Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Studienarbeit

Dirk Jagdmann <dirk@jagdmann.de>

Entwicklung eines Abrechnungssystems für Rheumatherapien

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Dirk Jagdmann

Thema der Studienarbeit
Entwicklung eines Abrechnungssystems für Rheumatherapien

Stichworte
Client-Server, Mobile Clients, Abrechnungssystem, Verschlüsselung

Kurzzusammenfassung
Die Rheuma Liga Hildesheim, ein gemeinnütziger Verein, bietet ihren Mitgliedern Therapien zur Linderung der Rheumaerkrankung an. Die Therapien werden von Therapeuten in Einzel- oder Gruppensitzungen durchgeführt. Die Behandlungskosten werden von den Krankenkassen erstattet. Für die Abrechnung dieser Leistungen zwischen der Rheuma Liga und den Krankenkassen soll ein Computersystem erstellt werden, welches die einzelnen Behandlungen der Mitglieder erfasst und einmal pro Quartal eine Abrechnung für die Krankenkassen erstellt.

Diese Studienarbeit zeigt, wie die Anforderungen der Rheuma Liga in einen funktionsfähigen Prototyp umgesetzt wurden. Der Prototyp umfasst stationäre und mobile Computer für die Benutzer des Systems und einem zentralen Server. Durch die Verwendung von Standardkomponenten wurde der Implementierungsaufwand minimiert und ein einheitliches System geschaffen, das auf unterschiedlichen Hardware- und Softwareplattformen betrieben werden kann.

Dirk Jagdmann

Title of the paper
Development of a medical accounting system

Keywords
Client-Server, Accounting, Internet technology, Cryptography, Mobile computer, Relational database

Abstract
In Hildesheim/Germany a non-profit society offers his members therapies against articular rheumatism. The therapies are paid by the medical insurance. Currently the accounting for the insurance is rendered manually. The society wants to build an electronic accounting system to acquire the therapy data and calculate the invoice for the insurance.

This paper presents a Client-Server system which collects data on therapies and members and generates invoices for all involved parties. As multiple users will use the system concurrently the server must be able to server many client simultaneously. The client software must be platform independent, because it will be used on various platforms including mobile computers and Unix/Linux workstations. By use of standard software components the amount of self written software components could be minimized, thus creating a inexpensive solution.

Inhaltsverzeichnis

1 Problemstellung

Die Rheuma-Liga Hildesheim (RHL) ist ein gemeinnütziger Verein, der für seine Mitglieder Therapien zur Rheumabehandlung anbietet. Es gibt unterschiedliche Therapieformen, z. B. Wassergymnastik im Schwimmbad oder Trockengymnastik in der Sporthalle. Die Therapien finden in der Regel in Gruppen statt, die von einem einzelnen Therapeuten geleitet werden. Der Therapeut wird von der RHL bestellt und pro Stunde, bzw. Gruppensitzung bezahlt.

Für jedes Mitglied wird von seiner Krankenkasse eine befristete Genehmigung zur Kostenübernahme einer oder mehrerer Therapieformen erteilt. Nur wenn eine solche Genehmigung vorliegt, erstattet die Krankenkasse die Kosten einer Therapie. Die Genehmigung legt außerdem fest, wie viele Therapien das Mitglied in der Woche besuchen darf, in der Regel zwei bis drei. Möchte das Mitglied an mehr Therapien teilnehmen oder liegt keine Genehmigung vor, dann muß die Therapie vom Mitglied privat bezahlt werden.

Jede der gesetzlichen Krankenkassen legt individuell fest, welchen Preis sie für eine Therapieform erstattet. Der von der RHL in Rechnung zu stellende Preis ist folglich von der Therapieform und der Krankenkasse des Mitgliedes abhängig. Privat versicherte Mitglieder zahlen alle den gleichen Betrag pro Therapieform.

In feststehenden Intervallen, normalerweise mit Abschluss eines Quartals, wird für die Krankenkassen eine Abrechnung erstellt, welche die Behandlungen akkumuliert pro Mitglied umfasst. Dabei müssen natürlich die unterschiedlichen Preise der verschiedenen Krankenkassen und Therapieformen berücksichtigt werden. Im Gegensatz dazu bekommen die Therapeuten ein monatliches Gehalt, welches sich nach der Anzahl der durchgeführten Sitzungen richtet.

Da die Kostenerstattung der Krankenkassen jedoch nicht ausreicht, die Therapien zu bezahlen, muß jedes Mitglied pro Therapie eine Zuzahlung leisten. Diese wird dem Mitglied monatlich in Rechnung gestellt. Zusätzlich muß das Mitglied Anfang Januar einen Jahresbeitrag zahlen. Alle Forderungen an Mitglieder werden in der Regel per Lastschrift vom Konto eingezogen.

Um einem Abrechnungsbetrug vorzubeugen, verlangen die Krankenkassen eine Bestätigung für jede in Anspruch genommene Therapie. Das Mitglied muß sich dazu auf einer Liste vor Therapiebeginn eintragen. Anhand dieser Listen erstellt die RHL die Abrechnungen mit den Krankenkassen. Zusammen mit den Abrechnungen wird eine Kopie der in Frage kommenden Listen verschickt, um den Krankenkassen eine Kontrolle der Abrechnungen zu ermöglichen.

Die Abrechnungen werden bislang manuell ohne EDV Hilfe erstellt. Dazu werden am Quartalsende die Unterschriftslisten von den Therapeuten eingesammelt und die Behandlungen für jedes Mitglied akkumuliert. Danach werden für jede Krankenkasse die Behandlungssummen der bei diesen Kassen versicherten Mitglieder akkumuliert. Bei dieser zweimaligen Übertragung und Summation entstehen viele (Rechen)Fehler.

Da die Leistungserbringer (Therapeuten) ihr Gehalt monatlich erhalten, die anfallenden Kosten jedoch nur einmal pro Quartal von den Krankenkassen erstattet werden, muß die RHL für die Therapeutengehälter in Vorleistung gehen. Durch einen starken Mitgliederzuwachs in den letzten Jahren hat sich jedoch der Aufwand zur Erstellung der Abrechnungen so stark vergrößert, dass bis zu 12 Monate zur Fertigstellung einer Quartalsabrechnung vergehen. Die RHL muß also ungefähr ein Jahr auf die Erstattung ihrer Kosten durch die Krankenkassen warten.

Die RHL hat aufgrund dieser Problematik beschlossen, ein EDV System erstellen zu lassen, welches Therapien erfasst und abrechnet. Zusätzlich soll das System eine Mitgliederverwaltung

beinhalten, so dass die Zuzahlungen und Jahresbeiträge ebenfalls vom System erfasst und eingefordert werden können.

2 Analyse

In den folgenden Abschnitten werden die Anforderungen an das System beschrieben. Dabei wird sich zeigen, dass die verschiedenen Subsysteme ganz unterschiedliche Aufgaben zu erfüllen haben und sich daraus große Unterschiede bezüglich Verfügbarkeit, Vertraulichkeit und Funktionsumfang ergeben.

2.1 Use Cases

Zur Festlegung des Funktionsumfangs betrachten wir die unterschiedlichen Use Cases, welche die Arbeitsschritte der Benutzer beschreiben. Sie ermöglichen uns die unterschiedlichen Leistungen zu definieren, die das System anbieten muss. Ausgehend von diesen Use Cases können alle weiteren Anforderungen an das System ermittelt werden.

Es gibt 15 voneinander unabhängige Vorgänge. Die einzelnen Vorgänge können jederzeit durchgeführt werden und müssen nicht in einer speziellen Reihenfolge ausgeführt werden.

1. Ein neues Mitglied wird in die RHL aufgenommen. Dazu müssen die persönlichen Daten des Mitgliedes aufgenommen werden (Adresse, Bankverbindung). Im Anschluss wird ein Anschreiben für das Mitglied erstellt und der (anteilige) Jahresbeitrag eingezogen.

2. Ein Mitglied tritt aus der RHL aus.

3. Änderung der Daten eines Mitgliedes (Adresse usw.)

4. Suchen eines Mitgliedes über Namen, Ort usw.

5. Änderung der Daten einer Krankenkasse, bzw. Eintragen einer neuen Krankenkasse, Löschung einer Krankenkasse. Insbesondere bei Löschungen von Krankenkassen, muss das System die Übernahme der Mitglieder in eine andere Krankenkasse durchführen.

6. Änderung der Behandlungspreise einer Krankenkasse.

7. Ansicht, welche Mitglieder bei einer Krankenkasse versichert sind.

8. Erteilen einer Genehmigung zur Behandlung. Die Genehmigungen werden von den Krankenkassen schriftlich den Mitgliedern zugestellt, welche die Genehmigung dann der RHL vorlegen.

9. Ansicht, welche Genehmigungen für ein Mitglied erteilt sind.

10. Erfassung einer Behandlung. Hierbei ist das Mitglied, die Behandlungsart, das Datum, die Zuzahlung und der Therapeut zu erfassen.

11. Ansicht, welche Behandlungen ein Mitglied in Anspruch genommen hat.

12. Abrechnung der Behandlungen mit den Krankenkassen. Für jede Krankenkasse muss eine Tabelle erstellt werden, wie viele Behandlungen jedes Mitglied der Krankenkasse im Abrechnungszeitraum erhalten hat. Die Rechnung kann den Krankenkassen entweder elektronisch oder als Ausdruck zugestellt werden.

13. Abrechnung für privat versicherte Mitglieder. Die Rechnung wird gedruckt und der Rechnungsbetrag elektronisch per Lastschrift eingezogen. Hat ein gesetzlich versichertes Mitglied Behandlungen ohne Genehmigung durch die Krankenkasse erhalten, so werden die Behandlungen privat abgerechnet.

14. Abrechnung der Zuzahlungen für Mitglieder. Für jede Behandlung müssen die Mitglieder eine Zuzahlung aufwenden, die nicht von der Krankenkasse erstattet wird. Einmal pro Monat werden die Zuzahlungen per Lastschrift eingezogen und eine Rechnung gedruckt.

15. Abrechnung des Jahresbeitrages für Mitglieder. Dieser wird am Jahresanfang von allen Mitgliedern eingezogen. Es sind Ausdrücke zu erstellen. Es gibt verschiedene Beitragsklassen (Mitglied, Familienmitglied, Rentner, Kind) mit unterschiedlichen Beitragssätzen.

2.2 Benutzer

Zwei verschiedene Benutzergruppen werden das System benutzen, Sachbearbeiter und Therapeuten. Die Benutzergruppe der Sachbearbeiter sind Mitarbeiter der RHL. Die Therapeuten führen die Therapien für Mitglieder durch. Wenn im folgenden Text die Begriffe Sachbearbeiter oder Therapeut benutzt werden, so ist damit eine der entsprechenden Benutzergruppe angehörige Person gemeint.

Die folgenden beiden Abschnitte beschreiben die Aufgaben und das Arbeitsumfeld der beiden Benutzergruppen.

2.2.1 Sachbearbeiter

Die Sachbearbeiter sind Angestellte der RHL. Sie arbeiten in den Räumlichkeiten der RHL und benutzen die dort bereits vorhandenen PCs für normale Büroarbeiten. Auf diesen Computern soll das System benutzt werden. Abrechnungen werden auf lokal vorhandenen Netzwerkdruckern gedruckt. Außer den Nummern 8 und 10 muß der Sachbearbeiter alle Use Cases ausführen können.

Die unterschiedlichen Sachbearbeiter sind grundsätzlich vertrauenswürdig und müssen nicht als unterschiedliche Personen im System abgebildet werden.

2.2.2 Therapeut

Die Use Cases 8 und 10 werden von Therapeuten benutzt. Der Therapeut muss sich am System anmelden, damit ihn das System identifizieren kann. Er besitzt einen Account im System, welcher ihn eindeutig identifiziert.

Bei Therapien in Schwimmbädern ist es i. d. R. nicht möglich, einen stationären Computer zu verwenden. Hier wird eine mobile Lösung benötigt. Ein mobiles Gerät kann aber auch in einer

Praxis benutzt werden. Ein Account ist nicht Gerät-gebunden, d. h. ein Therapeut kann sowohl mobile als auch stationäre Geräte verwenden.

Sowohl Therapeuten als auch Sachbearbeiter müssen gleichzeitig in der Lage sein mit dem System zu arbeiten, d. h. neue Daten einzutragen und Abfragen an das System zu erstellen. Die Benutzer arbeiten unabhängig voneinander und bemerken die eventuelle Anwesenheit anderer Benutzer im System nicht.

2.3 Datenspeicherung

Im vorigen Abschnitt wurde gezeigt, dass mehrere Geräte/Computer gleichzeitig mit dem System arbeiten sollen. Alle Computer sollen ferner mit denselben Datensätzen arbeiten. Dies kann am einfachsten durch eine zentrale Verwaltung der Datensätze erreicht werden, d. h. alle Datensätze werden von einem zentralen Computer, *Server* genannt, verwaltet. Die Geräte, die auf diesen Server zugreifen und mit denen die Benutzer arbeiten, werden *Clients* genannt.

Das System bzw. der Server muss über einen langen Zeitraum, mindestens ein Quartal, Daten sammeln. Da nicht gewährleistet werden kann, dass der Server in diesem Zeitraum ohne Unterbrechung betrieben werden kann, müssen die gesammelten Daten persistent gespeichert werden. Den unterschiedlichen Benutzergruppen entsprechen unterschiedliche Berechtigungen, Datensätze zu lesen und ändern. Das Serversystem muss diese Benutzergruppen abbilden und entsprechend Zugriffsrechte vergeben. Vor dem Zugriff müssen sich die Benutzer am Server authentifizieren, wodurch ihnen eine Gruppe und damit die Rechte zugeordnet werden können.

2.4 Revisionen/Zeit/Backup/Volumina

Die in die Abrechnungen einfließenden Daten werden über einen langen Zeitraum gesammelt, i. d. R. ein Quartal. Nach Abschluss dieses Abrechnungszeitraums werden die unterschiedlichen Abrechnungen erzeugt bzw. gedruckt. Das System ermöglicht den Zugriff auf alle vergangenen Abrechnungszeiträume und erlaubt es weiterhin, die diesen Zeiträumen entsprechenden Rechnungen beliebig oft zu drucken. Das System muss folglich sicherstellen, dass die für Abrechnungen relevanten Daten mit dem Abschluss eines Abrechnungszeitraumes nicht mehr geändert werden können.

Alle Daten, die Abrechnungen nicht beeinflussen[1], können jedoch sehr wohl geändert werden. Die vorhergehenden Werte müssen nicht erhalten bleiben, diese Datensätze müssen nicht revisionssicher sein. Die Datenbank enthält bei diesen Datensätzen immer den aktuellen Stand. Die für Abrechnungen relevanten Daten können dagegen weder geändert noch gelöscht werden, nachdem sie ins System eingetragen wurden. Dadurch ist sichergestellt, dass Abrechnungen nicht nachträglich verändert werden bzw. von vornherein falsche Abrechnungen erzeugt werden.

Für Server- und Clientcomputer muss ein Störungsmanagement existieren, welches Maßnahmen gegen alle potentiellen Störungsursachen enthält. Störungen sind: Datenverlust durch Hardware- oder Bedienfehler, Ausfall einzelner oder aller Hardwarekomponenten, Störung/Ausfall der (Netzwerk)Verbindung zwischen Client und Server.

[1] Genauer gesagt, Daten die die Rechnungsbeträge nicht beeinflussen, z. B. Adressen.

Fünf bis 10 Sachbearbeiter und ungefähr 20 Therapeuten benutzen das System, evtl. gleichzeitig. Der Server muß an 7 Tagen die Woche 24 Stunden lang verfügbar sein, obwohl die Mehrzahl der Zugriffe an Werktagen zu normalen Geschäftszeiten erfolgen wird.

Die RHL umfasst ungefähr 2100 Mitglieder. Sie nehmen pro Quartal ungefähr 100000 Behandlungen in Anspruch. Das System muss also pro Wochentag $\approx$ 1500 Transaktionen[2] durchführen:

$$\frac{100000\ \text{Behandlungen}}{12\ \text{Wochen} * 5\ \text{Tage}} = 1538 \frac{\text{Behandlungen}}{\text{Tag}}$$

2.5 externe Schnittstellen/Drucken

Die Use Cases 12–15 beschreiben 4 verschiedene Typen von Rechnungen, die erstellt werden. Die Rechnungen können am Bildschirm betrachtet oder ausgedruckt werden.

Die Rechnungsbeträge werden teilweise direkt per Lastschrift von Mitgliedern oder Krankenkassen eingezogen. Hierzu wird eine (elektronische) Schnittstelle zur Hausbank der RHL benötigt.

2.6 Datenschutz/Sicherheit/Authentifizierung

Das System speichert und verarbeitet unter anderem personenbezogene Daten. Es muss also den Bestimmungen des Bundesdatenschutzgesetzes (BDSG) entsprechen. Dies bedeutet vor allem, dass ein Zugriffsschutz vor unbefugten Personen auf die Daten vorhanden sein muss. Eine ausführliche Diskussion über Datenschutzbestimmungen im medizinischen Umfeld bietet [Bar04].

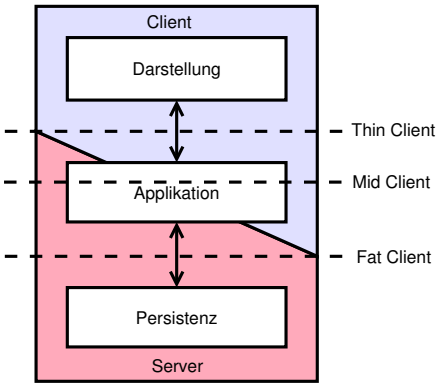
Der Datenbankserver muss vor einem (physikalischen) Zugriff geschützt werden, damit niemand die Datensätze an der *Konsole* manipulieren kann. Er sollte in einer gesicherten Umgebung betrieben werden, z. B. einem verschlossenem Raum oder einem entsprechend gesichertem Rechenzentrum. Der Zugang zum Server sollte ferner nur einem bestimmten Personenkreis gewährt werden.

Das Client und Server verbindende Netzwerk muß gesichert sein, um auszuschließen dass Dritte die übertragenen Daten abhören oder Daten während der Übertragung verändern, bzw. Datensätze in die Übertragung einfügen. Dies kann entweder durch die Verwendung von privaten Netzen sichergestellt werden, denen nur vertrauenswürdige, d. h. für das System berechtigte, Personen oder Computer angehören. Diese Lösung würde sich für den Betrieb innerhalb der Räume der RHL für die Sachbearbeiter anbieten. Werden öffentliche Netzwerke oder unsichere Kanäle zur Datenübertragung benutzt, so ist durch die Verwendung von geeigneten kryptographischen Protokollen die Vertraulichkeit der Datenübertragung sicherzustellen. Eine ausführliche Diskussion über Anforderungen an gesicherte Übertragungswege, insbesondere bei mobilen Geräten, findet sich in [PSZ04].

Vor der Benutzung des Systems muss sich jeder Benutzer am Client Computer anmelden/authentifizieren. Durch die Anmeldung wird er der entsprechenden Benutzergruppe zugeordnet. Ist die Anmeldung erfolgreich, so besteht eine gesicherte Verbindung zwischen dem Client und

[2] Mit einer Transaktion sind hier die Zugriffe auf die Datenbank gemeint, die eine Behandlung, bzw. deren Daten, in die Datenbank eintragen.

Abbildung 1: Schichten im Client Server Modell



dem Server solange, wie die Anmeldung gültig ist. Der Benutzer kann nun beliebig viele Aktionen ausführen. Nach einer gewissen Zeitspanne kann die Verbindung unterbrochen werden, wenn der Benutzer inaktiv war. Dadurch wird verhindert, dass ein Dritter über einen angemeldeten Benutzer Zugriff auf das System erhält. In der Regel sollte sich ein Benutzer jedoch vom System abmelden, wenn er den Client Computer unbeaufsichtigt lässt.

2.7 Zusammenfassung

Die Analyse des RHL Systems hat ergeben, dass verschiedene Subsysteme erforderlich sind, um den Funktionsumfang abzudecken. Die unterschiedlichen Benutzergruppen benutzen unterschiedliche Clientsysteme, die autonom mit einem zentralen Server kommunizieren. Durch den zentralen Server vereinfacht sich die Datenhaltung und Sicherung/Backup des Systems enorm. Aufgrund der Bearbeitung personenbezogener Daten muss die Kommunikation zwischen Clients und dem Server gegen den Zugriff Dritter geschützt werden.

3 Design

Aus der Analyse wurde ersichtlich, dass ein klassisches Client-Server Modell benutzt werden sollte, um das Abrechnungssystem zu erstellen. Mehrere Clients arbeiten unabhängig voneinander auf einem zentralen Server. Clientsysteme können dabei heterogen sein und auf unterschiedlichen Hardware- und Softwareplattformen betrieben werden. Die Applikationslogik sollte soweit wie möglich auf dem Server implementiert werden, damit so wenig Code wie möglich mehrfach für die unterschiedlichen Clients implementiert werden muss.

Ein Client-Server Modell lässt sich immer in die drei in Abbildung 1 gezeigten Schichten unterteilen. Die *Persistenz* Schicht speichert und verwaltet die Datensätze. Oft müssen die Datensätze gewisse Integritätsbedingungen erfüllen und gegen den gleichzeitigen Zugriff von mehreren Clients geschützt werden. Diese Funktionalität wird i. d. R. ebenfalls von der Persistenz Schicht angeboten. Der folgende Abschnitt “Datenbank” befasst sich ausführlich mit dieser

Schicht. Die *Applikations* Schicht verarbeitet die Datensätze und bildet damit die Logik der Applikation ab. Zur Darstellung des gegenwärtigen Zustandes benutzt sie die *Darstellungs* Schicht. Dieses Schichtenmodell hat den Vorteil, dass die Verarbeitung der Datensätze, welche ja die Vorgänge des Softwaresystems abbildet, von der persistenten Datenspeicherung und der Darstellung sauber getrennt wird. Änderungen innerhalb einer Schicht haben keine Auswirkungen auf die Funktion anderer Schichten.

Das grundsätzliche Problem ist nun, wie drei Schichten auf zwei beteiligte Computer (Client und Server) verteilt werden. Dabei sind zwei Zuordnungen fix; die Persistenz Schicht übernimmt der Server Computer und die Darstellung der Client. Die Applikations Schicht kann je nach Bedarf zwischen Client und Server aufgeteilt werden. Ist der überwiegende Teil dieser Schicht auf dem Client, so spricht man von einem *Fat Client*, da der Großteil der Datenverarbeitung auf dem Client geschieht. Beim *Thin Client* ist das Verhältnis genau umgekehrt, der Server übernimmt den Hauptteil der Verarbeitung, der Client stellt lediglich die Ergebnisse dar. Es ist auch möglich, dass Client und Server einen in etwa gleich großen Anteil der Datenverarbeitung übernehmen, in diesem Fall spreche ich von einem *Mid Client*. Alle Formen haben Vor- und Nachteile und je nach Anwendungsfall ist eine der Möglichkeiten vorzuziehen. In den Abschnitten 3.2.2 und 3.2.1 wird erläutert welche Aufteilung die Applikationen haben.

Dieses grundsätzliche Schichtenmodell ist auch unter dem Namen *Three-Tier-Model* bekannt. Typisch hierfür ist die strikte Trennung zwischen der Applikationslogik und der Datenspeicherung in zwei Schichten. Zusammen mit der dritten Darstellungsschicht ergibt sich der de-facto Standard für ein Client-Server System.

Charakteristisch für ein Client-Server System ist das Anfrage-Ergebnis[3] Verhalten. Der Server befindet sich solange im Wartezustand, bis ein Client eine Anfrage an ihn stellt. Diese Anfrage wird bearbeitet und das Ergebnis wird dem Client mit einer Antwort mitgeteilt. Danach wartet der Server auf weitere Anfragen eines Clients. Ohne eine Anfrage führt der Server keine Funktionen aus.[4]

Auch die Client Systeme befinden sich solange im Wartezustand, bis durch den Benutzer eine Aktion ausgelöst wird. Kann das Client System die Aktion nicht lokal ausführen (bei einem Thin-Client wäre dies immer der Fall), wird eine Anfrage an den Server gestellt. Der Client wartet auf die Antwort des Servers und kann in der Zwischenzeit keine weiteren Benutzeraktionen bedienen. Die Antwort des Server wird evtl. lokal bearbeitet und das Ergebnis dann dem Benutzer mitgeteilt. Danach wartet der Client auf eine erneute Aktion des Benutzers.

3.1 Datenbank

Das System muss über einen Zeitraum von mehreren Monaten Daten erfassen, sowie die erfassten Daten für mehrere Jahre für Verarbeitungen zur Verfügung stellen. Es ist in der Regel nicht möglich sicherzustellen, dass das Softwaresystem in diesen Zeiträumen ununterbrochen arbeiten kann. Die Datensätze müssen folglich auf persistenten Datenträgern gespeichert werden. Zusätzlich greifen mehrere Clients gleichzeitig auf die Datenbasis zu. Ein gleichzeitiger

[3] Request-Respone

[4] abgesehen von einem automatischen Backup, welches man aber auch als eine Anfrage an den Server betrachten kann, nur das diese von einem Zeitschalter gestartet wird.

Zugriff auf dieselben Datensätze muß jedoch verhindert werden, um Updateprobleme zu vermeiden. Dies kann durch serialisierten Zugriff, bzw. ein geeignetes Locking erreicht werden. Zugriffsschutz und Persistenz werden von einer Datenbanksoftware umgesetzt.

Es gibt unterschiedliche Ansätze, die in einer Datenbank verwalteten Datensätze zu strukturieren. Heute gebräuchlich sind die folgenden drei:[5]

hierarchische Datenbanken Alle Datensätze werden in einer Hierarchie (Baum) angeordnet. Die eigentlichen Datensätze befinden sich in den Blättern. Beziehungen zwischen den Datensätzen lassen sich nur über den Ort des Satzes im Baum herstellen.

relationale Datenbanken Alle Datensätze werden in (unterschiedlichen) Tabellen gespeichert. Die Tabellen sind voneinander unabhängig, jedoch können einzelne Zeilen zwischen je zwei Tabellen verknüpft werden.

objektorientierte Datenbanken Alle Datensätze werden ohne spezielle Struktur gespeichert. Es können beliebige Beziehungen zwischen je zwei Objekten erstellt werden.

Hierarchische Datenbanken können nur 1:N Beziehungen abbilden und sind somit nur für einfach strukturierte Datensätze bzw. deren Beziehungen untereinander einsetzbar. Relationale und objektorientierte Datenbanken können mit beliebig verknüpften Daten umgehen. Momentan sind die relationalen Datenbanken jedoch am ausgereiftesten was Zugriffsschutz, Zugriff mehrerer Benutzer, gleichzeitiger Zugriff unterschiedlicher Programme und Betriebssicherheit anbelangt. Das Design der Persistenzschicht wird deshalb auf eine relationale Datenbank ausgelegt.

Auf dem Markt gibt es heutzutage eine große Vielfalt an relationalen Datenbanken, die unterschiedliche Einsatzgebiete abdecken. Es macht wenig Sinn ein solches System selbst zu entwickeln, insbesondere da vorhandene Datenbanken zusätzliche Funktionen, wie Zugriffsschutz, Transaktionen oder Programmierbarkeit, anbieten, welche für das zu erstellende System ebenfalls benötigt werden. Durch die Verwendung einer bestehenden Datenbanksoftware kann also der Implementierungsaufwand drastisch verringert werden. Das nachfolgend beschriebene Design ist stark darauf ausgelegt, dass ein Großteil der Funktionalität von der Datenbank bereitgestellt wird.

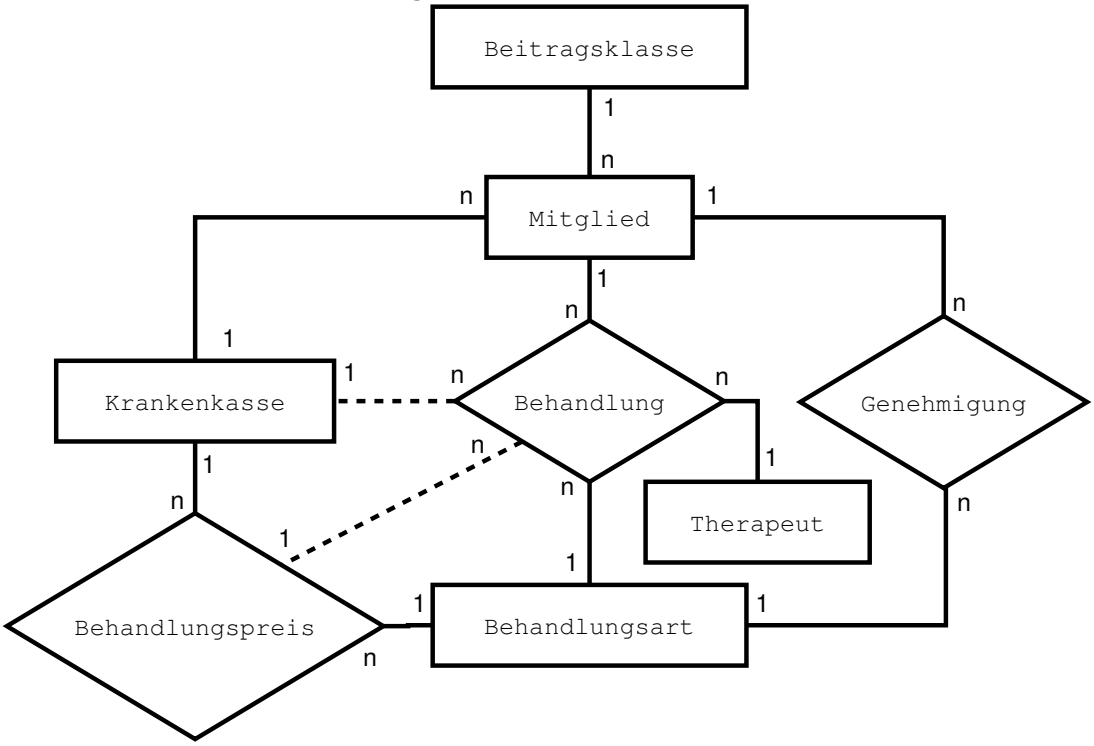
Den Zugriffsschutz auf die Daten, entsprechend den Rollen der Benutzer, deckt die Datenbank durch entsprechende Benutzerregeln ab. Da die Aufgaben der unterschiedlichen Rollen disjunkt sind, kann für jede Rolle ein Datenbankbenutzer angelegt werden, dem entsprechende Rechte auf die Datenbankobjekte (Tabellen, Funktionen, Queries) gewährt werden. Somit ist ausgeschlossen, dass bei fehlerhaften oder manipulierten Programmen Datensätze verändert werden können, für die der Benutzer (auf Clientseite) keine Rechte hat.

Da mehrere Benutzer die Datenbank gleichzeitig benutzen, muss verhindert werden, dass der selbe Datensatz von zwei oder mehr Benutzern simultan verändert wird. Dies kann durch den Einsatz von *Transaktionen*[6] und gleichzeitigem *Locking* von Datensätzen (Zeilen in den Tabellen) erreicht werden. Hat ein Benutzer durch das Starten einer Transaktion und damit einhergehendem Locking der verwendeten Zeilen in den Tabellen begonnen, die Datensätze zu

[5] Eine ausführliche Betrachtung der Vor- und Nachteile der unterschiedlichen Datenbanktypen enthält [Rol03].

[6] Hier ist mit einer Transaktion eine Abfolge von Befehlen an die Datenbank gemeint.

Abbildung 2: ERM Modell der Datenbank



verändern, so ist es allen weiteren Benutzern nur möglich, den ursprünglichen Zustand der Datensätze vor Beginn der Transaktion zu lesen. Schreibende Zugriffe auf die Datensätze, bzw. der Start einer weiteren Transaktion werden von der Datenbank solange verhindert, bis die erste Transaktion beendet wurde. Auf diese Weise, stellt die Datenbank sicher, das keine Inkonsistenzen entstehen können.[7]

Es gibt eine Reihe von Abfragen an die Datenbank, die nicht durch einen einzelnen relationalen *Befehl* ausgedrückt werden können, sondern mehrere Befehle benötigen. Teilweise müssen Befehle auch bedingt ausgeführt werden. Mit Hilfe von *Stored Procedures* werden diese Abfragen als neue Befehle oder Funktionen permanent in der Datenbank gespeichert. Da die Stored Procedures mittlerweile vollwertige Programmiersprachen geworden sind, kann die gesamte Konsistenzprüfung von Eingabedaten von datenbankinternen Funktionen realisiert werden.

3.1.1 Entity-Relationship-Modell

Als Beschreibung einer relationalen Datenbank bietet sich ein Entity-Relationship-Modell (ERM) nach Codd [Cod70] am besten an. Das in Abbildung 2 gezeigte Schema enthält alle Tabellen, um die für die Use Cases benötigten Datensätze in einer relationalen Datenbank abzubilden.[8]

[7] Natürlich nur unter der Voraussetzung, dass die vom Benutzer eingegebenen Daten korrekt sind.

[8] In diesem Diagramm sind die Kardinalitäten für Beziehungen zwischen den Tabellen wie folgt zu interpretieren: Ein Mitglied ist genau einer Beitragsklasse zugeordnet (die 1 steht bei der Entity Beitragsklasse), während zu

Die den Entities und Relationen entsprechenden Tabellen werden nun kurz beschrieben.

Krankenkasse Alle deutschen Krankenkassen sind in dieser Tabelle zu finden. Es werden Attribute wie Adresse(n) oder Bankverbindung verwendet.

Behandlungsart Diese Tabelle enthält die unterschiedlichen, vom System angebotenen, Behandlungsarten. Dies ist eine einfache Tabelle, die lediglich ein Paar aus Name und Nummer enthält.

Behandlungspreis Der Betrag, den die Krankenkassen für eine Behandlungsart erstatten, ist in dieser Tabelle abgelegt. Krankenkasse und Behandlungsart werden über Fremdschlüssel referenziert.

Beitragsklasse Die unterschiedlichen Beitragsklassen für Mitglieder.

Mitglied Sämtliche Mitglieder werden von dieser Tabelle verwaltet. Neben persönlichen Daten wie Adresse, Geburtsdatum, Geschlecht, Bankverbindung usw. werden auch das Eintrittsdatum in die RHL, sowie evtl. ein Austrittsdatum erfasst. Bei einem Austritt darf das Mitglied nicht aus der Tabelle gelöscht werden, da ansonsten auch alle erfassten Behandlungen gelöscht würden und somit keine korrekte Abrechnung mehr erstellt werden könnte. Jedes Mitglied ist einer Beitragsklasse und einer Krankenkasse zugeordnet.

Genehmigung Die Kostenübernahme für Therapien muß von den Krankenkassen für jedes Mitglied genehmigt werden. Die Genehmigungen werden mit der Tabelle Genehmigung erfasst.

Therapeut Therapeuten werden in dieser Tabelle erfasst.

Behandlung Diese Tabelle stellt die Kerntabelle des Systems dar. Jede Zeile erfasst eine durchgeführte Behandlung. Dazu müssen das Mitglied, die Behandlungsart und der Therapeut erfasst werden. Da ein Mitglied jedoch seine Krankenkasse zu einem beliebigen Zeitpunkt wechseln kann, kann die eine Behandlung erstattende Krankenkasse nicht über die Zuordnung beim Mitglied ermittelt werden, denn in der Mitglied Tabelle wird die jeweils aktuelle Krankenkasse erfasst. Die Tabelle Behandlung braucht also einen direkte Referenz auf die Krankenkasse, bei der das Mitglied zum Zeitpunkt der Behandlung versichert war. Die gleiche Überlegung gilt für den Behandlungspreis, der sich ebenfalls zu einem beliebigen Zeitpunkt ändern kann. Der Preis muss ebenfalls direkt von der Behandlung referenziert werden.

Bei der Entwicklung eines Datenbankmodells sollte man redundante Datensätze vermeiden. Eine einzelne Information sollte also nicht an zwei unterschiedlichen Stellen abgelegt werden, ansonsten entstehen leicht *Updateprobleme*. Es ist dann nicht mehr trivial zu entscheiden, welche der beiden Informationen die relevante ist. Durch Programm- oder Bedienfehler kann es vorkommen, dass beide Informationen nicht den identischen Inhalt haben, so dass die Datenbank

einer Beitragsklasse beliebig viele Mitglieder gehören können (n bei Mitglied).

möglicherweise inkonsistent ist. Das Datenbankmodell sollte also sicherstellen, dass Informationen nur einmal bzw. an einem Ort gespeichert werden.

Ein ERM lässt sich durch *Normalisation*[9] in eine redundanzfreie Form überführen. Dabei wird untersucht, ob ein ERM in der ersten, zweiten und dritten Normalform ist und evtl. verändert werden muß. Ist das ERM in der dritten Normalform, dann ist sichergestellt, dass keine redundanten Informationen in dem Modell gespeichert werden. Um die dritte Normalform zu erhalten, müssen jedoch oft viele zusätzliche Tabellen erzeugt werden und dass gewonnene ERM hat oftmals wenig Ähnlichkeiten mit dem ursprünglichen. Ferner sind Anfragen an eine normalisierte Datenbank oftmals ineffizient, da auch eine prinzipiell einfache Anfrage[10] mehrere Tabellen einbeziehen muss. Es wird daher oftmals bewusst darauf verzichtet die Datenbank vollständig redundanzfrei zu erstellen, damit der Zugriff auf die Datenbank einfacher/performanter wird.

Das vorgestellte Datenbankmodell ist nicht in der dritten Normalform. Dadurch wird die Erfassung von Abrechnungen stark vereinfacht. Zur Erfassung einer Behandlung muss nur eine neue Zeile in die Tabelle Behandlung geschrieben werden. Die in Abbildung 2 gestrichelten Linien zwischen *Behandlung* und *Behandlungspreis*, sowie *Behandlung* und *Krankenkasse*, deuten an, dass diese Beziehungen redundant sind. Sie könnten ebenfalls über die Beziehungen zwischen *Mitglied* und *Behandlungsart* hergestellt werden. Erfolgt der schreibende Zugriff auf *Behandlung* nicht direkt, sondern über eine Stored Procedure, dann kann die Datenbanksoftware sicherstellen, dass keine inkonsistenten Daten in die Tabelle *Behandlung* eingetragen werden. Die Stored Procedure stellt die Integrität der Datenbank sicher. Die (bewusste) Verletzung der dritten Normalform hat in diesem Fall keine negativen Auswirkungen. Die restlichen Entities und Relationen im ERM entsprechen den Vorgaben der dritten Normalform.

In der Datenbank werden alle Datensätze des Abrechnungssystems gespeichert. Entsprechend den Datenschutzbestimmungen ist sicherzustellen, dass nur berechtigte Personen / Benutzer / Rollen Zugriff auf die Datensätze haben. Insbesondere die Mitgliedsdaten und die durchgeführten Behandlungen sind vor unbefugtem Zugriff zu schützen. Dies wird durch Zugriffsfunktionen der Datenbank sichergestellt. Die zwei unterschiedlichen Benutzergruppen des Systems (Therapeuten und Sachbearbeiter) arbeiten mit unterschiedlichen Client Programmen (siehe 3.2). Diese Programme benutzen wiederum unterschiedliche Datenbankuser, welchen der lesende und schreibende Zugriff auf jede Tabelle bzw. Tabellenspalte je nach Bedarf gewährt wird. Die Datenbank kann also sicherstellen, dass ein Benutzer nur die für seine Aufgaben notwendigen Datensätze lesen und schreiben kann.

3.1.2 Umsetzung der Use Cases

Die in der Analyse (2.1) beschriebenen Use Cases lassen sich in der Regel in einfache Operationen auf das Datenbankmodell umsetzen. Im folgenden wird vor allem betrachtet, auf welche unterschiedlichen Tabellen für jeden einzelnen Vorgang zugegriffen wird.

1. Ein neues Mitglied wird durch Einfügen einer neuen Zeile in die Tabelle Mitglied angelegt.

[9] Beschrieben in dem Artikel von Codd oder [DBn].
[10] Wenn man von dem ursprünglichen ERM ausgeht

2. Beim Austritt eines Mitgliedes wird das Attribut *austrittsdatum* gesetzt. Die Tabellenzeile darf nicht gelöscht werden.

3. Bei Änderungen von Mitgliedsdaten werden die entsprechenden Attribute in einer Zeile der Tabelle Mitglied geändert.

4. Um ein Mitglied zu suchen, muss nur auf die Tabelle Mitglied zugegriffen werden.

5. Die Krankenkassen werden von einer einzigen Tabelle verwaltet. Soll eine neue Krankenkasse angelegt werden, so muss eine neue Zeile zur Tabelle Krankenkasse hinzugefügt werden. Zur Relation Behandlungspreis muß danach für jede Behandlungsart der Preis für diese Krankenkasse hinzugefügt werden. Änderungen wirken sich auf eine einzige Zeile aus. Sollte eine Krankenkasse gelöscht werden, so müssen vorher alle evtl. noch bei dieser Krankenkasse registrierten Mitglieder auf eine andere Krankenkasse umgetragen werden, danach wird in einem Attribut der Tabelle Krankenkasse diese als gelöscht markiert. Die entsprechende Zeile darf nicht aus der Tabelle gelöscht werden.

6. Ändert eine Krankenkasse ihre Sätze, so sind die Einträge in der Tabelle Behandlungspreis entsprechend zu ändern.

7. Die Übersicht, welche Mitglieder bei einer bestimmten Krankenkasse versichert sind, ist durch eine Suche über die Tabelle Mitglied zu erstellen.

8. Eine neue Genehmigung benötigt einen neuen Eintrag in die Tabelle Genehmigung.

9. Die Übersicht, welche Genehmigungen für ein Mitglied vorliegen, ist durch eine Suche in der Tabelle Genehmigung zu erstellen.

10. Eine Behandlung wird erfasst, indem eine neue Zeile in die Tabelle Behandlung eingefügt wird. Vorher muss jedoch geprüft werden, ob eine Genehmigung für das Mitglied am Tage der Behandlung und die Behandlungsart vorliegt. Für die Überprüfung ist ein zusätzlicher Zugriff auf die Tabelle Genehmigung nötig.

11. Die Übersicht, welche Behandlungen ein Mitglied in Anspruch genommen hat, ist durch eine Suche in der Tabelle Behandlung zu erstellen.

12. Die Abrechnung von Behandlungen für Krankenkassen ist durch eine Suche in der Tabelle Behandlung zu erstellen.

13. Die Abrechnung von Behandlungen für privat versicherte Mitglieder ist durch eine Suche in der Tabelle Behandlung zu erstellen.

14. Die Abrechnung der Zuzahlungen der Mitglieder ist durch eine Suche in der Tabelle Behandlung zu erstellen.

15. Die Abrechnung des Jahresbeitrages für Mitglieder ist durch eine Suche in der Tabelle Mitglied zu erstellen.

Bis auf die Fälle 5 und 10 sind alle Use Cases durch den Zugriff auf eine einzige Tabelle implementierbar. Bei 5 und 10 werden die benötigten Queries durch Stored Procedures in der Datenbank abgelegt, so dass die Applikationsschicht mit einer einzelnen Abfrage das Ergebnis erhält.

Die Datenbank stellt über ihre Integritätsfunktionalität sicher, dass kein ungültiger Zustand erreicht werden kann. Insbesondere werden die Attribute der Tabellen durch *check clauses* bei jeder Veränderung auf Konsistenz geprüft. Bei der Verwendung von Fremdschlüsseln werden keine *null values* zugelassen, so dass jede Referenz auf Tabellen eine gültige Referenz darstellt. Die Applikationsschicht ist somit nicht für die Konsistenz der Daten verantwortlich.

Natürlich kann die Datenbank die Korrektheit von Attributen wie der Anschrift oder der Bankverbindung nicht überprüfen bzw. sicherstellen. Jedoch haben falsche Werte in diesen Attributen keinen Einfluss auf die Funktion des Systems. Es würde lediglich eine Rechnung mit falscher Adresse erstellt werden. Es ist jedoch nicht möglich, durch Eingaben einen ungültigen Zustand des Systems zu produzieren oder eine falsche Abrechnung zu erzeugen.

3.2 Client Frontends

Sachbearbeiter und Therapeuten bilden die beiden Benutzergruppen. Im vorigen Kapitel wurde gezeigt, dass beide Gruppen unterschiedliche Use Cases benötigen. Es bietet sich an, zwei getrennte Applikation zu erstellen, für jede Benutzergruppe je eine Applikation. Neben der Funktionalität unterscheiden sich die beiden Applikationen in der Unterstützung von Eingabegeräten und Plattformen, auf welchen sie betrieben werden. Diese Applikationen werden ausschließlich auf Client Computern betrieben.

Benutzer der Clients müssen sich beim Applikationsstart gegenüber dem System authentifizieren. Es gibt drei grundsätzliche Verfahren, seine Identität/Rolle einem Computersystem mitzuteilen. Der Computer kann eine Eigenschaft des Körpers untersuchen, die einzigartig ist. Hierzu zählen vor allem die biometrischen Erkennungsmerkmale wie Fingerabdruck, Gesichts- oder Stimmerkennung. Eine Authentifizierung kann auch über einen Gegenstand hergestellt werden, den die Person besitzt, z. B. einen Schlüssel oder eine Smartcard. Als dritte Möglichkeit kann der Computer etwas abfragen, welches die Person kennt. Die weit verbreiteten Benutzername/Passwort Abfragen sind hierfür ein Beispiel. Die drei Verfahren haben je nach Anwendungszweck unterschiedliche Vor- und Nachteile. Eine ausführliche Diskussion über Authentifizierungsverfahren bietet [Bis02].

Die Clients des RHL Systems benutzen zwei unterschiedliche Verfahren zur Authentifizierung der unterschiedlichen beteiligten Personen. Die Benutzer beider Clients, also die Personen, welche die Software bedienen, melden sich über einen Benutzernamen und Passwort am System an. Die Patienten, die eine Therapie aufsuchen, werden über ihre Krankenversichertenkarte (KVK) identifiziert. Für die Sicherheitsbelange des Systems sind diese beiden Methoden ausreichend.

Beide Applikationen benutzen das gleiche Protokoll zur Kommunikation mit dem Server (siehe nächster Abschnitt). Die bearbeiteten Daten werden ausschließlich auf dem Server gespeichert. Es kann folglich durch einen Ausfall eines Clients (durch Hardware- oder Softwarefehler) folglich zu keinem Datenverlust kommen. Für die Client Computer muss somit keine Backup Strategie entwickelt werden (für Daten). Sollte ein Client Computer nicht funktionsfähig sein, so kann dieses Gerät einfach ausgetauscht werden, da auch die Benutzerauthentifizierung nicht

an das Gerät gebunden ist. Da keine Daten auf den Clients gespeichert werden, sind für die Programme auch keine besonderen Datenschutzbestimmungen zu beachten. Der gesamte Datenschutz muß serverseitig realisiert werden (siehe voriger Abschnitt).

Zur Kommunikation mit dem Server müssen die Clientcomputer ein Netzwerk benutzen. Aus Kostengründen ist der Aufbau eines privaten Netzwerks nicht machbar, so dass die Kommunikation über ein öffentliches Netzwerk (das Internet) erfolgt. Da bei einem öffentlichen Netzwerk nicht ausgeschlossen werden kann, dass dritte in die Kommunikation zwischen Client und Server eingreifen, muss die Verbindlichkeit und Vertraulichkeit der Kommunikation durch kryptografische Protokolle sichergestellt werden.

3.2.1 Webapplikation

Der Sachbearbeiter besitzt an seinem Arbeitsplatz bereits einen Computer, der zum Arbeiten mit dem System benutzt werden soll. Auf diesem Computer wird Windows oder Linux eingesetzt.

Windows und Linux sind miteinander nicht kompatibel, d. h. ein Programm kann nicht unter beiden Betriebssystemen benutzt werden. Da sich die Programmschnittstellen zum Betriebssystem (API) bei Windows und Linux erheblich unterscheiden, bzw. komplett unterschiedlich sind, ist es sehr aufwändig eine Applikation zu entwickeln, die mit einer gemeinsamen Code Basis auf beiden Betriebssystemen lauffähig ist. Ein Webbrowser ist jedoch standardmäßig bei Windows und Linux installiert. Da die Sprache HTML plattformunabhängig ist, sowie die Eingabe von Daten erlaubt, kann man hiermit eine sog. *Webapplikation* erstellen. Diese wird mittels des Webbrowsers dargestellt und erlaubt die Eingabe von Daten mit den üblichen Widgets wie Buttons, Textfelder, Listen usw. Da keine Software auf dem Client Computer installiert werden muss, kann jeder Arbeitsplatz mit Netzwerkanschluss sofort als Sachbearbeiter-Arbeitsplatz genutzt werden.

Da jeder Internet Nutzer den Webserver, der die Webapplikation betreibt, ansprechen kann, muß der Webserver eine Benutzer-Authentifikation unterstützen. Das HTTP[11] Protokoll, welches zur Übertragung von Webseiten im Internet benutzt wird, unterstützt eine Benutzerauthentifizierung [FHBH+99]. Dabei wird für jeden Seitenaufruf vom Browser ein Benutzername–Passwort Paar übertragen, welches vom Webserver überprüft wird.

Der Datenschutz verlangt unter anderem eine gesicherte Übertragung der Daten zwischen Client und Server Systemen. Das HTTP Protokoll stellt zwar sicher, dass die Daten ohne Übertragungsfehler transportiert werden, jedoch ist es möglich, die übertragenen Daten auf ihrem Weg von Dritten aufzuzeichnen oder zu manipulieren. Dies kann durch eine kryptographische Verschlüsselung der Daten unterbunden werden. Nur Client und Server besitzen die (geheimen) Schlüssel und sind in der Lage die Webseiten/Daten zu entschlüsseln. Asymmetrische Verfahren mit öffentlichen Schlüsseln erlauben eine verschlüsselte Kommunikation, ohne dass beide Partner über einen gemeinsamen, geheimen Schlüssel verfügen müssen. Es können damit zwei sich vorher nicht unbekannte Parteien gesichert kommunizieren. Das SSL/TLS[12] Protokoll baut eine verschlüsselte Verbindung zwischen zwei Computern auf, über welche dann andere Protokolle zum Austausch von Daten genutzt werden können. Die Kombination von HTTP und TLS nennt man HTTPS. Zu TLS und HTTPS siehe [DA99] und [Res00].

[11] HyperText Transfer Protocol, siehe [FGM+99]

[12] früher Secure Socket Layer, heute Transport Layer Security

Webbrowser waren ursprünglich zur Darstellung von statischen Inhalten vorgesehen. Die dargestellten Bilder und Texte haben sich nur selten geändert und wurden in der Regel in ganz normalen Dateien auf dem Server gespeichert. Diese Dateien wurden dann unverändert und komplett zum Webbrowser übertragen, der sie entsprechend darstellte. Für diesen Zweck war es ausreichend, ein einfaches, *zustandloses* Übertragungsprotokoll zu entwickeln, so dass keine Beziehung zwischen zwei aufeinander folgenden Aufrufen von Webseiten hergestellt werden muss. Nachdem die Webseite komplett übertragen wurde, wird die Verbindung zwischen Browser und Server abgebaut. Bei einem neuen Seitenaufruf wird dann eine neue Verbindung aufgebaut.

Für Webapplikationen ist dieses Verhalten jedoch nicht ausreichend. Webseiten von Webapplikationen haben in der Regel einen dynamischen Inhalt, der sich bei jeden Aufruf ändern kann. Der Server kann die gewünschten Webseiten folglich nicht einfach aus einer Datei auslesen, sondern muß diese für jeden Aufruf neu aus anderen Quellen generieren. Es gibt hierzu eine Vielzahl von Möglichkeiten, die von einer einfachen Ersetzung von bestimmten Wörtern in der Webseite über den Aufruf von externen Programmen auf Serverseite bis zur Integration von Skript- und Hochsprachen in den Webserver bzw. die Webseiten reichen.

Der Workflow, den die Webapplikation abbildet, erfordert in der Regel das Aufrufen von mehreren Webseiten. Da durch das HTTPS Protokoll kein Zusammenhang zwischen den Aufrufen hergestellt werden kann, muß durch *Session Management* dieser Zusammenhang hergestellt werden. Dazu muß i. d. R. neben den eigentlichen Inhalten und Informationen, die per HTTPS übertragen werden sollen, eine weitere Information übertragen werden, mit welcher der Webserver bzw. die Applikationslogik auf dem Server den Zusammenhang zwischen zwei HTTPS Übertragungen herstellen kann. Es gibt verschiedene Techniken wie URL-Rewriting, Hidden-Form-Data, Cookies oder IP-Tracking. Die verschiedenen Verfahren werden in [Bag03] und [Oll04] vorgestellt.

Die Webapplikation des RHL Systems benutzt Hidden-Form-Data als Technik zum Session Management. Dies bietet sich an, da auf fast allen Webseiten ein Formular zur Dateneingabe vorhanden ist. Auf Webseiten ohne Formulare, wo der Workflow durch die Benutzung eines Links abgebildet wird, wird durch das Hinzufügen eines URL-Parameters[13] die Form-Data übertragen.[14]

Der Webbrowser auf dem Clientcomputer dient lediglich zur Darstellung und Eingabe. Alle Datenverarbeitungen werden auf dem Server vorgenommen. Entsprechend der Beschreibung der unterschliedlichen Clientarten auf Seite 10 ist die Webapplikation ein Thin-Client.

3.2.2 KVKapplikation

Webbrowser unterstützen Tastatur und Maus zur Dateneingabe durch den Benutzer. Für einige Use Cases werden jedoch die Daten der Krankenversichertenkarte benötigt. Webbrowser haben standardmäßig keine Unterstützung für die dazu benötigten Kartenleser.[15] Die Kartenleser

[13] Dies ist nicht mit dem URL-Rewriting zu verwechseln

[14] Beide Möglichkeiten, Formulare oder URL-Parameter, sind von Server nicht zu unterscheiden, bzw. werden durch die gleiche API abgebildet.

[15] Es wäre zwar möglich ein Plugin für den Webbrowser zu entwickeln, dieses müsste jedoch wiederum auf allen Arbeitsplätzen installiert werden und der Vorteil, dass die Webapplikation ohne eine Installation benutzt werden kann wäre nicht mehr gegeben.

können nur über normale Programme angesprochen werden, die auf dem Client Computer ausgeführt werden. Solch ein Programm wird im folgenden *KVKapplikation* genannt.

Die KVKapplikation stellt Dienste für zwei unterschiedliche Personengruppen zur Verfügung. Der Therapeut bedient die KVKapplikation und kann mit ihr das (Server)System abfragen und Behandlungen eintragen. Um Behandlungen einzutragen, müssen die Daten des Patienten erfasst werden; sie werden von der KVK gelesen. Beide Personengruppen müssen natürlich vom Server authentifiziert werden.

Auch für die KVKapplikation bietet es sich an, HTTPS zur Kommunikation zwischen Client und Server zu verwenden. Aufgrund des Datenschutzes ist eine gesicherte Übertragung vorgeschrieben und das HTTPS Protokolls kann zur Authentifikation des Therapeuten benutzt werden. Dieser muß sich dann vor Benutzung der KVKapplikation mit einem Benutzernamen und Passwort anmelden.

Die Patienten werden über die Daten auf der KVK identifiziert und authentifiziert. Das System kann zwar nicht feststellen, ob die Daten einer eingelesenen KVK dem Patienten entsprechen, allerdings gibt es so gut wie keinen Betrug durch falsche oder gestohlene KVK in Deutschland, da jede in Deutschland lebende Person krankenversichert ist und eine Karte besitzt. Durch Diebstahl oder Fälschung kann sich ein potentieller Betrüger keinen Vorteil verschaffen. Eine KVK ist somit ausreichend zur Authentifizierung gegenüber dem Server.

Im Gegensatz zur Webapplikation benötigt die KVKapplikation kein Session Management, da die Use Cases durch einzelne, unabhängige Requests darstellbar sind.

Die Betriebssysteme[16] der unterschiedlichen Clientsysteme stellen alle eine grafische Oberfläche (GUI[17]) und Netzwerkfunktionen bereit. Allerdings sind diese nicht einheitlich anzusprechen, d. h. die Funktionsaufrufe (API) sind bei den unterschiedlichen Betriebssystemen unterschiedlich. Die Applikationslogik ist jedoch für alle Plattformen gleich. Die Implementation muß also die unterschiedlichen APIs zusammenführen, damit sie von der Applikationslogik benutzt werden können.

HTTPS wird normalerweise mittels der Netzwerkprotokolle TCP/IP übertragen. IP setzt dabei auf einem weiteren Protokoll auf, welches die (physikalische) Art der Datenübertragung abstrahiert. Ein Desktop Computer ist normalerweise per Kabel (Ethernet) an ein LAN angeschlossen oder macht eine Einwahl in das Internet per Modem, ISDN oder xDSL. Die mobilen Computer können mit unterschiedlichen Funknetzwerktechnologien betrieben werden (Wireless LAN, Bluetooth oder die mobilen Netze GSM, GPRS, UMTS)[18]. Da jedoch alle Netzwerktechnologien vom Betriebssystem des Computers unterstützt werden und über alle mit IP auf das Internet zugegriffen wird, braucht die KVKapplikation hiervon keine Kenntnisse besitzen. Es muß lediglich der Netzwerkzugang im Betriebssystem entsprechend konfiguriert werden, damit die KVKapplikation auf das Internet zugreifen kann.

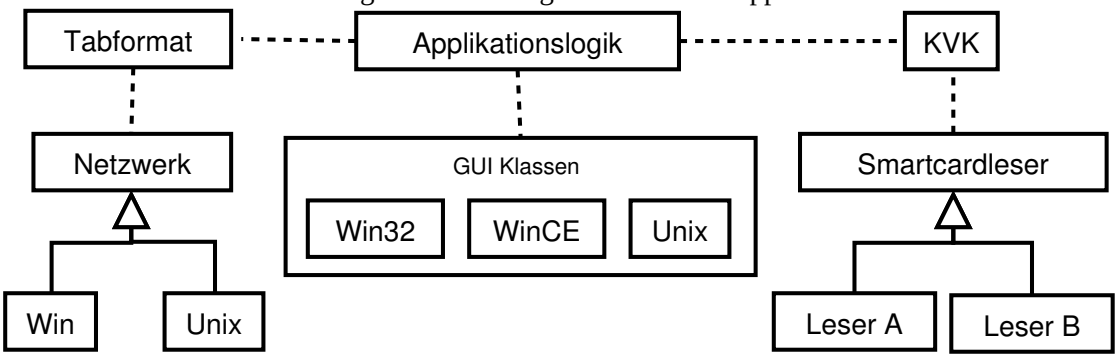
Abbildung 3 zeigt das (stark vereinfachte) Klassendiagramm der KVKapplikation. Windows und Unix verwenden vollkommen unterschiedliche Schnittstellen für Grafik und Netzwerk, so dass die Spezialisierungen keine gemeinsam verwendeten Code Teile enthalten. Da der Bildschirm auf einem Windows CE Gerät eine andere Ausrichtung hat und bedeutend kleiner ist als

[16] Die Begriffe Betriebssystem und Plattform werden in diesem Abschnitt synonym benutzt.

[17] Graphical User Interface

[18] [PSZ04] vergleicht die unterschiedlichen Netzwerktechnologien, die von mobilen Computern benutzt werden können.

Abbildung 3: Klassendiagramm der KVKapplikation



auf Desktop PCs benötigt auch die CE Applikation ihre eigene GUI Spezialisierung.

Die Eingabe von Tastatur und Maus wird von allen grafischen Oberflächen automatisch verarbeitet. Die Eingabe über einen Kartenleser muß die KVKapplikation jedoch selber steuern. Die Klasse *KVK* repräsentiert eine KVK-Smartcard und steuert bei Bedarf den Kartenleser an. Da es unterschiedliche Kartenleser gibt, die jeweils auf unterschiedliche Arten an den Computer angeschlossen sind (RS232, USB, PCMCIA), besitzt jeder Leser eine Spezialisierung der abstrakten Klasse *Kartenleser*.

Neben der Darstellung werden die Daten von der KVKapplikation vorverarbeitet bevor sie dem Server übermittelt werden. Ein Teil der Applikationslogik wird von der KVKapplikation übernommen. Auf dem Server werden die Daten dann weiter verarbeitet, bevor sie der Persistenz Schicht übergeben werden. Die Applikationsschicht ist bei der KVKapplikation auf Client und Server verteilt. Die KVKapplikation entspricht folglich einem Mid-Client.

3.3 Applikationsschicht auf dem Server

Die Kommunikation zwischen Clients und Server erfolgt mittels HTTPS. Dies ist ein sehr einfaches Protokoll, welches unstrukturierte Byteströme verschlüsselt zwischen zwei Computern überträgt. (Relationale) Datenbanken sind zwar in der Regel netzwerkfähig, d. h. sie stellen ihre Funktionalität anderen, an das Netzwerk angeschlossenen, Computern zur Verfügung, jedoch benutzen sie zur Kommunikation in der Regel ein proprietäres (Netzwerk) Protokoll. Dieses bietet oftmals auch keine Verschlüsselung. Da bei einem Wechsel der Datenbanksoftware nicht die Software auf allen Clients geändert werden soll, wird ein datenbank(hersteller)unabhängiges Format bzw. Protokoll zur Kommunikation zwischen Server (Datenbank) und Clients benötigt. Dieses muß sich mittels HTTPS transportieren lassen.

Beim Datenaustausch zwischen zwei Programmen, die evtl. auf unterschiedlichen Computern laufen, gibt es unterschiedliche Arten von Datenformaten: binär- und text basierte Formate. Binärformate enthalten die Daten als direkte Abbilder der Register- bzw. Hauptspeicherdarstellung der Bytes. Sie lassen sich in der Regel sehr einfach lesen und schreiben, sind jedoch nur wenig für eine plattformunabhängige Kommunikation geeignet, da unterschiedliche Speicherkonfigurationen wie Big- oder Little-Endian oder Registerbreiten (32 oder 64 bit) ein direktes Verarbeiten der Datensätze verhindern. Text basierte Formate benutzen eine hardwareunabhän-

Abbildung 4: Vergleich von XML und Tab-Format

```
Vorname  Name      Alter   Beruf
Dirk     Jagdman   27      Student
Michael  Stickel   32      Coder
Holger   Jagdman   55      Manager
```

≙ 100 Bytes

```
<tabelle>
<zeile>
<vorname>Dirk</vorname>
<name>Jagdman</name>
<alter>27</alter>
<beruf>Student</beruf>
</zeile>
<zeile>
<vorname>Michael</vorname>
<name>Stickel</name>
<alter>32</alter>
<beruf>Coder</beruf>
</zeile>
<zeile>
<vorname>Holger</vorname>
<name>Jagdman</name>
<alter>55</alter>
<beruf>Manager</beruf>
</zeile>
</tabelle>
```

≙ 333 Bytes

gige Darstellung als ASCII oder Unicode Text. Diese werden von allen Plattformen gleich interpretiert, jedoch müssen die Datensätze von einer Parserfunktion gelesen werden. Die Ein- und Ausgabe ist bei Textformaten aufwändiger als bei einem Binärformat, aber es eignet sich besser zum Datenaustausch zwischen heterogenen Systemen.

Der momentane Quasistandard bei textbasierten Datenformaten ist die Extended Markup Language (XML). Es gibt für alle wichtigen Programmiersprachen und Betriebssysteme Bibliotheken, die Ein- und Ausgabe im XML Format bereitstellen. Datensätze werden in XML als Baum dargestellt. In diesem Baum kann jeder Knoten unterschiedliche Kindknoten und unterschiedliche Knotenattribute besitzen. Möchte man den Inhalt einer Tabelle in XML beschreiben, so werden sehr viele XML Tags benötigt, um die einzelnen Spalten jeder Zeile zu beschreiben/begrenzen. Bei einem einfachen Datenformat, dass einfach durch das Tabulatorzeichen getrennte Datensätze je Zeile benutzt, wären die gleichen Daten mit deutlich weniger Bytes zu beschreiben (Tabformat). Im Beispiel von Abbildung 4 verbraucht die Darstellung der Daten als XML dreimal mehr Bytes als mit dem einfachen Tabformat. Ein Parser zur Eingabe von tabformatierten Daten ist außerdem einfacher als ein XML Parser zu erstellen.

XML ist ein allgemeineres Datenformat als das Tabformat, so wie ein Baum ein allgemeinerer Datentyp ist als eine Tabelle. Mit dem Tabformat lassen sich nur Tabellen darstellen. XML kann nur Bäume darstellen. Zwar kann jede Tabelle durch einen Baum dargestellt werden, jedoch nicht jeder Baum durch eine Tabelle.

Die Datensätze werden von einer relationalen Datenbank verwaltet, also einer Datenbank, welche mit Tabellen arbeitet. Die Ergebnisse von Anfragen an die Datenbank werden in Tabellenform geliefert. Das Tabformat bietet sich deshalb an, die Ergebnisse von Anfragen an den

Abbildung 5: Beispielabfrage der KVKapplikation

Anfrage: `https://server.com/suchen.pl?vorname=dirk&jahr=1976`

Query:
```
select vorname,name,geburt,beruf from mitglied where vorname =
'dirk' and datepart('year', geburt) = 1976;
```

Ergebnis:

Vorname	Name	Geburt	Beruf
Dirk	Jagdman	4.12.76	Student
Dirk	Fake	1.1.76	Schummler

Server zum Client zu übertragen. Sollte bei einer Erweiterung des Systems die Darstellung von Ergebnissen in Tabellenform nicht mehr möglich oder zweckmäßig sein, dann kann die Kommunikation auf XML umgestellt werden. Für die gegenwärtigen Anforderungen ist das Tabformat jedoch vollkommen ausreichend.

Anfragen an den Server müssen neben dem Typ der Anfrage lediglich einfache Key-Value Paare übertragen. Dazu kann der Request Teil des HTTPS Protokolls benutzt werden. Der Typ der Anfrage entspricht dann der Ressource (Webseite) auf dem Server, die zusätzlichen Werte werden als HTTPS Parameter an die URL der Anfrage angehängt. Abbildung 5 zeigt eine Beispielabfrage. Es soll nach Personen gesucht werden, die *Dirk* mit Vornamen heißen und *1976* geboren wurden. Es wird dazu auf dem Server *server.com* die Abfrage gestartet, die mit der Ressource (Webseite) */suchen.pl* verknüpft ist. Der Ressource werden die Parameter *vorname* und *jahr* übergeben. Das ganze wird mit der URL Syntax des HTTPS Protokolls ausgedrückt. Die Applikationsschicht wandelt diese URL-Anfrage in eine SQL Anfrage um und übergibt sie der Datenbank. Das Ergebnis der Datenbank wird dann in das Tabformat umgesetzt und an die KVKapplikation geschickt.

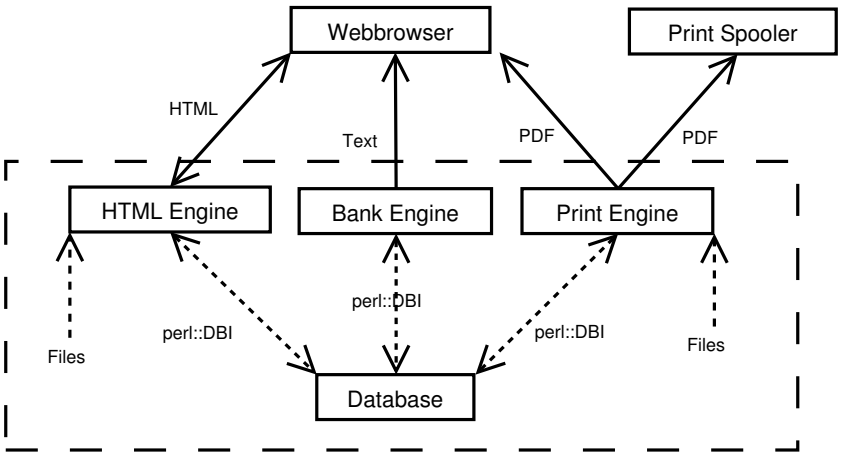
Die Applikationsschicht auf dem Server, welche die KVKapplikation bedient, ist schlank. Sie muß lediglich zwischen unterschiedlichen textbasierten Formaten konvertieren und eventuell auftretende Fehler behandeln.

3.3.1 Webapplikation

Die Webapplikation benötigt eine umfangreichere Applikationsschicht als die KVKapplikation. Internetbrowser können lediglich HTML Seiten darstellen, die sie zuvor angefordert haben. Beim Seitenaufruf können dann beliebige Parameter übergeben werden. Die Applikationsschicht auf dem Server ist dafür verantwortlich, aus den Ergebnissen der Datenbankabfragen, die in Tabellenform vorliegen, HTML Code zu erzeugen. Das Session Management, welches benötigt wird, um über das zustandslose HTTPS Protokoll einen Workflow abzubilden, muß von der Applikationsschicht bereitgestellt werden.

HTML Seiten sind jedoch nicht die einzige Ausgabe, welche die Applikationsschicht erzeugen muss. Für den Rechnungsdruck müssen die Rechnungen im PDF Format erzeugt werden, welche dann entweder von einem Webbrowser Plugin angezeigt werden oder direkt dem Drucker Spooler des Servers zum Ausdruck übergeben werden. Eine Vorlage, die das Layout der Rechnung enthält, ist auf dem Server gespeichert. In diese Vorlage werden dann die Posten der Rechnung eingesetzt und das generierte PDF wird entsprechend versandt. Lastschriften werden

Abbildung 6: Die unterschiedlichen Ausgabe Formate der Applikationsschicht



ähnlich behandelt. Die Applikationsschicht erstellt eine Textdatei in einem ähnlichen Format wie das bereits beschriebene Tabformat. Diese Textdatei enthält die Namen und Bankverbindung der Personen, von deren Konto eine Lastschrift abgebucht werden soll. Die Datei wird dann über den Webbrowser auf dem Desktop Computer des Benutzers gespeichert, wo sie von einer Homebanking Software importiert wird. Die Applikationsschicht besitzt zunächst keine direkte (elektronische) Schnittstelle zu den Rechenzentren der Banken.

In Abbildung 6 werden die unterschiedlichen Teile der Applikationsschicht gezeigt, welche Ausgaben in verschiedenen Datenformaten erzeugen — hier *Engines* genannt. Die Daten werden von allen Engines aus der Datenbank gelesen. Die Vorlagen für Webseiten und Rechnungen werden vom Dateisystem des Servers geladen. Alle Engines erzeugen dann ein Ausgabeformat, welches im Webbrowser darstellbar ist. Jedoch findet nur mit den HTML Webseiten eine Interaktion statt, d. h. die Applikationsschicht reagiert auf Eingaben des Benutzers im Webbrowser. Die Bankdatei und die Rechnungen können lediglich angezeigt und lokal (am Desktop des Benutzer) weiterverarbeitet werden. Dies ist durch die Richtung der Pfeile angedeutet.

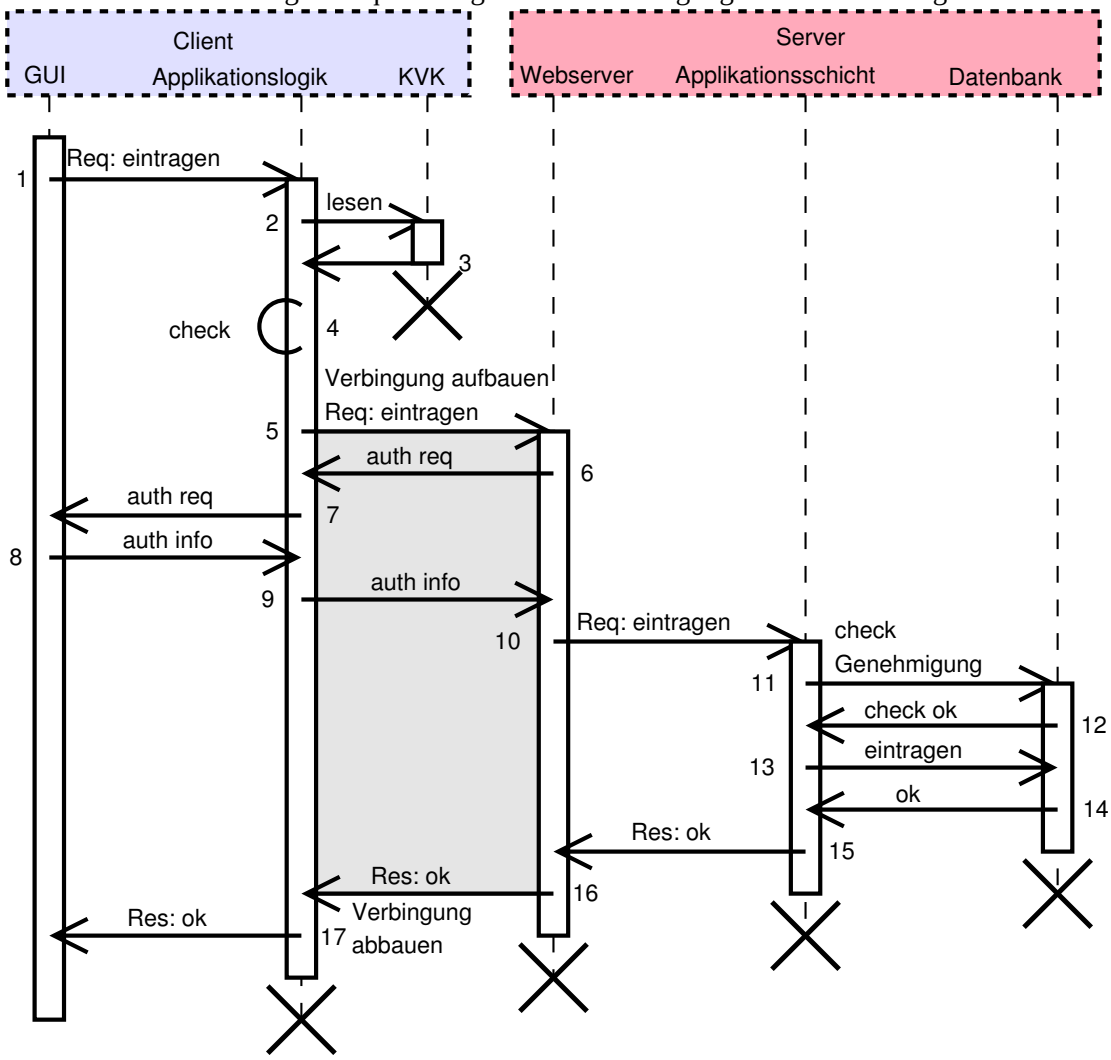
3.4 Beispiel

Als abschließendes Beispiel betrachten wir den Eintrag einer Behandlung in die Datenbank durch die KVKapplikation. Abbildung 7 zeigt ein Sequenzdiagramm, welches das Zusammenspiel der unterschiedlichen Komponenten auf Client und Server Ebene verdeutlicht.[19]

1. Der Vorgang beginnt, indem der Benutzer der KVKapplikation durch die GUI den Eintrag startet, z. B. durch Betätigung eines Buttons. Der generierte Event wird der Applikationslogik übergeben.

[19] Das große $\times$ am unteren Ende der Lebenslinie bedeutet nach dem UML Standard, dass das entsprechende Objekt zu diesem Zeitpunkt zerstört wird. In diesem Diagramm möchte ich damit andeuten, das der Zustand des Objektes ab diesem Zeitpunkt nicht mehr benötigt wird.

Abbildung 7: Sequenzdiagramm zur Eintragung einer Behandlung



2. Zunächst müssen die Daten des Patienten von der KVK gelesen werden. Dazu wird eine entsprechende Anfrage an die KVK-Komponente gemacht.

3. Konnte die KVK erfolgreich gelesen werden, so werden die Daten der Logikkomponente übergeben.

4. Die Daten der KVK werden soweit wie möglich von der KVKapplikation überprüft (Checksum, Gültigkeit).

5. Eine HTTPS Verbindung wird zum Webserver aufgebaut. Beim Verbindungsaufbau wird die Art der Abfrage (Behandlung eintragen) und die dafür notwendigen Daten der KVK übertragen. Die Verbindung bleibt bis zum Schritt 16 bestehen.[20]

[20] Angedeutet durch die Grau schattierte Fläche.

6. Die KVKapplikation wurde noch nicht beim Webserver authentifiziert, da die Daten beim HTTPS Protokoll nicht beim Verbindungsaufbau übertragen werden. Der Webserver schickt einen Autorisierungs Request an den Client.

7. Sollte der Benutzer der KVKapplikation sich noch nicht angemeldet haben, so muß spätestens jetzt seine Benutzername/Passwort Kombination abgefragt werden. In der Regel geschieht dies bereits beim Start der KVKapplikation. In diesem Fall würden die Schritte 7 und 8 übersprungen werden.

8. Die GUI übergibt Username und Passwort der Applikationslogik.

9. Die Autorisierungsdaten werden dem Webserver übermittelt.

10. Wenn der Webserver die KVKapplikation bzw. deren Benutzer autorisieren konnte, so wird die Anfrage an die Applikationsschicht übermittelt.

11. Die Applikationsschicht überprüft ob der Patient in der Datenbank vorhanden ist und ob eine Genehmigung zur Behandlung vorliegt.

12. Die Datenbank überträgt das Ergebnis der Prüfung wieder an die Applikationsschicht.

13. Ist eine Genehmigung erteilt, so wird die Behandlung in die Datenbank eingetragen.

14. Der Eintrag in die Datenbank wird der Applikationsschicht gemeldet.

15. Die Applikationsschicht erstellt eine (leere) Webseite und übergibt sie dem Webserver.

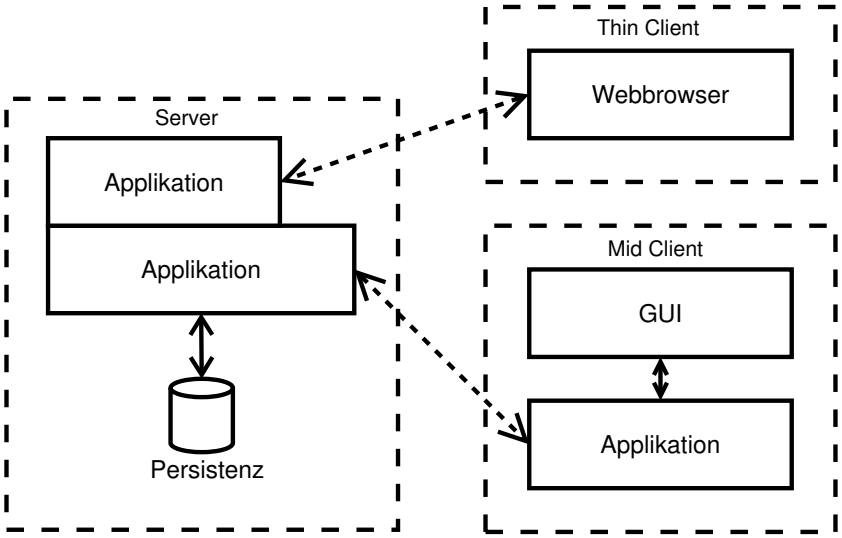
16. Der Webserver überträgt die (leere) Webseite an den Client. Da auch eine inhaltslose Webseite eine gültige HTTPS Übertragung ermöglicht, wird der Statuscode von HTTPS auf 200 (Ok) gesetzt. Ist die gesamte Webseite übertragen, so wird die HTTPS Verbindung abgebaut.

17. Anhand des Statuscodes erkennt die KVKapplikation die erfolgreiche Ausführung der Anfrage an den Server. Der Inhalt der Webseite wird nicht weiter verarbeitet. Das (positive) Ergebnis der Anfrage wird der GUI übergeben, damit der Benutzer den Abschluss des Vorgangs erkennen kann.

3.5 Zusammenfassung

Das Design hat gezeigt, dass die unterschiedlichen Anforderungen der Benutzergruppen zwei verschiedene Client Applikationen benötigen. Diese Applikationen müssen auf unterschiedlichen Hardware- und Softwareplattformen betrieben werden. Ferner implementieren die Applikationen unterschiedliche Teile der Applikationslogik (Thin- vs. Mid-Client). Auf der Serverseite ergeben sich dadurch zwei unterschiedliche Schnittstellen in die Applikationslogik. Die Verteilung der Schichten auf Clients und Server wird in Abbildung 8 veranschaulicht.

Abbildung 8: Schichten innerhalb von Clients und Server



4 Realisierung

Das im vorigen Kapitel entwickelte Design soll für ein Testsystem durch einen Prototyp implementiert werden. Der Prototyp soll in der Lage sein, alle für eine Abrechnung notwendigen Daten zu erfassen und eine Abrechnung zu erstellen. Wie bei Prototypen üblich, muss das Testsystem mit einem knappen Budget realisiert werden. Nach Möglichkeit sollte für die Client Systeme bei der RHL bereits vorhandene Hardware genutzt werden. Für die einzusetzende Software gibt es keine Vorgaben.

Dieses Kapitel erläutert, wie die Designvorgaben umgesetzt wurden. Die unterschiedlichen Softwarepakete von Drittherstellern für Clients und Server werden vorgestellt, zusammen mit dem sie *verbindenden* selbst entwickelten Sourcecode.

4.1 Hardware- und Softwarewahl

Aufgrund der Anforderung permanenter Verfügbarkeit des Serversystems kommen für den Server nur ausgereifte Server-Betriebssysteme in Frage. Zum Zeitpunkt der Entwicklung dieses Konzeptes waren dies OpenVMS, AS/400, S/390 und diverse Unix Systeme. Von diesen 4 Betriebssystemen bietet Unix die größte Auswahl an vorhandener Software. Es gibt (fast) jede Programmiersprache für Unix, alle Datenbankhersteller bieten eine Unix Version an und es gibt eine große Auswahl an Hardware Plattformen, auf denen das System installiert werden kann.

Die Client Computer können mit jedem Betriebssystem betrieben werden, welches eine ausreichende Netzwerk/Internet Unterstützung bietet. Dieses bieten mittlerweile alle Betriebssysteme, mit Ausnahme von kleineren Embedded Systemen. Die Unterstützung von grafischen Oberflächen ist heute ebenfalls Standard, so dass man bei der Wahl der Client Hardware und Betriebssysteme (fast) freie Auswahl hat.

Für den Prototyp wurden folgende Kombinationen von Hardware und Betriebssystemen ausgewählt. Der Server besteht aus einem 32bit x86 System und einem Unix Betriebssystem, vorzugsweise Linux. Die KVKapplikation kann mit Windows32, Windows CE und Unix/X11 betrieben werden, die Webapplikation sollte mit jedem modernen Webbrowser benutzbar sein, unabhängig vom Betriebssystem.

4.2 Datenbank

Die freie relationale Datenbank PostgreSQL enthält alle benötigten Funktionen. Im einzelnen werden neben den bekannten Zugriffen und Abfragen folgende Funktionen benötigt:

Transaktionen Komplexe Operationen auf der Datenbank, die durch mehrere Abfragen an die Tabellen zusammengesetzt werden, müssen als eine atomare Einheit behandelt werden. Dies verhindert, dass unterschiedliche Benutzer, welche die Datenbank gleichzeitig benutzen, einen inkonsistenten Zustand der Datenbank vorfinden. Durch das Ausführen der einzelnen Abfragen innerhalb einer Transaktion stellen sich diese für andere Benutzer als eine atomare Abfrage dar.

Stored Procedures Die Datenbank erlaubt es, selbst geschriebene Funktionen in der Datenbank zu speichern und von der Datenbank ausführen zu lassen. Mit dem Aufruf einer solchen Funktion kann beliebig komplexer Code ausgeführt werden.

Trigger Beim Zugriff (schreibend und/oder lesend) auf eine Tabelle kann vor oder nach dem Zugriff eine beliebige Funktion ausgeführt werden, auch selbst definierte Stored Procedures. Diese Funktion kann die Inhalte der Abfrage oder der Tabelle verändern oder aber die ganze Abfrage abbrechen. Mit Triggern sind unter anderem weiterreichende Konsistenzprüfungen möglich, als mit den einfacheren *check constraints*.

Das ERM aus dem vorigen Abschnitt lässt sich direkt in das Datenbankschema umsetzen. Jeder Entity und jeder Relation entspricht eine Tabelle. Die Beziehungen (Linien) zwischen diesen sind über die entsprechenden Fremdschlüssel hergestellt.

Für fast jedes Attribut findet eine Konsistenzprüfung auf der Datenbankebene statt. Per Definition müssen die Primärschlüssel natürlich eindeutig sein. Besteht der Primärschlüssel nur aus einer natürlichen Zahl[21], so wird dies durch den Datentyp *serial* sichergestellt. Bei kombinierten Primärschlüsseln, z. B. in der Tabelle Behandlungspreis, sind die einzelnen Attribute entweder Fremdschlüssel (und damit wiederum eindeutig) oder aber durch entsprechende *check constraints* gesichert. Bei sämtlichen Fremdschlüsseln sind keine *null values* erlaubt, so dass diese auf jeden Fall gültige Referenzen darstellen. Sollte der dazugehörige Primary Key sich ändern, dann wird die Änderung auch in die Fremdschlüssel übernommen.[22]

Die PostgreSQL Datenbank kann im laufendem Betrieb ein Image ihres Zustands in eine Datei auf dem Datenbankserver schreiben. Dieses Image kann dann auf ein externes Medium oder auf ein Netzwerklaufwerk kopiert werden. Eine Sicherung findet jede Nacht automatisch statt. Bei einem Datenbank- oder Serverfehler kann das Image auf einem neuen Server neu

[21] künstlicher Primärschlüssel
[22] ON UPDATE CASCADE Klausel

eingespielt werden und der Zustand der Datenbank von gestern wurde wiederhergestellt. Damit ist die Datenbank nicht hochverfügbar, da im schlimmsten Fall die gesamten Transaktionen eines Tages verloren gehen. Für eine höhere Verfügbarkeit muss man eine andere (relationale) Datenbank verwenden, die eine Datenspiegelung auf einen zweiten Server im laufenden Betrieb ermöglicht.[23] Bei einem Ausfall der Master Datenbank kann dann automatisch auf den Slave umgeschaltet werden, so dass das Gesamtsystem ohne Unterbrechnung betrieben werden kann.

4.3 Applikationsschicht, Webapplikation

Obwohl das Design *einen* Server für Datenbank und Applikationsschicht vorsieht, heißt dies nicht zwangsläufig, dass beide Teile auf *einem Computer* laufen müssen. Es kann durchaus für die Applikationsschicht und Datenbank je ein Computer verwendet werden. Für die Clients stellt dies trotzdem nur einen Server dar — da kein direkter Zugriff auf die Datenbank möglich ist, können sie nicht feststellen, wie die genaue Konfiguration des Servers aussieht. Es ist sogar möglich die Applikationsschicht und Datenbank auf mehr als zwei Computern zu betreiben (Round-Robin Webserver + Datenbankcluster), aber auch in diesem Fall bleibt der Server konzeptionell eine Einheit.

Die Applikationsschicht muß zwei unterschiedliche Clients bedienen. Die Logik der Webapplikation wird komplett auf dem Server realisiert. Dazu werden Webseiten in HTML generiert und versandt. Die KVKapplikation benötigt eine Schnittstelle für die Datensätze im Tabformat. Die Daten im Tabformat entsprechen den Ergebnissen der Anfragen an die Datenbank. Die eigentliche Verarbeitung der Ergebnisse nimmt die Applikationsschicht im Client vor. Die tabformatierten Datensätze für die KVKapplikation werden allerdings auch durch den Webserver übertragen.

Der Webserver ist der unter Unix weit verbreitete *Apache*. Er unterstützt das HTTPS Protokoll und verschlüsselt und sichert damit automatisch den ein- und ausgehenden Datenverkehr.

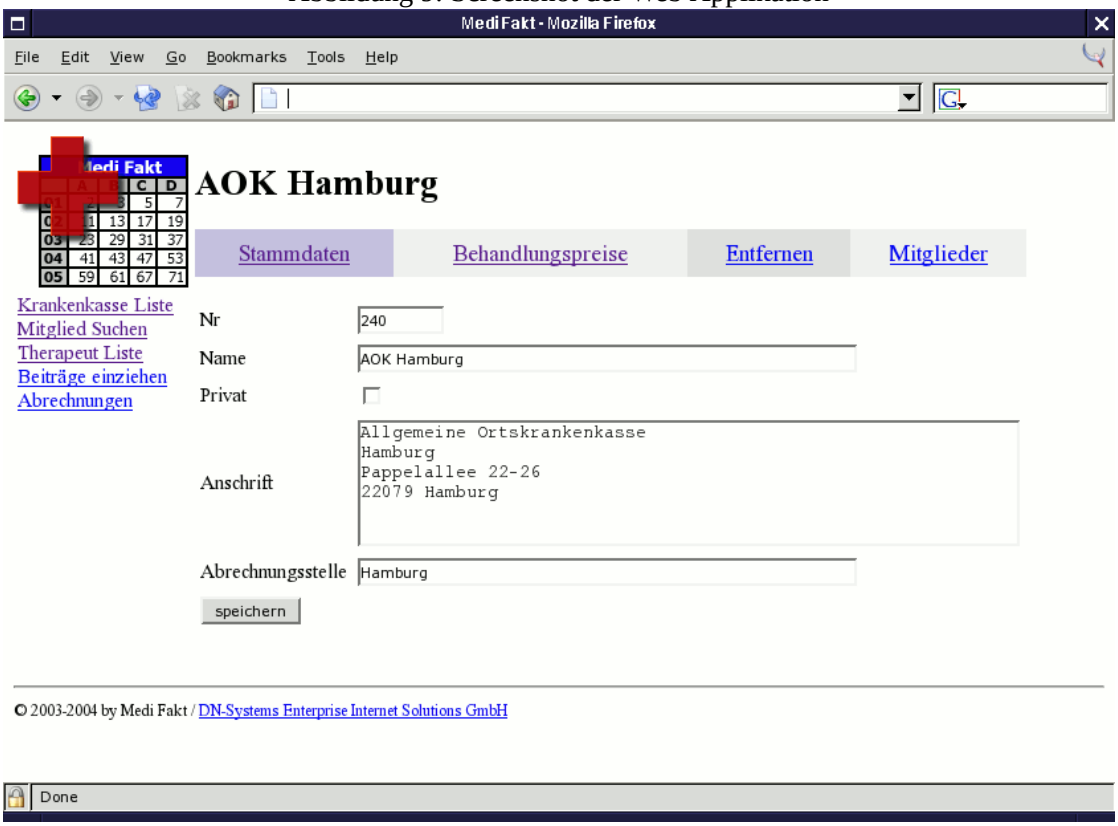
Die Applikationsschicht auf der Serverseite ist komplett in der Programmiersprache Perl implementiert. Perls Stärke ist die Verarbeitung von Text. HTML und das Tabformat sind text basierte Formate, die sich mit Perl leicht verarbeiten bzw. erstellen lassen. Die Schnittstellen zu relationalen Datenbanken verarbeiten die Daten ebenfalls in der Regel in Textform. Perl eignet sich daher ideal als Bindeglied zwischen den Text Formaten HTML und Tab und einer relationalen Datenbank. Ferner hat Perl eine Standard API für den Zugriff auf relationale Datenbanken (DBI - DataBase Interface), so dass ein Wechsel der Datenbank nur minimale Änderungen am Perl Quellcode erforderlich macht.

Der Teil der Applikationsschicht, der die Schnittstelle für die KVKapplikation anbietet, ist deutlich simpler als für die Webapplikation. Aus dem HTTPS Request wird eine Anfrage an die Datenbank erstellt. Das Ergebnis wird in das Tabformat umgewandelt und dem Client geschickt. Sollten bei der Abfrage Fehler auftreten, so wird dies durch den HTTPS Fehlercode signalisiert.

Die Webapplikation erfordert umfangreichere Funktionalität. Der Benutzer bedient die Webapplikation durch den Webbrowser, welcher Aktionen des Benutzers stets in die Abfrage einer (neuen) Webseite umsetzt. Durch die Übergabe von Parametern, welche i. d. R. Formularen auf der Webseite entnommen werden, wird die Applikationsschicht auf dem Server gesteuert. Sie

[23] Für PostgreSQL ist dieses Feature bereits für zukünftige Versionen angekündigt.

Abbildung 9: Screenshot der Web Applikation



setzt diese Parameter in Abfragen an die Datenbank um. Das Ergebnis der Datenbank wird dann in HTML Code umgewandelt (oftmals eine Tabelle) und als Webseite zum Browser geschickt. Der Perlcode, welcher die Applikationslogik implementiert, ist dabei in die Webseite integriert. Die Embperl Engine des Webservers verarbeitet diesen *eingebetteten* Code.

Zusätzlich zu Arbeiten mit der Datenbank haben Benutzer der Webapplikation die Möglichkeit, die Abrechnungen ihrer Hausbank elektronisch zu übermitteln, bzw. die Beträge per Lastschrift einziehen zu lassen. Dazu wird von der Applikationsschicht eine spezielle Textdatei generiert, welche die Kontodaten enthält. Diese wird genau wie Webseiten auch vom Webserver per HTTPS versendet und muss vom Benutzer lokal gespeichert werden. Danach kann er sie in eine (externe) Homebanking Software importieren. Eine direkte Verbindung zwischen dem Server und den Rechenzentren der Banken ist für den Prototyp nicht vorgesehen, da hierfür umfangreiche Zertifizierungen des Systems notwendig wären.

Über die Webapplikation wird ebenfalls der Rechnungsdruck gesteuert. Der Benutzer kann sich einzelne Rechnungen in einer Vorschau als PDF im Webbrowser betrachten[24] oder viele Rechnungen im Batchbetrieb drucken. Die Rechnungen werden mit dem Satzsystem LaTeX auf dem Server erzeugt. Das Grundgerüst der Rechnung ist in einer LaTeXDatei auf dem Server gespeichert. In diese Datei werden die Posten der Rechnung, Adresse des Empfängers usw.

[24] Dies setzt natürlich das Vorhandensein eines entsprechenden PDF-Viewers auf dem Desktop Computer voraus.

eingesetzt. Anschließend erstellt LaTeXauf dem Server die Datei. Die generierte Datei wird dann entweder per HTTPS an den Webbrowser versandt oder dem Druckerspooler des Servers übergeben. Anhang A und B zeigen je eine Rechnung für eine Mitglied und eine Krankenkasse.

4.4 KVKapplikation

Die KVKapplikation benutzt drei verschiedene Schnittstellenklassen: Netzwerk, GUI, Kartenleser. GUI und Netzwerk werden vom Betriebssystem bereitgestellt und durch eine Klassenhierarchie plattformunabhängig von der Applikation angesprochen. Der Zugriff auf Kartenleser erfolgt mittels zwei verschiedener Standards, dem CT-API Standard für den Zugriff auf den Kartenleser und dem CT-BCS Standard zum Zugriff auf die Smartcard. Die Hersteller der Kartenleser stellen i. d. R. eine Library zur Verfügung, die Funktionen zum Zugriff auf die Kartenleser anbietet. Diese Libraries sind normalerweise in C/C++ geschrieben und somit am einfachsten mit einem C/C++ Programm anzusprechen. Die Betriebssysteme, bzw. deren Netzwerk und GUI Schnittstellen können ebenfalls mit C/C++ angesprochen werden. C++ bietet die Abwärtskompatibilität zu C und durch seine objektorientierten Eigenschaften die Möglichkeit die vom Design geforderten Abstraktionen durch Klassenhierarchien zu erstellen. Die KVKapplikation wurde deshalb in C++ realisiert.

Die Windows KVKapplikation benutzt die normale GUI von Windows. Für Unix Systeme wird die QT GUI Bibliothek benutzt. Für die verschlüsselte Netzwerkübertragung mit HTTPS benutzt die Unix Implementation die OpenSSL Bibliothek, Windows32 und CE stellen HTTPS mit ihren normalen Netzwerkfunktionen zur Verfügung. Die Desktop Rechner bei Therapeuten wählen sich mittels ISDN bei Bedarf in das Internet ein. Die Smartcards werden mit Kartenlesern der Firma Kobil gelesen. Diese sind für die Verarbeitung von KVK zertifiziert. Die mobilen Rechner[25] kommunizieren über einen WLAN/ISDN Accesspoint, der sich ebenfalls bei Bedarf in das Internet einwählt.

Die KVKapplikation ist streng interaktiv ohne Nebenläufigkeiten.[26] Nach dem Programmstart und der Authentifikation des Benutzers wird auf ein Kommando vom Benutzer gewartet. Dieses Kommando liest entweder die Karte im Kartenleser ein oder stellt eine Abfrage an den Server. Während des Lesevorgangs bzw. der Abfrage darf die KVKapplikation blockieren. Dadurch ist ein einfacher Programmaufbau möglich, in dem nur auf Ereignisse der GUI reagiert werden muss.

Von den in Abbildung 3 gezeigten Klassen sind *Tabformat*, *Logikfunktionen* und *KVK* plattformunabhängig; der selbe Code (die selbe Quelldatei) wird von allen drei Implementierungen verwendet. Die Klassen *Netzwerk* und *Smartcardleser* sind abstrakte Interfaceklassen, über die *Tabformat* und *KVK* auf die jeweiligen Netzwerk- und Kartenleserklassen zugreifen. Obwohl die APIs für grafische Oberflächen von Windows CE und Windows32 (fast) identisch sind, wurden bei diesem Projekt komplett getrennte GUI Klassen erstellt, denn durch die unterschiedlichen Bildschirmformate eines Handhelds und eines Desktop Computers mussten komplett unterschiedliche GUI Dialoge erzeugt werden.

[25] Compaq Ipaq mit Omnikey PCMCIA Kartenleser und Cisco Aironet PCMCIA WLAN

[26] Auch die Webapplikation bietet keine Nebenläufigkeiten und ist interaktiv, d. h. reagiert nur auf Eingaben des Benutzers. Das Bedienkonzept für beide Applikationen ist also identisch.

Abbildung 10: Dialog zum Eintrag einer Behandlung unter Unix und CE

Der Vorteil dieser Aufteilung liegt darin, dass die Applikationslogik/schicht der KVKapplikation in den plattformunabhängigen Klassen *Tabformat*, *Logikfunktionen* und *KVK* enthalten ist. Bei zukünftigen Änderungen an der Applikationslogik oder der Applikationsschicht auf dem Server müssen nur die plattformunabhängigen Bestandteile der KVKapplikationen geändert werden.

4.5 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie sich das Design direkt umsetzen lässt. Durch die Verwendung bereits fertiger Softwarepakete (Datenbank, Webserver) muss nur wenig Code selber geschrieben werden. Den größten Anteil des selbst entwickelten Codes stellen die (plattformabhängigen) Teile der KVKapplikation. An zweiter Stelle steht der Teil der Applikationsschicht auf dem Server, auf dem die HTML Seiten für die Webapplikation erzeugt werden. Die wichtige Persistenzschicht, welche auch die Konsistenz der Daten garantiert, konnte komplett durch Funktionen der Datenbanksoftware ohne selbst geschriebenen Code implementiert werden.

Das Testsystem für die RHL wurde mit einem Server, je einem mobilen und festen Client für Therapeuten im Januar 2004 in Betrieb genommen. Die Sachbearbeiter der RHL arbeiten mit den vorhandenen Windows PCs und der Webapplikation. Anfang April wurde eine Abrechnung für das erste Quartal 2004 erstellt, die die Rechnungen für Mitglieder, welche therapiert wurden, und die Krankenkassen umfasst. Damit wurde der Testbetrieb erfolgreich abgeschlossen und das Funktionsprinzip der Software in der Praxis erprobt.

Abbildung 11: Smartcardleser mit PCMCIA und RS232

5 Resümee

Die Rheuma Liga Hildesheim möchte ein EDV-System erwerben, welches die Erfassung und Abrechnung von Rheuma Therapien ermöglicht. Bislang werden Therapien manuell mit Unterschriftslisten erfasst und von Mitarbeitern der RHL manuell abgerechnet. Durch ein EDV-System verspricht sich die RHL eine deutliche Verkürzung der Bearbeitungszeiten zur Erstellung der Abrechnungen, welche ferner keine Fehler beinhalten sollen.[27]

Das Analyse Kapitel zeigte, mit welchen Hardware Bestandteilen und Software Komponenten sich ein solches Abrechnungssystem realisieren lässt. Ein klassisches Client-Server System, in welchen Clients und Server über das Internet kommunizieren, ist die einfachste Möglichkeit solch ein System zu erstellen. Im Kapitel Design wurden die einzelnen Bestandteile spezifiziert. Das vorige Kapitel zeigte schließlich, dass sich das Design mit wenig Aufwand umsetzen läßt. Durch die Verwendung von vorhandener Software für die wichtigsten Teile des System (Datenbank, Webserver, Netzwerkkommunikation, GUI) ließ sich das Testsystem innerhalb von 2 Monaten erstellen.

Der im ersten Quartal 2004 durchgeführte Testbetrieb des Systems ist erfolgreich abgeschlossen worden. Das System konnte Rheumatherapien erfassen und im April wurde eine Abrechnung für Krankenkassen und Mitglieder erstellt. Damit wurde gezeigt, dass das vorgestellte Design eine funktionsfähige Lösung der Aufgabenstellung darstellt, welche im Kostenrahmen realisiert werden konnte. Nach einer eintägigen Schulung konnten die Sachbearbeiter und Therapeuten sicher mit dem System umgehen. Die Abrechnung geschieht nun auf *Knopfdruck*, wodurch die Rechnungen automatisch ausgedruckt werden und bereit sind für den Versand mit der Post.

Sollte sich die RHL zum Erwerb eines Produktivsystems entschließen, so muß das bisherige

[27] Bei der manuellen Abrechnung entstehen viele Übertragungs- und Rechenfehler.

Testsystem an einigen Stellen überarbeitet werden. Es muß eine umfangreichere Fehlerbehandlung implementiert werden, da momentan die Eingabedaten nur soweit überprüft werden, dass die Konsistenz der Datenbank gewährleistet ist. Des weiteren sind die für den Benutzer sichtbaren Fehlermeldungen oftmals die direkten SQL Fehlermeldungen, die für die Benutzer i. d. R. unverständlich sind. Das Grundgerüst der Implementation auf Client und Serverseite kann jedoch beibehalten werden, da der Prototyp nicht als *Wegwerfimplementation* geplant und realisiert wurde.

Das System ist bislang darauf ausgelegt, dass die RHL das System betreibt, also einen eigenen Server besitzt und in ihren Räumlichkeiten betreibt. Da die RHL wenig Know-How im Betrieb von Client-Server Systemen auf Internet Basis besitzt und der Betrieb eines Rechenzentrums auch nicht zum *Kerngeschäft* einer Rheuma Liga gehört, wird ein externer Dienstleister benötigt, welcher die Client- und Servercomputer betreut. Solch ein externer Dienstleister könnte natürlich auch für andere Rheuma Ligen das Abrechnungssystem betreiben. Dazu müsste die Software mandantenfähig gemacht werden, so dass alle Kunden (die verschiedenen Rheuma Ligen) mit einem Server arbeiten würden. Dazu muß nur das Datenbankmodell erweitert werden, die Applikationsschicht kann unverändert übernommen werden. Auf der Clientseite wären ebenfalls keine Änderungen notwendig. Der Betreiber würde damit für die Rheuma Ligen zu einen *Application Service Provider* (ASP). Der ASP kann neben dem Betrieb der Computer dann weitere Dienstleistungen anbieten, z. B. könnte die Abrechnung mit Patienten und Krankenkassen über die elektronischen Schnittstellen zur Bank erfolgen. Insbesondere der Rechnungsdruck und -versand stellt für die RHL einen großen Aufwand einmal pro Quartal dar, da über 2000 Rechnungen für die Mitglieder zu Drucken und Versenden sind, was einen großen manuellen Aufwand bedeutet. Der Erwerb einer automatischen Druck-, Kuvertier- und Frankiermaschine lohnt sich jedoch für solche Mengen nicht. Bietet jedoch ein ASP den Rechnungsdruck für mehrere Rheuma Ligen an, so wird der Erwerb einer solchen Maschine vom ASP wirtschaftlich.

Literatur

[air] Cisco Aironet 350 Wireless LAN client adapter. http://cisco.com/en/US/products/hw/wireless/ps4555/ps448/index.html.

[apa] Apache HTTP Server. http://httpd.apache.org/.

[Bag03] Don Bagwell. Understanding HTTP session management. Technical Report WP100316, IBM - Washington Systems Center, January 2003. 3.2.1

[Bar04] Roman Bartnik. no title yet. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, 2004. 2.6

[Bis02] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley Longman Publishing Co., Inc., 2002. 3.2

[car] Omnikey Cardman Mobile PCMCIA4000. http://omnikey.de/de/produkt_details.php?produkt=5&variante=5.

[Cod70] E. F. Codd. A relational model for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970. 3.1.1

[DA99] T. Dierks and C. Allen. RFC2246: The TLS Protocol. http://ietf.org/rfc/rfc2246.txt?number=2246, jan 1999. Version 1.0. 3.2.1

[dbi] DBI - Perl DataBase Interface. http://dbi.perl.org/.

[DBn] Database normalization. http://en.wikipedia.org/wiki/Database_normalization. 9

[emb] Embperl – a framework for building websites with perl. http://perl.apache.org/embperl/.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC2616: Hypertext Transfer Protocol – HTTP/1.1. http://ietf.org/rfc/rfc2616.txt?number=2616, jun 1999. 11

[FHBH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. RFC2617: HTTP Authentication: Basic and Digest Access Authentication. http://ietf.org/rfc/rfc2617.txt?number=2617, jun 1999. 3.2.1

[ipa] Hewlett-Packard IPAQ handheld computer. http://welcome.hp.com/country/us/en/prodserv/handheld.html.

[kob] Kobil CT-B1-S Smartcard Reader. http://www.kobil.de/d/products/smartcard/kaan-ct-b1-s.php.

[lat] LaTeXHome. http://www.latex-project.org/.

[Oll04] Gunter Ollmann. Web based session management. http://www.technicalinfo.net/papers/WebBasedSessionManagement.html, 2004. 3.2.1

[ope] OpenSSL - a free multiplatform cryptography library. http://www.openssl.org/.

[pos] PostgreSQL relational database. http://www.postgresql.org/.

[PSZ04] Patrick Postel, Sebastian Schünemann, and Jaroslaw Zdrzalek. Sicherheit in kommerziellen WLAN-Systemen. Master's thesis, Hochschule für Angewandte Wissenschaften Hamburg, jan 2004. 2.6, 18

[qt] QT - a platform independant widget toolkit. http://www.trolltech.com/.

[Res00] E. Rescorla. RFC2818: HTTP Over TLS. http://ietf.org/rfc/rfc2818.txt?number=2818, may 2000. 3.2.1

[Rol03] Fred D. Rolland. *Datenbanksysteme im Klartext*. Prentice Hall - Pearson Studium, 2003. 5

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw.§25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 8. April 2004 Dirk Jagdmann

A Quartalsrechnung für eine Krankenkasse

Rheuma-Liga Niedersachsen, AG Hildesheim
Rmerring 96
31337 Hildesheim

AOK Niedersachsen
Hans-Bckler-Allee 30
30173 Hannover

7. April 2004

Abrechnung 01.01.2004 - 31.03.2004

Sehr geehrte Damen und Herren

Bitte berweisen sie den Rechnungsbetrag für das erste Quartal 2004.

Name	Vorname	GebDatum	VNr	Art	Anzahl	Gesamt
Mustermann	Rita	02.03.1957	547542920	Trockentherapie	6	€13.80
aaaaaaaaa	aaaaaaaa	25.10.1927	535303551	Trockentherapie	6	€13.80
bbbbbbbb	bbbb	01.04.1948	543375439	Trockentherapie	5	€11.50
cccccc	ccccc	15.07.1947	543110929	Trockentherapie	6	€13.80
ddddddd	dddddd	20.06.1950	544352899	Trockentherapie	8	€18.40
eeeee	eeeeee	24.03.1946	574228538	Trockentherapie	1	€2.30
fffffff	ffffffff	09.02.1947	542948428	Trockentherapie	1	€2.30
gggggggg	gggggggg	12.07.1946	542729058	Trockentherapie	3	€6.90
hhhhhhhhh	hhhhhhhh	25.10.1927	535303551	Warmwassertherapie	1	€3.32
iiiiii	iiiii	03.09.1931	536679385	Trockentherapie	4	€9.20
jjjjjjjj	jjjjjj	04.05.1947	543037907	Trockentherapie	1	€2.30
kkkkkkkk	kkkkkkkk	10.06.1948	543452605	Trockentherapie	1	€2.30
llllll	lllllll	20.11.1934	537855984	Trockentherapie	2	€4.60
mmmmmmmm	mmmmmmmm	17.06.1931	536605978	Trockentherapie	8	€18.40
nnnnnn	nnnnnnnn		542729252	Trockentherapie	1	€2.30
ooooo	oooooo	13.10.1929	535992920	Trockentherapie	4	€9.20
				Rechnungssumme gesamt		**€134.42**

B Quartalsrechnung für ein Mitglied

Rheuma-Liga Niedersachsen, AG Hildesheim
Römerring 96
31337 Hildesheim

Frau
Rita Musermann
Musterstraße 4
31137 Hildesheim

7. April 2004

Therapie-Abrechnung 01.01.2004 - 31.03.2004

Sehr geehrte Frau Mustermann,

Sie haben vom 01.01.2004 bis 31.03.2004 Leistungen die folgenden Leistungen in Anspruch genommen.

Art	Anzahl	Einzelpreis	Betrag
	Zuzahlungen		
Trockentherapie	6	€0.50	€3.00
Rechnungssumme gesamt			**€3.00**

Wir bitten den Betrag von **€3.00** auf unser Konto zu überweisen.

Konto Nr	43 476
BLZ	259 500 01
Bank	Stadtsparkasse Hildesheim

Mit freundlichem Gruß

gez. Schmidt

C Version

1.0 erstes Release

1.01 2004-07-21: Verbesserung vieler Interpunktions- und Rechtschreibfehler