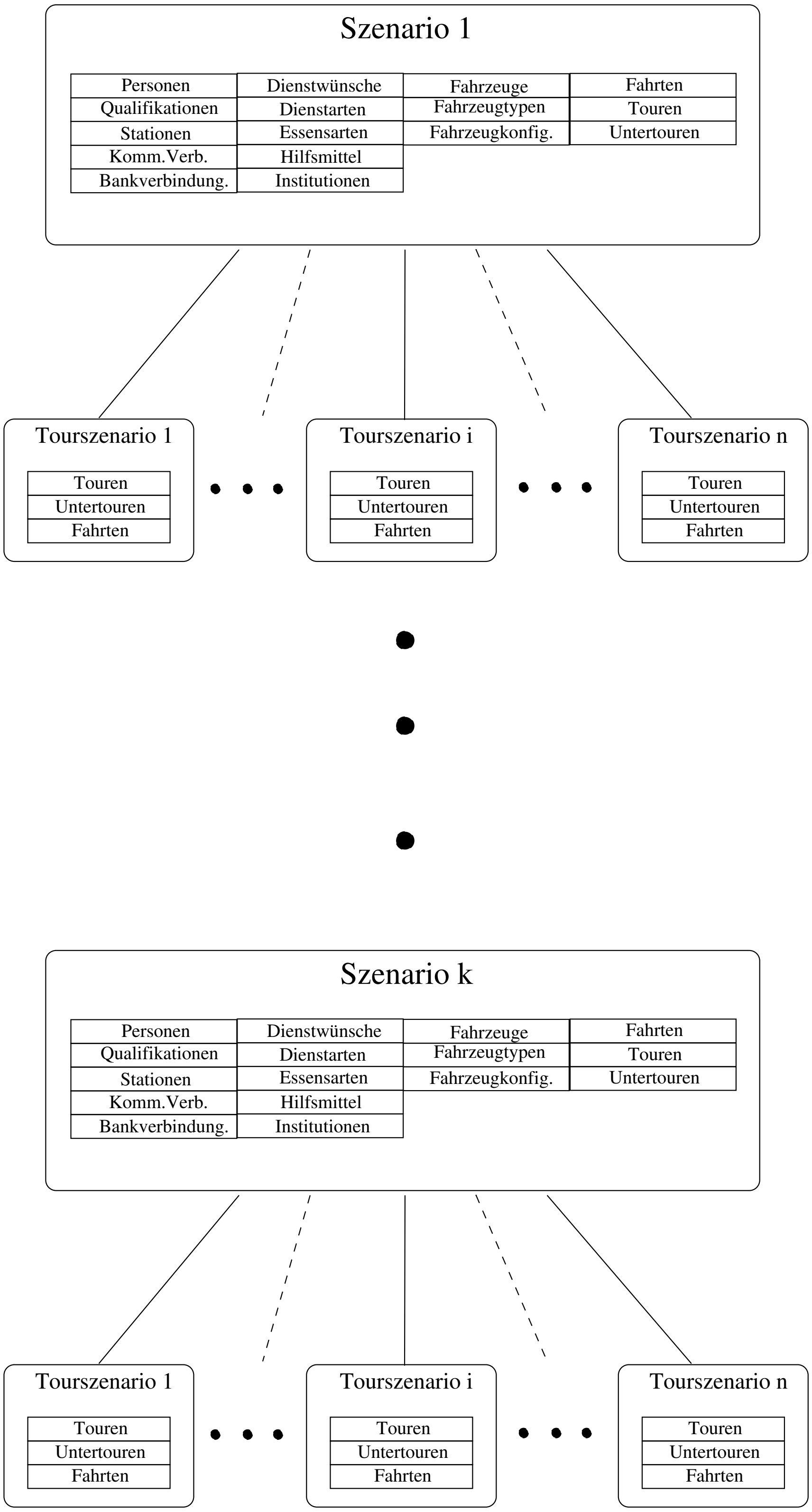


Abbildung 5.3: Aufbau mit Tourszenarien

- 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.

5.1.3.1 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.

5.1.4 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.

5.1.4.1 AnbindungsPunkt

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

5.1.4.2 StationsBeschreibung

PLZ : String . Die Postleitzahl des Ortes.

Ortsname : String .

Strasse : String . Straßenname und Hausnummer.

5.1.4.3 Strecke

von : AnbindungsPunkt .

nach : AnbindungsPunkt .

5.1.5 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.

5.1.5.1 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 $\Rightarrow$ 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 (4.2) 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

5.1.5.2 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).

5.1.6 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

5.2 Feinentwurf

5.2.1 Klasse Person und zugehörige Klassen

5.2.1.1 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.

5.2.1.2 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!

5.2.1.3 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.

5.2.1.4 Institution

- Attribute

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

5.2.1.5 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.

5.2.1.6 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.

5.2.1.7 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.

5.2.1.8 Ort

- Attribute

  PLZ : String Fünfstellige Postleitzahl.

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

5.2.1.9 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.

5.2.1.10 Sachbearbeiter

Unterklasse von Person

- Attribute

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

- Methoden

angestelltBei : Institution Gibt den Arbeitgeber des Sachbearbeiters zurück.

5.2.1.11 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).

5.2.1.12 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.

5.2.2 Klasse Dienstwunsch und zugehörige Klassen

5.2.2.1 Datumsspanne

- Attribute

  Beginn: Calendar (Mußwert) Datum

  Ende: Calendar Datum

5.2.2.2 Zeitspanne

- Attribute

  frühestens: Calendar Uhrzeit

  spätestens: Calendar Uhrzeit

Mindestens eines der Attribute muß einen Wert haben.

5.2.2.3 Zeitraum

- Attribute

  Zeitrahmen: Datumsspanne (Mußwert)

  Ausnahmezeiten: Vector of Datumsspanne

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

5.2.2.4 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?

5.2.2.5 Rhythmus

- Attribute

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

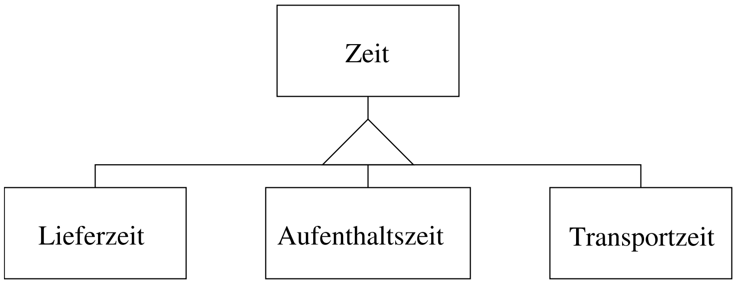
  NurWerktags: boolean

  MeldungAnFeiertagen: boolean

  Wochenrhythmus: int ≥ 1

  Uhrzeit: Zeit

5.2.2.6 Zeit: abstrakt



5.2.2.7 Lieferzeit: Zeit

- Attribute

  Lieferzeit: Zeitspanne (Mußwert)

5.2.2.8 Aufenthaltszeit: Zeit

- Attribute

  Beginn: Zeitspanne (Mußwert)

  Dauer: int Angabe in Minuten

5.2.2.9 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.

5.2.2.10 Essensart

- Attribute

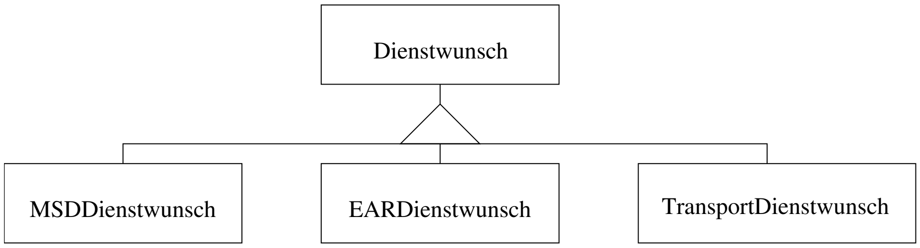
  Bezeichnung: String

  Beschreibung: String

  warm?: boolean

  Preis: int Angabe in Pfennig

5.2.2.11 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?

5.2.2.12 MSDDienstwunsch: Dienstwunsch

- Attribute

  Dienstort: Station Wo soll der Dienst ausgeführt werden?

  FaehrtKundeMit: boolean

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

  benoetigteSitzhilfe: Sitzhilfe

5.2.2.13 EARDienstwunsch: Dienstwunsch

- Attribute

  Lieferadresse: Station

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

5.2.2.14 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

5.2.3 Klasse Fahrzeug und zugehörige Klassen

5.2.3.1 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.

5.2.3.2 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.

5.2.3.3 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.

5.2.4 Klasse Tour und zugehörige Klassen

5.2.4.1 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 $\times$ 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.

5.2.4.2 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.

5.2.4.3 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.

5.2.5 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 5.1.3.1:

5.2.5.1 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.

5.2.5.2 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.

5.2.6 Vorgehensweise bei der Eingabe

5.2.6.1 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.

5.2.6.2 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.

5.2.6.3 Kunden löschen

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

5.2.6.4 Dienstwunsch hinzufügen/ändern

Siehe 5.2.6.2.

5.2.6.5 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.

5.2.6.6 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.

5.2.6.7 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.

5.2.6.8 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 5.2.6.5 und 5.2.6.6) 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 (5.1.6).

5.2.6.9 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.

5.2.6.10 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.

5.2.6.11 Einstellungen

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

5.2.7 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.

5.2.7.1 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.

5.2.7.2 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).

5.2.7.3 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.

Anhang A

Begriffslexikon

Um Mißverständnissen vorzubeugen, sind die im Begriffslexikon definierten Begriffe innerhalb der Projektgruppe TRO nur in der definierten Bedeutung zu benutzen. Dies gilt insbesondere für Begriffe die sowohl in der realen Welt als auch in unserem Modell vorkommen (z.B. Fahrzeug)! Es ist auch darauf zu achten, daß diese Begriffe nicht umschrieben werden (z.B. Synonyme um Wortwiederholungen zu vermeiden, Wortwiederholungen sind hier Pflicht, Motto: eine Bedeutung - ein Wort/Begriff).

Adresse: Eine Adresse besteht aus: Straße, Hausnummer, PLZ und Ort.

Ambulanter Pflegedienst: Fahrten zu verschiedenen Haushalten, um pflegerische Tätigkeiten zu verrichten. Zum APD wird nur Pflegepersonal eingesetzt.

Da die Fahrtkosten nur pauschal in Rechnung gestellt werden, soll die Anzahl der gefahrenen Kilometer optimiert werden.

Versorgte Personen: 60

APD: Ambulanter Pflegedienst

Behindertenfahrdienst: umfaßt regelmäßige Fahrten (zur Tagespflege, Dialysefahrten) und Individualfahrten (Taxi für Rollstuhlfahrer).

Bei Dialyse- und Individualfahrten muß es möglich sein, einen zweiten MA mitzunehmen. Bei Dialysefahrten ist außerdem zu beachten, daß diese auch an Feiertagen stattfinden und daß eine Aufenthaltsdauer von vier Stunden einkalkuliert werden muß.

Die Finanzierung erfolgt bei Fahrten zur Tagespflege durch Pauschalpreis ($\rightarrow$ KM sollen optimiert werden), ansonsten nach der Zahl der gefahrenen Kilometer und der Fahrzeugart.

Dialysefahrten: 10 Personen in 4 Touren
Tagespflegen: 20 Personen in 3 Touren
Individualfahrten: 3.000/Jahr, je Fahrt 1-2 Personen

Bezugsperson: Eine Person die in irgendeinem Bezug zu einem Kunden steht und für das DRK als Kontaktperson wichtig ist. Beispiele: Arzt, Kinder, Eltern, Sozialamt, Krankenkasse

BFD: Behindertenfahrdienst

Dienst: beschreibt die einer Person zugeordneten Tätigkeiten, beispielsweise ein Tour fahren, Tätigkeiten in der Zentrale verrichten, ...

Dienstart: gibt an um welchen Dienst es sich handelt. Vorgesehen sind die Diensttypen: APD, Dialysefahrt, EAR, MSD, Schultour, Taxifahrt

Dienstplan: Wochenplan mit Zuordnung Zivi/Pfleger → Dienst

Dienstwunsch: Repräsentiert einen Kundenauftrag. Er enthält die Dienstart, die gewünschten Stationen mit Abfahrts- und Ankunftzeiträumen, sowie erforderliche Randbedingungen, z.B. ob ein Rollstuhl mitgenommen werden muß.

EAR: Essen auf Rädern

Essen auf Rädern: beinhaltet wöchentliche Kaltlieferungen und regelmäßige Warmlieferungen. Für die Kaltlieferung ist in Zukunft ein Bus mit Wechselkühlvorrichtung vorgesehen. Bei der Warmlieferung ist zu beachten, daß das erste Essen nicht vor 10:45 Uhr und das letzte Essen nicht nach 12:30 Uhr ausgeliefert werden darf.

Die Finanzierung erfolgt über den Essenspreis (→ Optimierung nach KM und Zeit).

Anzahl Kaltessen: 5 Touren mit je 35 Essen
Anzahl Warmessen: 1 Tour mit 30 Essen

Fahrt: Beim Erstellen des Dienstplanes werden Untertouren um ein konkretes Datum ergänzt. Zudem muss einer Fahrt ein Fahrzeug und ein oder zwei Mitarbeiter zugeordnet werden, die nicht mit den (Soll-)Mitarbeitern der Obertour übereinstimmen müssen.

Fahrtwunsch: ist ein Dienstwunsch.

Fahrzeug (Fzg): Ist ein konkret vorhandenes DRK-Fahrzeug mit Angaben zu: DRK-interner Fzg-Nr, Kfz-Nr, Modellbezeichnung des Herstellers, Fzg-Typ, Fzg-Konfiguration, Sperrzeit (Inspektionen ...), HU, ASU, Kundendienst.

Fahrzeug-Konfiguration: Aktuelle Konfiguration eines Fahrzeugs, beispielsweise: #Sitzplätze, #Rollstuhlplätze, #vorhandene Sitzhilfen je Sitzhilfentyp, Kühlmöglichkeit

Fahrzeug-Typ: besteht aus allen möglichen Konfigurationen und den Kosten für einen gefahrenen Kilometer.

Hausschlüssel: ist der Schlüssel eines Kunden, mit dem Mitarbeiter des DRK in die Wohnung des Kunden gelangen können. Hausschlüssel stehen dem DRK (von einigen Kunden) zur Verfügung und kann dort anhand einer eindeutigen Nummer identifziert werden.

Klapp-Rollstuhl: Rollstuhl, der zusammengeklappt und damit in einem PKW untergebracht werden kann.

Kunde: Ein Kunde des DRK, welcher mit folgenden Angaben gespeichert wird: Person, benötigte Anzahl Mitarbeiter mit welchen Qualifikationen, Zuordnung gewünschter Mitarbeiter, Kontaktperson(en), Hausschlüssel

Mitarbeiter (MA): Person, Urlaub, Ist-Arbeitszeit, Einstellungsdatum, Entlassungsdatum, Verfügbarkeitszeit (Krankheit, Fortbildung, etc.), Qualifikation

Mobilie Soziale Dienste: Fahrten zu verschiedenen Haushalten zur Nachbarschaftspflege.

Da die Fahrtkosten (Anfahrt, Einkaufsfahrt, ...) nur pauschal in Rechnung gestellt werden, soll der MSD nach KM optimiert werden.

Versorgte Personen: 180

MSD: Mobiler sozialer Dienst (Kundenbetreuung vor Ort).

Person: Oberklasse aller für unser System relevanten Personen (Kunde, Zivi, ...), die folgende Daten beinhaltet: Adressen, Telefon/Fax-Nummern, Bankverbindung

Qualifikationen: Ein Liste mit Qualifikationen an bzw. des Mitarbeiters, wie beispielsweise: BFD, MSD (sollte vom Kunden erweiterbar sein)

Rollstuhl: Rollstuhl, der nicht zusammengeklappt werden kann.

Rollstuhlplätze: Anzahl der Plätze für Rollstühle, d.h. Plätze, an denen kein eingebauter Sitz ist.

Rückfahrt: entweder eine eigene Tour (Schultour) oder als Rundtour aufgefaßt (EAR, Taxifahrten)

Schulfahrdienst: regelmäßige, schultäglich stattfindende Fahrten zu verschiedenen Schulen, zu denen alle Fahrzeuge eingesetzt werden können. Die Fahrtdauer ist gering zu halten und darf zwei Stunden nicht überschreiten. Fahren anfallgefährdete Behinderte mit, muß maximal ein weiterer MA mitgenommen werden.

Die Kosten des Schulfahrdienstes werden nach den gefahrenen Kilometern und der Fahrzeugart ermittelt.

Beförderte Personen: 140 Anzahl der Touren: 25

Sitzplätze: Anzahl der vorhandenen eingebauten Sitzplätze ohne den Fahrersitz und ohne Rollstuhlplätze (Sitzbänke werden entsprechend ihrer Größe als mehrere Einzelsitze gewertet).

Startzeit: durch unseren Datumstyp repräsentiert (Datum, Rhythmus und Uhrzeit)

Station: Adresse eines Kunden bzw. eines beliebigen Ortes (Schule, Mülldeponie, ...)

Tour: Eine Tour wird durch ihre numerische Tournummer eindeutig identifiziert. Sie besteht aus maximal zwei Soll-Mitarbeitern (Fahrer und Begleiter), max. einem Soll-Fahrzeug und einer Menge von Dienstwünschen einer Dienstart, die bezüglich ihres Datums zusammenpassen.

Untertour: Untertouren werden aus Touren erzeugt, indem eine Stationenfolge mit zugehörigen Startzeiten festgelegt wird, d.h. Untertouren finden immer zur gleichen Uhrzeit statt und haben eine feste Anfangsstation, die nicht Teil eines Dienstwunsches sein muß (Verbindung zur vorigen Untertour). Untertouren werden anhand einer Untertournummer identifiziert, welche sich aus der Tournummer und einem alphanumerischen Teil zusammensetzt.

Verkehrs-Modul: Anbindung des Verkehrstools an TRO, sowie Datenverwaltung für die Verkehrsdaten.

Verkehrs-Tool: Ein verwendetes (externes) Verkehrstool, das austauschbar ist. Nach aktuellem Stand wird „Map And Guide“ eingesetzt.

Zielzeit: s. Startzeit

Zivi: MA mit abgeleisteter Dienstzeit.

Literaturverzeichnis

[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.