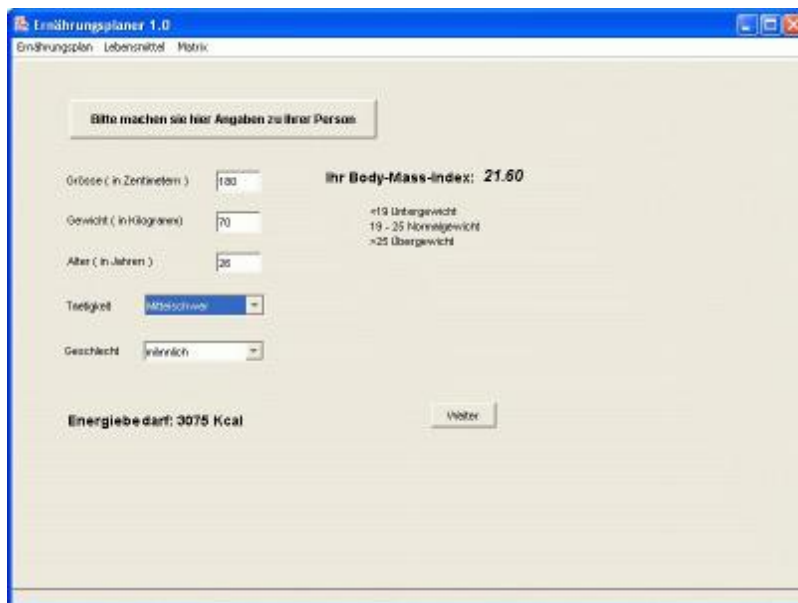


Ernährungsplaner



Ausarbeitung zum Programm Ernährungsplaner im Fach Anwendungen der betrieblichen Systemforschung
8. Semester Wirtschaftsinformatik an der FH-Konstanz

Steffen Kreidler
data@fh-konstanz.de

Matthias Siegert
mat@fh-konstanz.de

Armin Kugler
akugler@fh-konstanz.de

Wintersemester 2003/2004

Zusammenfassung

Thema: Ernährungsplaner

Autoren (Matrikelnr): Steffen Kreidler (272727), Matthias Siegert (271474) und Armin Kugler (271582)

Lehrveranstaltung: Anwendungen betrieblicher Systemforschung im 8. Semester Wirtschaftsinformatik

Termin: Referat gehalten am 26.01.2004

Kapitel 1 enthält den Zweck des Dokuments und eine kurze Einführung in das Projekt. Kapitel 2 enthält den Projekthintergrund und einige Informationen über das bisherige Programm, Kapitel 3 beschäftigt sich mit den Anforderungen und den Zielen für das neue Programm. Kapitel 4 enthält den theoretischen Hintergrund für die Berechnung des BMI, des Tagesbedarfs und das verwendete LP-Modell. In Kapitel 5 ist die Implementierung der Anwendung beschrieben, dabei zuerst die Gesamtarchitektur und anschließend die drei Komponenten GUI, Anwendung und Solverschnittstelle. Kapitel 6 enthält einige Erläuterungen zu den verwendeten Technologien, im Kapitel 7 befindet sich die Anleitung für die Installation des Programms und die Zusammensetzung der Applikation. Das Kapitel 8 beschreibt kurz unsere Projekterfahrungen und den Aufwand der einzelnen Projektmitglieder. Das letzte Kapitel Ausblick beschreibt mögliche Erweiterungen und Verbesserungen für künftige Projektgruppen

.

Inhalt

| | | |
|----------|--|-----------|
| 1 | EINLEITUNG | 4 |
| 2 | PROJEKTHINTERGRUND | 4 |
| 3 | PROJEKTAUFTRAG | 5 |
| 4 | BERECHNUNGSGRUNDLAGEN | 5 |
| 4.1 | BMI..... | 6 |
| 4.2 | Tagesbedarf | 7 |
| 4.3 | LP-Modell | 8 |
| 4.3.1 | Lebensmittelgruppen: | 8 |
| 4.3.2 | Lebensmittel | 9 |
| 4.3.3 | Zielfunktion: | 10 |
| 4.3.4 | Restriktion Tagesbedarf:..... | 10 |
| 4.3.5 | Restriktion Lebensmittelgruppe: | 10 |
| 4.3.6 | Restriktion Benutzerauswahl: | 11 |
| 4.3.7 | Erweiterbarkeit: | 11 |
| 5 | IMPLEMENTIERUNG | 11 |
| 5.1 | Architektur | 11 |
| 5.2 | GUI..... | 13 |
| 5.2.1 | Startbildschirm – Klasse mainFrame | 13 |
| 5.2.2 | Erährungsplan erstellen | 16 |
| 5.2.3 | Lebensmittelliste bearbeiten | 22 |
| 5.3 | Solver Schnittstelle | 26 |
| 5.4 | Anwendung und Datenbankbindung | 28 |
| 5.4.1 | Die Klasse DiätplanerApplication | 28 |
| 5.4.2 | Die Klasse XMLListHandler | 29 |
| 6 | VERWENDETE TECHNOLOGIEN | 29 |
| 7 | ADMINISTRATION | 30 |
| 7.1 | Installation | 30 |
| 7.2 | Zusammensetzung der Applikation..... | 31 |
| 8 | PROJEKTERFAHRUNGEN UND AUFWAND..... | 31 |
| 9 | AUSBLICK | 32 |

1 Einleitung

Im Rahmen der Vorlesung "Anwendungen der betrieblichen Systemforschung" gibt es zwei Möglichkeiten, den Leistungsnachweis zu erbringen:

- Erstellung eines Softwareprototyps, welcher sich in die Methodendatenbank integrieren lässt und eine Methode des Operation Research implementiert.
- Erstellung eines Planspiels, welches in nachfolgenden Semestern innerhalb der Vorlesung angewandt werden kann, um Methoden des Operation Research zu vertiefen.

Unsere Entscheidung fiel auf die Implementierung eines Softwareprototyps. In dieser Hausarbeit wird das Programm beschrieben, mit dem Blickpunkt auf seine Funktionalität, seine Anbindung an einen Solver, die Bedienung und die technische Realisierung.

Unser Auftrag war das „Diätplaner“ Programm weiterzuentwickeln, da es viele Funktionen vorsieht, die noch nicht implementiert sind. Bei genauerer Durchsicht des Programms stellten wir jedoch fest, dass an eine vernünftige Weiterentwicklung dieses Programms nicht zu denken war (siehe Projekthintergrund). Nach erneuter Rücksprache mit Professor Grütz haben wir uns dann entschlossen, das Projekt von Grund auf neu zu implementieren. Zur besseren Entscheidung bekam es einen anderen Namen und eine andere Zielsetzung, der Ernährungsplaner war geboren.

2 Projekthintergrund

Es existierte bereits eine Anwendung Diätplaner, die allerdings in der bisherigen Form nicht funktionierte und auch in der Architektur Nachteile besaß. In der bisherigen Anwendung wurde eine mit JDBC-Treibern angesprochene Access-Datenbank als Datenbasis verwendet, dies hatte zur Folge, dass bei jedem Programmstart auf einem Rechner zuerst der ODBC-Treiber in der Windows-Systemsteuerung installiert werden musste. Viele Programmteile waren nicht implementiert (z.B. Lebensmittel hinzufügen / Lebensmittel löschen), außerdem lieferte das Programm fast immer das gleiche Tagesmenü, egal welche Lebensmittel vom Benutzer ausgewählt wurden. Die grafische Oberfläche war sehr instabil und auch vom Layout nicht zufrieden stellend. Darüber hinaus fehlte die Ausgabe des LP-Modells und der Ergebnisse des Solvers, dies erschwerte die Fehlersuche zusätzlich. Der theoretische LP-Ansatz des Programms war sehr gut und nachvollziehbar, lediglich die Berechnung des Tagesumsatzes wurde von uns überarbeitet. Nach dieser Analyse des Programms entschlossen wir uns zu einer kompletten Neuentwicklung, lediglich einige Teile der theoretischen Grundlagen aus der Ausarbeitung des Programms wurden von uns übernommen.

3 Projektauftrag

Nach einer gründlichen Analyse der bisherigen Anwendung definierten wir die Anforderungen an die von uns zu entwickelnde Applikation. Diese Anforderungen waren:

- Berechnung des Tagesbedarfs in Abhängigkeit von Größe, Alter, Geschlecht, Tätigkeit und Gewicht,
- Errechnung des BMI (Body-Mass-Index), um festzustellen, ob die Person Übergewicht, Untergewicht oder Normalgewicht hat und Berücksichtigung des BMI bei der Berechnung des Tagesbedarfs,
- Aufteilung des Tagesmenüs in Frühstück, Mittagessen und Abendessen,
- Berücksichtigung aller ausgewählten Lebensmittel für das Tagesmenü,
- Benutzung eines XML-Files als Datenbasis,
- Möglichkeit des Hinzufügens von Lebensmitteln,
- Möglichkeit des Löschens von Lebensmitteln,
- Erstellen einer benutzerfreundlichen Oberfläche mit ansprechendem Layout,
- Programmieren einer einfachen Installationsroutine,
- Bessere Einbindung des Solvers,
- Klare Komponententrennung des Programms bei der Entwicklung in View-Application-DatabaseInterface-SolverInterface,
- Feste Definition der Schnittstellen zwischen den Komponenten, um getrennte Programmierung zu ermöglichen.

Das Ziel war eine benutzerfreundliche, möglichst fehlerfreie und nützliche Anwendung, die auch für folgende Projektgruppen möglichst leicht erweiterbar sein sollte.

4 Berechnungsgrundlagen

Nach tagelangen Recherchen im Internet und in der Universitätsbibliothek stellten wir fest, dass Ernährung ein sehr komplexes Thema ist. Es gab leider nicht „Den Ansatz“ sondern viele verschiedenen Ansätze und Meinungen zur richtigen Ernährung. Gerade bei der Recherche in der Uni-Bibliothek stellten wir fest, dass bei einer perfekten Ernährung sehr viele Aspekte zu berücksichtigen sind, darunter die Herkunft (Asiaten haben einen völlig anderen Bedarf als Europäer), das Klima, Krankheiten, genetische Besonderheiten, Blutgruppe, Art der Arbeit usw. Beispielweise bestand das „*Handbuch der menschlichen Ernährung*“ an der Universität aus drei Bänden mit jeweils 500 (!) Seiten und unzähligen Tabellen. Aus Zeitgründen suchten wir deshalb im Internet nach einfacheren Lösungen, die zwar nicht so genau sind, aber für unsere Zwecke ausreichend und in absehbarer Zeit realisierbar sind. Für unsere Anwendung beschlossen wir daher, vorerst nur Kalorien zu berücksichtigen, aber eine einfache Erweiterung des Programms möglich zu machen um weiteren Gruppen die Implementierung weiterer Faktoren zu ermöglichen.

Unser Programm berechnet aus Größe, Alter, Geschlecht, Gewicht und Tätigkeit der Person zuerst den Body-mass-index (BMI). Aus dem BMI kann abgelesen werden, ob die Person Normalgewicht, Untergewicht oder Übergewicht hat. Je nach BMI wird anschließend der Tagesbedarf an Kalorien berechnet, der als Input für das LP-Modell dient. Das erzeugte LP-Modell berechnet schließlich aus dem Tagesbedarf

und den gewählten Gerichten die optimale Menüzusammenstellung für einen Tag. Als Referenz für die einzelnen zu berechnenden Größen dienen die folgenden Unterkapitel.

4.1 BMI

Der Body-Mass-Index (BMI) wird im Ernährungsplaner berechnet um festzustellen, ob eine Person über- bzw untergewichtig ist. Der BMI Wert wird durch folgende Formel ermittelt:

$$\text{BMI} = \text{Gewicht} / (\text{Größe in cm}/100)^2$$

Ein Beispiel:

Gewicht 70 kg
Größe: 180 cm

$$\Rightarrow \text{BMI} = 70 / 1,8^2$$

$$\Rightarrow \text{BMI} = 21,6$$

Ein BMI Wert unter 19 bedeutet Untergewicht, ein BMI Wert über 25 bedeutet Übergewicht. Diese Zahlen sind aber von Quelle zu Quelle leicht unterschiedlich, am häufigsten wird jedoch 19 und 25 für Mann und Frau genannt. Der BMI gibt auch Auskunft über das Risiko von Herz-Kreislauf-Erkrankungen:

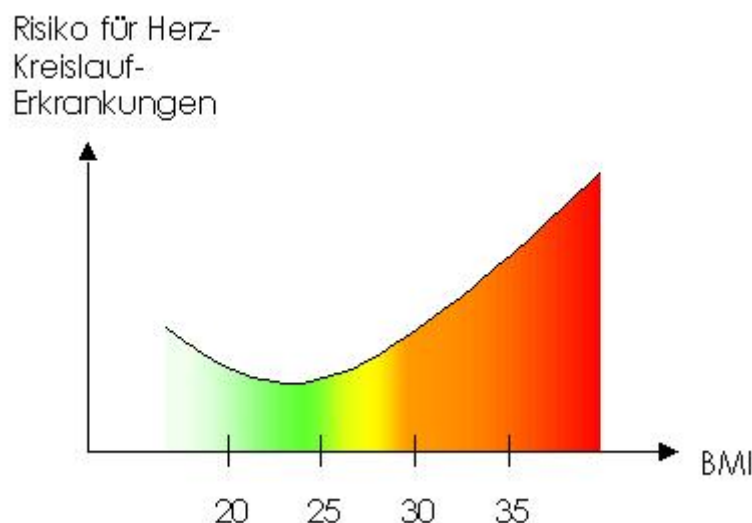


Abb.1: Risiko von Herz-Kreislauf-Erkrankungen

Der Ernährungsplaner berechnet den täglichen Kalorienbedarf für eine Person. Je höher das Gewicht einer Person, desto mehr Kalorien für den Tagesbedarf werden benötigt. Dies soll aber nicht für über- bzw. untergewichtige Personen gelten, da ja sonst ein stark Übergewichtiger auch noch extra viele Kalorien zu sich nehmen dürfte.

Deshalb wird bei BMI Werten über 25 bzw. unter 19 das Normalgewicht als Berechnungsgrundlage verwendet:

$$\text{Normalgewicht} = \text{Größe in cm} - 100$$

Ein Beispiel:

Gewicht: 85 kg
Größe: 175 cm
BMI = 27,75

Da der BMI größer als 25 ist, wird als Berechnungsgrundlage für das Normalgewicht

$$175 - 100 = \mathbf{75\ kg}$$

verwendet. Als Eingangsinput für den Tagesbedarf der Person werden folgende Daten verwendet:

Gewicht: 75 kg
Größe: 175 cm

4.2 Tagesbedarf

Der tägliche Kalorienbedarf (Gesamtumsatz) errechnet sich aus dem Grundumsatz + Leistungsumsatz.

- der Grundumsatz ist die Energiemenge, die ein Mensch bei Ruhe für die Aufrechterhaltung der lebenswichtigen Körperfunktionen (Atmung, Herzschlag, Drüsenfunktion) pro Tag benötigt.
- der Leistungsumsatz ist die Energiemenge, die unser Körper innerhalb eines Tages benötigt, um Arbeit zu verrichten.

Die Literatur ist sich über die Definition des Grund- und Leistungsumsatzes einig, völlige Uneinigkeit herrscht jedoch bei der Berechnung dieser Werte. Egal welche Quelle aufgeschlagen wird, die Formeln sind immer unterschiedlich. Nachdem wir eine längere Literaturrecherche an der Unibibliothek durchgeführt hatten, sind wir zu dem Ergebnis gekommen, dass wohl kein gemeinsamer Nenner in diesem Formelwirrwarr gefunden werden kann. Für den Grundumsatz wird teilweise die Formel „1 kcal pro Stunde für 1 kg Körpergewicht“ verwendet (dies war auch die Formel des alten Diätplaners, der Leistungsumsatz wird dabei unterschlagen). Andere Formeln für den Grundumsatz berücksichtigen sogar den Fett- bzw. Muskelanteil und verschiedene Hormone im Körper.

Der Einfachheit halber suchten wir dann eine renommierte Internetseite (als leicht nachvollziehbare Referenz), die den Energiebedarf berechnet und mindestens folgende Einflussfaktoren berücksichtigt:

- Größe
- Gewicht
- Alter
- Geschlecht
- Art der Tätigkeit

Nachdem wir auf ein paar Seiten die Formeln untersucht haben, entschieden wir uns als Referenz folgende Seite zu benutzen:

<http://www.hochdruck-aktuell.de/public/test/energiebedarf.html>

Zur Berechnung des Grundumsatzes verwendet der Ernährungsplaner (und unsere Referenzseite) folgende Formel:

| | (1. Teil) | (2. Teil) | (3. Teil) |
|------|---------------------------------|---|-------------|
| Mann | 297,9 * Gewicht ^{0,75} | * (1+0,004*(30-Alter) + 0,010 * (Größe/Gewicht ^{0,33} – 43,4)) | |
| Frau | 275,3 * Gewicht ^{0,75} | * (1+0,004*(30-Alter) + 0,018 * (Größe/Gewicht ^{0,33} – 42,1)) | |

Der Grundumsatz wird hauptsächlich durch das Gewicht (1. Teil) beeinflusst, der 1. Teil der Formel wird noch durch den 2. + 3. Teil korrigiert.

Der 2. Teil berücksichtigt das Alter der Person, ist eine Person über 30 Jahre alt werden Kalorien abgezogen, 29 Jahre und jünger bekommen Kalorien addiert. Der 3. Teil der Formel berücksichtigt die Größe der Person im Verhältnis zum Gewicht.

Nachdem der Grundumsatz berechnet ist, muss für den Gesamtumsatz (täglicher Energiebedarf) noch der Leistungsumsatz hinzuaddiert werden.

Der Leistungsumsatz wird durch folgende Formel berechnet:

$$\text{Leistungsumsatz} = \text{Grundumsatz} * \text{Tätigkeitsfaktor}$$

| Tätigkeitsfaktor | |
|------------------|-------------------------|
| 0,5 | leichte Tätigkeit |
| 0,75 | mittelschwere Tätigkeit |
| 1 | schwere Tätigkeit |

Ist der Energiebedarf berechnet (Grundumsatz + Leistungsumsatz), muss der Energiebedarf noch von KJ (Kilojoule) in kcal (Kilokalorien) umgerechnet werden. Dazu wird folgende Formel verwendet:

$$\text{Energiebedarf in kcal} = (\text{Grundumsatz} + \text{Leistungsumsatz}) / 4,18$$

Hinweis für folgende Projektgruppen: Alle Berechnungen sind in der Klasse „Energiebedarf.java“ gekapselt, durch Austausch der Klasse kann sehr einfach eine andere Formel zum berechnen des Energiebedarfs herangezogen werden.

4.3 LP-Modell

Nach Berechnung des BMI und des Tagesbedarfs wird das benötigte LP-Modell erzeugt. Das LP-Modell besteht aus der Zielfunktion und drei verschiedenen Arten von Restriktionen. Es wurde zum größten Teil aus der Vorgängerapplikation übernommen. Jedes mögliche Gericht wird durch eine Variable repräsentiert, die entweder den Wert Null (Lebensmittel wird nicht für das Tagesgericht verwendet) oder den Wert eins (Lebensmittel ist im Tagesgericht enthalten) besitzen kann.

4.3.1 Lebensmittelgruppen:

Jedes Lebensmittel gehört zu genau einer Lebensmittelgruppe. Es gibt 11 verschiedene Arten von Lebensmittelgruppen:

| LebensmittelgruppenID | Lebensmittelgruppenname |
|-----------------------|-------------------------|
| 0 | Milchprodukte |
| 1 | Cerealien |
| 2 | Backware_Frühstück |
| 3 | Getränk_Frühstück |
| 4 | Hauptgericht |
| 5 | Beilage |
| 6 | Dessert |
| 7 | Getränk_Mittagessen |
| 8 | Brotbelag |
| 9 | Backwaren_Abendessen |
| 10 | Getränk_Abendessen |

Abbildung 2 - Verwendete Lebensmittelgruppen

4.3.2 Lebensmittel

Die Datenbasis enthält einige Lebensmittel, über die graphische Oberfläche können allerdings neue Lebensmittel hinzugefügt und alte Lebensmittel gelöscht werden. Die Informationen über Lebensmittel und ihre Kalorienanzahl bekommt man am besten über folgenden Link:

<http://www.kalorien-tabelle.de/tab.html>

Die folgende Tabelle enthält die Lebensmittel der Standarddatenbasis zusammen mit Ihren Kalorien:

| Gruppe | Name | Menge | Kalorien | GruppenID: |
|----------------------|----------------|-------|----------|------------|
| Milchprodukte | Milch (3,5 %) | 200 | 132 | 0 |
| Milchprodukte | Milch (1,5 %) | 200 | 98 | 0 |
| Cerealien | Haferflocken | 60 | 230 | 1 |
| Cerealien | Cornflakes | 80 | 295 | 1 |
| backware_fruehstueck | Butterhörnchen | 45 | 185 | 2 |
| backware_fruehstueck | Butterbrezel | 50 | 180 | 2 |
| getraenk_fruehstueck | Kaffee | 125 | 100 | 3 |
| getraenk_fruehstueck | Kakao | 150 | 140 | 3 |
| hauptgericht | Rumpsteak | 150 | 430 | 4 |
| hauptgericht | Schweinebauch | 125 | 216 | 4 |
| beilage | Maccaroni | 100 | 359 | 5 |
| beilage | Bratkartoffeln | 150 | 294 | 5 |
| dessert | Eiscreme | 35 | 59 | 6 |
| dessert | Apfelkompott | 150 | 100 | 6 |
| getraenk_mittagessen | Weissbier | 500 | 190 | 7 |
| getraenk_mittagessen | Apfelsaft | 200 | 95 | 7 |

| | | | | |
|---------------------------|-----------------|-----|-----|----|
| Brotbelag | Emmentaler | 30 | 90 | 8 |
| Brotbelag | Gänseleberpaste | 25 | 80 | 8 |
| backwa- ren_abendessen | Bauernbrot | 50 | 95 | 9 |
| backwa- ren_abendessen | Roggenbrot | 45 | 100 | 9 |
| getraenk_abendessen | Weinschorle | 125 | 45 | 10 |
| getraenk_abendessen | Cola | 330 | 185 | 10 |
| getraenk_abendessen | Cola Light | 330 | 1 | 10 |

Abbildung 3 - Verwendete Lebensmittel

Von jeder Lebensmittelgruppe muss mindestens ein Lebensmittel im Tagesmenü enthalten sein, damit eine ausgeglichene Ernährung gewährleistet ist. Darüber hinaus kann der Benutzer Gerichte wählen, die auf jeden Fall im Tagesmenü vorkommen sollen, dies wird ebenfalls im LP-Modell berücksichtigt. Schließlich muss noch der Tagesbedarf der Person für eine Restriktion verwendet werden, damit keine Unterernährung möglich ist.

4.3.3 Zielfunktion:

Zuerst wird die Zielfunktion generiert, dabei ist das Ziel der Anwendung die Minimierung der Gesamtkalorien unter Berücksichtigung der Deckung des Tagesbedarfs. Das bedeutet die Summe der gesamten Kalorien soll minimiert werden. Dies führt zu folgender Zielfunktion:

$$X_1 * \text{Kalorien}(X_1) + X_2 * \text{Kalorien}(X_2) + X_3 * \text{Kalorien}(X_3) + \dots + X_n * \text{Kalorien}(X_n) \rightarrow \text{Min}$$

4.3.4 Restriktion Tagesbedarf:

Die Restriktion Tagesbedarf sorgt dafür, dass der Tagesbedarf an Kalorien gedeckt wird, das bedeutet, die Summe der Gesamtkalorien muss einen bestimmten Wert überschreiten. Daraus ergibt sich folgende Restriktion:

$$X_1 * \text{Kalorien}(X_1) + X_2 * \text{Kalorien}(X_2) + X_3 * \text{Kalorien}(X_3) + \dots + X_n * \text{Kalorien}(X_n) \geq \text{Tagesbedarf}$$

4.3.5 Restriktion Lebensmittelgruppe:

Die zweite Sorte von Restriktionen stellt sicher, dass von jeder Lebensmittelgruppe mindestens ein Lebensmittel vorhanden ist. Dafür existiert im LP-Modell für jede Lebensmittelgruppe genau eine Restriktion von folgender Form:

$$\text{Lebensmittelgruppe 1: } X_1 + X_2 + X_3 + X_4 \geq 1$$

$$\text{Lebensmittelgruppe 2: } X_5 + X_6 + X_7 + X_8 \geq 1$$

Usw.

Jede Zeile enthält dabei nur Variablen, die Lebensmittel repräsentieren, die zu dieser Lebensmittelgruppe gehören.

4.3.6 Restriktion Benutzerauswahl:

Die dritte Sorte von Restriktionen bildet die Lebensmittelauswahl des Benutzers auf das LP-Modell ab. Jedes Lebensmittel, welches der Benutzer ausgewählt hat, soll auch im Menü vorkommen. Je mehr Lebensmittel der Benutzer ausgewählt hat, desto eingeschränkter werden logischerweise die Möglichkeiten des Solvers für die Menüzusammenstellung. Hat der Benutzer aus jeder Lebensmittelgruppe ein Lebensmittel ausgewählt und ist mit den Kalorien der Tagesbedarf gedeckt, so bildet die eigene Zusammenstellung auch das Tagesmenü. Hat der Benutzer gar nichts oder nur wenige Lebensmittel ausgewählt, so hat der Solver die größten Variationsmöglichkeiten, um eine optimale Lösung zu finden. Für jedes ausgewählte Lebensmittel entsteht so eine neue Restriktion, die sicherstellt, dass dieses Lebensmittel auch in der Lösung vorkommt:

| | | |
|-----------|-----------|---|
| X1 | | = 1 (Lebensmittel X1 wurde ausgewählt) |
| | X5 | = 1 (Lebensmittel X5 wurde ausgewählt) |
| | X6 | = 1 (Lebensmittel X6 wurde ausgewählt) |

Usw.

4.3.7 Erweiterbarkeit:

Das LP-Modell ist sehr gut erweiterbar. Erweiterungen werden dabei durch zusätzliche Restriktionen implementiert. Mögliche Erweiterungen wären z.B. Restriktionen, die die Versorgung mit Spurenelementen und Vitaminen sicherstellen oder den Fettgehalt reduzieren. Auch Restriktionen, die Lebensmittel ausschließen, die vom Benutzer nicht gewünscht werden sind vorstellbar, dafür wäre allerdings eine extra Auswahl in der Benutzeroberfläche nötig. Bei der Datenbasis, die als XML-File realisiert ist, können neue Daten über Lebensmittel (z.B. Vitamine, Fettgehalt, Kohlenhydrate usw.) einfach als neue Tags eingefügt werden, ohne das bestehende Klassen geändert werden müssen. Lediglich die Methode `executeBerechnung()` der Klasse `DiätplanerApplication` muss für das neue LP-Modell angepasst werden, da sie die Matrix für den Solver generiert. Zusätzlich muss natürlich noch die grafische Oberfläche für die Eingabe neuer Lebensmittel um die neuen Informationen erweitert werden.

5 Implementierung

5.1 Architektur

Bei der Entwicklung des Programms wurde von Anfang an Wert auf eine strikte Trennung des Programms in mehrere Komponenten gelegt. Damit war auch die parallele Entwicklung der Applikation von allen drei Teammitgliedern gewährleistet. Das Programm besteht aus vier Komponenten. Diese sind:

- View,
- Application,
- Database,
- Solver.

Die View-Komponente, die für die gesamte graphische Oberfläche zuständig ist, wurde von Armin programmiert und besteht aus der Klasse MainFrame und den Hilfsklassen JPanelWerte, JPanelAuswahl, JPanelAusgabe, JPanelLebensmittelHinzufügen und JPanelLebensmittelLöschen.

Die Application-Komponente steuert den Programmablauf und enthält die Programmlogik. Sie ist die zentrale Komponente der Anwendung, die Schnittstellen zur View-Komponente, Database-Komponente und Solver-Komponente besitzt.

Die Database-Komponente besteht aus dem XML-Listhandler, der ein XML-File verwaltet, das alle relevanten Daten über die Lebensmittel enthält.

Die Solver-Komponente bildet die Anbindung zum verwendeten Solver LP-Solve. Sie besteht aus einem Interface und einer Solverklasse, sowie einer Klasse Matrix, die, das LP-Modell enthält.

Zusätzlich implementierten wir eine Klasse DiätplanerException, die für das Fehlerhandling zuständig ist.

Auf dem folgenden UML-Klassendiagramm sind die wichtigsten Klassen mit Ihren Assoziationen zur Veranschaulichung abgebildet.

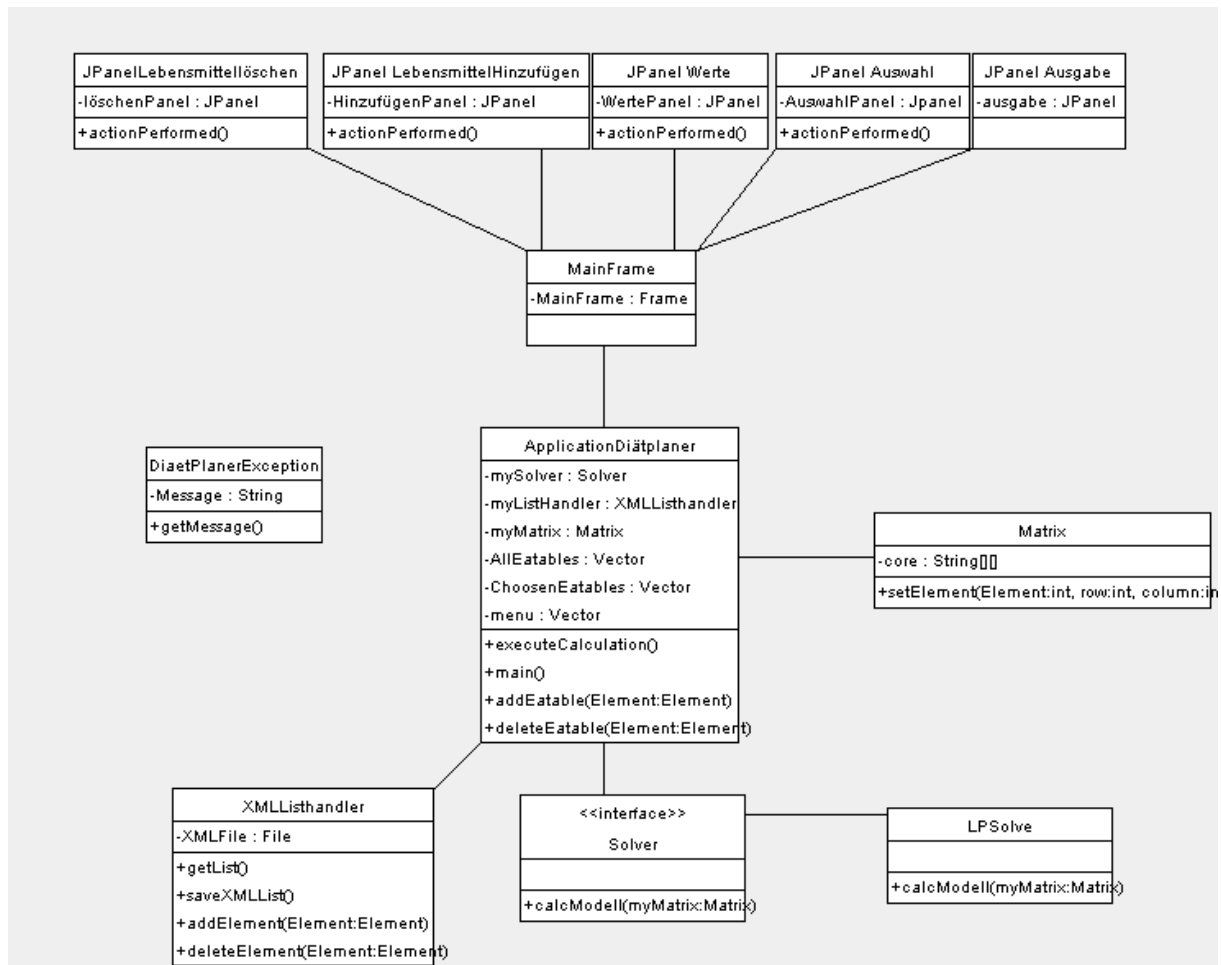


Abbildung 4 - UML-Klassendiagramm

In den folgenden Kapiteln sind die einzelnen Komponenten genauer beschrieben:

5.2 GUI

Die Oberfläche wurde mit swing Elementen realisiert. Sie soll leicht verständlich sein und für Benutzer intuitiv durch die Menüs führen.

5.2.1 Startbildschirm – Klasse mainFrame

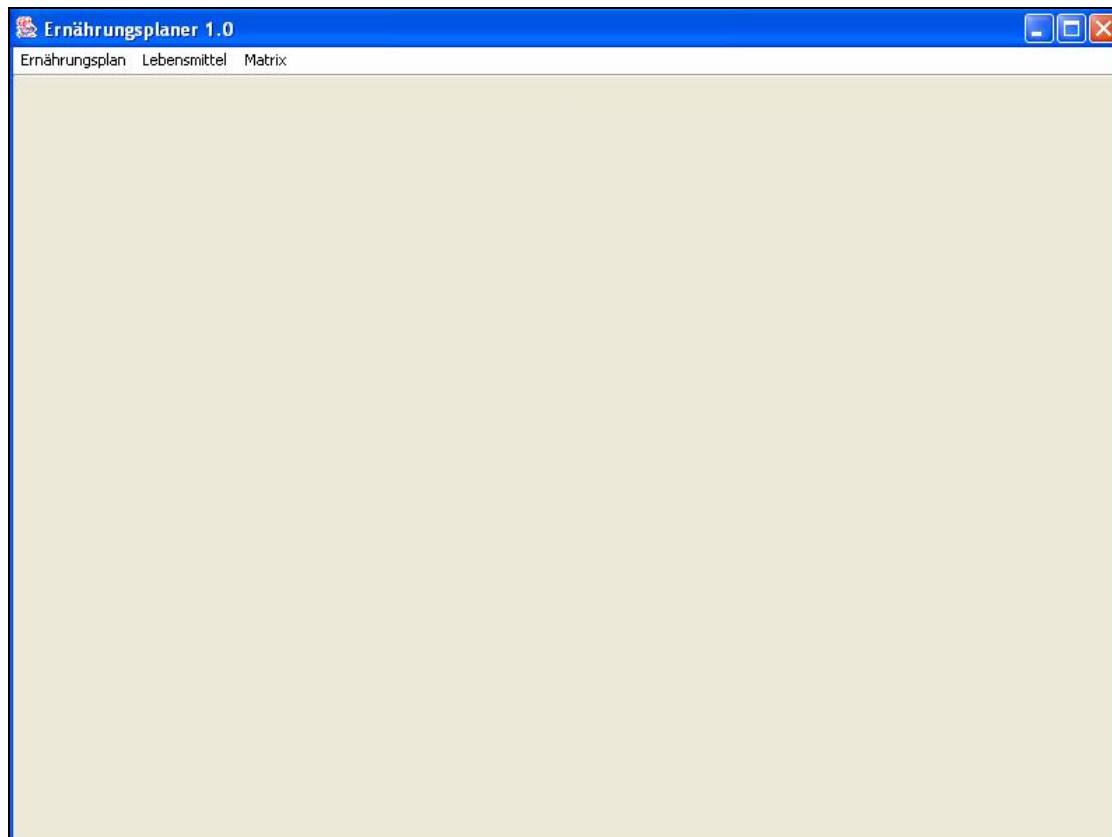


Abbildung 5 - Startbildschirm

Im Startbildschirm kann man in der Menüleiste zwischen den einzelnen Funktionen des Programms auswählen. Diese Stellen einzelne Arbeitsschritte dar, die nur einzeln aufgerufen werden können. Hat man sich einmal für einen Arbeitsschritt entschieden, sind die anderen Funktionen für die Dauer der Bearbeitung deaktiviert. Dieses Fenster stellt die Zentrale des Programms dar. Nach jedem Arbeitsschritt kehrt der Benutzer automatisch wieder hierher zurück.



Abbildung 6 – Menü Ernährungsplan

Unter dem Menüpunkt „Ernährungsplan“ können Sie entweder mit dem Menüpunkt „Plan erstellen“ einen neuen individuellen Ernährungsplan generieren oder mit „Exit“ das Programm beenden.



Abbildung 7 – Menü Lebensmittel

Der Menüpunkt Lebensmittel beinhaltet die Bearbeitung der im Programm zur Verfügung stehenden Lebensmittel. Mit dem Menüpunkt „hinzufügen“ können sie die Lebensmittelliste um ein neues Element erweitern, mit dem Menüpunkt „löschen“ ein Element aus der Liste entfernen.



Abbildung 8 – Menü Matrix

Hinter dem Menüpunkt „Matrix“ verbirgt sich die Möglichkeit, sich die zur Berechnung des Tagesmenüs kreierte Matrix anzeigen zu lassen oder auf die gesonderte Ausgabe zu verzichten. Dieser Menüpunkte wurde auf Wunsch von Professor Grütz hinzugefügt.

Verantwortlich hierfür ist die Klasse „mainFrame“. Sie ist eine reine Darstellungsklasse.

```
public mainFrame(Vector allEatables, ApplicationDiaetplaner ap) {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try {

        allEatables_=allEatables;
        ap_=ap;
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

Listing 1 – Klasse mainFrame

Die Klasse bekommt eine Referenz auf die Klasse „ApplicationDiaetplaner“ übergeben. Diese liefert alle nötigen Methoden, die zur Bearbeitung notwendig sind. Dazu bekommt sie mit dem Vector „allEatables“ die Liste aller Lebensmittel, die sie an die entsprechenden Klassen der aufgerufenen Folgefenster weiterreicht.

Die wichtigste Methode in dieser Klasse ist die Methode „closechild“.

```
public void closechild(int jPanel,JPanel target, int kalorien, Vector
resultVector)
{
    contentPane.remove(target);
    if (jPanel==1) //Weiter zum 2.Schritt
    {
        kalorien_=kalorien;
        myJPanelEingabeWerte = new
        JPanelAuswahl(allEatables_, ap_,kalorien_,this);
        this.repaint();
        contentPane.add(myJPanelEingabeWerte, BorderLayout.CENTER,1);
        this.validate();
    }
    if (jPanel==2) //Weiter zu 3. Schritt
```

```

{
    JPanelAusgabe myJPanelAusgabe= new
    JPanelAusgabe(resultVector,ap_.getMenuCalories(),kalorien_,this);
    contentPane.add(myJPanelAusgabe, BorderLayout.CENTER,1);
    this.validate();
}
if ( (jPanel==3)|| (jPanel==4)|| (jPanel==5)) //Zurück zum mainFrame
{
    jMenuPlanErstellen.setEnabled(true);
    jMenuItemLebensmittelhinzufuegen.setEnabled(true);
    jMenuItemLebensmittelloeschen.setEnabled(true);
    this.validate();
}

this.repaint();
}

```

Listing 2 – Klasse mainFrame.closechild

Da die einzelnen Panels sich nicht selbst schließen können, wird am Ende jedes Panels die Klasse „mainFrame“ beauftragt, das entsprechende Panel (JPanel target) aus der contentPane zu entfernen, und im Falle einer Serie von Panels das nächste anzuzeigen. Damit die Methode auch weiß, welches Panel sie als nächstes anzeigen soll, bekommt sie eine Identifikation mitgesendet (int jPanel), aufgrund derer das nächste Element ausgewählt und angezeigt wird.

5.2.2 Ernährungsplan erstellen

Das Herzstück des Programms ist die Erstellung eines individuellen Menüplans, der den jeweiligen Tagesbedarf deckt. Dieser Tagesbedarf errechnet sich aus den persönlichen Daten des Benutzers.

5.2.2.1 Eingabe der Werte

Im ersten Schritt des Arbeitsganges werden die persönlichen Daten und Tätigkeiten des Benutzers eingelesen.

Abbildung 9 – Ernährungsplan Schritt 1

Der Benutzer gibt hier seine Größe, Gewicht und Alter an. Hieraus errechnet sich der auf der rechten Seite automatisch angezeigte Body-Mass-Index. Im Falle eines Übergewichts oder Untergewichts wird zusätzlich ein Hinweis angezeigt. Danach sollte der Benutzer noch die Schwere seiner Tätigkeit angeben und sein Geschlecht.

Die Angaben werden auf Vollständigkeit und Typsicherheit geprüft, falls dies nicht zutrifft erscheint eine entsprechende Fehlermeldung.

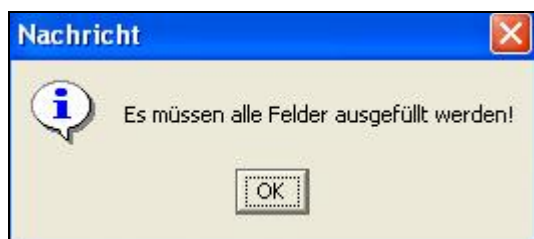


Abbildung 10 – Ernährungsplan Fehler

Dieses Panel wird durch die Klasse „JPanelWerte“ gezeichnet und die Klasse „Energiebedarf“ errechnet den BMI. Interessant in diesem Panel ist die Methode `updateScreen()`. Sie aktualisiert nach jeder Veränderung der Oberfläche die Anzeige auf der rechten Hälfte des Panels.

```
private void updateScreen()
{
    //...//
}
```

```

try // aktualisiere Werte
{
    if (jTextFieldGroesse.getText().length() >0)
        groesse = new Integer(jTextFieldGroesse.getText()).intValue();
    if (jTextFieldGewicht.getText().length() >0)
        gewicht = new Integer(jTextFieldGewicht.getText()).intValue();
    if (jTextFieldAlter.getText().length() >0)
        alter = new Integer(jTextFieldAlter.getText()).intValue();
}
catch(Exception e)
{
    //...//
}

// neuen BMI berechnen

if (groesse > 0 && gewicht >0)
{
    bmi=Energiebedarf.bmi(gewicht,groesse); //Aufruf Energiebedarfsklasse
    //...//

    //Gegebenenfalls die Textausgabe schreiben

    jLabelUnterUeber.setText("<html>Sie haben Untergewicht, für die
    Berechnung des Energiebedarfs wird deshalb das Normalgewicht
    (Körpergröße - 100) als Grundlage verwendet.</html>");
    //...//
}

```

Listing 3 – jPanelWerte.updateScreen

Diese Methode wird von verschiedenen Listnern aufgerufen, die die Aktionen von Maus, Checkboxen und Tastatur überwachen.

5.2.2.2 Auswahl Lebensmittel

Wenn die Dateneingabe abgeschlossen ist erfolgt der zweite Schritt, die Auswahl der Lebensmittel.

Abbildung 11 – Ernährungsplan Schritt 2

In diesem Bildschirm kann der Benutzer für jede der drei Tagesmahlzeiten auswählen, auf welche Lebensmittel er auf keinen Fall verzichten will. Diese werden dann im endgültigen Menüplan sicher enthalten sein. Man könnte hier auch gar nichts auswählen. Das Programm ergänzt dann die Auswahl zu dem endgültigen Menüplan. Zu einer vollwertigen Mahlzeit gehört jeweils ein Element aus jeder Gruppe. Es können aber auch, wie hier bei den Milchprodukten, mehrere Elemente pro Gruppe mittels der Shift- oder der STRG- Taste ausgewählt werden.

Falls hier so viele Lebensmittel angegeben werden dass ihre Summe größer ist als der Tagesbedarf an Kalorien das verlangen würde, wird das Menü nicht optimal sein. Jedoch ist das dann die eigene freie Entscheidung des Benutzers.

Für diese Auswahl ist die Anzeigeklasse „JPanelAuswahl“ verantwortlich. Diese zeichnet auch die einzelnen Textboxen. Der Inhalt der Textboxen wird über die Factory-Klasse „JListModelFactory“ generiert. Diese wird mit dem entsprechenden Gruppenindex beauftragt das Datenmodell für die jeweilige Textbox zusammenzustellen und zurückzugeben.

```
JList jList1 = new JList( myJListModelFactory.createGroup(0) );
```

Listing 4 – JList

In der „createGroup()“ Methode der Factory Klasse selbst werden dann die Elemente mit dem Gruppenindex aus der Liste der gesamten Lebensmittel ausgesucht und dem Datenmodell hinzugefügt. Am Ende wird dann das Modell zurückgeben und in der Anzeigeklasse der Textbox zugeordnet.

```

public Vector createGroup(int zielGruppe)
{
    //...//
    //Durchlaufen der Lebensmittelliste
    for (int i = 0; i < Eatables_.size(); i++)
    {
        Element myElement = (Element) Eatables_.get(i);
        temp = new Integer(myElement.getChild("GruppenID").getText());
        elemGruppe = temp.intValue();

        // falls das Element in der Lebensmittelliste zu der gesuchten Gruppe
        // gehört, wird es dem Datenmodell hinzugefügt
        if (elemGruppe == zielGruppe_)
        {
            // ... //

            jlistModelData.add(tmpstr);
        }
    }
    return jlistModelData;
}

```

Listing 5 – jListFactory.createGroup

5.2.2.3 Anzeige des Menüplans

Nachdem die Auswahl durch den Benutzer abgeschlossen wurde, errechnet das Programm den endgültigen Menüplan. Dieser muss aus jeder Gruppe mindestens ein Lebensmittel enthalten, und natürlich möglichst genau den Tagesbedarf des Benutzers decken.

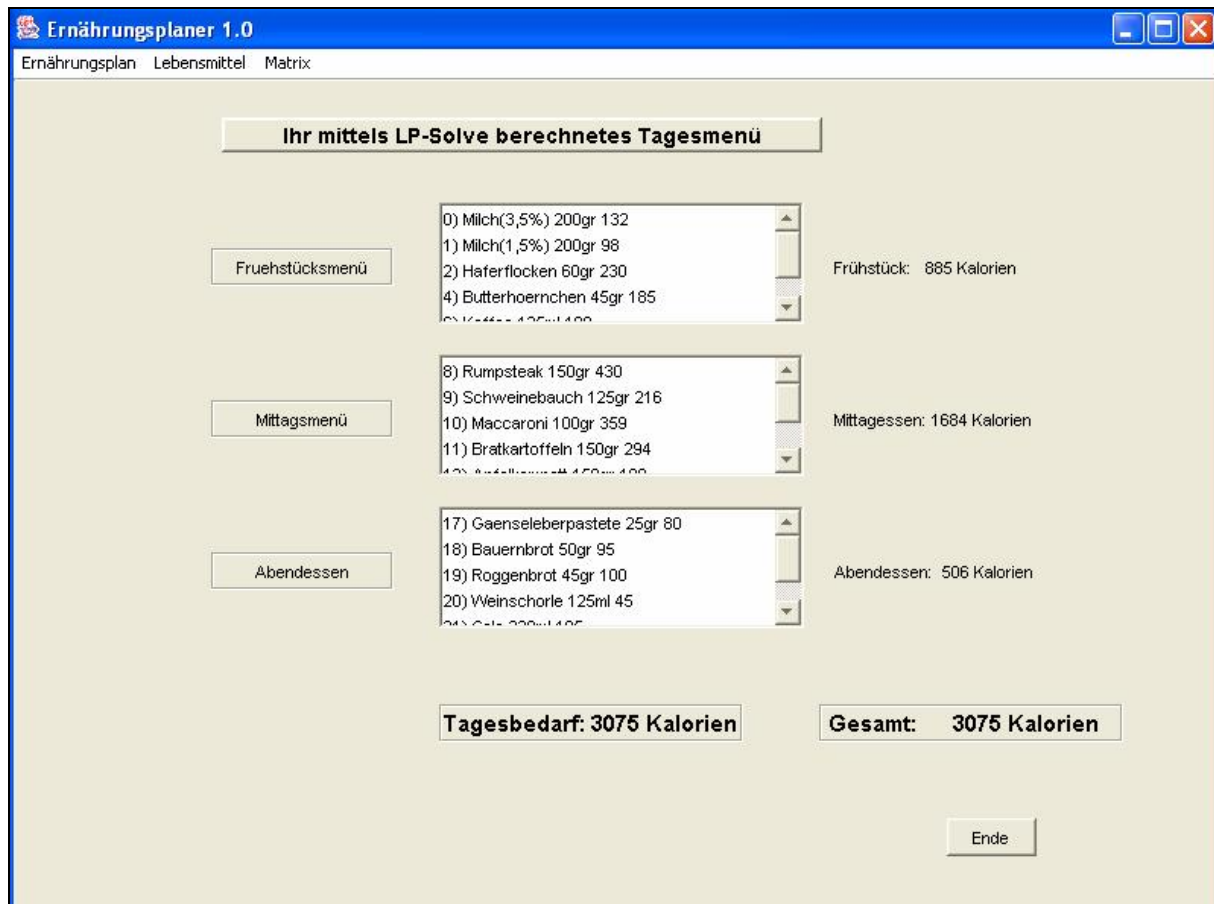


Abbildung 12 – Ernährungsplaner Schritt 3

Das Tagesmenü wird nach Frühstück, Mittag- und Abendessen sortiert ausgegeben. Innerhalb der Mahlzeiten sind die Lebensmittel in den Gruppen wie auf der vorherigen Auswahlseite dargestellt. Dazu werden die enthaltenen Kalorien der jeweiligen Mahlzeit angezeigt. Zum Schluss wird dann der errechnete Tagesbedarf den in den Menüs enthaltenen Kalorien gegenübergestellt.

Falls vor der Berechnung die Anzeige der Matrix ausgewählt wurde, wird diese hier in einem Textfenster mit Erläuterungen angezeigt. Mit dem Button „Ende“ kommt man zurück zum Starbildschirm.

Dieses letzte Fenster des Zyklus zur Ernährungsplanerstellung wird von der Klasse „JPanelAusgabe“ dargestellt. Die Inhalte der Textfenster werden auch hier wieder durch die Factory Klasse „JListModelFactory“ bereitgestellt.

```
public Vector createMenue(String menue)
{
    // ... //
    // die Liste der vom Solver berechneten Menüelemente wird durchlaufen
    for (int i = 0; i < Eatables_.size(); i++)
    {
        if(Eatables_.get(i)!=null)
        {
            //...//
            // Falls das Element im Frühstück enthalten ist wird es
            // hinzugefügt
        }
    }
}
```

```

        if (menue_.equals("Fruehstueck"))
        {
            if ( elemGruppe < 4 )
            {
                // ... //
                jlistModelData.add(tmpstr);
            }
            //... // Entsprechend für die anderen beiden Mahlzeiten
        }
    }
    return jlistModelData;
}

```

Listing 6 – JListFactory.createMenue

Diesmal aber wählt die Methode „createMenue“ die passenden Elemente aus. Sie durchläuft die Liste der Lebensmittel, die, vom Solver errechnet, in den Menüs vorkommen. Die Elemente werden der entsprechenden Mahlzeit sortiert hinzugefügt, das fertige Datenmodell an die Anzeigeklasse zurückgegeben und dargestellt.

5.2.3 Lebensmittelliste bearbeiten

Die Liste von Lebensmitteln ist dynamisch veränderbar. Man kann beliebig Elemente hinzufügen oder löschen. Die Eingaben werden vor dem Eintragen in die Liste geprüft, um die Liste konsistent zu halten.

5.2.3.1 Lebensmittel hinzufügen

Um ein neues Lebensmittel anzulegen müssen alle vorgesehenen Felder gefüllt werden, und zwar mit Namen, Menge (in Gramm oder Millilitern), den darin enthaltenen Kalorien und der Zuordnung zu einer Lebensmittelgruppe. Die Lebensmittelgruppe wird aus einem Drop-Down Menü ausgewählt. Das erspart dem Benutzer eine Texteingabe und so wird sichergestellt, dass die betreffende Gruppe schon existiert. Als Eingaben in den letzten Beiden Feldern sind nur ganze Zahlen erlaubt.

The screenshot shows the 'Ernährungsplaner 1.0' application window. The title bar is blue with the text 'Ernährungsplaner 1.0' and standard window controls. Below the title bar is a menu bar with 'Ernährungsplan', 'Lebensmittel', and 'Matrix'. The main area is a light beige color. At the top, there is a button labeled 'Bitte geben Sie die Daten des neuen Elements ein'. Below this, there are four input fields: 'Name' with the text 'Fanta', 'Lebensmittelgruppe' with a dropdown menu showing 'Getränk_mittagessen', 'Menge (gr/ml)' with the text '330', and 'Kalorien' with the text '150'. At the bottom, there are three buttons: 'Verwerfen', 'Übernehmen', and 'Schliessen'.

Abbildung 13 – Element hinzufügen

Mit einem Klick auf „Verwerfen“ werden alle Felder gelöscht und das Drop Down Feld auf den Ursprungszustand zurückgesetzt. Falls der Benutzer mit Eingabe fertig ist kann er mit einem Klick auf den Button „Übernehmen“ das Element hinzufügen. Das Programm prüft nun die Richtigkeit der Angaben und gibt im Fehlerfall die entsprechenden Fehlermeldungen. Falls eine Angabe fehlen sollte warnt es den Benutzer mit der Meldung dass nicht alle Felder ausgefüllt wurden.

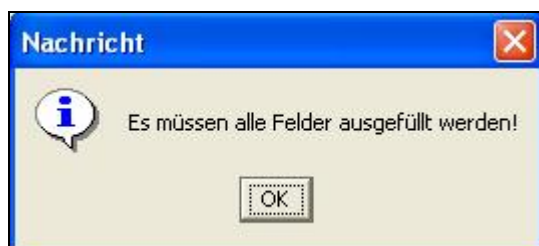


Abbildung 14 – Hinzufügen Fehler leer

Falls die Felder „Menge“ und „Kalorien“ mit falschen Datentypen befüllt wurden weist das Programm auch daraufhin.



Abbildung 15 – Falscher Datentyp

Die Fehlermeldungen müssen durch ein „OK“ bestätigt werden und können dann korrigiert werden.

Ein Klick auf den „Schliessen“ Button beendet die Eingabe und führt zum Startfenster zurück, nicht übernommene Eingaben gehen verloren!

Im Code findet man dieses Panel in der Klasse „jPanelLebensmittelhinzufuegen“. Interessant hier sind die Funktionen „Entries2Element()“ und die Implementierung des „Übernehmen“ Buttons.

```
public Element Entries2Element()
{
    int gruppenID;
    String name, amount, calories;
    name = jTextFieldname.getText();
    gruppenID = jComboBoxGruppe.getSelectedIndex();
    amount = jTextFieldMenge.getText();
    calories = jTextFieldKalorien.getText();
    // Aufruf er Application Methode
    return ap_.newEatable(name, gruppenID, amount, calories);
}
```

Listing 7 – jPanelLebensmittelhinzufuegen.Entries2Element()

Die Methode sammelt alle Daten und sendet sie in der Rückgabe an das ApplicationDietplaner Objekt. Dessen Methode liefert dann das neue „Element“ zurück und übergibt es auch gleich außen weiter.

```
void jButtonUebernehmen_actionPerformed(ActionEvent e)
{
    if (//kein Feld leer){
        if( // alle Felder gültig){
            try{
                //neues Element anlegen, im Fehlerfall ausgaben erzeugen
                Element newElem=this.Entries2Element();
                if(!(this.alreadyexists(newElem))){
                    ap_.addEatable(newElem);
                    JOptionPane.showMessageDialog(null,
                        "Lebensmittel " +jTextFieldname.getText() +" erfasst!");
                }
                else{
                    JOptionPane.showMessageDialog(null,"Lebensmittel existiert
                    schon!");
                }
            }
            catch (DietplanerException e1) {
                e1.getMessage();
            }
            else {
                JOptionPane.showMessageDialog(null,"Nur ganze Zahlen als
                Eingabe erlaubt!");
            }
        }
        else {

```



```
JOptionPane.showMessageDialog(null,"Es müssen alle Felder ausgefüllt werden!"); } }
```

Listing 8 – jPanelLebensmittelhinzufuegen. jButtonUebernehmen()

5.2.3.1 Lebensmittel löschen

Um ein Element wieder zu löschen kann man es aus der Liste der bestehenden Elemente auswählen. Dies erspart langes Eingeben und Prüfen. Im Übrigen werden dadurch ähnliche Elemente vor dem löschen durch Verwechslung gelöscht.

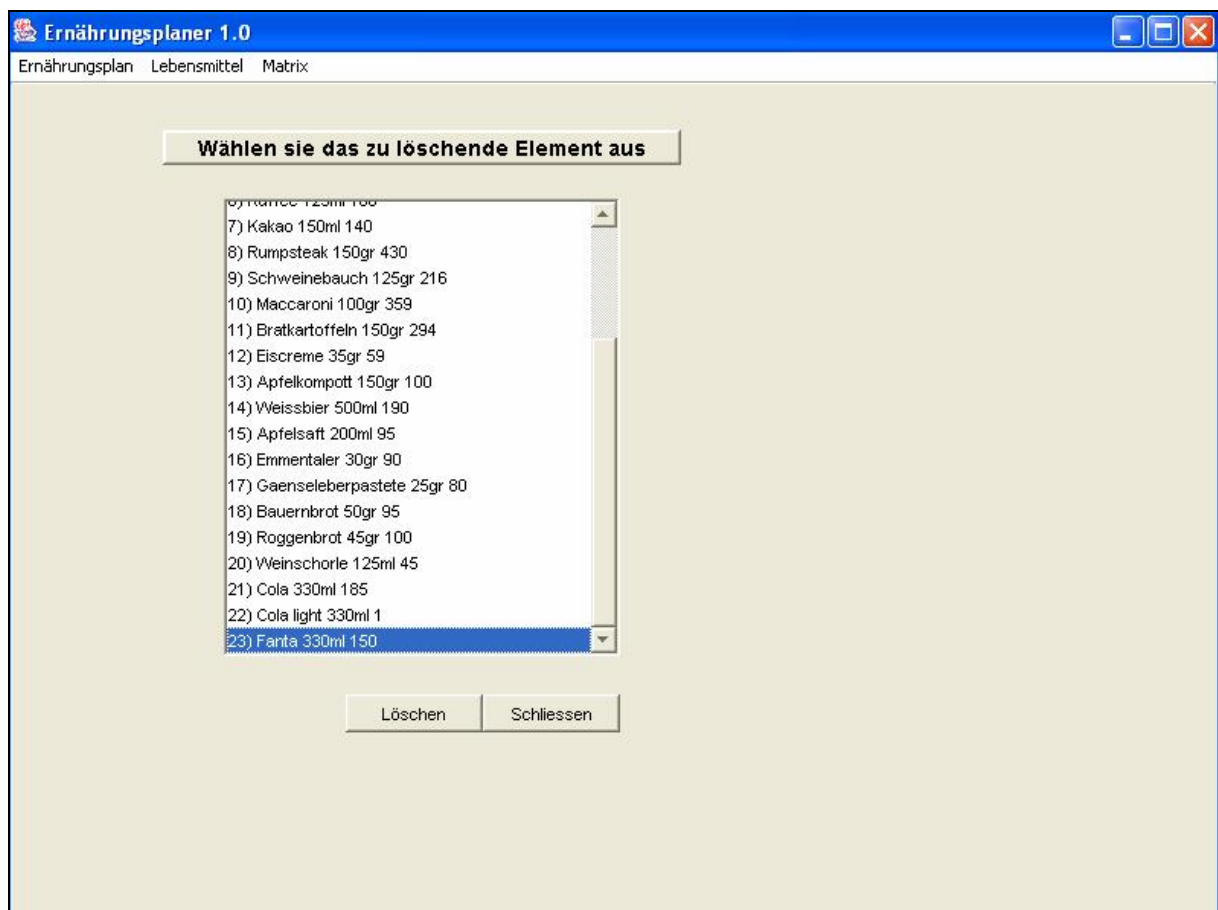


Abbildung 16 – Element löschen

Nach einem Klick auf den „Löschen“ Button wird das Element aus der Liste gelöscht. Die Liste wird aktualisiert und man erhält eine Bestätigung.



Abbildung 17 – Löschbestätigung

Mit einem Klick auf den „Schließen“ Button wird diese Maske beendet und zum Startbildschirm zurückgekehrt.

Die wichtigste Funktion in der hierfür zuständigen Klasse „JPanelLebensmittelloeschen“ ist wohl die Implementierung des Löschbuttons selbst. Der Inhalt der Liste wird wie immer durch die Klasse „JListModelFactory“ bereitgestellt.

```
void jButtonLoeschen_actionPerformed(ActionEvent e){
    int sel = jListZuLoeschen.getSelectedIndex();
    ListModel lm = jListZuLoeschen.getModel();
    //...//
    ID = temp.intValue();
    elem = (Element) allEatables_.get(ID);
    //Löschvorgang
    try {
        JOptionPane.showMessageDialog(null, "Das Lebensmittel "+value+" wurde
        erfolgreich geloescht!");
        //Aufruf der ApplicationNDiaetplaner Methode um es aus der Datei zu
        //loeschen
        ap_.deleteEatable(elem);
        myJListModelFactory= new JListModelFactory(allEatables_);
        jListZuLoeschen.setListData(myJListModelFactory.createAll());
    }
    catch (DiaetplanerException ex) {
        ex.getMessage();
    }
}
```

Listing 9 – JPanelLebensmittelloeschen. jButtonLoeschen()

5.3 Solver Schnittstelle

Bei der Solver Schnittstelle wurde besonders darauf Wert gelegt, dass sehr leicht andere Solver mit eingebunden werden können. Zu diesem Zweck wurde ein Interface entwickelt, dass die Änderung aller Daten ermöglicht, die nötig sind, um einen Solveraufruf zu starten:

```
public interface Solver
{
    public double[] calcModel(Matrix lpModell, Matrix grenzen);
    public double[] calcModel(Matrix lpModell);

    public String getTempVerzeichnis();
    public void setTempVerzeichnis(String tempVerzeichnis);

    public String getLpSolvVerzeichnis();
    public void setLpSolvVerzeichnis(String lpSolvVerzeichnis);
}
```

Listing 10 – Solver Interface

Am Quellcode kann abgelesen werden, dass das Solver- und Tempverzeichnis dynamisch (auch zur Laufzeit) angepasst werden kann. Zum Berechnen muss der Methode calcModel() **zwei Matrizen** in folgender Form übergeben werden:

| | x 1 | x 2 | | b |
|----------------------|-------------|-------------|----------------|--------------|
| Zielfunktion | 1 | 2 | ---> | max ! |
| Restriktion 1 | -2.1 | 1 | < | 3 |
| Restriktion 2 | 1 | .2 | < | 8 |
| Restriktion 3 | 4 | -1.5 | < | 0 |

Abb. 18: 1. Matrix für LP Modell

| Variable | untere Grenze | obere Grenze | Ganzzahl |
|----------|---------------|--------------|----------|
| x 1 | 1 | 5 | nein |
| x 2 | 0 | 0 | ja |

Abb.19: 2. Matrix für Grenzen und Ganzzahl

Wenn keine Grenzen und Ganzzahligkeit zum Lösen des LP Modells notwendig sind, kann die Grenzen Matrix auch weggelassen werden.

Zum Erstellen einer Matrix wurde eine Klasse „Matix.java“ entwickelt, diese Klasse kann eine Matrix dynamisch erweitern und nimmt zum Füllen der Felder je nach Beliebigen Ganzzahlen, Kommazahlen oder Zeichenketten entgegen.

Für die erste Version des Ernährungsplaners wurde die Klasse „LP_Solve.java“ entwickelt, welche den Solver „LP Solve“ implementiert. Soll ein weiterer Solver in das Programm mit eingebunden werden, muss nur das Interface Solver implementiert werden. Die Matrizen bleiben wie gehabt und müssen nur übergeben werden.

Zum Berechnen der Matrix mit LP Solve wird im Temp Verzeichnis die Inputdatei (diaet.lp) für den Solver erzeugt. Für obiges Beispiel hat die Inputdatei folgendes Format:

```
max: 1x1 + 2x2;

R1: -2.1 x1 + 1 x2 < 3;
R2: 1 x1 + .2 x2 < 8;
R3: 4 x1 + -1.5 x2 < 0;
x1>=1;
x1<=5;

int x2;
```

Listing 11 – Inputdatei für LP Solve

Nach Erzeugen der Inputdatei erstellt die Solverklasse eine „diaet.bat“ Datei im Temp Verzeichnis, um den Solver mit folgendem Aufruf zu starten:

```
lp_solve.exe" -p <c:\temp\diaet.lp >c:\temp\diaet.out
```

Dieser Aufruf wird in einem eigenen Thread gestartet, damit ein fehlerloses Schreiben den Ergebnisdatei (diaet.out) sichergestellt ist. Der Vaterthread wartet so lange, bis die Ergebnisdatei fertig geschrieben im Outputformat von LP Solve vorliegt:

```
Value of objective function:          30.875
x1                                4.875
x2                                13
These are the duals from the node that gave the optimal solution.
R1                                0
R2                                0
R3                                0.25
```

Listing 12 – Outputdatei von LP Solve

Die Solverklasse parst diese Datei und gibt ein Array mit den ausgelesenen Ergebnissen zurück.

Der interessierte Leser kann die Klasse „SolverMain.java“ genauer anschauen, diese Beispielklasse erstellt eine kleine Matrix (mit dem hier aufgeführten Beispiel) und ruft den LP Solve auf.

5.4 Anwendung und Datenbankankbindung

Die Anwendung und Datenbankankbindung sind in zwei Klassen implementiert. Die Klasse DiätplanerApplication enthält die Anwendungslogik, die Klasse XMMListhandler kümmert sich um die Datenbank, in unserem Fall ein XML-File. Die wichtigsten Methoden der beiden Klassen werden in den folgenden Unterkapiteln beschrieben.

5.4.1 Die Klasse DiätplanerApplication

Die Klasse ApplicationDiätplaner bildet das Herzstück der Anwendung. Sie enthält die Mainfunktion, die das Programm startet und die graphische Oberfläche aufbaut. Zusätzlich triggert sie die wichtigsten Operationen auf der Datenbank bzw. dem Solver. Zur Verwaltung der Datenbank arbeitet sie mit der Klasse XMMListhandler zusammen, die wiederum die direkten Zugriffe auf das XML-File durchführt. Zur Berechnung benützt Sie das Interface „Solver“, um die Berechnung auf dem Solver zu starten. Die wichtigsten Informationen der Anwendung werden in drei Vektoren gespeichert. Der Vector AllEatables enthält alle Lebensmittel, die in der Datenbank verfügbar sind und wird beim Programmstart initialisiert. Der Vector ChosenEatables enthält alle Lebensmittel, die vom Benutzer ausgewählt wurden und die deshalb auf jeden Fall im Menü vorkommen müssen. Der Vector menu enthält schließlich nur noch die Lebensmittel, die das ausgerechnete Tagesmenü bilden. Die wichtigsten Methoden der Klasse ApplicationDiätplaner und Ihre Funktionsweise werden im Folgenden erklärt.

Die Methode addEatable()

Die Methode addEatable(Element Eatable) fügt neue Lebensmittel in die Datenbasis ein. Sie ruft dafür die Methode addEatable(Element Eatable) des XMMListhandlers auf, der daraufhin seine interne Liste aktualisiert. Anschließend wird die neue Liste über saveToXMLFile() in das XML-File übernommen. Danach wird noch der AllEatables-Vector aktualisiert, der alle verfügbaren Lebensmittel enthält.

Die Methode deleteEatable()

Die Methode deleteEatable(Element Eatable) löscht Lebensmittel aus der Datenbasis. Sie funktioniert nach dem gleichen Prinzip wie addEatable(). Zuerst wird die Methode deleteEatable(Element Eatable) des XMMListhandlers aufgerufen, anschließend wird saveToXMLFile() aufgerufen und der AllEatables-Vector aktualisiert.

Die Methode executeCalculation()

executeCalculation(Vector Chosen Eatables, in calories) führt die Berechnung des Tagesmenüs auf dem Solver durch. Der Vector Chosen Eatables enthält die aus-

gewählten Lebensmittel des Benutzers, die Variable calories enthält den ausgerechneten Mindesttagesbedarf.

Im ersten Schritt wird aus den verschiedenen Informationen Lebensmittel, Lebensmittelgruppen, ausgewählte Lebensmittel, Tagesbedarf und Kalorienanzahl der einzelnen Lebensmittel eine Matrix erzeugt, die anschließend der Solverschnittstelle zur Berechnung übergeben wird. Die Zusammensetzung der Matrix entspricht dabei dem LP-Modell, das in Kapitel 4.3 näher beschrieben wird. Das Ergebnis des Solvers wird in den Vector menu geschrieben, dieser Vector enthält jetzt nur noch die Lebensmittel, die für das Tagesmenü bestimmt sind.

5.4.2 Die Klasse XMLListHandler

Die Klasse XmlListHandler bietet die Möglichkeit, ein XML-Dokument zu öffnen, das XML-Dokument gegen ein Schema zu validieren und dessen Inhalt in einem JDOM Baum zu speichern. Dieser JDOM-Baum ist eine Abbildung des XML-Files auf eine interne Baumstruktur und bietet die Möglichkeit, innerhalb des XML-Files zu navigieren.

Unter verschiedenen Hilfsmethoden ist die Methode getList() die Wichtigste, die eine Referenz auf eine Liste zurückgibt. Diese Liste enthält alle Lebensmittel des XML-Files als einzelne Elemente. Werden Elemente, sprich Lebensmittel, über diese Liste geändert, werden diese Änderungen direkt am JDOM-Baum durchgeführt. Über die Methode saveXMLList() werden die Änderungen vom JDOM-Baum in das XML-File übernommen. Um Elemente innerhalb des JDOM-Baumes zu löschen oder hinzuzufügen werden die Methoden addElement() und deleteElement() angeboten.

6 Verwendete Technologien

Für die Entwicklung unseres Programms haben wir folgende Tools verwendet:

ENWICKLUNSWERKZEUG:

-JBuilder 9:

Die Personal Edition wird unter

http://www.borland.com/products/downloads/download_jbuilder.html

zum Download angeboten. Es handelt sich dabei um eine freie Vollversion zur nicht-kommerziellen Nutzung.

-Inno Setup 4.0.11:

Bei Inno-Setup handelt es sich um ein Open-Source-Installationsprogramm, das unter www.jrsoftware.org heruntergeladen werden kann. Mithilfe des Programms können dann Installationsdateien für eigene Programme erstellt werden.

XML-EDITOR:

-XMLSpy:

Zwar lässt sich XML auch in einem normalen Editor erstellen und bearbeiten, jedoch ist die Arbeit mit XML in einem speziellen XML-Editor viel komfortabler. Oft ermögli-

chen XML-Editoren nicht nur die einfache Erstellung von gültigen XML- oder XSL-Dateien, sondern können auch XML-Bäume validieren (= auf Gültigkeit prüfen) und bieten Unterstützung für die Generierung von XML-Schematas aus dem XML-Code. Für das Erstellen, Editieren und Validieren unserer XML-Dateien verwendeten wir deshalb den XML-Editor XML-Spy der Version 2004 von der Altova GmbH. Eine zeitlich begrenzte, aber voll einsatzfähige Probeversion kann man unter www.altova.com herunterladen.

7 Administration

7.1 Installation

Beim Installieren des Ernährungsplaners sind 3 mögliche Installationstypen zu unterscheiden. Die Installations CD ist in jedem Fall erforderlich.

1. Installation mit dem mitgelieferten Setup Programm (siehe 7.1.1)
2. Installation von Hand (ohne Setup) (siehe 7.1.2)
3. Installation der Sourcen (nur für Folgeprojekte oder Quellcodeänderungen nötig – siehe 7.1.3)

7.1.1 Installation mit Setup

Die Installation mit Setup funktioniert nur unter Microsoft Windows. Beim Einlegen der Installations CD wird die „Setup.exe“ im Root Verzeichnis der CD automatisch aufgerufen. Ist die Autorun Funktionalität des Betriebssystems ausgeschaltet, muss sie von Hand gestartet werden.

Als Standard Installationspfad wird „c:\Programme\Ernaehrungsplaner“ vorgeschlagen. Optional kann ein Icon auf dem Desktop erstellt werden. Die Deinstallation ist wie unter Windows üblich unter Systemsteuerung – Software – Ernaehrungsplaner möglich. Es werden alle Dateien gelöscht ausser der Lebensmittel.xml. Beim Installieren wird (falls nötig) das Verzeichnis „C:\Temp“ angelegt, in dem die Berechnung des Solvers gespeichert wird.

Wird der Ernährungsplaner mit dem Setup ein zweites mal auf den Rechner installiert (in das gleiche Verzeichnis), werden alle Dateien überschrieben mit der Ausnahme von Lebensmittel.xml, dadurch wird verhindert, dass ein erneuter Setup Aufruf die aktuelle Lebensmittelliste überschreibt.

Das Programm kann mit den erstellten Verknüpfungen im Startmenü (Programme – Ernaehrungsplaner – Ernaehrungsplaner) oder auf dem Desktop gestartet werden.

7.1.2 Installation von Hand

Bei der Installation von Hand muss von der Installations CD das Verzeichnis „Exec“ komplett (mit Unterverzeichnissen) lokal auf die Festplatte kopiert werden. Im Installationsverzeichnis kann dann die Batch Datei start.bat aufgerufen werden. Das Verzeichnis „c:\temp“ muss von Hand angelegt werden.

Die Installation von Hand wird nicht empfohlen, sie ist nur nötig wenn das Programm z.B. unter einem anderen Betriebssystem wie Linux getestet werden soll.

Wir haben das Programm ausschließlich unter Windows programmiert, ausgeführt und getestet, für ein fehlerfreies Verhalten unter anderen Betriebssystemen kann kei-

ne Aussage gemacht werden. Auf jeden Fall müsste die Batch Datei zum Beispiel für ein Unix-System in ein Shellsript umprogrammiert werden.

7.1.3 Installation der Sourcen

Das Verzeichnis „Sourcen“ muss lokal auf die Festplatte kopiert werden. Es enthält ein JBuilder9 Projekt „diaetplaner.jpx“ das mit dem JBuilder geöffnet werden kann (getestet mit Version 6 und 9).

Sehr Wichtig ist das richtige Einstellen der benötigten Libraries im JBuilder, wird zu viel oder zu wenig eingebunden ist kein kompilieren möglich.

Unter “\Sourcen\Libraries\ Benötigte Library Projekt.doc” auf der Installations CD sind die einzeln benötigten Libraries aufgeführt. Wichtig ist das Befolgen des Abschnitts „Anmerkungen“ der Datei „Benötigte Library Projekt.doc“.

Die Libraries sind im Verzeichnis „\Sourcen\Libraries“ verfügbar.

7.2 Zusammensetzung der Applikation

Bei der Installation wird die Java Runtime Version 1.4.1_02 im Installationsverzeichnis mit auf die Festplatte kopiert. Es werden keinerlei System- oder Registryeinträge vorgenommen, dadurch werden Konflikte mit anderen Java Versionen vermieden.

Die Java Runtime wird in der Batch Datei (start.bat) direkt über

SET Path = j2re1.4.1_02\bin;%PATH%

in die Laufzeitumgebung eingebunden. Nach einbinden der benötigten Runtime wird in der Batch Datei die entsprechende Java Klasse aufgerufen, um die Applikation zu starten.

Benötigte Dateien zum Ausführen der Ernährungsplaner Applikation:

1. \ernaehrungsplaner.jar
Dies ist das Herzstück der Applikation, es enthält alle Quellen und Libraries in kompilierter Form.
2. \lebensmittel.xml
XML Datei in der alle Lebensmittel gespeichert sind.
3. \solver\lp_solve.exe
Eingebundener Solver um das LP Model zu lösen.

8 Projekterfahrungen und Aufwand

Wir waren ein sehr gutes Team und haben von Anfang an viel Wert auf eine gute Organisation und die Definition von Schnittstellen gelegt, um paralleles Arbeiten zu ermöglichen, dies hat sich im Laufe des Projektes auch ausgezahlt. Durch die klare Trennung in Komponenten waren die Verantwortlichkeiten klar getrennt, außerdem konnten wir damit die Stärken und Kenntnisse jedes einzelnen sehr gut ausnützen. Armin war für die grafische Oberfläche zuständig, Matthias kümmerte sich um die Anwendungslogik und Datenbankbindung, Steffen war für die Solveranbindung zuständig. Zusätzlich kümmerte sich Steffen noch um Teile der Anwendungslogik (Die Berechnung des BMI und des Tagesbedarfs) und die Installationsroutine. Der Aufwand für die einzelnen Projektmitglieder war ungefähr gleich und errechnet sich für jedes Projektmitglied aus folgenden Teilen:

| | |
|--|----------------|
| - Analyse des bisherigen Programms: | 3 Stunden |
| - Recherche nach Berechnungen, Kalorientabellen, Quellen | 7 Stunden |
| - Organisation und Meetings | 4 Stunden |
| - Programmierung | 20 Stunden |
| - Ausarbeitung | 10 Stunden |
| - Präsentationen | 1 Stunde |
| - Gesamt | 45 Stunden |

9 Ausblick

Das Programm bietet natürlich noch Ansatzmöglichkeiten für Erweiterungen.

Für eine genaue Ernährungsplanung könnte man noch die Fett- und Spurenelementanteile eines jeden Lebensmittels in die Berechnung einbeziehen. Das würde aber wohl mehr Eingabedaten erfordern, ob mehr oder weniger Lebensmittel mit Fetten (vielleicht auch Unterscheidung zwischen tierischen und pflanzlichen Fetten) in die Menüplanung einfließen sollen. Denn aus den reinen Kalorien ist kein Rückschluss auf den Bedarf einer Person an speziellen Inhaltsstoffen eines Lebensmittels möglich.

In unserem Programm ist nur die Eingabe von gewünschten Lebensmitteln vorgesehen. Man könnte auch leicht einbauen, dass bestimmte Elemente nicht gewünscht sind und auf keinen Fall im Endmenü enthalten sein dürfen.

Denkbar wäre auch diesen Ernährungsplaner als Einzelmodul zu sehen, um eine wirkliche Diät zu planen. Das könnte so aussehen, dass man angeben kann wie viel man in einer gewissen Zeitspanne abnehmen möchte und das Programm errechnet dann die Tagespläne für die entsprechenden Tage. Natürlich muss dann jeden Tag das Gewicht vom Programm angepasst und nachgeführt werden. Auch muss dann eine genaue Analyse der Person getroffen werden, denn Diäten wirken nicht bei jedem Menschen gleich.

Ebenfalls könnten andere Solver in das Programm mit eingebunden werden, es müsste nur das Solver Interface genutzt werden.