Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Jan Szensny

Entwicklung eines E-Payment-Systems
auf Basis mobiler Endgeräte

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Szensny

Entwicklung eines E-Payment-Systems auf Basis mobiler Endgeräte

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Softwaretechnik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Abgegeben am 20. Oktober 2005

Jan Szensny

Thema der Diplomarbeit

Entwicklung eines E-Payment-Systems auf Basis mobiler Endgeräte

Stichworte

ePayment, Zahlung, Zahlmethoden, Ferienclub, Kredit, PDA, eWallet, eCash, DRM, Kerberos

Kurzzusammenfassung

Da die Ferienclubs ihren Gästen heutzutage den Urlaub so bequem wie möglich gestalten wollen, liegt die Frage nahe, warum nicht dem Gast anbieten die Geldbörse und seine Ausweise im Tresor zu lagern, und alle Bezahl- und Identifikations-Aufgaben einem kleinen persönlichen Helfer zu übergeben. Dieser kleine persönliche Helfer soll ein mobiles Endgerät sein, welches dem Gast als Geldbörse und Ausweis dienen und zusätzlich auch weitere Dienste zur Verfügung stellen könnte. Diese Arbeit beschäftigt sich mit der Entwicklung eines Systems, welches die Geldbörse des Gastes ersetzen soll. In diesem System soll der Gast ein Guthaben auf seine elektronische Geldbörse laden und dieses innerhalb und außerhalb des Clubnetzwerkes verwenden können. Zusätzlich kann der Gast innerhalb des Clubs noch ein Kreditkonto zur Verfügung gestellt bekommen über welches er, ohne Guthaben zu besitzen, bis zu einem gesetzten Limit Waren erwerben kann. Im Rahmen dieser Arbeit wird ein solches System von der Vision ausgehend über die Analyse bis zum Entwurf vorgestellt. Zusätzlich wurde ein Prototyp entwickelt, der beweist, dass das vorgestellte Konzept funktioniert.

Jan Szensny

Title of the paper

Development of an ePayment-System based on mobile devices

Keywords

ePayment, Payment, payment methods, Club Holiday, Credit, PDA, eWallet, eCash, DRM, Kerberos

Abstract

Nowadays holidays clubs want to provide their guests with vacations that should be as comfortable as possible. So, why not store all real money and all documents of identification in the safe deposit and use an small personal assistant for paying and all tasks of identification. This small personal assistant could be a mobile device that could also provide further services. This work concerns itself with the development of a system, which is to replace the purse of the guest. In this system the guest should be able to load virtual money to his electronic purse and use it within and outside of the club network.Additionally the guest should be able to use an credit account within the club network.This credit account could be used to buy goods up to a specified limit, while the guest do not possess sufficient money. In the context of this work the vision, the analysis as well as the draft of such a system are presented. Additionally a "'Proof-of-Concept"' prototype has been developed.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation

Es gibt viele Probleme in Bezug auf Geld während des Urlaubs. Abgesehen davon, dass man i.d.R. immer zuwenig davon besitzt, ergeben sich, je nach Urlaubsort bzw. -land, weitere Probleme.

Wie viel ist das jetzt in Euro? Wie kann ich bezahlen? Welchen Preis hat er gerade genannt, wie viel muss ich zahlen? Wohin mit meiner Geldbörse? Wo ist mein Clubausweis? Ich will aber nicht, dass mein Sohn sich Bier kauft!

Dies sind nur ein paar der Probleme die während des Urlaubs auftreten könnten. Und um den Ausgleich einiger dieser Probleme soll sich das hier entwickelte ePayment-System kümmern.

Entwicklung eines ePayment-Systems auf Basis mobiler Endgeräte

Was ist ein ePayment-System?
Ein ePayment-System stellt ein System dar, welches den Austausch von Zahlungsmitteln auf elektronischer Basis in einem definierten System ermöglicht.

Im Internet sind bereits einige, zum Teil weltweit agierende, ePayment-Unternehmen vertreten, z. B. PayPal oder moneybookers.com um nur zwei zu nennen.

Bei herkömmlichen Bezahlsystemen, sei es normale Währung die wir aus unserem Portemonnaie nehmen, Coupons oder Perlen wie sie vor einiger Zeit in Ferienclubs eingesetzt wurden, findet immer ein direkter Tausch statt, der Käufer übergibt dem Verkäufer eine bestimmte Menge an Zahlungsmitteln und erhält dafür die gewünschte Ware ausgehändigt.

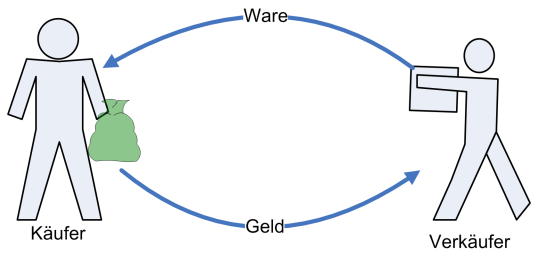


Abbildung 1.1.: allgemeiner Tauschablauf beim Einkauf

In der vorstehenden Grafik 1.1 würde ein ePayment-System die Rolle folgender Elemente übernehmen:

1. Zahlungsmittel
2. Zahlungsweg

Jede Transaktion von Zahlungsmittel wird also durch das ePayment-System durchgeführt. Gleichzeitig verwaltet dieses System die Zahlungsmittel für den Nutzer.

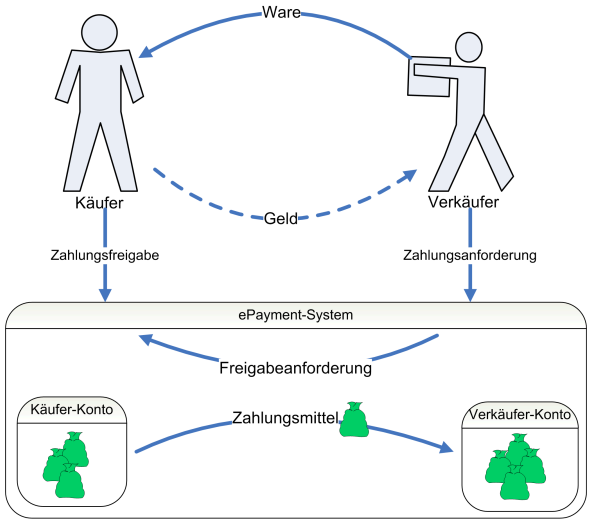


Abbildung 1.2.: Tauschablauf beim Einsatz eines ePayment-Systems

Wie aus der vorstehenden Abbildung 1.2 zu ersehen ist, wird für die Zahlungstransaktion zwischen Käufer und Verkäufer von beiden Seiten nur jeweils eine Aktion benötigt. Der Verkäufer teilt dem System mit „wer“ „was“ zu zahlen hat und der Käufer teilt dem System mit, ob diese Transaktion korrekt ist und gibt somit ggf. die Transaktion frei, sagt dem System also dass der geforderte Betrag an den Fordernden übertragen werden kann.

Was kann man unter „auf Basis mobiler Endgeräte“ verstehen?

Um diese Frage zu beantworten sollen zuerst die „mobilen Endgeräte“ definiert werden.

Diese sind allgemein definiert transportable Geräte, die in der Lage sind, die für dieses System benötigte Software auszuführen.
Im Abschnitt „Experimente“ des Kapitels „Analyse“ (3.3.1 ab S. 17) werden die für diese Arbeit genutzten Geräte, sowie dabei aufgetretene Probleme, beschrieben.

Mit der Definition „auf Basis mobiler Endgeräte“ soll gemeint sein, dass das hier entstehende System derart gestaltet werden soll, dass zu mindest auf der Seite des Gastes ein mobiles Endgerät wie z. B. ein PDA oder ein Handy zum Einsatz kommen soll.

1.2. Zielsetzung

Im Verlauf dieser Ausarbeitung soll ein kleiner Prototyp dieses Systems entstehen, der exemplarisch eine mögliche Nutzungsart für dieses ePayment-System darstellen soll.

So soll dieser Prototyp die grundlegenden Zahlungsfunktionen innerhalb eines Clubnetzwerkes bereitstellen, auf die mit Hilfe von „externen“ Geräten zugegriffen werden soll. Die Schnittstelle auf Clubseite stellen z. B. die Terminals an Rezeption und Bars dar. Auf Seite des Gastes soll ein kleines mobiles Endgerät, z. B. ein Mobiltelefon oder ein PDA, zum Einsatz kommen.

Mit den Problemen die in Bezug auf die Sicherheit in einem solchen System entstehen können bzw. existieren sowie möglichen Lösungsansätze in diesem Zusammenhang hat sich Andre Lüpke in seiner Diplomarbeit „Entwurf einer Sicherheitsarchitektur für den Einsatz mobiler Endgeräte“ (Lüpke 2004) beschäftigt.

Im Anschluss an diese Einleitung wird zuerst die Vision, die hinter dieser Diplomarbeit steht, sowie die Idee aus der Sie entstanden ist, vorgestellt (siehe 2, ab S. 11).

Hierauf folgt die Analyse (siehe 3, ab S. 13), die anhand eines Use-Cases eine grobe Definition des zu entwickelnden Systems aufzeigen soll. Da die Analyse bereits in Szensny (2005) erfolgt ist, wird hier nur eine kurze Analyse wiedergegeben werden. Zusätzlich werden in diesem Kapitel einige Experimente die im Rahmen der Analyse sowie der weiteren Entwicklung durchgeführt wurden und die hierzu verwendete Hardware vorgestellt.

Anschließend soll auch schon mit dem Entwurf (siehe 4, ab S. 22) fortgefahren werden. In diesem Abschnitt werden die einzelnen Komponenten und teilweise Verwandschaften mit bereits bestehenden und teils auch etablierten Softwarelösungen vorgestellt.

Nachdem die theoretischen Vorarbeiten nun erledigt sind, wurde ein Prototyp als „Proof of Concept“ auf Grund der bisher erarbeiteten Informationen implementiert.
Im Rahmen des Kapitels „Evaluation“ (siehe 5, ab S. 51) wird nun auf Besonderheiten eingegangen, die im Zusammenhang mit der Implementierung aufgetreten sind.

Zum Abschluss soll, im Rahmen einer Zusammenfassung dieser Arbeit, noch ein Ausblick (siehe 6, ab S. 71) genommen werden, an welchen Stellen des hier entstandenen Systems weitere Entwicklungen stattfinden könnten bzw. sollten.

2. Vision

Als Grundlage dieser Arbeit stand die Idee[1] eines Systems, welches den Gästen eines Ferien-Clubs zu jeder Zeit sämtliche Informationen über die aktuellen und allgemeinen Angebote des Clubs bietet. Als Zugriffspunkt auf diese Informationen sollten PDAs dienen, die an jeden Gast bei dessen Ankunft im Club ausgehändigt werden.
Diese Idee wurde weiter gesponnen bis Funktionen wie das Buchen von Ausflügen, Tennisplätze oder Plätzen in Restaurants in das System gedacht waren.
Eine weitere Überlegung bestand darin, den PDA gleichzeitig, neben der Funktion als Informationsquelle, noch zum Club-Ausweis zu machen, wodurch der Gast alleine durch das Mitführen des aktivierten Gerätes seine Zugangsberechtigung zu den einzelnen Bereichen des Clubs aufzeigen konnte.

An dieser Stelle wurde dann die Frage gestellt, weshalb soll ein Clubgast nun immer diesen PDA und seine Geldbörse (oder die jeweiligen Zahlungsmittel) mit sich führen. Würde es nicht genügen, wenn der Gast seinen PDA dabei hätte?

Mit dieser Frage war die Grundidee zu dieser Arbeit geschaffen, aber das reichte noch nicht. Nun wurde diese Grundidee zu einer großen Vision entwickelt.

So sollte dieses System dem Gast die absolute Kontrolle über sein Guthaben geben. Das Guthaben des Gastes sollte auf dem PDA verwaltet werden. Der Gast sollte also sein Guthaben zu jederzeit, wie in einer realen Geldbörse, mit sich führen und kontrollieren können.
Gleichzeitig soll dem Gast die Möglichkeit gegeben werden an geeigneten Stellen auf ein Kredit-Konto zurückgreifen zu können.

Als Besonderheit soll dann noch eine Erweiterung dieses mittlerweile gewachsenen Systems hinzukommen. Dem Gast soll die Möglichkeit gegeben werden die Verwendung seines Guthabens zu limitieren. Ob nun ein tägliches oder wöchentliches Limit für den Guthabenverbrauch eingestellt wird oder direkt einzelne Artikel bzw. Artikelarten gesperrt bzw. limitiert werden sollte vollständig in der Hand des Gastes liegen.
Als Grundlage zu dieser Überlegung stand das ständige Problem dem hin und wieder die Clubmitarbeiter unterliegen, zu unterscheiden ob ein jugendlicher Gast alt genug ist um bestimmte Getränke (z. B. alkoholische) zu erwerben.

[1] Nachzulesen in „ePayment in Ferienclubs: Entwurf eines ePaymentsystems unter Nutzung von PDA's“ (Szensny 2005)

Durch die Einrichtung eines Tokensystems, das wie beschrieben frei einstellbar ist, sollte so z. B. den Eltern die Möglichkeit gegeben werden den Alkoholerwerb ihrer Sprösslinge zu unterbinden.
Natürlich kann der Einsatz auch für andere Gastgruppen interessant sein.

Neben diesen Funktionen, die als Erweiterung des Grundsystems gedacht sind, gibt es natürlich auch noch die Möglichkeit dieses System für Funktionen einzusetzen, die nicht als Bezahlfunktionen im eigentlichen Sinne gesehen werden.
Als Beispiel wären hier die bereits zu Beginn dieses Abschnittes genannten Buchungsmöglichkeiten in einem Clubsystem zu nennen. So könnten Gäste z. B. jede Woche einmal einen Tennisplatz für eine Stunde kostenlos gestellt bekommen. Oder sie könnten einmal in der Woche ein Spezialitätenrestaurant besuchen.
Zur Realisierung könnten in unserem System z. B. spezielle „Club-Token“ eingesetzt werden, die zur Buchung eines solchen Angebots aufgebraucht würden bzw. nach Ablauf verfallen.

Weiterhin kam die Überlegung auf, bei größeren Club-Anlagen könnte nicht unbedingt jeder Bereich sinnvoll ans Clubnetzwerk angebunden werden. Jedoch wäre für das aktuell geplante System eine Anbindung ans Clubnetzwerk nötig.
Somit sollte die Funktionsfähigkeit auch dann erhalten bleiben wenn der PDA des Gastes keine Anbindung ans Clubnetzwerk hat.
So soll der Gast mit seinem PDA z. B. auch an der clubeigenen Strandbar zahlen können trotz der Tatsache, dass diese nicht direkt am Clubnetzwerk hängt und somit nicht auf das ePayment-System zugreifen kann.
Weiterhin soll Gästen die Möglichkeit gegeben werden ihr Guthaben miteinander auszutauschen während sie nicht in Reichweite des Clubnetzwerkes sind (z. B. am Strand oder beim Einkaufen im Nachbarort).

Es gibt sicherlich noch viele Erweiterungsmöglichkeiten die bisher nicht genannt wurden.
Diese hier vorgestellte Vision soll nun als Grundlage für die weiteren Schritte dienen.

3. Analyse

3.1. Grundlagen

In der Vision ist ein ziemlich komplexes System entstanden, das nun soweit möglich realisiert werden soll.
Die Hauptfunktion dieses Systems soll darin bestehen den gesamten Bereich der Zahlungsmittel in diversen Transaktionen zu übernehmen.
Auf der einen Seite des Systems befindet sich z. B. ein Club-Mitarbeiter, der eine Getränkebestellung an der Bar in das System eingibt und somit den entsprechenden Betrag in Zahlungsmitteln vom Gast anfordern.
Auf der anderen Seite befindet sich ein Gast, der diese Anforderung bestätigen bzw. bei nicht gewollten Transaktionen ablehnen soll.

Die komplette Verarbeitung der Transaktionen, insbesondere alle Prüfungen, erfolgt innerhalb des ePayment-Systems.

Für die weitere Entwicklung dieses Systems soll der nachfolgende Use-Case dienen.

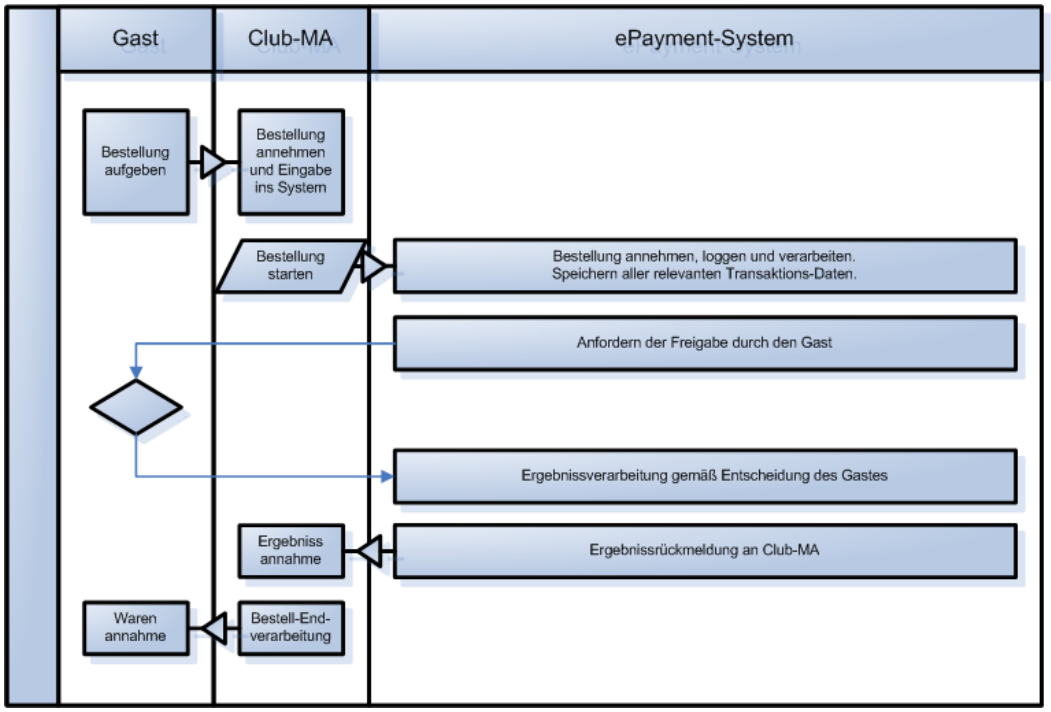


Abbildung 3.1.: Ablaufdiagramm (grobe Darstellung)

Ein Gast gibt bei einem Club-Mitarbeiter eine Bestellung auf.
Der Club-Mitarbeiter gibt diese Bestellung ins ePayment-System des Clubs ein und startet die Transaktion.
Das ePayment-System prüft nun ob die Transaktion von System-Seite her durchgeführt werden kann und fordert im Anschluss daran eine Freigabe des entsprechenden Gastes an.
Aufgrund der Rückmeldung des Gastes bzgl. der Transaktion verarbeitet das ePayment-System die Transaktion entsprechend zu Ende und meldet das Transaktionsergebnis an den Club-Mitarbeiter zurück.
Dieser führt daraufhin die Bestellung aus (positive Rückmeldung) oder teilt dem Gast den Grund für die Nicht-Zahlung mit.

Zu diesem Beispiel wurden einige Dialog-Beispiele erstellt. Diese sind im Anhang B ab Seite 78 zu finden.

3.2. Komponenten

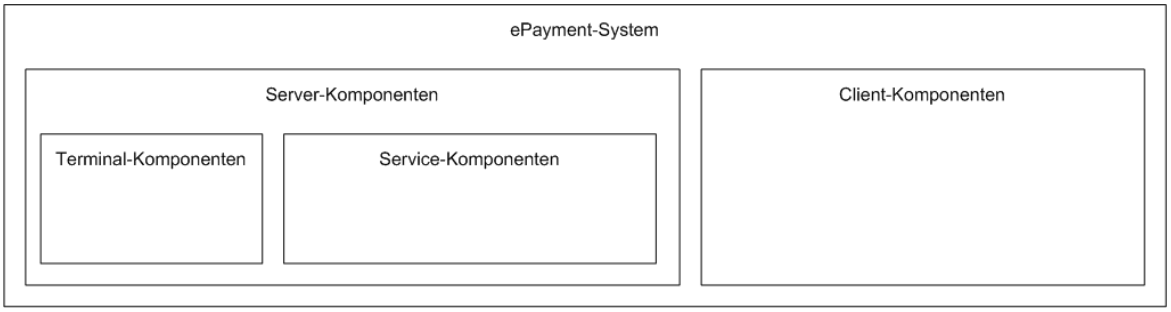


Abbildung 3.2.: Komponenten-Diagramm

Wie aus dem Komponenten-Diagramm (Abb. 3.2) hervorgeht, soll es sich bei diesem System um eine Client-Server-Struktur handeln, eigentlich sogar um zwei Client-Server-Strukturen.
Auf der einen Seite stehen die Server-Komponenten, welche für die Verarbeitung und Verwaltung der Zahlungsmittel sowie der Zahlungsanforderungen zuständig sind.
Auf der anderen Seite befindet sich die Client-Komponente, die in erster Linie für die Bestätigung eingehender Zahlungsanforderungen zuständig ist.

Außerdem besteht die Server-Komponente selbst auch noch aus zwei Teilsystemen.
Als erstes wären die Service-Komponenten als Teilsystem genannt, die jeglichen vom System benötigten Service bereitstellen sollen.
Als zweites kommen dann noch die Terminal-Komponenten hinzu, die im Grunde genommen nur die Benutzer-Schnittstellen des Systems darstellen.

Eigentlich könnten die Terminal-Komponenten ebenfalls auf seiten der Client-Komponenten stehen. Jedoch sind bei dem aufgezeigten Diagramm bereits einige Informationen mehr berücksichtigt worden.
So sollen die Terminal-Komponenten kurz gesagt selbst nur ein Minimum an Logik erhalten und nur eine Logging-Funktion für Transaktionen erhalten. Sie sollen nur als „Thin-Clients“ implementiert werden.
Die Client-Komponente hingegen wird aus mehreren Teilsystemen bestehen, die im Grunde genommen, autonom arbeiten können sollen. Die Client-Komponente soll also als „Rich-Client“ implementiert werden.

Nachdem nun die drei Hauptkomponenten dieses Systems kurz ausgearbeitet wurden, soll das System an dieser Stelle noch etwas detailliert werden.

Wie bereits in der Analyse in „ePayment in Ferienclubs: Entwurf eines ePaymentsystems unter Nutzung von PDA’s“ (Szensny 2005) erarbeitet wurde, soll dieses System eine Kombination aus eWallet- und eCash-Lösungen enthalten.

D.h. jeder Gast besitzt auf seinem mobilen Endgerät eine elektronische Geldbörse (eWallet), in der er sein Guthaben direkt verwalten kann. Zusätzlich erhält jeder Gast noch ein Guthaben-Konto bzw. Kredit-Konto innerhalb des Clubnetzwerkes.
Bei Transaktionen kann der Gast dann entweder von seinem eWallet-Guthaben-Konto Zahlungen tätigen oder seinen „Kreditrahmen“ im eCash-System ausnutzen.

Wie bereits in den vorhergehenden Abschnitten angesprochen, soll noch ein Tokensystem entwickelt werden.
Dieses Tokensystem soll dazu dienen dem Gast die Möglichkeit zu bieten seinen Guthabenverbrauch zu beschränken. Alternativ soll es auch möglich sein, bestimmte Waren (z.B. alkoholische Getränke) vom Erwerb auszuschließen.

Weitergehende Details können in „ePayment in Ferienclubs: Entwurf eines ePaymentsystems unter Nutzung von PDA’s“ (Szensny 2005) nachgelesen werden.

Zur Verdeutlichung sei hier nochmal eine globale Ablaufdarstellung:

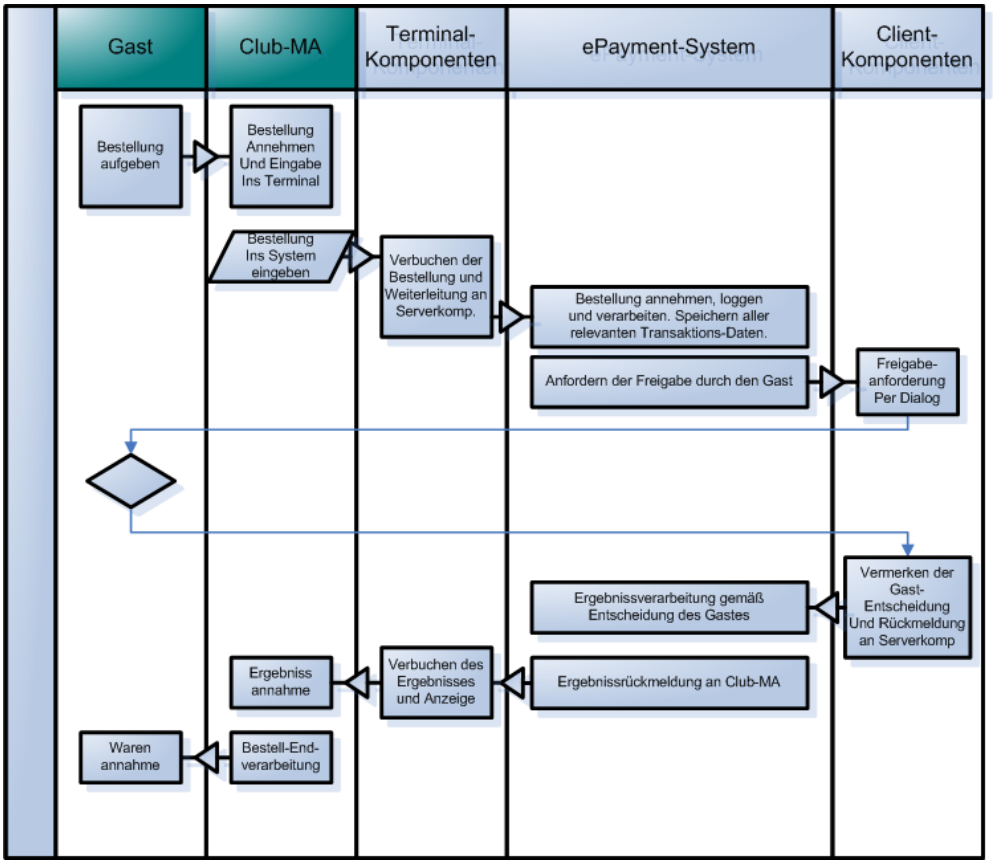


Abbildung 3.3.: Ablaufdiagramm (feinere Darstellung)

3.3. Experimente

3.3.1. Die Hardware

Im Rahmen dieser Arbeit wurden als mobile Endgeräte ein, durch die HAW[1] gestellter, PDA „HP iPAQ Pocket PC H5550“ sowie ein privater Laptop genutzt.

Technische Daten - PDA

- 400 MHz Prozessor mit Intel® XScale™Technologie
- 48 MB ROM; 128 MB RAM
- 3,8-Zoll-Transreflektiv-TFT-Display mit 64.000 Farben
- Integriertes Wireless LAN (802.11b)
- Bluetooth™(V.1.1)
- Biometrisches Fingerabdruck-Lesegerät
- OS: MS® Pocket PC 4.20

Technische Daten - Laptop

- 800 MHz Intel® Celeron® Prozessor
- 128 MB RAM
- wahlweise 10/100 Mbit Ethernet
- oder 54G WLAN (802.11b/g)

Weiterhin wurden mehrere Desktop-PCs mit unterschiedlichen Hardware-Spezifikationen eingesetzt.
Diese Hardwarekomponenten wurden ebenfalls während der Realisierung eingesetzt um die entstehende Software zu testen.

[1] HAW: Hochschule für Angewandte Wissenschaften Hamburg

Als Testnetzwerk für die Analyse sowie die spätere Realisierung wurde ein privates, WLAN-fähiges Netzwerk mit Internetanbindung verwendet. Die Verbindung zwischen WLAN, Ethernet und Internet wurde durch einen WLAN-Router bereitgestellt.
Über die Ethernet-Schnittstelle sind zwei Desktops sowie ein Server an das Netzwerk angeschlossen.

Im WLAN befanden sich überwiegend nur die oben genannten mobilen Endgeräte. Aufgrund der fliessenden Grenzen von WLANs mischten sich teilweise weitere unbekannte und nicht zum Testaufbau gehörige Empfangsgeräte in die Netzwerkkommunikation. Nicht nur aus diesem Grund wurde das Netzwerk als minimum mit einem 128-bit WEP-Schlüssel gesichert.

Da alle WLAN Geräte auch WPA-Verschlüsselung unterstützen sollten, wurde dieses Verfahren als erstes getestet. Leider gab es hierbei diverse unerklärbare Einschränkungen beim Einsatz von beiden mobilen Endgeräten, egal ob im Wechsel oder gleichzeitig.

3.3.2. Die Vorabtests

Bereits bevor mit der Entwicklung der eigentlichen Software begonnen wurde, wurden die ersten Tests durchgeführt.

3.3.2.1. Die „Usability-Tests“

Zu Beginn wurden „Usability-Tests“ durchgeführt. Mit Hilfe dieser Tests sollte festgestellt werden, welche Informationen der Software im allgemeinen über das Terminal übergeben werden sollten und welche Informationen wiederum auf der Clientseite dem Gast dargestellt werden könnten bzw. müssten.

Für diesen Test wurde auf einem im lokalen Netzwerk bereits vorhandenen Webserver eine Webseite bereitgestellt, die die möglichen Informationen im Web-Browser des PDA darstellen konnte. Auf diese Weise wurden mögliche Layouts für die „Push-2-Pay“-Dialoge auf dem PDA entwickelt (siehe nachfolgende Bilder 3.4 und 3.5).

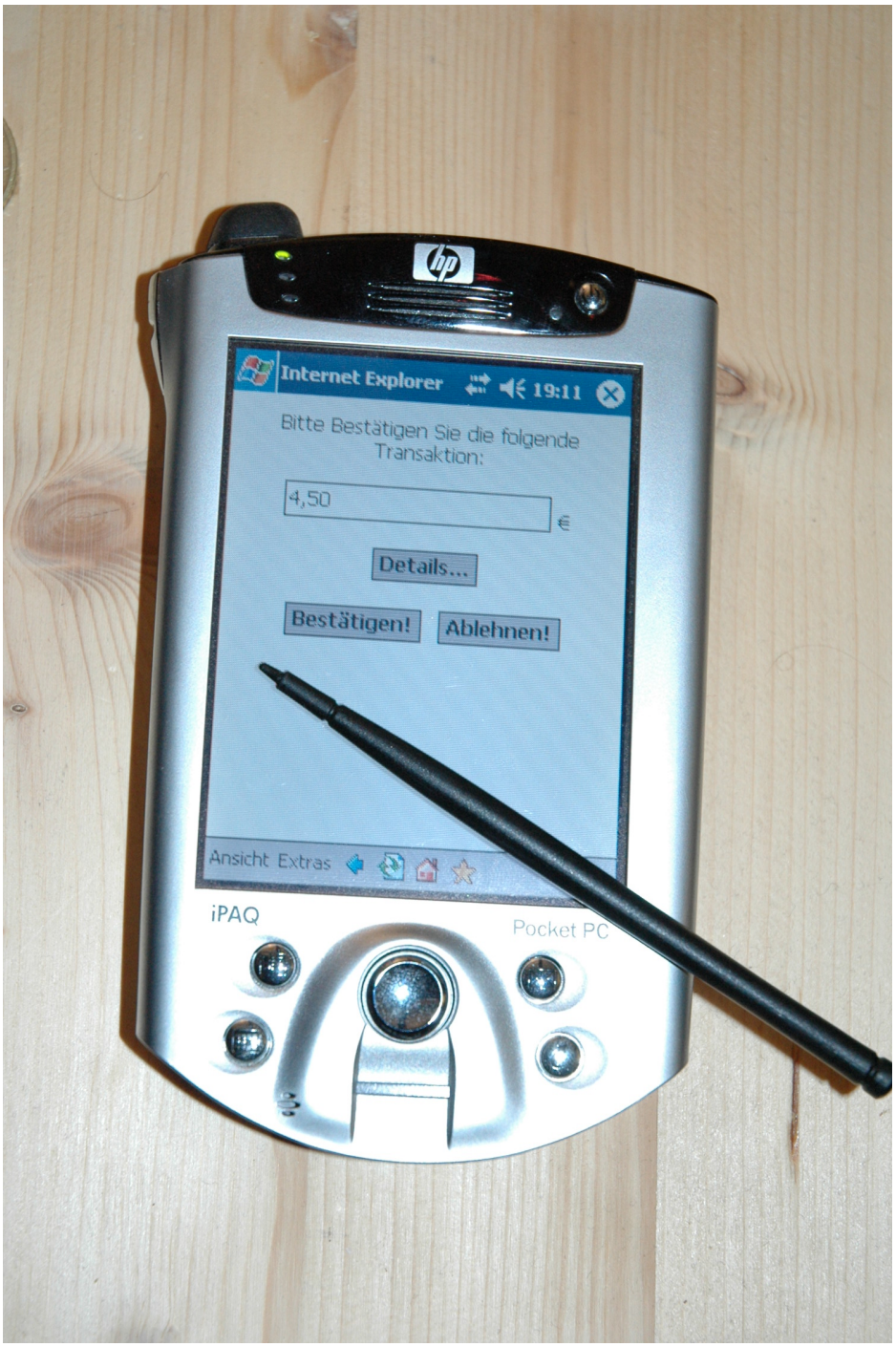


Abbildung 3.4.: Beispiel-Dialog (Referenz - Implementation)

Abbildung 3.5.: Beispiel-Dialog (Referenz - Implementation)

3.3.2.2. Java und seine Schranken

Nachdem die ersten Vorabtests durchgeführt waren, bestand noch die Frage wie es beim verwendeten PDA mit der Java Unterstützung aussieht.

Bereits im Software-Lieferumfang des Gerätes befand sich die Jeode*Runtime* , die eine vollständige Implementation der *Personal Java 1.2 Spezifikation* von Sun ist.

Durch manuelle Fehlerkorrekturen im separat bei Sun herunter ladbaren Swing-Paket der Version 1.1.1 und anschließender erneuter Kompilierung als Java 1.2 Paket ist es möglich dieses Paket in die Jeode*Runtime* einzubinden, womit neben der vorhandenen Unterstützung für AWT ebenfalls eine SWING-Unterstützung gestellt wird.

Zusätzlich wurde CrE-ME V4.00 von NSIcom in einer Evaluation Version getestet.
Leider war hier mit ähnlichen Problemen wie bereits bei der Jeode*Runtime* angesprochen zu rechnen. Zusätzlich gab es noch Probleme bei der Installation und der Verwendung dieser Runtime. Daher wurden weitergehende Tests mit dieser Runtime abgebrochen.

Abgesehen von der nachrüstbaren SWING-Unterstützung bestanden weitere Probleme im fehlen von benötigten Funktionen wie z.B. den Serialisierungsfunktionen (z.B. ObjectStreams) sowie die allgemeinen XML-Fähigkeiten.

Um diese Problematik zu umgehen müssten für diese fehlenden Funktionen entsprechende Workarounds erstellt werden. Wrapper-Klassen, die die entsprechenden Objekte aus den eingehenden „Text“-Nachrichten erstellen. Da die im Rahmen dieser Arbeit zu erstellende Software jedoch nur als „Proof-of-Concept“ implementiert werden soll, wurde bewusst vorerst auf diese Workarounds verzichtet, und der bereits zuvor beschriebene Laptop wurde als mobiles Endgerät eingesetzt.

4. Entwurf

Nachdem nun im vorhergehenden Analyse-Abschnitt die grobe Struktur dieses Systems ausgearbeitet wurde, sollen in diesem Abschnitt die Einzelheiten detaillierter herausgearbeitet werden.

Diese Detaillierung wird in drei Teile aufgeteilt.

Als erstes werden die Hauptkomponenten näher betrachtet und in feinere Strukturen zerlegt. Anschließend wollen wir uns dem Bereich der Kommunikation widmen, um einen ersten Ansatz zu erstellen wie unsere einzelnen Komponenten untereinander Kommunizieren. Abschließend wird noch einmal auf die geplante Erweiterungen, wie z.B. das Token-System oder das eWallet-System, eingegangen.

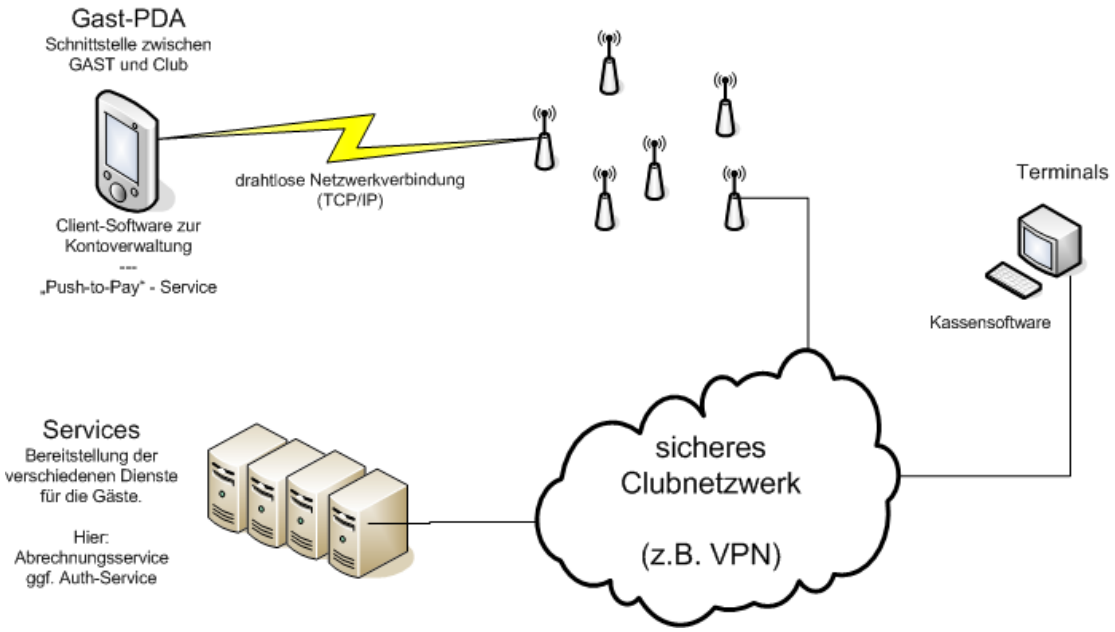
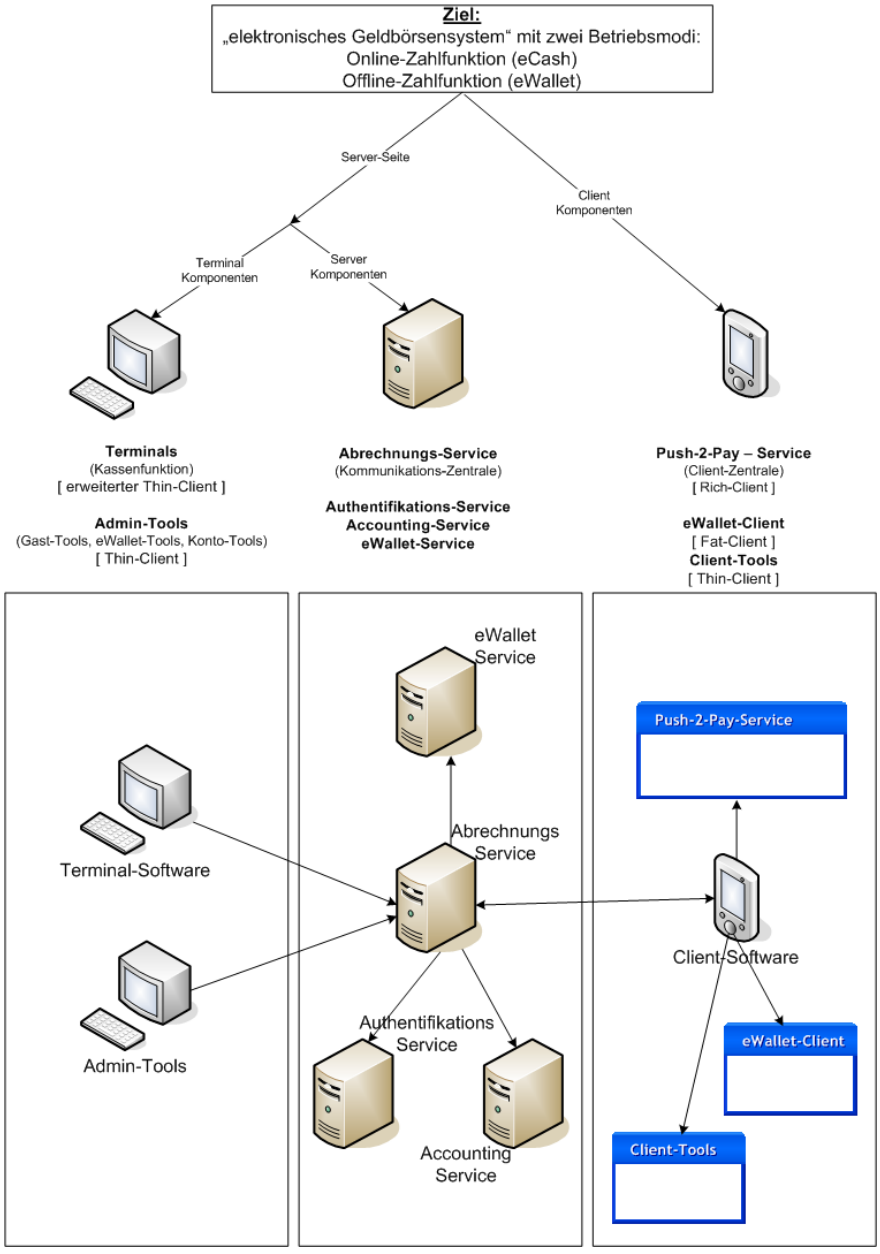


Abbildung 4.1.: Grobansicht des Systems

4.1. Grund-Komponenten



4.1.1. Client-Komponenten

- eWallet (elektronische Geldbörse)
- Client-Tools (Kontoverwaltungssoftware)
- Client - „Push-to-Pay - Service“ (Bestätigungsdialog für Transaktionen)

4.1.1.1. eWallet

Dieser Softwareteil verwaltet das eWallet-Guthaben des Gastes lokal auf dem PDA.
Außerdem stellt die eWallet Schnittstellen bereit, mit deren Hilfe Informationen wie Guthabenstand oder Umsätze von anderer „lokaler“ Software abgerufen werden können.
Zusätzlich soll noch die Möglichkeit gegeben werden, Transaktionen mit Hilfe der eWallet durchzuführen, wenn einmal kein Club-Netzwerk zur Verfügung steht (z.B. Ausfall des Club-Netzwerkes oder Bereiche in denen kein Netzwerk ausgebaut ist).

4.1.1.2. Client-Tools

Die Client-Software soll für den Gast die Möglichkeit bieten auf seine Kontoinformationen zuzugreifen, ohne dass er sich zu einem speziellen Terminal oder zu einem Club-Mitarbeiter begeben muss.
Die bisher vorgesehenen Funktionen sind:

- Abrufen allgemeiner Kontoinformationen
- Abrufen des Kontostandes
- Abrufen der Umsätze
- Guthabentransfer zu anderen Gästen
- Sonstige Verwaltung des Guthabens
  - z.B. Tokensystem

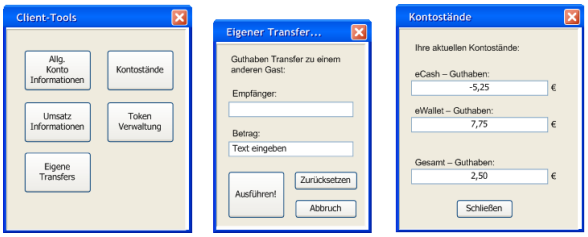


Abbildung 4.2.: Beispiel für die einige Client-Tools

4.1.1.3. Client - „Push-to-Pay - Service“

Die „Push-to-Pay“-Funktion stellt einen kleinen Server auf dem PDA bereit, auf den z. B. der Abrechnungsservice zugreifen kann um sich Transaktionen bestätigen zu lassen.
An dieser Stelle soll vorerst davon ausgegangen werden, dass der Gast in seinen PDA eingeloggt sein muss um diesen Dialog bestätigen zu können. Somit hat er die Authentifizierung bereits am PDA durchgeführt, und damit bestätigt, dass er der „Besitzer“ des Gerätes ist. Daher wird es uns fürs erste genügen, wenn der Gast zur Bestätigung einer Transaktion einen einfachen OK-Button betätigt.
Somit kann für diese Funktion zusammengefasst werden, dass Sie auf einem bestimmten noch festzulegenden Port ins Netzwerk lauscht. Eine eintreffende Nachricht wird entsprechend verarbeitet und ggf. wird die Bestätigung durch den Gast angefordert. Zum Abschluss wird diese Transaktion lokal geloggt und die Antwort an den anfragenden Partner zurück geschickt.
Das zu verwendende Protokoll soll an einer späteren Stelle beschrieben werden.

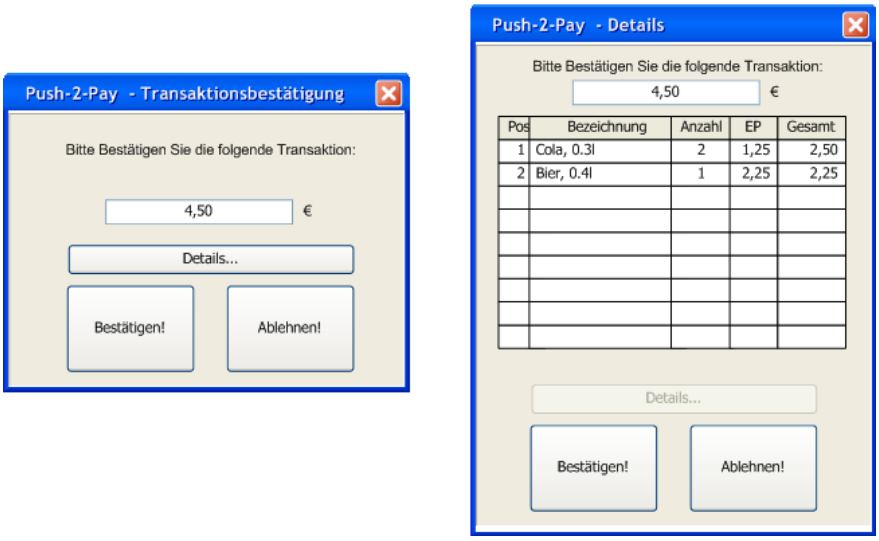


Abbildung 4.3.: Beispiel für den „Push-to-Pay“-Dialog

4.1.2. Server-Komponenten

Zu den Server-Komponenten gehören in diesem Entwurf folgende Komponenten:

- Terminalkomponente
- Abrechnungsservice (Kontroll-Komponente)
- Authentifizierungsservice
- Accountingservice

Ausgehend von diesen Komponenten könnte eine Transaktion wie folgt ablaufen.

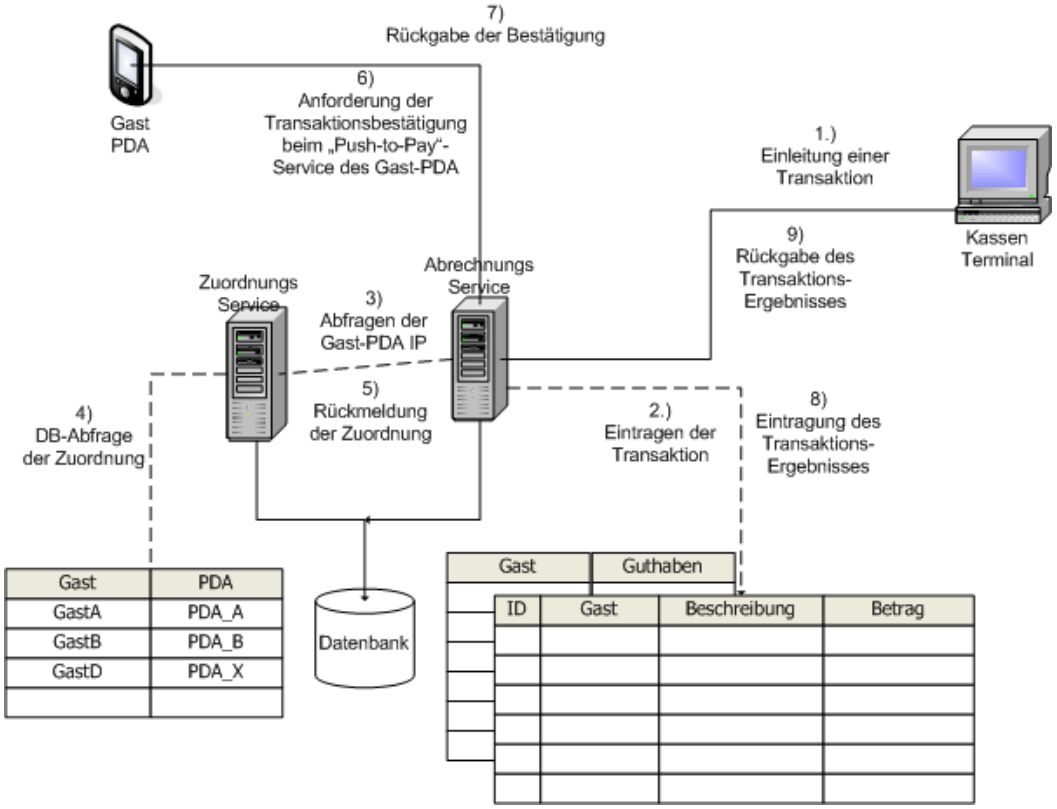


Abbildung 4.4.: Darstellung eines Transaktionsverlaufes

4.1.2.1. Terminalkomponente

Diese Kassensoftware dürfte die am einfachsten zu realisierende Komponente darstellen. Sie soll dem Clubpersonal an entsprechenden Terminals, z. B. an Bars, die Schnittstelle bereitstellen, über die einzelne Bestellungen von Gästen in das System eingegeben werden können.

Vom Abrechnungsservice entgegengenommen, würde die weitere Verarbeitung dieser Transaktionen vom System vorgenommen werden, bis ein Endergebnis an das Terminal zurückgemeldet werden würde.

Die Hauptfunktion dieser Komponente besteht, wie bereits zuvor beschrieben, nur aus einer grafischen Benutzerschnittstelle zum ePayment-System.

Da jedoch zumeist bereits individuelle Kassensoftware-Lösungen in den Clubs eingesetzt werden, kann man evtl. davon ausgehen, dass diese Lösungen zur Verwendung unserer Schnittstelle umgestaltet werden könnten, statt sie neu zu entwickeln.

Daher soll hier nur ein kleiner Prototyp ohne weitergehende Funktionalität entwickelt werden, der wie folgt aussehen könnte:

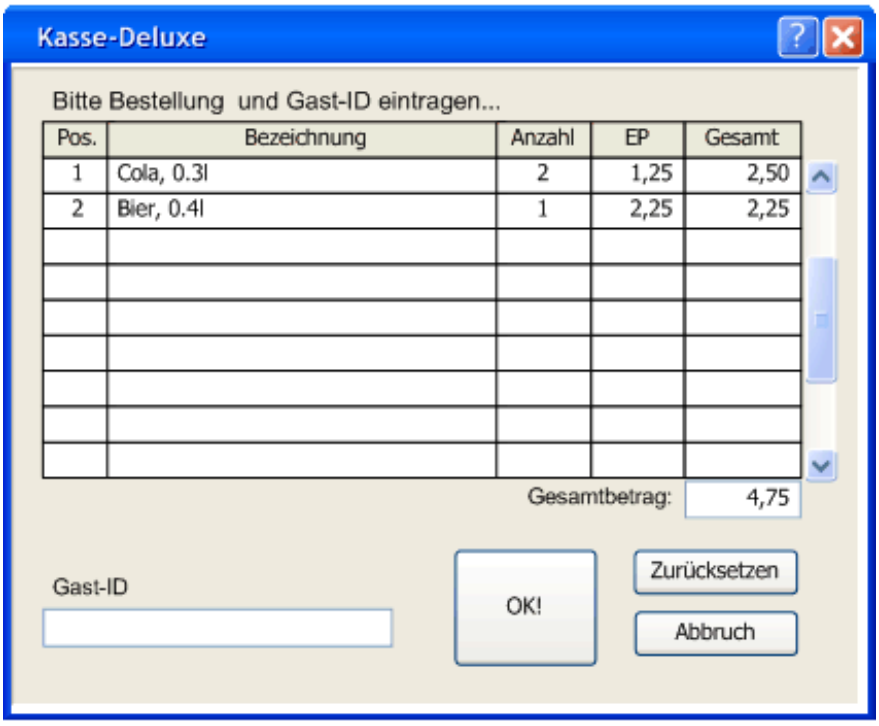


Abbildung 4.5.: Erweiterter Kassen-Dialog

Die Funktionsweise dieser Terminalkomponente sollte wie folgt aussehen:

Nach der Eingabe aller bestellten Waren in der korrekten Anzahl und der Eingabe der GastID des bestellenden Gastes, soll diese Komponente eine Verbindung zum Abrechnungsservice des ePayment-Systems aufbauen. Die entsprechenden Kontaktdaten werden dem Terminal über eine Konfigurationsdatei mitgegeben.
Nach dem Verbindungsaufbau werden die Daten in einer vordefinierten Struktur übertragen und die Verbindung wieder geschlossen.

Sobald der Abrechnungsservice die Transaktion zu Ende verarbeitet hat und ein Ergebnis vorliegt, baut dieser automatisch wieder eine Verbindung zu dem auftraggebenden Terminal auf und übermittelt das Transaktionsergebnis.
Dieses Ergebnis braucht nur noch von der Terminal-Komponente dargestellt zu werden um es dem Bediener mitzuteilen.

4.1.2.2. Abrechnungsservice

Diese Komponente, die hier nur als Abrechnungsservice bezeichnet wird, stellt das Herzstück des Systems dar.

So dient sie im Grunde genommen als „Kommunikationszentrale“, über die sämtliche Kommunikationen des ePayment-Systems geroutet werden.

Sei es im Online-System, mit Verbindung zu allen anderen Services oder im Offline-System, bei dem ein eigenes „lokales“ ePayment-System existiert, welches jedoch ohne die Datenbestände der Gäste und des Clubs auskommen muss.

In allen Fällen werden die eingehenden Nachrichten je nach Nachrichtentyp vom Abrechnungsservice an die entsprechenden Empfänger weitergereicht bzw. vom Abrechnungsservice selbst verarbeitet.

In der Regel wird jedoch der Abrechnungsservice selbst der Empfänger der Nachrichten sein. Je nach Art der Nachricht wird diese also entsprechend verarbeitet und andere Services werden ggf. einbezogen.

Entsprechend der nachfolgenden Darstellung soll der Ablauf einer gewöhnlichen Transaktion wie folgt aussehen:

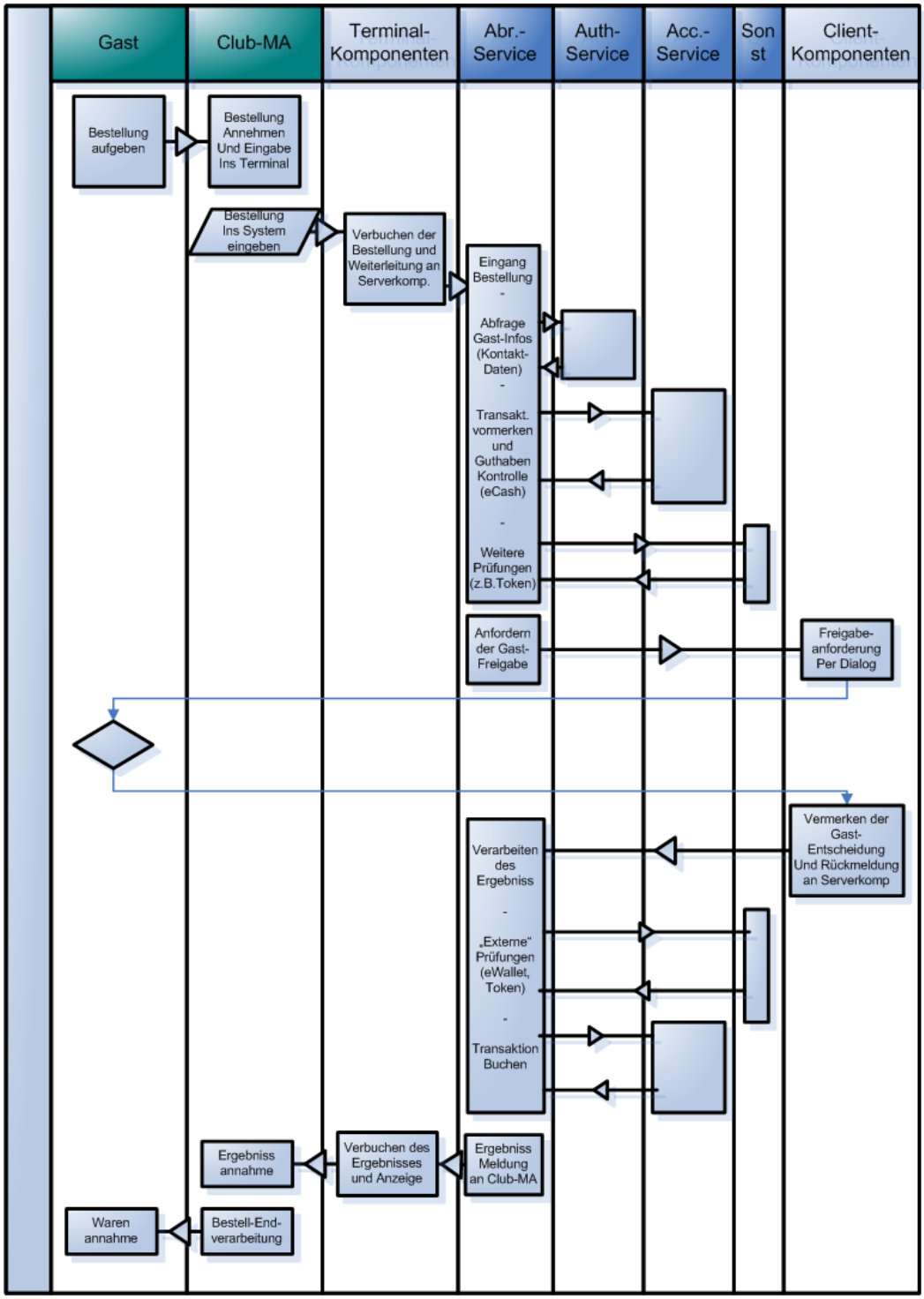


Abbildung 4.6.: Ablauf einer Transaktion

Zu beachten ist hier, dass der Abrechnungsservice die Kontrolle nie vollständig abgeben soll, sondern nur Informationen von anderen Services einholt bzw. an andere Services weitergibt.

Erhält der Abrechnungsservice z. B. eine Bestellung eines Terminals, so könnte er beim Authentifizierungs-Service z. B. die Gast-Konto-IDs sowie die Kontakt-Daten von Gast und Terminal anfordern.

Anschließend sollen die Transaktionsdaten an den Accounting-Service übergeben werden, der diese als „offen“ speichert und daraufhin prüft ob die Transaktion durchführbar ist (genug Guthaben vorhanden) und eine entsprechende Rückmeldung gibt.
An dieser Stelle könnten weitere Services (z. B. ein Token-System) eingebunden werden.
Wenn von Seiten des Abrechnungsservices keine Ablehnung zur Transaktion vorliegt, kann die Freigabe vom Client angefordert werden, womit die Verarbeitung dieser Nachricht beim Abrechnungsservice beendet ist.

Würde jedoch z.B. eine Freigabe von einem Client hereinkommen, so könnte er die Nachricht an „externe“-Services weiterreichen (z.B. einen eWallet-Service, der eWallet-Token des Clients auf Gültigkeit prüft), die ggf. korrigierte Nachrichten zurück geben.

Sollten alle externen Services fehlerfrei durchlaufen sein, so würde die nun gültige Zahlungsnachricht an den Accounting-Service weitergereicht werden, der die entsprechende Transaktion mit dem jeweiligen Ergebnis dieser Nachricht abschließt.

Abschließend meldet der Abrechnungsservice dann das Ergebnis der Transaktion noch an das auftraggebende Terminal zurück, damit der Auftraggeber ebenfalls informiert ist.

4.1.2.3. Authentifizierungs-Service

Da unser Ziel darin besteht, Transaktionen durch den Gast auf dessen eigenen PDA bestätigen zu lassen, jedoch für den Zugriff auf den PDA über ein Netzwerk (z. B. WLAN) eine eindeutige Identifikation (z. B. IP- oder MAC-Adresse) nötig ist, stellt sich an dieser Stelle das erste Problem.
Da es dem Gast nicht zumutbar ist, dass er für jede Transaktion diese eindeutige Identifikationsnummer seines PDA's aufsagt, abgesehen von der Schwierigkeit für den Gast sich diese zu merken, wird dem Gast und seinem PDA eine einfach zu merkende ID, wie z. B. die Zimmernummer, zugeordnet.
Damit nun aber der Abrechnungsservice weiß, über welchen PDA er die Freigabe durch den Gast einholen muss, wird dieser Zuordnungsservice eingerichtet.
Dieser verwaltet nur die Zuordnung von Gast-ID zu PDA-ID, wodurch es für den Abrechnungsservice möglich wird, bei Erhalt einer Transaktionsanforderung bzgl. GastA eine Freigabeanforderung an den PDA von GastA (PDA_A) zu senden.
Für eine einfache Funktion dieses Services sollte sichergestellt sein, dass jeder PDA eine feste IP-Adresse im Clubnetzwerk besitzt. Ob dieses statisch auf dem jeweiligen PDA oder evtl. besser über DHCP erfolgt, soll an dieser Stelle dem jeweiligen Club freigestellt bleiben.
Somit kann ein Mitarbeiter an der Rezeption bei der Personalisierung des PDA für den Gast, die Zuordnung zwischen Gast und PDA bei diesem Service anmelden, woraufhin dieses entsprechend abgespeichert wird.

4.1.2.4. Accounting-Service

Der Accounting-Service stellt die „Bank“ in diesem System dar.
Hier werden die Guthaben-Konten der Gäste sowie das allgemeine Konto des Clubs verwaltet.

So wird von diesem Service für jeden Gast sowie den Ferienclub ein eigenes „virtuelles“ Konto geführt, über welches sämtliche eCash-Transaktionen[1] abgerechnet werden.

Dieser Service bietet dem Abrechnungsservice die Möglichkeit zu prüfen, ob ein Konto über genügend eCash-Guthaben verfügt um eine Transaktion erfolgreich durchzuführen.
Zu beachten ist, dass hier nur über das eCash-Guthaben Auskunft gegeben werden kann, da nur der entsprechende Client über das vorhandene eWallet-Guthaben Kenntnis hat.

[1] Transaktionen die nicht mit Hilfe der eWallet des Gastes abgeschlossen werden

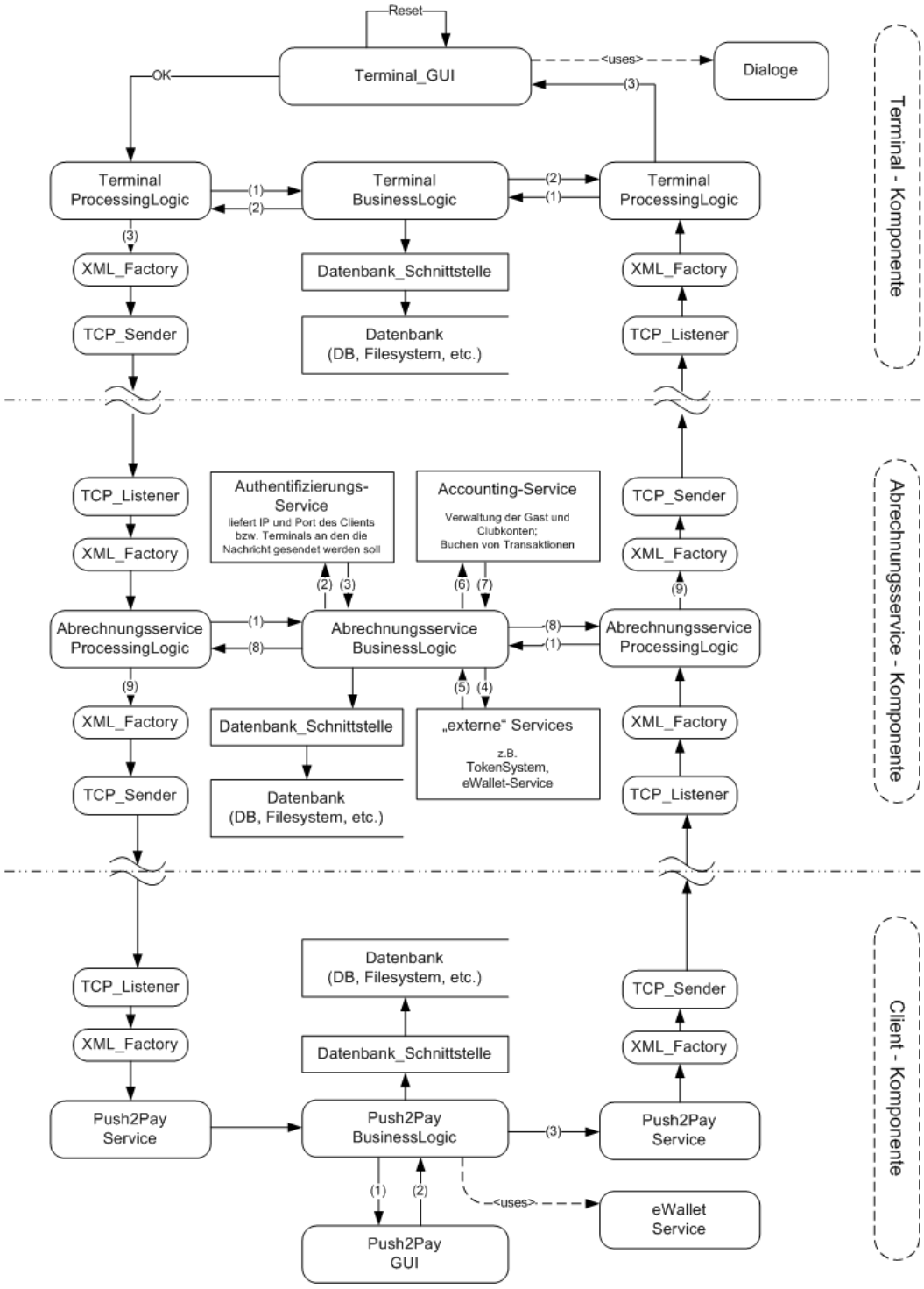


Abbildung 4.7.: Grobdarstellung des geplanten Ablaufs im Rahmen einer Transaktion

4.2. Kommunikation

In diesem Abschnitt werden wir uns im allgemeinen sowie im Detail der Kommunikation in unserem System widmen.

Hierzu werden wir als erstes festlegen, wie die Kommunikation zwischen den einzelnen Teilnehmern auf Netzwerkebene erfolgen soll. Im Anschluss daran wollen wir einen ersten Protokoll-Entwurf für die Kommunikation zwischen den einzelnen Komponenten vorstellen.

4.2.1. Netzwerkkommunikation

Für das gesamte System wollen wir festlegen, dass die einzelnen Komponenten über ein TCP/IP-basiertes Netzwerk kommunizieren sollen. Weiterhin soll festgelegt werden, dass sämtliche Kommunikation, die mit den Transaktionen zusammenhängt per TCP ausgeführt werden soll. Die Verwendung von UDP-Datenpaketen soll jedoch für z. B. Broadcast-Mitteilungen (z. B. für das eWallet-System) möglich bleiben.
Weiterhin soll an dieser Stelle definiert werden, dass die einzelnen Serverkomponenten auf festen Netzwerk-Ports agieren sollen. Die Festlegung auf bestimmte Ports soll jedoch vorerst offen bleiben.

4.2.2. Das Protokoll - Ein erster Entwurf

Für die Kommunikation zwischen den einzelnen Komponenten im Zusammenhang mit Transaktionen soll das folgende Protokoll als erster Entwurf dienen.
Dieser Entwurf soll auch für weitere Kommunikationen zwischen den Komponenten genutzt werden und wird hierzu an den entsprechenden Stellen entsprechend definiert werden.

```
ID: 12345
SENDER: Sender_ID
SENDER_TYPE: TERMINAL | SERVICE | CLIENT
SENDER_IP: 192.168.10.10
SENDER_PORT: 12125

TRANS_ID: 3445379
MESSAGETYPE: BUCHUNG | AUTH_ANFORDERUNG |
             AUTH_ANTWORT | BUCHUNGS_BEST

FROM: Gast_X
TO: CLUB
MESSAGETIME: {timestamp}

MESSAGE_START
  $MESSAGE$
MESSAGE_END
```

Der Message-Bereich ($Message$) wird je nach Messagetyp durch die auf den nachfolgenden Seiten vorgestellten Nachrichtenteileersetzt:

Buchung - Die Übermittlung der Buchungsdaten von einem Terminal oder PDA zum Abrechnungsservice:

```
BETRAG: 5,20
TOKEN_START
 { lfd-Nr : Token-ID : Anzahl : Wert }
 1 : 1253 : 1 : 2,70
 2 : 1327 : 1 : 1,25
 3 : 1327 : 1 : 1,25
TOKEN_END
```

```
PRODUCTS_START
 { lfd-Nr : Product-ID : Bezeichnung : Anzahl
    : Einzelpreis : Gesamtpreis }
 1 : 1753 : Bier, 0.4l : 1 : 2,70 : 2,70
 2 : 233 : Cola, 0.3l : 2 : 1,25 : 2,50
PRODUCTS_END
```

Die Auth-Anforderung wird vom Abrechnungsservice an den PDA des Gastes gesendet. Zu diesem Zweck werden die Daten vom „Auftraggeber“, also Terminal oder PDA, an den PDA des Gastes weitergereicht. Zusätzlich werden der Übertragung noch Guthaben-Informationen bzgl. des eCash-Guthabens des Gastes hinzugefügt. Somit weiß der PDA bzw. der Gast über welches Guthaben er auf seinem eCash-Konto verfügt und kann die Zahlung entsprechend durch die eWallet ergänzen oder komplett von einem Konto bezahlen. In jedem Fall teilt der Abrechnungsservice dem Gast-PDA mit, ob der Gast das Tokensystem aktiviert hat oder nicht, wodurch der PDA weiß, ob er eine Tokenprüfung durchführen muss und entsprechende Token mit der Antwort zurücksenden muss.

```
ECASH_GUTHABEN: -3,20
ECASH_LIMIT: -10,00
ECASH_GESAMT: 6,80
TOKENSYSTEM: 0 | 1
BETRAG: 5,20
TOKEN_START
 { lfd-Nr : Token-ID : Anzahl : Wert }
 1 : 1253 : 1 : 2,70
 2 : 1327 : 1 : 1,25
 3 : 1327 : 1 : 1,25
TOKEN_END
PRODUCTS_START
 { lfd-Nr : Product-ID : Bezeichnung : Anzahl
    : Einzelpreis : Gesamtpreis }
 1 : 1753 : Bier, 0.4l : 1 : 2,70 : 2,70
 2 : 233 : Cola, 0.3l : 2 : 1,25 : 2,50
PRODUCTS_END
```

Die Auth-Antwort - in dieser sendet nun der PDA des Gastes die Nachricht, ob die Transaktion bestätigt oder abgelehnt wurde, sowie einen zugehörigen Informations-Text. Außerdem wird dem Abrechnungsservice mitgeteilt, von welchem Konto wie viel Guthaben verwendet werden soll, um den geforderten Betrag zu begleichen. Abschließend werden dem Abrechnungsservice noch die verwendeten Token übermittelt, die zur Freischaltung der einzelnen Produkte benötigt werden.

```
PAY_LIST
 EWALLET: 5,20
 ECASH: 0,00
PAY_LIST_END
TOKEN_PAY_START
 1 : 1253 : 0 : 2,70
 TOKENSID1
 TOKENSID2
 ...
 TOKENSID27
 2 : 1327 : 1 : 0,00
 TOKENSID30
 3 : 1327 : 1 : 0,00
 TOKENSID31
TOKEN_PAY_END
TRANS_OK: PASSED | FAILED
TRANS_MESSAGE: erfolgreich | abgelehnt
               | fehlendes Guthaben | ...
```

Die Buchungsbestätigung - diese Nachricht stellt die Rückmeldung des Ausführungsstatus an den „Auftraggeber“ dar. Hierbei besteht die Nachricht nur aus einer entsprechenden Erfolgs bzw. Misserfolgs-Nachricht.

```
TRANS_OK: PASSED | FAILED
TRANS_MESSAGE: erfolgreich | abgelehnt
               | fehlendes Guthaben | ...
```

4.3. sonstige Komponenten / Erweiterungen

Viele der in dieses System geplanten Erweiterungen sind mit bereits existierenden Systemen vergleichbar. Da es teilweise einfacher ist, ein System aufgrund eines bestehenden, vergleichbaren Systems zu beschreiben, werden wir in diesem Kapitel die geplanten Komponenten mit Ihren existierenden Verwandten vergleichen.

So werden wir in den folgenden Abschnitten als erstes betrachten, wie wir in unserem System sicherstellen können, dass der Benutzer wirklich der Nutzer ist, als der er sich ausgibt. Anschließend wollen wir noch im Zusammenhang mit dem geplanten Tokensystem auf die mögliche Verwandtschaft mit DRM[2] eingehen.

4.3.1. Zugriffsrechte -> Kerberos

In den heutigen Netzwerken liegen immer mehr vertrauliche Daten.
Diese sind zwar mit Zugriffsbeschränkungen versehen - doch Zweifel bleiben.
Gibt sich jeder als der aus, der er wirklich ist? (Wächtler 1999)

Auch in dem hier entstehenden System liegen vertrauliche Daten bereit, auf die jeder, der mit den richtigen Zugriffsrechten (egal ob er legal oder illegal in den Besitz dieser gekommen ist) ausgestattet ist, zugreifen kann.
Für den ersten Prototypen wollen wir davon ausgehen, dass jeder Netzwerkteilnehmer wirklich ehrlich ist und sich nicht als jemand anderes ausgibt.

Diese Annahme wird sich jedoch sehr schnell als Fehler herausstellen, spätestens beim ersten „offiziellen“ Einsatz wird eine zusätzliche Absicherung des Systems nötig werden.
An dieser Stelle soll eine Möglichkeit betrachtet werden, mit deren Hilfe ein Vertrauensverhältnis zwischen den Kommunikationspartnern in unserem Netzwerk realisiert werden könnte.

[2] DRM - Digital Rights Management (oder ironisch auch als „Digital Restriction Management„ benannt)

4.3.1.1. Kerberos

Kerberos ist ein Protokoll dessen Entwicklung in den achtziger Jahren im Rahmen des Projekt Athena[3] begann.

Auf den Seiten des MIT ist unter „Designing an Authentication System: a Dialogue in Four Scenes“ Bryant (1988) ein Dialog nachzulesen, der einen möglichen Ablauf der Kerberos-Protokoll-Entwicklung beschreibt. In diesem Dialog wird auf den Umstand eingegangen, dass ein vorhandenes „Hochleistungs“ Computersystem zumeist von anderen belegt ist. Um dieses Problem etwas zu mildern, wurde ein verteiltes System geplant, das Projekt Athena.

Bei der Realisierung dieses Systems traten nun allerdings die Probleme auf.
Die User könnten zwar von jedem Zugriffspunkt des Netzes an Ihre Daten kommen, jedoch würden gleichzeitig die Passwörter der User auf jedem Systemen einzeln verwaltet werden. Würde der User also sein Passwort ändern wollen, so müsste er es theoretisch auf jedem einzelnen Systemen ändern.

Um dieses Problem zu umgehen und gleichzeitig mögliche Sicherheitslücken zu beseitigen, wurde entsprechend das Kerberos-Protokoll entwickelt.

Die Grundfunktion dieses Protokolls ist recht simpel.
Grundlagen:

- User und Serverdienste haben jeweils eigene geheime Schlüssel
  (bei Usern abgeleitet aus dem Passwort)
- Passwörter und Schlüssel werden nie als Klartext über das Netzwerk versendet
- Der Kerberos Server (KDC[4]) kennt die geheimen Schlüssel aller User und Services

[3] Projekt Athena: ein Computernetzwerk das in den achtziger Jahren am MIT (Massachusetts Institute of Technology) entwickelt wurde `http://web.mit.edu/acs/athena.html`

[4] KDC : Key Distribution Center

Mit diesen Grundlagen können wir an dieser Stelle vereinfacht beschreiben, wie das Kerberos-Protokoll funktioniert.

1. Der Client fordert ein „Ticket“ vom KDC an, hierzu teilt er dem KDC mit, wie seine Benutzerkennung lautet und mit welchem Server er Kontakt aufnehmen möchte. Zusätzlich wird der Anforderung ein Einmalstempel (Nonce) hinzugefügt.
   `{ Nutzerkennung, Zielserver, Nonce_1 }`
2. Der KDC erzeugt ein Ticket welches folgende Daten enthält:
   `{ Nutzerkennung, Zielserver,`
   `Gültigkeitszeitraum (t1, t2), Sitzungsschlüssel }`
   Dieses Ticket verschlüsselt der KDC mit dem geheimen Schlüssel des Zielservers und packt das so verschlüsselte Ticket in die Antwortnachricht an den Client. Zusätzlich enthält die Antwort noch den geheimen Sitzungsschlüssel und den vom Client in der Anfrage gesendeten Einmalstempel.
   Diese gesamte Antwort wird abschließend noch mit dem geheimen Schlüssel des Nutzers verschlüsselt und dann zurückgesendet.
   `{ Sitzungsschlüssel, Nonce_1,`
   `{Ticket(Nutzer, Zielserver)}Zielserverschlüssel`
   `}Nutzerschlüssel`
3. Nun kann der Client seine eigentliche Anfrage an den Zielserver senden.
   Hierzu sendet er das mit dem Schlüssel des Zielservers verschlüsselte Ticket, eine neue Authentifizierungsnachricht, welche er mit dem Sitzungsschlüssel verschlüsselt sowie eine Dienstanforderung und einen neuen Nonce an den Zielserver.
   `{Ticket(Nutzer, Zielserver)}Zielserverschlüssel,`
   `{Nutzerkennung, Zeitstempel}Sitzungsschlüssel,`
   `Dienstanforderung, Nonce_2`
4. Der Zielserver wird nun das Ticket mit seinem eigenen Schlüssel entschlüsseln und dessen Gültigkeit prüfen. (Ist er der richtige Empfänger und ist das Ticket noch gültig?)
   Mit dem im Ticket enthaltenen Sitzungsschlüssel kann der Zielserver nun die Authentifizierungsnachricht des Client entschlüsseln und den darin enthaltenen Nutzernamen mit dem im Ticket hinterlegten Nutzernamen abgleichen, wodurch bestätigt wird, daß der Nutzer wirklich der Nutzer ist.
   Um nun dem Client gegenüber zu bestätigen, dass er auch wirklich der richtige Server ist und über den nötigen Schlüssel für das Ticket verfügt, sendet der Zielserver abschließend noch den Nonce, den er vom Client mit der Dienstanforderung erhalten hatte, mit dem Sitzungsschlüssel verschlüsselt an den Client zurück.
5. Nun haben sich beide Systeme gegenseitig authentifiziert!

Von diesem Moment bestehen bei diesem System nur noch zwei Sicherheitslücken.

Da zum Entschlüsseln der Antwort des KDC das Passwort des Nutzers benötigt wird, muss dieses vom Nutzer für jedes angeforderte Ticket eingegeben werden, sofern es nicht im Speicher des Client gehalten werden soll, was eine weitere Sicherheitslücke darstellen würde.

Um dieses Problem zu lösen, wurde der KDC um einen zusätzlichen Service, den „Ticket Granting Service“ (TGS) erweitert.
Somit fordert der Client von nun an beim Authentifizierungsdienst des KDC (AS) ein Ticket für den TGS an.

Die Besonderheit im Zusammenhang mit der Erweiterung des KDC um den TGS, besteht darin, dass alle Ticket-Anforderungen des Clients statt mit dem Passwort des Benutzers nun mit dem Sitzungsschlüssel aus dem Ticket des Authentifizierungsdienstes (AS) verschlüsselt sind. Somit braucht das Passwort vom Benutzer nur einmalig für die Entschlüsselung der Antwort des AS bei einer Ticketanforderung eingegeben werden.
Von dem Moment an wird das entschlüsselte und zeitlich begrenzte Ticket für die Anforderung von Dienst-Tickets beim TGS benutzt.

Jedoch bleibt immer noch eine Sicherheitslücke in diesem System bestehen.

So kann jeder User im Namen eines anderen beim Authentifizierungsdienst ein Ticket für den TGS anfordern und dieses „in aller Ruhe“ versuchen per z. B. Brute-Force zu knacken. Immerhin kennt der Bösewicht ja den in der Antwort enthaltenen Nonce.

Weitergehende Informationen rund um das Thema „Kerberos“ sind z. B. in den folgenden Quellen zu finden: MIT Kerberos, RFC 1510, sowie allgemein im Wikipedia (a)

4.3.1.2. Die Lösung für unser System

Da auch in dem entstehenden System immer ein Vertrauensverhältniss zwischen den beteiligten Komponenten bestehen soll, muss an dieser Stelle überlegt werden, wie uns die Erkenntnisse aus dem Kerberos-Protokoll weiterhelfen könnten.

Zu Beachten ist hierbei, dass der Gast fürs erste keine Authentifizierung gegenüber unserem System vornimmt. Wir gehen zunächst davon aus, dass wer auf dem PDA eingeloggt ist, auch über die nötigen Berechtigungen verfügt.

Mit diesen Grundannahmen könnte nun eine eigene Variante des Kerberos-Protokolls implementiert werden, bei der der Gast bzw. sein PDA zu Beginn seines Clubaufenthaltes ein Ticket-Granting-Ticket (TGT) vom Authentifizierungsdienst erhält, welches nur während seines Clubaufenthaltes gültig ist. Da dieses Ticket nur in Verbindung mit dem richtigen Benutzernamen gültig ist, kann bei geeigneter Wahl des Nutzernamens ein ausreichend starkes Vertrauensverhältnis aufgebaut werden.

Sollte zu einem späteren Zeitpunkt eine Erweiterung erfolgen, bei der sich der Gast in regelmäßigen Abständen oder sogar bei jeder Transaktion dem Abrechnungsservice gegenüber authentifizieren muss, so kann der Gültigkeitszeitraum des TGT entsprechend reduziert werden bzw. kann auf das TGT verzichtet werden, und es muss für jede Inanspruchnahme eines Dienstes eine erneute Authentifizierung erfolgen.

4.3.1.3. Ein eTrust-Protokoll Entwurf

Für das zu entwickelnde Authentifizierungs-Protokoll steht fest, dass es auf dem Kerberos-Protokoll basieren soll. Daher sollen an dieser Stelle fürs Erste einige Anforderungen an unser Protokoll festgelegt werden.

- Ticket-Granting-Ticket (TGT)
  - wird bei unserem Protokoll für den Anfang einmalig bei der Ankunft des Gastes auf dessen PDA gespeichert und ist bis zum Ende seines Aufenthaltes gültig.
  - Gültigkeit des TGT kann zu einem späteren Zeitpunkt auf kürzere Zeiträume begrenzt werden, nach deren Ablauf der Gast sich erneut Authentifizieren muss.
  - Für eine Per-Transaktion-Authentifizierung könnte auf das TGT verzichtet werden. In diesem Fall erfolgt die Authentifizierung für jedes Dienst-Ticket einzeln oder es muss für jedes Dienst-Ticket ein neues TGT angefordert werden.

- Dienst-Ticket (Ticket)
  - Für jede Dienstanforderung muss ein neues Ticket angefordert werden (Einmaltickets)
  - Die Tickets werden nur vom KDC ausgestellt. Zur Entschlüsselung der Nachrichten des KDC wird der TGT-Sitzungsschlüssel aus dem TGT benötigt.
- Authentifizierungsdienst (AS)
  - Der AS stellt dem Nutzer auf Anfrage ein TGT aus, welches der Nutzer mit Hilfe seines geheimen Schlüssels entschlüsseln muss um es weitergehend zu nutzen.
  - Der AS verfügt über die geheimen Schlüssel der Nutzer und Services
  - Alle ausgehenden Nachrichten werden mit dem geheimen Schlüssel des anfordernden Nutzers verschlüsselt.
  - Das TGT wird mit dem geheimen Schlüssel des TGS verschlüsselt.
  - Das TGT sowie die Antwort an den anfordernden Nutzer enthalten einen geheimen Sitzungsschlüssel, mit dessen Hilfe der Client und der TGS sicher kommunizieren können.
- Ticket-Granting-Service (TGS)
  - Der TGS stellt nur dann ein Ticket aus, sofern der anfragende Nutzer ein gültiges TGT vorlegt.
  - Der TGS verfügt über die geheimen Schlüssel aller Nutzer und Services
  - Alle ausgehenden Nachrichten werden mit dem geheimen Sitzungsschlüssel aus dem TGT verschlüsselt.
  - Das Dienst-Ticket wird mit dem geheimen Schlüssel des angeforderten Dienstes verschlüsselt
  - Das Dienst-Ticket sowie die Antwort an den anfordernden Nutzer enthalten einen geheimen Sitzungsschlüssel, mit dessen Hilfe der Client und der Dienst sicher kommunizieren können.
- Dienste
  - Jeder Dienst der eine Authentifizierung benötigt, führt die Ihm übertragenen Aufträge nur dann aus, wenn ein gültiges Ticket vorliegt.

Mit diesen Grundlagen können wir nun einen ersten Entwurf für die Verwendung dieses Sicherheitsprotokolls erstellen.

- Ankunft / Check-IN
  - Hinterlegung des geheimen Schlüssels des Nutzers in der Datenbank des KDC (AS und TGS)
  - Anforderung eines TGT beim AS
  - Speicherung des vom AS bereitgestellten TGT auf dem PDA des Gastes
- eigene Transaktionen
  - vorbereiten der Transaktion auf dem PDA
  - anfordern eines Tickets beim TGS zur Kommunikation mit dem Abrechnungsservice
  - senden der Transaktionsanforderung mit dem Ticket an den Abrechnungsservice
  - warten auf eine Rückmeldung des Abrechnungsservice
- ankommende Transaktionsanforderungen
  - eintreffen einer Transaktionsanforderung vom Abrechnungsservice
  - Verarbeitung der Transaktionsanforderung auf dem PDA
  - Anforderung eines Tickets für die Kommunikation mit dem Abrechnungsservice mit Hilfe des TGT
  - senden der Antwort mit dem Ticket an den Abrechnungsservice.
- Ablauf des Aufenthaltzeitraumes (Abreise)
  - Das TGT verliert seine Gültigkeit, der Gültigkeitszeitraum ist abgelaufen.
  - Der Nutzer erhält Authentifizierungs-Fehlermeldungen beim Versuch Transaktionen durchzuführen.

Weiterhin bleibt zu überlegen, ob bereits der Abrechnungsservice bei der Kontaktaufnahme zum PDA des Gastes ein Ticket beantragen und mitschicken muss.

4.3.2. Tokensystem

Als Erweiterung soll unser System noch ein Tokensystem erhalten.

Dieses Tokensystem soll dem Gast auf Wunsch die Möglichkeit bieten, sein Guthaben zweckgebunden einzuteilen, wodurch sich dem Gast die Möglichkeit bietet, dass er sich selbst Limits setzen kann und somit nur bis zu einem bestimmten Betrag z. B. alkoholische Getränke oder Süßspeisen erwerben kann. Alle über diesen Betrag hinausgehenden Zahlungen würden bereits vom System abgelehnt werden und der Gast wäre gezwungen, die sich selbst gesetzten Limits einzuhalten.

Die Kontrollfunktion des Systems könnte in verschiedenen Stufen realisiert werden:

1. Hinweisend - Es erfolgt nur ein Hinweis beim Überschreiten des gesetzten Limits. Zahlungen werden jedoch weiterhin ausgeführt.
2. Warnend - Es erfolgt ein Hinweise beim Überschreiten des Limits.
   Alle weiteren Zahlungen werden verweigert.
3. Verweigernd - Es erfolgt ein warnender Hinweis beim Unterschreiten eines vordefinierten Restwertes. Zahlungen werden nur noch ausgeführt, sofern genügend Guthaben vorhanden ist. Ein gesetztes Limit wird ignoriert.

Um dieses Tokensystem zu realisieren gibt es zwei Möglichkeiten.

- Guthaben in Form von einzelnen Token mit jeweiliger Zweckbindung
- Guthaben und Token getrennt verwalten.

4.3.2.1. Guthaben in Tokenform

Bei dieser Tokenform stellt jedes Token einen gewissen Betrag dar und ist fest an einen Zweck gebunden.
Beim Aufladen eines Guthabens werden die Token vorerst „zweckfrei“ angelegt und der Gast kann dem Guthaben auf Wunsch zu einem späteren Zeitpunkt einen Zweck zuweisen.

Die Vorteile dieses Systems bestehen darin, dass das Guthaben immer direkt Zweckgebunden ist und zu jeder Zeit genügend Guthaben für einen bestimmten Zweck vorhanden ist.
Als Nachteil bei diesem System muss jedoch aufgeführt werden, dass die einzelnen Konten (eWallet und eCash) jeweils separat mit Token versorgt werden müssen, d. h. der Gast muss immer bestimmen welche Token auf welchem Konto liegen.
Sollte der Gast bestimmte Token auf seinem eCash-Konto hinterlegt haben, so könnte er nicht an Terminals zahlen, welche nicht direkt am Clubnetzwerk hängen und somit auf eine eWallet-Zahlung bestehen.

4.3.2.2. Guthaben und Token

Im Gegensatz zur ersten Lösung ist dieses Tokensystem nicht für den „manuellen“ Gebrauch verwendbar.
Hierbei verfügt der Gast (bzw. sein PDA) uneingeschränkt über das Guthaben. Zusätzlich kann der Gast jedoch noch das Tokensystem extra aktivieren oder deaktivieren. Solange das Tokensystem deaktiviert ist, genügt das Guthaben um Zahlungen zu tätigen.
Sollte das Tokensystem jedoch vom Gast aktiviert werden, so müssen bei allen Transaktionen neben dem Guthaben zusätzlich die entsprechenden Token vorhanden und übertragen werden.
Hierbei gilt: jedes Token kann nur einmal verwendet werden!

4.3.2.3. Definition von Token

Um im weiteren Verlauf des Tokensystem-Entwurfs von bestimmten Voraussetzungen ausgehen zu können, werden wir an dieser Stelle einige Definitionen vornehmen.

Definition:

- Das Tokensystem kann aktiviert oder deaktiviert sein
  - bei deaktiviertem Tokensystem kann das Guthaben ohne vorhanden sein von Token aufgebraucht werden
  - bei aktiviertem Tokensystem müssen bei jeder Transaktion Token mit entsprechendem Wert bzw. in entsprechender Anzahl an den Abrechnungsservice übertragen werden.
- Bei der erstmaligen Aktivierung des Tokensystems durch den Gast, werden dem Gast „General-Token“ von jedem Tokentypen bereitgestellt, d. h. der Gast hat keine Einschränkungen und kann vorerst wie mit deaktiviertem Tokensystem Zahlungen vornehmen.
- Bei Bedarf kann der Gast einzelne Einschränkungen vornehmen. So kann er sich selbst Limits setzen:
  - Mengen-Limits: Nur eine bestimmte Anzahl von Waren einer bestimmten Sorte (z. B. max. 5 alkoholische Getränke)
  - Guthaben-Limits: Nur einen bestimmten Betrag für Waren einer bestimmten Sorte (z. B. max. 5 Euro für alkoholische Getränke)
  - Verbot: Waren einer bestimmten Art sind nicht erlaubt (keine alkoholischen Getränke)
- Jedes Token entspricht genau einem Typ:
  z. B. alkoholische Getränke, Limonaden, Süßspeisen, etc...
- Jedes Token hat genau einen exakten Gegenwert
  - Mengen-Token: Jedes Token ist für genau ein Stück eines Artikels gültig
  - Wert-Token: Jedes Token besitzt genau einen Wert von x Euro (z. B. 0,10 Euro

- Jedes Transaktion fordert:
  - den entsprechenden Betrag vom Guthaben des Gastes
  - sofern der Gast das Tokensystem aktiviert hat:
    die entsprechende Anzahl von Token für die einzelnen Waren
    - entweder Wert-Token
    - oder Mengen-Token
    - oder auch gemischt
- Die Token werden vom Abrechnungsservice ausgegeben.
- Der Abrechnungsservice hat Kenntnis darüber ob das Tokensystem aktiviert ist oder nicht
- Der Abrechnungsservice weiß wie viele Token ein Gast von welcher Sorte besitzt.

Auf Grundlage dieser Definitionen können wir unser Tokensystem nun weiter ausführen. Somit können wir vorgeben, dass bei jeder Transaktion vom Abrechnungsservice mitgeteilt wird, ob Token erforderlich sind und wenn ja, welche Tokentypen in welcher Anzahl bzw. welchem Wert nötig sind.

Nun kann der PDA des Gastes die benötigten Token aus seinen Datenbeständen herausnehmen und diese an den Abrechnungsservice mit der Erfolgsbestätigung übermitteln. Sollten nicht genügend Token vorhanden sein, so würde die Transaktion je nach eingestellter Stufe mit einer Warnmeldung beim Gast durchgeführt oder direkt abgelehnt werden.

Zwar kennt auch der Abrechnungsservice alle Token die ein Gast besitzt und könnte daher die Tokenabrechnung selbsttätig durchführen, jedoch stellt sich hierbei erneut das Problem, dass bei eWallet-Transaktionen außerhalb des Clubnetzes der Gast-PDA ebenfalls selbständig über die Tokenverwendung entscheiden muss und somit in jedem Fall ein Abgleich der Token zwischen Gast-PDA und Abrechnungsservice erfolgen muss.

Da somit in jedem Fall eine „doppelte Buchführung“ von Nöten ist, soll an dieser Stelle definiert werden, dass die letztendliche Verwendungsverantwortung beim PDA des Gastes liegt und der Abrechnungsservice nur für die Prüfung der korrekten Verwendung der Token sowie die globale Bestandsverwaltung dieser zuständig ist.

Zusätzlich sollen die Token noch ein Verfallsdatum erhalten, wodurch die Token eben nur für einen bestimmten Zeitraum gültig sind. Nach Ablauf dieses Zeitraumes verfallen die Token automatisch.
Damit der Gast nicht jeden Tag erneut seine Tokeneinstellungen vornehmen muss, kann eine automatische „Tokengenerierung“ aktiviert werden. In einem solchen Fall werden vom Clubservice automatisch - z. B. täglich - die vom Gast gewünschten Token generiert und auf seinen PDA transferiert.

Neben den vom Gast definierten Token kann aber auch der Club dem Gast einzelne Token zur Verfügung stellen. Über diese „Spezial-Token„ könnten dem Gast z. B. täglich eine halbe Stunde Surfen oder wöchentlich eine Stunde Tennis zur Verfügung gestellt werden.

In regelmäßigen Abständen würde das Clubsystem die Tokenbestände mit dem Gast-PDA abgleichen um evtl. Missbrauch oder Fehlfunktionen aufzudecken.

4.3.2.4. Die DRM-Komponente

Dieses Tokensystem könnte man leicht mit den heute aktuellen DRM-Systemen vergleichen, wobei in unserem Fall die zweite ironische Übersetzung von DRM - Digital Restriction Management - besser passen dürfte.
Wie bei echten DRM-Systemen werden die einzelnen Token hierbei an einen Gast gebunden und können nur von diesem genutzt werden, sofern eine direkte Verbindung zum Clubnetzwerk besteht.
Einzige Einschränkung jedoch bleibt, dass dieses System nur dann richtig funktioniert, solange eine Verbindung zum Clubnetzwerk besteht oder eine andere autorisierte Quelle Zugriff auf die Daten hat, um sicherzustellen, dass die benötigten Änderungen wirklich vorgenommen werden.
In dem Fall, dass dementsprechend keine Verbindung zum Clubnetzwerk besteht und nur über die eWallet-Funktion Zahlungen geleistet werden sollen, besteht eine erhebliche Missbrauchsmöglichkeit durch einen manipulierten Client.

4.3.3. eWallet - System

Bisher sind wir bei der Entwicklung dieses Systems nur auf die mit dem Clubnetzwerk verbundene Version eingegangen.
Jedoch wollen wir zusätzlich noch die Möglichkeit bieten, dass ein Gast auch an Terminals Zahlungen bzw. Transaktionen tätigen kann, die nicht direkt ans Clubnetzwerk angeschlossen sind.

Als erstes ist hier zu überlegen, wie die einzelnen Teilnehmer in diesen kleinen „Privat-Netzen“ voneinander erfahren.

Als einfachste Lösung soll in diesen Netzwerken ein eigenständiger, angepasster Abrechnungsservice zur Verwendung kommen.
Dieser Service könnte über einen kleinen integrierten Accounting-Service, eine spezielle Version des Authentifizierungs-Service sowie einen eWallet-Service verfügen.

Der Accounting-Service dieses Systems würde dann nur zwei Konten verwalten. Ein Club-Konto sowie ein eWallet-Konto.

Der Authentifizierungs-Service könnte bei einer entsprechenden Anforderung eine Broadcast-Nachricht ins Netzwerk aussenden und nach dem entsprechenden Client fragen. Auf diese Weise könnten beide Seiten (Server und Client) erfahren, wie die jeweils andere Seite erreichbar ist.

Jedoch stellt sich trotzdem ein Problem. Die Sicherheit und Authentizität des Guthabens, also die Frage ob der Gast-PDA wirklich vorhandenes Guthaben zur Zahlung übermittelt oder nur behauptet, dass er über dieses Guthaben verfügt.
Es ist zwar möglich dieses zu gegebenen Abrechnungszeitpunkten, z. B. Schichtwechsel oder jeden Abend einmal, zu prüfen, jedoch sollte trotzdem nicht erst die Möglichkeit geboten werden, dass ein Gast durch einen manipulierten PDA ungerechtfertigt Transaktionen durchführt und sich so Waren oder Dienstleistungen erschleicht.
Daher muss an dieser Stelle eine Absicherung gefunden werden, mit deren Hilfe sichergestellt werden kann dass:

1. das zur Zahlung übermittelte Guthaben (bzw. auch die Token) echt ist
2. der Gast-PDA die Token nicht einfach behält und z. B. nur die „Transaktion vergisst“ und bei der nächsten Transaktion erneut die selben Guthaben-Token verbraucht.

Eine zusätzliche Absicherung des eCash-Guthabens ist nicht nötig, da die Verwaltung dieses Guthabens vollständig vom Accounting-Service durchgeführt wird und der Client nur Mitteilungen liefert, wie dieses Guthaben verwendet werden soll.

Beim eWallet hingegen ist nur der Gast-PDA ständig über das Guthaben exakt informiert und kann allen Kommunikationspartnern ein bestimmtes Guthaben vorspielen.

Da die eWallet-Services in der Regel keinen direkten Zugriff auf die eWallet-Clients haben, sondern nur über Nachrichten kommunizieren können, entsteht das Problem, dass keine absolut sichere Überprüfung der Software erfolgen kann, und somit davon ausgegangen werden muss, dass eine ordnungsgemäße, unmanipulierte Software eingesetzt wird.

Daher ist es von besonderer Bedeutung in regelmäßigen Abständen Abgleiche zwischen den eWallet-Services in den Offline-Bereichen sowie dem offiziellen Haupt-eWallet-Service des Clubs durchzuführen um Fehler bzw. evtl. Betrugsversuche zu entdecken.

5. Realisierung

Im Rahmen dieser Diplomarbeit soll ein kleiner Prototyp des hier vorgestellten Systems als „Proof of Concept“ implementiert werden.

Alle Komponenten dieses Prototyps werden in „Java“ implementiert, um eine möglichst starke Plattformunabhängigkeit zu erreichen. Aufgrund der geplanten Nutzung der Software auf unterschiedlichsten Zielplattformen bot sich nur „Java“ an.
Da es zur Zeit für beinahe jede Plattform eine „Java virtuelle Maschine“ gibt, die mehr oder weniger leicht bzw. kostengünstig nachgerüstet werden kann, konnte die Entscheidung für den Prototyp relativ leicht gefällt werden.

Für eine spätere Realisierung des kompletten Systems könnten die einzelnen Komponenten auch in unterschiedlichen Sprachen realisiert werden.

Trotz der Festlegung auf „Java“ bleibt noch zu erwähnen, dass es zur Zeit leider noch nicht ohne weiteres möglich ist dieses System auf einer absolut realitätsnahen Basis zu entwickeln.

Leider beruhen die meisten angebotenen Java virtuellen Maschinen für mobile Endgeräte noch auf frühen Java Versionen (zumeist 1.1 oder 1.2 selten auch 1.3). Daher fehlen diesen JVMs viele benötigte Funktionen, die somit nach gebaut oder aus späteren Versionen extrahiert werden müssten (siehe auch 3.3.2.2, S.21).

Da die zuvor beschriebenen Einschränkungen in der Regel nur für die Software-Komponenten auf den mobilen Endgeräten gelten, muss die Client-Komponenten an diese Anforderungen angepasst werden.

Bei allen anderen Geräten im System (Terminals und Server) kann, aufgrund der weniger angespannten Speicher und Prozessor-Ausstattung, eine aktuelle JVM eingesetzt werden, wodurch alle benötigten Funktionen vorhanden und verwendbar sind.

Da dieser Prototyp nur die Machbarkeit dieses Systems belegen soll, wurde auf den Einsatz von mobilen Endgeräten, für die keine aktuelle JVM erhältlich war, verzichtet und statt dessen ein per WLAN mit dem Netzwerk verbundener Laptop verwendet.
Die Terminal- und Service-Komponenten wurden auf herkömmlichen Desktop-PCs ausgeführt.

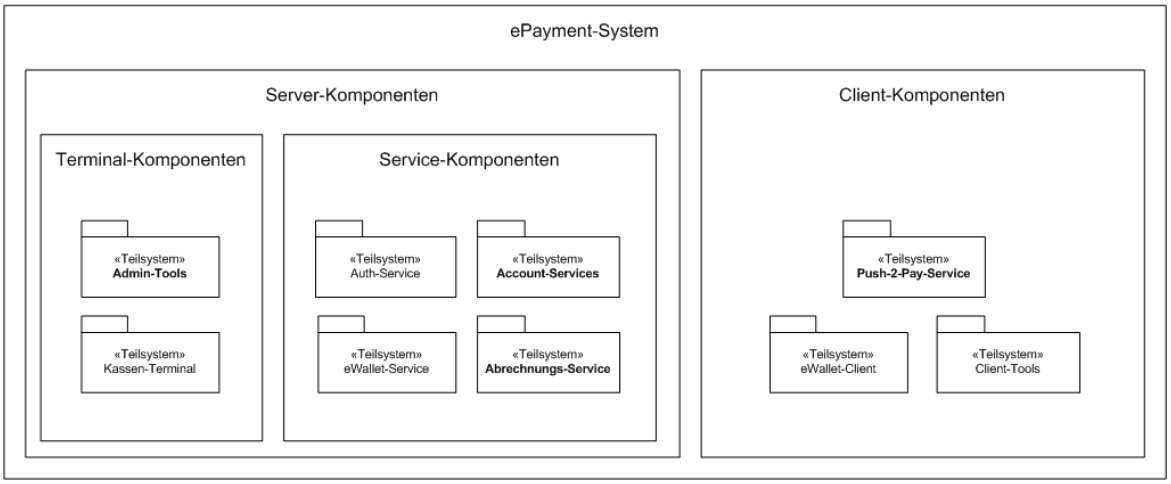


Abbildung 5.1.: Übersicht über das ePayment-System und seine Komponenten

Im Rahmen der „Proof-of-Concept“-Implementierung wurden die in der vorstehenden Grafik dargestellten Komponenten implementiert.

Daher soll in diesem Kapitel beschrieben werden, welche Techniken im Rahmen dieser Implementierung eingesetzt wurden und weshalb diese Techniken zum Einsatz kamen. Ebenfalls soll beschrieben werden, welche Schwierigkeiten während dieser Implementierungen aufgetreten sind.

5.1. Grundlagen

5.1.1. Kommunikation

Zur Kommunikation zwischen den beteiligten Komponenten wird eine TCP-Verbindung genutzt.

Hierzu erhält jede Komponente einen eigenen Listener, der als eigenständiger Thread auf eingehende Nachrichten wartet und diese dann an die Komponente weiterreicht. Erst in der Hauptkomponente wird die Nachricht decodiert und verarbeitet.

Der hier verwendete Listener erfordert die Initialisierung mit einem Objekt, welches das Interface „ListenerInterface“ und somit auch die folgende Methode implementiert:

```
public void newIncommingTransmission
      (InputStream in, OutputStream out);
```

Um selber Nachrichten an die anderen Komponenten schicken zu können, verwendet jede Komponente noch einen eigenen TCP-Sender, welcher die benötigten Funktionen zur Verfügung stellt, um die nötige Verbindung zum anderen Teilnehmer herzustellen und die Nachricht zu übertragen.

Hierzu sei erwähnt, dass der TCP-Sender nur die Kontrolle über die Verbindung bereitstellt. Die sendende Komponente ist selbst für die eigentliche Kommunikation über den Nachrichtenkanal verantwortlich.

5.1.2. Die Realisierung der Nachrichten

Um den Nachrichtenaustausch für's erste einfach zu halten sind für jeden Nachrichtentyp eigene Nachrichten-Objekte vorhanden. Diese werden mit Hilfe einer Wrapper-Klasse in ein XML-Format serialisiert.

Wie im Abschnitt „Das Protokoll“ (Kapitel 4.2.2) geschildert, besteht die Kommunikation aus einer Hauptnachricht, welche bei jedem Nachrichtentyp die selbe Struktur hat, sowie einer darin enthaltenen Nachricht. Diese besitzt je nach Nachrichtentyp eine eigene Struktur (siehe Kapitel 4.2.2, S.34).

Durch die Möglichkeit Objekte zu serialisieren, können die entsprechenden Nachrichten-Objekte vom Sender entsprechend befüllt und in serialisierter Form an den Empfänger gesendet werden. Dort wird die Nachricht wieder zu einem Nachrichten-Objekt deserialisiert und kann wie ein lokal erstelltes Nachrichten-Objekt verwendet werden.

Für die Objekt-(De-)Serialisierung wurden im Rahmen der Entwicklung zwei Komponenten getestet, welche die Objekte in eine XML-Form serialisieren.

Warum XML?
Die Extensible Markup Language (XML) ist eine Ableitung von SGML[1].

XML bietet die Möglichkeit die Struktur des Inhaltes über eine „Grammatik“ (DTD oder XSD) zu definieren, was jedoch im Rahmen der prototypischen Implementation vernachlässigt werden kann.
Aufgrund der immer stärker werdenden Verbreitung von XML-Dokumenten im Bereich des Datentransfers wie auch der Datenspeicherung, bietet sich die Verwendung dieses Datenformates an. Da XML plattformunabhängig und lizenzfrei ist, sollte ein Protokoll im XML-Format leichter in bestehende Software implementiert werden können als eine Protokoll in einem rein proprietäre Format.

Weiterführende Quellen zum Thema XML sind z. B. Wikipedia (b), XML Core Working Group Public Page, W3C sowie XML in 10 Punkten

[1] SGML (Standard Generalized Markup Language) eine Metasprache zum definieren verschiedener Auszeichnungssprachen (ISO 8879:1986)

Als erstes wurde der in Java enthaltene XMLEncode bzw. XMLDecoder verwendet.
Dieser Java eigene XMLEncoder generiert einerseits XML-Dokumente mit einem Header bzgl. der verwendeten XML-Version sowie dem verwendeten Encoding, jedoch werden die Nachrichten gleichzeitig durch eine extreme Verschachtelung stark aufgebläht. Zusätzlich können nur „serialisierbare“ Objekte mit dieser Komponente serialisiert werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_02" class="java.beans.XMLDecoder">
 <object class="de.jabbago.ePay.communication.messages.MainMessage">
  <void property="from">
   <string>123</string>
  </void>
  <void property="id">
   <long>1</long>
  </void>
  <void property="sender_id">
   <string>TestTerminal</string>
  </void>
  <void property="messagetime">
   <object class="java.util.GregorianCalendar">
    <void property="time">
     <object class="java.util.Date">
      <long>1121787304093</long>
     </object>
    </void>
   </object>
  </void>
  .
  .
  .
 </object>
</java>
```

Wie aus dem vorgehenden „kurzgefassten“ Ausschnitt einer Nachricht ersichtlich wird, werden Objekte von der Klasse java.beans.XMLEncoder derart serialisiert, dass die entstehende Nachricht als Wurzelknoten ein Element mit dem Namen „java“ erhält. Darin enthalten sind die serialisierten Objekte.
Die Felder des serialisierten Objektes werden durch ein Element mit Namen „void“ und dem Attribut „property“, dem der Name des jeweiligen Feldes zugewiesen wird, dargestellt.
Der Inhalt des Objekt-Feldes selbst wird als Inhalt dieses Elementes dargestellt, wobei diese

Darstellung wiederum in der Form erfolgt, dass ein Element dargestellt wird, welches die Bezeichnung des Datentyps des Inhaltes als Namen erhält. Erst als Inhalt dieses Elementes wird der eigentliche Inhalt des Feldes dargestellt.

Als zweite Variante zur Generierung von XML aus Objekten wurde dann XStream getestet. Diese Bibliothek bietet eine Serialisierung ähnlich der des Java-eigenen XML-Encoders an. Vorteile dieser Bibliothek bestehen jedoch darin, dass hierbei auch Objekte serialisiert werden können, die nicht dafür vorgesehen sind, sowie die Möglichkeit Klassendefinitionen mit kürzeren Aliasen zu versehen, wodurch eine lange Klassendefinition z. B. durch einen kürzeren, exakteren Alias ersetzt werden kann (statt „java.util.Vector“, einfach und kurz „Zeiger“). Zum Vergleich daher an dieser Stelle die selbe Nachricht wie zuvor bereits mit dem Java-eigenen XML-Encoder erstellt, hier noch einmal in der XStream-Version:

```
<de.jabbago.ePay.communication.messages.MainMessage>
  <from>123</from>
  <id>1</id>
  <sender__id>TestTerminal</sender__id>
  <messagetime>
    <time>1121786637859</time>
  </messagetime>
  .
  .
  .
</de.jabbago.ePay.communication.messages.MainMessage>
```

Wie eindeutig erkennbar enthält diese Version vergleichbare Informationen, jedoch wurde auf die explizite Darstellung der Feldtypen verzichtet. Aufgrund dieser Einfachheit der erzeugten Daten wurde dieses Paket für die weitere Entwicklung dem gesamten Projekt über eine entsprechende Wrapper-Klasse bereitgestellt. Dieser Wrapper erzeugt aus einem übergebenen Objekt mit Hilfe der XStream-Bibliotheken ein XML-Dokument und schreibt dieses in einen übergebenen Stream, bzw. liest aus einem gegebenen Stream die Daten aus und generiert aus dem empfangenen XML-Dokument wieder das zugehörige Objekt.

5.1.2.1. Die Sicherheit der Nachrichten

Zum Thema Sicherheit im Rahmen dieses Systems, wurden einige Regeln definiert, welche von den einzelnen Nachrichten eingehalten werden müssen.

So wurde festgelegt, dass jede Nachricht, die an bzw. über den Abrechnungs-Service versendet wird, mit dem privaten Schlüssel eines asynchronen Verfahrens (hier RSA) signiert wird, um die Herkunft und Authentizität der Nachricht zu bestätigen.

Weiterhin sollen alle Nachrichten, die besonders sicherungswürdige Inhalte (wie z. B. eWallet-Guthaben oder Token anderer Art) transportieren, mit Hilfe eines Session-Keys (einem Schlüssel eines synchronen Verfahrens, im Prototyp: TripleDES) verschlüsselt werden. Hierbei wird die innere Nachricht verschlüsselt und in Form einer *„SecureMessage“* versendet. Jeder Teilnehmer kann SecureMessages entpacken und die Originale Nachricht wieder herstellen, sofern er über den jeweils gültigen Session-Key verfügt.

Die Clients bzw. Terminals können, bei Bedarf, einen neuen Session-Key vom Abrechnungsservice (bzw. AuthService, der über den Abrechnungsservice erreicht wird) anfordern. Hierzu erzeugen Sie eine „SessionKey-Request“-Nachricht, deren Inhalt ein in XML-Serialisiertes „BasicGastInfo“-Objekt ist, welches mit Hilfe eines nur für diese Nachricht generierten Session-Keys verschlüsselt wird. Zusätzlich wird der Session-Key mit dem allgemein gültigen eigenen privaten Schlüssel sowie dem öffentlichen Schlüssel des Abrechnungsservices verschlüsselt und in die Nachricht eingefügt.
Der Abrechnungsservice kann nun durch auslesen des Nachrichtensenders aus der Hauptnachricht den richtigen Public-Key des Absenders sowie den eigenen Private-Key auf den verschlüsselt übersendeten Nachrichten-Schlüssel anwenden und so den richtigen Schlüssel für die Dekodierung der Gast-Informationen erhalten. Anschließend wird die Gast-Information entschlüsselt und die enthaltenen Daten werden mit den beim Auth-Service hinterlegten Gast-Informationen abgeglichen.
Sofern beide Seiten die gleichen Informationen vorhalten, hat sich der Sender durch die Nachricht authentifiziert, und der Auth-Service generiert eine neue „SessionKey-Answer“-Nachricht, in der der Client bzw. Terminal den neuen Session-Key übermittelt bekommt. Gleichzeitig speichert der Auth-Service den Session-Key in den User-Daten des Gastes ab, für spätere Kommunikationen.
Der Client bzw. Terminal fordern bei einem abgelaufenen Session-Key automatisch einen neuen Session-Key beim Abrechnungsservice (bzw. Auth-Service) an und stellen die eigentliche Nachricht (für die der Session-Key benötigt wird) zurück, bis ein neuer gültiger Session-Key zur Verfügung steht.

Unabhängig von der Nachrichtensicherheit selbst, wird hier, wie bereits angedeutet, angenommen, dass das vorhandene Netzwerk sicher ist, und ein abfangen bzw. mitschneiden der Nachrichten nicht möglich ist.

5.1.3. Transaktionen und Transaktionslisten

Bei dem hier entwickelten Projekt sollen Nachrichten zwischen mehreren beteiligten Systemen ausgetauscht werden. Da sich jede Nachricht auf eine bestimmte Transaktion (z. B. Buchung von Getränken, Tennisplätzen, etc...) bezieht, jedoch nicht in jeder Nachricht unnötigerweise alle Informationen über die betroffene Transaktion mit gesendet werden sollen (Vermeidung von unnötigen Redundanzen), verfügt jede Komponente über eine eigene Transaktionsliste, in der alle Transaktionen, an der die Komponente beteiligt ist, abgelegt und gepflegt werden. Die Transaktionen selber wiederum sind Objekte, welche eine bestimmte Transaktion detailiert beschreiben. So enthält ein Transaktions-Objekt folgende Informationen:

```
long localID
String trans_id
Calendar msg_time
String payer
String payReceiver
Double amount
Vector productList
String status
String status_msg
```

Diese Transaktions-Objekte werden aus den, der Nachricht entnommenen, Informationen erstellt. Der Abrechnungs-Service sowie der Client verwalten in Ihren Transaktions-Objekten zusätzlich noch folgende Informationen:

```
double eCashAmount;
EWalletTokenList eWalletAmount;
```

Die Informationen über „status“, „status_msg“, „eCashAmount“ sowie „eWalletAmount“ werden erst bei Abschluss der Transaktion hinzugefügt.

5.1.4. Konfiguration

Um die Anpassung an unterschiedliche Einsatzsysteme zu erleichtern und zu vermeiden, dass für jedes Einsatzsystem der Code angepasst und die Komponenten erneut kompiliert werden müssen, wurden „Konfigurationsobjekte“ erstellt, deren Daten in entsprechenden Konfigurationsdateien hinterlegt sind.
Jede Komponente besitzt ein eigenes Konfigurationsobjekt, welches neben Komponentenspezifischen Daten ebenfalls allgemeine Konfigurationsobjekte enthält, wie z. B. die beiden folgenden Objekte (hier in XML-serialisierter Form)

```
<netConf>
  <sender__id>P2P_Service</sender__id>
  <sender__type>CLIENT</sender__type>
  <sender__ip>127.0.0.1</sender__ip>
  <sender__port>5000</sender__port>
  <dns__ip>localhost</dns__ip>
  <dns__port>53</dns__port>
  <server__ip>localhost</server__ip>
  <server__port>6000</server__port>
</netConf>

<blConf>
  <confFile>.CConf</confFile>
  <transLogFile>.CTrans</transLogFile>
  <dbClass>
     de.jabbago.ePay.dataConnections.DefaultFileDataStore
  </dbClass>
</blConf>
```

Durch die Verwendung dieser Konfigurationsdateien können auf einfache Weise Konfigurationen angepasst werden.
So kann dem Client beispielsweise die IP und der Port des Abrechnungsservices mitgeteilt werden ohne bei jedem Aufruf entsprechende Parameter übergeben zu müssen.
Zusätzlich wurde so eine Möglichkeit geschaffen, mit der eine Klasse angegeben werden kann, die das Logging der Transaktionen übernimmt.
Zu Beachten bleibt, dass diese Konfigurationsdaten nur die Konfigurationsvorgaben darstellt und diese durch die Software bei Bedarf geändert werden können.

5.2. Komponenten

Bevor nachfolgend die Realisierung der einzelnen Komponenten beschrieben wird, soll an dieser Stelle vorab die Struktur der Komponenten dargestellt werden.

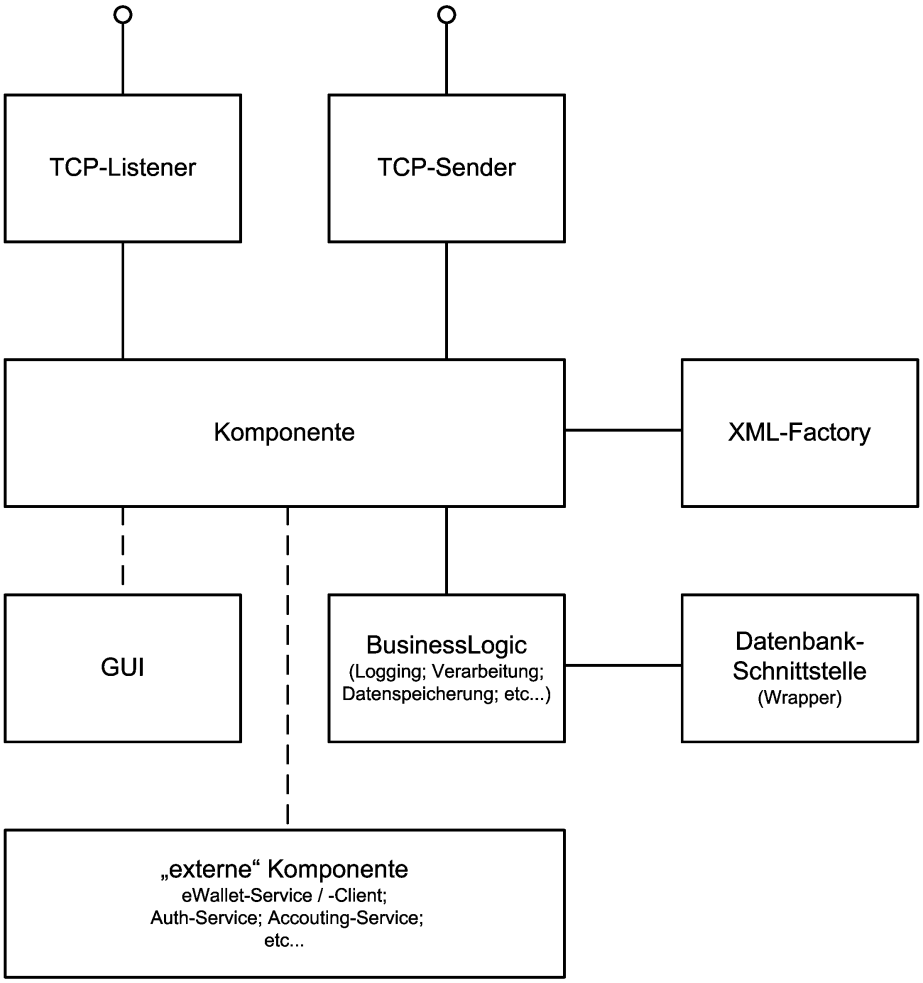


Abbildung 5.2.: Grundaufbau der Komponenten

Diese Grafik 5.2 stellt hauptsächlich die Grundstruktur der drei Hauptkomponenten (Client-, Terminal-Komponente sowie Abrechnungsservice) dar. Alle weiteren Komponenten verzichten in der Regel auf die Implementierung von TCP-Listener und -Sender, da sie direkt an die sie nutzenden Komponenten gebunden werden.

5.2.1. Client-Komponenten

Der Client dieses Projektes besteht aus zwei einzelnen Komponenten. Auf der einen Seite ist ein Verwaltungstool, auf der anderen Seite die wichtigste Komponente auf Client-Seite, der „Push2Pay“-Service.

5.2.1.1. Push-2-Pay - Service

Der Push2Pay-Service besteht aus einer Implementierung, die im Hintergrund läuft und mit Hilfe eines TCP_Listener-Threads auf eingehende Nachrichten wartet.
Unter Verwendung einer Konfigurationsdatei (siehe 5.1.4) wird der Service initialisiert. Um die Informationen über den Gast auch außerhalb des Clubnetzwerkes zur Verfügung zu haben (z. B. bei einem Neustart des Clientsystems), werden diese in einer erweiterten Konfigurationsdatei eingepflegt. Somit stehen dieser Komponente neben der allgemeinen Konfigurationsdaten zusätzlich die Gast-Daten zur Verfügung.

```
<guestID>
  <ownID>123</ownID>
  <name>Szensny</name>
  <callingName>Jan</callingName>
  <roomNr>1725</roomNr>
</guestID>
```

Die hierin enthaltene *ownID* teilt dem Client die GastID des Gastes mit. Somit kann eine eingehende Nachricht ignoriert werden, wenn die GastID der Nachricht nicht mit der GastID des, diesen Client nutzenden, Gastes übereinstimmt.

Grundlegend reagiert der P2P-Service nur auf folgende Nachrichten:

```
- AUTH_ANFORDERUNG
   1) betrifft die Nachricht diesen Gast (GastID)?
   2) vermerken der eingegangenen Transaktion
    3) prüfen ob Tokensystem aktiv
       -> nein: weiter bei 5)
       -> ja: weiter bei 4)
    4) Prüfe ob die angeforderten Token in
       ausreichender Zahl vorhanden sind.
       -> nein: automatisches Ablehnung der Transaktion,
                weiter bei 8)
       -> ja: weiter bei 5)
    5) Prüfe ob der Gast ein eWallet-Guthaben besitzt
       und ob dieses zur Deckung des Betrages reicht
    6) Feststellung der möglichen Kontokombinationen zur Zahlung
       (keine Kontovermischung, Teilzahlungen)
    7) mind. ein Konto verfügt über genügend Guthaben:
       -> nein: automatische Ablehnung der Transaktion
      -> ja: manuelle Bestätigung bzw. Ablehnung durch den Gast
    8) vermerken des Transaktionsergebnisses
    9) erstellt einer AUTH_ANTWORT Nachricht,
       welche das Ergebniss wiedergibt,
   10) Senden der AUTH_ANTWORT an den Abr-Service


- EWALLET_REQUEST
   0) Weitergabe der Nachricht an den eWalletClient
```

Alle anderen Nachrichtentypen werden ignoriert.

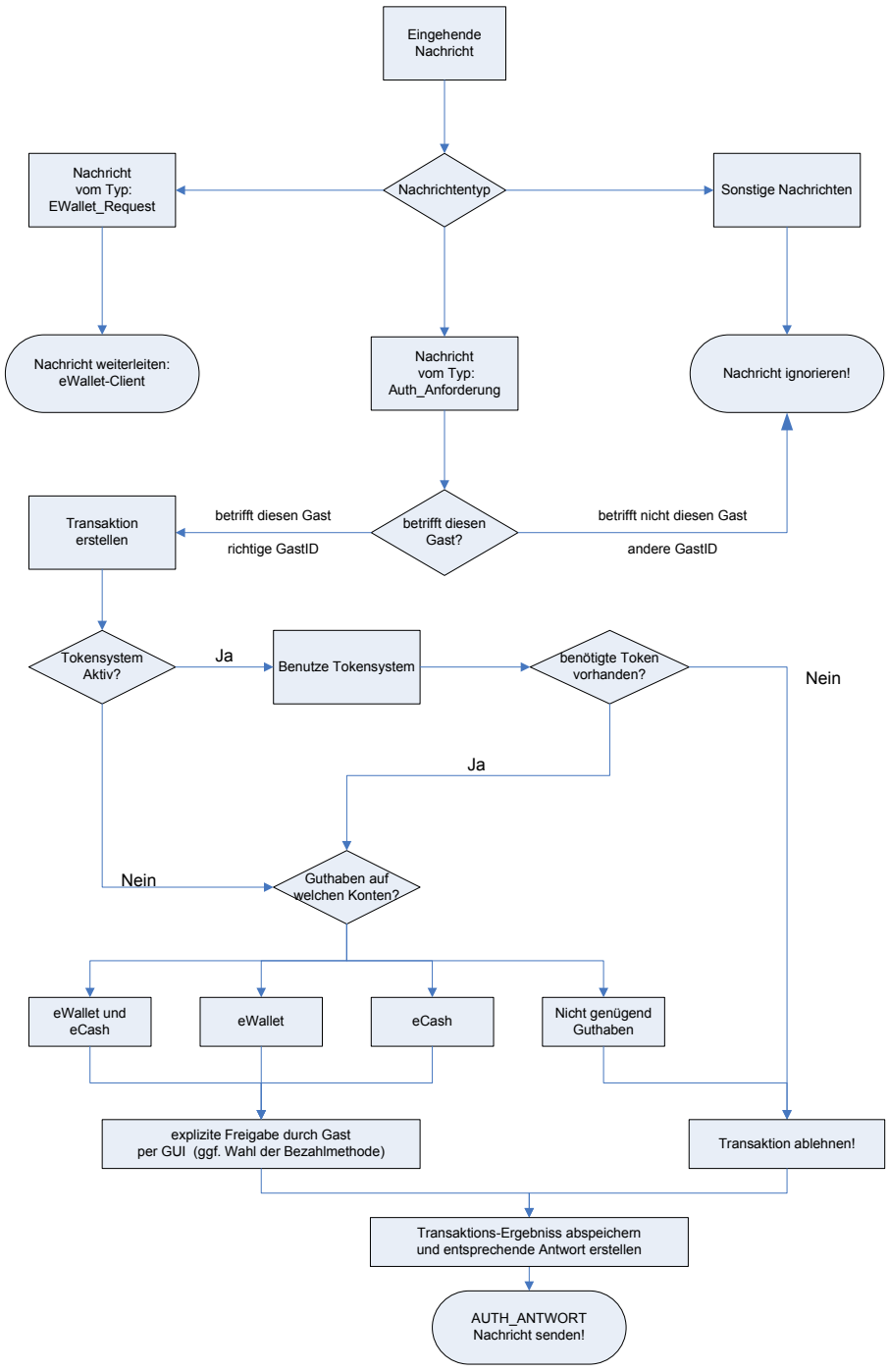


Abbildung 5.3.: Verarbeitungsablauf im Client-System

Wie die anderen Komponenten auch, wird hier ebenfalls die bereits zuvor beschriebene Transaktionsliste (5.1.3) zur Verwaltung der Transaktionen verwendet.

Wie in der Darstellung zur Verarbeitung von Nachrichten (Abb. 5.3) angedeutet, soll diese Komponente einen eWallet-Client nutzen.
Dieser eWallet-Client stellt dem Push2Pay-Service (wie im nachfolgenden Abschnitt ausführlicher beschrieben 5.2.3.2) Funktionen bereit, mit deren Hilfe dieser feststellen kann, über wie viel eWallet-Guthaben der Gast verfügt und sich ggf. eine entsprechende Anzahl von eWallet-Token zurückliefern lassen kann. Diese Token erhält der P2P-Service in Form einer *EWalletTokenList* und bindet diese in die Antwort-Nachricht an den Abrechnungsservice als eWallet-Zahlbetrag ein.

Um die Verarbeitung zu erleichtern, wurde ein spezielles PayBy-Objekt erstellt.
Diesem Objekt werden die folgenden Werte übergeben:

```
double amountToPay      =>  Gesamtbetrag der zu zahlen ist
double eCashTotal       =>  eCash-Guthaben lt. Abrechnungsservice
double eWalletTotal     =>  eWallet-Guthaben lt. eWallet-Client
double eCashPayAmount   =>  Betrag der per eCash gezahlt wird
double eWalletPayAmount =>  Betrag der per eWallet gezahlt wird
```

Zusätzlich zu den Settern und Gettern für die vorgenannten Felder, bietet dieses PayBy-Objekt noch weitere unterstützende Funktionen (siehe A.2), die bei der Überprüfung verschiedenster Parameter hilft.
So bietet ein PayBy-Objekt z. B. die Möglichkeit zu überprüfen, welche Konten (eWallet und/oder eCash) über ein ausreichendes Guthaben zur Begleichung des angeforderten Betrages verfügen.
Weiterhin kann das PayBy-Objekt unter gewissen Umständen auf Anforderung selbständig festlegen, welches Konto zur Zahlung herangezogen werden soll (z. B. wenn nur ein Konto über ausreichend Guthaben verfügt).

In jedem Fall erleichtert dieses Objekt den Austausch von Guthaben- und Bezahlinformationen zwischen den beteiligten Komponenten, da hier sämtliche Informationen in einem Objekt vorgehalten und verwaltet werden können.

5.2.1.2. Client-Tools

Wie bereits zuvor im Entwurf beschrieben (siehe 4.1.1.2) stellt diese Komponente dem Gast eine Schnittstelle zur Verfügung, über die er die im Club sowie auf seinem mobilen Endgerät gespeicherten Daten zu seiner Person und seinen Konten abrufen und verwalten, sowie evtl. sogar eigene Transaktionen (mit anderen Clubgästen) durchführen kann.
Um die Funktion der eigenen Transaktionen für die Clientkomponente zu realisieren, könnte die Implementierung aus der Terminalkomponente herangezogen werden, bzw. es könnte eine einfachere Implementierung für diese Clientkomponente erfolgen, da nur eine entsprechende Nachricht erstellt und versendet werden müsste.
Für die vom Verwaltungstool benötigten Nachrichten existiert an dieser Stelle noch keine eigene Nachrichtendefinition.

Diese Client-Tools wurden im Rahmen des hier erstellten Prototyps nicht implementiert, da sie aufgrund der eher nur darstellenden Funktion nicht zur Grundfunktionalität des gesamten Systems beitragen.

5.2.1.3. Terminalkomponente

Welche Dienste soll ein Terminal bieten?

In erster Linie sollen die einzelnen Buchungsvorgänge, die an einem Terminal stattfinden und über das ePayment-System abgerechnet werden sollen, in einer lokalen Transaktionshistorie vermerkt und die gesamte Transaktion zur Ausführung an den Club-internen Abrechnungsservice weitergeleitet werden. Im Anschluss daran braucht die entsprechende Rückmeldung des Abrechnungsservices, ob eine Transaktion erfolgreich oder misslungen ist, nur noch entsprechend in der Transaktionshistorie vermerkt und dem Bediener des Terminals mitgeteilt werden.

Im Rahmen dieses Projektes wurde entgegen dem Entwurf eine „vollständige“ Terminal-Software entwickelt. Hiermit ist gemeint, dass neben der Transaktionslogik (Transaktionshistorie und -verwaltung sowie dem Nachrichtentransfer) auch eine entsprechende GUI implementiert wurde, welche alle nötigen Funktionen im Rahmen dieses Projektes bereitstellt (Erfassung von Bestellungen und Gast-ID sowie Darstellung diverser Dialoge).

Wie im Entwurf beschrieben, würde für den realen Einsatz die Implementierung einer einfachen Schnittstelle genügen, die in eine externer Kassen-Software implementiert werden könnte. Diese Schnittstelle würde die Bestelldaten der Kassen-Software entgegennehmen, im richtigen Format an den Abrechnungsservice weiterleiten und die von diesem kommende Antwort-Nachricht an die Kassen-Software zurückgeben.

5.2.2. Server-Komponenten

Diese Komponente besteht grundlegend aus drei Teilkomponenten.

1. Abrechnungs-Service
2. Authentifikations-Service
3. Konto-Service

Der Abrechnungs-Service selbst verarbeitet eingehende Anfragen selbstständig unter Verwendung des Authentifizierungs- und Konto-Services.

Spezielle Nachrichten, die nicht durch diesen Server zu verarbeiten sind, werden an die entsprechenden Komponenten bzw. Services weitergeleitet (z. B. eWallet-Nachrichten).

Im Rahmen dieser Entwicklung wurden der Authentifizierungs- sowie der Konto-Service nur als Sub-Komponenten der Server-Komponente implementiert, könnten jedoch im Rahmen einer Weiterentwicklung zu eigenständigen Komponenten mit eigener Schnittstelle entwickelt werden.

5.2.2.1. Abrechnungsservice

Der Abrechnungs-Service stellt die Zentrale Komponente dieses Systems dar. Eingehende Nachrichten die nicht vom Abrechnungs-Service selbst zu verarbeiten sind, werden an die entsprechende Komponente weitergeleitet (z. B. eWallet-Nachrichten).

Standardmäßig werden von dieser Komponente Buchungs- und AuthAntwort-Nachrichten verarbeitet.

```
BUCHUNG:
   vermerken der Transaktion (Konto-Service)
   IP und Port des Zahlers ermitteln (Auth-Service)
   AUTH_ANFORDERUNG - Nachricht erstellen
   und an den Client senden

AUTH_ANTWORT:
   vermerken des Transaktionsergebnisses (Konto-Service)
   IP und Port des Empfängers ermitteln (Auth-Service)
   BUCHUNG_BEST - Nachricht erstellen
   und an den Client senden
```

Buchungsbestätigungs- sowie Auth-Anforderungs-Nachrichten werden in der Regel nur vom Abrechnungsservice selbst erstellt und an die Clients und Terminals versendet.

Daher werden eingehende Nachrichten dieser Art, genau wie Nachrichten, die nicht von der Komponente selbst verarbeitet bzw. weitergeleitet werden, automatisch ignoriert.

5.2.2.2. Authentifizierungs-Service

Der Authentifikations-Service ist die einzige Komponente, die zu jeder Zeit über alle Terminals und Gäste im Netzwerk Kenntnis hat.
So kennt dieser Dienst zu jedem Gast bzw. Terminal folgende Informationen:

- Gast-ID
- Namen (Vorname und Nachname bei Gästen)
- Geburtsdatum (bei Gästen)
- Zimmer-Nummer (bei Gästen)
- Anreise- sowie Abreise-Datum (bei Gästen)
- Account-ID (beim Accounting-Service)
- Netzwerkdaten (IP, Port, Version des Clients)
- Verschlüsselungs-Informationen
- weitere (individuelle) Gast-Informationen

Somit kann dieser Dienst bei einer Anfrage die entsprechenden Daten aus seiner Datenbank auslesen und zurück melden.
Er liefert z. B. Konto-ID bzw. Netzwerk-IP und -Port des PDA oder zusätzliche Informationen, wie z. B. ob der Gast das Tokensystem aktiviert hat.

Dieser Service ist auf Basis des Singleton-Patterns realisiert, wodurch mehrere, inkonsistente Instanzen dieses Dienstes vermieden werden sollen. Zusätzlich verfügt er über keine eigenen Kommunikationsfähigkeiten, wodurch er nur über den Abrechnungsservice, welcher ihn direkt einbindet, kommunizieren kann.

Der Auth-Service generiert auch die Session-Keys, welche von den Clients bei Bedarf angefordert werden. Hierdurch kennen nur der empfangende Client sowie der Auth-Service selbst diesen Session-Key.

Aufgrund des Prototyp-Status dieser Implementierung erfolgt die Speicherung der Daten unverschlüsselt. Bei einer Produktiv-Implementierung sollte die Speicherung dieser Daten jedoch ebenfalls in verschlüsselter Form erfolgen, um ein leichtes ausspionieren der Daten zu vermeiden.

5.2.2.3. Accounting-Service

Der Konto-Service ist die Komponente, welche alle Transaktions-Logs verwaltet und jederzeit über das eCash-Guthaben der Gäste informiert ist.
So muss für jeden Gast ein eCash-Konto eingerichtet werden, wobei die Gast-Informationen des Authentifikations-Services auf dieses Konto verweisen.

Bei jeder neuen Transaktion, die vom Abrechnungsservice eingereicht wird, ermittelt der Konto-Service die Konto-ID des Zahlenden sowie des Zahlungsempfängers mit Hilfe des Authentifizierungs-Services und verbucht die bisher offene Transaktion auf den beteiligten Konten.

Nach dem eine Transaktion bestätigt bzw. abgelehnt wurde, werden die zugehörigen Transaktionen bei den Beteiligten um das Transaktionsergebnis und eine entsprechende Status-Nachricht ergänzt. Zusätzlich wird das Guthaben bei einer erfolgreichen Transaktion entsprechend angepasst.

Dieser Service ist auf Basis des Singleton-Patterns realisiert, wodurch mehrere, inkonsistente Instanzen dieses Dienstes vermieden werden sollen.

Er verfügt über eine Kontenliste, in der alle Konten hinterlegt sind.
Diese Konten selbst wiederum enthalten die folgenden Daten:

```
String accountID => Die ID dieses Accounts
double guthaben     => Das aktuelle Guthaben
double limit        => Das Überziehungslimit
Hashtable transList => Liste aller eigenen Transaktionen
```

Im Zweifelsfall lässt sich der Guthabenbetrag jedes Kontos über die jeweilige Transaktionslisten wiederherstellen.

Da auch dieser Service keine eigene Kommunikations-Schnittstelle besitzt, kann er nur über den ihn einbindenden Service Nachrichten entgegennehmen. Selbst eine Verbindung zum überstehenden Abrechnungsservice kann dieser Service jedoch nicht herstellen.

Genau wie beim Auth-Service sollte bei einer späteren produktiv Realisierung dieser Komponente zwecks Datensicherheit eine verschlüsselte Datenspeicherung implementiert werden.

5.2.3. sonstige Komponenten / Erweiterungen

5.2.3.1. Token-System

Auf Grund der starken Komplexität dieses Bereiches, wurde das Token-System im Rahmen dieses Prototyps nicht implementiert.

Jedoch wurden den Beispiel-Waren in der Terminal-Komponente bereits einige Token zugeteilt. Diese Token werden im Rahmen einer Transaktion bis zur Client-Komponente durch gereicht, bleiben jedoch unberücksichtigt.

5.2.3.2. eWallet-System

Das eWallet-System selbst besteht in der Implementation aus zwei Komponenten.
Auf der einen Seite steht ein eWallet-Service, der eWallet-Token erzeugen und diese an den jeweiligen Client senden kann.
Gleichzeitig ist der eWallet-Service für die Kontrolle von ihm übergebenen eWallet-Token zuständig und weisst den Abrechnungsservice an, ggf. entsprechende Beträge über das Konto des eWallet-Services zu verrechnen.
Hierzu teilt der eWallet-Service dem Abrechnungsservice mit, dass die Transaktion geändert wurde und nicht mehr vom Gast bezahlt würde sondern nun vom eWallet-Service.
Dementsprechend würde der eWallet-Service eine angepasste Nachricht zurückgeben, die, an den Accounting-Service weitergereicht, für die nötigen Korrekturen sorgt.

Auf der anderen Seite befindet sich ein eWallet-Client, der vom Push-2-Pay-Service des Client-Gerätes eingebunden wird und die eWallet-Token des Gastes verwaltet.
Dieser eWallet-Client teilt dem Push-2-Pay-Client auf Anfrage mit, über wie viel eWallet-Guthaben der Gast noch verfügt und gibt ggf. eine entsprechende Anzahl eWallet-Token an den Client aus.
Diese eWallet-Token werden vom Push-2-Pay-Client in der Auth-Antwort-Nachricht an den Abrechnungsservice gesendet.

Der eWallet-Client enthält zwei Listen. Eine Liste beinhaltet alle noch gültigen unverbrauchten eWallet-Token.
Die andere Liste hingegen wird mit den benutzten eWallet-Token befüllt.
Die Liste der verbrauchten Token wird zu dem Zweck geführt, um den Abgleich der verbrauchten Token mit dem eWallet-Service zu erleichtern. Dieser Abgleich wurde aufgrund des Prototyp-Status dieser Implementierung bisher jedoch noch nicht implementiert.

6. Zusammenfassung

Im Verlauf meiner Diplomarbeit habe ich mich mit verschiedenen Themen befasst. So habe ich mich in Bezug auf die Sicherheit mit den Themen Kerberos, DRM sowie WLAN-Sicherheit beschäftigt.

Weiterhin wurden verschiedenste ePayment-Systeme betrachtet, die bereits für den Zahlungsverkehr im Internet Verwendung finden. Zu nennen seien hier neben den bereits erwähnten „Echt-Geld“ Bezahlsystemen - PayPal und moneybookers.com - noch Export Force 2.0, welches die Möglichkeit bietet, Transaktionen mit einer imaginären Internet-Währung namens Klamm-Lose durchzuführen.

Bei der Betrachtung dieser ePayment-Systeme stellte sich heraus, dass sie nur vergleichbar sind mit dem eCash-Teil des hier entwickelten Systems. Die in diesen System verwendeten Sicherheits-Mechanismen sind für das hier geplante ePayment-System nicht ausreichend.

Für das hier zu entwickelnde ePayment-System wurde eine Kombination aus zentral verwaltetem (eCash) sowie lokal verwaltetem (eWallet) Guthaben verwendet.
Für das eWallet-System spricht, dass es über die Fähigkeit verfügt, sowohl Online- als auch Offline-Transaktionen[1] durch zuführen.
Das eCash-System hingegen kann nur Online-Transaktionen durchführen, da hier die Guthaben-Informationen, die zur Durchführung einer Transaktion nötig sind, auf dem zentralen Server hinterlegt und verwaltet sind. Aufgrund dieser zentralen Guthabenverwaltung bietet dieses eCash-System jedoch zusätzlich noch die Möglichkeit, dass den Benutzern individuelle Kredite eingeräumt werden können, da diese hier direkt berücksichtigt und kontrolliert werden können.

Das in dieser Arbeit vorgestellte, aufgrund des Projekt-Umfanges jedoch nicht realisierte, Tokensystem ist unabhängig vom eingesetzten ePayment-System. In diesem System würden die Token jedoch auf dem Client verwaltet werden, da diese somit bei jeder Transaktion berücksichtigt werden könnten.

[1] Online-Transaktion sind Transaktionen, die im Clubnetzwerk ausgeführt werden. Offline-Transaktionen werden außerhalb des Clubnetzwerkes ausgeführt und es ist ein spezielles Terminal nötig.

Ebenfalls seien noch die Überlegungen zum Thema „Implementierung“ erwähnt. Hier stellte sich als erstes die Frage nach der Programmiersprache in der der Prototyp implementiert werden sollte. Da der Prototyp auf möglichst vielen Plattformen in der gleichen Version laufen sollte, ohne für jedes System separate Versionen zu erstellen und zu compilieren, kamen nur Java und Dot.Net für die Implementierung in die engere Wahl.
Da es, von Java wie auch von Dot.Net, virtuelle Maschinen für jede hier verwendete Plattform gibt, bestand auch hier keine Einschränkung beim Einsatz einer bestimmten Sprache. Aufgrund bereits existierender Erfahrungen mit Java, wurde diese Sprache für die Implementierung des Prototyps gewählt.

Wie im Kapitel Analyse (3.3.2.2 S. 21) beschrieben, wurden mehrere „Java virtuellen Maschinen“ auf dem, zur Verfügung gestellten, PDA getestet. Aufgrund der, während der Tests, festgestellten Probleme, wurde auf die Lauffähigkeit der Client-Komponente, auf einem PDA oder einem ähnlichen mobilen Endgerät, verzichtet.

Der im Rahmen dieser Diplomarbeit entwickelte Prototyp stellt nur ein „Proof-of-Concept“ dar. Damit ist gemeint, dass dieser Prototyp nur aufzeigt, dass das hier vorgestellten ePayment-System, in der geschilderten Form, lauffähig ist. Vor einer endgültigen Realisierung dieses Systems sollten jedoch noch weitere Punkte überlegt und Erweiterungen entwickelt werden.

So wäre es eine interessante Aufgabe dieses System an einen Netzwerkweiten LDAP-Server anzubinden, welcher dann als Datenspeicher für den Auth-Service dienen könnte. Hierdurch könnten alle Informationen zu den Netzwerk-Teilnehmern (Gäste, Terminals und Services) an einer zentralen Stelle verwaltet werden.

Als weiterer Entwicklungspunkt stünde die Realisierung der Offline-Funktion wie im Entwurf beschrieben.
Hierzu sollten weitere Möglichkeiten einer Offline-Anbindung ausgearbeitet, sowie mögliche Sicherheitslücken betrachtet und natürlich nach Möglichkeit geschlossen werden.
Zusätzlich bliebe zu klären, wie die korrekte Verarbeitung innerhalb des Clients gewährleistet werden könnte, ohne dass ein direkter Hardwarezugriff durch die Server-Komponenten erfolgt.

Ebenfalls sollte dieses ePayment-System noch um das bereits beschriebene Tokensystem erweitert werden.

Zur Erhöhung der Sicherheit der Daten auf dem PDA sowie der Kommunikation zwischen PDA und Abrechnungsservice, wäre es beim für die Entwicklung bereitgestellten *HP iPAQ h5550* möglich den integrierten Fingerprint-Scanner zum signieren bzw. verschlüsseln der Daten bzw. der Kommunikation zu verwenden. Hierzu könnte eine von HP bereitgestellte BioAPI verwendet werden.

Auch wenn dem Prototyp noch einige Funktionen, die im Rahmen des Konzeptes erarbeitet wurden, fehlen, so ist zumindest bewiesen, dass das Konzept an sich funktionsfähig ist.

Bereits in der aktuellen Fassung könnte dieser Prototyp eingesetzt werden. Beispielsweise an stationären Bezahlterminals in der Kantine eines Unternehmens. Bei geeigneter Auswertung der Kontoinformationen könnten die Zahlbeträge bei der Gehaltsabrechnung verrechnet werden. Oder z. B. auch in der hiesigen Mensa.

Ich bin sicher, dass in einigen Jahren ein vergleichbares System in den ersten Ferienclubs zum Einsatz kommen wird. Da, wie inzwischen fast überall, auch in den Ferienclubs Computersysteme dort vorhanden sind, wo man sie gar nicht bemerkt oder auch nur vermuten würde, dürfte der Einsatz eines ePayment-Systems, wie in dieser Arbeit beschrieben, keinen allzu großen Aufwand mehr bedeuten. Als größter Hinderungsgrund neben den Anschaffungskosten für die benötigten mobilen Endgeräte dürfte zur Zeit der Nutzer selbst gesehen werden.

Der überwiegende Teil der Club-Gäste wird derzeit noch in einem Alter sein, in dem es nicht so leicht fällt, innerhalb kurzer Zeit und auf entspannte Weise, den Umgang mit der nötigen Technik zu erlernen. Club-Gäste aus diesen Generationen sind den Umgang mit dieser Art von ubiquitären Systemen in der Regel nicht gewohnt und der Einsatz würde bei den betroffenen Club-Gästen vermutlich eher negative Auswirkungen auf die eigentlich erwünschte Erholung haben.

Die zur Zeit heranwachsenden Generationen hingegen dürften nur geringe bis gar keine Probleme beim Erlernen der Funktionsweise eines solchen Systems haben, da sie zur Zeit mit der Allgegenwärtigkeit von Computern aufwachsen und daher unbefangener sind im Umgang mit ubiquitären Systemen.

So kann es sein, dass in einigen Jahren die nötige Hardware bereits in die Armbanduhr integriert ist. Also jeder seinen Computer am Handgelenk spazieren trägt. In einem solchen Fall bräuchte die nötige Client-Software, mit den richtigen Konfigurationsdaten versehen, nur noch auf die Armbanduhr aufgespielt zu werden.

Nach dem ich mich im Rahmen dieser Diplomarbeit stark mit vielen möglichen Arten von ePayment-Systemen beschäftigt habe, bin ich zu der Feststellung gekommen, dass neben dem Thema „ubiquitous computing“, welches heute immer mehr in den Vordergrund tritt, in den nächsten Jahren auch das Thema „ePayment“ noch stark an Bedeutung gewinnen wird.

A. Anhang

A.1. Nachrichten - XML-Format

java.beans.XMLEncoder

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_02" class="java.beans.XMLDecoder">
 <object class="de.jabbago.ePay.communication.messages.MainMessage">
  <void property="from">
   <string>123</string>
  </void>
  <void property="id">
   <long>1</long>
  </void>
  <void property="sender_id">
   <string>TestTerminal</string>
  </void>
  <void property="sender_ip">
   <string>127.0.0.1</string>
  </void>
  <void property="sender_port">
   <int>7000</int>
  </void>
  <void property="sender_type">
   <string>TERMINAL</string>
  </void>
  <void property="to">
   <string>CLUB</string>
  </void>
  <void property="trans_id">
   <string>TestTerminal-1121787304-0</string>
  </void>
```

```
  <void property="message_type">
   <string>BUCHUNG</string>
  </void>
  <void property="messagetime">
   <object class="java.util.GregorianCalendar">
    <void property="time">
     <object class="java.util.Date">
      <long>1121787304093</long>
     </object>
    </void>
   </object>
  </void>
  <void property="message">
   . . .
  </void>
 </object>
</java>
```

XStream - XMLEncoder

```
<de.jabbago.ePay.communication.messages.MainMessage>
  <id>1</id>
  <sender__id>TestTerminal</sender__id>
  <sender__type>TERMINAL</sender__type>
  <sender__ip>127.0.0.1</sender__ip>
  <sender__port>7000</sender__port>
  <trans__id>TestTerminal-1121786637-0</trans__id>
  <message__type>BUCHUNG</message__type>
  <from>123</from>
  <to>CLUB</to>
  <messagetime>
    <time>1121786637859</time>
  </messagetime>
  <message
     class="...">
    .
    .
    .
  </message>
</de.jabbago.ePay.communication.messages.MainMessage>
```

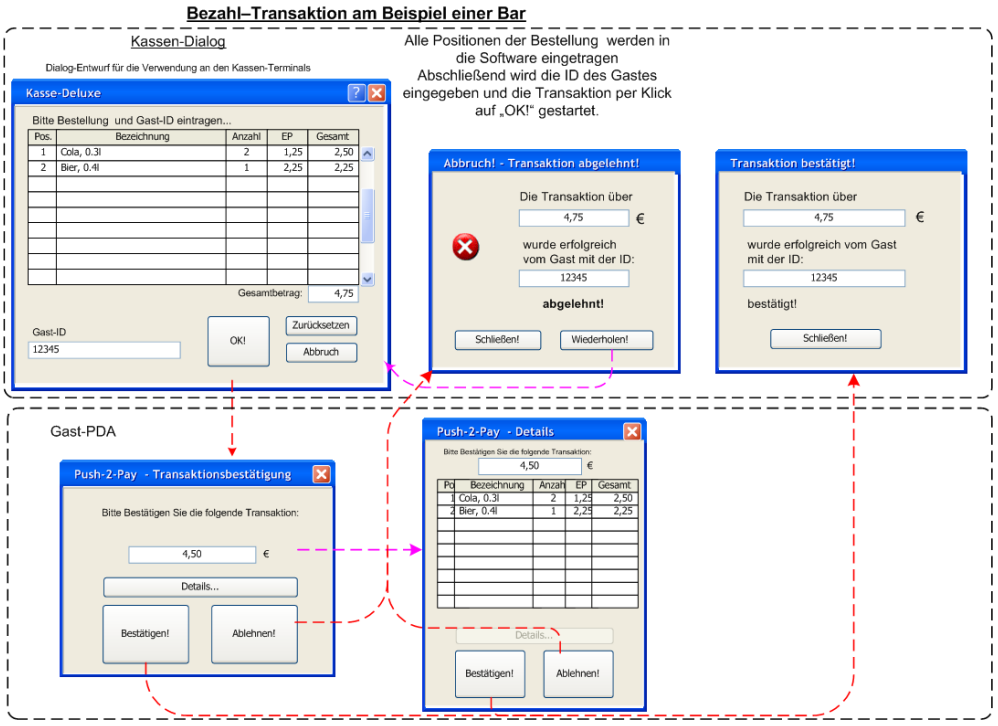
A.2. Spezielle Objekte

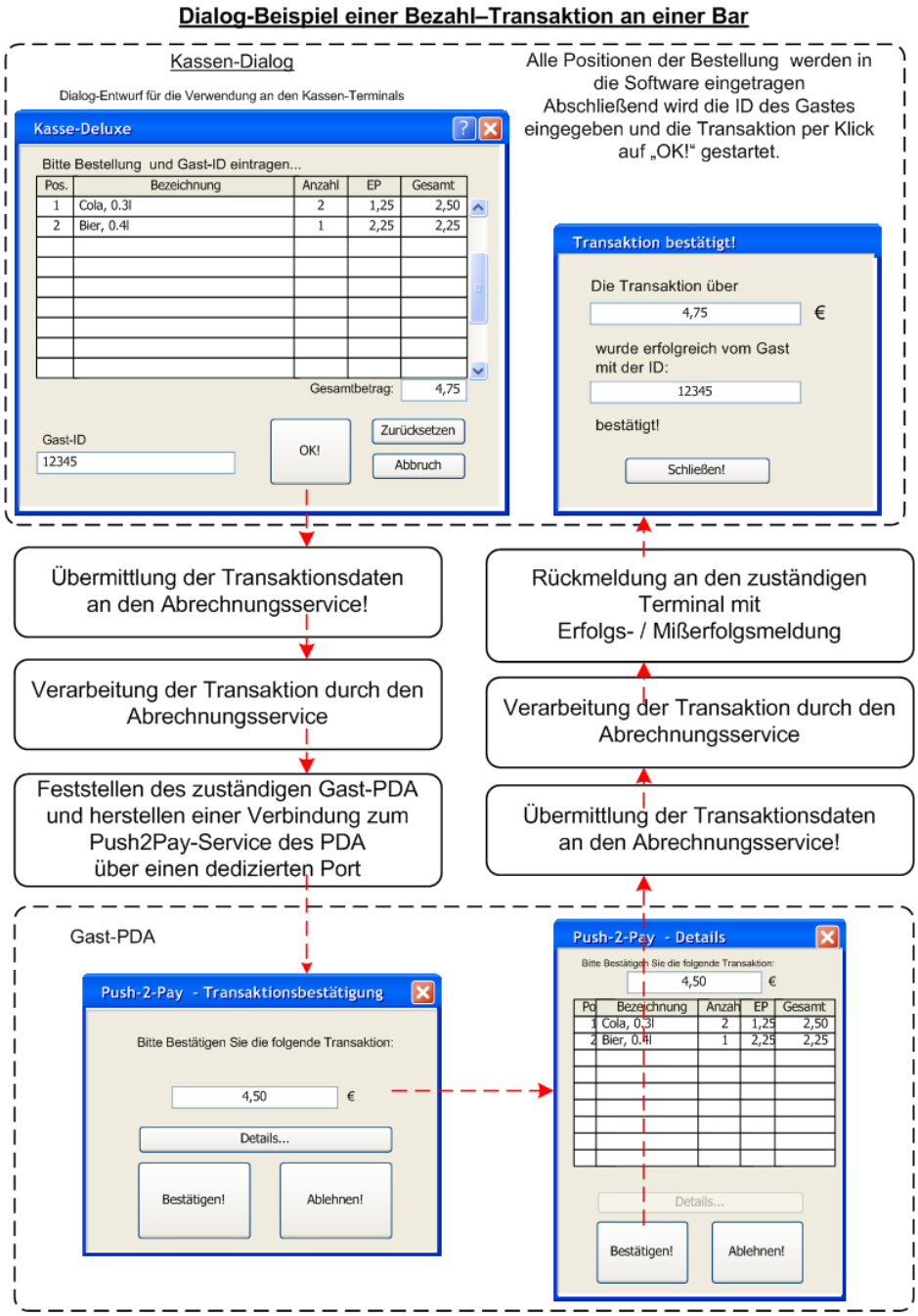
Das PayBy-Objekt

```
public boolean isSetPayAmount()
  * sagt aus ob bei diesem PayBy-Objekt mindestens ein
  * Guthabenbetrag (eWalletTotal oder eCashTotal) gesetzt wurde.


public int acountsWithMoney()
* Liefert einen int-Wert, der aussagt,
* welches Konto über genügend Guthaben verfügt
* um den geforderten Betrag (amountToPay) zu begleichen.
* 0: kein Guthaben
  * 1: eCash
  * 2: eWallet
  * 3: beides


public String acountsWithMoneyToString()
* entspricht der Funktion acountsWithMoney()
* jedoch erfolgt die Darstellung hier in textueller form
  * 0: NoMoney
    * 1: eCash
    * 2: eWallet
  * 3: DUALPAY
  * other: ERROR


public double totalMoney()
* Liefert das Gesamtguthaben,
* welches durch dieses PayBy-Objekt repräsentiert wird.


public double remainingMoneyTotal()
* Liefert den Betrag, der nach Zahlung des gegebenen Betrages
* (amountToPay) noch insgesamt auf den Konten verbleibt
* (eWalletTotal + eCashTotal)
* => Restguthaben nach der Transaktion
```

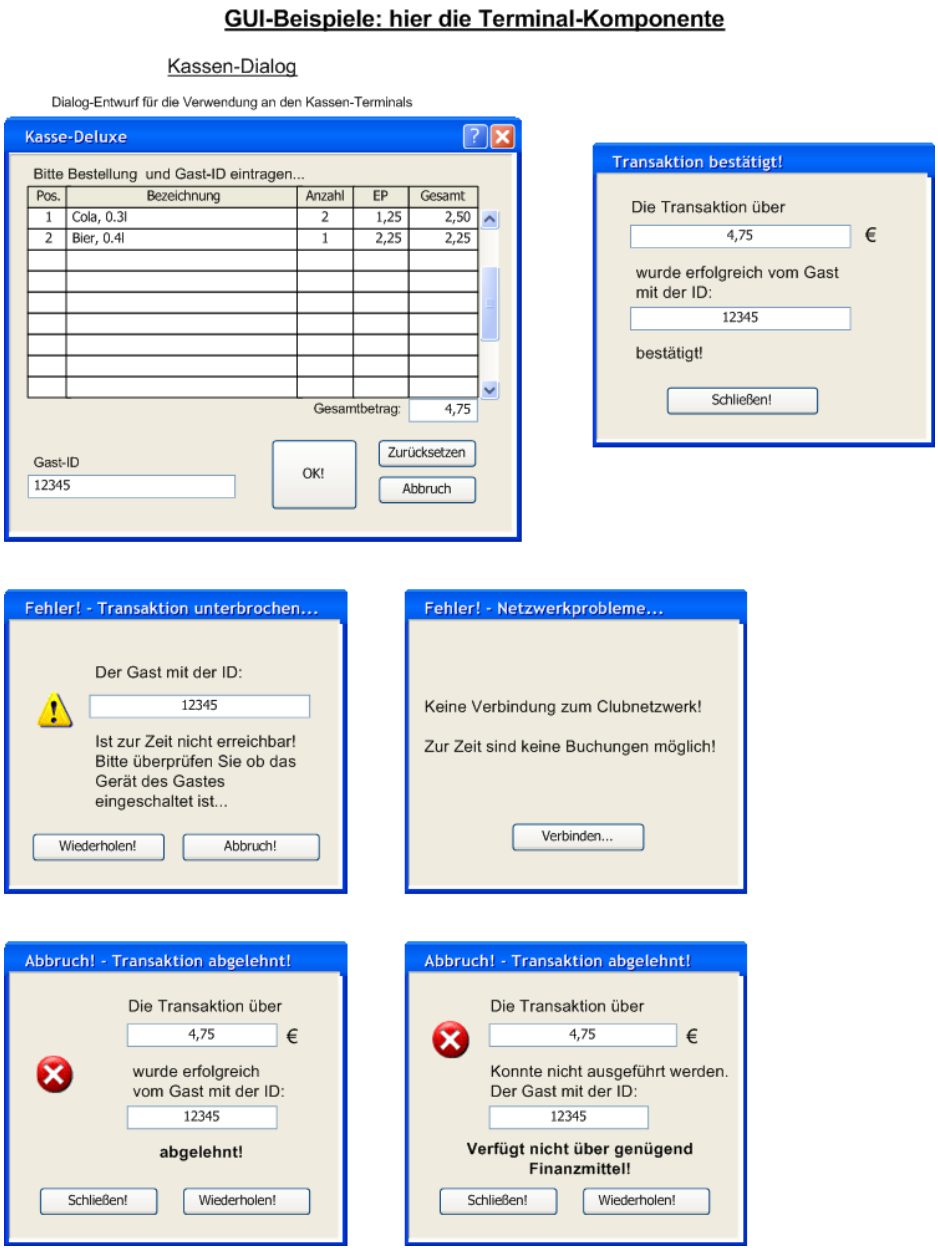
```
public boolean setAutoPayAmount()
* Setzt die Zahlbeträge automatisch, sofern
* nur ein Account über ausreichend Guthaben verfügt
* und gibt ein  true  als Antwort zurück.
* Andernfalls erfolgt keine Änderung und es wird
*  false  als Antwort zurück geliefert.
```

B. GUI - Anhang


Bezahl–Transaktion am Beispiel einer Bar
Kassen-Dialog
Dialog-Entwurf für die Verwendung an den Kassen-Terminals
Kasse-Deluxe
Bitte Bestellung und Gast-ID eintragen...
Pos. Bezeichnung Anzahl EP Gesamt
1 Cola, 0.3l 2 1,25 2,50
2 Bier, 0.4l 1 2,25 2,25
Gesamtbetrag: 4,75
Gast-ID
12345
OK!
Zurücksetzen
Abbruch
Alle Positionen der Bestellung werden in die Software eingetragen
Abschließend wird die ID des Gastes eingegeben und die Transaktion per Klick auf „OK!“ gestartet.
Abbruch! - Transaktion abgelehnt!
Die Transaktion über
4,75 €
wurde erfolgreich vom Gast mit der ID:
12345
abgelehnt!
Schließen!
Wiederholen!
Transaktion bestätigt!
Die Transaktion über
4,75 €
wurde erfolgreich vom Gast mit der ID:
12345
bestätigt!
Schließen!
Gast-PDA
Push-2-Pay - Transaktionsbestätigung
Bitte Bestätigen Sie die folgende Transaktion:
4,50 €
Details...
Bestätigen!
Ablehnen!
Push-2-Pay - Details
Bitte Bestätigen Sie die folgende Transaktion:
4,50 €
Pos. Bezeichnung Anzahl EP Gesamt
1 Cola, 0.3l 2 1,25 2,50
2 Bier, 0.4l 1 2,25 2,25
Details...
Bestätigen!
Ablehnen!


Dialog-Beispiel einer Bezahl–Transaktion an einer Bar
Kassen-Dialog
Dialog-Entwurf für die Verwendung an den Kassen-Terminals
Kasse-Deluxe
Bitte Bestellung und Gast-ID eintragen...
Pos. Bezeichnung Anzahl EP Gesamt
1 Cola, 0.3l 2 1,25 2,50
2 Bier, 0.4l 1 2,25 2,25
Gesamtbetrag: 4,75
Gast-ID
12345
OK!
Zurücksetzen
Abbruch
Alle Positionen der Bestellung werden in die Software eingetragen Abschließend wird die ID des Gastes eingegeben und die Transaktion per Klick auf „OK!“ gestartet.
Transaktion bestätigt!
Die Transaktion über
4,75 €
wurde erfolgreich vom Gast mit der ID:
12345
bestätigt!
Schließen!
Übermittlung der Transaktionsdaten an den Abrechnungsservice!
Verarbeitung der Transaktion durch den Abrechnungsservice
Feststellen des zuständigen Gast-PDA und herstellen einer Verbindung zum Push2Pay-Service des PDA über einen dedizierten Port
Rückmeldung an den zuständigen Terminal mit Erfolgs- / Mißerfolgsmeldung
Verarbeitung der Transaktion durch den Abrechnungsservice
Übermittlung der Transaktionsdaten an den Abrechnungsservice!
Gast-PDA
Push-2-Pay - Transaktionsbestätigung
Bitte Bestätigen Sie die folgende Transaktion:
4,50 €
Details...
Bestätigen!
Ablehnen!
Push-2-Pay - Details
Bitte Bestätigen Sie die folgende Transaktion:
4,50 €
Pos. Bezeichnung Anzahl EP Gesamt
1 Cola, 0.3l 2 1,25 2,50
2 Bier, 0.4l 1 2,25 2,25
Details...
Bestätigen!
Ablehnen!

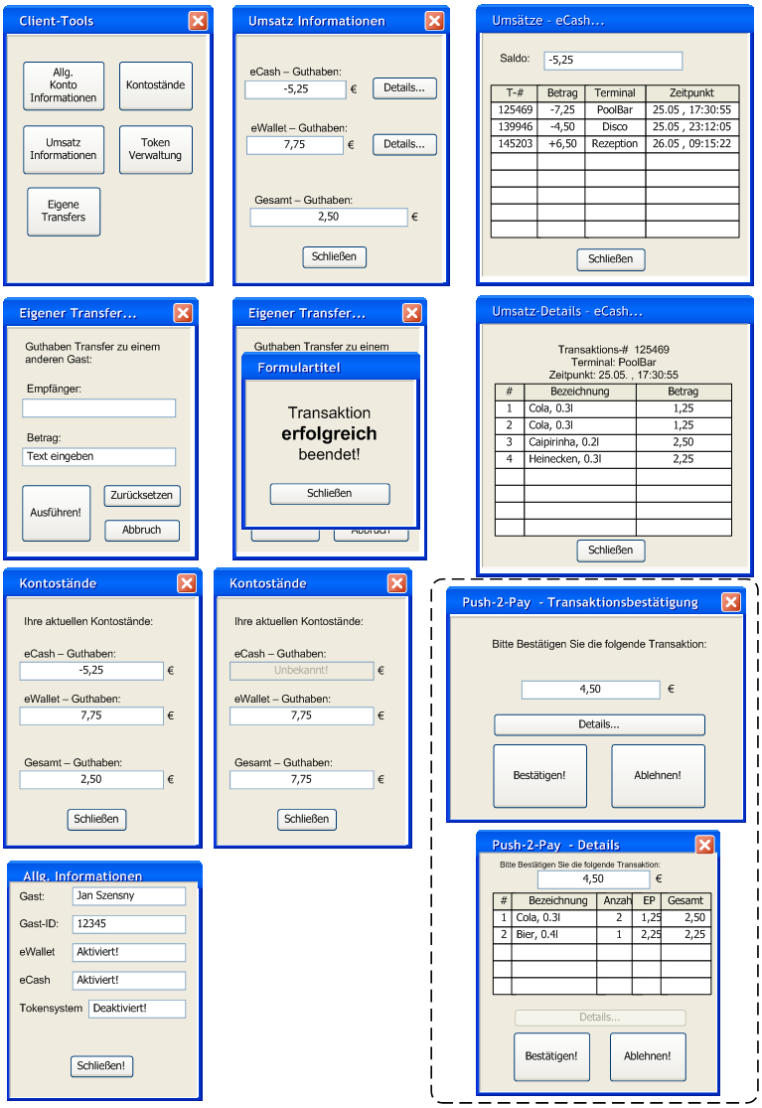
Terminal - GUIs



Beschreibung der Abbildungen von links oben nach rechts unten:

(1) Terminal - Eingabe-Dialog (2) positive Ergebniss-Rückmeldung
(3)+(4) Netzwerkfehler (5) durch Gast abgelehnte Transaktion
(6) vom System abgelehnte Transaktion (Guthabenmangel)

Client - GUIs



Beschreibung der Abbildungen von links oben nach rechts unten:

(1) Client-Tools Hauptmenü (2) Guthabenübersicht (Umsatzinformationen)
(3) eCash-Umsatz - Details (4) Dialog für Guthabentransfer an andere Gäste
(5) Erfolgsmeldung zum Guthabentransfer an andere Gäste
(6) Transaktions-Details (hier eCash-Transaktion) (7) Kontostandsanzeige (Online)
(8) Kontostandsanzeige (Offline)
(9) allg. Informationen zu den Kontoeinstellungen und -informationen des Gastes
Kasten rechts unten: Push-2-Pay - Bestätigungs-Dialog (a) einfach (b) detailliert

C. CD-ROM - Inhalt

Auf der beiliegenden CD-ROM sind folgende Daten enthalten:

```
/
/bin
/bin/client                 -- Client-Komponente
/bin/service                -- Service-Komponente sowie Admin-Tool
/bin/terminal               -- Terminal-Komponente
/doc
/doc/thesis.pdf             -- Dieses Dokument
/doc/graphics               -- Grafiken aus dieser Arbeit
/doc/bibliography           -- Literatur-Quellen (bibtex)
/src
/src/eclipseProject         -- Das Eclipse-Projekt Verzeichniss
                               der Implementation
```

Literaturverzeichnis

ACM : *ACM: Association for Computing Machinery*. – URL http://www.acm.org/. – Zugriffsdatum: 2005-10-14

anonymous 2001 ANONYMOUS: *Der neue Hacker's guide - Sicherheit im Internet und im lokalen Netz*. Markt+Technik Verlag, 2001. – ISBN 3-8272-5931-2

BioAPI HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.: *Biometrics API - Revision 0.6 (DRAFT)*. – URL http://devresource.hp.com/drc/technical_papers/Bioapi.jsp. – Zugriffsdatum: 2005-10-14

Bryant 1988 BRYANT, Billy: Designing an Authentication System: a Dialogue in Four Scenes. (1988), Februar. – URL http://web.mit.edu/kerberos/www/dialogue.html. – Zugriffsdatum: 2005-05-22

ComputerBase COMPUTERBASE: Kerberos (Informatik). In: *ComputerBase Lexikon*. – URL http://www.computerbase.de/lexikon/Kerberos_(Informatik). – Zugriffsdatum: 2005-10-14

CrE-ME V4.00 von NSIcom NSICOM LTD. (Hrsg.): *CrE-ME Runtime*. – URL http://www.nsicom.com/. – Zugriffsdatum: 2005-10-14

Export Force 2.0 LUKAS KLAMM: *Klamm - Export Force 2.0*. – URL http://ef.klamm.de. – Zugriffsdatum: 2005-10-14

Jeode*Runtime* INSIGNIA SOLUTIONS (Hrsg.): *JeodeRuntime*. – URL http://www.insignia.com. – Zugriffsdatum: 2005-10-14

Jeode*Runtime* - Documentation INSIGNIA SOLUTIONS (Hrsg.): *JeodeRuntime - Documentation*. – URL http://www.cs.unc.edu/~lindsey/7ds/notes/jeode/. – Zugriffsdatum: 2005-10-14

Lüpke 2004 LÜPKE, Andre: Entwurf einer Sicherheitsarchitektur für den Einsatz mobiler Endgeräte. (2004), April. – URL http://users.informatik.haw-hamburg.de/~ubicomp/papers.html. – Zugriffsdatum: 2005-10-03

MIT Kerberos MIT (Hrsg.): *Kerberos: The Network Authentication Protocol*. – URL http://web.mit.edu/kerberos. – Zugriffsdatum: 2005-10-14

moneybookers.com : *moneybookers.com*. – URL http://www.moneybookers.com. – Zugriffsdatum: 2005-10-14

PayPal : *PayPal*. – URL http://www.paypal.com. – Zugriffsdatum: 2005-10-14

RFC 1510 IETF (Hrsg.): *RFC 1510 - The Kerberos Network Authentication Service (V5)*. – URL http://www.ietf.org/rfc/rfc1510.txt. – Zugriffsdatum: 2005-10-14

Swing-Paket der Version 1.1.1 SUN (Hrsg.): *Swing 1.1.1*. – URL http://java.sun.com/products/jfc/download.html. – Zugriffsdatum: 2005-10-14

Szensny 2005 SZENSNY, Jan: ePayment in Ferienclubs: Entwurf eines ePaymentsystems unter Nutzung von PDAs. (2005), März. – URL http://users.informatik.haw-hamburg.de/~ubicomp/papers.html. – Zugriffsdatum: 2005-10-03

Tanenbaum 2000 TANENBAUM, Andrew S.: *Computernetzwerke*. 3. revidierte Auflage. Pearson Studium, 2000. – ISBN 3-8273-7011-6

W3C WORLD WIDE WEB CONSORTIUM (Hrsg.): *Extensible Markup Language (XML) 1.0 (Third Edition)*. – URL http://www.w3.org/TR/REC-xml/. – Zugriffsdatum: 2005-10-14

Wächtler 1999 WÄCHTLER, Peter: Kerberos - Eine Frage des Vertrauens. In: *Linux Magazin* (1999), Mai. – URL http://www.linux-magazin.de/Artikel/ausgabe/1999/05/Kerberos/kerberos.html. – Zugriffsdatum: 2005-05-22

Wikipedia a WIKIPEDIA: Kerberos (Informatik). In: *Wikipedia*. – URL http://de.wikipedia.org/wiki/Kerberos_%28Informatik%29. – Zugriffsdatum: 2005-10-14

Wikipedia b WIKIPEDIA: XML. In: *Wikipedia*. – URL http://de.wikipedia.org/wiki/Xml. – Zugriffsdatum: 2005-10-14

XML Core Working Group Public Page WORLD WIDE WEB CONSORTIUM (Hrsg.): *XML Core Working Group Public Page*. – URL http://www.w3.org/XML/Core. – Zugriffsdatum: 2005-10-14

XML in 10 Punkten W3C COMMUNICATIONS TEAM: *XML in 10 Punkten*. – URL http://www.w3c.de/Misc/XML-in-10-points.html. – Zugriffsdatum: 2005-10-14

XStream : *XStream*. – URL http://xstream.codehaus.org. – Zugriffsdatum: 2005-10-03

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 20. Oktober 2005	
Ort, Datum	Unterschrift