iVisite
(mobile Patientenakte)
Dokumentation

Michael Kudla B.Sc[1], Andreas Günther B.Sc.[2]
HTW Berlin University of Applied Science
Treskowallee 8, 10318 Berlin
[1]michael.kudla@student.htw-berlin.de
[2]andreas.guenther@student.htw-berlin.de

17. Oktober 2011

Inhaltsverzeichnis

1 Ziel

Die vorliegende Dokumentation beschreibt die Idee, den Entwurf und die prototypische Implementierung einer mobilen elektronischen Patientenakte. Dieses prototypische System bildet die Grundlage, um medizinisches Personal bei der alltäglichen Patientenvisite zu unterstützen. Dabei wird eine Lösung angestrebt, die keine großflächige Infrastruktur benötigt. Damit sind WLAN-Lösungen ausgeschlossen und stattdessen ist die Nutzung der RFID-Technologie mit Bluetooth vorgesehen. Die Kopplung der NFC Technik mit Bluetooth ermöglicht die prinzipielle Umsetzung einer NFC- Peer- To- Peer Anwendung. Dabei wird ein RFID- Tag als Autorisierungsschlüssel genutzt, um im Anschluss einen Kommunikationskanal zwischen zwei Geräten zu erstellen beispielsweise mit Bluetooth.

2 Motivation

Elektronische Patientenakten sind gegenüber papierbehafteten Akten wesentlich komfortabler für medizinisches Personal wie etwa beim Transport. Ein weiterer Vorteil ist, dass die Patientendaten dezentral eingesehen werden können und sind daher einfacher, im Kontext des Patienten abzurufen. Digitale Systeme können ebenfalls die Datenintegrität sicherstellen und vor Gefahren wie etwa der Datenmanipulationen schützen, da die Patientendaten nur autorisiertem Personal zur Verfügung stehen. Im Rahmen von dauerhaften Protokollierungen des Patientenzustands können bereits heute schon Algorithmen Rückschlüsse auf den zukünftigen Patientenzustand schließen. Es können bereits 80%ig zutreffende Wahrscheinlichkeiten nachgewiesen werden, ob ein Pflegepatient unter häuslichem Missbrauch leidet. Ein weiterer Vorteil für den Einsatz von elektronischen Patientenakten ist aufgrund der ganzheitlichen Protokollierung die bessere Prüfbarkeit von tatsächlich erbrachten Gesundheitsdienstleistungen.

3 Herausforderungen

Für die Entwicklung der Client Anwendung sind iPhone, iPod und auch iPad Geräte vorgesehen. Dabei ist die Datenübermittlung die eigentliche Herausforderung, da das Client-Gerät über Bluetooth kein Serial Port Profile (SPP)

unterstützt und somit keine serielle Datenübertragung möglich ist. Um dennoch Informationen an den Client via Bluetooth zu senden, wurde das Human Interface Device (HID) Bluetooth-Profil verwendet, welches jedoch als großen Nachteil nur unidirektionale Kommunikation zum Client ermöglicht. Dabei zu beachten war, dass die gesendeten Daten an den Client auf das englischsprachige Tastaturlayout des Geräts abgebildet wurden, was ein entsprechendes Umwandeln der Daten notwendig machte. In Folge dessen musste für die Datenübermittlung ein eigenes Protokoll entworfen und implementiert werden, wodurch garantiert werden sollte, dass eine vollständige und korrekte Datenübertragung stattfindet (siehe Abschnitt 6).

4 Allgemeiner Workflow

Die allgemeine Vorgehensweise am System sieht vor, dass ein mobiler Client – versehen mit einem RFID-Tag – sich an einem Terminal anmeldet. Dabei wird die MAC[1]-Adresse, die auf dem RFID Tag gespeichert wurde, an ein zentrales System versendet, um Informationen zu dieser MAC- Gerätenummer zu erhalten. Diese bereits kodierten Daten (siehe Abschnitt 6) werden vom Terminal über die bereits erlangte Bluetooth Verbindung letztendlich an den Client über Tastaturbefehle verschickt. Somit ist die Kommunikation am Terminal abgeschlossen und es kann erneut die Informationsverarbeitung vom Terminal durch neue RFID Tags gestartet werden.

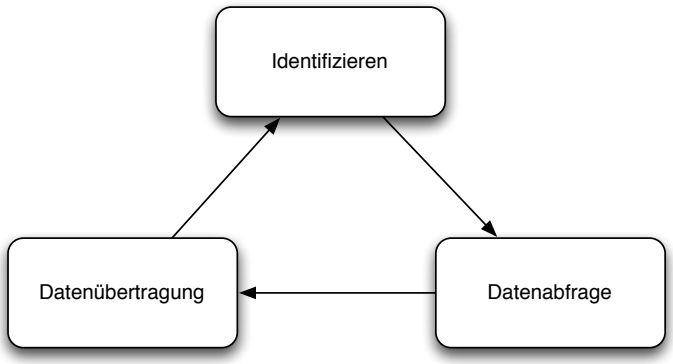

Abbildung 1: Allgemeiner Workflow des Systems

[1]Media-Access-Control

5 Architektur

Das Gesamtsystem, wie in Abbildung 2 dargestellt , gliedert sich in drei Subsysteme: Client, Terminal und Server. Das mobile Endgerät in Verbindung mit einem RFID-Tag bilden zusammen den Client. Das Terminal besteht aus einem RFID-Reader, einem Arduino-Microkontroller mit den notwendigen Hardwaremodulen für Ethernet und Bluetooth (siehe Abschnitt 7.2) und einem Ethernet-Hub, der alle Terminalkomponenten miteinander sowie diese auch mit dem Server verbindet.

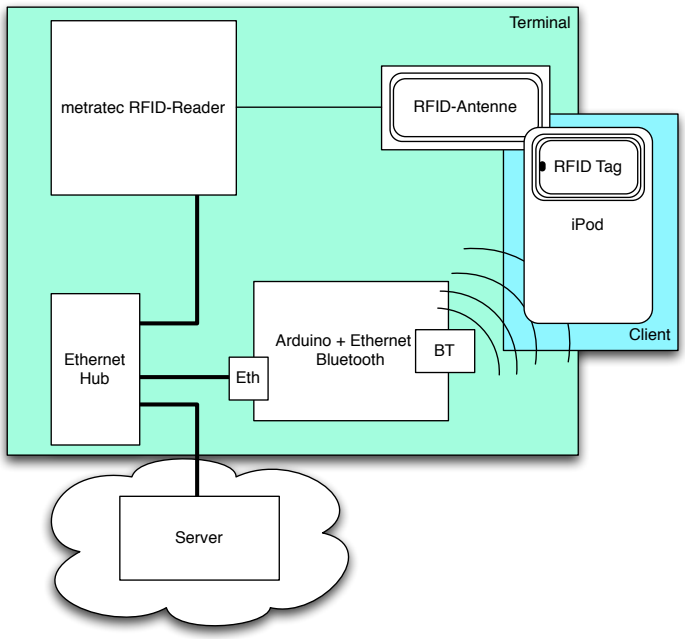

Abbildung 2: Aufbau des Gesamtsystems

6 Protokoll

Das Protokoll beschreibt zwei Rahmentypen für den Datenaustausch. Zum einen einen Transportrahmen und zum anderen einen Datenrahmen in dem die Nutzdaten enthalten sind. Der Datenrahmen wird bei der Übertragung in den Transportrahmen integriert, als dessen Payload. In Abbildung 3 ist schematisch dargestellt, wie beide Rahmen aufgebaut sind. Abgesehen von

Start- und Endsymbol haben alle Felder der Beiden Rahmentypen eine Variable länge.

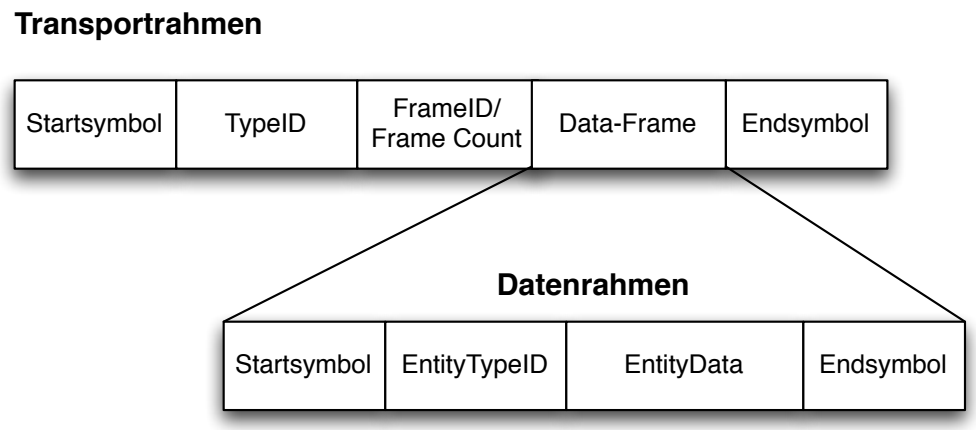

Abbildung 3: Schematischer Protokollaufbau

6.1 Transportrahmen

Dieser Rahmen dient zum Transport der eigentlichen Nutzdaten bzw. gibt an, wie viele Datenrahmen im Anschluss gesendet werden. Er besteht aus *Start-* und *Endsymbol*, Einer *TypID*, einem Feld zur Angabe der *FrameID* bzw. der Anzahl der folgenden Frames (*Frame Count*) und einem Feld (*Data-Frame*) für den zu transportierenden Datenrahmen. Ob der Rahmen Nutzdaten führt oder Aufschluss über die Folgepakete gibt wird durch einen Rahmentyp festgelegt. Für diesen Rahmen sind zwei Typen definiert:

- Initial-Frame (Wert: 0 für *FrameID*)
- Data-Frame (Wert: 1 für *FrameID*)

Der Frame-Type Initial-Frame definiert einen initialen Transportrahmen, der die Anzahl der Folgerahmen angibt. Hier wird auf den *Data-Frame*-Teil des Rahmens komplett verzichtet. Die Anzahl der folgenden Transportrahmen wird im Feld *FrameID/Frame Count* angegeben. Beim Typ Data-Frame übermittelt der Transportrahmen einen Datenrahmen als Nutzlast.

Um die variable Feldlänge der TypID zu gewährleisten, wurde zwischen diesem Feld und dem Folgefeld ein zusätzliches Trennzeichen eingefügt (siehe Abschnitt 6.3), welches auf Abbildung 3 nicht dargestellt ist, da dieses fester Bestandteil des Feldes *FrameID/Frame Count* ist.

6.2 Datenrahmen

Der Datenrahmen dient dazu die eigentlichen Nutzdaten zu kapseln, die der Client nach empfang weiterverarbeitet. Er besitzt ebenfalls Start- und Endsymbol, wie der Transportrahmen. Dazu besitzt er ein Typfeld (*EntityTypeID*), welches angibt, welcher Entität die Nutzdaten angehören. Es sind insgesamt drei Entitäten-Typen definiert:

- Arzt (Wert: 0 für *EntityTypeID*)
- Patient (Wert: 1 für *EntityTypeID*)
- Behandlung (Wert: 2 für *EntityTypeID*)

Im Anschluss an das *EntityTypeID*-Feld folgen die eigentlichen Daten im *EntityData*-Feld. Die Daten der Entität werden durch ein spezielles Zeichen getrennt und in das *EntityData*-Feld eingesetzt.

6.3 Steuersymbole

In Tabelle 1 werden alle verwendeten Steuersymbole aufgeführt und deren Zuordnung zu den Protokollfeldern. Hier ist allerdings zu beachten, dass die genutzten Steuersymbole möglichst nicht innerhalb der eigentlichen Nutzdaten benutzt werden sollten, sonst kommt es zu Fehlinterpretationen des Protokolls.

In Listing 1 ist ein typischer Datensatz des Protokolls gezeigt. Zuerst wird der Initiale Transportrahmen gesendet, der die Anzahl der folgenden Frames bekanntgibt. Danach folgen die Transportrahmen mit den Datenrahmen als Nutzlast.

[	Startsymbol Transportrahmen
]	Endsymbol Transportrahmen
(	Startsymbol Datenrahmen
)	Endsymbol Datenrahmen
#	Identifikationszeichen für *FrameID/Frame Count*
|	Trennsymbol für Nutzdaten

Tabelle 1: Steuersymbole des Protokolls

```
[0#3][1#0(0|<Data>)][1#1(1|<Data>)][1#2(2|<Data>)]
```

Listing 1: Beispiel eines Protokolldatensatzes

7 Systembestandteile

Im Folgenden werden die einzelnen Komponenten des Systems beschrieben. Wie sind sie aufgebaut und welche Funktion haben sie? Auf den RFID-Reader der Firma Metratec wird nicht näher eingegangen.

7.1 Client

Der Client besteht, wie eingangs erwähnt, aus einem iPod und einem RFID-Tag. Das RFID-Tag identifiziert das Gerät am Terminal und enthält zusätzlich die Bluetooth-MAC-Adresse des iPods, mit Hilfe dessen eine Bluetooth-Verbindung vom Terminal zum Client aufgebaut wird. Im Anhang A sind Screenshots des Clients aufgeführt.

Die iOS-Anwendung auf dem iPod besteht aus fünf Komponenten (siehe Abbildung 4) und ist nach dem von Apple modifizierten Model-View-Controller-Pattern aufgebaut.

Da für die Bluethoothverbindung das HID-Profil benutzt wird, empfängt die Anwendung innerhalb des *Communication Controller*s die Daten auch an der Schnittstelle zur Zeicheneingabe, an der Methode *insertText:*, die jedes *UIView* im Cocoa-Framework besitzt. Hier muss allerdings noch ein zusätzliches Mapping vorgenommen werden, da die empfangenen Zeichen nicht den ge-

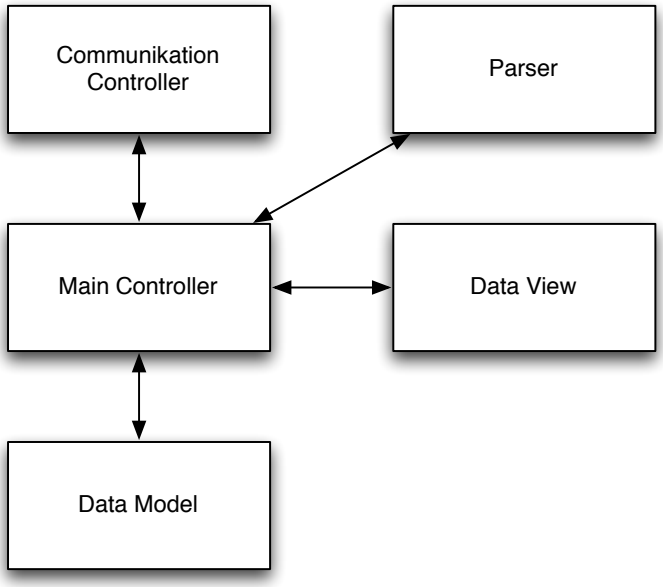

Abbildung 4: iOS-Client Architektur

sendeten entsprechen. Dies liegt darin begründet, dass durch die Nutzung es HID-Profiles für den Datenaustausch das Terminal für den Client eine externe Tastatur repräsentiert. Daher werden alle eingehenden Zeichen anhand ihres Tastaturcodes für die englische Standardbelegung interpretiert und anhand des aktuell am Client-System eingestellten Tastaturlayouts an das Programm weitergegeben. Somit erhält man bei einem abweichenden Tastaturlayout von der englischen Belegung auch andere Zeichen. Klassisches Beispiel: für ein gesendetes z erhält man ein y bei eingestelltem deutschem Tastaturlayout. Ganz besonders Kritisch ist dies beim versenden von Sonderzeichen, die beim hier spezifizierten Protokoll definiert sind.

Nachdem ein kompletter Transportrahmen eingegangen ist, extrahiert der *Parser* die Nutzdaten und gibt sie an den *Main Controller* weiter, welcher diese anschließend an das *Data View* weitergibt um sie dem Nutzer anzuzeigen.

7.2 Arduino

Das Arduino ist das Herzstück des Terminals. Es steuert sowohl den RFID-Reader als auch die Serveranfragen und die Datenübertragung. Dazu ist es mit einem Ethernetshield und einem Bluetoothmodul (Bluegiga WT32) ausgestattet. Wie das Arduino mit dem Bluetoothmodul verschaltet ist, kann Abbildung 11 entnommen werden. Das Ethernetshield ist einfach auf das Arduino aufgesteckt.

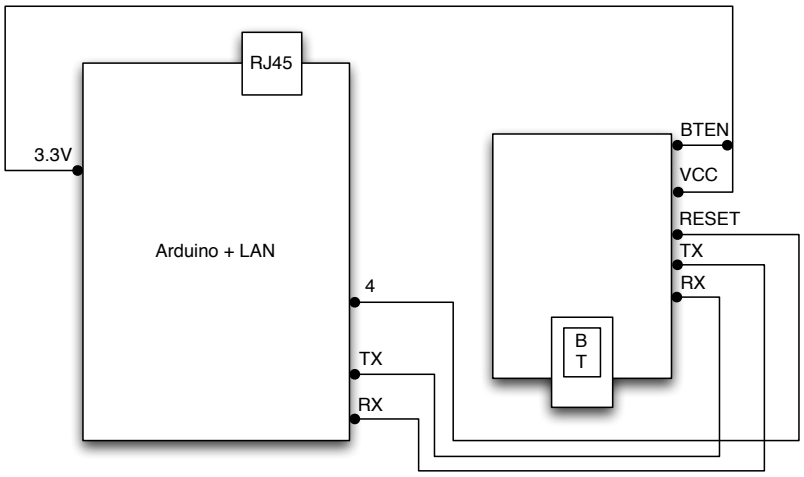

Abbildung 5: Verschaltung des Arduino mit dem Bluetoothmodul

Das Arduino initialisiert zunächst den RFID-Reader und weißt ihn anschließend an, in Antennenreichweite befindliche RFID-Tags aufzuspüren. Befindet sich ein Tag innerhalb der Reichweite, ließt das Arduino neben der UID des Tags auch die darauf enthaltenen Daten, die MAC-Ardesse des Bluetooth-Gerätes für die Verbindung, aus. Im Anschluss versucht das Arduino eine Bluetoothverbindung mit dem mobilen Client anhand der ausgelesenen MAC-Adresse aufzubauen. Steht diese Verbindung, wird nun eine Verbindung zum Datenserver aufgebaut und die Daten für die UID des RFID-Tags abgerufen. Alle Daten, die nun vom Server gesendet werden leitet das Arduino zum Client über die aufgebaute Bluetoothverbindung weiter. Dies wird solange gemacht, bis vom Server ein Quittierungszeichen, hier ein @, empfangen wurde, welches dem Arduino signalisiert, dass die Datenübertragung beendet ist. Anschließend, werden beide Verbindungen abgebaut und das System wartet wieder auf RFID-Tags in Reichweite. In einem Fehlerfall verfällt das Arduino auch wieder in diesen Status zurück. Dieser Funktionelle Ablauf ist in Abbildung 6 dargestellt.

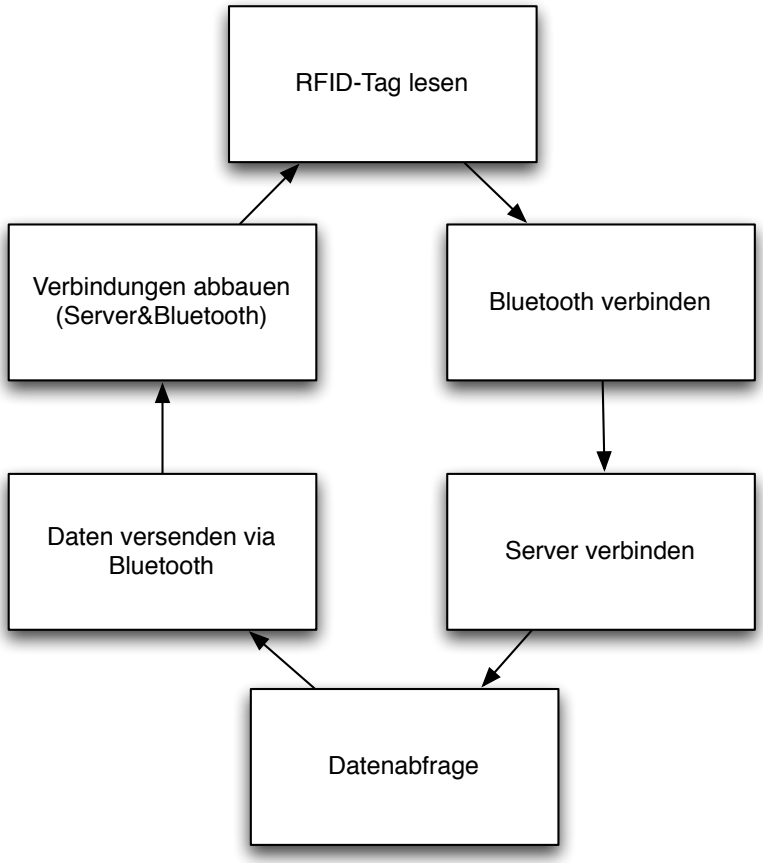

Abbildung 6: Arduino funktionaler Ablauf

7.3 Server

Die zentrale Datenhaltung des Systems wird mittels einer MySQL- Datenbank realisiert. Aus Performance-Gründen mussten die Daten zum Terminal bereits kodiert werden, welche innerhalb der serverseitigen Web Service Geschäftslogik zusammengefasst wurde. Damit liefert der Server die Basis dafür, dass mehrere Terminals mit dem Server kommunizieren können.

Als Systemgrundlage diente ein Apache 2.2.17 Web-Server mit der PHP-Erweiterung in der Version 5.3.5. Für den Web Service wurde die NuSOAP Bibliothek genutzt, die grundsätzliche SOAP Web Service Funktionalitäten kapselt, ohne eine PHP Erweiterung installieren zu müssen.

Da sich der Client nur einmalig am Terminal registriert und folgend Daten vom Server abgeholt werden, müssen diese einmalig vom Server abgerufen werden. Dabei werden die Daten in das notwendige Protokoll (siehe Abschnitt 6) umgesetzt und dem Terminal zurückgeliefert.

Dabei werden die Daten innerhalb des Web-Service in der folgenden Form kodiert:

Beispiel für den Arzt- Frame (Type-ID = 0)

```
(type-ID | Name      | Datensatz-ID | mobile-ID)
(0       | Dr. House| 1            | 123123123123123)
```

Die mobile-ID ist die Grundlage für die Zuordnung des Arzt-Gerätes zum System, da die ID die MAC-Adresse des Client Geräts repräsentiert.

Beispiel für den Patient- Frame (Type-ID =1)

```
(type-ID | Name | Geschlecht | Patient seit | Arzt-ID | Datensatz-ID)
(1       | alex | m          | 01112009     | 1       | 1 )
```

Beispiel für den Behandlungs- Frame (Type-ID=2)

```
(type-ID|Beschreibung|Plan-Datum|Ist-Datum|Patienten-ID|Datensatz-ID)
(2      | Massage    | 23112011 | 23112011| 1          |1 )
```

Bei der Übermittlung der Daten erhält der Arzt lediglich die Daten seiner zugehörigen Patienten.

8 Fazit & Ausblick

Das vorliegende Projekt zeigt, wie ein System für eine mobile Patientenakte entworfen und entwickelt werden könnte und es dazu zwangsläufig keine großflächige Infrastruktur benötigt wird. Die Umsetzung von mobilen Patientenakten bietet als entscheidenden Vorteil die zeitliche Entlastung des medizinischen Personals, um sich gezielter auf die Bedürfnisse der Patienten einzugehen. Der derzeitige Stand des Projekts bietet jedoch noch einige Optimierungsmöglichkeiten seitens der Systemkomponenten, um einen professionellen Einsatz zu gewährleisten.

Bei der Serverkomponente bieten die WebService Funktionalitäten zusätzliche Autorisierungsfunktionen zwischen Server und den Terminals. Außerdem bieten Web-Services hohe Systemskalierbarkeit und einfache Erweiterbarkeit.

Im Betrieb des Terminals fällt auf, dass die Übertragungsgeschwindigkeit vom Terminal zum Client relativ langsam ist. Das liegt daran, dass das Arduino die vom Server empfangen Daten byteweise weitersendet und den Versand des nächsten Bytes um 40 Millisekunden verzögert. Dies war nötig, damit auch alle gesendeten Zeichen am Client ankommen. Bei einer geringeren Verzögerung kann es zu erheblichen Datenverlust kommen. Ein möglicher Grund dafür könnte sein, dass Eingaben an einer Tastatur durch einen Nutzer nur in einer, mit der Anschlagsschnelligkeit zusammenhängenden, bestimmten Geschwindigkeit vorgenommen werden können. Daher ist zu vermuten, dass das iOS auch nur in einer gewissen Geschwindigkeit auf Tastatureingaben reagiert und bei höheren Übertragungsgeschwindigkeiten nicht alle gesendeten Daten empfangen kann. Ein weiter Punk in diesem Zusammenhang ist, dass die eingehenden Daten noch einmal gepuffert werden, bevor sie letztendlich an den Client versendet werden. Durch die Eliminierung des Pufferns an dieser Stelle, könnte die Übertragungsgeschwindigkeit erhöht werden.

Um die Herausforderung des Apple Clients erfolgreich umsetzen zu können, benötigte es einiger Umgehungslösungen bezüglich der Clientkommunikation. Hier musste für die Datenübertragung das HID-Profil genutzt werden, um Daten per Bluetooth an den Client zu versenden. Ein Datenaustausch vom Client zum Terminal ist mit diesem Profil allerdings nicht möglich. Ob Apple in Zukünftigen iOS-Versionen das dafür nötige SPP-Profil integriert bleibt abzuwarten. Solange könnte zumindest eine Umstellung auf einen mobilen Client, der dieses Profil unterstützt (z.B. Android, Geräte mit J2ME Unterstützung) und die Umstellung des Terminals auf das SPP-Profil vorgenommen werden.

Des Weiteren ist bei der Übertragung der Daten mittels emulierter Tastatureingabe noch keine Datensicherheit gegeben.

A Client Screenshots

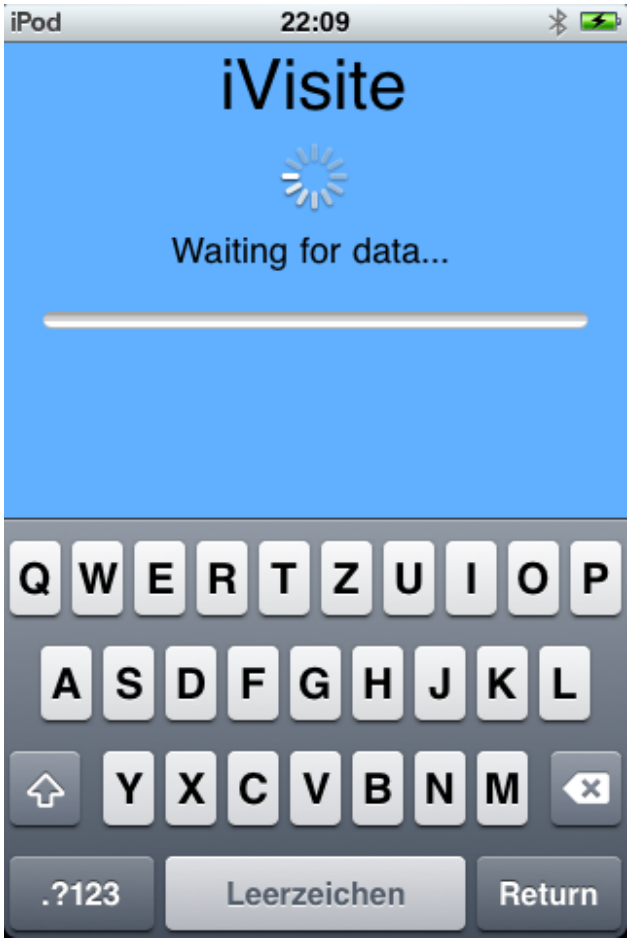

Abbildung 7: Client: Warten auf Bluetoothverbindung mit Terminal (Simuliert Tastatur)

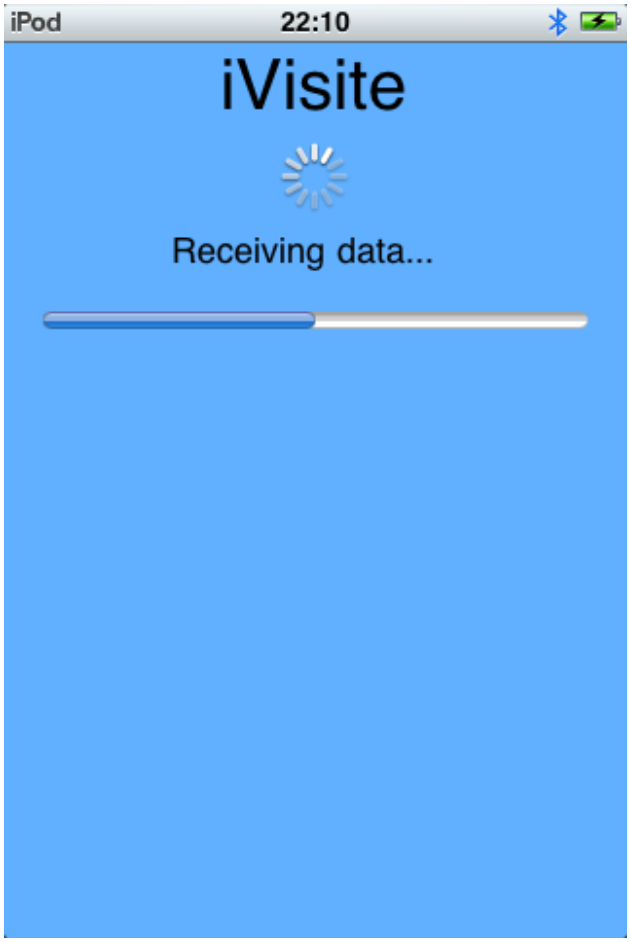

Abbildung 8: Client: Datenempfang

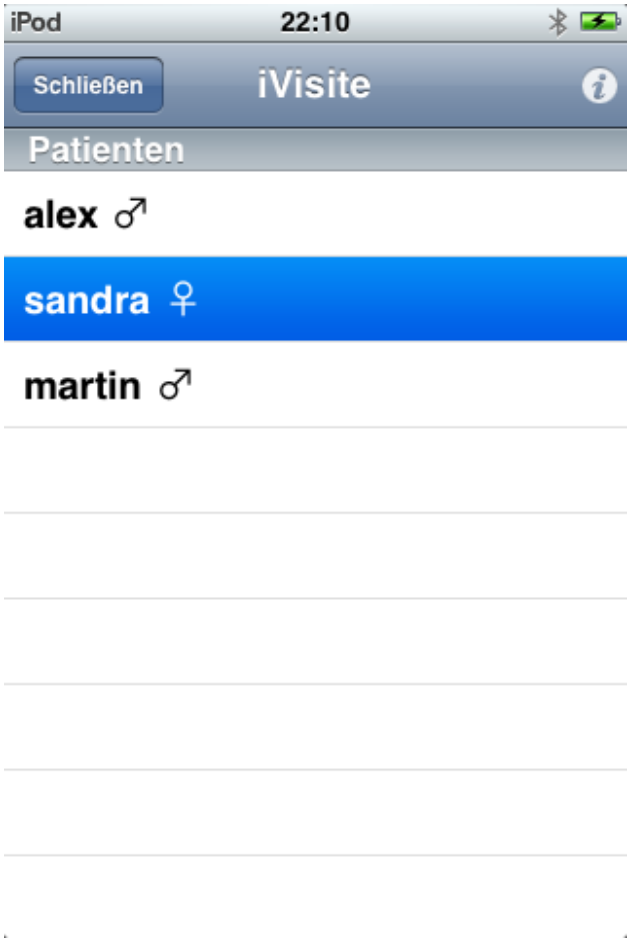

Abbildung 9: Client: Patientenliste

Abbildung 10: Client: Liste der Behandlungen für einen Patienten

Abbildung 11: Client: Behandlungsdetails