BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences
Fachbereich VI – Informatik und Medien

Entwicklung einer Point of Sale angebundenen Retail - Applikation (APP) für TPiSCAN auf Basis des mobilen Betriebssystems Android von Google

Technischer Bericht

Durchgeführt bei der WINCOR NIXDORF AG
im Bereich Retail Store Solutions

Autor:	Christian Rau
Studiengang:	Technische Informatik Bachelor
Ausbildungsbetrieb:	WINCOR NIXDORF AG
Fachbetreuung:	Prof. Dipl.-Ing. Helmut Keutner
Betreuung im Ausbildungsbetrieb:	Dipl.-Ing.(FH) Michael Reschke, Dr.-Ing. Sönke Kannapinn (Diplom-Informatiker)

Inhaltsverzeichnis

1 Schutzklausel

Die Inhalte dieser Arbeit sind **nicht** zur Veröffentlichung gedacht und unterliegen der Geheimhaltung.
Dies schließt auch alle Code-Listings, die sich im Anhang befinden, mit ein.

2. Einleitung

Bei der vorliegenden Arbeit handelt es sich um einen Bericht meines Praxissemesters bei der Wincor Nixdorf AG im Bereich Retail Store Solutions in Berlin.

Laut ihrem Internet-Auftritt ist Wincor Nixdorf (WN) „einer der weltweit führenden Anbieter von IT-Lösungen und -Services für Retailbanken und Handelsunternehmen".
„Wincor Nixdorf ist in rund 100 Ländern präsent, davon in 41 mit eigenen Tochtergesellschaften. Insgesamt arbeiten mehr als 9 000 Mitarbeiter im Konzern".[i]
„Wincor Nixdorf wurde am 1. Oktober 1999 als eigenständige Gesellschaft gegründet. Vorläuferunternehmen waren die Siemens Nixdorf Retail & Banking Systems GmbH (1998-1999), die Siemens Nixdorf Informationssysteme AG (1990-1998), die Nixdorf Computer AG (1968-1990) und das Labor für Impulstechnik (1952-1968) des deutschen Computerpioniers Heinz Nixdorf."[ii] Die Kapitalbeteiligungsgesellschaften von Kohlberg Kravis Roberts und Goldman Sachs Capital Partners übernahmen 1999 die Siemens Nixdorf Retail and Banking Systems GmbH und änderten den Namen in Wincor Nixdorf.[iii]

„Der Name steht für "win" und "core" – für gewinnen und für Kernkompetenz in den Branchen Handel und Banken." [1] 2004 ging das Unternehmen schließlich an die Börse. In Abb.1 sieht man den Kursverlauf an der Frankfurter Börse.[iv]

Abb. 1: Aktienkurs WN 2012 iv

Die Wincor Nixdorf AG hat ihren Hauptsitz in Paderborn und stellt Geldautomaten, Kiosksysteme, Kassensysteme und Peripherie, Flaschenautomaten sowie selbstbedienungsfähige Computerkassen, sogenannte Self-Checkout-Lösungen her. Dazu bietet Wincor Nixdorf passend die entsprechenden Dienstleistungen und Software an, unter anderem auch die Software TPiSCAN, die in Berlin und teilweise in den USA entwickelt wird.

[1] Zitat siehe [iii]

3. Fachliches Umfeld

3.1 TPiSCAN

TPiSCAN ist eine Java-basierte Applikation, die im Filialbetrieb auf Self-Checkout-Kassensystemen von Wincor Nixdorf eingesetzt wird. Sie ist plattformunabhängig (Betriebssysteme Linux und Windows) und könnte sogar auf Kassen weiterer Hersteller eingesetzt werden. TPiSCAN erweitert dabei die eigentliche Abverkaufssoftware – auch „Point of Sale“ (POS) genannt –, welche zur Verwaltung der Transaktionen und der dahinter angeschlossenen Logistik sowie zur Beschaffung aktueller Artikel- und Rabattinformationen dient, um die im Self-Checkout (SCO) erforderlichen Funktionen und Geräte. Die Kommunikation zwischen TPiSCAN und der POS-Software regelt eine einheitliche Schnittstelle, die durch eine Adapter-Bibliothek verkörpert wird. Diese Bibliothek ermöglicht die Integration der bereits vorhandenen POS-Applikation des jeweiligen Unternehmens („Retailer“). Etwa 40 solcher unterschiedlicher POS-Integrationen sind bereits erfolgt, unter anderem auch mit den WN-eigenen POS-Produkten TP.net und TPLinux.
TPiSCAN hat eine sehr flexible, intuitive und ergonomisch gestaltete grafische Benutzeroberfläche, die den Nutzer führt und durch multimediale Inhalte unterstützt. [v]
Zu TPISCAN gehören unter anderem auch Management- und Monitoring-Werkzeuge. [v]

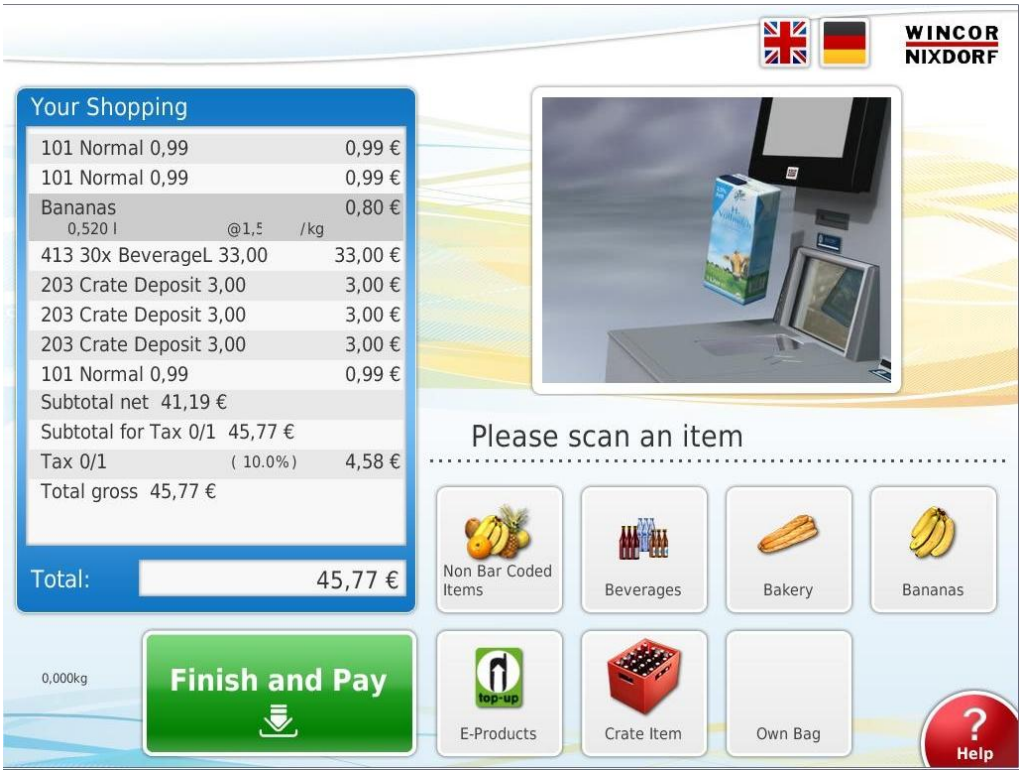

Abb. 2: TPiSCAN Oberfläche

3.2 TPiSHOP

Des Weiteren bietet Wincor Nixdorf auch die Software „TPiSHOP“ für “mobile shopping” an. Sie ermöglicht dem registrierten Kunden einer Einkaufskette, der im Besitz einer Kundenkarte ist, seine Waren beim Einkaufen im Laden eigenständig mit spezialisierten mobilen Handgeräten vorzuscannen (Abb. 3: Ein Kunde scannt den Barcode eines Produktes). Der Kunde erhält dadurch den Vorteil, seinen Aufenthalt im Kassenbereich (Abb. 4: Self-Checkout Terminals im Kassenbereich) deutlich zu verkürzen und gewinnt bereits vor dem Bezahlen einen Überblick über seinen Warenkorb.

Abb. 3: Ein Kunde scannt den Barcode eines Produktes [vi]

Abb. 4: Self-Checkout Terminals im Kassenbereich [vi]

Die robusten, mobilen Handgeräte sind so genannte PDAs, die auch im industriellen Umfeld Verwendung finden. Sie basieren meist auf Windows CE und verfügen über integrierte Barcode-Scanner (siehe Abb. 5 und Abb. 6). TPiSHOP sorgt über ein umfangreiches Batteriemanagement dafür, dass die PDAs immer geladen sind, wenn der Kunde zu Beginn seines Einkaufs einen aus der Ladestation entnimmt (siehe Abb. 7).

Abb. 5: Beispiel: integrierte Barcode-Scanner-Hardware im PDA [2]

Abb. 6: TPiSHOP PDA von Datalogic [vi]

Abb. 7: Ladestationen (TPiSHOP) [vi]

[2] (Foto wurde selber aufgenommen)

Die TPiSHOP-PDAs sind per WLAN an die WN-eigene POS-Software TP.net angebunden, welche wie eine Art Zwischenstation zum eigentlichen Ziel-POS des Retailers fungiert. Es managt die Transaktionen von bis zu einigen hundert mobilen Geräten mit einer deutlich kleineren Anzahl von TP.net-Instanzen (unter Einsatz einiger interner Kunstgriffe). Die dabei erzeugten Transaktionen beinhalten Artikelinformationen über die eingescannten Artikelcodes.

Die Artikelpreise und ihre möglicherweise gewährten Rabatte sind dabei nicht unbedingt akkurat. Um dies zu gewährleisten, müssten die Artikel direkt im Ziel-POS des Retailers verbucht werden, da nur dort die aktuell korrekten Preise einschließlich aller Rabatte errechnet werden können. Grund dafür ist, dass das hinter TPiSHOP liegende TP.net aus Gründen hoher Integrationsaufwände in der Regel nicht in vollem Umfang über alle Rabattregeln des Ziel-POS informiert gehalten werden kann.
Mit TPiSHOP müssen für den Bezahlvorgang die Transaktionen schließlich von TP.net an das Ziel- POS übertragen werden. Dies geschieht typischerweise durch Übergabe einer einfachen Liste der Scancodes von TP.net an das Ziel-POS. Demnach bedeutet dies Integrationsarbeit auf Seiten des POS-Herstellers oder des Retailers.

Das fehlende Wissen über Rabatte behindert auch gezielte Marketingaktionen; dies ist in der Vergangenheit eine wesentliche Schwäche der TPiSHOP-Lösung gewesen.
Die in der Vergangenheit benutzten, auf Windows CE basierenden mobilen TPiSHOP-PDAs sind aus Kundensicht in die Jahre gekommen, da die Kunden selber zunehmend leistungsfähigere und funktionalere Geräte, sogenannte Smartphones, besitzen und permanent mit sich führen.

3.3 Android und der Markt der Mobilen Plattformen

In der Wikipedia Enzyklopädie steht: „Android ist sowohl ein Betriebssystem als auch eine Software-Plattform für mobile Geräte wie Smartphones, Mobiltelefone, Netbooks und Tablets, die von der Open Handset Alliance (Hauptmitglied: Google) entwickelt wird.“ [vii]

Die verbreitetsten, den Markt am stärksten dominierenden Plattformen sind die des Suchmaschinen-Marktführers Google mit dem Betriebssystem Android, das des Geräte- und Softwareherstellers Apple mit iOS sowie Blackbarry OS von RIM.

Neben diesen Vorreitern gibt es noch eine Reihe weiterer vielversprechender Konkurrenten wie z.B. Microsoft, die in Zukunft zusätzlich an Bedeutung gewinnen könnten [3]. Der Hersteller Nokia hat die in der Vergangenheit erfolgreiche Plattform Symbian zu Gunsten von Windows Mobile auf seinen neuen Smartphone Modellen abgelöst. Nokia könnte aber auch zukünftig sein Angagement für die Meego-Plattform wieder verstärken.

Die Diagramme 1 und 2 visualisieren den momentanen Markt der mobilen Geräteplattformen und dessen Prognosen über die zukünftige Entwicklung. [viii]

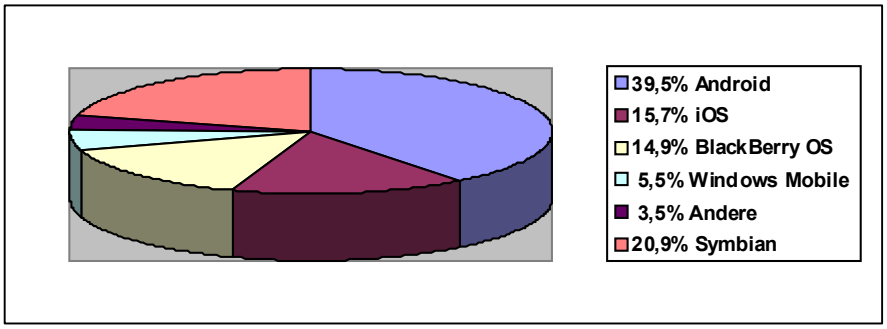

Diagramm 1: Marktaufteilung 2011 (eigene Darstellung) [viii]

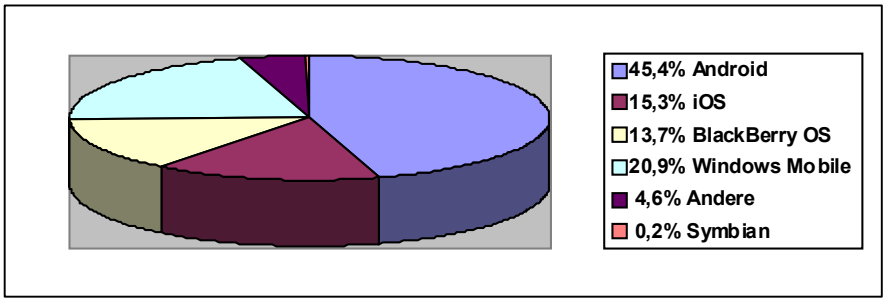

Diagramm 2: Prognose der Marktaufteilung bis 2015 (eigene Darstellung)[viii]

[3] Vgl.: viii

4. Aufgabenstellung

4.1 Die Idee für eine neue Mobile-Shopping-Lösung

Die Zukunft des Mobilen Shoppings wird heute, nicht nur bei Wincor Nixdorf, vor allem in Smartphones gesehen, die immer leistungsfähiger werden und immer mehr Verbreitung finden.
Somit könnten mit einer zukünftigen smartphone-basierten Lösung mehr Kunden erreicht werden. Auch der permanente Zugang zum Internet auf einem Smartphone entwickelt sich zunehmend zur Standardausstattung.
„Weltweit gibt es derzeit mehr Mobiltelefone als Computer, Kreditkarten oder Fernsehgeräte. Für das Jahr 2015 wird prognostiziert, dass mehr Leute über ihr Smartphone, Tablet oder ein anderes mobiles Gerät ins Internet gehen als über einen stationären Computer (IDC Study 2011).“ [ix]
Durch den Zugang zum Internet lässt sich jetzt auch E-Commerce mit Mobile Shopping verbinden, um mehrere Verkaufskanäle besser bedienen zu können („Multikanalstrategie“). Dies führt letztlich, so hofft der Handel, zu einer stärkeren Kundenbindung.[4]
Die neuen veränderten Anforderungen an Mobiles Shopping in Verbindung mit Multikanalstrategien und deren Marketing erfordern für ein zeitgemäßes Produkt ein Umdenken und eine Neuentwicklung bei Wincor Nixdorf.

Die grundlegende technische Idee für so ein neues Produkt bestand nun darin, die in TPiSCAN vielfach bewährte POS-Adapter-Schnittstelle zu nutzen, um darüber vom Smartphone aus mit einer POS-Instanz eine Verkaufstransaktion direkt durchzuführen. Diese Art der Realisierung ermöglicht dann auch den direkten Zugang zu aktuellen Artikelinformationen mit all ihren möglichen Rabatten.

[4] vgl. [ix]

4.2 Die Aufgabe während meiner Praxisphase

Meine Aufgabe war es, eine Demonstrations-Anwendung zu erstellen, die die Idee der oben beschriebenen Architektur umsetzt und vorerst prinzipiell zeigt, dass es technisch möglich ist, über ein Smartphone direkt mit einem POS über das bewährte WN POS-Interface zu kommunizieren. Zielplattform sollte das Betriebssystem Android sein, welches auf Smartphones und Tablet-PCs zum Einsatz kommt.
Als wichtiges Ziel im Rahmen meiner Aufgabe sollte das Herstellen einer 1:1-Anbindung zwischen der SCO-Seite und der POS-Seite sein. Die POS-Instanz sollte dabei zeitgemäß auf einem virtualisierten Rechner laufen und über das Netzwerk für ein Smartphone als SCO-Seite erreichbar sein. Bei Erfolg könnten mehrere dieser Strukturen aufgebaut werden, um dann auch mehrere Smartphones bedienen zu können. Ob es darüber hinaus möglich ist, mit wenigen POS-Instanzen viele Smartphones auf der SCO-Seite abzufertigen, ist offen. Eine entsprechende Studie sollte nicht Bestandteil meiner Praktikumsaufgabe sein und später durchgeführt werden.
Die Applikation sollte in der Lage sein, Barcodes zu scannen und eingescannte Artikel gegen ein angebundenes POS zu verkaufen. Zudem sollte der Warenkorb mit den bereits gebuchten Artikeln anzeigbar sein. Der soweit gescannte Einkauf, also die offene Transaktion, sollte zum abschließenden Bezahlen an einen Pay-Tower (POS-Terminal) übergeben werden können. Die Übergabe sollte mit einem zweidimensionalen Barcode erfolgen. Der Code sollte vom Display des Smartphones am Pay Tower abgescannt werden. Die Fähigkeit des Pay Towers dies so zu tun, sollte ich dabei als vorausgesetzt ansehen.

Nach der Verwirklichung meiner Aufgabe könnte zukünftig auch web-basierte Werbung in der App eingeblendet werden. Denkbar wäre zudem auch, dass weitergehende Artikelinformationen angezeigt werden oder das ein virtueller Einkaufzettel bzw. Einkaufsplaner oder gar das Einkaufen von zu Hause („Multichannel“) integriert wird. Der Kunde soll beim Einkaufen multimedial unterstützt und zum Kauf der Artikel angeregt werden.
Das Projekt lässt sich als prestigeträchtiges Innovationsprojekt einstufen und hat durchaus eine gewisse strategische Bedeutung.

5. Lösungsansätze

Auf dem Markt der Smartphones existiert eine ganze Reihe verschiedener Plattformen bzw. Geräte mit ihren spezifischen Betriebssystemen (siehe 3.3).
Durch die Fragmentierung des Marktes muss bedauerlicher Weise für jede weitere unterstützte Plattform die Applikation neu entwickelt werden. Begegnen könnte man dem Problem mit sogenannten Webapps. Webapps basieren auf Internettechnologien wie HTML und Javascript. Sie stützen sich also auf den Browser der Geräte als Laufzeitumgebung. Er ist auf den einzelnen Plattformen sehr ähnlich und ermöglicht unter Verwendung spezialisierter Frameworks wie jQuery Mobile, dass eine Webapp auf dem Endgerät aussieht wie eine App, die direkt und mit der Hersteller-API für das Gerät geschrieben wurde (native App).
Allerdings ermöglichen diese mit Javascript erstellten Webapps nicht den Zugriff auf Gerätefunktionalitäten (vgl. [x]).
Bei dem Projekt ist jedoch durch die Auswertung von Barcodes der intensive Kameraeinsatz gefordert, was eine Webapp erst einmal ausschloss.
Da die verfügbaren Entwickler in der Abteilung ihren Schwerpunkt eher bei der Programmiersprache Java als bei der Skriptsprache Javaskript haben, entschied man sich für das Projekt erst einmal für eine native Android-Umsetzung in Java, um erste Erfahrungen und Wissen im aktuellem Bereich der Smartphones zu sammeln und aufzubauen.
Bei dem Projekt wurde auf einen plattformunabhängigen Ansatz der Entwicklung verzichtet, da auch die aktuelle Diskussion um native Apps, Web-Apps oder gar Hybrid-Apps, die eine Mischung beider Welten darstellen, noch nicht mit einem befriedigenden Ergebnis zu Ende geführt werden konnte.

6. Durchführung

6.1 Vorbereitungen

Zu Beginn meines Praktikums stand die Einrichtung meiner Arbeitsumgebung, die auch die Einrichtung meines PC beinhaltete. Dies beanspruchte etwas Zeit, weil noch unternehmensspezifische Einrichtungsvorgänge von der Sicherheitsabteilung vorgenommen werden mussten.
Danach schulten mich einzelne Entwickler über die Architektur von TPiSCAN und die POS-Anbindung über die POS-Schnittstelle.
Zur Entwicklung einer Android-App bietet sich die IDE Eclipse an, da sie von Google empfohlen und durch ein Plugin (ADT Plugin) unterstützt wird. Zudem gab es bisher viel positive Erfahrung mit Eclipse in der Abteilung.
Die Installation des Android SDK und des ADT Plugin für Eclipse wird sehr gut und ausführlich auf der Entwicklerseite für Android beschrieben [xi] und kann auch von dort kostenlos bezogen werden. [xii]

Um mit der Arbeit zu beginnen, musste ich verstehen, wie der POS Adapter funktioniert.
Er soll hier nun oberflächlich betrachtet werden.
Das Interface definiert zunächst abstrakt die Kommunikation zwischen POS und SCO. Sie besteht grundsätzlich aus Methoden mit Eingabe- und Ergebnisparametern, die entweder in der Richtung SCO → POS ausgeführt werden können (d.h. SCO ruft auf und POS führt aus) oder umgekehrt in der Richtung POS → SCO (d.h. POS ruft auf und SCO führt aus).
Die Methoden sind in Gruppen unterteilt, wobei aus jeder Gruppe stets höchstens eine Methode aktiv sein kann. Man spricht hier daher auch von einer Synchronisationsgruppe. Das Interface definiert vier solche Gruppen: Es gibt eine „Main-Gruppe“, die Methoden in der Richtung SCO → POS enthält. In der umgekehrten Richtung gibt es entsprechend die „POS-Events-Gruppe“. Zu den beiden Gruppen gibt es eine dazugehörige „Cancel-Gruppe“, die einen Main- bzw. POS-Events-Call abbrechen kann für den Fall, dass z.B. auf die Antwort zu lange gewartet werden muss. Für die Umsetzung des Interfaces steht je eine inhaltlich identische Bibliothek für die Programmiersprachen C++, C# und Java zur Verfügung. Mit ihr ist es möglich, einen spezifischen POS-Adapter auf der POS Seite zu implementieren. Die POS-Entwickler können dabei den Adapter in ihr POS integrieren oder extern anbinden, wie es in der Abbildung unten zu sehen ist. Die Abteilung benutzt für die Entwicklung von TPiSCAN ein eigens entwickeltes „Demo-POS“ welches das Verhalten eines POS simuliert und dabei das Gegenstück der

POS-Adapter-Schnittstelle in Java implementiert. Die SCO-Seite, also TPiSCAN, benutzt die Java-Variante der Bibliothek, wie auch die zu entwickelnde Android-App. Interessant fand ich den Ansatz, dass alle Bibliotheken für die verschiedenen Sprachen größtenteils durch einen eigenen Generator erzeugt wurden.
Für eine spätere Umsetzung für iOS müssten die Bibliotheken eventuell für Objective-C portiert werden, bzw. durch den Generator zusätzlich neu erzeugt werden. Es wäre dabei denkbar, den bestehenden C-Code direkt in Objective-C-Code einzubinden.

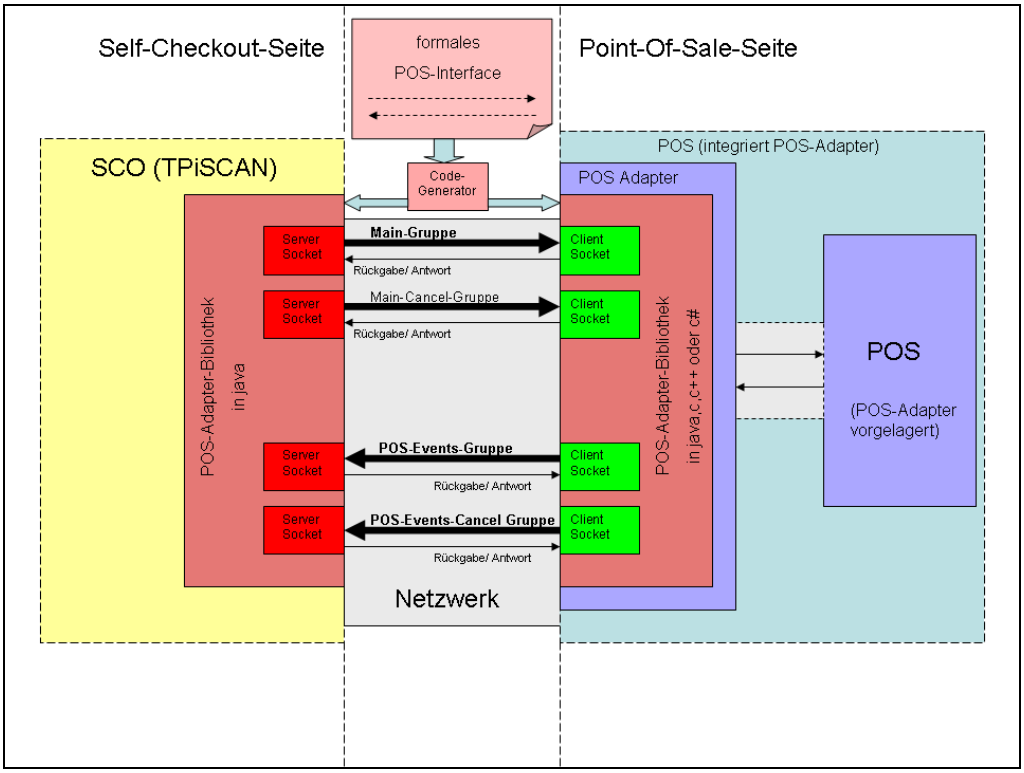

Abb. 8: POS-Adapter (eigene Darstellung)

Nachdem ich mich intensiv mit dem POS-Adapter und den einzelnen Methodenaufrufen und Objekten, die in deren Parametern vorkommen können, beschäftigte hatte, machte ich mich an die restlichen Vorbereitungen, die für die Entwicklung erforderlich waren.
Hierzu importierte ich unter anderem das Demo-POS Projekt über SVN nach Eclipse. Im weiteren Verlauf des Projektes erstellte ich davon zu Testzwecken mit Eclipse eine ausführbare JAR-Datei, welche alle weiteren projektbezogenen Abhängigkeiten enthielt. Sie sollte auf einem virtuellen Rechner ausgeführt werden[5], um einen realen Test über das Netzwerk mit einer WLAN-Strecke

[5] Vgl. Anforderungen oben bei 4. Aufgabenstellung

durchzuführen.
Den virtualisierten Rechner erstellte ich mit VirtualBox. Ich habe mich dabei vor allem aus lizenzrechtlichen Gründen für VirtualBOX OSE entschieden.
Es ist mit VirtualBox im Vergleich zu Qemu, welches zuerst benutzt werden sollte, unkomplizierter, Netzwerk bezogene Einstellungen umzusetzen. VirtualBOX OSE besitzt ein optionales GUI, den VirtualBox OSE Manager (Abb. 9), um die „Virtuellen Maschinen“, wie die virtualisierten Abbilder der virtuellen Rechner oft genannt werden, zu verwalten.

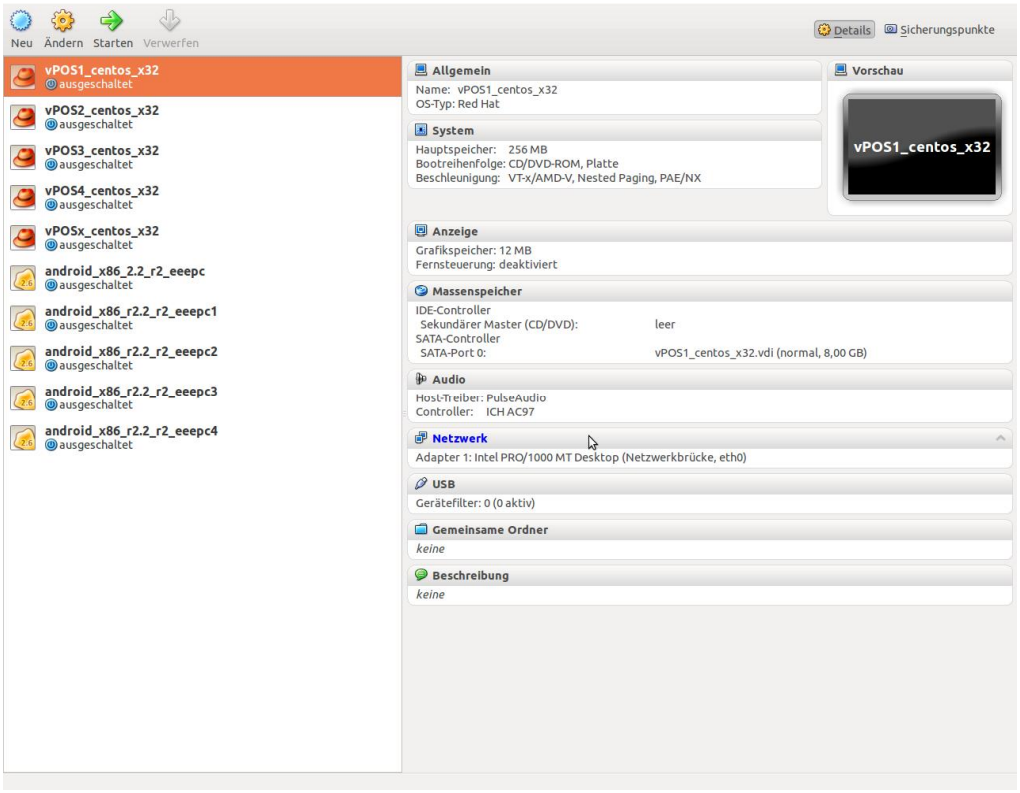

Abb. 9: VirtualBox OSE Manager in Ubuntu Linux als Hostsystem

Nach dem Erstellen einer Virtuellen Maschine mit dem Betriebssystem CentOS, welches bei Wincor Nixdorf bereits verwendet wird, führte ich später erfolgreiche Tests mit dem Demo-POS durch. Ein Vorteil von Virtuellen Maschinen ist, dass man sie relativ einfach kopieren kann und damit schnell zu weiteren Instanzen eines POS kommt.

Man kopiert hierzu das virtuelle Festplattenabbild („Image“) mit der Dateierweiterung .vdi mit Hilfe des VBoxManage Befehls in einem Terminal.

Hier ein Beispiel:

```
user@server:~$ VBoxManage clonehd /home/tpiscan/VirtualBox\
VMs/vPOS1_centos_x32/vPOS1_centos_x32.vdi /home/tpiscan/VirtualBox\
VMs/vPOSx_centos_x32/vPOSx_centos_x32.vdi
```

clonehd weist hier VBoxManage an von der Quelldatei eine verwendbare Kopie zu erzeugen.

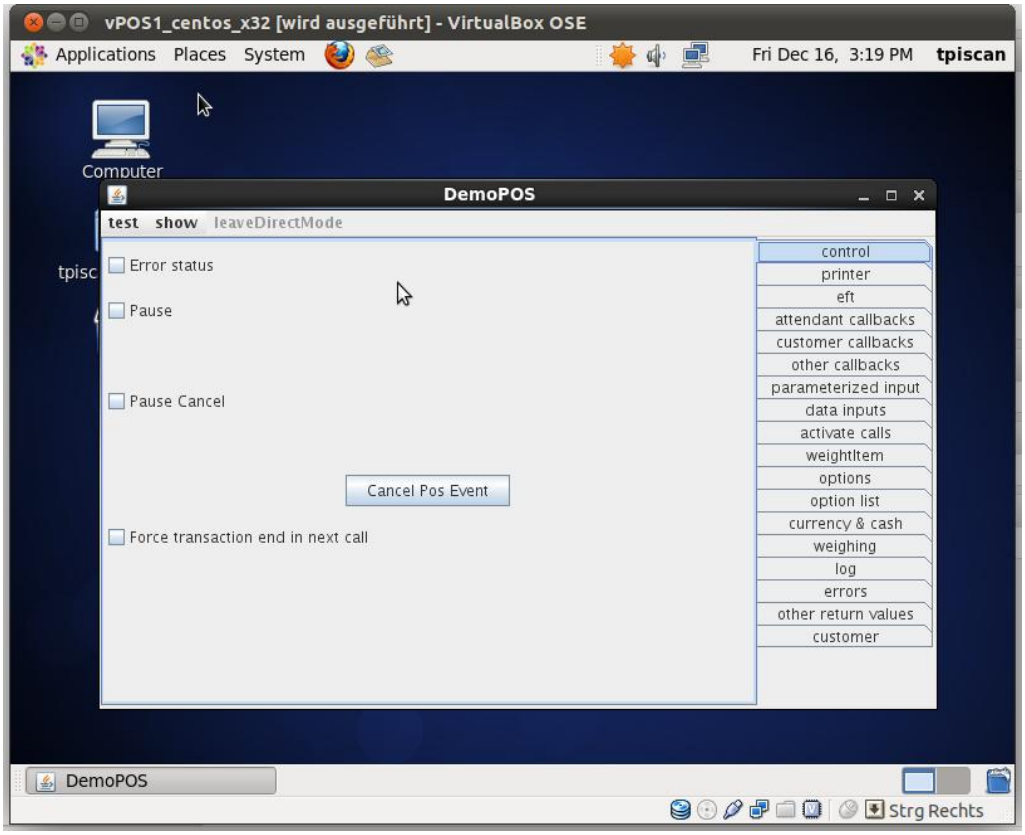

Abb. 10: Virtuelle Maschine mit CentOS 6 und automatisch gestartetem Demo-POS

Virtual Box lässt sich wie gezeigt also auch relativ einfach über ein Terminal steuern. Hier noch ein Beispiel für das Starten und Stoppen der Virtuellen Maschine mit dem Namen “vPOSx_centos_x32“:

```
user@server:~$ VBoxManage startvm vPOSx_centos_x32
Waiting for the VM to power on...
VM has been successfully started.
user@server:~$ VBoxManage list runningvms
"vPOSx_centos_x32" {0006e8d4-f9d7-4268-873e-3c8b9958a5a8}
user@server:~$ VBoxManage controlvm "vPOSx_centos_x32" poweroff
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
```

controlvm bietet noch einige andere Optionen, die den Zustand der Virtuellen Maschine betreffen. Hier ein Auszug aus VBoxManage help:

```
VBoxManage controlvm       <uuid>|<name>
                    pause|resume|reset|poweroff|savestate|
                    acpipowerbutton|acpisleepbutton|
                    keyboardputscancode <hex> [<hex> ...]|
                    setlinkstate<1-N> on|off |
                    nic<1-N> null|nat|bridged|intnet|hostonly
                          [<devicename>] |
                    nictrace<1-N> on|off
                    nictracefile<1-N> <filename>
                    natpf<1-N> [<rulename>],tcp|udp,[<hostip>],
                               <hostport>,[<guestip>],<guestport>
                    natpf<1-N> delete <rulename>
                    guestmemoryballoon <balloonsize in MB>]
                    gueststatisticsinterval <seconds>]
                    usbattach <uuid>|<address> |
                    usbdetach <uuid>|<address> |
                    vrde on|off |
                    vrdeport <port> |
                    vrdeproperty <name=[value]> |
                    vrdevideochannelquality <percent>
                    setvideomodehint <xres> <yres> <bpp> [display] |
                    setcredentials <username> <password> <domain>
                                [--allowlocallogon <yes|no>] |
                    teleport --host <name> --port <port>
                          [--maxdowntime <msec>] [--password password]
                    plugcpu <id>
                    unplugcpu <id>
                    cpuexecutioncap <1-100>
```

Für die Tests im Netzwerk mit den über WLAN angebundenen Smartphones, richtete ich einen Router mit WLAN-Access-Point-Funktionalität ein.
Es handelte sich hierbei um das Model WRT54GL vom Hersteller Linksys (Abb. 11).
Die WLAN-Verbindung soll später nach erfolgreichen Tests durch mobiles Internet (GPRS, HSDPA, usw.) ersetzt werden.
Als Smartphone kam hauptsächlich ein HTC Wildfire (Abb. 12) zum Einsatz. Aber auch Geräte von Samsung wie das Samsung Galaxy SII.

Abb. 11: WLAN-Router Linksys WRT54GL

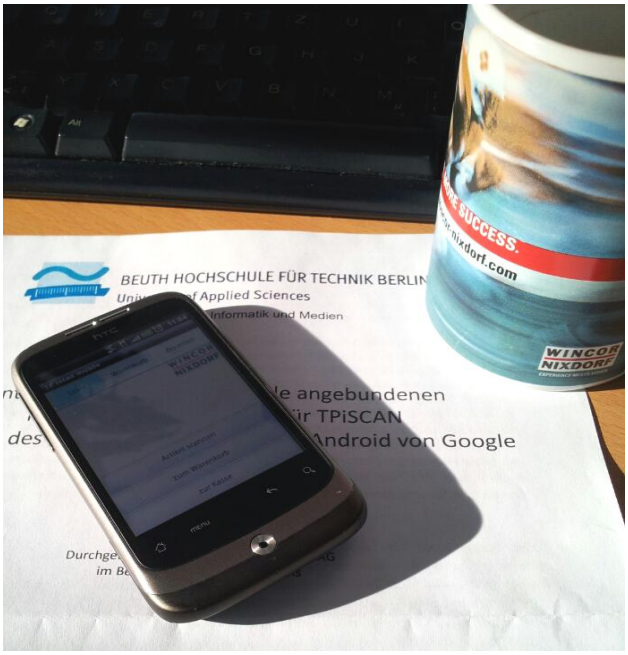

Abb. 12: HTC Wildfire (im Bild links unten)

6.2 Android App

Das Erstellen des Grundgerüsts der App gestalltete sich nicht all zu schwierig, da ich mich zuvor schon intensiver mit dem Thema Android-Programmierung im Rahmen des Wahlfaches Projektmanagement auseinandergesetzt hatte. In Projektmanagement bei Herrn Hausburg hatten ich und mein Team eine alternative Softwaretastatur auf der Android-Plattform umgesetzt.
Für den Einstieg in die Android Programmierung hat mir sehr das Buch „Android 2 Grundlagen und Programmierung“ [xiii] von Arno Becker und Marcus Pant geholfen.
Die Erstauflage kann im Übrigen auch auf der Webseite zum Buch kostenlos als PDF bezogen werden. [xiv]

Die Anforderung, Barcodes zu scannen und zu erzeugen, schien anfangs sehr schwer zu sein. Ich recherchierte, ob es Alternativen gab, um diese Funktionalitäten nicht komplett selber umsetzen zu müssen.
Eine geeignete Möglichkeit fand ich im Zebra Crossing Projekt „zxing“.[xv]
Zxing ist eine Open-Source-Bibliothek, die das Verarbeiten und Erzeugen von Barcodes ermöglicht. Beim Zxing Projekt wird die Kamera eines Smartphones zum Scannen der Barcodes benutzt. Die Bibliothek ist in Java geschrieben und wurde bereits für weitere Programmiersprachen wie Objective-C oder C++ portiert. Die Portierung nach Objective-C könnte später ebenfalls für eine eventuelle Portierung des eigenen Projektes für die iPhone-Plattform interessant sein.
Auf der Zxing-Projektseite steht eine Beipiel Android App unter derselben Lizenz (Apache License 2.0) wie die Zxing-Kernbibliothek zur „freien“ Adaption zur Verfügung. Das Beispiel erleichterte mir das Verständnis zur Benutzung der Bibliothek erheblich.
Nach der Integration der Zxing-Bibliothek implementierte ich die Kommunikation mit dem Demo-POS mit Hilfe des POS Adapters. Dabei konnte ich mich an der Implementation aus TPiSCAN orientieren. Vielen Dank an dieser Stelle an Thomas Schicktanz, der mich an dieser Stelle beispielhaft unterstützte, sowie an Dr. Sönke Kannapinn der mich sehr gut betreut hat und mir meine Fragen zum POS-Adapter umfangreich beantwortete.
Bei der Entwicklung stellte sich heraus, dass die socket-basierte Kommunikation, wie sie bisher zwischen TPiSCAN und POS über Serversockets auf TPiSCAN-Seite realisiert wurde, ungünstig für die Umsetzung auf einem Smartphones ist. Bisher repräsentierte TPiSCAN den Server, weil TPiSCAN die Hoheit über die POS-Anwendung, die Benutzerschnittstelle und die angeschlossenen Geräte hat. Bei Smartphones, die später womöglich über den Telefonprovider anstatt über ein WLAN mit dem POS kommunizieren sollen, ist es allerdings eher ungünstig, selber als Server zu agieren, da die IP-Adresse der Geräte zu Beginn der Kommunikation dem Client nicht bekannt ist. Der Client, in dem Falle das POS, initiiert

normalerweise immer die Kommunikation mit dem Server. Bei dem Projekt kommt aber der Kunde in den Laden und startet seine App, und das POS wartet dabei auf einen Kommunikationspartner.

6.3 TPiSCAN-Mobile-Server

Zunächst hatte ich im Verlauf meines Praktikums also eine Variante implementiert, bei der die App die Serversocket bereit hält.
Es musste nun überlegt werden, wie man diese Beziehung umkehrt, ohne etwas in der Implementierung auf der POS-Seite zu verändern, da dies Aufwand beim Retailer bedeuten würde. Die Lösung war naheliegend und lag bei dem Server, der später unter anderem ermöglichen soll, mit weniger POS-Instanzen auszukommen. Seine Grundfunktionalität besteht jedoch darin, den Smartphones freie POS-Instanzen zuzuweisen. Hier entstand die Idee, dass dieser Server als Vermittler zwischen POS und SCO-App auftritt und dabei die Serversockets bereit hält, um die Beziehung Client—Server umzukehren (siehe Abb. 13). Er müsste dabei die Datenströme intern entsprechend weiterleiten und später sogar umleiten können.

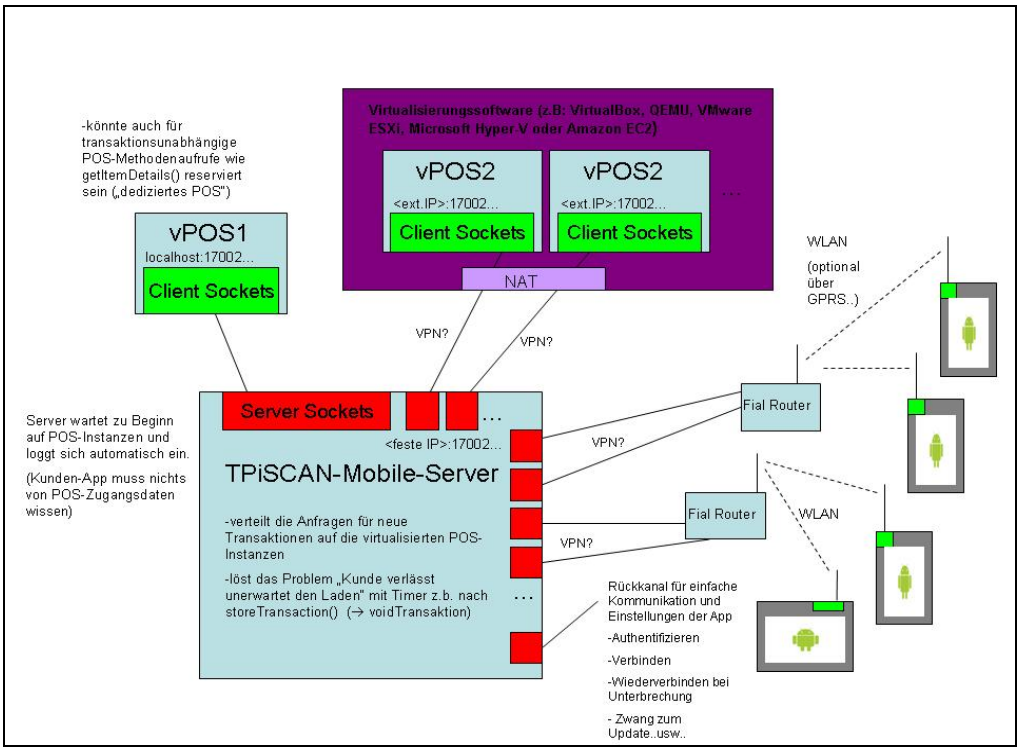

Abb. 13: TPiSCAN-Mobile-Server (eigene Darstellung)

Ich programmierte den „TPiSCAN-Mobile-Server“ [6] nach diesem Konzept und implementierte folgendes Verhalten:
Beim Systemstart wartet der Server erst auf ein POS oder mehrere POS-Instanzen, die sich mit ihm über je einen Adapter mit Serversockets verbinden. Danach hällt er sich für die asynchronen „Verbindungsgesuche“ der Smartphones bereit, die sich über einen zusätzlichen Socket am Server registrieren können[7] und in Folge über je einen eigenen POS-Adapter angebunden werden. Dabei werden sie einem freien POS zugeordnet und miteinander verbunden. In diesem miteinander Verbinden der internen Socketverbindungen bzw. Adapter steckt neben der Implementierung des Servers die Schwierigkeit.
Es ergeben sich einige Vorteile, die sich bei der Verwendung der Adapterbibliothek gegenüber dem bloßen Verbinden der Sockets bzw. der Datenströme ergeben. Beispielsweise ist es möglich, wenn die Adapter der POS- und der SCO-Seite verbunden werden, dass der Server die obligatorische Anmeldung am POS vornimmt und sich über den POS-Status informieren kann. Zum Beispiel „läuft bereits eine Transaktion?“, was gleichbedeutend mit „ist die POS-Instanz belegt?“ ist. Später könnten auch häufige transaktionsunabhängige POS-Methodenaufrufe an eine dafür dedizierte POS-Instanz geschickt werden (eventueller Performance-Gewinn).
Da der POS-Adapter bisher nicht in der Lage war, die POS-Methodenaufrufe und Antworten an einen anderen POS-Adapter weiterzuleiten, war es notwendig, die Adapter Bibliotheksklassen für den Server zu modifizieren. Ich programmierte dazu die Klasse SocketMethodForwarder, die die Funktionalität der Adapterklassen SocketMethodSender und SocketMethodReceiver entsprechend vereinte, um das „Forwarding“, also das gezielte Weiterleiten der Datenströme innerhalb des Servers zu erreichen.
Der SocketMethodForwarder erhielt neben einer forwardMethod noch eine zusätzliche Methode zum direkten Senden einer POS-Methode an die POS-Seite.
So ist es möglich, dass der Server die Kommunikation selber beeinflussen kann, in dem er POS-Methodenaufrufe auch von sich aus versendet.
Ich programmierte den Server also so, dass er sich sofort in angebundene POS-Instanzen einloggt, den jeweiligen Status abfragt und damit weiß, ob sie eventuell doch durch eine laufende Transaktion belegt sein könnte. Für eingeloggte und „freie“ POS-Instanzen lässt er dann in einer Schleife kontinuierlich Smartphones registrieren und aktiviert in Folge jeweils das „Forwarding“ der Methodenaufrufe. Beim Registrieren der Smartphones wird den Smartphones mitgeteilt, mit welchem Port sie sich verbinden sollen, da jede POS-Adapterverbindung auf ihrem eigenen Portbereich läuft. Das „Forwarding“ läuft kontinuierlich in einer Schleife und in jeweils einem eigenen Thread in Analogie zur SocketMethodReceiver-Klasse.

[6] Die Namensgebung ist frei gewählt und nicht endgültig
[7] Hierbei wird auch die IP-Adresse dem System bekannt.

6.4 Testmethoden/Entwicklung des Servers

Zum Testen und Entwickeln benutzte ich die oben beschriebenen Hilfsmittel. Für die Entwicklung des Servers entschied ich mich zusätzlich zu meinem HTC-Wildfire, virtuelle Maschinen mit Android aufzusetzen, durch die ich in der Lage bin, Tests durchzuführen, ohne reale Hardware in Form weiterer Smartphones beschaffen zu müssen.
Eine Motivation war es auch, dass mir einige Entwickler anfangs ihr privates Smartphone für einen Test des Servers zu Verfügung stellten, ohne die ich zu dem Zeitpunkt nicht in der Lage gewesen wäre, einen Test mit mehreren Clients durchzuführen.
Eine weitere Option wäre es gewesen, mehrere Android-Emulatoren[8] dafür zu verwenden, jedoch wollte ich Tests über ein reales Netzwerk durchführen und nicht lokal auf meinem Entwicklungsrechner, da schon zwei Emulatoren viel Ressourcen verbrauchen und vor allem ein Netzwerk eben nur emulieren. Auch eine Virtuelle Maschine mit CentOS und Android-Emulator (Abb. 14) wäre zwar möglich, ist jedoch nicht besonders komfortabel. Eine Virtuelle Maschine mit Android ist sofort nach dem Start im Netzwerk verfügbar. Es muss also nicht erst in der Virtuellen Maschine der Emulator gestartet werden. Zudem stellen diese virtuellen Android-Maschinen geringere Anforderungen an den Entwicklungsrechner.[9] Für die Virtualisierung der POS-Instanzen und Android-Maschinen wurde mir schließlich ein Server mit mehreren Prozessorkernen und mit entsprechend viel Arbeitsspeicher zur Verfügung gestellt, dessen Ressourcen allerdings auch begrenzt sind.

[8] Sie sind Teil des Android SDK. Verbindungen müssen zusätzlich mit einem „Redirekt“ bzw. „Forward“ konfiguriert werden, wenn Android Serversockets hat; siehe Listing Redirect im Anhang
[9] Sie verbrauchen weniger Arbeitsspeicher und Rechenkapazität.

Abb. 14: Bedienoberfläche des Android-Emulators

Die virtuellen Android-Maschinen ließen sich unter Zuhilfenahme des Android-x86 Projekts [xvi] umsetzen. Das Erstellen einer solchen virtuellen Maschine, ist gut auf der Projektseite beschrieben. [xvii]

Um nun die entwickelte App auf die virtuellen Android-Maschinen zu verteilen, nutzte ich die Möglichkeit, sie wie den Android-Emulator des Android-SDK durch die Android Debug Bridge (adb) zu verbinden. Das „Android Development Tools (ADT) Plugin“ für Eclipse tut dies im Übrigen auf dieselbe Art und Weise. Leider konnte ich Eclipse nicht dazu bringen, sich auch mit den virtuellen Android-Maschinen zu verbinden oder Apps auf ihnen nahezu „zeitgleich“ zu starten.

Um mir das Entwickeln und Testen zu erleichtern, erstellte ich eine Reihe von einfachen Batch-Scripten, mit denen ich automatisiert bestimmte Aufgaben über das Netzwerk ausführen kann.

Die Batch-Scripte ermöglichen:

- das Verbinden der virtuellen Android-Maschinen (bzw. Geräte) an den adb-Server mit und ohne Entsperren des Bildschirms (Tastensperre)
- das Verteilen, also Installieren der App über das Netzwerk
- das „zeitgleiche“ Starten der App auf allen Geräten
- das Beenden der Apps auf allen Geräten
- das Senden eines Keyevents (z.B. „BACK“) an alle Geräte
- das Installieren, Starten und Stoppen von Robotium-Tests mit Logging-Funktion

Durch diesen beschriebenen Aufbau ist es möglich, Belastungstests in einem Netzwerk für den Server durchzuführen. Dabei kann unter anderem das Verhalten der einzelnen Geräte, also der mögliche Programmablauf, der sonst durch Interaktion gesteuert wird, genau festgelegt werden. Die automatisierte Steuerung des Android User Interface ermöglicht Robotium. Es handelt sich hierbei um ein Test-Framework, dass das Erstellen von Blackbox-Tests für Android ermöglicht.

Ich programmierte einen Robotium-Test, der wiederholt einen POS-Methodenaufruf auslöst. Diesen Test startete ich dann schließlich mit Hilfe der Batch-Scripte auf allen Geräten. Die Listings zu den Scripten und dem Robotium-Test sind im Anhang enthalten. Sie sind streng vertraulich!
Leider stellte sich bei wiederholten Tests und intensivem Debugging heraus, dass einige der POS-Methodenaufrufe zwar korrekt beim Server ankamen und weitergeleitet wurden, jedoch die Antwort des Aufrufs fehlerhaft beim Server ankam. Dies führte dazu, dass die betreffende App nicht mehr in der Lage war, mit dem Server zu kommunizieren, weil dieser den entsprechenden Forwarding-Thread beendete. Der Fehler trat sporadisch auf und wird vermutlich auf der POS-Seite oder bei der Übertragung auf Netzwerkebene verursacht.

Bis zum Ende des Praktikums konnte der Fehler leider nicht bis zu Ende analysiert und behoben werden. Es ist auch nach einem kleinen Codereview durch meinen Betreuer nicht auszuschließen, dass es sich um eine fehlerhafte Implementierung in der POS-Adapter-Schnittstelle handelt. Ich erstellte sogar ein kleines Testprogramm, welches nur die POS-Adapter-Schnittstelle untersucht, indem es mehrere POS-Adapter wie in meinem Server anbindet und ihnen permanent POS-Methodenaufrufe schickt. Jedoch stellte sich hier der Fehler nicht ein.

Unabhängig von der oben beschriebenen Fehlersuche in der Kommunikation führte ich zusätzlich zu den Robotium-Tests auch einen Stresstest für die Android-App an sich durch, um weitere verborgene Fehler zu finden. Hierfür eignet sich das Kommandozeilen-Programm „Monkey“, welches durch das Android SDK zur Verfügung steht. Es verhält sich wie ein Affe, der keine Ahnung von der Bedienung eines Smartphones hat und wahllos auf dem Gerät herumdrückt. Man kann dem Testprogramm allerdings mitteilen auf welches Programm er sich beschränken soll und wie oft es drücken, also eine zufällige Aktion auslösen soll. Gestartet wird der Befehl monkey über die Android Debug Bridge (ADB).

Über die ADB ist es mir also auch möglich, über das Netzwerk einen derartigen Stresstest auf allen Geräten zu starten, die zuvor an sie angebunden wurden.

Der Befehl für die die entwickelte App sah beispielsweise so aus:

```
adb -s 192.168.1.106:5555 shell monkey -v -v -s 12345 -p com.wn.retail.iscan.android.iscandroid 500
```

Der Befehl setzt sich wie folgt zusammen:

adb -s 192.168.1.106:5555	-s spezifiziert die Emulator/Geräte Instanz
shell	öffnet eine ADB-Shell auf dem Gerät
-v	führt zur Ausgabe von Informationen über aufgetretene Fehler, Ein weiteres -v dahinter macht die Ausgabe noch detailreicher
-s	kann optional angegeben werden, -s steht dabei für „seed“ und setzt den Ausgangswert für den Zufallsgenerator, den monkey verwendet. Dies ist notwendig für den Fall, dass die Abfolge der Ereignisse reproduzierbar sein soll.
-p	Bei Angabe von –p erlaubt man monkey nur Aktivitäten aus dem angegebenen Paket auszuführen. Es ist auch möglich den Parameter mehrfach anzugeben, um weitere Pakete zu spezifizieren.
500	gibt schließlich an wie viele Ereignisse ausgelöst werden sollen

Neben diesen Oberflächentests könnte man zukünftig auch Modultests durchführen, um zu testen, ob sich die einzelnen Kassen so verhalten, wie es erwartet wird. Klassen die unabhängig von der Android-API sind, lassen sich mit normalen JUnit -Tests untersuchen.
Für Klassen, die Android-Komponenten sind oder von ihnen oder anderen Systemkomponenten abhängen, bietet die Android-API im Paket android.test für fast alle Klassen eine Testfallbasisklasse an („Android-JUnit-Tests“). Selbst der Anwendungskontext für die Testklassen lässt sich durch spezielle Kontexttestklassen ersetzen und verändern.
Der Artikel „TÜV für Android-Apps“ aus der Mobile Developer 1⁄2011 (S.26) beschreibt dieses spezielle Thema sehr ausführlich. [xviii]

6.5 Webview

Bei der Diskussion um die Umsetzung die sich im Zusammenhang mit einer zukünftigen Umsetzung für andere Plattformen ergeben, trifft man neben Webapps als Lösungsansatz auch auf Hybridapps. Sie binden in einer Nativen App ein Browserelement ein, welches dann den Inhalt einer Webapp darstellt.
Ich habe mich, auch zur Erprobung der Methodik, dazu entschieden, ein solches Element, eine sogenannte Webview, einzubinden. Mit ihr lassen sich nicht nur Webapps anzeigen, sondern eben auch ganznormale Webseiten. Zukünftige Funktionen wie das Einblenden von Werbung oder das Darstellen von Artikel-Informationsseiten ließen sich so umsetzen. Die Webview band ich in die Startansicht der App ein, um dort gegebenenfalls Werbung einzublenden zu können.

Abb. 15: SCO-App mit einer Webview

Die Webview müsste künftig auf eine Webseite zugreifen, die dann speziell aufbereitete Inhalte zur Verfügung stellt. Es könnte z.B. eine Webseite mit einer animierten Bildergalerie dargestellt werden. Die Abbildung 15 zeigt die Webview, die hier nichts anderes tut, als einen Teilausschnitt einer Internetseite eines bekannten Retailes anzuzeigen. Die Abbildung vermittelt einen ersten Eindruck wie sich eine Werbe-Funktionalität mit einer Webview umsetzen ließe.

6.6 Server und App in Aktion

Zur besseren Darstellung der Funktion der App wurde ein Demonstrationsvideo und Screenshots (siehe unten) erstellt. Sie zeigen die Self-Checkout-App, während sie über WLAN, verbunden mit dem Server, mit einer POS-Instanz kommuniziert. In dem Video wird die Bedienung der App, wie sie bei einem Einkauf beim Retailer stattfinden könnte, beispielhaft einmal durchgegangen.

Zu Beginn sieht man wie Barcodes eingescannt werden (Abb. 16). Danach sieht man den Warenkorb mit den folglich eingebuchten Artikeln (Abb. 17) und anschließend die Übergabe (Abb. 18) der Transaktion. Hierfür wird die Transaktion mittels QR-Code (Abb. 19) angezeigt. Das Video befindet sich im Anhang dieses Berichts auf einer CD.

Abb. 16: App scannt einen Barcode

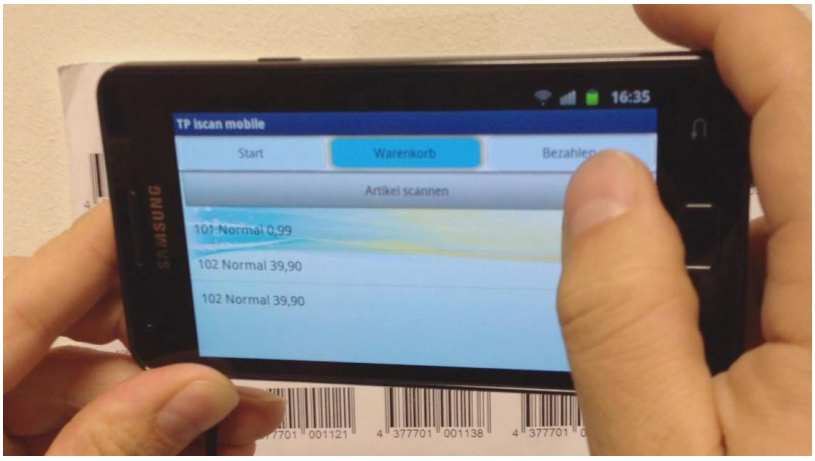

Abb. 17: Warenkorb in der App

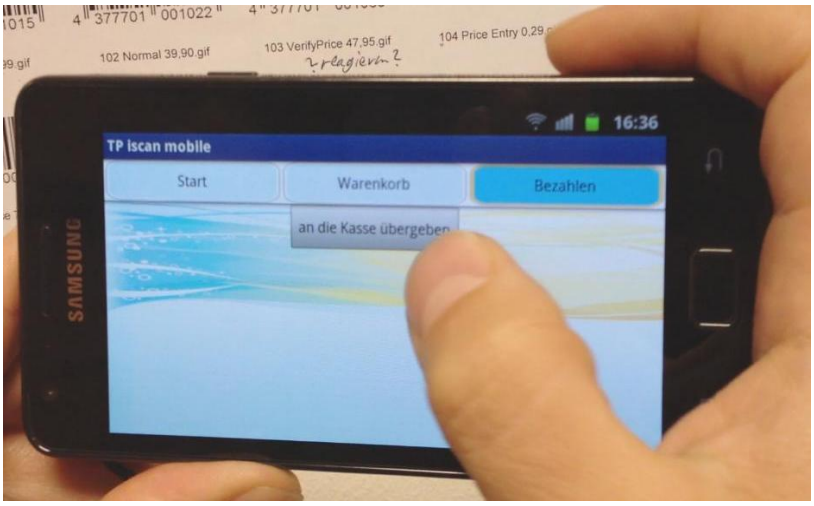

Abb. 18 Benutzer übergibt Transaktion an die Kasse

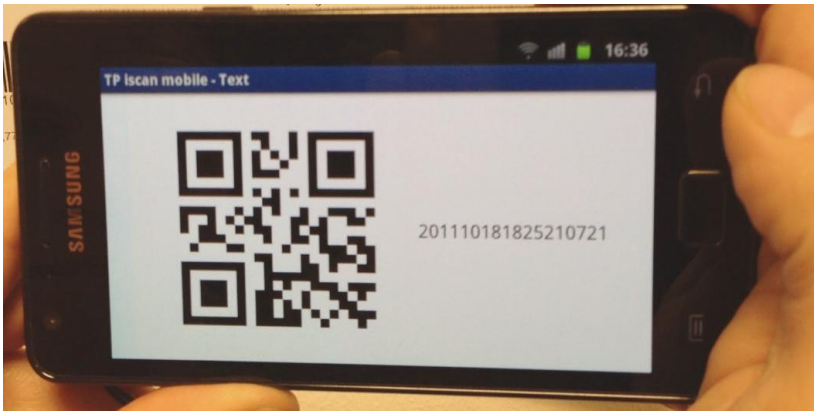

Abb. 19: Die App zeigt einen transaktionsspezifischen QR-Code zum Einlesen am PayTower an.

7. Zusammenfassung und Ausblick

7.1 Zusammenfassung

Meine Aufgabe, eine Demonstrations-App für Android zu erstellen, die die technische Machbarkeit demonstriert, ein Smartphone direkt mit einem POS über das bewährte WN POS-Interface zu verbinden wurde erfüllt.
Ich setzte die Funktion der App Barcodes zu scannen und darstellen zu können durch die Integration der Open Source Bibliothek „Zxing“ um.
Ich programmierte im weiteren Verlauf des Praktikums eine Serverkomponente, die das mehrfache Anbinden solcher 1:1-Anbindungen ermöglicht.
Für die Entwicklung richtete ich mir eine verteilte Testumgebung ein. Hierbei setzte ich mich intensiv mit dem Thema Virtualisierung auseinander und entwickelte geeignete Testmethoden, um die geforderten Funktionalitäten der App entwickeln und überprüfen zu können.
Gegen Ende meines Praktikums experimentierte ich noch mit Webviews, um gegebenenfalls kundenspezifische Werbung darstellen zu können.

7.2 Ausblick

Zukünftig könnten noch viele Verbesserungen und Funktionen hinzugefügt werden. Die App und der Server müssten noch dahingehend verbessert werden, dass auch das gezielte Abbauen oder auch Wiederherstellen von Verbindungen bei Verbindungsabbrüchen behandelt wird. Auch das Authentifizieren der Kommunikationspartner und das Absichern der Verbindung gegen Kompromittierung muss implementiert und getestet werden. Aspekte der Sicherheit im Allgemeinen müssten folglich näher untersucht werden. Auch die spätere Verteilung der App über Googles Android Market muss noch näher durchdacht werden. Damit eng im Zusammenhang steht auch, dass die App zukünftig den Anforderungen des jeweiligen Retailers anpassbar sein muss. Das wiederum schließt auch abweichendes Verhalten in zukünftigen optionalen Funktionen wie das Scannen und Behandeln von Coupons und Pfandrückgabebons mit ein. Die App bzw. der Quellcode muss dabei modularisierbar werden und als eine Art Baukasten für die endgültige für den Retailer spezifische App agieren.

Weitere Funktionen wie das Einblenden von Werbung, Rabataktionen und weitergehender Produktinformationen warten auf eine Umsetzung. Es gibt auch einige Funktionen die zum Themenkomplex Multichannel gehören und sehr

attraktiv für einen Retailer und seine Kunden seien könnten. Zu nennen wären hier die Integration von einem virtuellem Einkaufzettel bzw. Einkaufplaner oder gar das Einkaufen bzw. Bestellen, von zu Hause aus.
Des Weiteren ist auch die parallele Portierung bzw. Entwicklung für andere Plattformen wie iOS ein noch zu bearbeitendes Thema. Dazu könnte auch näher untersucht werden, ob die Erstellung von Webapps bzw. Hybridapps sich lohnen würde, um den Aufwand von Parallelentwicklungen zu minimieren.

Nach meinem Praktikum wurde ich als Werkstudent übernommen und habe somit die Möglichkeit die Mobile-Shopping-App weiter zu entwickeln. Des Weiteren wurde mir in Aussicht gestellt, auch meine Bachelorarbeit bei Wincor Nixdorf zu schreiben.

9. Abbildungsverzeichnis

10. Index

11. Anhang

(Auflistung der Anhänge)

Ausdruck: Pressemeldung IDC

Ausdruck: TPiSCAN_Datasheet.pdf

Ausdruck: TPiShop_Datasheet.pdf

Kopie: mobile Developer 11/2011 „Viele Wege führen zur Plattform“

Kopie: retail technology 04.2011 „Mobile im Anmarsch“

Listings:

- Ausdruck: redirect.doc
- Batch-Scripte:
  - connectDevicesToAdb.bat
  - connectWithUnlookScreens.bat
  - appTestRollout.bat
  - StartApps.bat
  - StopApps.bat
  - startRobotiumTest.bat
  - StopTestApps.bat
  - R1.bat
  - R2.bat
  - R3.bat
  - sendBackToAll.bat
  - testTabs.java

Kopie: mobile Developer 1/2011 „TÜV für Android-Apps“

CD: mit dem Demonstrationsvideo „WN_mobile_shopping_app.MOV “ der Android App

DVD : „WN mobile shopping“: DVD-Video-Version der CD

12. Bestätigung des Berichtes

In diesem technischen Bericht sind Inhalt und Ablauf der praktischen Tätikkeit von Christian Rau bei der WINCOR NIXDORF AG im Bereich Retail Store Solutions ersichtlich und korrekt dargestellt.

Unterschrift der Ausbildungsbeauftragten der Ausbildungsstelle:

Herr Dipl.-Ing.(FH) Michael Reschke: _____ _____

Ort, Datum Unterschrift

Dr.-Ing. Sönke Kannapinn
(Diplom-Informatiker): _____ _____

Ort, Datum Unterschrift

13. Quellen- und Literaturverzeichnis

[i] http://www.wincor-nixdorf.com/internet/site_DE/DE/WincorNixdorf/Company/company_node.html

[ii] http://de.wikipedia.org/wiki/Wincor_Nixdorf

[iii] Vgl. Pressemitteilung von 1999:
http://www.wincor-nixdorf.com/internet/site_DE/DE/WincorNixdorf/Press/pressreleases/1998-1999/KKRundGoldmanSachs_22_10_1999.html

[iv] Vgl. http://www.boerse-frankfurt.de/DE/index.aspx?pageID=35&ISIN=DE000A0CAYB2 (10 Jahre Chart)

[v] Vgl. TPiSCAN_Datasheet.pdf Ausdruck im Anhang oder:
http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/52006/publicationFile/51783/TPiSCAN_Datasheet.pdf

[vi] Vgl. TPiShop_Datasheet.pdf Ausdruck im Anhang oder:
http://www.wincor-nixdorf.com/internet/cae/servlet/contentblob/51342/publicationFile/61764/TPiShop_Datasheet.pdf
siehe auch
http://www.wincor-nixdorf.com/internet/site_DE/DE/Products/Software/Retail/POSSolutions/TPiShop/tpishop_container.html

[vii] http://de.wikipedia.org/wiki/Android_(Betriebssystem)

[viii] vgl. Pressemeldung IDC Ausdruck im Anhang oder:
http://www.idc.com/getdoc.jsp?containerId=prUS22762811

[ix] retail technology 04.2011 „Mobile im Anmarsch“ S.26 Kopie im Anhang

[x] vgl. mobile Developer 11/2011 „Viele Wege führen zur Plattform“ S.21 Kopie im Anhang

[xi] http://developer.android.com/sdk/installing.html

[xii] http://developer.android.com/sdk/index.html

[xiii] Android 2 Grundlagen und Programmierung,
dpunkt.verlag GmbH, ISBN 978-3-89864-677-2

[xiv] http://www.androidbuch.de/index.php/gratis-pdf.html bzw.
http://www.androidbuch.de/download/android-buch.pdf

[xv] http://code.google.com/p/zxing/

[xvi] http://www.android-x86.org/

[xvii] http://www.android-x86.org/documents/virtualboxhowto

[xviii] Vgl. mobile Developer 1/2011 „TÜV für Android-Apps“ S.26 (Kopie im Anhang)

Zusätzlich zum Nachlesen:
http://en.wikipedia.org/wiki/Self_checkout

http://en.wikipedia.org/wiki/Point_of_sale

http://de.wikipedia.org/wiki/POS-Terminal