NFC-basierte Software: Verbreitung durch Vereinfachung

Studiengang Medieninformatik

Masterarbeit

vorgelegt von

Dennis Becker
geb. in Weilburg

Benjamin Rühl
geb. in Alsfeld

durchgeführt an der
Technischen Hochschule Mittelhessen, Friedberg

Referent der Arbeit:	Prof. Dr. Cornelius Malerczyk
Korreferent der Arbeit:	Prof. Dr. Stephan Euler

Friedberg, Juni 2012

Selbstständigkeitserklärung

Wir erklären, dass wir die eingereichte Masterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von uns angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht haben.

Friedberg, Juni 2012

Dennis Becker Benjamin Rühl

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einleitung

1.1 Motivation

Unter Near-Field-Communication (NFC) versteht man einen Übertragungsstandard, der den kontaktlosen Austausch von Daten zwischen zwei Geräten beschreibt [1]. Charakteristisch für eine NFC-Verbindung ist der einzuhaltende Abstand von maximal 4 cm der beteiligten Geräte [2]. Durch diese geringe maximale Entfernung beider Geräte zueinander gilt die Kommunikation via NFC als weitestgehend sicher. Für Dritte ist es fast unmöglich, unbemerkt Datenverkehr mitzuschneiden [3].

Abbildung 1.1: NFC-Kontaktpunkt am Bahnhof, Quelle:[2]

Eine weitere wichtige Eigenschaft ist die geringe Baugröße der benötigten technischen

Komponenten (siehe Beispiel in Abbildung 1.2). Somit lassen sich auch kleine elektronische Geräte um die Fähigkeit des kontaktlosen Datenaustausches erweitern. Besonders interessant ist die Verbindung von NFC mit dem mobilen Telefon. Laut des Statistischen Bundesamts Deutschland war das Mobiltelefon im Jahr 2010 mit ca. 89% Abdeckung (in allen deutschen Haushalten) das am weitesten verbreitete technische Gerät nach Fernseher, stationärem Telefon und Kühlschrank [4]. Google erkannte die Möglichkeiten dieser neuen Verbindung und brachte im Dezember 2010 das Nexus S mit eingebautem NFC-Chip auf den Markt.[1] Hier wurde auch die Version Gingerbread des Google-eigenen Betriebssystems Android eingesetzt, die über API-Schnittstellen für die Verwendung der NFC-Technologie verfügt.[2] Insbesondere der Umstand, dass man sein Smartphone nahezu jederzeit bei sich trägt, erweitert die Nutzungsmöglichkeiten von NFC enorm.

Diese Arbeit wird sich in vielen Bereichen auf die spezielle Anwendung von NFC in Smartphones konzentrieren. Da aktuell nur Android die Möglichkeit von NFC bietet, ist der technische Teil dieser Arbeit auf dieses Betriebssystem beschränkt. Zudem werden alle Prototypen mit Android-Smartphones entwickelt und Nutzertests mit ihnen durchgeführt.

Abbildung 1.2: Samsung Nexus S-Deckel mit NFC-Antenne

NFC ist keine neue Technologie. Bereits seit dem Zweiten Weltkrieg beschäftigen sich Wissenschaftler mit dem kontaktlosen Austausch von Daten [5]. Damals brachte man am eigenen Kriegsgerät Transponder zur Freund-Feind-Erkennung an. Mit Hilfe eines Radars konnte man diese Transponder auslesen und dann bestimmen, ob es sich um eigene oder feindliche Truppen handelt. Im Laufe der späteren Weiterentwicklung entstand die Radio-Frequency-Identification (RFID). Ein RFID-System zeichnet sich durch zwei Komponenten aus: einem passiven Transponder und einem aktiven Lesegerät. Das Lesegerät baut ein elektromagnetisches Wechselfeld auf und versorgt somit den passiven Transponder durch

[1] `http://www.android.com/devices/detail/nexus-s`, abgerufen am 28.06.2012
[2] `http://developer.android.com/guide/topics/nfc/nfc.html`, abgerufen am 28.06.2012

Induktion mit Strom. Dies ermöglicht dem Lesegerät das Auslesen eines Transponders, der selbst über keine aktive Stromversorgung verfügt. Es findet also eine unidirektionale Verbindung statt, die in ihrer Anwendung meist nur für die Identifizierung von Objekten benutzt wird [6].

Tatsächlich baut NFC auf der gleichen technologischen Grundlage auf wie RFID, erweitert diesen Standard jedoch unter anderem um die Möglichkeit zwei aktive Geräte miteinander kommunizieren zu lassen. Jedes NFC-Gerät kann in drei verschiedenen Modi arbeiten: Es kann Transponder auslesen und beschreiben, einen Transponder emulieren oder eine Peer-to-Peer-Verbindung mit einem weiteren aktiven Gerät aufbauen [7]. Was sind also die besonderen Neuerungen durch NFC?

War man bisher nur im Besitz einer kontaktlosen Smartcard[3], konnte man lediglich die passive Rolle in der Verbindung zweier Geräte übernehmen. Besitzer eines Smartphones mit eingebauter NFC-Technologie können in der heutigen Zeit jedoch sowohl passiv Daten zur Verfügung stellen als auch aktiv versenden. Somit generieren sich eine Menge neuer Einsatzgebiete, die den Benutzer im täglichen Leben unterstützen können. Jeden realen Gegenstand kann man durch das Anbringen eines NFC-Tags um einen virtuellen Kontext erweitern. So genügt während eines Museumsbesuchs eine einfache Berührung des Smartphones mit dem angebrachten NFC-Tag und das Smartphone erkennt, um welches Ausstellungsstück es sich handelt (Abbildung 1.3). Zusätzlich kann es multimediale Informationen abrufen und anzeigen. Die Möglichkeiten gehen allerdings über einfache Identifikationsszenarien hinaus. Beispielsweise war man bisher bei nahezu jeder Art von Marketing auf die Rolle des Konsumenten festgelegt. Informationen wurden gezeigt, verarbeitet und oft wieder vergessen. Mit NFC lassen sich moderne, interaktive Marketingstrategien verwirklichen. Vorstellbar wäre es z.B., dass das Smartphone als interaktiver Controller einer entfernen Anwendung verwendet wird. So kann ein Kunde beim nächtlichen Schaufensterbummel die auf dem Computer hinter dem Schaufenster laufende Anwendung über sein Smartphone bedienen. Eine einfache Berührung des auf der Scheibe angebrachten NFC-Tag genügt, um eine Applikation auf dem Smartphone zu starten und es als aktiven Controller zu bestätigen. Nun kann der Kunde zum Beispiel aus verschiedenen Marken, Größen oder Farben seinen persönlichen Schuh zusammenstellen und das fertige Produkt auf dem im Schaufenster integrierten Display begutachten. Ist er mit seiner individuellen Konfiguration zufrieden, speichert er sie ab. Nach einer Kaufentscheidung ermöglicht ihm die Applikation auf seinem Smartphone die Konfiguration an das Geschäft zu übermitteln. Sie benachrichtigt ihn auch sobald das Produkt fertig produziert wurde. Die Bezahlung erfolgt am Tag der Abholung mit seinem Smartphone an einer NFC-Bezahlstation.

Da NFC als weitestgehend sicher gilt, eignet es sich ebenfalls für Nutzungsszenarien mit Sicherheitsaspekten. Ein einziges Smartphone könnte als Schlüssel für beliebig viele Zugangskontrollen fungieren, die bisher durch herkömmliche Schlüssel oder RFID-Kontaktkarten geregelt wurden. Anstatt mehrere Schlüssel für Auto, Wohnungstür und Bürotür mit sich zu führen, benötigt man dann nur noch das Smartphone. Auch das tägliche Bezahlen von klei-

[3]Eine Smartcard ist eine Plastikkarte im EC-Karten-Format mit eingebautem Schaltkreis, die optional über einen eigenen Mikroprozessor, Hardware-Logik oder Speicher verfügt. `http://de.wikipedia.org/w/index.php?title=Chipkarte&oldid=102423633`

Abbildung 1.3: NFC-Tag im Museum of London, Quelle:`http://goo.gl/LPwXt`, abgerufen am 28.06.2012

nen Beträgen ist nun innerhalb von Sekundenbruchteilen möglich [8]. Die Liste an Beispielen ließe sich beliebig weiterführen. So sinnvoll die Nutzung erscheint, so verwunderlich muss die bisher zurückhaltende Umsetzung wahrgenommen werden.

1.2 Problemstellung

NFC besitzt das Potential an vielen Stellen in das tägliche Leben eingebunden zu werden. Doch bisher wird es lediglich in wenigen Pilotprojekten eingesetzt. Nun stellt sich die Frage, wieso die Integration von NFC noch nicht weiter vorangeschritten ist. Mit der Markteinführung des Samsung Galaxy S als erstem Smartphone mit integrierter NFC-Technologie war nur ein Bruchteil der Smartphone-Nutzer im Besitz eines NFC-fähigen Telefons. Dementsprechend hoch war die Skepsis bei Entwicklern, ob sich die Technologie überhaupt durchsetzen könnte. Seit Anfang 2012 kündigen jedoch immer mehr namhafte Hersteller mobiler Geräte Smartphones mit NFC-Funktionalität an. Alleine im Mai 2012 können Hersteller wie Sony, Samsung, HTC, LG, Huawei und Intel mehrere NFC-Smartphones in ihrem Portfolio vorweisen.[4]

Nun liegt es an den Entwicklern ihre Applikationen um sinnvolle NFC-Funktionalitäten zu erweitern. Doch die Integration gestaltet sich schwieriger als gedacht, denn es gibt viele technische Spezifikationen zu beachten, was auf Entwickler abschreckend wirken kann. Schnittstellen der Android-API ändern sich und bieten je nach Betriebssystemversion unterschiedliche NFC-Funktionalitäten. Funktionen, die man in der neuesten Android-Version

[4] `http://www.nfcworld.com/nfc-phones-list/`, abgerufen am 28.06.2012

Abbildung 1.4: Oben Links: Tischaufsteller in Restaurant (Quelle: `http://goo.gl/oZSc7`, abgerufen am 28.06.2012), oben rechts: Kontaktpunkt des Rhein-Main-Verkehrsbundes (Quelle: `http://goo.gl/bQj6o`, abgerufen am 28.06.2012), unten: Visitenkarte mit NFC und QR-Code-Funktion (Quelle: `http://goo.gl/xGvjw`, abgerufen am 28.06.2012)

kennen gelernt und als notwendig erachtet hat, fehlen in älteren Versionen und erfordern Mehraufwand, sobald sie nachgebaut werden müssen. All diese Probleme können bewirken, dass die Programmierung von NFC unter Android als zu aufwendig betrachtet wird - und schlussendlich als inpraktikabel. Dieses Problem scheint nicht neu zu sein, denn es wurden bereits einige Frameworks auf Basis der Android-API von third-party-Entwicklern erstellt (siehe Kapitel 2). Leider beschränken sich diese Frameworks nur auf ein individuelles Teilproblem. Die Einen vereinfachen zwar einen Teilaspekt der NFC-Entwicklung, vernachlässigen aber die Rückwärtskompatibilität wichtiger Funktionen. Andere Entwickler kapseln in ihrem Framework lediglich neue Funktionalitäten. Bisher erfüllt kein Framework die Vorraussetzungen einer umfangreichen und trotzdem einfach anzuwendenden NFC-Erweiterung, welche es schafft, die Akzeptanz von Entwicklern und somit die Anzahl von NFC-Applikationen zu steigern.

Doch auch eine Vielzahl an verfügbaren NFC-Applikationen bringt keinen Mehrwert, wenn ihre Überlegenheit gegenüber bestehenden Standards dem durchschnittlichen Android-

Nutzer nicht klar ersichtlich sind. Bisher wurden z.B. für das Identifizieren von Objekten meist QR-Codes[5] verwendet, die von Smartphones mit eingebauter Kamera gescannt wurden. NFC ermöglicht genau diese Erkennung von Objekten viel komfortabler, jedoch sind Benutzer bisher nur den Barcode gewöhnt. Sie müssen also erst noch mit dieser neuen Technologie vertraut gemacht werden. Erst wenn sie NFC als Vorteil erkennen und die Verbindung zwischen Smartphone und NFC-Tag in verschiedenen Szenarien wiedererkennen, werden ältere Technologien zu Gunsten von NFC aufgegeben.

Man kann festhalten: Es gibt noch einige Probleme, mit denen NFC als neue Technologie zu kämpfen hat. Ziel dieser Arbeit ist eine Analyse der bisherigen Marktakzeptanz NFC-basierter Technologien und der darauf Einfluss nehmenden Faktoren. Zudem werden wichtige Lösungsansätze für eine erfolgreichere Verbreitung NFC-basierter Software ermittelt und anhand von Prototypen, basierend auf Googles Betriebssystem Android, exemplarisch umgesetzt.

1.3 Ziel und Aufbau dieser Arbeit

Wie bereits einleitend definiert, ist es Ziel dieser Arbeit zu untersuchen, mit welchen Voraussetzungen sich NFC als innovative Technik schneller am Markt behaupten kann. Grundlage für diese Überlegung stellt die sogenannte Diffusionstheorie dar, die in der ersten Version bereits 1962 von Everett Rogers entwickelt wurde [9]. Darauf aufbauend existieren fünf Stufen, die den Prozess der Adoption einer Technologie in der Bevölkerung beschreiben[6].

Knowledge - von einer Innovation erfahren

Persuasion - von einer Innovation im positiven oder negativen Sinn überzeugt werden

Decision - sich für oder gegen eine Innovation entscheiden

Implementation - die Innovation implementieren

Confirmation - die Innovationsentscheidung bestätigen und weiter nutzen oder rückgängig machen

In der ersten Phase (Knowledge) erfährt die Person von der Innovation und leitet daraus ein eigenes Bedürfnis ab. Dabei kann sich das Bedürfnis, wie es bei NFC der Fall ist, erst nach der Innovation zeigen. Mit NFC wird vorerst kein Problem behoben, welches bereits bei den Nutzern aktiv besteht. Erst wenn der Nutzer die Innovation kennt, ergeben sich Erleichterungen für ihn. So war zum Beispiel der Datenaustausch zwischen zwei Geräten auch vor NFC möglich, ebenso die Nutzung von Terminals oder Smartcards. Allerdings waren die technischen Hürden so hoch, dass es zu keinem Einsatz im Alltag kam. Diese Vorgänge werden durch NFC bequemer und schneller. Bequemlichkeit und Zeitmangel sind zwei Eigenschaften der Menschen, die das Bedürfnis für NFC in der heutigen Zeit rechtfertigen.

[5] QR-Codes sind zweidimensionale Codes, die Daten binär in Form von schwarzen und weißen Rechtecken als quadratische Matrix darstellen.

[6] `http://de.wikipedia.org/w/index.php?title=Diffusionstheorie&oldid=85122375`, abgerufen am 28.06.2012

In den beiden weiteren Phasen der Adaption befinden sich die Elemente, in denen die Autoren dieser Arbeit das meiste Potenzial sehen einzugreifen und die Verbreitung von NFC zu vereinfachen, bestenfalls zu beschleunigen. In der Persuasion-Phase (deutsch: Überredung, Überzeugung) bildet sich der Nutzer einen eigenen Eindruck der Innovation. Sein Gefühl spielt dabei eine große Rolle, rationale Argumente treten in den Hintergrund. Er macht sich jedoch auch Gedanken um die Kompatibilität, die Komplexität und die Vorteile, die er durch die Innovation erlangen würde [10]. Der technische Aspekt von NFC, die Übertragung von Daten zwischen zwei Geräten, nimmt hierbei nicht die Rolle der Innovation ein. Die Innovation besteht aus den unterschiedlichen Nutzerszenarien, die durch NFC alltagstauglich werden. Da sich Nutzer über die reine technische Komponente von NFC nur wenige Gedanken machen können, ist es an dieser Stelle wichtig, passende Nutzerszenarien zu erstellen und zu entwickeln, um das Interesse an der Innovation zu wecken. Wenn durch die Existenz passender Nutzerszenarien das Interesse geweckt wurde, ist es wichtig, dass die Umsetzung der Szenarien möglichst nah an den perfekten Vorstellungen der Nutzer liegt und sie diese am Ende auch testen können. Dieser Schritt ist die Decision-Phase [10].

Basierend auf der Annahme, dass nicht NFC an sich, sondern die Anwendungsfälle der Technologie die Innovation ausmachen, ist es wichtig, diese technisch möglichst gut und schnell zu entwickeln. Da sich diese Masterarbeit auf Smartphones mit NFC konzentriert, kann man bei der Innovation, einem NFC-Nutzerszenario, von einer Applikation (kurz: App) reden - jedenfalls bei dem Teil der Anwendung, mit dem der Endanwender in Berührung kommt. Während des Erstellungsprozesses einer App haben sich zwei Aspekte als besonders wichtig herausgestellt, die es aufzuzeigen und nach Möglichkeit zu vereinfachen gilt.

Auf der einen Seite steht die Programmierung der App. In diesem Bereich ist es wichtig, dass die richtigen Werkzeuge bereitgestellt werden, damit Entwickler NFC in die eigenen Applikationen einbauen und somit neue Ideen umsetzen können. Denn nur durch das richtige Angebot für unterschiedliche Nutzerkreise kann das Interesse geweckt werden. Damit NFC aber nicht in jedem Szenario als neue Technologie erlernt und entdeckt werden muss, gibt es eine zweite Seite, die Grundsätze für die Bedienung und die Handhabung von NFC-Apps. Muss es außerhalb des App-Ökosystems noch weitere Maßnahmen geben, um dem Endanwender NFC näher zu bringen? Muss der Nutzer NFC optisch erkennen können? Was sind dessen Erwartungen an ein solches System? Damit beide Seiten ausreichend beleuchtet werden können, wird im Laufe der Arbeit immer zwischen diesen beiden Bereichen unterschieden: die Sicht des Entwicklers auf NFC und die Sicht des Endanwenders auf NFC.

1.3.1 Aus Sicht der Entwickler: Hilfe durch ein neues Framework

Die Akzeptanz von Entwicklern ist eine treibende Kraft für die Verbreitung einer neuen Technologie (vgl. Kapitel 1.3). Um NFC als eine neue Technologie zu einer höheren Zustimmung in der Gemeinde der Android-Entwicklung zu verhelfen, wird im Rahmen dieser Masterarbeit SimpleNFC entwickelt. SimpleNFC, realisiert als high-level-Framework, stellt dafür die wichtigsten Basisfunktionen wie das Senden und Empfangen von NFC-Nachrichten (sowohl bi- als auch unidirektional) zur Verfügung. Darauf aufbauend bietet es vordefinierte NFC-Nachrichtentypen, welche der eigenen Applikation automatisch zugeordnet werden können. Moderne Entwurfsmuster, wie das Observer-Pattern, ermöglichen eine gezielte Überwachung

definierter NFC-Nachrichten und erweitern diese um einen direkten Kontextbezug. Entwickler können somit bereits bestehende Methoden auf die NFC-Funktionalität anwenden. Ziel ist es, ein Werkzeug zu erschaffen, welches durch eine abstrahierte Schicht die technischen Spezifikationen vor dem Entwickler verbirgt und ihm stattdessen einen Zugang zur Technologie gewährt, wie er ihn bereits aus verschiedenen Bereichen der Programmierung kennt.

1.3.2 Aus Sicht der Nutzer: Anwendungen nach ihren Bedürfnissen

Obwohl schon einige Versuche von Unternehmen dieser Industrie unternommen wurden, die Vorteile von NFC aufzuzeigen, scheint der Endanwender noch nicht überzeugt. In dieser Arbeit soll eine Analyse erfolgen, warum das Verlangen der Endanwender nach NFC noch nicht ausgeprägt ist und mit welchen Mitteln Nutzern NFC nähergebracht werden kann. Dazu wird untersucht, wie viele Personen schon mit dem Begriff NFC vertraut sind und welche Anwendungsfälle ihnen helfen würden, sich mit der Technologie auseinanderzusetzen. Zudem wird analysiert, wie benutzerfreundlich aktuelle Anwendungen mit NFC sind und ob es nötig ist, ein einheitliches Design für NFC zu entwickeln und zu gebrauchen, um einen Wiedererkennungseffekt zu erzielen. Anhand von zwei Prototypen wird geforscht, wie die Nutzer auf grundlegende Anwendungsfälle reagieren. Darauf aufbauend sollen Schlussfolgerungen gezogen werden, wie Prozesse optimiert werden können, um einen möglichst großen Nutzerkreis anzusprechen und für die Technologie zu begeistern, so dass die Nachfrage nach NFC-Geräten und NFC-Anwendungen langfristig steigt.

1.4 Aufteilung zwischen den Autoren

Da diese Masterarbeit von zwei Autoren verfasst wurde, folgt eine Auflistung und Aufteilung der geschriebenen Kapitel auf die beiden Autoren. Beide Studenten haben sich ab dem Ende des Bachelor-Studiums auf die Entwicklungen im mobilen Bereich konzentriert. Nicht nur das Interessengebiet der beiden war hier zu verorten, sondern auch im Job neben dem Studium wurden hauptsächlich mobile Applikationen entwickelt. Unterschieden hat sich immer nur die unterste Ebene - die unterschiedliche Fokussierung auf Android und iOS. Da sich Benjamin Rühl auf Android als Betriebssystem spezialisiert hat, sind in dieser Arbeit viele der tiefgreifenden Technologien rund um Android und dessen Implementierung von NFC von ihm beschrieben. Die Umsetzung der Prototypen und die Entwicklung des Frameworks entstanden in gemeinsamer Arbeit.

Benjamin Rühl, Matrikelnummer 805506:

- Einleitung
- Stand der Technik - Untersuchung aktueller Frameworks
- Grundlagen von NFC - NFC-Entwicklung unter Android
- SimpleNFC - Entwicklung eines NFC-Frameworks für Android

- Ergebnisse und Ausblick

Dennis Becker, Matrikelnummer 805865:

- Einleitung
- Stand der Technik - NFC Pilotprojekte am Markt
- Grundlagen von NFC
- Potenzielle Nutzer von NFC: Kenntnisse und Bedürfnisse
- Ergebnisse und Ausblick
- Zusammenfassung

Kapitel 2

Stand der Technik und Probleme

2.1 Untersuchung aktueller Frameworks

Ein wichtiger Faktor für den Erfolg einer Technologie ist die Akzeptanz der Entwickler. Diese zu erreichen kann mitunter ein schwieriges Unterfangen sein. Um möglichst vielen Entwicklern die Verwendung einer neuen Technologie zu ermöglichen, ist es sinnvoll, deren Anwendung so einfach wie möglich zu gestalten. Doch wie kann man dieses Ziel verwirklichen? Der erste Schritt ist eine umfangreiche und sinnvoll aufgebaute Programmierschnittstelle (API) der zu verwendenden Technologie. Werden bereits existierende Systeme durch neue Technologien ergänzt, wie es bei Android und NFC der Fall war, erfolgt die Bereitstellung einer API durch das Wirtssystem, dem Betriebssystem. Charakteristisch für solche APIs ist ihre Unveränderlichkeit durch Dritte. Veränderungen können lediglich vom Hersteller des Betriebssystems in Form von Betriebssystem-Updates eingepflegt werden. Wie regelmäßig diese Updates erscheinen, ist meist nicht vorhersehbar. Sicher ist jedoch, dass bei jedem Betriebssystemupdate die Fragmentierung der auf dem Markt befindlichen Betriebssystemversionen zunimmt. Grund dafür sind die Lebenszyklen der Geräte, die mit einer bestimmten Betriebssystemversion enden. Somit kann durch Updates des Betriebsystems nicht immer garantiert werden, dass alle Geräte in den Genuss von verbesserten Software-Schnittstellen kommen. Wie bereits angedeutet, können Drittentwickler keinen Einfluss auf bestehende Programmierschnittstellen nehmen. Als zweiten Schritt bietet sich die Entwickelung von Frameworks an. Dieses Kapitel beschäftigt sich mit der Vorstellung bereits existierender Frameworks und deren Einteilung in zwei Gruppen. Zudem werden die Frameworks analysiert und schlussendlich aufgezeigt, warum keines der bisher existierenden Frameworks das volle Potential ausschöpft, um die Entwicklung von NFC-Apps auf Android einfacher zu gestalten.

Frameworks lassen sich grob in zwei Gruppierungen unterteilen: Low-level- und High-level-Frameworks. Sie unterscheiden sich hauptsächlich in der Tiefe ihres Abstraktionsgrades. High-level-Frameworks verfügen, wie der Name vermuten lässt, über einen hohen Abstraktionsgrad. Dies bedeutet: Sie kapseln umfangreichen und komplizierten Code für definierte Anwendungsszenarien und bilden somit eine Art Wrapper der bestehenden Schnittstellen. Programmlogik, die vorher mühsam in vielen Zeilen Code programmiert werden musste, kann nun durch den Aufruf weniger Zeilen umgesetzt werden. Die Einarbeitungszeit in eine neue

Technologie verringert sich, da Entwickler im besten Fall viel weniger technische Grundlagen verstehen sondern nur noch anwenden müssen. Dies vereinfacht und beschleunigt zum Einen den Entwicklungsprozess, zum Anderen profitieren Entwickler von einer geringeren Komplexität, die den Code besser lesbar und wartbarer macht [11]. All diese Vorteile werden dadurch erreicht, dass high-level-Frameworks den Fokus auf vordefinierte Anwendungsszenarien legen. Erfolg und Misserfolg eines Frameworks dieser Kategorie definieren sich also durch das Erkennen von sinnvollen Nutzungsszenarien.

Die Gruppe der low-level-Frameworks realisieren weniger abstrakte Funktionen. Meist bilden sie die Basis für weiterführende Frameworks und zeichnen sich durch eine hardwarenahe Programmierung aus. Im Gegensatz zu high-level-Frameworks unterliegen sie keinen Einschränkungen im Bezug auf bestimmte Nutzerszenarien. Vielmehr bieten sie einen großen Baukasten an Funktionen an, welche miteinander verknüpft, alle in ihrem Fachgebiet anfallenden Szenarien abdecken können. Diese große Freiheit geht mit einer wachsenden Komplexität einher. Für Entwickler wird Wissen über Einzelheiten der Technologie und eine gewisse Expertise vorausgesetzt. Erst mit einem gewissen Maß an Erfahrung können Entwickler effektiv mit low-level-Framworks arbeiten [11].

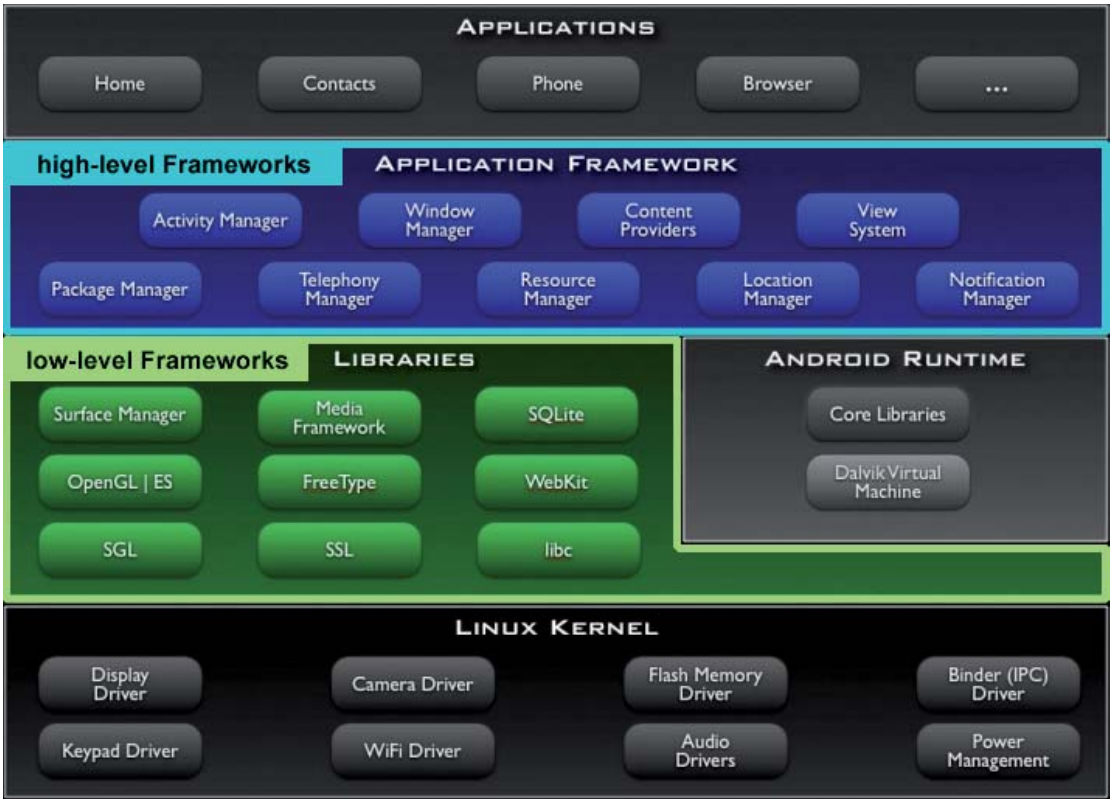

Abbildung 2.1: Abbildung der einzelnen Softwareschichten des Android Betriebssystems, Quelle: `http://goo.gl/OCNm`, abgerufen am 28.06.2012

Gliedert man beide Arten von Frameworks in die Architektur von Android ein und stellt

diese bildlich dar, würde man den Unterschied ebenfalls an der Zugehörigkeit zu den unterschiedlichen Komponentenschichten erkennen. Wie in Abbildung 2.1 zu sehen, werden low-level-Frameworks in der Schicht der Bibliotheken (Library) abgebildet. Die Nähe zur Hardware wird ebenfalls durch die Wahl der Programmiersprache deutlich. Üblicherweise werden Android-Applikationen in der höheren Programmiersprache Java programmiert. Alle Bibliotheken der Libraries-Schicht sind jedoch in C bzw. C++ geschrieben. High-level-Frameworks findet man dagegen in der Gruppe der Application Frameworks. Sie stellen auf Basis der darunterliegenden Libraries Schnittstellen für definierte Anwendungsszenarien zur Verfügung, wie das Aufbauen eines Telefongespräches oder die Bestimmung des aktuellen Standpunktes. Bibliotheken dieser Kategorie werden, wie Applikationen auch, in Java geschrieben.

Die Tatsache, dass bereits einige Projekte rund um NFC existieren, zeigt, dass die Technologie per se von Entwicklern angenommen wurde. Doch noch betrachtet jedes Framework einen anderen Blickwinkel der Technologie, so dass einige Bibliotheken nicht von mobilen Entwicklern genutzt werden können und andere zwar im Ansatz vielversprechend aussehen, letztendlich jedoch einige Fragen offen lassen. Im Folgenden werden die bekanntesten und aussichtsreichsten Projekte für unterschiedliche Plattformen, Anwendungsszenarien und Zielgruppen vorgestellt.

Open NFC

Der bekannteste Vertreter der low-level-Frameworks ist Open NFC[1]. Wie der Name schon vermuten lässt, handelt es sich dabei um eine freie Softwareimplementierung der NFC-Technologie. Der größte Vorteil von Open NFC ist das reichhaltige Angebot an unterstützter Hardware. Es kann sowohl in allen bekannten Desktop-Betriebssystemen (Mac OS X, Windows und Linux) als auch in eingebetteten Systemen (Nuclus) oder mobilen Geräten (Android) eingebunden werden. Das macht Open NFC vor allem bei Entwicklern beliebt, die Applikationen für verschiedene Hardwaresysteme entwickeln. Dazu können Entwickler auf einen beachtlichen Umfang von Funktionen zugreifen, die in drei Gruppen unterteilt sind. Ähnlich wie bei der Einteilung in Framework-Kategorien sind auch die drei Schichten der Interfaces anhand ihres Abstraktionsgrades unterteilt.

Low-level Interfaces bieten Schnittstellen für einen direkten Zugriff auf NFC Controller oder das Secure Element (siehe Kapitel 3.6).

Intermediate-level Interfaces beinhalten Interfaces für das Emulieren von kontaktlosen Chipkarten und den Zugriff auf unformatierte Daten eines Tags.

High-level Interfaces beinhalten die meiste Funktionalität, so zum Beispiel das Auslesen und Beschreiben von Tags. Neben Funktionen zum Parsen und Erstellen von NDEF-Messages [2], ermöglichen high-level Interfaces ebenfalls die Kommunikation mit anderen Geräten. Tags können beschrieben oder ausgelesen werden; Informationen können

[1] `http://open-nfc.org/`, abgerufen am 28.06.2012

[2] NDEF-Messages sind ein vom NFC-Forum definiertes Austauschprotokoll für NFC-Nachrichten [12]

mittels peer-to-peer-Funktionen zwischen zwei aktiven Geräten ausgetauscht werden; virtuelle Tags können emuliert werden. Open NFC bietet durch diese Schicht ebenfalls die Möglichkeit direkte Verbindungen zwischen zwei aktiven Geräten aufzubauen, welche im Anschluss für den Datenaustausch genutzt werden können.

Vergleicht man den Funktionsumfang zwischen Open NFC und der NFC-Implementierung von Android 4.0.4 bietet Open NFC nur wenige Möglichkeiten mehr an. Darunter fallen eine größere Anzahl an unterstützenden NFC-Protokollen, Tags, Smartcards sowie die Möglichkeit direkte Verbindungen zwischen zwei aktiven Geräten aufbauen zu können (Connection Handover). Letztere Funktion bietet einen erheblichen Vorteil gegenüber der NFC-Implementierung in Android. Aufgrund der geringen Übertragungsgeschwindigkeit von 424 kbit/s mittels NFC ist es nicht möglich größere Dateien zu übertragen[5]. Sogar Bilddateien müssen über einen anderen Kanal übermittelt werden. Dieses Problem wird durch Connection Handover gelöst. Dafür werden lediglich die für das Aufbauen einer Verbindung nötigen Daten mittels NFC ausgetauscht. Die eigentliche Kommunikation findet im Anschluss über einen eigenständige Technologie statt. Welche Technologie zum Austausch der Daten benutzt wird, hängt von den teilnehmenden Geräten ab. Das vom NFC-Forum verfasste Handover Protocol beschreibt dazu zwei Parteien [13]:

Handover Requester Startet eine Handover-Anfrage mittels NDEF-Nachricht. Die Nachricht beinhaltet eine priorisierte Liste von Verbindungstechnologien, die von dem NFC-Gerät unterstützt werden.

Handover Selector Antwortet auf eine Handover-Anfrage. Dazu wird aus der Liste der gewünschten Technologien und der eigenen Liste an verfügbaren Technologien eine gemeinsame Schnittmenge gebildet. Die daraus resultierende Liste an Technologien wird ebenfalls per NDEF-Nachricht an den Requester weitergegeben.

Weiterhin unterscheidet das Protokoll zwischen zwei verschiedenen Verbindungsarten:

Negotiated Handover Beschreibt die Kommunikation zwischen zwei Geräten, die beide direkt über eine Implementierung der NFC-Technologie verfügen.

Static Handover Beschreibt den Aufbau einer Verbindung zwischen einem NFC-Gerät und einem externen Gerät, welches selbst über keine NFC-Technologie verfügt.

Beim üblichen Anwendungsfall wird die Verbindung, wie in Abbildung 2.2 zu sehen, zwischen zwei aktiven Geräten aufgebaut. Beide Geräte verfügen über die NFC-Technologie und eine Liste an verfügbaren Technologien zum Aufbau einer direkten Verbindung. Dabei fragt der Requester nach verfügbaren Technologien, der Selector antwortet und die Verbindung kann aufgebaut werden.

Besonders praktisch ist die Möglichkeit Verbindungen mit Geräten herzustellen, die selbst über keine NFC-Technologie verfügen. Dazu können beispielsweise alle Zugangsdaten zu einem externen Gerät auf einem einfachen Tag gespeichert werden. Der Tag nimmt dabei die Rolle des Selectors ein. Ein NFC-Gerät liest dann, in der Rolle als Handover Requester,

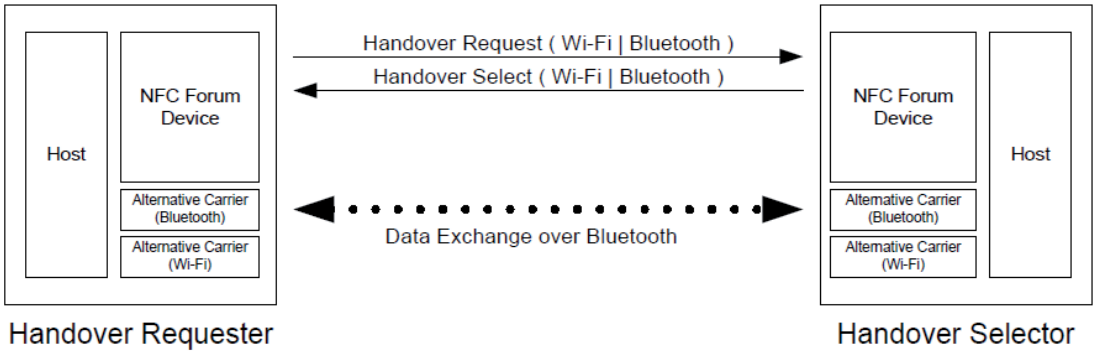

Abbildung 2.2: Negotiated Handover, Quelle: [13]

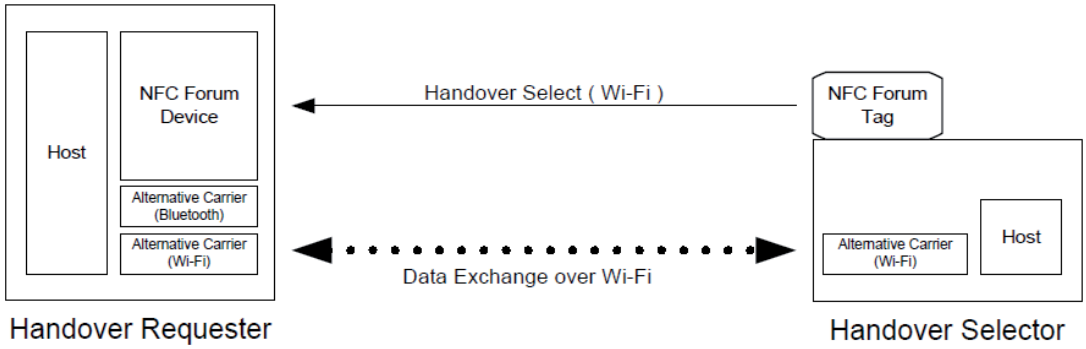

Abbildung 2.3: Static Handover, Quelle: [13]

den Tag aus und baut eine Verbindung mit dem referenzierten Gerät auf (siehe Abbildung 2.3).

Leider ist es für Android-Entwickler nicht ohne einen beträchtlichen Mehraufwand möglich, Open NFC zu nutzen. Ein eigens dafür angelegter Quick Porting Guide[3] beschreibt alle Schritte, die nötigt sind, um Open NFC in Android einzubinden. Auf fünfzehn Seiten wird dort beschrieben, woher Entwickler den Quellcode von Open NFC bekommen, wie ein neuer Kernel und zum Schluss ein ganzes Android-System inklusive Open NFC kompiliert werden muss.

libnfc

Einen ähnlichen Ansatz verfolgt libnfc[4]. Genau wie Open NFC implementiert auch libnfc seine eigene NFC-Logik. Dabei ist die Liste der kompatiblen Hardware jedoch kürzer. Bisher lässt sich libnfc nur in Linux-Derivate, Mac OS X und Windows einbinden. Da es jedoch unter

[3] `http://goo.gl/sYqSY`, abgerufen am 28.06.2012
[4] `http://www.libnfc.org/`, abgerufen am 28.06.2012

der GNU Lesser General Public License[5] veröffentlicht wurde, kann sich jeder an der Entwicklung beteiligen. Trotzdem ist der Funktionsumfang, verglichen mit Open NFC, geringer. So gibt es beispielsweise keinerlei Funktionen, die dem Connection Handover von Open NFC vergleichbar wären. Auch das NDEF-Protokoll wurde nicht implementiert. Wie jedes andere low-level-Framework muss auch libnfc händisch kompiliert werden. Der low-level-Charakter tritt bei libnfc verstärkt in den Vordergund. Es bleibt ein Framework welches durch eine einheitliche Schnittstelle und verschiedenste Hardwaretreiber den Zugriff auf rudimentäre NFC-Funktionalitäten liefert.

Zwar bieten beide Frameworks den Vorteil einer plattformübergreifenden Implementierung des NFC-Standards und beschränken sich nicht nur auf Android als Betriebssystem bzw. Java als Programmiersprache, jedoch ist der Aufwand für die Einbindung solcher Frameworks in das Betriebssystem eines Smartphones mit viel Aufwand verbunden. Dafür wäre es nötig den Quellcode des Betriebssystems herunterzuladen, das Framework einzubinden und den resultierenden Quellcode zu kompilieren. Anschließend kann man das kompilierte System nur auf speziellen Geräten installieren, deren Software vorher entsperrt[6] wurde, um den Zugang für Dritte zu ermöglichen. Die Zielgruppe dieser Art Frameworks setzt sich überwiegend aus Hardware-Herstellern zusammen, die aus persönlichen Gründen eine Alternative zu Googles eigener NFC-Implementierung bevorzugen. Es ist zu keinem Zeitpunkt möglich, bestehende Android-Systeme um Frameworks dieser Art zu erweitern. Dafür werden Bibliotheken eines höheren Abstraktionsgrades benötigt, die der Gruppe der high-level-Frameworks.

Der wesentliche Vorteil von high-level-Frameworks liegt in der Kapselung von Nutzungsszenarien. Im besten Fall enthalten sie sogar Funktionen einer neueren API-Version, die das Gerät aufgrund seines Lebenszyklus nicht mehr von offizieller Seite erlangen würde. Vor allem diese Erweiterung und Simplifizierung einer Technologie verschafft Entwicklern einen großen Mehrwert und rückt die Technologie mehr ins Rampenlicht. Dafür ist weder eine Veränderung des Betriebssystems noch ein besonderes Endgerät vonnöten. Der aus solchen Frameworks resultierende Vorteil besteht meist aus einer Vereinfachung der Programmierung bestimmter Features unter Berücksichtigung definierter Nutzungsszenarien. Dabei wird die bestehende Betriebssystem-API verwendet, um Funktionsweisen für detaillierte Anwedungsfälle bereitzustellen. Dadurch können zum Einen Applikationen, deren Funktionsumfang diese Nutzungsszenarien beinhalten sollen, von ihren Entwicklern mit weniger Aufwand umgesetzt werden. Zum Anderen ermöglicht es komplexe Routinen zu kapseln, um Entwicklern einen vereinfachten Zugang anzubieten.

AllJoyn

Ein bekannter Vertreter der zweiten Kategorie ist AllJoyn[7]. Für das Projekt zuständig ist das Qualcomm Innocation Center, ein Tochterunternehmen des bekannten Herstellers von Digitalmobilfunktechnologie Qualcomm. AllJoyn beschränkt seinen Funktionsumfang auf die direkte Kommunikation zwischen zwei Geräten. Dabei erfolgt der Verbindungsaufbau der

[5] `http://goo.gl/fjquK`, abgerufen am 28.06.2012

[6] Entsperrte Smartphones ermöglichen das Aufspielen von selbstkompilierten Android-Versionen oder Software, die nicht von Google dafür vorgesehen wurde.

[7] `https://www.alljoyn.org/`, abgerufen am 28.06.2012

beiden Geräte über NFC, der spätere Datenaustausch kann entweder über Bluetooth oder WiFi erfolgen. Als mögliche Einsatzgebiete stellt Qualcomm „multi-player gaming“, „social media sharing“oder „multi-user productivity tools„vor. Um AllJoyn in die eigene Applikation einzubinden, gibt es drei verschiedene Prozedere. Bei der Installation als. . .

. . . Android Service kann die Datenverbindung lediglich über WiFi aufgebaut werden. Dabei wird AllJoyn als eigenständige Applikation installiert und unterliegt dem Ressourcenmanagement des Betriebssystems. Dies bedeutet, dass die Applikation ebenso wie jede andere Applikation vom Betriebssystem geschlossen werden kann, sollte einmal nicht genügend Arbeitsspeicher frei sein.

. . . Android Dämon mit WiFi-Unterstützung erfolgt der Datenaustausch ebenfalls nur über WiFi. Bei dieser Art der Implementierung wird der Dämon[8] automatisch nach jedem Boot-Vorgang gestartet. Voraussetzung für diese Implementierung sind ein entsperrtes Smartphone und Kenntnisse über das Kommandozeilenprogramm von Android. Zielgruppe für diese Art der Implementierung sind Original-Equipment-Manufacturer (kurz OEM) oder Android-Entwickler, die eine gewisse Expertise vorweisen können.

. . . Android Dämon mit WiFi- und Bluetooth-Unterstützung ist zwar sowohl Bluetooth als auch WiFi als Datenverbindung möglich, die Einrichtung ist allerdings so kompliziert, dass Qualcomm zu einen der beiden vorherigen Implementationen rät. Diese Implementation setzt eine eigene kompilierte Android-Version voraus, die auf einem entsperrten Gerät installiert werden muss.

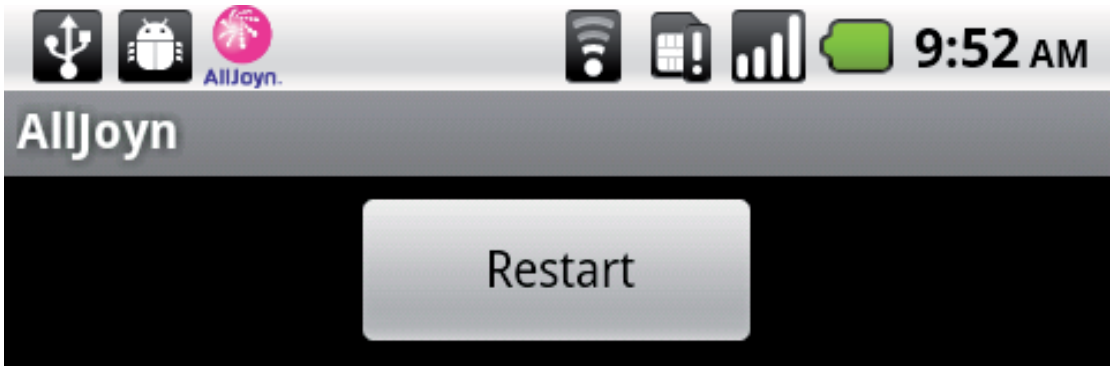

Abbildung 2.4: Implementierung von AllJoyn als eigenständige Applikation, Quelle: `http://goo.gl/80ut5`, abgerufen am 28.06.2012

Obwohl man AllJoyn zur Kategorie der high-level-Frameworks zuordnet, ist Qualcomm auf die Akzeptanz von Erstausrüstern angewiesen. Für Entwickler von mobilen Applikationen ist das Framework nicht interessant. Aufgrund der verschiedenen Implementationen ist ein zuverlässiger Einsatz von AllJoyn entweder nicht gewährleistet oder zu kompliziert bzw. technisch zu aufwendig.

[8] Dienste die auf Unix-Systemen im Hintergrund laufen und einen Service zur Verfügung stellen.

nfctools

Weniger aufwendig ist die Nutzung von Adrian Stabiszewski's nfctools. Das Framework ist eine Sammlung von Java-Klassen, die die Nutzung von NFC vereinfachen soll. Einsetzbar ist das Framework auf allen Systemen, die eine Java-Runtime zur Verfügung stellen. Leider funktioniert die implementierte NDEF-Funktionalität nur in Verbindung mit drei Smartcards: Mifare classic 1K, Mifare classic 4K und Ultralight/C. Auch fehlt es dem Framework an einer Dokumentation, die Entwicklern den Einstieg erleichtern. Beziehen kann man nfctools von GitHub[9].

nfcpy und CocoaTag

Projekte, die einen ähnlichen Ansatz verfolgen, sind nfcpy[10] und CocoaTag[11]. nfcpy stellt die NFC-Funktionalität in der Programmiersprache Python zur Verfügung. Zwar werden sowohl das NDEF-Protokoll als auch verschiedene Tag-Spezifikationen unterstützt, der Zugriff auf Tags kann allerdings nur durch sechs Lesegeräte (teils mit Einschränkungen) erfolgen. CocoaTag bietet die gleiche Funktionalität für Apples hauseigene Programmierschnittstelle Cocoa. Das Projekt kann auf github ausgecheckt und später als eigenes Projekt in Apples IDE(integrated development environment zu deutsch Integrierte Entwicklungsumgebung) XCode importiert werden. Sowohl CocoaTag als auch nfcpy bieten nur einen rudimentären Zugriff auf Funktionen von NFC. Zwar werden die wichtigsten Protokolle unterstützt, darüber hinaus nehmen diese Frameworks den Entwicklern aber keine Arbeit ab.

EasyNFC

Das bekannteste und vielversprechenste high-level-Framework ist EasyNFC[12]. Entwickelt und betreut wird es von Mitarbeitern und Studenten der Universität Stanford. Eines der Ziele von EasyNFC ist die Vereinfachung der NFC-Entwicklung unter Android. Dazu besteht das Framework aus wenigen Klassen, die folgende Funktionalität bieten:

- Kommunikation zwischen zwei aktiven Geräten
- Beschreiben von NFC-Tags
- Lesen bzw. Empfangen von NFC-Nachrichten

EasyNFC kapselt dafür Funktionen der Android-API und vereinfacht somit die Kommunikation über NFC. Einschränkungen, die Android von Natur aus mit sich bringt, wurden erkannt und zum größten Teil behoben. EasyNFC ermöglicht eine Entwicklung nach bestehenden Standards, so können Programmierer zum Beispiel einen EventHandler für Ndef-Nachrichten registrieren.

[9] `https://github.com/grundid/nfctools`, abgerufen am 28.06.2012
[10] `https://launchpad.net/nfcpy`, abgerufen am 28.06.2012
[11] `https://github.com/johnno/CocoaTag`, abgerufen am 28.06.2012
[12] `https://github.com/Mobisocial/EasyNFC`, abgerufen am 28.06.2012

Betrachtet man den Umfang des Frameworks, so erkennt man, dass EasyNFC jedoch lediglich die Kommunikation zwischen zwei aktiven Geräten bzw. einem aktiven Gerät und einem passiven Tag behandelt. Die zu übertragenden Nachrichten müssen weiterhin von den Entwicklern selbst definiert und erstellt werden. Auch bietet EasyNFC kein vereinfachtes Nachrichtensystem oder einen Kontextbezug für einzelne NFC-Nachrichten. Alle Nachrichten, die empfangen oder geschrieben werden, sind reine byte-Nachrichten ohne erweiterte Strukturvorgabe. Anbieten würde sich eine automatische Zuordnung zwischen Nachrichten und einer Ziel-Applikation, die jedoch ebenfalls fehlt. Die Grundidee hinter EasyNFC ist gut durchdacht, sie bildet jedoch nur einen Teil dessen ab, was bei der Vereinfachung des Nachrichtenaustausches mittels drahtloser Kommunikation möglich ist.

Zusammenfassend lässt sich festhalten, dass es bereits einige NFC-Frameworks für unterschiedlichste Einsatzgebiete gibt. Die wenigsten davon sind allerdings in der Lage, die Entwicklung auf mobilen Android-Geräten zu vereinfachen. Nur ein einziges verfügt über ein durchdachtes Konzept und Potential. Bisher betrachtet jedoch noch keines der genannten Frameworks Aspekte wie eine vereinfachte Nachrichtenerstellung, einen direkten Kontextbezug von NFC-Nachrichten zu Handler-Klassen oder die Rückwärtskompatibilität von Funktionalität in neueren Versionen. Diese Arbeit beschäftigt sich deshalb unter anderem damit, einen besseren Ansatz zu formulieren und anhand eines NFC-Frameworks umzusetzen.

2.2 NFC-Pilotprojekte am Markt

Schon zu Beginn des Jahres 2011 wurde 2011 von mehreren Medien als das Jahr des Durchbruchs von NFC betitelt[13,14]. Seitdem gibt es vermehrt unterschiedlichste Ansätze und Pilotprojekte von großen Unternehmen, um die Vorteile von NFC zu nutzen, zu demonstrieren und auf den Markt zu bringen. Dennoch hat sich bis zum Frühjahr 2012 wenig an der Gesamtsituation verändert. Wie im Jahr zuvor auch schon vermutet, scheint nun der Durchbruch allerdings bevorzustehen. Das Marktforschungsinstitut IMS Research sagt voraus, dass zu den 35 Millionen Smartphones mit NFC im Jahr 2011, alleine 2012 80 Millionen Geräte hinzukommen [14].

Schon 2007 führten RMV, VGF, Telekom und Nokia regional einen NFC-Ticketing-Service ein. Der Fahrgast kann vor dem Einsteigen sein Handy an einen NFC-Tag halten. Ein Programm wird automatisch geöffnet, mit dem man das Ziel seiner Fahrt auswählen und das benötigte Ticket online kaufen kann (Abbildung 2.5).

Kurze Zeit später wollte die Deutsche Bahn das System auf Basis von NFC ausweiten und alle Bahnhöfe im gesamten Langstreckengebiet mit entsprechenden Kontaktpunkten ausstatten. Unter dem Namen Touch & Travel wurde das System 2008 auf der CeBIT vorgestellt [15]. Allerdings stellte sich in der Entwicklung heraus, dass sich noch nicht genügend Anwender mit NFC-Handys am Markt befinden, so dass bei der flächendeckenden Einführung 2011 NFC noch immer nicht die treibende Technologie darstellte und die Nutzermasse für ein

[13] http://futurezone.at/future/1950-2011-wird-das-jahr-von-nfc.php, abgerufen am 28.06.2012

[14] http://www.macwelt.de/artikel/_News/375876/mwc_2011_wird_entscheidendes_jahr_fuer_bezahlung_per_nfc/1, abgerufen am 28.06.2012

Abbildung 2.5: RMV-Kontaktpunkt am Bahnhof, Quelle: `http://goo.gl/i1k0l`, abgerufen am 28.06.2012

reines NFC-System zu klein war. Die neue Lösung sieht nun vor, dass durch einen QR-Code oder die manuelle Eingabe einer Identifikationsnummer des Kontaktpunktes die Position des Nutzers bestimmt wird. Anders als bei dem Pilotsystem von 2007 steigt der Nutzer nach dem Einchecken am Einstiegspunkt in den Zug ein und checkt am Zielbahnhof aus. Erst dann muss er den gefahrenen Weg bezahlen.

Als treibende Kraft für die Verbreitung von NFC wird das Bezahlen über NFC genannt. Viele Firmen haben Interesse daran, ihre Bezahltechniken auch über NFC zu ermöglichen - nicht zuletzt gibt es bereits ein funktionierendes Geschäftsmodell. Für Deutschland kann als ein Beispiel die Sparkasse herangezogen werden: Alle 45 Millionen EC-Karten werden in Zukunft zusätzlich mit NFC ausgestattet, die meisten davon in den Jahren 2012 und 2013 [16]. Um die Karten auch nutzen zu können, haben unterschiedlichste Unternehmen (beispielsweise Edeka, Douglas und dm-drogerie) begonnen, kompatible Lesegeräte in ihre Kassensysteme zu integrieren [17]. Diese Infrastruktur kann zukünftig natürlich auch mit Smartphones genutzt werden.

Visa erhofft sich vor allem durch die Olympiade in London 2012 einen für sie erfolgreichen Start mit NFC. In Zusammenarbeit mit Samsung wird den Athleten das Samsung Galaxy S III Smartphone zur Verfügung gestellt [18]. „Bis zu den Olympischen Spielen wird es mehr als 140.000 kontaktlose Terminals in Großbritannien geben. Von dem Moment, an dem die Besucher in Heathrow landen, werden sie das Erlebnis des kontaktlosen Zahlens nutzen können, von Taxis über Einkaufsläden bis zum Olympiapark selbst“, schreibt Visa in

einer Pressemitteilung [19]. Auch auf der Webseite[15] wird bereits für die NFC-Funktionalität geworben. Dabei geht es neben der Nutzung mit dem Smartphone hauptsächlich um den Einsatz von NFC-Chips in Kreditkarten.

Die Stadt Hamburg wird im Laufe des Jahres 2012 ein Netz an NFC-Kontaktpunkten errichten, an denen Touristen Informationen zu Sehenswürdigkeiten abrufen können. Dazu wurde der französische NFC-Spezialist Connecthings[16] beauftragt. Ziel ist es, vor allem die jährlichen 270.000 Kreuzfahrtbesucher zu überzeugen, mehr Zeit in der Stadt zu verbringen [20].

Die vorgestellten Pilotprojekte zeigen auf der einen Seite, dass bereits große Projekte geplant wurden und umgesetzt werden. Auf der anderen Seite liegen noch keine Berichte über den Erfolg oder Misserfolg dieser vor. Einzig gibt es eine Studie für den italienischen Raum, die untersucht hat, ob und in welchem Umfang sich die Nutzerakzeptanz gegenüber kontaktlosen Bezahlmethoden verbessert hat [21]. Die Autoren kommen hier zu dem Schluss, dass sich Nutzer nur für diese Technik entscheiden, wenn sie eine starke Beziehung zu ihr aufgebaut haben. Das passiert allerdings nur, wenn sie gleich zu Beginn Vorteile wie beispielsweise Rabatte und Coupons im Verkaufsprozess nutzen können.

Auf dem First International Workshop on Near Field Communication im Jahr 2009 stellten Kneißl et al. das Projekt „All-I-Touch“als eine Kombination aus NFC und Lifestyle vor [22]. Bei diesen voll funktionierenden Prototypen wurden Produktinformationen am Verkaufsplatz mit Aspekten von sozialen Netzwerken verknüpft. Snowboards wurden mit NFC-Tags versehen, über die man Informationen über das Produkt finden konnte. Gleichzeitig konnte man seinen Facebook-Account angeben, über den solche Aktionen festgehalten wurden. Nach Ansicht der Autoren benötigt es weitere Anwendungsfälle, die über das Zahlungsszenario hinausgehen, um NFC weiter zu verbreiten. Sie kommen zu dem Schluss, dass die Verbindung mit Facebook der entscheidende Vorteil bei ihrer Applikation ist. So profitieren nicht nur die Händler von vielen Informationen über die Käufer, sondern auch der Nutzer erhält bessere und persönlichere Informationen über die Produkte. Bei der Präsentation des Projekts auf der CeBIT war das Feedback so positiv, dass die Autoren davon ausgehen, dass dieses Produkt auf dem Markt akzeptiert werden könnte.

Auch die Forschungsgruppe der BMW AG hat noch keine Berichte über den erfolgreichen Einsatz von NFC veröffentlicht, die über den Status von Prototypen hinausgehen. Dennoch haben sie mit ihren Prototypen gezeigt, dass der Proof of Concept erbracht ist: Die Arbeit lässt das Potenzial von NFC im Umfeld von Automobilen erkennen [23].

Da es noch keine Evaluation bisheriger im Einsatz befindlicher Systeme gibt, ist es wichtig, bei der Entwicklung neuer Anwendungen auf die Nutzerfreundlichkeit zu achten. Damit garantiert man eine möglichst hohe Akzeptanz bei den Nutzern und trägt damit zu der Verbreitung von NFC bei.

Riekki et al. [24] untersuchten das Potenzial von Benutzeroberflächen im Kontext von NFC-Applikationen. Dabei geht es um die Nutzung in sogenannten interaktiven Räumen. Da man bei NFC immer in den direkten Kontakt mit der Applikation kommt, handelt es

[15] `http://www.visa.de/de/golden_space/warum_visa/unsere_zahlungsinnovationen.aspx`, abgerufen am 28.06.2012

[16] `http://www.connecthings.com/`

sich bei diesen Räumen um das unmittelbare Umfeld des Nutzers. Unterdessen kann es unterschiedlichste Aktionen geben, die der Nutzer auslöst. Das Umfeld muss also nicht dynamisch auf den Nutzer reagieren können, sondern der Nutzer wählt gezielt die Elemente aus, mit denen er interagieren möchte [25].

Wenn NFC-Tags als grafische Icons repräsentiert werden, kann der Nutzer schnell die verfügbaren Aktionen in seinem Umfeld erkennen. Wenn die Art der Aktion sich zudem im Icon widerspiegelt, kann der Nutzer auch den Zweck des Tags erkennen und diesen schnell nutzen [26]. Viele Publikationen über Tags mit grafischen Icons gibt es bislang nicht. Bereits 2006 untersuchte Arnall [27] die Verbindung zwischen Informationen und realen Gegenständen. Välkkynen et al. [28] arbeiteten Design-Vorschläge für RFID-Tags aus. Zuletzt führten Hang et al. [29] Usability-Tests mit drei unterschiedlichen Arten von Symbolen durch. Erklärende Icons (Abbildung 2.6), die versuchen dem Nutzer näherzubringen, wie die Technologie funktioniert. Auffällige Icons (Abbildung 2.7), die dem Nutzer ins Auge fallen sollen und Aktions-Icons, die die damit verbundene Aktion darstellen (Abbildung 2.8).

Abbildung 2.6: Erklärende Symbole, Quelle: [29]

Abbildung 2.7: Auffällige Symbole, Quelle: [29]

Ein Smartposter einer Museumsaustellung mit der Möglichkeit zum Ticketkauf war die Grundlage für die Untersuchung. Die Studie kam zu dem Ergebnis, dass der Großteil der Befragten ein einzelnes, erklärendes Symbol bevorzugen, um den Vorgang einmalig zu starten und die restlichen Aktionen auf dem mobilen Gerät selbst zu erledigen. Diese Vorkenntnisse wurde unter anderem dazu genutzt, um die in Kapitel 5 genutzten Prototypen zu entwickeln.

Da es nur wenige Studien gibt, die sich damit beschäftigen, welche Szenarien von den potenziellen Nutzern überhaupt gewünscht werden, wird sich das Kapitel 5 unter ande-

Abbildung 2.8: Aktions-Symbole, Quelle: [29]

rem darauf konzentrieren, zu untersuchen, welche Arten von Anwendungen sich die Nutzer vorstellen können. Auch die beiden Prototypen in Abschnitt 5.2 geben zwar einen Anwendungsfall vor, lassen bei den Befragungen aber Freiheiten für ähnliche Szenarien.

Kapitel 3

Grundlagen von NFC

3.1 Geschichtliche Entwicklung

Wie zuvor erwähnt, ist NFC eine Technologie zum kontaktlosen Austausch von Daten über kurze Distanzen. Schon 2002 wurde NFC von NXP Semiconductors unter Zusammenarbeit mit Sony entwickelt. Technisch basiert NFC auf RFID (Radio Frequency Identification) und Chipkarten. Aus diesem Grund wird zunächst die Entwicklung dieser zwei Techniken beleuchtet, um dann auf NFC im Speziellen einzugehen.

3.1.1 Von RFID über Chipkarten zu NFC

Die Radio Frequenzy Identification dient zur Erkennung und Identifikation von elektromagnetischen Wellen. Erstmals im Zweiten Weltkrieg eingesetzt, dient es auch heute noch bei einigen Armeen zur Freund-Feind-Erkennung [30]. In den letzten Jahrzehnten wurden unterschiedlichste Lösungen für verschiedene Einsatzbereiche entwickelt. Angefangen von der einfachen Produktidentifizierung (beispielsweise beim Diebstahlschutz von Waren in Kaufhäusern) über das Bezahlen und die Nutzung von Zutrittskontrollen (beispielsweise Skipässe und Tickets) bis hin zur Tierkennzeichnung [5]. Durch den technologischen Fortschritt konnten die Sender immer kleiner und kostengünstiger produziert werden - heute gibt es RFID-Chips, die nicht mehr vom menschlichen Auge erkennbar sind, weil sie so klein sind (Abbildung 3.1).[1] Abbildung 3.2 zeigt eine Übersicht von verschiedenen Anwendungsfällen, zugehörigen Techniken und Standards, die daraus entstanden sind.

Mit der ersten Einführung einer Plastikkarte des Diners Club begann 1950 die Geschichte der Chipkarten.[2] Hauptsächlich als bargeldloses Bezahlmedium kann die Chipkarte heute aus keinem Portemonnaie mehr weggedacht werden. Zunächst gab es nur die Prägung auf der Karte als Sicherheitsmerkmal, es folgten der Magnetstreifen und später der Mikrochip. Durch die Entwicklung von Mikroprozessoren und Kryptoprozessoren erreichte die Chipkarte einen

[1] `http://www.technovelgy.com/ct/Science-Fiction-News.asp?NewsNum=939`, abgerufen am 28.06.2012

[2] `http://inventors.about.com/od/cstartinventions/a/credit_cards.htm`, abgerufen am 28.06.2012

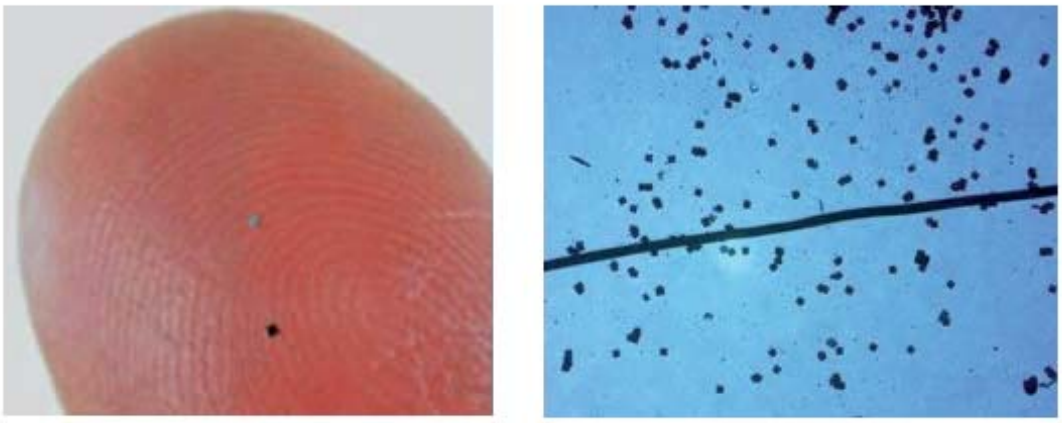

Abbildung 3.1: RFID-Chips auf einem Finger und „RFID-Puder“neben einem menschlichen Haar, Quelle: `http://goo.gl/AH6J`, abgerufen am 28.06.2012

neuen Meilenstein. 1996 wurde in Deutschland die Geldkarte als elektronische Geldbörse eingeführt [31].

NXP Semiconductors und Sony sind beides Unternehmen, die führend in der Herstellung von kontaktlosen Chipkarten sind. NXP nennt seine Produktreihe MIFARE, Sony ihre FeliCa. Diese beiden Firmen entwickelten 2002 gemeinsam NFC, was zu beiden Smartcard-Varianten kompatibel ist. Die Verschlüsselungsmethoden wurden allerdings nicht übernommen. Zusammen mit Nokia wurde 2004 das NFC Forum ins Leben gerufen, welches 2012 bereits über 160 Mitglieder zählt. Ziel des NFC Forums ist die weltweite Standardisierung von NFC. Mittlerweile gibt es viele namhafte Firmen, die sich dem Forum angeschlossen haben, um weiter an einem gemeinsamen Standard zu arbeiten. Dazu zählen zum Beispiel Microsoft, VISA, Mastercard, Samsung und Intel. Das Besondere an NFC ist, dass keine strikte Trennung mehr zwischen Lesegerät und Transponder existiert. Bei einem RFID-System gibt es immer einen aktiven und einen passiven Partner. Die Lesegeräte sind stets aktiv und versorgen den Chip mit Energie, um auf eine Anfrage des Lesegerätes zu antworten. Der Daten- oder Nachrichtenaustausch beginnt also jedesmal mit dem Lesegerät, bei NFC hingegen werden beide Funktionen miteinander vereinigt. Das NFC-Gerät kann zum Einen als aktives Lesegerät fungieren und einen Chip mit Energie versorgen. Zum Anderen kann das Gerät eine kontaktlose Chipkarte emulieren.

3.1.2 Die ersten Mobiltelefone

Das interessanteste Gerät zur Integration von NFC ist von Beginn an das Mobiltelefon. Der Nutzer trägt es häufig bei sich und es muss kein neues Gerät erfunden werden, nur um Daten mit NFC auszutauschen. Das erste kommerzielle Mobiltelefon war das Nokia 3220 in einer speziellen Version mit sogenannter “NFC Shell„. Dabei wurde das NFC-Modul in die Gehäuseschale eingebaut. Über die Zeit waren es von unterschiedlichen Herstellern

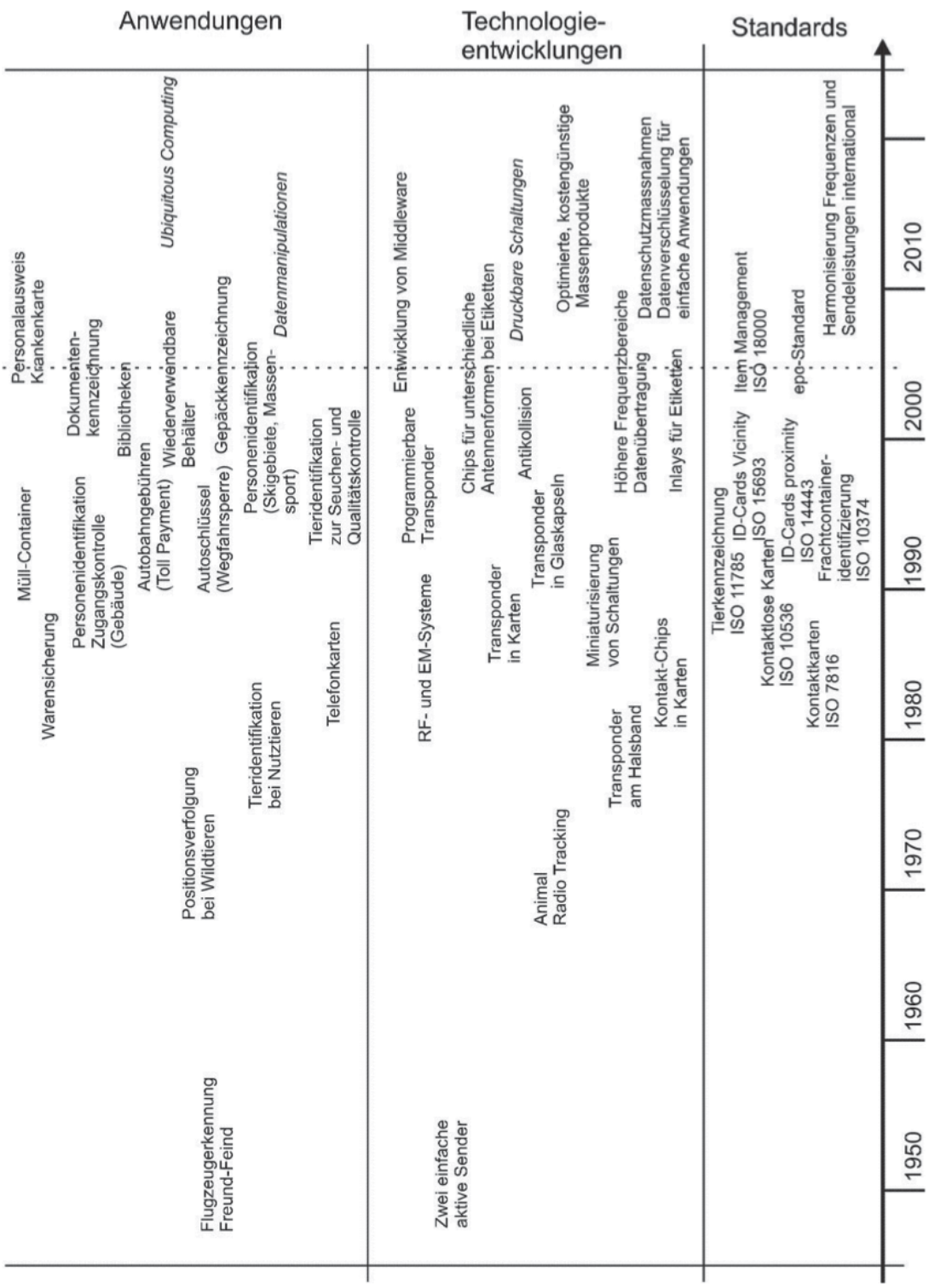

Abbildung 3.2: Entwicklung von RFID-Systemen seit 1945 auf drei Ebenen: Standards, Technik und Anwendung, Quelle: [6]

Mobiltelefone mit NFC auf dem Markt, zum Beispiel das Sagem my700X, das Samsung D500E und das BenQ T80 [5]. Deutlich interessanter wird die Interaktion mit dem Gerät und möglichen NFC-Gegenstücken mit der Verbreitung der Smartphones. Trotz des großen Potenzials und den etlichen Smartphones, die 2011 und 2012 auf den Markt gekommen sind, ist der Anteil der neuen Geräte mit NFC-Funktionalität sehr gering.

3.2 Technischer Aufbau

Die Basistechnologie bei NFC ist die gleiche wie bei RFID, was den Vorteil hat, dass es bereits viele bestehende Systeme gibt, die kompatibel mit neuen NFC-Anwendungen sind. Das ist beispielsweise für Bezahlfunktionen mit Kreditkarten ein großer Vorteil [5]. NFC ist ein induktiv gekoppeltes System, das mit einem hochfrequenten magnetischen Feld arbeitet und eine Frequenz von 13,56 MHz nutzt. Es gibt drei unterschiedliche Betriebsarten von NFC: passive, semi-passive und aktive Systeme. Bei aktiven Systemen haben beide Partner, das Lesegerät und der Transponder, eine eigene Stromversorgung. Bei den anderen beiden Arten wird die Energie vom Lesegerät übertragen, in dem die Antenne mit sinusförmigen Strom versorgt wird. In der Transponderantenne wird dadurch eine Spannung indiziert. Bei passiven und semi-passiven Systemen gibt es eine feste Rollenverteilung zwischen dem Lesegerät und dem Transponder. Es gibt für die Datenübertragung sowohl einen Kanal zwischen dem Lesegerät und dem Transponder als auch einen zweiten Kanal in umgekehrter Richtung. Aktive Systeme teilen sich einen Kanal, die Rollen werden ständig bei Bedarf gewechselt.

Um Daten übertragen zu können, müssen diese an ein passendes Modulationsverfahren angepasst und dafür codiert werden. Es gibt bei NFC drei unterschiedliche Modulationsverfahren, die eingesetzt werden: n-ASK (Amplitude-shift keying, Amplitudenumstastung), n-PSK (Phase-shift keying, Phasenumstastung) oder n-FSK (Frequenzy-shift keying, Frequenzumtastung). Bei den Codierungsverfahren gibt es viele unterschiedliche Alternativen, zum Beispiel die einfachste Variante, dass jedes Datenbit in einem bestimmten Signalpegel codiert wird (NRZ-L-Codierung).

Bei der Datenübertragung vom Lesegerät zum Transponder (der sogenannte Uplink) wird also das Trägersignal, das gleichzeitig auch zur Energieversorgung des Transponders dient, mit einem Datenstrom modeliert. Bei passiven und semi-passiven Systemen gibt es zusätzlich den Downlink. Die Kopplung der Spulen des Lesegerätes und des Transponders wirkt nicht nur auf den Transponder sondern auch andersherum. Durch Impedanzänderungen im Transponder kann eine Amplituden- oder Phasenänderung der Spannung bewirkt werden. Diese Spannungsänderungen sind für das Lesegerät wiederum messbar. Das Signal kann folglich vom Transponder modeliert werden, um Daten zurück zum Lesegerät zu übertragen.

Gibt es ein System mit mehreren Lesegeräten und Transpondern, dann benötigt man weitere Techniken, um die einzelnen Geräte untereinander zu unterscheiden und die Kommunikationskanäle nicht zu überlagern, wodurch sie unbrauchbar würden [32].

3.3 Kontaktlose Chipkarten

Wichtiger Bestandteil von NFC-Systemen sind Smartcards beziehungsweise kontaktlose Chipkarten. In Smartphones werden viele sicherheitskritische Aufgaben mit Hilfe von Smartcards (hier der SIM-Karte) gelöst. Nach Definition ist eine Smartcard eine Kunststoffkarte mit normierten Abmessungen und einem Microchip. Es gibt Smartcards nur mit eigenem Speicher auf den geschrieben und gelesen werden kann oder auch komplexe Prozessorkarten, die eigene Betriebssysteme und Dateisysteme beinhalten. Die Technologie ist in ISO-Normen standardisiert. Neben kontaktbehafteten Chipkarten gibt es die kontaktlosen Chipkarten. Hier haben sich hauptsächlich zwei Typen von Karten durchgesetzt, die auch mit den im NFC-Forum genannten Spezifikationen konform sind.

Das ist zum Einen MIFARE, eine Produktreihe der Firma NXP Semiconductors, ehemals Philips Semiconductors. Ursprüngliches Einsatzgebiet waren elektronische Bezahlsysteme für den öffentlichen Personalverkehr. Laut NXP Semiconductors sollen bereits über eine Milliarde MIFARE-Chipkarten verkauft worden sein, was das System zum weitverbreitesten der Welt macht. Es gibt unterschiedliche Typen von MIFARE-Karten. Das andere große Chipkarten-System nennt sich FeliCa (von Felicity Card) und wurde von der Firma Sony entwickelt. Auch FeliCa ist kompatibel zum NFC-Standard und wird von Sony in verschiedenen Bauformen und Speichergrößen angeboten.

3.4 Die drei Modi von NFC

Die NFC-Architektur gliedert sich in drei Betriebsarten:

- Peer-to-Peer-Modus
- Reader/Writer-Modus
- Card-Emulation-Modus

Der Peer-to-Peer-Modus ermoglicht die Kommunikation zwischen zwei NFC-Geraten. Die Grundlagen fur diesen Modus sind in NFCIP-1 normiert. Im Reader/Writer-Modus kann ein NFC-Gerat mit passiven Transpondern, sogenannten NFC-Forum-Tags, kommunizieren. Obwohl dies nicht durch das NFC Forum spezifiziert ist, konnen NFC-Gerate oft auch mit anderen kontaktlosen Chipkarten, die keinem der NFC-Forum-Tag-Formate entsprechen, interagieren. Der optionale Card-Emulation-Modus erlaubt einem NFC-Gerat mit RFID-Lesegeraten zu kommunizieren. Das NFC-Gerat verhalt sich dazu wie eine kontaktlose Smartcard.

Die Grundlage fur den Reader/Writer-Modus und den Card-Emulation-Modus bilden die RFID-Normen. Neben dem Peer-to-Peer-Modus, in dem ein NFC-spezifisches Protokoll fur den Datenaustausch eingesetzt wird, konnen NFC-Gerate auch uber eine Kombination aus Reader/Writer-Modus und Card-Emulation-Modus uber die ublichen RFID-Protokolle kommunizieren [5].

3.5 Sicherheitsaspekte unter NFC

Die Protokollebenen von NFC enthalten keine Schutzmaßnahmen gegen Angriffe. Die NFC-Schnittstelle lässt sich, wie andere drahtlose Übertragungstechnologien auch, relativ einfach abhören. Die Kommunikation zwischen zwei NFC-Geräten ist an mehreren Stellen gefährdet. Zum Einen kann die Kommunikation auch aus größerer Entfernung abgehört werden. Der Angreifer kann nicht nur die übetragenden Informationen analysieren, sondern die Daten erneut schicken, um eine bestimmte Transaktion nochmals auszuführen (eine sogenannte Reply-Attacke). Des Weiteren kann mit Hilfe eines Störsenders die Datenübertragung gestört oder gezielt beeinflusst werden. Zudem kann ein Angreifer Anfragen beantworten, bevor der echte Partner mit seiner Antwort beginnt. Zuletzt gibt es noch die sogenannte Man-in-the-Middle-Attacke, bei der der Angreifer zwischen beiden Geräten sitzt und die Anfragen jeweils weitergibt, sie selbst allerdings verarbeiten und verändern kann.

Die meisten dieser Angriffe können verhindert werden, indem kryptographische Verfahren genutzt werden. Durch die Verschlüsselung der Nachrichten ist der Kanal gegen das Abhören Fremder geschützt. Die gezielte Manipulation von Nachrichten kann durch Authentifizierungscodes (bei NFC sogenannten MAC, Message Authentication Codes) verhindert werden. Dieser Schlüsselaustausch und ein kryptographischer Kommunikationskanal schützen allerdings nicht vor der Man-in-the-Middle Attacke. Wenn der Angreifer es schafft, ein Gerät zwischen die beiden echten Partner zu bringen, können die Nachrichten verändert werden, da NFC-Geräte keine eindeutige Identifikation haben. Allerdings kann sich der Anwender in den meisten Anwendungsfällen selbst vor einer Man-in-the-Middle-Attacke schützen. Damit dieser Angriff funktioniert, muss ein sichtbares Gerät zwischen den beiden Partnern untergebracht werden. Dieses muss zusätzlich in beide Richtungen abgeschirmt sein, damit die magnetischen Felder sich nicht überschneiden, d.h. ein unsichtbares Anbringen ist praktisch kaum möglich. Höchstens bei fest installierten und nicht bewachten Geräten könnte es zu Manipulationen kommen [5] [3].

3.6 Das Secure Element in mobilen NFC-Geräten

In mobilen NFC-Geräten gibt es vier wichtige Hardware-Bestandteile, die durch weitere Funktionen von beispielsweise Mobiltelefonen (Modem) unterstützt werden:

- NFC-Antenne
- NFC-Controller
- Host-/Basisbandcontroller
- Secure Element

NFC-Anwendungen beinhalten oft sicherheitskritische Funktionen. Diese reichen vom Bezahlen mit einer elektronischen Brieftasche, über das Ausweisen mit geheimen Schlüsseln, dem Austausch von Kreditkarten-Informationen bis hin zu unterschiedlichen Arten von Tickets, die gespeichert werden müssen. Man benötigt also einen besonderen Speicher, der sicher genug ist, solche geheimen Daten aufzubewahren. Diese Umgebung für das Ausführen und

Speichern von sicherheitskritischen Daten und Applikationen wird Secure Element bezeichnet.

Theoretisch gibt es viele Varianten, wie das Secure Element in Mobiltelefonen umgesetzt werden kann. Man kann spezielle Software schreiben ohne ein eigenes Hardwareteil, bei dem allerdings keine Manipulationssicherheit vorhanden wäre. Eine andere Möglichkeit wäre, fest in das Mobiltelefon integrierte Hardware zu nutzen. Diese Methode hätte den Nachteil, dass jedes Mal, wenn der Nutzer sein Mobiltelefon wechselt, alle seine persönlichen und geheimen Daten auf irgendeine Weise übertragen werden müssten. Die letzte Möglichkeit in dieser groben Unterteilung wäre die, austauschbare Hardware zu benutzen. Damit ist der Nutzer in der Lage, die Hardware in ein neues Mobiltelefon einzusetzen, sobald er dieses erneuert. Allerdings braucht man dafür dann passende Steckplätze. Diese sind zum Einen mit höheren Materialkosten, aber auch mit einem größeren Platzverbrauch verbunden. Aus diesen Gründen hat man sich bei den meisten aktuellen Einsätzen dazu entschieden, die SIM-Karte als Secure Element zu verwenden. Diese ist in jedem Mobiltelefon vorhanden und erfüllt auch die Smartcard-Voraussetzungen. Einziger Nachteil dabei ist, dass die SIM-Karten an die Provider gebunden sind. Das heißt, wenn ein Nutzer seinen Mobilfunkanbieter wechselt, wechselt sich auch seine SIM-Karte und die privaten Daten darauf müssen neu konfiguriert werden [33].

3.7 NFC-Entwicklung unter Android

3.7.1 Die unterschiedlichen NFC-Protokolle

Sobald Daten auf einem Tag gespeichert werden sollen, muss man sich entscheiden, in welchem Format die Daten abgelegt werden sollen. Entweder man entscheidet sich für ein bereits bestehendes Format oder definiert ein eigenes. Je nach Anwendungsgebiet haben beide Möglichkeiten ihre Berechtigung. In den meisten Fällen ist es jedoch sinnvoll auf bereits bestehende Formatvorgaben zuzugreifen. Nur so kann man gewährleisten, dass auch andere Geräte die Daten eines Tags lesen und verarbeiten können.

In diesem Sinne unterstützt auch Android eine Fülle von verschiedenen, weniger verbreiteten Datenformaten wie z.B. Mifare Classic[3], NfcA/B/F/V[4] oder IsoDep[5]. Der Zugriff auf solcherart formatierte Daten gestaltet sich wenig komfortabel. Genauer gesagt, liest man lediglich unformartierte Bytes aus. Einfacher gestaltet sich der Zugriff auf Tags, die in einer NDEF-Formatierung vorliegen. Das NDEF-Protokoll wurde im Juli 2006 durch das NFC-Forum verabschiedet[12]. Sowohl in der gängigen Literatur als auch bei einem Großteil existierender NFC-Frameworks wird NDEF als das übliche Datenformatierungsprotokoll angegeben. So wird auch in Googles Developer Guide für die Android Entwicklung geraten, NDEF als Formatierung zu verwenden. Neben einer einfachen Entwicklung wirbt Google mit einer maximalen Unterstützung des NDEF-Formates durch Android-Telefone.

Der komfortablen Handhabung von serialisierten Daten im NDEF-Format liegt eine einfache Kapselung von Daten zugrunde. Ein Tag kann beliebig viele NDEF-Messages und

[3] Properitäres Datenformat der Firma NXP Semiconductors.
[4] Ein durch die internationale Normenreihe ISO 14443 definiertes Datenformat für kontaktlose Chipkarten.
[5] Definiert durch NFC Forum.

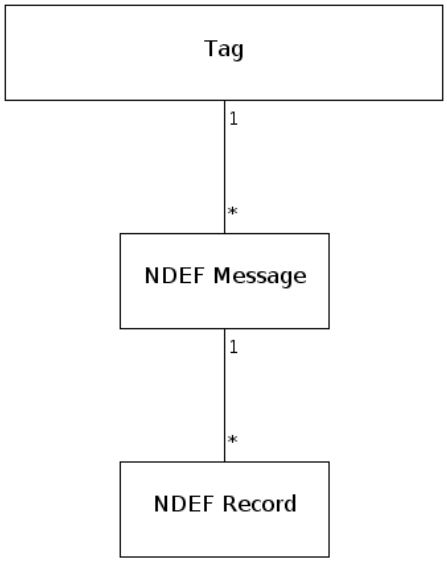

Abbildung 3.3: Aufbau eines Tags mittels NDEF-Messages und NDEF-Records

eine NDEF-Message beliebig viele NDEF-Records enthalten. Anzahl der Messages bzw. Records und Umfang der enthaltenen Daten sind nur durch die Kapazität des Tags begrenzt. NDEF-Messages dienen lediglich als Kapselung von Daten. Der eigentliche Inhalt wird in NDEF-Records gespeichert. Damit ein NDEF-Record dem Protokoll entspricht, muss es wohlgeformt sein, bedeutet: Folgende Informationen sind Pflichtangaben:

Type Name Format Beschreibt die Art und Weise, wie der Eintrag type interpretiert werden soll.

type Beschreibt die Typen-Art des Records. Abhängig vom Type Name Format (TNF) sind unterschiedliche Typen erlaubt.

ID Eindeutige Zeichenkette anhand welcher man den Record später identifizieren kann.

payload Enthält den eigentlichen Inhalt des Records.

3.7.2 Zuordnung von Tags zu Applikationen

Sobald ein Android-Smartphone einen Tag ausliest, versucht es die Daten einer Applikation zuzuweisen. Der Mechanismus, der diese Aufgabe übernimmt, wird Tag Dispatch System genannt. In drei Schritten analysiert das System, die auf dem Tag gespeicherten Daten, kapselt diese in einem Intent und übergibt es an eine passende Applikation.

Sollte der Inhalt in Form des NDEF-Protokolls vorliegen, wird in Schritt 1 der erste NDEF-Record einer NDEF-Message ausgelesen. Aus den Feldern TNF und type versucht das System nun einen validen MIME-Type oder eine valide URI zu erstellen. Die Daten werden im Anschluss gemeinsam mit MIME-Type oder URI in ein Intent verpackt. In Schritt

3 wird, abhängig von dem Typ des erstellten Intents, eine passende Applikation gesucht und das Intent übergeben. Die App kann nun die Daten verarbeiten.

Bei der Kapselung der Daten in einem Intent können drei verschiedene Intent-Typen erstellt werden:

ACTION_NDEF_DISCOVERED Wird erstellt, sobald der Tag Daten im NDEF-Format enthält.

ACTION_TECH_DISCOVERED Dieses Intent wird in mehreren Fällen erstellt. Fall 1: Die Daten des Tags liegen im NDEF-Format vor und auf dem Telefon existiert keine App, die sich als Handler für ACTION_NDEF_DISCOVERED registriert hat. Fall 2: Die Daten liegen im NDEF-Format vor, können aber weder einem MIME-Type oder einer URI zugeordnet werden. Fall 3: Der Tag enthält Daten, die mittels eines anderen NFC-Protokolls formatiert wurden.

ACTION_TAG_DISCOVERED Dieses Intent wird erstellt, sofern keine Applikation existiert, die sich als Handler für ACTION_NDEF_DISCOVERED oder ACTION_TECH_DISCOVERED registriert hat.

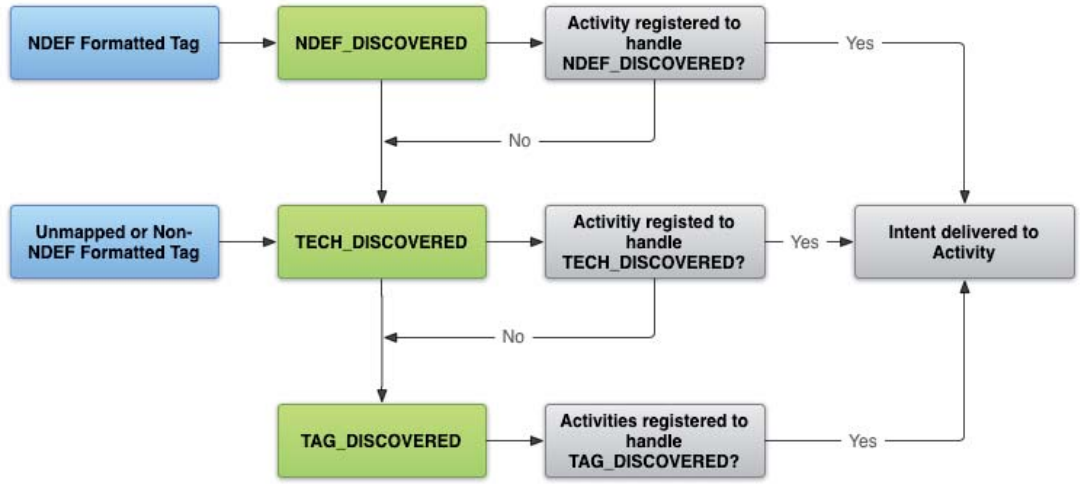

Abbildung 3.4: Ablauf des Tag Dispatch Sytem Quelle: `http://goo.gl/65FJ5`, abgerufen am 28.06.2012

Um dem Betriebssystem mitzuteilen, welche Art von Intents die Applikation verarbeiten kann, bietet Android eine Schnittstelle namens Intent-Filter an. Diese Filter werden in der Manifest-Datei[6] eingetragen.

[6] Jede Applikation hat eine AndroidManifest.xml Datei in ihrem Wurzelverzeichnis. Sie beschreibt grundlegende Eigenschaften der Applikation, wie zum Beispiel eine Applikation die als Einstiegspunkt in die Applikation aufgerufen wird.

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
</intent-filter>
```

Listing 3.1: Intent-Filter für NDEF-Tags mit MIME-Type

Listing 6 zeigt die Definition eines Intent-Filter mittels MIME-Type. Enthält eine Applikation diesen Intent-Filter, so registriert sie sich beim Betriebssystem für Tags, deren Daten im NDEF-Format vorliegen und aus reinem Text bestehen. Weitere Unterscheidungen können aufgrund von definierten URIs getroffen werden. Listing 8 beschreibt einen Filter für eine URI in Form eines Links auf die Webseite „http://developer.android.com/index.html“.

```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED" />
    <category android:name="android.intent.category.DEFAULT" />
   <data android:scheme="http"
              android:host="developer.android.com"
              android:pathPrefix="/index.html" />
</intent-filter>
```

Listing 3.2: Intent-Filter für NDEF-Tags mit URI

Seit Android Ice Cream Sandwich, Version 4.0 gibt es außerdem noch eine weitere, einfachere Möglichkeit NDEF-Messages einer bestimmten Applikation zuzuordnen: Android Application Records. Es handelt es sich dabei um NDEF Records, die einer Application eindeutig zugewiesen werden können. Um die Zuordnung eindeutig zu gestalten, muss der Entwickler den Namespace der Zielapplikation kennen. Dann kann er auf einfachem Wege seine NDEF-Message um einen ApplicationRecord erweitern (siehe Listing 4.22). Sollte die gewünschte Applikation nicht auf dem Zielgerät installiert sein, öffnet Android den Google Play Store[7] eigenständig, damit die App installiert werden kann.

```
NdefMessage msg = new NdefMessage(
        new NdefRecord[] {
            ...,
            NdefRecord.createApplicationRecord("de.bnjmnrhl.nfclabs")}
```

Listing 3.3: Android Application Records

[7] Bezugsquelle von Android Applikationen. Sowohl als Android App als auch als Webseite benutzbar (`https://play.google.com/store`, abgerufen am 28.06.2012)

3.7.3 NFC-Kommunikation zwischen zwei Geräten

Googles Android bietet nicht nur die Möglichkeit Smartcards auszulesen, auch die direkte Kommunikation zwischen zwei aktiven Geräten ist möglich. Dafür gibt es zwei verschiedenartige Mechanismen. Mit dem API Level 10 (Android 2.3.3) konnte man NDEF-Nachrichten per ForegroundNdefPush an andere Geräte übermitteln. Mit Android 4.0 wurde Android Beam eingeführt und ForegroundNdefPush als veraltet markiert. Android Beam bietet im Grunde die gleiche Funktionalität, ist jedoch für den Entwickler komfortabler zu programmieren. Mit der Möglichkeit NDEF-Messages ohne den Umweg über eine Smartcard von Gerät A nach Gerät B zu übermitteln, entfallen auch Limitierungen, wie die geringe Speichergröße. Allerdings gilt es weiter zu beachten, dass die Übertragungsrate zwischen zwei Geräten ebenfalls bei maximal 424 kbit/s liegt.

Kapitel 4

SimpleNFC - Entwicklung eines NFC-Frameworks für Android

4.1 Anforderungen und Vorrangehensweise

Wie in Kapitel 3.7 beschrieben, unterstützt die NFC-Schnittstelle von Android unterschiedlichste Möglichkeiten auf NFC-Funktionalitäten zuzugreifen. Dabei gilt es die Übersicht über verschiedenste Protokolle und unterschiedliche Betriebssystemversionen zu behalten. Möchte man Daten auf Tags speichern, kann man sich meist nicht sofort für ein definiertes Protokoll entscheiden, sondern muss sich im Vorfeld informieren, wie weit das Protokoll verbreitet ist, welche Einschränkungen oder Vorteile es mit sich bringt und welche, auf dem Markt befindlichen, technischen Geräte eine Unterstützung garantieren.

Recherchen darüber können viel Zeit in Anspruch nehmen und enden nicht selten in Verwirrungen. Doch nicht nur von Protokollen gibt es verschiedene Varianten. Auch die Fragmentierung von Android-Versionen mit NFC-Unterstützung nimmt zu. Mit jedem weiteren Betriebssystemupdate wird der Funktionsumfang von NFC erweitert. Was sich im ersten Moment positiv liest, hat für Entwickler jedoch einen bitteren Beigeschmack. Möchten sie die neuen Funktionen nutzen und ihre Applikationen weiterhin einer breiten Masse an Benutzern zugänglich machen, liegt es an ihnen im Quellcode Unterscheidungen für verschiedene Versionen des Betriebssystems zu treffen. Das Ergebnis ist eine App, dessen neuesten NFC-Möglichkeiten nur denjenigen Nutzern zur Verfügung steht, die auf ihrem Smartphone die aktuellste Android-Version installiert haben.

Als Entwickler muss man all diese Aspekte bei der Programmierung bedenken. Leider verschiebt sich dabei der Fokus: weg von der eigentlichen Entwicklung hin zu Diskussionen über Kompatibilität von Soft- und Hardware. Kapitel 4.2 beschäftigt sich mit einer Analyse des technischen IST-Zustandes und beschreibt dessen Unzulänglichkeiten. Mit einem Blick auf alle Mängel wird ein möglicher SOLL-Zustand (siehe Kapitel 4.3) definiert, der einen Großteil der Einschränkungen aufhebt. Wie diese theoretischen Ideen umgesetzt und der SOLL-Zustand erreicht wird, beschreibt das dritte Kapitel 4.4. Das Kapitel 4.5 im Anschluss erläutert detailliert die einzelnen Teilbereiche des entstandenen Frameworks.

4.2 Ist-Zustand

Da NFC als Standard noch sehr jung ist[1], existieren relativ viele verschiedene Normen. Die meisten Normen wurden als properitäre Normen von Hardware-Herstellern ins Leben gerufen. Manche davon wurden von offiziellen Gremien, wie der Internationale Organisation für Normung, verabschiedet, manche befinden sich noch immer im Status der Abstimmung. Android-Geräte mit NFC-Chips unterstützen lediglich genormte Standards. Diese werden in zwei Kategorien eingeteilt: NfcA, NfcB, NfcF, NfcV, IsoDep und Ndef gehören zu den obligatorischen Protokollen, welche jedes Android-Gerät interpretieren kann. Optional können Hersteller das System noch um Protokolle für MifareClassic, MifareUltralight und NdefFormatable erweitern. Betrachtet man die dafür erschaffene Schnittstelle von Android, so sieht man, dass der Zugriff auf die meisten Standards (genauer gesagt: alle außer Ndef) sehr rudimentär aufgebaut ist. Die Kommunikation mit der Smartcard erfolgt lediglich über raw bytes (unformatierter Text) und es gibt keinerlei Strukturvorgaben oder Einteilungen in Messages und Records.

Lediglich für den Ndef-Standard gibt es eine komfortablere Schnittstelle. Wie in Kapitel 3.7 beschrieben, definiert das Ndef-Protokoll Formatierungsvorgaben wie die NdefMessage und den NdefRecord. NdefMessages, als Behältnis für NdefRecords, erfordern keine besondere Konfiguration. Sie werden mit einem Array aus NdefRecords initialisiert und sind einsatzbereit. NdefRecords hingegen können vielfältig konfiguriert werden. Nicht die einzelnen Felder für sich, vielmehr die vielfachen Kombinationen aus Type Name Formats und Record Type Definitions machen es schwierig für jedes Anwendungsszenario die richtige Kombination zu finden.

Dabei kann es, abhängig von der Wahl des TNF, zu mehr oder weniger komplexen Kombinationen kommen. Als Type Name Format gibt Android sieben verschiedene Formate (siehe Abbildung 4.1) vor. Enthalten sind Formate für leere Records, Records mit bekannten bzw. unbekannten Inhalten, Records mit Inhalten, die aus dem Internet geladen werden müssen und noch einige mehr. Jedes Format schreibt dabei vor, welcher Inhalt für die Record Type Definition erlaubt ist. Dabei gibt es sowohl TNFs, die für die Record Type Definition keinen Inhalt als auch eine individuelle Zeichenkette als Inhalt vorsehen. Lediglich für TNF_WELL_KNOWN existiert eine Liste an Auswahlmöglichkeiten (siehe Abbildung 4.2).

Wie man unschwer erkennt, gibt es eine Vielzahl von möglichen Kombinationen, die es erfahrenen Entwicklern ermöglicht, NFC über das eigentliche Ökosystem Android hinaus zu verwenden. Für die meisten Nutzerszenarien würde jedoch auch bedeutend weniger Komplexität ausreichen. Doch selbst wenn sich Entwickler für das Ndef-Protokoll entscheiden, ist der Zugriff auf Tags nicht so komfortabel wie er sein könnte. Vor dem eigentlichen Beschreiben ist es nötig,ein gewisses Maß an Vorarbeit zu leisten. Dazu muss der Entwickler jedes mal überprüfen, ob der Tag beschreibbar ist und die Kapazität für die Daten ausreicht. Danach wird kontrolliert, ob der Tag bereits formatiert ist. Sollte er unformatiert vorliegen, muss er vor dem Beschreiben formatiert werden. Da jedoch nicht alle Tags für das Ndef-Protokoll formatiert werden können, muss auch diese Kompatibilität überprüft werden. All

[1] In 2002 wurden die ersten Entwürfe für eine NFC-Norm zusammen von NXP Semiconductors und Sony veröffentlicht. Quelle: `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53430`, abgerufen am 28.06.2012

Record Type Definition (RTD)	Mapping
RTD_ALTERNATIVE_CARRIER	Falls back to ACTION_TECH_DISCOVERED.
RTD_HANDOVER_CARRIER	Falls back to ACTION_TECH_DISCOVERED.
RTD_HANDOVER_REQUEST	Falls back to ACTION_TECH_DISCOVERED.
RTD_HANDOVER_SELECT	Falls back to ACTION_TECH_DISCOVERED.
RTD_SMART_POSTER	URI based on parsing the payload.
RTD_TEXT	MIME type of text/plain.
RTD_URI	URI based on payload.

Abbildung 4.1: Liste der möglichen Type Name Formats, Quelle: `http://goo.gl/65FJ5`, abgerufen am 28.06.2012

Type Name Format (TNF)	Mapping
TNF_ABSOLUTE_URI	URI based on the type field.
TNF_EMPTY	Falls back to ACTION_TECH_DISCOVERED.
TNF_EXTERNAL_TYPE	URI based on the URN in the type field. The URN is encoded into the NDEF type field in a shortened form: *<domain_name>:<service_name>*. Android maps this to a URI in the form: vnd.android.nfc://ext/*<domain_name>:<service_name>*.
TNF_MIME_MEDIA	MIME type based on the type field.
TNF_UNCHANGED	Invalid in the first record, so falls back to ACTION_TECH_DISCOVERED.
TNF_UNKNOWN	Falls back to ACTION_TECH_DISCOVERED.
TNF_WELL_KNOWN	MIME type or URI depending on the Record Type Definition (RTD), which you set in the type field. See Table 2. for more information on available RTDs and their mappings.

Abbildung 4.2: Liste der möglichen Record Type Definitions für den TNF TNF_WELL_KNOWN, Quelle: `http://developer.android.com/guide/topics/nfc/nfc.html`, abgerufen am 28.06.2012

diese Kontrollen müssen von Entwicklern explizit implementiert werden, es gibt dafür keinerlei abstrakte Fehlerbehandlung. Abgesehen von Protokollen und deren Komplexität gibt es in Android ebenfalls keine Unterstützung bei der Verarbeitung von NdefMessages. Sobald eine Applikation mehrere verschiedene NdefMessages mit unterschiedlichen Typen von Inhalten lesen soll, gibt es unterschiedliche Teilbereiche der Applikation, die unterschiedliche Typen von Inhalten verarbeiten. Man stelle sich folgendes Szenario vor: Ein Museum entwickelt eine Applikation, die es ermöglicht, NFC-Tags an Ausstellungsstücken zu lesen und zusätzliche Informationen anzuzeigen. Dabei können die zusätzlichen Informationen sowohl als Text als auch als Video vorliegen. Die Architektur der Applikation würde es vorsehen für jeden Typ von Informationen eine eigene Activity anzulegen, welche die Informationen in einer angebrachten Art und Weise darstellt. Damit eine Zuordnung von Text-Tag zur Text-Activity und Video-Tag zur Video-Activity gewährleistet wird, müssen Änderungen an zwei Stellen vorgenommen werden. NdefRecords für Text und Video erhalten unterschiedliche MIME-Types, sodass man sie auseinander halten kann. Die Zuordnung von Tag zu verantwortlicher Activity erfolgt anschließend in der AndroidManifest.xml-Datei. Dort werden für jede Activity ein neuer Intent-Filter angelegt. Tags mit einem Text-MIME-Type werden an die Text-Activity und Tags mit Video-MIME-Type an die Vide-Activity weitergeleitet. Je größer die Zahl an unterschiedlichen Typen von Tag, desto größer wird der Aufwand und die Komplexität eine Zuordnung von Tag zu verantwortlicher Instanz zu gewährleisten.

```
1  ...
2  <activity android:name="de.example.museum.InfoText"
3    <intent-filter>
4      <action android:name="android.nfc.action.NDEF_DISCOVERED" />
5          <data android:mimeType="application/de.example.museum.text" />
6          <category android:name="android.intent.category.DEFAULT" />
7    </intent-filter>
8  </activity>
9
10 <activity android:name="de.example.museum.InfoVideo"
11   <intent-filter>
12         <action android:name="android.nfc.action.NDEF_DISCOVERED" />
13         <data android:mimeType="application/de.example.museum.video" />
14         <category android:name="android.intent.category.DEFAULT" />
15     </intent-filter>
16 </activity>
17 ...
```

Listing 4.1: Intent-Filter für die Zuweisung von NdefMessage zu Activity

Neben dieser detaillierten Art der Zuordnung gibt es mit der Einführung von Android 4.0 eine weitere, gröbere Art der Zuordnung. Wie bereits im Kapitel 3.7.2 erläutert, können NdefMessages mittels Android Application Records Anwendungen eindeutig zugewiesen werden. Die Sinnhaftigkeit dieser Erweiterung steht außer Frage. Jedoch kommen lediglich Nutzer eines aktuellen Smartphones mit der aktuellsten Android-Version in den Genuss dieser Funktion, vorausgesetzt Entwickler scheuen nicht den Mehraufwand, der entsteht, sobald man im Quellcode zwischen unterschiedlichen Betriebssystem-Versionen unterscheiden muss um die

neusten Erweiterungen nutzen zu können. An dieser Stelle ist es wünschenswert, die gleiche Funktionalität auch für ältere Geräte nutzbar zu machen.

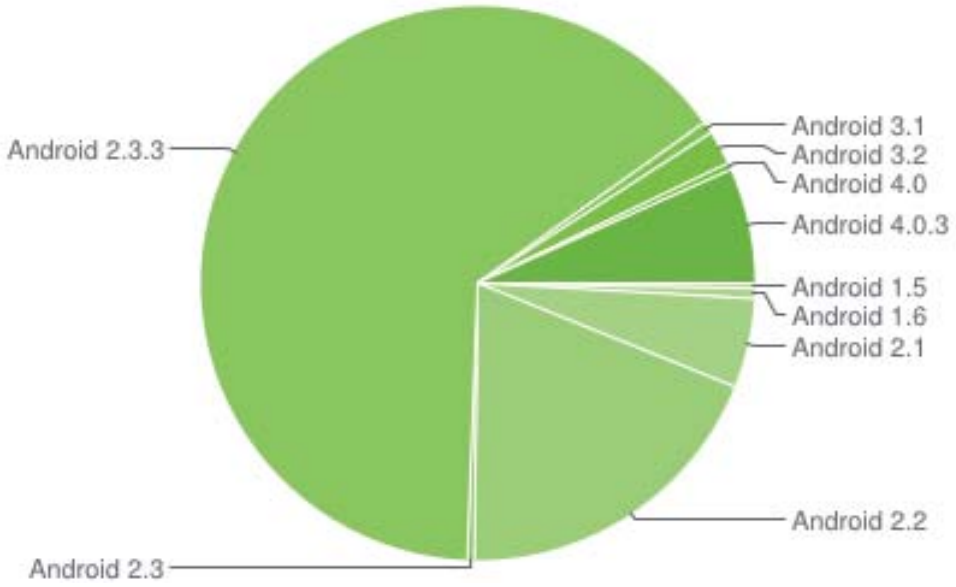

Abbildung 4.3: Fragmentierung aller auf dem Markt befindlichen Android-Versionen, Stand Juni 2012, Quelle: `http://goo.gl/BvZRw`, abgerufen am 28.06.2012

4.3 Soll-Zustand

Um Programmierern die Arbeit zu erleichtern, ist es wichtig, die Komplexität von NFC als Technologie so gut es geht zu verringern. Ein erster Schritt wäre es, die technischen Anforderungen von NFC vor einem potentiellen Entwickler fernzuhalten. Dazu definiert man alle möglichen Nutzerszenarien, die auftreten könnten. Die benötigten API-Aufrufe werden in anwendungsspezifischen Methoden gekapselt und dem Entwickler in Form einer einfach zu verstehenden Schnittstelle angeboten. Da man jedoch nicht alle möglichen Nutzerszenarien voraussehen kann, dürfen durch eine Kapselung in Nutzerszenarien keine Einschränkungen entstehen. Es muss weiterhin möglich sein, NFC-Funktionalitäten auf einer weniger abstrakten Art und Weise zu verwenden. Der einfache und leicht zu verstehende Charakter der Schnittstelle definiert sich aus eindeutig beschreibenden Methodennamen. Beim Betrachten der Dokumentation soll der Entwickler sofort die Methoden finden, die er für sein Vorhaben benötigt. Dabei beschreiben die Methodennamen nicht den technischen Kontext der Methode, sondern welcher spezifische Anwendungsfall mit ihnen aufgerufen wird. Nachrichten, die geschrieben oder übertragen werden, bekommen abstraktere Bezeichnungen. Den Entwickler interessiert es nicht, im schlimmsten Fall verwirrt es ihn sogar, wenn erst technische Einzelheiten vor ihm verborgen werden, dann aber Nachrichten plötzlich NdefMessages und NdefRecords genannt werden. Neben anwendungsspezifischen Funktionen gibt es außerdem

Methoden, die Meta-Informationen bereithalten. Zum Beispiel: Bietet das System überhaupt eine NFC-Unterstützung oder kann ich bestimmte, API-Level abhängige Funktionen benutzen? Unzulänglichkeiten der bestehenden Android-API können außerdem beseitigt werden, indem NFC-Nachrichten einer verantwortlichen Instanz zugewiesen werden können. Sobald solch eine Nachricht gelesen wird, leitet sie das System automatisch an die dafür vorgesehene Instanz weiter und diese verarbeitet die Nachricht. Somit entfällt das Eintragen von Intent-Filter in der AndroidManifest.xml-Datei. Schlussendlich soll es ebenfalls möglich sein auf Geräten mit älterer Betriebssystem-Version ebenso von den Neuerungen späterer Versionen zu profitieren. Ziel soll es sein, ein sich selbsterklärendes System zu entwickeln, welches durch eine einfache Einbindung all diese Anforderungen erfüllt, den Entwickler bei seiner Arbeit aktiv unterstützt und Unterschiede zwischen verschiedenen API-Leveln ausgleicht.

4.4 Zwischen Ist und Soll: SimpleNFC

Das Kapitel zuvor beschrieb relativ abstrakt ein System, welches Unzulänglichkeiten in bestehenden Android-Versionen beheben kann. Wie dieses System in der Realität aussieht, soll das folgende Kapitel erläutern. Zu Beginn muss geklärt werden, welche Art der Implementierung den Anforderungen entspricht:

erweitertes Android Die sauberste Lösung wäre es, den Quelltext von Android zu erweitern und daraus eine lauffähige Betriebssystemversion zu kompilieren. Ein erweitertes Android wäre zwar die sauberste, aber auch die am wenigsten effektivste Lösung. Nur Nutzer mit einem entsperrten Smartphone könnten das Systemabbild aufspielen und von den Vorteilen profitieren.

andere NFC-Implementierung Wie im Kapitel 2 beschrieben, gibt es bereits alternative Implementierungen für NFC. Open NFC als Beispiel bietet einige Funktionalitäten, die Android bisher nicht bietet. Allerdings hat die Einbindung von low-level Frameworks ähnliche Nachteile wie ein erweitertes Android. Die Tatsache, dass Nutzer etwas an ihrem Smartphone ändern müssen, ist schon per se falsch.

high-level Framework Wenn überhaupt sollte der Mehraufwand bei den Entwicklern liegen. Diesen müssten sie aufbringen, wenn sie die NFC-Funktionalität von Android zum Beispiel durch die Einbindung eines high-level-Frameworks erweitern wollen würden. Android bietet von Haus aus Mechanismen um high-level-Frameworks sehr leicht einzubinden. Sogenannte Library-Projekte sind eigenständig zu entwickelnde Projekte, die die Funktonalität von Bibliotheken kapseln und sich sehr einfach in bestehende Projekte einbinden lassen.

Unter Berücksichtigung der Anforderungen fällt die logische Wahl also auf ein Android-Library-Projekt. Dazu wird das Framework wie ein normales Android-Projekt entwickelt. Die Einbindung in das Zielprojekt erfolgt komfortabel über die Android Developer Tools[2] (siehe Abbildung 4.4).

[2]Die Android Developer Tools sind ein, von Google bereitgestelltes, PlugIn für die Entwicklungsumgebung Eclipse. Es integriert eine einfache Verwaltung von androidtypischen Ressourcen und Projekten.

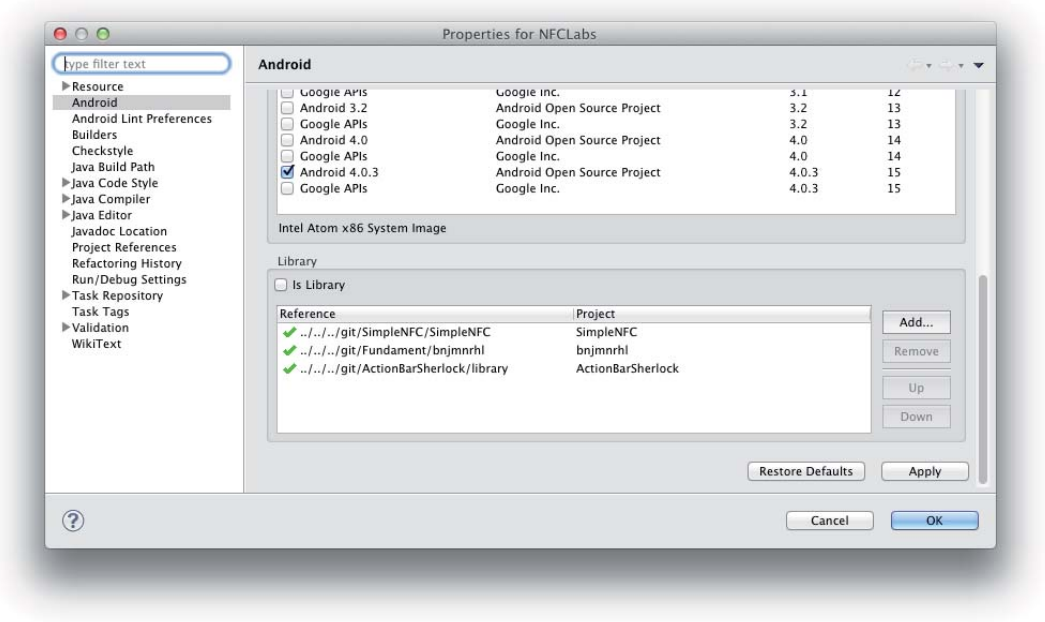

Abbildung 4.4: Einbindung eines Library-Projektes in ein Android-Projekt

Um die Komplexität so niedrig wie möglich zu halten, wird bei der Entwicklung des Frameworks auf die Hilfe weiterer Bibliotheken verzichtet. Zum Einen bläht dies das Ergebnis nicht unnötig auf, zum Anderen schließt es Überschneidungen mit anderen Bibliotheken aus, die später ebenfalls in die Applikation eingebunden werden. Damit technische Spezifikationen für den Entwickler ausblendet werden können, muss sich das Framework, anstelle des Entwicklers, für eine Formatierung von NFC-Nachrichten entscheiden. Um eine möglichst große Kompatibilität mit anderen Geräten, egal ob Smartphones oder andersartige Lesegeräte, zu gewährleisten, basiert das Framework auf dem Ndef-Format. Neben den zuvor genannten Vorteilen ist zu erwarten, dass es durch das NFC-Forum stetig weiterentwickelt wird. Damit die Komplexität weiter verringert wird, werden Nachrichten, die über NFC gelesen/geschrieben/gesendet werden, vereinfacht nur noch NfcMessages genannt. Dabei besteht eine NfcMessage weiterhin aus mindestens einem NfcRecord. NfcMessages respektive NfcRecords kapseln sämtliche Funktionen ihrer Namensväter, bieten jedoch erweiterte Funktionalität. Anstatt NdefRecords mit all ihren technischen Eigenschaften zu erstellen und dann an eine NdefMessage weiterzureichen, genügt ein Methodenaufruf der NfcMessage, damit die Nachricht um einen NfcRecord erweitert wird. Als Parameter müssen statt verschiedenen byte-Arrays lediglich eine ID und der Inhalt als String übergeben werden. Das gleiche Prozedere wird auf die drei verschiedenen Kommunikationsarten von NFC angewendet. Das Beschreiben von Tags, die direkte Kommunikation zweier Geräte und das erzwungene Auslesen von Tags wird in prozessorientierten Methoden gekapselt. Die Methodennamen bestehen, parallel zur Bezeichnung der NfcMessages bzw. NfcRecords, aus einer abstrakten Beschreibung

des Anwendungsszenarios. Für das Schreiben von Tags und den Austausch von Nachrichten zwischen zwei Geräten gibt es die Möglichkeit Event-Listener zu registrieren. Mit ihrer Hilfe ist es für den Entwickler möglich, gezielt auf Ereignisse, wie Fehler oder der Abschluss eines Schreibvorganges, zu reagieren. Bisher musste man beim Schreiben von NFC-Nachrichten bei jedem Schreibvorgang die Konfiguration des Tags abfragen. Ist er beschreibbar? Bietet der Tag genügend Speicherplatz? Ist er bereits formatiert? All diese Abfragen soll nun das Framework für den Entwickler übernehmen. Treten Probleme auf, die automatisiert zu lösen sind, wie zum Beispiel das Formatieren eines bisher unformatierten Tags, werden sie automatisch vom System behoben. Über alle anderen Fehler, die eine Interaktion des Entwicklers erfordern, wird der Entwickler über Exceptions informiert. Dieses Vorgehen fügt sich nahtlos in die Reihe bestehender Vorangehensweisen für die Ausnahmenbehandlung bei Ein- und Ausgabeoperationen ein. Eine weitere grundlegende Vereinfachung bietet die Möglichkeit NfcRecords mit einem Kontext zu versehen. Dazu können einzelne Activities als verantwortliche Instanzen für NfcRecords definiert werden. Die Zuordnung geschieht dafür über die ID des jeweiligen Records. Sobald ein Tag ausgelesen wird und dieser einen NfcRecord enthält, für den eine verantwortliche Activity registiert wurde, startet das System automatisch die verknüpfte Activity und übergibt die NfcMessage. Somit können gezielt Activities aufgerufen werden, ohne dass lästige Intent-Filter in der AndroidManifest.xml-Date definiert werden müssen. Ein letzter Aspekt, den es noch umzusetzen gilt, ist eine möglichst einfache Handhabung verschiedener Android-Versionen. Um den Entwickler auch in dieser Hinsicht zu unterstützen, wird es Methoden geben, die Informationen über die Betriebssystem-Version und dessen Funktionalität liefern werden. Wichtig ist jedoch, dass Entwickler kein Wissen über die unterschiedlichen API-Level vorweisen müssen. Alle Informationen werden in gewohnt abstrakter, prozessorientierter Weise vermittelt. Nutzt man verschiedene Methoden von verschiedenen API-Leveln, kann es mitunter vorkommen, dass bereits veraltete, als deprecated markierte Methoden verwendet werden. Das ist im Allgemeinen nicht gewünscht und gilt als schlechter Programmierstil. Oft ist es deshalb eine Gradwanderung veraltete Methoden weiterhin durch Frameworks verfügbar zu machen. Jedoch liegt es in der Verantwortung der Entwickler abzuwägen, inwieweit sie den Empfehlungen der Dokumentation nachkommen wollen und veraltete Methoden vernachlässigen. Damit Entwickler sich dessen bewusst sind, werden sämtliche Methoden, die sich veralteter Funktionalität bedienen, ebenfalls als deprecated markiert.

4.5 Implementierung

4.5.1 Kapselung bestehender Funktionalität

Die Verwendung von NFC unter Android ist mit gewissen Einschränkungen verbunden. Im besten Fall muss der Entwickler mit einem erhöhten Mehraufwand und einer gesteigerten Komplexität rechnen. Im schlimmsten Fall verkomplizieren sie die Einbindung von NFC und sorgen für kritische Fehler, die die Applikation zum Absturz bringen können. Diese Unzulänglichkeiten lassen sich durch ein einfaches Verfahren eliminieren: die Kapselung der verantwortlichen Logik in einer weiteren Klasse. Notwenige Codezeilen werden aus der Architektur der eigentlichen Applikation heraus in eine weitere Klasse extrahiert. Dadurch

geben Entwickler die Verantwortung an ein Framework ab und können sich vollends auf die Programmierung der eigentlichen Applikation konzentrieren. Dieses Kapitel bestimmt alle Einschränkungen und beschreibt, wie SimpleNFC diese durch Kapselung der verantwortlichen Logik in einer neuen Klasse behebt. Allen Einschränkungen liegt der Activity Lifecycle zugrunde. Android erlaubt den Aufruf von bestimmten Methoden erst dann, wenn sich die Activity in einem sichtbaren Zustand befindet, also nach oder innerhalb der *onResume*-Callback-Funktion. Diese Einschränkung betrifft die Methoden *enableForegroundDispatch()* und *enableForegroundNdefPush()* der Klasse *NfcAdapter*. Beide registrieren eine Activity als verantwortliche Instanz für die Kommunikation über NFC. Erstere Methode liefert den gelesenen Tag an die Activity. Diese kann die enthaltenen Daten nun auslesen oder den Tag neu beschreiben. Die Zweite ermöglicht die Kommunikation mit einem weiteren aktiven Gerät. Sobald eine Activity auf diesem Wege registriert wurde, muss sie auch wieder vom System abgemeldet werden. Auch für das Abmelden ist ein genauer Zeitpunkt festgelegt: Vor dem Pausieren der Activity, also vor oder während der *onPause*-Callback-Funktion.

Hinweis Activity Lifecycle
Applikationen unter Android bestehen aus einer Aneinanderreihung von verschiedenen Activities. Jede Activity erfüllt einen ganz bestimmten Zweck und stellt dafür ihre eigene Oberfläche zur Verfügung. Eine E-Mail Applikation könnte beispielsweise aus zwei verschiedenen Activities bestehen: Activity A zeigt alle E-Mails in einer Liste an und Activity B stellt den eigentlichen Inhalt einer E-Mail dar. Sobald der Nutzer aus der Liste eine E-Mail auswählt, wird Activity B gestartet. Sie legt sich dabei über Activity A, die nun nicht mehr zu sehen ist. Wie der Name schon vermuten lässt, beschreibt der Activity Lifecycle den Lebenszyklus einer Activity. Von der Instanziierung bis hin zur Zerstörung kann sie bis zu sieben verschiedene Status durchlaufen. Entwickler können und müssen je nach aktuellem Status Veränderungen an der Activity vornehmen. Jede Statusveränderung wird dem Entwickler durch verschiedene Callback-Methoden kundgetan. Die wichtigsten Callback-Methoden lauten:

onCreate wird aufgerufen, sobald die Activity erstellt ist. Meist werden hier all die Ressourcen geladen und Objekte instanziiert, die die Activity im Laufe ihrer Existenz benötigt. In jedem Fall wird in dieser Methode die Oberfläche für die Activity definiert.

onStop wird aufgerufen, sobald die Activity nicht mehr zu sehen ist. Entweder ist die Activity dazu zerstört worden oder ein neue Activity überlagert die alte. Diese Callback-Funktion eignet sich um Ressourcen freizugeben.

onResume wird aufgerufen, sobald eine Activity wieder in den Vordergrund tritt. Wird beispielsweise Activity B geschlossen, weil der Benutzer die E-Mail löscht, tritt die Liste mit den verbliebenen E-Mails in den Vordergrund und onResume wird aufgerufen. Innerhalb dieser Methode können Datensätze erneuert werden, um die Activity wieder auf den aktuellen Stand zu bringen.

onDestroy wird aufgerufen, sobald eine Activity zerstört wird. Sie eignet sich parallel zu onStop-Methode gut um Ressourcen freizugeben.

Die Lösung dieses Problem ist eine der Kernverantwortlichkeiten der Klasse *Nfc*. Sie kümmert sich um den zeitlich korrekten Aufruf der Methoden. Voraussetzung für diese Aufgabe ist die Kenntnis über den aktuellen Zustand der Activity. Entwickler rufen dazu nach der Instanziierung eines *Nfc*-Objektes die zwei Methoden *onResume* und *onPause* auf. Listing 4.2 zeigt das Gerüst eine Activity mit allen benötigten Methodenaufrufen, die das Framework voraussetzt.

```
1 public class Example extends Activity {
2   private Nfc mNfc;
3
4   @Override
5   protected void onCreate(Bundle savedInstanceState) {
6     super.onCreate(savedInstanceState);
7
```

```
8       mNfc = new Nfc(this);
9     }
10
11    @Override
12    protected void onPause() {
13      super.onPause();
14      mNfc.onPause();
15    }
16
17    @Override
18    protected void onResume() {
19      super.onResume();
20      mNfc.onResume();
21    }
22  }
```

Listing 4.2: Gerüst einer Activity und deren Einbindung von SimpleNFC

Für Anwender des SimpleNFC Frameworks bildet die Klasse *Nfc*-Klasse den Einstiegspunkt in das Framework. Ihr teilt er den aktuellen Zustand der Activity mit und sie stellt für ihn die Schnittstelle zu jeglicher NFC-Funktionalität dar. Dafür kann er aus einem Fundus bestehender Methoden schöpfen. Alle verfügbaren Methoden lassen sich in drei Kategorien unterteilen:

Meta-Methoden Methoden dieser Kategorie liefern lediglich Informationen über die NFC-Konfiguration des Smartphones.

Kommunikations-Methoden Methoden dieser Kategorie beschränken sich auf die Kommunikation mit unterschiedlichen Geräten. Sie realisieren die oft erwähnten Anwendungsszenarien und ermöglichen beispielsweise das Beschreiben von Tags oder den Austausch von Daten zwischen zwei aktiven Geräten.

Kontext-Methoden Diese Methoden implementieren den Kontextbezug zwischen einem NfcRecord und einer Activity.

Charakteristisch für alle Methoden dieser Klasse ist eine abstraktere Namensgebung. Der Name gibt Aufschluss über dessen Funktionalität, verbirgt jedoch technische Andeutungen. So hätte man die Methode *supportsBeam()* auch *isICS()* nennen können. Hinter beiden Namen steckt die gleiche Funktionalität, nämlich die Überprüfung des API-Levels. Ist das API-Level 14 oder höher, so liegt mindestens die Betriebssystem-Version Ice Cream Sandwich vor und die Beam-Technologie wird unterstützt. Allerdings ist die letzte Art der Namensgebung nicht sofort begreiflich und passt somit nicht in das Gesamtkonzept von SimpleNFC. Namen von Methoden beziehen sich stets auf ein bestimmtes Anwendungsszenario und sollten nicht erst in der Dokumentation nachgeschlagen werden müssen.

Neben *supportsBeam()*, welche Aufschluss über die Unterstützung von Android Beam liefert, gibt es in Kategorie der Meta-Methoden noch *isAvailable()* und *containsMessage()*. Erstere liefert als Ergebnis true zurück, sofern das Smartphone die NFC-Technologie unterstützt, emphfalse, wenn sie nicht vorhanden ist. *containsMessage()* erwartet als Parameter

ein Objekt vom Typ Intent. Die Methode überprüft, ob das Intent-Objekt eine NfcMessage vorhält. Wenn ja, gibt sie true, wenn nein, false zurück. In Kapitel 4.5.3 wird beschrieben, wann diese Methode zum Einsatz kommt.

Die Kategorie der Kommunikations-Methoden besteht aus *writeToTag()*, *beamToDevice()*, *pushToDevice()*, *dispatchAllMessages()* und *reset()*. Ausgenommen von *reset()* bietet jede Methode eine andere Art der Kommunikation an. Mit *writeToTag()* ist das Beschreiben von Tags möglich. *beamToDevice()* nutzt, auf Geräten die mindestens mit Android-Version 4.0 laufen, die Beam-Technologie um Nachrichten zwischen zwei Geräten auszutauschen. Gleiche Funktionalität bietet *pushToDevice()* mit dem Unterschied, dass statt der Beam-Technologie veralteter Code verwendet wird. Mit *dispatchAllMessages()* erzwingt eine Applikation den Zugriff auf gelesene Tags. Dadurch können Tags gelesen werden, die normalerweise automatisch an andere Applikationen weitergegeben werden. SimpleNFC kann innerhalb einer Activity die Art der Kommunikation beliebig verändern. Es ist also möglich mit *writeToTag()* einen Schreibvorgang zu beginnen und danach die Kommunikation mittels *beamToDevice()* auf Push zu ändern. Außerdem können Entwickler mittels *reset()* alle begonnenen Kommunikationsarten gestoppt bzw. abgebrochen werden. Damit die Kommunikation auch ohne die Einschränkungen des Android Lifecycles funktioniert, muss *Nfc* alle dafür benötigten Informationen, wie zum Beispiel die NfcMessage und Art der Kommunikation, vorhalten. Dazu versetzt sich die Klasse *Nfc* selbst in fünf unterschiedliche Modi.

SLEEP ist der Standard-Modus. Sobald eine Instanz von *Nfc* erstellt oder die Methode *reset()* aufgerufen wird, befindet sie sich in diesem Modus.

WRITE ist der Modus, welcher durch *writeToTag()* aufgerufen wird. Das Framework wartet nun auf das Erkennen eines Tags, um ihn zu beschreiben.

BEAM symbolisiert den Status des Beamens. Sobald ein zweites aktives Gerät erkannt wird, startet das Framework den Austausch der NfcMessage. Dies wird eingeleitet durch *beamToDevice()*.

PUSH ist ähnlich dem Modus BEAM. Auch hier wird auf ein zweites aktives Gerät gewartet, um eine NfcMessage zu senden. Wird eingeleitet durch *pushToDevice()*.

FOREGROUND beschreibt den Modus, in welchem alle in Reichweite befindlichen Tags abgefangen werden. Dies wird eingeleitet durch *dispatchAllMessages()*.

Listing 4.3 zeigt einen Vorteil dieses Prozederes. Die Anweisung zum Beschreiben eines Tags kann schon vor dem Aufruf von onResume, z.B. innerhalb der onCreate-Callback-Methode, erfolgen. SimpleNFC merkt sich die dafür notwendigen Informationen und startet die Kommunikation, sobald die Activity in den sichtbaren Status wechselt.

```
1 public class Example extends Activity {
2   private Nfc mNfc;
3
4   @Override
5   protected void onCreate(Bundle savedInstanceState) {
```

```
6       super.onCreate(savedInstanceState);
7
8       NfcMessage message = new NfcMessage();
9       message.addRecord("url", "http://www.benjamin-ruehl.de");
10
11      mNfc = new Nfc(this);
12      mNfc.writeToTag(message);
13    }
14
15    @Override
16    protected void onPause() {
17      super.onPause();
18      mNfc.onPause();
19    }
20
21    @Override
22    protected void onResume() {
23      super.onResume();
24      mNfc.onResume();
25    }
26  }
```

Listing 4.3: Kommunikations-Methoden können bereits in der onCreate-Methode aufgerufen werden.

SimpleNFC erweitert die Kommunikation außerdem um Callback-Funktionen. Für die Methoden writeToTag() und *beamToDevice()* gibt es jeweils eine überlagerte Methode, die neben der NfcMessage noch einen Event-Listener erwartet. Auch emphdispatchAllMessages() erwartet einen Event-Listener, dieser ist jedoch Pflicht. Je nach Methode kann der Event-Listener ein Objekt folgender Typen, implementiert als Interface, sein:

NfcWriteListener ermöglicht beim Schreiben von Nachrichten eine Rückmeldung über Erfolg oder Misserfolg. Bei einem Misserfolg können verschiedene Callback-Methoden aufgerufen werden. Das übergebene Exception-Objekt beschreibt den Grund des Misserfolges näher. Mögliche Exceptions sind: *NfcDisabledException*, *IOException*, *FormatException*, *ReadOnlyException*, *LowCapacityException*, *NDEFException*.

NfcBeamListener bietet eine Callback-Funktion an, welche aufgerufen wird, sobald die Nachricht erfolgreich übermittelt wurde.

NfcForegroundListener schreibt zwei Callback-Methoden vor. Sobald ein Tag gelesen wird, werden dessen Daten interpretiert. Liegen die Daten in Form einer NfcMessage vor, wird die NfcMessage-Callback-Funktion aufgerufen. Bei einer Formatierung als NdefMessage wird die NdefMessage-Callback-Funktion aufgerufen. Als Parameter wird die jeweilige Message geliefert.

SimpleNFC strukturiert seine Komponenten in sechs Paketen. Das Paket *activity* enthält die Activity *NfcConnector*. Sie implementiert den Zugriff auf fremde Geräte oder Tags. *exception* innerhalb des *entity*-Pakets beinhaltet mögliche Exceptions. *NfcMessage* und *NfcRecord* befinden sich ebenfalls innerhalb des *entity*-Pakets. Listener, die als Interface für

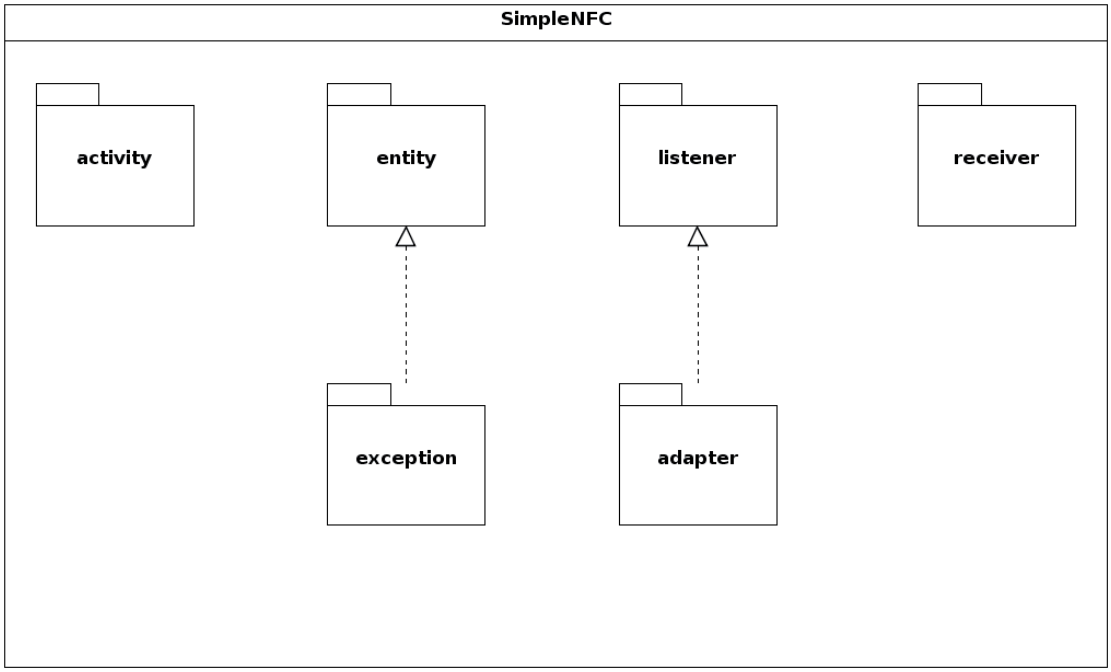

Abbildung 4.6: Paketdiagramm SimpleNFC

Rückmeldungen der Kommunikations-Methoden gedacht sind, befinden sich innerhalb des Pakets *listener*. Das Paket beinhaltet außerdem ein weiteres Paket: *adapter* mit der Klasse *NfcWriteAdapter*. Das letzte Paket *receiver* beinhaltet zwei BroadcastReceiver, die für die Kommunikation mit der *NfcConnector*-Activity benötigt werden. Weitere Informationen dazu liefern die nächsten Kapitel.

4.5.2 Nachrichtenerstellung

Charakteristisch für SimpleNFC ist ein abstrakter Umgang mit NFC. Dies zeigt sich beim Nachrichtensystem. NdefMessage und NdefRecord wurden aufgrund der Vereinfachung in den neuen Klassen *NfcMessage* und *NfcRecord* gekapselt. Eine *NfcMessage* besteht dazu aus verschiedenen Methoden, die den Umgang mit NdefMessages vereinfachen, sowie der eigentlichen NdefMessage. Methoden mit Auswirkungen auf die Struktur der NdefMessage bauen diese bei jedem Aufruf neu auf. Wird beispielsweise ein neuer Record mittels *addRecord()* hinzugefügt, werden alle NdefRecords aus der vorgehaltenen NdefMessage in eine Liste extrahiert, die Liste um den NdefRecord des *NfcRecord*s erweitert und eine neue NdefMessage aus der Liste der NdefRecords instanziiert. *NfcRecord*s sind ähnlich aufgebaut, beschränken sich jedoch auf Methoden, die den Inhalt beschreiben und liefern.

Bisher war die Erstellung von NdefMessages stets durch protokolltypische Konfigurationen geprägt. SimpleNFC hält diese Konfigurationen von Entwicklern fern und gestaltet die Erstellung von NFC-Nachrichten als Prozess. Vergleicht man Listing 4.3 mit Listing 4.4 wird deutlich, wie einfach der Erstellungsprozess aufgebaut ist.

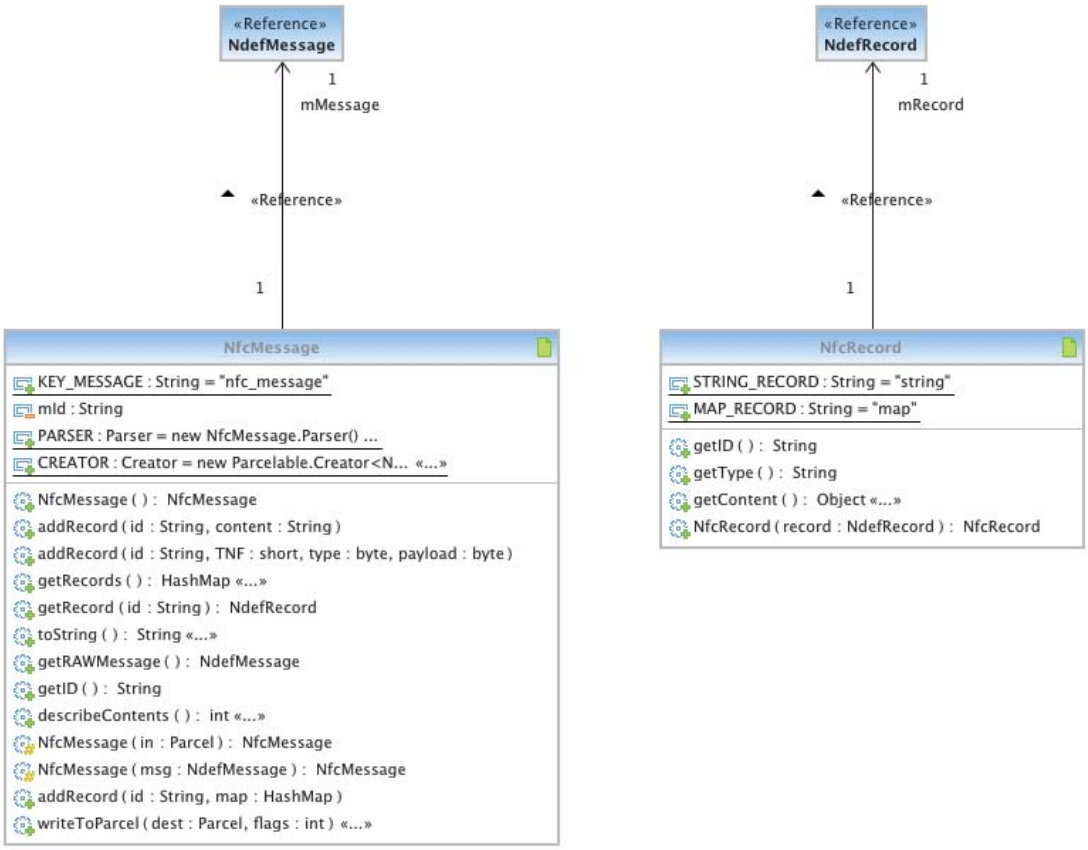

Abbildung 4.7: Klassendiagramm NfcMessage und NfcRecord

```
1  public class Example extends Activity {
2    @Override
3    protected void onCreate(Bundle savedInstanceState) {
4      super.onCreate(savedInstanceState);
5
6      NdefMessage message;
7      NdefRecord record = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
8          NdefRecord.RTD_TEXT,
9          "url".getBytes(),
10         "http://www.benjamin-ruehl.de".getBytes());
11
12     message = new NdefMessage(new NdefRecord[]{record});
13
14     this.write(message);
15   }
16
17   @Override
18   protected void onPause() {
19     super.onPause();
```

```
20      mNfc.onPause();
21    }
22
23    @Override
24    protected void onResume() {
25      super.onResume();
26      mNfc.onResume();
27    }
28
29    private void write(NdefMessage message){
30      [...]
31    }
32  }
```

Listing 4.4: Erstellung einer NdefMessage

Durch überlagerte Methoden der *NfcMessage* ist es möglich, verschiedene Arten von *NfcRecord*s zu erstellen. Neben einfachem Text erlaubt SimpleNFC ebenfalls HashMaps als Inhalt eines *NfcRecord*s zu definieren. Dazu serialisiert SimpleNFC die HashMap mittels OutputStream und speichert den daraus resultierenden byte-Array als Inhalt des *NfcRecord*s ab. Sobald per *getContent()* von *NfcRecord* der Inhalt wieder abgerufen wird, erkennt der *NfcRecord* die Art des *NfcRecord*s anhand des type-Feldes und interpretiert seinen byte-Inhalt.

```
1  public void addRecord(String id, HashMap<String, String> map) throws
       IOException, ClassNotFoundException{
2    ArrayList<NdefRecord> records = new ArrayList<NdefRecord>(Arrays.
         asList(mMessage.getRecords()));
3
4    ByteArrayOutputStream byteOut = new ByteArrayOutputStream();
5    ObjectOutputStream objectOut = new ObjectOutputStream(byteOut);
6    objectOut.writeObject(map);
7    objectOut.close();
8
9    records.add(new NdefRecord(NdefRecord.TNF_WELL_KNOWN, NfcRecord.
         MAP_RECORD.getBytes(), id.getBytes(), byteOut.toByteArray()));
10   mMessage = new NdefMessage(records.toArray(new NdefRecord[]{}));
11 }
```

Listing 4.5: NfcRecord mit HashMap als Inhalt

Für individuelle *NfcRecord*s gibt es weiterhin die Möglichkeit *NfcRecord*s auf die protokolltypische Art und Weise zu erstellen. SimpleNFC bedient sich den übergebenen Parametern und leitet diese an den Konstruktor des NdefRecords weiter. Lediglich die Identifikationszeichenkette (im folgenden ID genannt) muss dazu in einen byte-Array umgewandelt werden.

```
1  public void addRecord(String id, short TNF, byte[] type, byte[] payload
       ){
2      ArrayList<NdefRecord> records = new ArrayList<NdefRecord>(Arrays.
           asList(mMessage.getRecords()));
3      records.add(new NdefRecord(TNF, type, id.getBytes(), payload));
4      mMessage = new NdefMessage(records.toArray(new NdefRecord[]{}));
5  }
```

Listing 4.6: Erstellung eines NfcRecords auf die bisher übliche Art durch Angabe von protokolltypischen Informationen

Neben dem eigentlichen Inhalt benötigt jeder *NfcRecord* noch eine Zeichenkette für die Identifizierung. Mit Hilfe dieser Zeichenkette können *NfcRecord*s um einen Kontextbezug erweitert werden (siehe Kapitel 4.5.4). Mit SimpleNFC ist es möglich, eine *NfcMessage* an eine bestimmte Applikation zu binden. Beim Lesen solch einer Nachricht wird anstelle einer Auswahlbox mit allen kompatiblen Applikationen direkt die Zielapplikation geöffnet. Applikationsgebundene Nachrichten definieren als ersten Record einen MIME-Type-Record. Das type-Feld beinhaltet dann den Namespace der Applikation. Entwickler können sich solche *NfcMessage*s durch die Methode *obtainMessagBoundToThisApp()* der *Nfc*-Klasse erstellen lassen. Die Zuordnung von NdefMessage zu Applikation übernimmt das Betriebssystem durch einen Abgleich des eingetragenen Namespaces und allen verfügbaren Intent-Filtern. Für SimpleNFC ist die Einbindung der NfcConnector-Activity samt Definition des Intent-Filter obligatorisch. Kapitel 4.5.3 beschreibt die Einbindung der dafür notwendigen *NfcConnector*-Klasse und deren Nachrichtenverarbeitung.

```
1  public NfcMessage obtainMessageBoundToThisApp(String messageId) {
2    NfcMessage message = new NfcMessage();
3    message.addRecord(messageId, NdefRecord.TNF_MIME_MEDIA,
4        ("application/" + mActivity.getPackageName()).getBytes(),
5        new byte[] {});
6    return message;
7  }
```

Listing 4.7: Erstellung einer NfcMessage, die an eine Applikation gebunden ist

4.5.3 Nachrichtenverarbeitung

NFC-Nachrichten können unter Android auf verschiedenen Wegen empfangen und gesendet werden. SimpleNFC stellt dazu eine verantwortliche Instanz zur Verfügung, die alle Kommunikationskanäle überwacht und darüber kommunizieren kann: die Klasse *NfcConnector*. Sie stellt die Schnittstelle zwischen SimpleNFC und Android dar. Informationen, die durch SimpleNFC in einer abstrakteren Form vorliegen, werden von ihr so angepasst, dass sie durch Methoden von Android verarbeitet werden können. Andersherum werden Informationen, die durch das Betriebssystem empfangen werden, in eine abstraktere Form aufbereitet und an

die verantwortlichen Klassen von SimpleNFC weitergegeben. Dazu kann der *NfcConnector* auf drei verschiedene Arten genutzt werden:

Informationen lesen Der *NfcConnector* empfängt gelesene Tags vom Betriebssystem und bereitet dessen Informationen auf.

Informationen schreiben Der *NfcConnector* beschreibt verfügbare Tags.

Informationen abfangen Der *NfcConnector* kann alle Nachrichten abfangen, ungeachtet deren Zuordnung zu anderen Applikationen.

Intent-Filter können unter Android sowohl für Activities als auch für BroadcastReceiver definiert werden. BroadcastReceiver besitzen im Gegensatz zu Activities keinerlei Oberflächen. Sie dienen der Verarbeitung von Daten im Hintergrund, eignen sich also perfekt für die Aufgaben des *NfcConnector*s. Bei der Verarbeitung von NFC-Tags macht Android jedoch eine Ausnahme: Intent-Filter für die Kommunikation mittels NFC dürfen nur für Activities definiert werden. Also wurde der *NfcConnector* als Unterklasse von Activity implementiert. Activities bringen jedoch Charaktereigenschaften mit sich, die für die Anwendung in SimpleNFC nicht erwünscht sind. So besitzen auch leere Activities eine Oberfläche, die dem Benutzer angezeigt wird. Um dieses Verhalten zu unterbinden, wird der Hintergrund der leeren Activity auf unsichtbar gesetzt und die Titelleiste ausgeblendet. Android bietet dafür von Haus aus schon ein bestehendes Theme[3] an. Zugewiesen wird das Theme in der AndroidManifest.xml-Datei. Wird SimpleNFC dazu benutzt um Daten mittels NFC auszutauschen, wird der *NfcConnector* als unsichtbare Activity gestartet. Sobald er seine Aufgaben abgeschlossen hat, schließt er sich von selbst. Für die Einbindung von SimpleNFC ist es also Pflicht, den *NfcConnector* in der AndroidManifest.xml-Datei einzutragen. Listing 4.8 zeigt das XML-Gerüst, welches lediglich um den Namespace der Zielapplikation erweitert werden muss.

```
1  <activity
2      android:name="de.simplenfc.activity.NfcConnector"
3      android:theme="@android:style/Theme.Translucent.NoTitleBar" >
4      <intent-filter>
5          <action android:name="android.nfc.action.NDEF_DISCOVERED" />
6
7          <data android:mimeType="application/de.bnjmnrhl.nfclabs" />
8
9          <category android:name="android.intent.category.DEFAULT" />
10     </intent-filter>
11 </activity>
```

Listing 4.8: Einbindung des *NfcConnector*s

[3] Themes verändern das Aussehen von Activities und deren Bedienelemente in vielerlei Hinsicht. Außerdem ist es möglich, UI-Elemente wie die Titelleiste, auszublenden.

Informationen lesen

Instanzen von *NfcConnector* werden stets durch das Versenden eines Intents gestartet. Lediglich die Absender unterscheiden sich je nach Art der Aufgabe. Einer der möglichen Absender ist das Android-Betriebssystem. Sobald es einen Tag ausliest, wertet das Tag Dispatch System die enthaltenen Daten aus und startet eine Activity, die die Daten verarbeiten kann. Der *NfcConnector* ist solch eine Activity, sie bildet den Einstiegspunkt der Applikation beim Lesen von Tags. Dafür wird ihr Eintrag in der AndroidManifest.xml-Datei um einen Intent-Filter erweitert. Standardmäßig leitet der Intent-Filter nur *NdefMessages* an die Activity weiter, die mit der gleichen Applikation geschrieben wurden. Dieses Verhalten kann durch weitere Intent-Filter konfiguriert werden. Sobald der *NfcConnector* gestartet wird, beginnt er den Tag zu bearbeiten. Dazu extrahiert er aus dem Intent alle verfügbaren NdefMessages und bereitet die Daten auf, indem er jede NdefMessage in eine *NfcMessage* umwandelt. Anhand der ID überprüft SimpleNFC, ob es Activities gibt, die sich für die *NfcMessage* registriert haben (siehe Kapitel 4.5.4). Sofern es eine verantwortliche Activity gibt, wird diese gestartet. Gibt es keine, startet er stattdessen die Standard-Activity der Applikation.

```
1  private void readMessage(Intent intent){
2    if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())){
3      Parcelable[] rawMessages = intent.getParcelableArrayExtra(
           NfcAdapter.EXTRA_NDEF_MESSAGES);
4      if(rawMessages != null){
5        NfcMessage[] parsedMessages = new NfcMessage[rawMessages.length];
6
7        for (int i=0; i<rawMessages.length; i++) {
8          NdefMessage msg = (NdefMessage) rawMessages[i];
9          parsedMessages[i] = NfcMessage.PARSER.parseFromNdefMessage(msg)
               ;
10       }
11
12       boolean processed = new Nfc(this).handleMessages(parsedMessages);
13       if(!processed){
14         PackageManager manager = this.getPackageManager();
15         Intent launchIntent = manager.getLaunchIntentForPackage(this.
               getPackageName());
16         this.startActivity(launchIntent);
17       }
18     }
19
20     this.finish();
21   }
22 }
```

Listing 4.9: Die Methode *readMessage()* der Klasse *NfcConnector* wird beim Lesen von Tags aufgerufen und interpretiert die enthaltenen Daten

Informationen schreiben

Für das Beschreiben von Tags ist das Framework der verantwortliche Sender des Intents. Bis das Intent abgesendet werden kann, muss jedoch ein wenig Vorarbeit geleistet werden. Diese Vorarbeit ist der erste Part des Schreibvorgangs, Part 2 implementiert den eigentlichen Zugriff auf den Tag. Beide Parts sind zeitlich voneinander getrennt und werden von unterschiedlichen Klassen implementiert. Die Klasse *Nfc* realisiert die Vorarbeit in der Methode *enableWriteMode()* (siehe Listing 4.10).

```
1  private void enableWriteMode(){
2    if(Nfc.DEBUG)Log.v(TAG, "\tprepare to write message "+mMessage);
3
4    mNfcReceiver = new NfcConnectorStateReceiver(mWriteListener){
5
6      @Override
7      public void onReceive(Context context, Intent intent) {
8        super.onReceive(context, intent);
9
10       mMode = Mode.SLEEP;
11       mMessage = null;
12       mWriteListener = null;
13     }
14
15   };
16
17   IntentFilter filter = new IntentFilter(NfcConnectorStateReceiver.
         ACTION_STATECHANGED);
18   mActivity.registerReceiver(mNfcReceiver, filter);
19
20   Intent intent = new Intent(mActivity, NfcConnector.class);
21   intent.setExtrasClassLoader(NfcMessage.class.getClassLoader());
22   intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
23   intent.putExtra(NfcConnector.EXTRA_MODE, NfcConnector.MODE_WRITE);
24   intent.putExtra(NfcMessage.KEY_MESSAGE, mMessage.getRAWMessage().
         toByteArray());
25
26   PendingIntent pendingIntent= PendingIntent.getActivity(mActivity, 0,
         intent, PendingIntent.FLAG_UPDATE_CURRENT);
27   NfcAdapter.getDefaultAdapter(mActivity).enableForegroundDispatch((
         Activity) mActivity, pendingIntent,
28       new IntentFilter[] { new IntentFilter(NfcAdapter.
             ACTION_TAG_DISCOVERED) }
29   , null);
30 }
```

Listing 4.10: Erstellung des *NfcConnector*-Intents und Kapselung in einem PendingIntent

Dazu erstellt sie ein Intent, welches den *NfcConnector* startet. Als Zusatzinformationen werden dem Intent die zu schreibende Nachricht als byte-Array und eine Modusanweisung hinzugefügt. Durch die Modusanweisung weiß der *NfcConnector* später, dass er den verfügbaren Tag beschreiben soll. Damit das Intent erst ausgeführt wird, wenn ein Tag in

Reichweite ist, kapselt *Nfc* es in einem PendingIntent. Das resultierende PendingIntent wird im Anschluss durch die native NFC-Schnittstelle von Android mit Hilfe eines Intent-Filters registriert. Ebenfalls zu den Vorarbeiten gehört die Instanziierung des *NfcConnectorStateReceiver*. Er wird nach dem eigentlichen Schreibvorgang dazu verwendet, Meldungen über eventuelle aufgetretene Fehler oder über das erfolgreiche Beschreiben an die Activity zu übertragen. Mit der Registrierung des PendingIntents ist die Vorarbeit abgeschlossen. Der Schreibvorgang wird fortgesetzt sobald das System einen Tag erkennt.

Hinweis PendingIntent
PendingIntents sind „schlafende"Intents. Sie beinhalten ein normales Intent, welches auf die übliche Art und Weise erstellt und konfiguriert werden kann. Der Charakter eines schlafenden Intents kommt von dem zeitlichen Abstand zwischen Zeitpunkt der Instanziierung und Ausführung. Meist werden sie benutzt um Aktionen zu definieren, die durch in der Zukunft auftretende Ereignisse ausgelöst werden sollen. Sobald das Ereignis eintritt, wird das enthaltene Intent gestartet. Da es in der eigenen Applikation erstellt wurde, besitzt es alle Rechte der eigenen Applikation. Man erlaubt mit der Weitergabe des PendingIntents, also fremden Applikationen, den Zugriff auf die eigene Applikation.

Sobald das Tag Dispatch System einen Tag in Reichweite erkennt startet es das zuvor registrierte PendingIntent und der *NfcConnector* wird gestartet. Direkt nach der Instanziierung des *NfcConnector*s wird aus dem Intent die Modusanweisung ausgelesen (siehe Listing 4.11).

```
1  private void handleIntent(Intent intent){
2    if (intent.hasExtra(EXTRA_MODE) && intent.getStringExtra(EXTRA_MODE).
         equals(MODE_WRITE)) {
3      this.writeMessage(intent);
4    } else if (intent.hasExtra(EXTRA_MODE) && intent.getStringExtra(
         EXTRA_MODE).equals(MODE_READ)){
5      this.readMessage(intent);
6    } else if(intent.hasExtra(EXTRA_MODE) && intent.getStringExtra(
         EXTRA_MODE).equals(MODE_FOREGROUND)){
7      this.handleForegroundMessage(intent);
8    }else {
9      this.readMessage(intent);
10   }
11 }
```

Listing 4.11: Unterscheidung der verschiedenen Modi eines *NfcConnector*

Beinhaltet das Intent die Modusanweisung zum Beschreiben von Tags, so wird der Schreibmodus mit der Methode *writeMessage()* gestartet. *NfcConnector* unterteilt den Schreibvorgang in zwei Methoden. *write()* implementiert den Schreibvorgang einer NdefMessage durch die Android-API. Ihre Aufgabe ist es, zu überprüfen, ob der Tag im Ndef-Format vorliegt, er beschreibbar ist und seine Kapazität ausreicht, um alle Daten der NdefMessage zu speichern. Sollten nicht alle Voraussetzungen für einen erfolgreichen Schreibvorgang gegeben sein, teilt *write()* dies durch Exceptions mit. Für jedes mögliche Problem definiert SimpleNFC

eine Exception. Die zweite, am Schreibvorgang beteiligte Methode, ist *writeMessage()*. Sie überprüft, ob das Smartphone NFC unterstützt, extrahiert die benötigte *NfcMessage* aus dem Intent, startet den Schreibvorgang mit *write()* und teilt der Activity das Ergebnis dieser Aufrufe mit. Die Kommunikation erfolgt dabei über das Senden von BroadcastIntents. Empfangen wird dieses Intent von der Klasse *NfcConnectorStateReceiver*, die die Activity zuvor instanziiert hat. Die Zuordnung von Intent zu *NfcConnectorStateReceiver* erfolgt über einen eindeutigen Ereignisnamen, welcher als statischer String durch die Klasse *NfcConnectorStateReceiver* selbst angeboten wird. Fehler, die innerhalb von *write()* auftreten, werden durch den try-catch-Block abgefangen und dem BroadcastIntent übergeben. Im Falle eines erfolgreichen Schreibvorgangs wird dem Intent ein Erfolgs-Indikator übergeben. Im Anschluss wird das Intent als BroadcastIntent gesendet und der *NfcConnector* schließt sich selbst.

```
1  private void writeMessage(Intent intent) {
2    Intent broadcastIntent = new Intent(NfcConnectorStateReceiver.
         ACTION_STATECHANGED);
3    Bundle b = new Bundle(1);
4
5    if(!NfcAdapter.getDefaultAdapter(this).isEnabled()){
6      b.putSerializable(NfcConnectorStateReceiver.
           EXTRA_EXCEPTION_NFCDISABLED, new NfcDisabledException());
7      this.sendBroadcast(broadcastIntent.putExtras(b));
8      this.finish();
9    }
10
11   if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(intent.getAction())) {
12     Tag detectedTag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
13     NfcMessage message = null;
14     try {
15       message = NfcMessage.PARSER.parseFromNdefMessage(new NdefMessage(
             intent.getByteArrayExtra(NfcMessage.KEY_MESSAGE)));
16     } catch (FormatException e) {
17       Log.e(TAG, e.getMessage());
18     }
19
20     try {
21       this.write(detectedTag, message);
22       if(Nfc.DEBUG){
23         NdefRecord records[] = message.getRAWMessage().getRecords();
24         String log = "\tNdefMessage written. Included Records:";
25         for (int i=0; i<records.length; i++) {
26           log += "\n\t["+i+"] tnf: "+records[i].getTnf()
27               +", type: "+new String(records[i].getType())
28               +", id: "+new String(records[i].getId())
29               +", payloadlength: "+records[i].getPayload().length
30               +", payload: "+new String(records[i].getPayload());
31         }
32
33         Log.v(TAG, log);
34       }
35     } catch (IOException e) {
```

```
36          Log.e(TAG, e.getMessage());
37          b.putSerializable(NfcConnectorStateReceiver.EXTRA_EXCEPTION_IO, e
                );
38        } catch (FormatException e) {
39          Log.e(TAG, e.getMessage());
40          b.putSerializable(NfcConnectorStateReceiver.
                EXTRA_EXCEPTION_FORMAT, e);
41        } catch (ReadOnlyException e) {
42          Log.e(TAG, e.getMessage());
43          b.putSerializable(NfcConnectorStateReceiver.
                EXTRA_EXCEPTION_READONLY, e);
44        } catch (LowCapacityException e) {
45          Log.e(TAG, e.getMessage());
46          b.putSerializable(NfcConnectorStateReceiver.
                EXTRA_EXCEPTION_LOWCAPACITY, e);
47        } catch (NDEFException e) {
48          Log.e(TAG, e.getMessage());
49          b.putSerializable(NfcConnectorStateReceiver.EXTRA_EXCEPTION_NDEF,
                 e);
50        } catch (Exception e) {
51          Log.e(TAG, e.getMessage());
52        }
53
54        if(b.size() == 0){
55          b.putBoolean(NfcConnectorStateReceiver.EXTRA_WRITTEN, true);
56        }
57
58        this.sendBroadcast(broadcastIntent.putExtras(b));
59        this.finish();
60      }
61    }
```

Listing 4.12: Implementierung des Schreibvorgangs

Hinweis BroadcastReceiver
In einem Betriebssystem wie Android treten regelmäßig die verschiedensten Ereignisse auf: E-Mails werden abgerufen, Anrufe entgegengenommen und SMS empfangen. Damit Applikationen auf diese Ereignisse reagieren können, wird jedes Ereignis mit einem eindeutigen Ereignisnamen versehen, in einem Intent gekapselt und losgesendet. Neben Applikationen gibt es noch ein weiteres Konstrukt, welches Intents empfangen und verarbeiten kann: der BroadcastReceiver. Im Gegenteil zu einer Activity besitzt der BroadcastReceiver keine Oberfläche. Deshalb eignet er sich besonders gut als Kommunikationsmittel zwischen Applikationen oder Activities. Um gezielt auf Ereignisse zu horchen, kann man unter Android BroadcastReceiver mit einem Intent-Filter beim Betriebssystem registrieren. Wie in Listing 4.13 zu sehen, erfolgt die Definition des Intent-Filters über den eindeutigen Ereignisnamen. Es ist außerdem möglich eigene Intents mit eigenen Ereignisnamen zu definieren. Dadurch kann man entweder eine Schnittstelle für andere Applikationen bereitstellen oder innerhalb der eigenen Applikation mehrere Activities miteinander kommunizieren lassen. BroadcastReceiver werden innerhalb von Activities registriert und noch vor dem Ausblenden der Activity abgemeldet.

```
1  public class Example extends Activity {
2    private BatteryReceiver mReceiver;
3
4    @Override
5    protected void onCreate(Bundle savedInstanceState) {
6      super.onCreate(savedInstanceState);
7
8      mReceiver = new BatteryReceiver();
9
10     IntentFilter filter = new IntentFilter(Intent.ACTION_BATTERY_LOW);
11     this.registerReceiver(mReceiver, filter);
12   }
13
14   @Override
15   protected void onStop() {
16     super.onStop();
17
18     this.unregisterReceiver(mReceiver);
19   }
20
21
22   private class BatteryReceiver extends BroadcastReceiver{
23
24     @Override
25     public void onReceive(Context context, Intent intent) {
26       // handle low battery state
27     }
28
29   }
30 }
```

Listing 4.13: Definition und Registrierung eines BroadcastReceivers zum Überwachen des Batteriestatus

Konstrukte wie der *NfcConnectorStateReceiver* ermöglichen SimpleNFC einen abstrakteren Aufbau des Fehlermanagements. Durch die Implementierung des *NfcConnector*s als Activity kommt es vor, dass im *NfcConnector* auftretende Fehler Einfluss auf andere Activities haben. Bisher war es nicht möglich solche Fehler gezielt an verantwortlichen Instanzen weiterzureichen. Dies ändert sich mit SimpleNFC. Auftretende Fehler werden in verschiedensten Exceptions gekapselt und mittels Intents an die verantwortlichen Stellen gesendet. Eine dieser verantwortlichen Stellen ist der *NfcConnectorStateReceiver*. Wie der Namen schon vermuten lässt, empfängt er Intents die Statusveränderungen des *NfcConnector*s mitteilen. Durch ihn ist es möglich, Exceptions innerhalb der verantwortlichen Activity zu behandeln. Außerdem bildet er die Grundlage für den *NfcWriteListener*, welcher auftretende Exceptions an den Entwickler weiterleitet.

```
1  public class NfcConnectorStateReceiver extends BroadcastReceiver {
2    public static final String ACTION_STATECHANGED = "
       simplenfc_statechanged";
3    public static final String EXTRA_EXCEPTION_NFCDISABLED = "
       nfcdisabledexception";
4    public static final String EXTRA_EXCEPTION_IO = "ioexception";
5    public static final String EXTRA_EXCEPTION_FORMAT = "formatexception"
       ;
6    public static final String EXTRA_EXCEPTION_READONLY = "
       readonlyexception";
7    public static final String EXTRA_EXCEPTION_LOWCAPACITY = "
       lowcapacityexception";
8    public static final String EXTRA_EXCEPTION_NDEF = "ndefexception";
9    public static final String EXTRA_WRITTEN = "tag_written";
10
11   private NfcWriteListener mListener;
12
13
14   public NfcConnectorStateReceiver(NfcWriteListener listener){
15     this.mListener = listener;
16   }
17
18
19   @Override
20   public void onReceive(Context context, Intent intent) {
21     if(intent.getAction().equals(ACTION_STATECHANGED)){
22       Bundle b = intent.getExtras();
23
24       if(b.containsKey(EXTRA_EXCEPTION_NFCDISABLED)){
25         this.mListener.onNfcException((NfcDisabledException)b.
             getSerializable(EXTRA_EXCEPTION_NFCDISABLED));
26       }else if(b.containsKey(EXTRA_EXCEPTION_IO)){
27         this.mListener.onNfcException((IOException)b.getSerializable(
             EXTRA_EXCEPTION_IO));
28       }else if(b.containsKey(EXTRA_EXCEPTION_FORMAT)){
29         this.mListener.onNfcException((FormatException)b.
             getSerializable(EXTRA_EXCEPTION_FORMAT));
30       }else if(b.containsKey(EXTRA_EXCEPTION_READONLY)){
```

```
31            this.mListener.onNfcException((ReadOnlyException)b.
                  getSerializable(EXTRA_EXCEPTION_READONLY));
32          }else if(b.containsKey(EXTRA_EXCEPTION_LOWCAPACITY)){
33            this.mListener.onNfcException((LowCapacityException)b.
                  getSerializable(EXTRA_EXCEPTION_LOWCAPACITY));
34          }else if(b.containsKey(EXTRA_EXCEPTION_NDEF)){
35            this.mListener.onNfcException((NDEFException)b.getSerializable(
                  EXTRA_EXCEPTION_NDEF));
36          }else if(b.containsKey(EXTRA_WRITTEN)){
37            context.unregisterReceiver(this);
38            this.mListener.onNfcMessageWritten();
39          }
40        }
41      }
42    }
```

Listing 4.14: Implementierung des NfcConnectorStateReceiver als Grundlage für den NfcWriteListener

Informationen abfangen

Die dritte Kommunikationsart ermöglicht das Abfangen von allen Tags. Normalerweise vermittelt Android nur Tags an Activities weiter, die deren Intent-Filtern entsprechen. Es ist jedoch auch möglich, den Empfang aller Tags ohne Unterscheidung auf Protokoll oder MIME-Type zu erzwingen. Diese Funktionalität wird durch die *Nfc*-Methode *enableForegroundDispatch()* gestartet. Ihr Aufbau ähnelt der Methode *enableWriteMode()*. Für die Kommunikation zwischen der Activity und *NfcConnector* wird die Klasse *NfcForegroundReceiver* instanziiert und mittels Intent-Filter registriert. Das *NfcConnector*-Intent benötigt eine Modusanweisung, die dem *NfcConnector* später mitteilt sich in den Foreground-Modus zu begeben. Parallel zu *enableWriteMode()* wird auch hier das Intent in einem PendingIntent gekapselt und mittels Intent-Filter bei Android's nativer NFC-Schnittstelle registriert.

```
1  private void enableForegroundDispatch(){
2    if(Nfc.DEBUG)Log.v(TAG, "\tprepare to read all messages");
3
4    IntentFilter filter = new IntentFilter(NfcForegroundReceiver.
         ACTION_MESSAGE_RECEIVED);
5    mActivity.registerReceiver(mForegroundReceiver, filter);
6
7    Intent intent = new Intent(mActivity, NfcConnector.class);
8    intent.addFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);
9    intent.putExtra(NfcConnector.EXTRA_MODE, NfcConnector.MODE_FOREGROUND
         );
10
11   PendingIntent pendingIntent= PendingIntent.getActivity(mActivity, 0,
         intent, PendingIntent.FLAG_UPDATE_CURRENT);
12   mAdapter.enableForegroundDispatch(mActivity, pendingIntent,
13       new IntentFilter[] { new IntentFilter(NfcAdapter.
             ACTION_TAG_DISCOVERED) }
```

```
14      , null);
15  }
```

Listing 4.15: *Nfc*-Schnittstelle zum Abfangen aller verfügbaren Tags

Auch hier startet das Betriebssystem den *NfcConnector* sobald sich ein verfügbarer NFC-Tag in Reichweite befindet. Der *NfcConnector* entscheidet auf Basis der Modusanweisung welche Art der Kommunikation stattfinden soll und ruft die Methode *handleForegroundMessage()* auf. Diese interpretiert die im Intent enthaltenen NFC-Nachrichten auf zwei verschiedene Arten und Weisen. Wurden die NdefMessages mittels SimpleNFC geschrieben, werden sie in *NfcMessage*s gekapselt. Alternativ werden sie weiterhin als NdefMessages vorgehalten. Sobald alle Nachrichten interpretiert wurden, wird ein BroadcastIntent mit dem Ereignisnamen des NfcForegroundReceivers erstellt und die Nachrichten hinzugefügt. Zum Schluss wird das BroadcastIntent abgesendet und der *NfcConnetor* schließt sich selbstständig.

```
1  private void handleForegroundMessage(Intent intent){
2    if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction())){
3      Parcelable[] rawMessages = intent.getParcelableArrayExtra(
           NfcAdapter.EXTRA_NDEF_MESSAGES);
4
5      if(rawMessages != null){
6        try{
7          NfcMessage[] parsedMessages = new NfcMessage[rawMessages.length
               ];
8
9          for (int i=0; i<rawMessages.length; i++) {
10           NdefMessage msg = (NdefMessage) rawMessages[i];
11           parsedMessages[i] = NfcMessage.PARSER.parseFromNdefMessage(
                 msg);
12         }
13
14         Intent broadcastIntent = new Intent(NfcForegroundReceiver.
               ACTION_MESSAGE_RECEIVED);
15         Bundle b = new Bundle(1);
16         b.putParcelableArray(NfcForegroundReceiver.EXTRA_NFCMESSAGE,
               parsedMessages);
17         this.sendBroadcast(broadcastIntent.putExtras(b));
18       }catch (Exception e) {
19         Log.e(TAG, "handleForegroundMessage", e);
20
21         NdefMessage[] parsedMessages = new NdefMessage[rawMessages.
               length];
22
23         for (int i=0; i<rawMessages.length; i++) {
24           parsedMessages[i] = (NdefMessage) rawMessages[i];
25         }
26
27         Intent broadcastIntent = new Intent(NfcForegroundReceiver.
               ACTION_MESSAGE_RECEIVED);
28         Bundle b = new Bundle(1);
```

```
29            b.putParcelableArray(NfcForegroundReceiver.EXTRA_NDEFMESSAGE,
                  parsedMessages);
30            this.sendBroadcast(broadcastIntent.putExtras(b));
31          }finally{
32            this.finish();
33          }
34        }
35      }
36    }
```

Listing 4.16: Interpretation der abgefangenen NdefMessages

Empfänger des BroadcastIntents ist die Klasse *NfcForegroundReceiver*. Sie empfängt Intents deren Ereignisnamen mit ihrer öffentlich String-Konstante *ACTION_MESSAGE_RECEIVED* übereinstimmen. Danach überprüft sie, ob das Intent *NfcMessage*s oder NdefMessages enthält und ruft den per Kontruktor übergebenen *NfcForegroundListener* auf. Instanzen der Klasse *NfcForegroundListener* besitzen zwei verschieden Methoden: *onNfcMessageReceived()* und *onNdefMessageReceived()*. Erstere liefert als Parameter die *NfcMessage*, letztere die NdefMessage. Sie ermöglichen dem Entwickler gezielt auf den Empfang bestimmter Typen von NFC-Nachrichten zu reagieren.

```
1   public class NfcForegroundReceiver extends BroadcastReceiver {
2     public static final String ACTION_MESSAGE_RECEIVED = "
          simplenfc_writer_statechanged";
3     public static final String EXTRA_NFCMESSAGE = "nfcmessage";
4     public static final String EXTRA_NDEFMESSAGE = "ndefmessage";
5
6     private NfcForegroundListener mListener;
7
8     public NfcForegroundReceiver(NfcForegroundListener listener){
9       mListener = listener;
10    }
11
12    @Override
13    public void onReceive(Context context, Intent intent) {
14      if(intent.getAction().equals(ACTION_MESSAGE_RECEIVED)){
15        Bundle b = intent.getExtras();
16
17        if(b.containsKey(EXTRA_NFCMESSAGE)){
18          mListener.onNfcMessageReceived((NfcMessage) b.getParcelable(
                EXTRA_NFCMESSAGE));
19        }else if(b.containsKey(EXTRA_NDEFMESSAGE)){
20          mListener.onNdefMessageReceived((NdefMessage)b.getParcelable(
                EXTRA_NDEFMESSAGE));
21        }
22      }
23    }
24  }
```

Listing 4.17: Implementierung des NfcForegroundReceiver als Unterklasse von BroadcastReceiver

4.5.4 Kontextbezug

SimpleNFC bietet die Möglichkeit *NfcMessage*s in einen direkten Bezug zu einer verantwortlichen Activity zu stellen. Das vereinfacht besonders die Verarbeitung von verschiedenen Typen von *NfcMessage*s. Die Schnittstelle für den Entwickler bildet die Methode *addMessageHandler()* der Klasse *Nfc*. Ihr muss man sowohl die ID der NFC-Nachricht als auch das Class-Objekt der verantwortlichen Activity übergeben.

```
1  public void addMessageHandler(String nfcMessageId, Class activity){
2    NfcMessageHandler.getInstance(mActivity).registerMessageId(
         nfcMessageId, activity);
3  }
```

Listing 4.18: *Nfc*-Schnittstelle für das Registrieren eines MessageHandlers

Intern wird die Zuordnung zwischen *NfcMessage* und Activity durch die Klasse *NfcMessageHandler* verwaltet. Für die Verwaltung der Zuordnungen bietet die Klasse zwei Methoden an. *registerMessageId()* fügt eine neue Zuordnung hinzu und erwartet als Parameter die ID der *NfcMessage* und das Class-Objekt der verantwortlichen Activity. Der Methode *handleMessages()* werden eine oder mehrere *NfcMessage*s übergeben. Sie sucht mittels ID der *NfcMessage* nach einer vorhandenen Zuordnung und führt, falls eine vorhanden sein sollte, diese aus. Ausgeführt wird eine Zuordnung, indem ein Intent mit dem Class-Objekt als Paramter erzeugt und gestartet wird. Android startet dann automatisch die gewünschte Activity. Dem Intent wird außerdem die benötigte *NfcMessage* mitgegeben.

```
1  public boolean handleMessages(NfcMessage[] messages){
2    if(messages == null || messages.length == 0) return false;
3
4    for (NfcMessage msg : messages) {
5      if(this.mMessageId2Class.containsKey(msg.getID())){
6        String className = this.mMessageId2Class.get(msg.getID());
7        Intent intent = null;
8        try {
9          intent = new Intent(this.mContext, Class.forName(className));
10       } catch (ClassNotFoundException e) {
11         Log.e(TAG, e.getMessage());
12       }
13       intent.putExtra(NfcMessage.KEY_MESSAGE, msg);
14
15       this.mContext.startActivity(intent.addFlags(Intent.
             FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TOP));
16       return true;
17     }
18   }
19   return false;
20 }
```

Listing 4.19: Verarbeitung von *NfcMessage*s

Aufgrund der Tatsache, dass Applikationen unter Android eine beschränkte Lebensdauer besitzen, ist es nötig, die Zuordnungen für die Zukunft zu sichern. Das Betriebssystem kann aus unterschiedlichen Gründen Applikationen schließen. Daher müssen die Zuordnungen während der Laufzeit serialisiert und mit der ersten Verwendung von SimpleNFC wieder deserialisiert werden. Das Serialisieren wird nach jeder Veränderung der Zuordnungen und das Deserialisieren bei der ersten Benutzung von SimpleNFC ausgeführt. Da sich der Zugriff auf Dateien in der Regel als zeitaufwenig erweist, wurde die Klasse *NfcMessageHandler* nach dem Singleton-Entwurfsmuster entwickelt. Dadurch reduziert sich die Anzahl der lesenden Zugriffe auf die Serialisierungsdatei auf einen pro Applikationsaufruf.

```
1  private void serializeToFile(){
2    try {
3      JSONObject json = new JSONObject(mMessageId2Class);
4
5      FileOutputStream fos = mContext.openFileOutput(FILENAME, Context.
           MODE_PRIVATE);
6      BufferedOutputStream bos = new BufferedOutputStream(fos);
7      bos.write(json.toString().getBytes());
8      bos.close();
9      fos.close();
10
11   } catch (FileNotFoundException e) {
12     e.printStackTrace();
13   } catch (IOException e) {
14     e.printStackTrace();
15   }
16 }
```

Listing 4.20: Serialisierung aller Zuordnungen zwischen Activity und *NfcMessage* in eine Datei

Die Zuordnungen werden in Form einer Hashtabelle vorgehalten. Sowohl Schlüssel als auch Wert dieser Hashtabelle sind vom Typ String. Da Android eine einfache Umwandlung von solchen Hashtabellen in JSON[4]-Objekte bietet und sich JSON-Objekte ohne Aufwand in Fließtext umwandeln lassen findet die Serialisierung über diesem Umweg des JSON-Parsers statt. Listing 4.20 zeigt deutlich, dass nur mit einer Zeile Code eine Hashtabelle in ein JSON-Objekt umgewandelt werden kann. Im Anschluss kann es mittels FileOutputStream in eine Datei geschrieben werden.

[4]JSON steht für JavaScript Object Notation. Es ist ein Datenformat für den Austausch zwischen Applikationen. JSON ermöglicht die Definition von Objekten und unterstützt eine Reihe von Datentypen.

```
1  private void deserializeFromFile(){
2    if(!Arrays.asList(mContext.fileList()).contains(FILENAME)){
3      return;
4    }
5
6    try {
7      FileInputStream fis = mContext.openFileInput(FILENAME);
8
9      BufferedReader br = new BufferedReader(new InputStreamReader(fis));
10     StringBuilder content = new StringBuilder();
11     String line;
12     while ((line = br.readLine()) != null) {
13         content.append(line);
14     }
15     fis.close();
16
17     mMessageId2Class = new HashMap<String, String>();
18     JSONObject json = new JSONObject(content.toString());
19     Iterator<String> keys = json.keys();
20     while (keys.hasNext()) {
21       String key = (String) keys.next();
22       mMessageId2Class.put(key, json.getString(key));
23     }
24
25   } catch (FileNotFoundException e) {
26     e.printStackTrace();
27   } catch (StreamCorruptedException e) {
28     e.printStackTrace();
29   } catch (IOException e) {
30     e.printStackTrace();
31   } catch (JSONException e) {
32     e.printStackTrace();
33   }
34 }
```

Listing 4.21: Deserialisierung aller Zuordnungen zwischen Activity und *NfcMessage* aus einer Datei

Der Deserialisierungs-Aufruf im privaten Konstruktor der Klasse. Die Methode *deserializeFromFile* ließt dazu die Serialisierungsdatei mittels FileInputStream ein. Aus dem Inhalt der Datei, welcher als String vorliegt, kann nun ein JSON-Objekt erstellt werden. Anschließend wird jeder Eintrag des JSON-Objekts in eine Hastabelle überführt und die Zuordnungen sind deserialisiert.

4.5.5 Kompatibilität mit älteren Versionen von Android

Kompatibilitätsprobleme zwischen verschieden alten Betriebssystemversionen können in zwei Teilbereiche eingeteilt werden:

Neue Funktionalitäten sind meist Bestandteil von neuen Betriebssystemversionen. Sie erweitern die API um neue Schnittstellen.

API-Veränderungen treten auch bei neuen Betriebssystemversionen auf. Meist werden bestehende Methoden als veraltet markiert und auf neue verwiesen.

Ein high-level-Framework kann den ersten Aspekt relativ gut behandeln. Funktionalitäten können meist mittels anderer Technologien nachgebaut werden. Eine dieser Technologien sind die Android Application Records. Ein Application Record beschreibt die zu öffnende Applikation durch die Angabe des Namespaces. Haupteinsatzgebiet ist das Binden einer NdefMessage an die eigene Applikation.

```
1 NdefMessage msg = new NdefMessage(
2   new NdefRecord[] {
3       ...,
4       NdefRecord.createApplicationRecord("com.example.android.beam")}
```

Listing 4.22: Generierung eines Android Application Records

Ähnliche Funktionalität liefert die Methode *obtainMessageBoundToThisApp()*. Sie gibt eine *NfcMessage* zurück, die automatisch an die eigene Applikation gebunden ist. Dabei liest sie den Namespace der Applikation aus, die SimpleNFC als Bibliothek einbindet, und erstellt daraus einen Application-MIME-Type. Dieser MIME-Type wird dem NdefRecord als Inhalt übergeben. Die Verbindung aus Application-MIME-Type und dem Intent-Filter des *NfcConnector*s lässt Android beim Empfangen der gerade erstellten *NfcMessage* die Applikation startet, welche SimpleNFC eingebunden hat.

Auf den zweiten Aspekt kann man als Drittentwickler in der Regel keinen Einfluss nehmen. Sollen allerdings mehrere Android-Versionen in einer Applikation unterstützt werden, muss der Entwickler selbstständig herausfinden, welche Methoden er unter welcher Betriebssystemversion verwenden kann. Für diese Überprüfung ist Recherche und Aufwand notwendig, die dem Entwickler durch SimpleNFC abgenommen wird. Mit Hilfe von SimpleNFC ist es möglich, *NfcMessage*s durch zwei verschiedene Methoden auf ein anderes Smartphone zu übertragen: *beamToDevice()* und *pushToDevice()*. Da *pushToDevice()* sich Schnittstellen bedient, die ab Android 4.0 als veraltet gelten und durch die Beam-Technologie ersetzt wurden ist es sinnvoll zu überprüfen ob das Smartphone die Beam-Technologie bereits unterstützt. SimepleNFC bietet diese Überprüfung durch die Methode *supportsBeam()* der Klasse *Nfc*an. Sie liefert als Ergebnis true, falls das Smartphone die Beam-Technologie unterstützt, oder false, falls Beam nicht unterstützt wird. Auf diese Weise kann komfortabel zwischen verschiedenen Android-Versionen unterschieden werden, das Programmieren von kompatiblen Applikationen gestaltet sich weniger komplex.

```
1 public class Nfc {
2   private static final int VERSION_ICECREAMSANDWICH = 14;
3   private static final int VERSION_GINGERBREAD = 10;
4
5   private Activity mActivity;
6   private NfcAdapter mAdapter;
7
```

```
8     ...
9
10    public boolean isAvailable(){
11      if(mAdapter == null) return false;
12      else return true;
13    }
14
15    public NfcMessage obtainMessageBoundToThisApp(String messageId){
16      NfcMessage message = new NfcMessage();
17      message.addRecord(messageId, NdefRecord.TNF_MIME_MEDIA, ("
            application/"+mActivity.getPackageName()).getBytes(), new byte
            [] {});
18      return message;
19    }
20
21    public boolean supportsBeam(){
22      if(Build.VERSION.SDK_INT >= VERSION_ICECREAMSANDWICH) return true;
23      else return false;
24    }
25
26    ...
27
28  }
```

Listing 4.23: Auszug der Klasse *Nfc* mit Methoden, die die Kompatibilität mit älteren Versionen erhöhen

4.6 Evaluation von SimpleNFC: Bewertung durch einen Entwickler und Vergleiche

Die Entscheidung über den Einsatz einer Bibliothek ist stets abhängig von deren Eigenschaften. Neben den eindeutigen positiven Eigenschaften gibt es Aspekte, die einen Einsatz verhindern können. Frameworks können die Größe der Applikation erheblich beinflussen. Dabei bläht nicht nur der eigentliche Quellcode die endgültige App-Größe auf, auch Abhängigkeiten zu weiteren Frameworks benötigen Speicherplatz. Besitzt das gewünschte Framework selbst weitere Abhängigkeiten, kann es außerdem zu Überschneidungen zwischen diesen und den Abhängigkeiten der von der App direkt eingebundenen Frameworks kommen.

SimpleNFC verzichtet komplett auf die Einbindung weiterer Bibliotheken. Sämtliche Aufgaben wurden mit Hilfe nativer Schnittstellen unter Android umgesetzt. Auch die Größe von SimpleNFC ist vernachlässigbar. 27 Kilobyte beträgt die Größe der kompilierten jar-Datei, welche in die Zielapplikation eingebunden werden muss. Die geringe Größe resultiert aus der fehlenden Notwendigkeit von Ressourcen wie Bildern und Texten. Ein weiteres wichtiges Kriterium stellt die Performance dar. SimpleNFC kann als abstrakte Schicht über der nativen NFC-Schnittstelle von Android gesehen werden. Charakteristisch für eine solche Schicht ist neben dem Vorteil der abstrakteren Programmierung ein gesteigerter logistischer Mehraufwand. Daten müssen vorgehalten oder gar serialisiert werden und die Anzahl an Methodenaufrufen steigt an. Die meisten dieser Aspekte können vernachlässigt werden, da

sie eine Notwendigkeit der erweiterten Funktionalität darstellen. Es gibt keine vergleichbaren Funktionalitäten von Android, die man messen und mit SimpleNFC in eine Relation setzen könnte. Lediglich bei der Kapselung von bestehenden Funktionalitäten kann man SimpleNFC mit der nativen Schnittstelle vergleichen. Die Kapselung von NdefMessage bzw. NdefRecord in *NfcMessage* bzw. *NfcRecord* ermöglicht eine einfachere Erstellung von NFC-Nachrichten. Dabei spart der Entwickler Zeit und Codezeilen, die eigentliche Erstellung der NFC-Nachricht zur Laufzeit jedoch wird verlängert. Um diesen vermeintlichen Nachteil besser einschätzen zu können, wurde eine Messung vorgenommen, die das Erstellen von *NfcMessage*s mit der Erstellung von *NdefMessage*s vergleicht. Listing 4.24 und Listing 4.25 zeigen die Zeilen deren Ausführungszeit in 400 Messungen bestimmt wurden. Als Ergebnis ergab sich für das Erstellen einer *NfcMessage* durch SimpleNFC eine durchschnittliche Ausführungszeit von 743s. Demgegenüber stehen durchschnittlich 398s für das Erstellen einer NdefMessage. Die Differenz beträgt 345s, was umgerechnet 0,345ms sind. Testgerät war ein Samsung Nexus S, auf welchem sämtliche Hintergrundprozesse deaktiviert wurden. Aufgrund der geringen Differenz kann man von einer nicht spürbaren Verzögerung sprechen.

```
1 NfcMessage msgNfc = new NfcMessage();
2 msgNfc.addRecord("examplerecord", "This is an Example");
```

Listing 4.24: Erstellung einer *NfcMessage* durch SimpleNFC

```
1 NdefRecord recordNdef = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
2           NdefRecord.RTD_TEXT, "examplerecord".getBytes(),
3           "This is an Example".getBytes());
4 NdefMessage msgNdef = new NdefMessage(new NdefRecord[] { recordNdef });
```

Listing 4.25: Erstellung einer *NdefMessage*

Weiterhin würde es sich anbieten die Zeit für das Beschreiben eines Tags mit und ohne SimpleNFC zu messen. Jedoch beinhaltet dieses Szenario durch das Heranführen des Smartphones an den Tag die Interaktion eines Menschen. Dadurch können keine identischen Vorraussetzungen geschaffen werden, die zuverlässige Messungen gewährleisten würden.

Für eine weitere Evaluation wurde das Framework in einer frühen Entstehungsphase an den Entwickler Julian Faupel weitergegeben. Im Zuge eines Projektes an der Technischen Hochschule Mittelhessen verwendete er SimpleNFC um eine Applikation um NFC-Funktionalität zu erweitern. Als besonders komfortabel empfand er das Binden einer NFC-Nachricht an die eigene Applikation. Dazu Julian Faupel:

> „Besonders praktisch ist auch, dass durch das Framework automatisch die Signatur der eigenen App auf den Tag geschrieben wird und somit die App automatisch geöffnet wird. Ohne das Framework hätte man sich per Hand um diese Funktionalität kümmern müssen."

Grundsätzlich beschreibt er SimpleNFC als „sehr komfortabel" und die „[...] Implemen-

tierung in die eigene App [funktionierte] sehr gut und [er] konnte schnell mit der NFC Funktionalität starten“. Weiterhin lobte er die Einführung der *NfcMessage* und das einfache Beschreiben von Tags.

> „Auch der Schreibvorgang ging sehr einfach, mittels der Übergabe einer NFC-Message an die WriteNFC Klasse.“

Verbesserungspotential sah Julian in der Art der Einbindung des Frameworks. In einer frühen Version von SimpleNFC wurde das Framework als jar-Datei zur Verfügung gestellt. Da dies zu Problemen bei der Einbindung in die Zielapplikation führte, wurden zukünftige Versionen als Android-Library-Projekt angeboten. Auch der *NfcWriteAdapter* entstand aus Julians's Intention heraus.

> „Zum Abfangen der Exceptions könnte man eine Standard-Implementierung des Interfaces NFCWriteListener zu dem Projekt hinzufügen. Vielleicht so ähnlich wie bei dem Beispiel in der Anleitung.“

Schlussendlich fiel sein Feedback durchweg positiv aus und er ist froh „[...] sich nicht mehr um die Details mit Byte-Array etc. kümmern [...]“ zu müssen.

Kapitel 5

Potenzielle Nutzer von NFC: Kenntnisse und Bedürfnisse

5.1 Feldumfrage zum aktuellen Kenntnisstand

In der Literatur sind keine aktuellen Zahlen zu finden, die angeben, wie weit verbreitet NFC ist. Vor allem gibt es keine Statistiken über das Wissen der Endanwender. Bevor in den folgenden Abschnitten untersucht wird, mit welchen Mitteln NFC dem Nutzer besser vermittelt werden kann, wurde eine Feldumfrage im Rahmen dieser Arbeit erstellt. Ziel der Umfrage war, einen Überblick darüber zu bekommen, ob die Anwender bereits NFC kennen und in welchem Umfang sie sich unterschiedliche Nutzungsszenarien vorstellen können, die damit zusammenhängen. Es wurde Wert darauf gelegt, möglichst unterschiedliche Personen zu befragen, um einen Querschnitt der Bevölkerung zu erzeugen.

Die Umfrage wurde mit dem Online-Tool von Google Docs[1] erstellt und ausschließlich über das Internet verteilt und beantwortet. Diese Einschränkung ist für die spätere Auswertung wichtig, da damit der Anspruch nicht gehalten werden konnte, einen Schnitt durch alle Bevölkerungsschichten zu machen. Auch hat sich die Umfrage größtenteils über Foren und Facebook verteilt. Es handelt sich demzufolge um Nutzer, die technisch nicht ganz unerfahren sind. Obwohl für die Befragung zunächst keine Zielgruppe festgelegt wurde, hat sich durch die Umstände eine Zielgruppe ergeben, die auch für das Ergebnis als sinnvoll zu erachten ist. So wird beispielsweise eine 80-jährige Frau, die heute kein Internet nutzt und kein Handy besitzt, auch in Zukunft kein Smartphone kaufen, mit dem sie bargeldlos im Supermarkt bezahlen kann.

Insgesamt wurden 342 Personen befragt. Das Verhältnis zwischen Männern und Frauen lag bei genau 50%. Das Thema der Umfrage wurde den Teilnehmern erst mit der ersten Frage mitgeteilt, so dass im Vorhinein keiner durch ein technisches, beziehungsweise ihm unbekanntes Thema, abgeschreckt wurde.

[1] `http://docs.google.com`

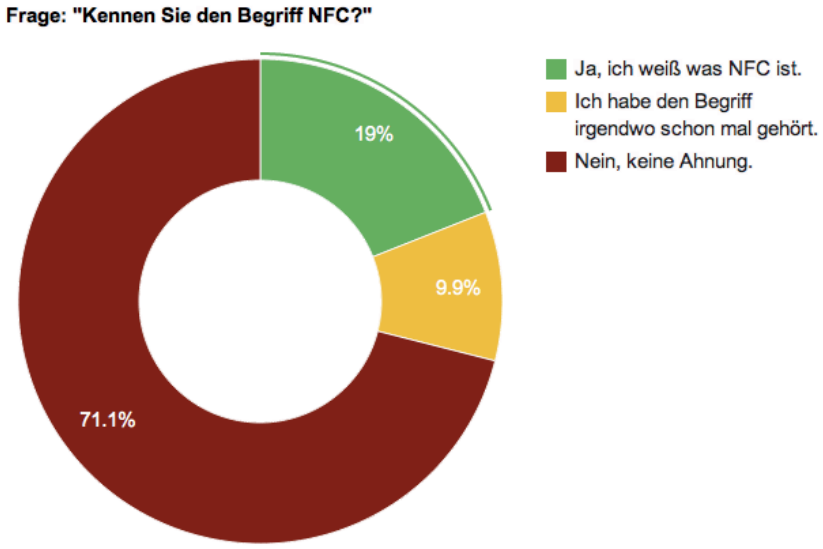

Abbildung 5.1: Umfrage: Kennen Sie den Begriff NFC?

5.1.1 Allgemeine Kenntnis von NFC

Die ersten Fragen zielten auf die allgemeine Kenntnis von NFC ab, ohne vorherige Einführung oder Erklärung. Auf die erste Frage, ob der Begriff „NFC“bekannt ist, antworteten nur 19% mit „Ja, ich weiß was NFC ist“. Die große Mehrheit konnte mit dem Begriff nichts anfangen. 10% haben den Begriff zumindest schon mal wahrgenommen.

Immerhin kennt somit jeder fünfte der Befragten den Begriff NFC, was zunächst als viel erscheint. Bricht man allerdings diese Zahl auf Personen runter, die normale Computerkenntnisse besitzen (im Fragebogen angegeben mit Kenntnissen über eMails, Internet und Textverarbeitung) kommt man nur noch auf 1,8%, denen die Abkürzung NFC bekannt ist. Daraus lässt sich schließen, dass zumindest unter Computerprofis und Nutzern mit sehr guten Kenntnissen einige mit dem Begriff NFC vertraut sind.

Wenn der Nutzer bei der ersten Frage „Kennen Sie den Begriff NFC?“mit „Ich habe den Begriff irgendwo schon mal gehört.“geantwortet hat, wurde er als nächstes gefragt, ob ihm der Begriff „Near Field Communication“(also nicht das Akronym) mehr sagen würde. 50% der Befragten konnten mit dem ausgeschriebenen Begriff dann etwas anfangen, den anderen 50% hat dieser Hinweis nicht weiter geholfen. Man könnte die These daraus ableiten, dass der ausgeschriebene Begriff Nahfeldkommunikation bzw. Near Field Communication zu einem gewissen Anteil selbst beschreibend ist und dem Verständnis des Nutzers hilft. Deswegen sollte auch in der Technologievermittlung nicht nur die Abkürzung NFC benutzt werden. Dies ist allerdings nur eine Vermutung, die bestätigt werden müsste und durch die vorliegende Umfrage nicht abschließend geklärt werden kann.

Alle Befragten, die angaben, dass ihnen der Begriff NFC bzw. Near Field Communication etwas sagt, wurde gebeten, ihre Vorstellungen von NFC kurz zu beschreiben. Interessanterweise beinhalteten 51% der Antworten einen Bezahlvorgang als Beispiel von NFC. Dies

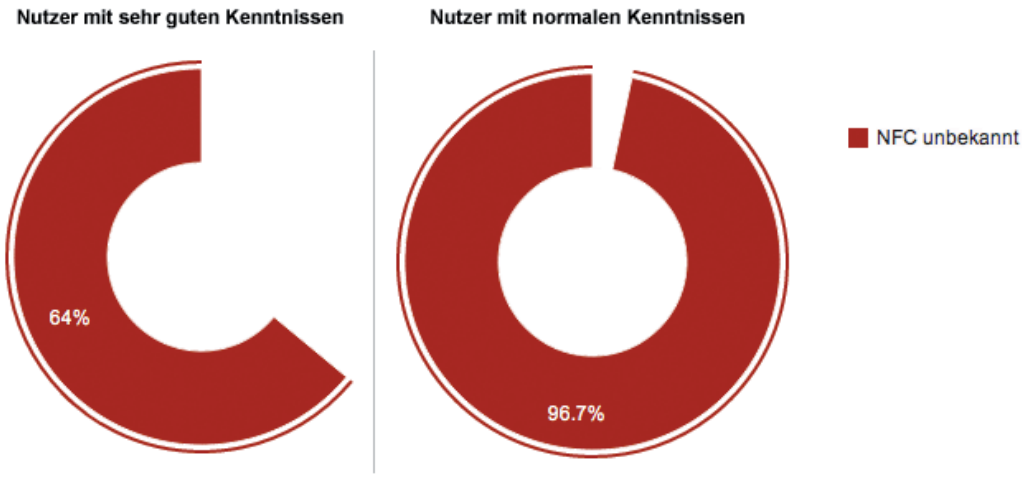

Abbildung 5.2: Kenntnis von NFC auf Nutzergruppen aufgeteilt

bestätigt das Empfinden, dass vor allem Banken und Zahlungsdienstleister auf eine rasche Einführung von NFC hoffen und dort ein großes Geschäftsfeld sehen. Zudem haben sie es bereits geschafft, die Nutzer auf mögliche Bezahlszenarien zu sensibilisieren. Eine Auswahl repräsentativer Antworten:

> „Ermöglicht berührungsloses Bezahlen in Geschäften."
>
> „Bezahlen über Funk [...]"
>
> „Ich halte mein Handy im Vorbeigehen an einen Empfänger und tätige damit meine Zahlung."
>
> „Mit dem Handy im Alltag zu bezahlen statt Bargeld zu verwenden"
>
> „z.b. bezahlen per Handy an der Kasse"
>
> „Chipkarte oder Handy zum Bezahlen verwenden"
>
> „[...] So kann beispielsweise mit dem Mobiltelefon bezahlen kann."
>
> „Einfache und schnelle Zahlungsmöglichkeit mit dem Handy bis zu einem gewissen Betrag"
>
> „Damit ist es in Zukunft möglich mit dem Smartphone bargeldlos im Einkaufszentrum zu bezahlen. Dies ist sicherer als andere Übertragungsarten, da es nur über sehr kurze Entfernungen möglich ist."

Alle Teilnehmer, die angaben, dass sie NFC kennen, wurden zusätzlich gefragt, ob sie NFC bereits im Alltag anwenden. Nur 14% der NFC-Kenner haben NFC bereits im Alltag eingesetzt. Drei Teilnehmer nutzen NFC für öffentliche Verkehrsmittel (RMV, DB

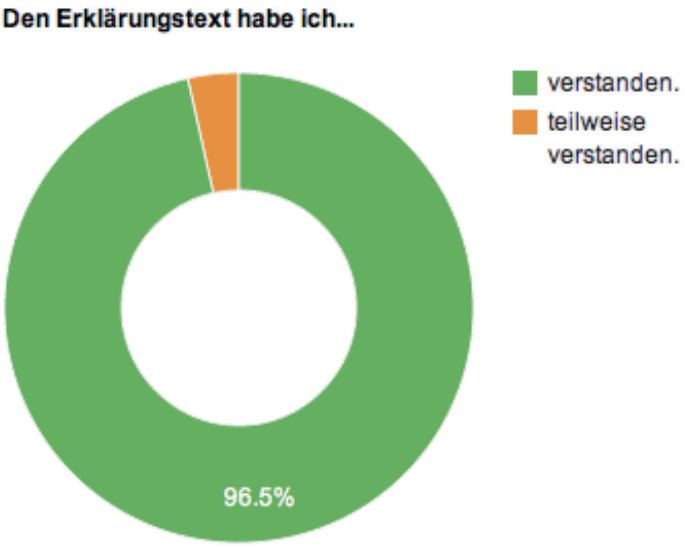

Abbildung 5.3: Verständnis des Erklärungstextes

Touch&Travel und Oyster Card[2]). Weitere drei Teilnehmer nutzen Android-Funktionen, um Daten zwischen Smartphones auszutauschen (Android Beam[3]).

5.1.2 Verständnis von NFC

Allen Teilnehmern des Fragebogens wurde im nächsten Schritt ein kurzer Erklärungstext angezeigt:

> „Near Field Communication oder kurz NFC ist ein Übertragungsstandard (ähnlich wie Bluetooth) zum kontaktlosen Übertragen von Daten (Informationen) zwischen zwei Geräten. In unserem Fall ist eines der Geräte ein Handy oder Smartphone. Ein Vorteil gegenüber ähnlichen Technologien ist die schnelle und einfache Nutzung. Beispiel: Man legt sein Smartphone auf eine speziell gekennzeichnete Fläche an einer Kasse und kann damit bezahlen."

Eine Quote von 97% der Befragten, die den Text verstanden haben, lässt darauf schließen, dass es möglich ist, potenziellen Nutzern in wenigen Sätzen zu erklären, worum es sich bei NFC handelt. Die Erklärung reicht allerdings nicht dazu aus den Nutzer so weit an NFC heranzuführen, dass er für sich selbst eigene Nutzungsszenarien erkennen kann. Dazu fehlt ihm das Wissen über die technischen Möglichkeiten und Grenzen.

[2] Kartensystem der Londoner Personennahverkehrs
[3] `http://www.android.com/about/`, abgerufen am 28.06.2012

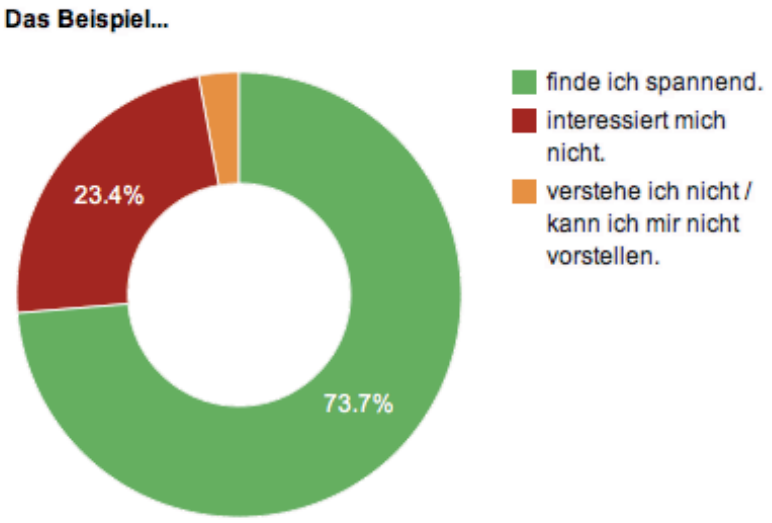

Abbildung 5.4: Verteilung Beispiel Bezahlung

5.1.3 Akzeptanz von unterschiedlichen Nutzungsszenarien

In dem Erklärungstext wurde außerdem ein erstes Beispiel beschrieben. Dabei wurde ohne weitere Details nur erwähnt, dass man an einer Kasse (egal in welchem Umfeld) mit seinem Smartphone zahlen kann, indem man sein Smartphone auf eine gekennzeichnete Fläche legt. Dreiviertel der Nutzer (73%) finden dieses Szenario spannend. Dies kann als eine weitere Bestätigung dafür angesehen werden, dass vor allem das Zahlungsszenario ein guter Treiber für die Verbreitung von NFC werden kann.

Das nächste Szenario beschreibt ein einfaches Zutrittsszenario. Man kauft im Vorhinein ein Ticket für eine Veranstaltung und die Zugangsinformationen werden auf dem Smartphone gespeichert. Mit dem Smartphone kann man sich nun Zugang verschaffen ohne zusätzliche Kontrollen, so wie in Abbildung 5.5. Der Originaltext aus dem Fragebogen:

> „Man kauft online ein Ticket für ein Konzert oder eine Veranstaltung. Das Ticket wird automatisch auf dem Handy gespeichert. Kommt man am Veranstaltungsort an, muss man nur sein Handy über ein Lesegerät halten und kann die Schranken passieren."

Die Antworten verteilen sich ähnlich wie im ersten Beispiel (Abbildung 5.6). Auf Grund der anders gestalteten Antworten sind die Befragten etwas verhaltener. Dennoch liegt die Ablehnung dieses Szenarios bei nur 24%. Immerhin 43% würden gerne diese Funktionen nutzen.

Interessant ist, dass das dritte Beispiel von 70% der Befragten abgelehnt beziehungsweise als zu unsicher eingestuft wird (Abbildung 5.7). Das Szenario wurde im Fragenbogen folgendermaßen beschrieben:

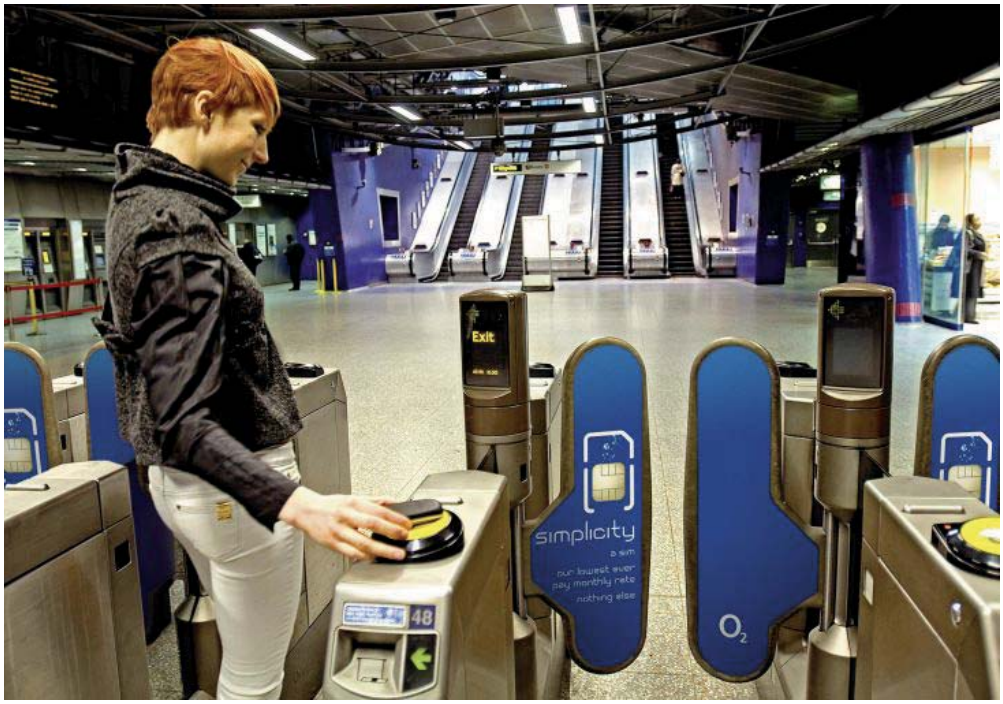

Abbildung 5.5: Schranken mit NFC-Zugang, Quelle: `http://goo.gl/wnFQh`, abgerufen am 28.06.2012

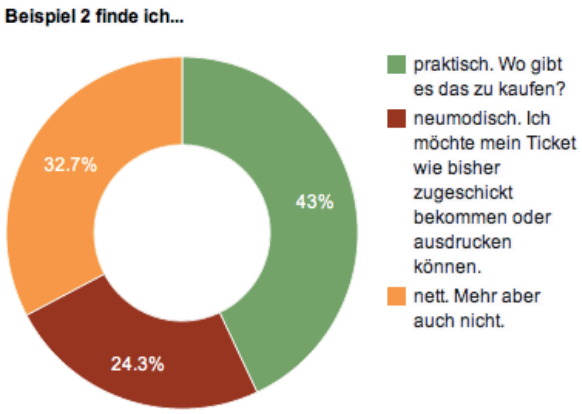

Abbildung 5.6: Verteilung Beispiel Zutrittskontrolle

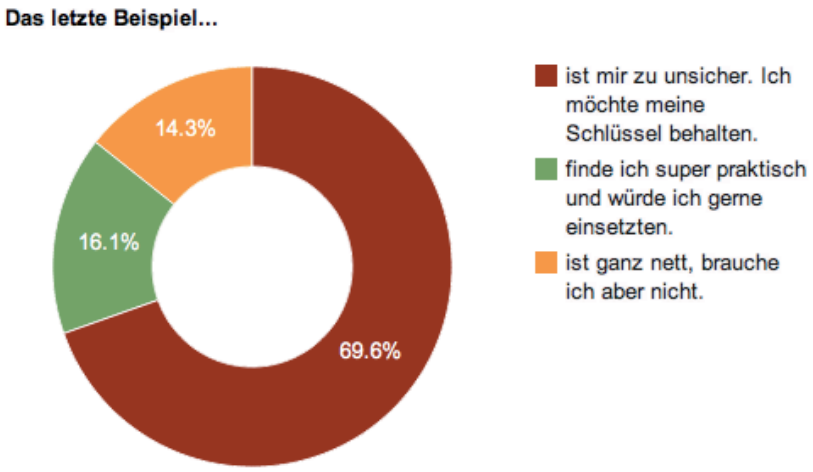

Abbildung 5.7: Verteilung Beispiel Smartphone als Schlüssel

> „Das Handy als Schlüssel: Auf dem Mobiltelefon sind Informationen über Zutrittsrechte gespeichert, d.h. man kann sein Auto, seine Wohnung, sein Büro mit dem Handy öffnen, einfach in dem man das Handy in die Nähe eines Lesegerätes hält."

Nur 16% können sich einen Einsatz vorstellen und würden gerne in den Gebrauch dieser Funktionen kommen. Gliedert man diese Zahlen nach Befragten, die zu Beginn angaben NFC zu kennen (vgl. Abbildung 5.8), fällt auf, dass diese Nutzer sicherheitsrelevanten Szenarien offener gegenüberstehen. Daraus kann man ableiten, dass Nutzer, die sich schon mit der Thematik auseinandergesetzt haben, die Sicherheit von NFC besser einschätzen können. In der Erklärung von NFC wurde zudem kein Bezug auf sicherheitsrelevante Aspekte genommen. Möchte man Nutzer in diesem Bereich gewinnen, muss dementsprechend dafür gesorgt werden, dass den Nutzern die Angst vor der Technologie genommen wird.

5.1.4 Interesse an NFC

Der letzte Abschnitt der Umfrage zielte auf die Bereitschaft beziehungsweise das Interesse der Nutzer ab, NFC zukünftig selbst zu nutzen. Erstaunlicherweise geben 40% der Befragten an, dass sie bereit wären für die Funktionalität von NFC einmalig Geld zu bezahlen (durch einen höheren Anschaffungspreis des jeweiligen Smartphones). Zwar handelt es sich bei 48% dieser Befragten bei dem genannten Betrag nur um 10 €, immerhin 10% sind dennoch bereit 50 € zusätzlich zu zahlen (Abbildung 5.9).Wenn man als Nutzer bereit ist, einen Betrag in dieser Höhe auszugeben, muss man klare Vorstellungen im Kopf haben, welche Vorteile einem NFC in der späteren Nutzung bietet.

Das hohe Interesse der Nutzer an NFC (vgl. Abbildung 5.10) ist eine positive Grundlage für die weitere Verbreitung von NFC. Nur 13% zeigen kein Interesse an NFC. Erneut sind diejenigen, die bereits NFC kannten, interessierter an der Technologie. So haben 93% der

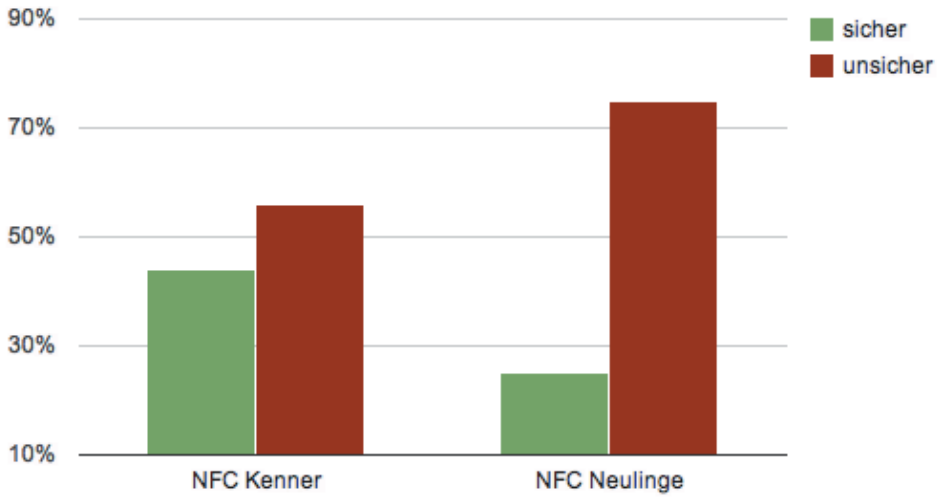

Abbildung 5.8: Prozentverteilung Kenner/Nicht-Kenner auf Sicher/Unsicher

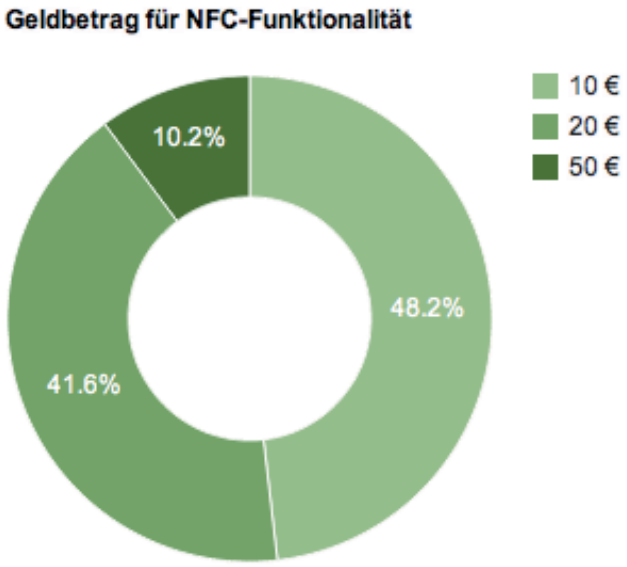

Abbildung 5.9: Geldbetrag für NFC-Funktionalität

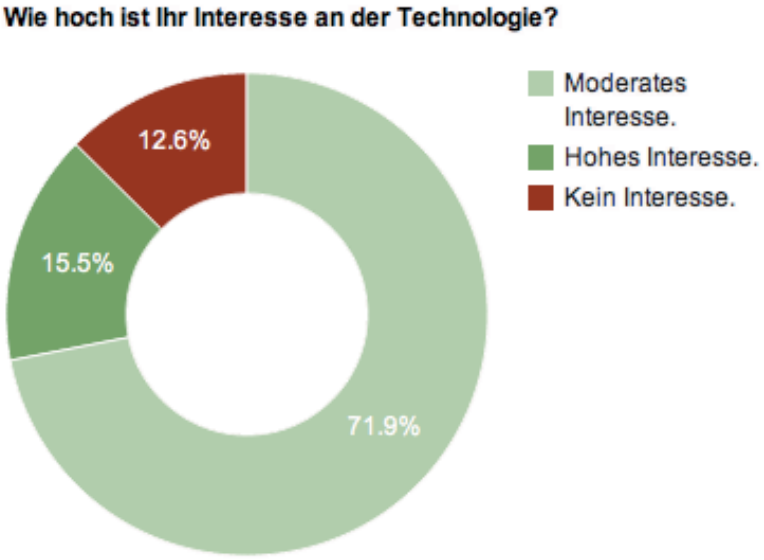

Abbildung 5.10: Interesse an NFC

Kenner Interesse, bei Nicht-Kennern liegt der Prozentsatz bei 85%. Demnach scheint auch hier das Verhältnis zu stimmen, je mehr man bereits mit NFC zu tun hatte, desto mehr kann man sich NFC in unterschiedlichen Bereichen vorstellen und desto höher ist das daraus resultierende Interesse.

Obwohl nach der Befragung 97% angaben, dass sie NFC verstanden haben und wissen, wofür sie es einsetzen können, haben nur 28% konkrete Vorstellungen, wofür sie persönlich NFC nutzen wollen. Dafür gab es bei den Vorstellungen viele unterschiedliche Szenarien, eine Auswahl der interessantesten und relevantesten Antworten werden im Folgenden aufgelistet.

„Auto starten, private und geschäftliche Schleusen/Türen öffnen“

„Mautstraßen. beim Vorbeifahren an der Mautstelle automatisch bezahlen , ohne aufwändiges und stauförderndes Stehenbleiben“

„Automatisierung der Lichter/Geräte im Haus“

„Kleingeldersatz für Einkäufe mit geringem Wert, Parkuhr, Parkhaus, etc“

„Bordkarten“

„Dauerkartenbesitzer für Fussballstadion“

„Für den Datenaustausch von Handy zu PC, wobei man die Geräte auf einen bestimmten Tisch legt und die Daten einfach transferiert.“

„‚Pairing ‘vom Smartphone mit anderen Geräten (z.B. Hifi-Anlage, NAS, ..)“

„Lagerverwaltung“

„Zeiterfassung“

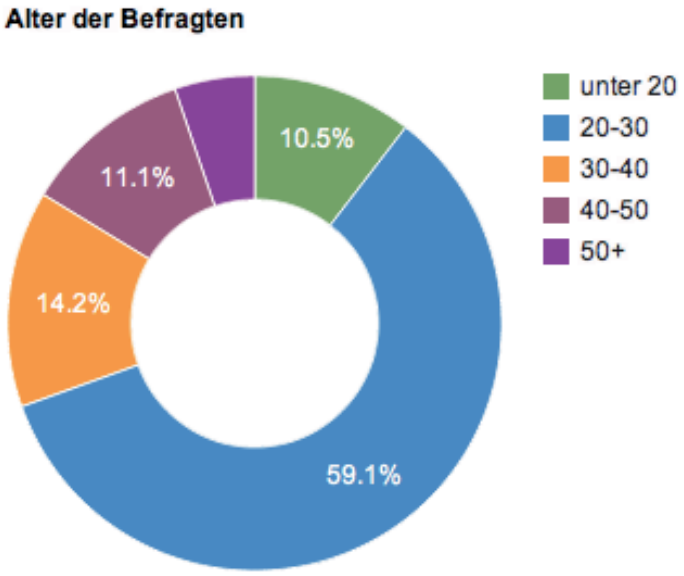

Abbildung 5.11: Altersverteilung

„Nähere Wareninformationen im Einzelhandel erhalten."

„Fernseher, Backofen, Radio an-/ausmachen"

Zur Vollständigkeit ist die Altersverteilung in Abbildung 5.11 zu sehen. Die Befragten gehören den unterschiedlichsten Altersgruppen an. Grundsätzlich gab es zwischen den Altersgruppen keine signifikanten Unterschiede in der Auswertung. In den drei wichtigen Standpunkten Sicherheitsbedenken (Schlüsseleinsatz als zu unsicher eingestuft), Zahlungsbereitschaft (Bereitwilligkeit für NFC-Funktionalität zusätzlich zu zahlen) und generellem Interesse an NFC verteilen sich alle Gruppen ähnlich. Die geringeren Sicherheitsbedenken Altersgruppe 40 bis 50 treten aus der Reihe, ebenso die hohe Zahlungsbereitschaft der unter 20-Jährigen. Beide Auffälligkeiten können zunächst nicht logisch begründet werden und müssten in einer weiteren Studie untersucht werden. (siehe Abbildung 5.12)

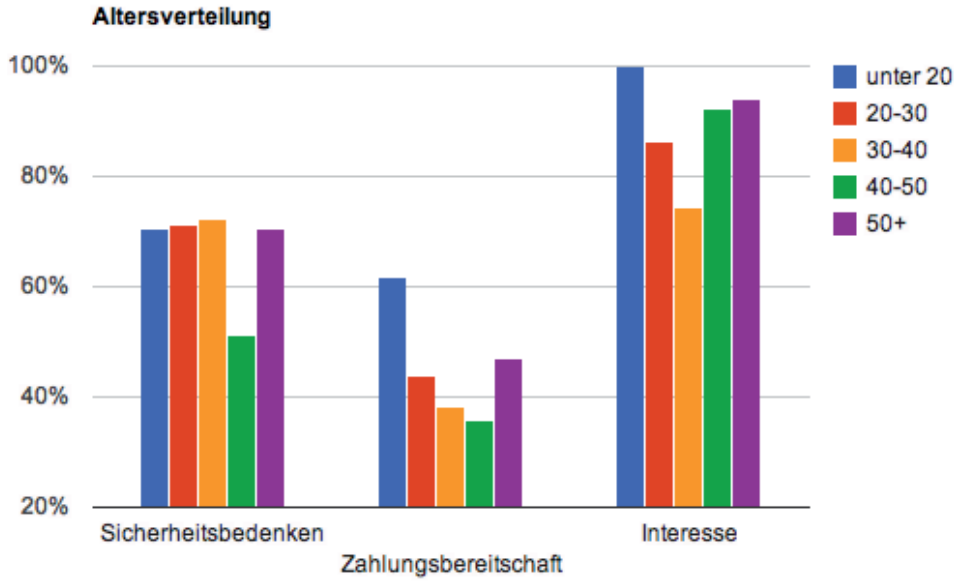

Abbildung 5.12: Standpunkte nach Altersgruppe

5.2 Evaluation der Anwendungsbeispiele

Nachdem in der Umfrage das generelle Verständnis von NFC untersucht wurde, sollen zwei Prototypen von neuartigen NFC-Anwendungen dazu genutzt werden, zu erforschen, welche Faktoren für den Erfolg von NFC wichtig sind. Dabei geht es um den praktischen Einsatz von NFC, bei welchem folgende Fragen im Fokus stehen sollen:

- Können die Nutzer mit der Technologie schnell umgehen?
- Haben sie gegebenenfalls Ängste?
- Nutzen sie lieber eine altbekannte Technik, um das gleiche Ziel zu erreichen?
- Besteht tatsächlich ein Vorteil gegenüber vergleichbaren Techniken, wie QR-Codes oder Bluetooth?
- Wo genau liegen diese Vorteile und werden diese ausgenutzt?

Zudem soll erforscht werden, ob Nutzer Interesse an solchen Anwendungsszenarien haben und diese tatsächlich in ihrem Alltag nutzen wollen.

5.2.1 „Leinwand“: Interaktion mit öffentlichen Projektionsflächen

Grundlage des ersten Prototypen ist die Bedienung einer entfernten, öffentlichen Leinwand mit dem eigenen Smartphone. Dabei spielt es keine Rolle, welche Größe die Leinwand hat oder ob es sich dabei um einen Fernseher oder eine große Projektionsfläche handelt. Wichtig ist, dass der Nutzer normalerweise keinen Zugang zu dieser Präsentationsfläche hätte, ein Bildschirm mit Tastatur und Maus als Eingabemedien (wie beispielsweise bei Terminals) ist demzufolge uninteressant für diese Untersuchung. In der Realität könnten diese Leinwände viele unterschiedliche Funktionen und Nutzerszenarien abdecken. So könnten beispielsweise Kaufhäuser nach den Ladenöffnungszeiten ihre Schaufenster als interaktive Leinwand nutzen, um Produkte anzupreisen. Anzeigetafeln in öffentlichen Gebäuden (zum Beispiel im Umfeld einer Hochschule) könnten über einen solchen Zugang einfach verändert werden.

Der genutzte Prototyp simuliert ein Szenario bei einem Autohändler. Der Kunde sieht auf einer großen Leinwand einen Wagen und kann diesen nach seinen Wünschen konfigurieren. Dazu muss er sein NFC-Smartphone über eine gekennzeichnete Stelle halten und verbindet sich so automatisch mit der Leinwand, beziehungsweise dem zuständigen Computer. In einem Nutzertest wurde untersucht, wie Nutzer auf dieses Szenario reagieren und ob sie verstehen, wie sie sich mit der Leinwand verbinden können. Der Testaufbau ist in Abbildung 5.13 zu sehen.

Die Nutzer wurden zu Beginn des Tests gefragt, ob sie NFC beziehungsweise Near-Field-Communication kennen. Wenn diese Frage verneint wurde, erhielten sie eine kurze Einführung in das Thema und die Funktionen (speziell in Verbindung mit einem Smartphone). Auf der Leinwand ist zum Startzeitpunkt ein rotes Volkswagen Golf Cabrio zu sehen. Die Tester erhielten nun die Aufgabenstellung: „Ändere die Farbe des Cabrios auf der Leinwand in weiß„. Neben objektiv messbaren Parametern, wie die Zeit zum Erreichen des Ziels,

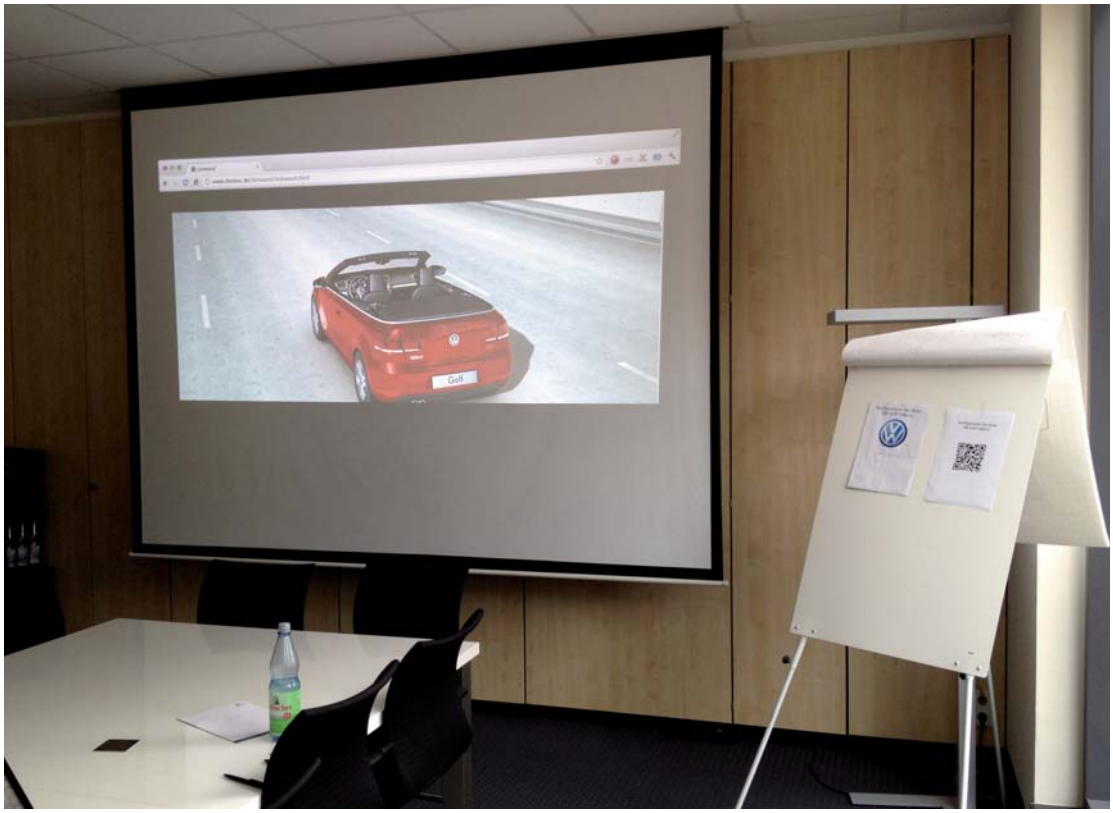

Abbildung 5.13: Aufbau des Leinwand-Prototyps

wurde vor allem auf die Reaktionen und Probleme der Nutzer geachtet, die sich schwer in Zahlen ausdrücken lassen. Nicht desto trotz gab es klare Tendenzen, die in den folgenden Abschnitten erläutert werden sollen. Zur Vollständigkeit ist in Abbildung 5.14 die Oberfläche zum Ändern der Farbe abgebildet, die aufgerufen wird, sobald der Nutzer die Anwendung aktiviert.

In der ersten Testreihe hingen an dem Aufsteller zwei unterschiedliche Abbildungen (vgl. Abbildung 5.15). Die eine war mit einem VW-Logo versehen und der Aufschrift „Einfach Handy an das Logo halten“. Die andere Seite war mit einem QR-Code bedruckt, mit dem der Tester die Applikation starten konnte, genauso wie durch das Berühren des Logos auf der ersten Seite.

Da es sich bei diesen Tests nicht um eine Meinungsumfrage, sondern um eine Art Usability-Studie handelt, reichen laut Nielsen wenige Probanden aus, um die meisten Probleme mit der Anwendung zu erkennen [34]. Bereits fünf Probanden finden demnach 80% der Probleme, die auftreten.

Trotz der Kenntnis über NFC haben acht von neun Probanden nach der Aufgabenstellung eine Barcode-Scanner-App auf dem zur Verfügung gestellten NFC-Smartphone gesucht, um damit den QR-Code zu scannen. Im Gespräch danach wurde gefragt, warum der QR-Code gescannt wurde, obwohl erst eine Minute vorher eine Einführung in NFC gegeben

Abbildung 5.14: Oberfläche der Steuerungs-App

wurde. Der Großteil gab an, dass ihm die Technik bekannt vorkomme und man so wohl am schnellsten zum Ergebnis kommen würde. Interessant war auch, dass vier der befragten Personen bemerkten, dass ihnen das Papier mit dem Logo nicht so vorkomme, als könne man damit tatsächlich technisch interagieren. Viele drehten ungläubig das Papier um und suchten vergeblich nach Kabeln oder anderen technischen Teilen. Um dieses Problem zu umgehen, kamen die Vorschläge den Kontaktpunkt dreidimensional zu gestalten. Gegebenenfalls würde auch eine andere Oberfläche als Papier dazu beitragen, dass der Nutzer diesen als technischen Kontaktpunkt erkennt.

Da es nur wenige Probanden in der ersten Testreihe geschafft haben, NFC zu nutzen, um an das Ziel der Aufgabenstellung zu kommen, wurde für die zweite Testreihe ein neues Logo für die NFC-Schnittstelle genutzt. Die nun genutzte Seite ist in Abbildung 5.16 zu sehen. Statt dem VW-Logo wurde jetzt ein Piktogramm genutzt, bei dem ein Smartphone und eine Fläche mit Funkstrahlen zu erkennen sind.

Mit dieser Änderung kamen zumindest vier von fünf Probanden der zweiten Runde direkt auf die Idee, das Smartphone an das Papier zu halten. Allerdings hat es nur bei zwei

Abbildung 5.15: Auswahl der ersten Testreihe

Abbildung 5.16: NFC-Piktogramm, genutzt in der zweiten Testreihe

Abbildung 5.17: RSS-Feed-Icon und N-Mark, Quellen: `http://goo.gl/mJ9Yn` und `http://goo.gl/6q2iA`, abgerufen am 28.06.2012

Probanden reibungslos funktioniert. Die beiden anderen haben das Smartphone entweder zu weit von dem Papier entfernt gehalten oder waren mit der Rückseite vom Smartphone nicht auf der richtigen Höhe des NFC-Tags. Es ist davon auszugehen, dass diese Verhaltensweise nur beim ersten Kontakt mit der Technologie zu beobachten ist.

Nach der Aufgabenstellung wurden die Probanden befragt, welche Ideen sie zu der Gestaltung des Kontaktpunktes haben. Obwohl fast alle ein Piktogramm für hilfreich halten, waren über 90% aller Befragten der Meinung, dass es ein einheitliches Logo für die Anwendung von NFC geben soll. Dabei ist es irrelevant, ob das Logo irgendeine besondere Aussagekraft hat, wichtig ist ein guter Wiedererkennungswert. Fast alle waren sich einig, dass man sich schnell an ein Logo gewöhnt, sobald es regelmäßig eingesetzt wird. Als Beispiel wurde das RSS-Logo[4] genannt - es gab nie eine Standardisierung, trotzdem hat sich über die Jahre ein Logo entwickelt, das zwar mit Abweichungen, international erkannt wird (Abbildung 5.17).

Das NFC Forum, dass sich mit den Normen und Standards von NFC befasst, entwickelte ein Logo für NFC, dass einheitlich genutzt werden soll. Das sogenannte N-Mark (siehe Abbildung 5.17) soll nicht nur zum Marketing eingesetzt werden, sondern auch an den Stellen platziert werden, an denen Geräte ihre NFC-Funktionalität besitzen. In einer Informationsbroschüre[5] werden die Vorteile eines einheitlichen Logos aufgezeigt. Es ist gut möglich, dass sich das Logo in Zukunft als globaler Standard etabliert. In den aktuellen Projekten ist es allerdings noch nicht überall zu sehen.

In Abbildung 5.18 sieht man unterschiedliche Einsatzmöglichkeiten des N-Mark getauften Logos. Oben links ist der aktive Kontaktpunkt einer Kamera abgebildet, an dem mit NFC kommuniziert werden kann. Das zweite Bild in der ersten Reihe zeigt ein Einstellungsmenü für spezielle NFC-Funktionalitäten. Auch das dritte Bild zeigt die Position des Kontaktpunktes, diesmal in einem Laptop integriert. Auf dem ersten Bild der zweiten Reihe ist ein Supermarktregal zu sehen, über NFC können weitere Informationen zu entsprechenden Produkten abgerufen werden, sobald das Smartphone in die Nähe des Logos gehalten

[4] RSS ist ein Internet-Nachrichtenformat, über das Nutzer über Änderungen und Neuerungen einer Internetseite (Nachrichte, Blogs, etc.) informiert.

[5] `http://www.nfc-forum.org/resources/N-Mark/brandguide.pdf`, abgerufen am 28.06.2012

Abbildung 5.18: Unterschiedliche Anwendungsfälle der N-Mark

wird. Mittig sind Medikamente zu sehen, über die man nach Berühren weitere Informationen erhalten kann. Das rechte Bild der zweiten Reihe zeigt ein fiktives Busplakat, über das die aktuellen Abfahrtszeiten abgerufen werden können. In der letzten Reihe ist der Einsatz auf Produkten als Werbung für die NFC-Funktionalität abgebildet. Links wird auf einer Webseite für die NFC-Funktionalität geworben, in der Mitte auf einem Plakat und im letzten Bild als Spezifikation auf einer Produktverpackung.

Aus diesen Tests kann bereits ein wichtiges Ergebnis abgeleitet werden. Die Nutzer brauchen selbst nach einer Einleitung in NFC klare Anweisungen, wie sie mit einem NFC-Tag umzugehen haben. Dabei hilft zum direkten Verständnis ein Piktogramm, für zukünftige Anwendungen wünschen sich die Nutzer ein einheitliches Erscheinungsbild von NFC-Applikationen.

Alle Probanden wurden gebeten, nachdem sie die beiden Möglichkeiten QR-Code und NFC-Tag verstanden und ausprobiert haben, die Aufgabenstellung erneut mit der jeweiligen Technologie zu erledigen. Dabei wurde die Zeit gemessen. Im Durchschnitt war die Bedienung mit NFC um neun Sekunden schneller. Der Durchschnitt beim Erledigen der Aufgabe mit QR-

Code war 20,78 Sekunden, bei NFC gerade einmal 12,4 Sekunden (Abbildung 5.19). Was hierbei außer Acht gelassen wurde, aber die Unterschiede in einem realen Szenario noch deutlicher machen würde, ist, dass bei dem Prototypen keine echte Verbindung mit dem Computer aufgebaut wurde und NFC so nicht alle Vorteile ausspielen konnte. Der Prototyp basiert auf einer Webanwendung, bei der das Smartphone über eine Socket-Verbindung mit dem node.js[6] Server verbunden wurde. Diese Verbindung lief über das Internet. Dadurch konnte auch mit dem Barcode und dem Aufruf einer URL direkt auf die Leinwand zugegriffen werden. In einem realen Fall könnte der Kontaktpunkt mit einem lokalen Computer verknüpft sein und den Pairing-Prozess zwischen Anwendung (Leinwand) und Smartphone automatisch übernehmen. Diese Möglichkeit bietet ein QR-Code nicht. Dadurch könnte eine sichere Verbindung hergestellt werden, so dass nur Personen in direkter Umgebung die Möglichkeit haben, mit der Leinwand zu interagieren.

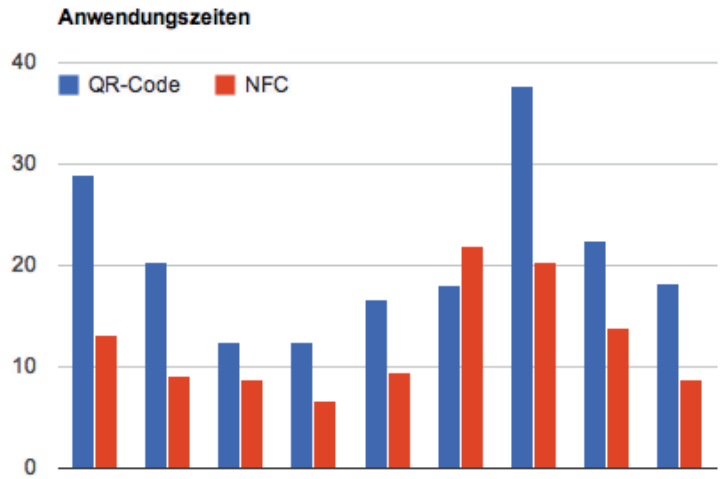

Abbildung 5.19: Ausführungszeiten im Vergleich

Ein weiterer Bestandteil der Untersuchung im Hinblick auf die Fragestellung dieser Arbeit ist es, zu erforschen, welche Szenarien sich die Nutzer vorstellen können und in Zukunft gerne nutzen würden. Der genutzte Prototyp steht für einen allgemeinen Anwendungsfall: eine öffentlich einsehbare Projektionsfläche, deren Inhalte von den Passanten verändert werden können. Nur knapp über die Hälfte der Befragten fand diesen Anwendungsfall sinnvoll und konnte sich den realen Einsatz vorstellen. Auffälligerweise hatten gleich drei der Befragten die gleiche Idee für einen Gebrauch dieser Technik. Sie wünschen sich in Clubs und Diskotheken eine Leinwand mit der aktuellen Playlist. Durch einen Kontaktpunkt (beispielsweise in der Nähe vom DJ) kann man die Playlist verändern und für bestimmte Lieder stimmen, die dann in der Reihenfolge weiter nach oben rutschen. Eine andere Idee war das Zusammenstellen von passender Kleidung in einem Schaufenster, jedoch mit Einschränkungen: „Ich glaube allerdings, dass so was Neumodisches nicht zu allen Läden passt, beziehungsweise dort Sinn

[6] `http://nodejs.org/`

macht. HM wäre ein passender Laden für so was.“. Betrachtet man diese Ergebnisse unter dem Aspekt der Diffusionstheorie, muss man klar feststellen, dass in der Knowledge-Phase keine Probleme existieren, die durch die Innovation mit NFC gelöst werden können. Hier liegt eine der großen Hürden für NFC: Es ist nur in wenigen Fällen in der Lage, wirklich neue Möglichkeiten zu generieren. In der Regel hätte man unter Umständen die Anwendungsfälle auch ohne NFC lösen können (wie im oben gezeigten Beispiel per QR-Code) - durch NFC werden diese Prozesse allerdings schneller, sicherer und einfacher für den Nutzer. Diese Vorteile sind jedoch schwer zu vermitteln, da sie nicht klar erkenntlich sind. Die meisten Nutzer werden also für sich selbst keine realen Einsätze von NFC sehen, bis sie diese nicht selbst ausprobiert haben und aktiv von den Vorteilen überzeugt werden konnten.

Was sind demzufolge die Schlüsse für eine erfolgreiche Verbreitung von NFC? Für ein Szenario, wie dem oben beschriebenen Leinwand-Fall, befinden sich die Entwickler Mitte 2012 noch in einer Art Teufelskreis: Würde man als Firma eine große Kampagne starten, gäbe es zu wenige Nutzer mit kompatiblen Smartphones um diese Kampagne wirtschaftlich erfolgreich zu gestalten. Gibt es keine Anwendungsbeispiele für NFC, gibt es auch kein Verlangen nach NFC-Smartphones aus Richtung der Nutzer. Dann bemühen sich auch die Hersteller nicht, NFC in ihren Smartphones zu integrieren. Glücklicherweise gibt es mit dem Bezahlszenario eine Anwendung, die sowohl die Hersteller als auch die Nutzer dazu treibt, die Technologie zu adoptieren (vgl. Kapitel 2.2). Trotzdem sollten Entwickler in der Konzeption gerade zu Beginn beachten, dass die Nutzer NFC noch nicht besonders gut kennen. So muss der Kontaktpunkt besonders gekennzeichnet werden. Entweder durch eine physische Anhebung oder einem besonderen Material oder durch eindeutige Piktrogramme, die den Nutzer dazu verleiten, sein Smartphone an diesen Kontaktpunkt zu halten. Auch wenn das N-Mark des NFC-Forums noch keine globale Nutzung und Wiedererkennung bei den Nutzern erfahren hat, halten es die Autoren dieser Arbeit für sinnvoll, das Logo an möglichst vielen Stellen zu integrieren.

5.2.2 „HomeField “: NFC-Tags in der eigenen Wohnung

Allgemeine Beschreibung

HomeField soll den Einsatz des Smartphones im täglichen Alltag zu Hause verbessern. Die Grundidee besteht darin, an verschiedenen Stellen in der eigenen Wohnung NFC-Tags zu installieren, die bei Berührung unterschiedliche Aktionen ausführen können. Oft hat man sein Smartphone bereits am Mann, sodass keine speziellen Vorbereitungen nötig sind. In den Abbildungen 5.20 bis 5.22 sind mögliche Kontaktpunkte abgebildet.

Die Abmessungen der Tags sind sehr klein, dadurch kann man diese meist unsichtbar in der eigenen Wohnung integrieren. Zumal man selbst das System installiert, braucht man keine auffälligen Icons oder Symbole, die einen auf den jeweiligen Tag hinweisen.

(1) Wohnzimmertisch Hier könnte beispielsweise eine Verknüpfung zur Hausautomation liegen. Legt man sein Smartphone an eine bestimmte Stelle auf den Tisch, hat man die Möglichkeit, die Musik der Anlage zu wechseln und die Lautstärke zu kontrollieren. An anderer Stelle könnte die Kontrolle über die Lichter im Zimmer gestartet werden. Voraussetzungen sind natürlich existierende Systeme in der Wohnung.

Abbildung 5.20: Mögliche HomeField-Tags im Wohnzimmer

(2) Kühlschrank Beim Blick in den Kühlschrank fällt auf, dass die Milch bald leer ist. Beim Kontakt mit dem Tag öffnet sich automatisch die Einkaufszettel-App und man kann benötigte Einkäufe eintragen.

(3) Küche In guter Position in der Küche eignet sich ein Kontaktpunkt dazu, einen Timer starten zu können. Oft muss es beim Kochen schnell gehen und man ist gerade im Stress. Wenn man jetzt das Smartphone an den Punkt legt, braucht man nur noch die Zeit eingeben und schon läuft die Stoppuhr. Man könnte auch eine Reihe von Icons anbringen, die oft genutzte Zeiten repräsentieren. Diese starten den Timer dann mit einer bestimmten Zeit und so muss der Nutzer gar nicht mehr mit dem Gerät interagieren.

(4) Flur Im Eingangsbereich der Wohnung könnte ein Tag angebracht werden, der bei Berührung Dienste auf dem Smartphone aktiviert oder deaktiviert. So könnte man beim Verlassen der Wohnung das WLAN-Modul ausschalten und das GPS aktivieren oder die gegenteiligen Funktionen beim Betreten. So kann das Smartphone in jedem Bereich optimal genutzt und Strom gespart werden.

Abbildung 5.21: Mögliche HomeField-Tags im Schlafzimmer

(5) Sofa Oft liegt die Fernbedienung an einer Stelle, an der man sie nicht wiederfindet oder man muss aufstehen, um sie zu holen. Ein NFC-Tag könnte unsichtbar in die Armlehnen eines Sofas integriert werden, woraufhin sich bei Kontakt die Steuerung für das Fernsehgerät öffnet. Optional könnte man auch noch technische Parameter übertragen, um beispielsweise die Verbindung (von Smartphone zu Fernseher oder Spielekonsole) über Bluetooth zu vereinfachen und zu beschleunigen.

(6) Schlafzimmer Auf dem Nachttisch kann ein NFC-Tag integriert werden. Legt man sein Smartphone abends auf den Nachttisch, wird der Flugzeugmodus aktiviert. Dadurch ist man zum Einen während des Schlafs nicht mehr der Strahlung ausgesetzt, zum Anderen wird Strom gespart.

(7) Bad Neben der Toilette ist ein Tag platziert, der das aktuelle Lieblingsspiel startet.

Ein großer Vorteil von HomeField ist die einfache Installation. Tags können mittlerweile kostengünstig erworben werden, die Preise liegen ca. zwischen 1€ und 2€ pro Tag[7,8]. Mit

[7] `http://www.nfc-tag-shop.de/nfc-tag-mifare-classic-10`, abgerufen am 28.06.2012
[8] `http://www.nfc-tag.de/shop/15-nfc-tags-ntag203-25mm-164-byte/`, abgerufen am 28.06.2012

Abbildung 5.22: Mögliche HomeField-Tags im Bad

der entsprechenden Applikation können die einzelnen Aktionen auf einem Tag gespeichert werden. Dieser muss dann nur noch in der Wohnung untergebracht werden.

Umfrage und Evaluation

Einen Usability-Test zu dieser Anwendung durchzuführen stellt sich als sehr schwierig heraus. Viele der Vorteile werden nur durch die häufige Nutzung im Alltag klar und können nur dann richtig eingeschätzt werden. Die Machbarkeit der oben aufgelisteten Funktionalitäten wurde allerdings getestet. Ist einmal die Möglichkeit programmiert worden eine spezielle App bei Kontakt mit einem Tag zu öffnen, kann der Nutzer das Smartphone für das Öffnen jeglicher Apps konfigurieren. Spezielle Telefonfunktionen hingegen, wie das Ein- und Ausschalten des WLAN-Moduls, können nicht um weitere Funktionalitäten erweitert werden, ohne die Basisversion der HomeField-App[9] zu verändern. Möchte ein Nutzer beispielsweise in Zukunft sein Bluetooth-Modul aktivieren sobald er einen speziellen Tag berührt, muss diese Funktion zuvor innerhalb der HomeField-App integriert worden sein.

[9] Die HomeField-App bietet eine Oberfläche um neue Tags zu konfigurieren und übernimmt die Aufgabenverteilung beim Lesen eines Tags.

Um zumindest ein Gefühl für die Anwendung zu bekommen, wurde die Armlehne eines Sofas mit einem Tag versehen, mit dem eine Applikation zur Steuerung des Fernsehergerätes geöffnet werden konnte. Zudem wurde den Befragten die oben abgebildeten Zeichnungen gezeigt, NFC und das Konzept von HomeField erklärt. Die Befragten waren zwischen 24 und 35 Jahren alt und kamen zum überwiegenden Teil aus der technischen Branche. Grundsätzlich kann festgehalten werden, dass alle Befragten positiv auf die Anwendung reagiert haben. 82% haben mittleres bis starkes Interesse bekundet, diese Technologie selbst im eigenen Heim zu installieren. Auch kamen spontane Ideen, für welche Funktionen sie die Tags zusätzlich zu den oben genannten installieren würden:

- Öffnen des zuletzt gelesenden Buches, sowohl im Wohn-, als auch im Schlafzimmer.
- Öffnen der aktuellen Fernsehzeitschrift auf dem Sofa.
- Im Schlafzimmer der Wecker automatisch aktivieren.
- Akkuprüfung beim Betreten der Wohnung.
- Legt oder stellt man das Smartphone auf die Kommode, soll automatisch eine Diashow gestartet werden.
- Hinter Fotos von Freunden und Familie ist jeweils ein Tag mit Kontaktinformationen zu dieser Person und man kann diese direkt anrufen.
- In Speisekarten von Lieferdiensten soll ein Tag integriert sein, über den man direkt in eine App kommt um die Speisen zu bestellen.
- Weitere Verknüpfungen zur bestehenden Hausautomation, wie das Steuern von Jalousien, dem Aktivieren der Alarmanlage und dem Starten der Kaffeemaschine.

Diese Vielzahl an weiteren Ideen zeigt, dass die Befragten sich direkt mit der Technologie anfreunden konnten. Es ist ein realistischer Anwendungsfall, den sich alle Befragten vorstellen konnten und den sie einfach auf das persönliche Umfeld übertragen konnten.

Auffällig ist die hohe Zahlungsbereitschaft der Befragten (Abbildung 5.23). 36% der potenziellen Nutzer können sich vorstellen, über 50 € für die Installation der Tags zu bezahlen. Nimmt man einen Preis von 1 € pro Tag und rechnet pauschal 5 € für die HomeField-Applikation, kann man schon für 15 € HomeField mit 10 Kontaktpunkten bei sich zu Hause integrieren. Somit wären 81,8% der Befragten mit den Kosten einverstanden und dieser Faktor wäre für den Erfolg einer solchen Applikation nebensächlich, beziehungsweise würde er noch positiv auf eine Kaufentscheidung einwirken.

Abschließend gaben 77% an, beim nächsten Kauf eines Smartphones auf NFC zu achten. Da kein Vergleichswert vorliegt, kann man nicht sagen, ob die HomeField-Anwendung bei den Befragten dazu beigetragen hat. Trotzdem zeigt diese Zahl, dass die Nutzer für NFC sensibilisiert werden können, was im Umkehrschluss zu einer höheren Verfügbarkeit von NFC in der Zukunft beiträgt.

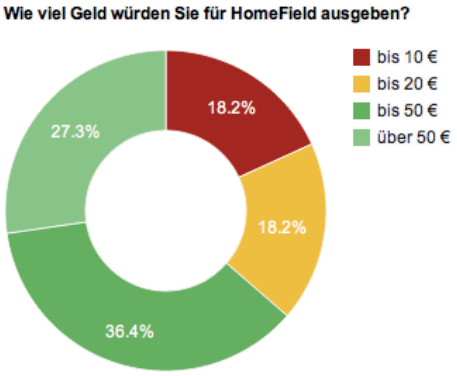

Abbildung 5.23: Zahlungsbereitschaft der Befragten

Kapitel 6

Ergebnisse und Ausblick

Ziel dieser Arbeit war es unter anderem, Entwicklern die Nutzung von NFC zu erleichtern. Zwar gibt es bisher eine Fülle an verschiedensten Bibliotheken rund um die Technologie NFC, die sich dieses Ziel ebenfalls gesetzt haben, die wenigsten davon erkennen und behandeln jedoch die wichtigsten Aspekte.

Lediglich EasyNFC besticht durch ein gut durchdachtes Konzept, beinhaltet aber auch nicht alle Möglichkeiten. Zwar werden bestehende Unzulänglichkeiten, wie die Abhängigkeit von Lebenszyklen, erkannt und behandelt, jedoch beschränkt sich das Konzept auf die reine Kapselung von bestehenden Funktionalitäten. Dadurch wird ganz klar abgegrenzt: EasyNFC gestaltet die NFC-Schnittstelle um, erweitert sie aber nicht. Probleme, wie ein fehlender Kontextbezug von NFC-Nachrichten zu einer verantwortlichen Instanz, werden von dem Framework nicht gelöst. Hier werden Entwickler weiterhin sich selbst überlassen.

Sie müssen sich in technische Spezifikationen einlesen und an mehreren Stellen, sowohl mittels Programmier- als auch Auszeichnungssprache, Funktionalität einpflegen, die ebensogut ein Framework übernehmen kann. EasyNFC konzentriert sich auf die Kernfunktionalität und richtet sich bei der Wahl der Klientelen eher an erfahrene Entwickler.

SimpleNFC stellt an sich selbst den Anspruch von Entwicklern aller Erfahrungsstufen genutzt werden zu können. Es soll sowohl von Anfängern mit wenig Erfahrung in der Programmierung unter Android, als auch von Experten der Java-Entwicklung gleichermaßen genutzt werden können. Dazu unterstützt SimpleNFC Anfänger durch Methoden, die technische Spezifikationen verbergen. Experten können weiterhin Methoden nutzen, die mittels umfangreichen Parametern beliebig konfiguriert werden können.

Ähnlich wie EasyNFC löst auch SimpleNFC das Problem mit der Abhängigkeit des Lifecycles, sodass NFC-Nachrichten in jedem Zustand der Activity geschrieben werden können. Auch der abstraktere Aufbau und die Definition von Anwendungsszenarien haben beide Frameworks gemeinsam.

Darüber hinaus bietet SimpleNFC jedoch noch weitere Funktionalität, die in EasyNFC nicht enthalten ist. Eines der größten Probleme unter Android stellt die aufwändig zu implementierende Zugehörigkeit einzelner NFC-Nachrichten zu einzelnen Activities dar. Dieses Problem löst SimpleNFC, indem es die Möglichkeit bereitstellt, Activities als verantwortliche Instanz für ausgewählte NFC-Nachrichten registrieren zu können. Dies ermöglicht Entwick-

lern eine gezielte Weitergabe von NFC-Nachrichten an die dafür vorgesehenen Klassen. Ebenso wichtig wie die Funktionen selbst ist die Art des Zugriffs. Java-Entwickler sind bestimmte Vorgehensweisen und Entwurfsmuster gewöhnt. So werden die verantwortlichen Activities in der Art des Observer-Pattern registriert. Schnittstellen dieser Art werden von Entwicklern sofort verstanden, ohne dass eine Einarbeitung nötig ist.

Auch die Erstellung und Verarbeitung von NFC-Nachrichten wird durch SimpleNFC erheblich vereinfacht. Zwar wurde an dem Konzept von Message und Record festgehalten, die protokolltypischen Konfigurationen wurden jedoch durch eine Kapselung in abstraktere Klassen ausgeblendet. Entwickler können auf ein Set von verschiedenen Record-Typen zugreifen, die Inhalt in Form von Text, byte-Arrays und Hashtabellen abspeichern können. Bisher war es nur über Umwege möglich Benachrichtigungen über das Ergebnis einer NFC-Kommunikation zu erlangen. SimpleNFC bietet für alle Arten der NFC-Kommunikation ereignisorientierte Rückmeldungen, die von Listenern empfangen werden können. Rückmeldungen enthalten dabei auch Fehlermeldungen. Mit SimpleNFC ist es somit erstmals möglich gezielt auftretende Exceptions zu behandeln.

Für die Entwicklung von Applikationen, die auf verschiedenen Betriebssystemversionen laufen sollen, bietet SimpleNFC die Möglichkeit zur Laufzeit gezielt die nötigen Informationen über vorhandene oder fehlende NFC-Funktionalität abzurufen. Auf diese Weise können Entwickler eine Applikation gleichzeitig für mehrere Android-Versionen programmieren und von den Vorteilen neuerer Betriebssystemversionen profitieren. Als Ergebnis aus dieser Arbeit geht ein Framework hervor, welches das Entwickeln mit NFC dem gewohnten Programmierparadigma anpasst und dadurch stark vereinfacht.[1]

Sowohl die Umfrage wie auch die Auswertung der Versuche mit den Prototypen haben gezeigt, dass die Nutzer bereit für NFC sind und ein starkes Interesse daran zeigen. Somit steht der Verbreitung von NFC aus Sicht der Nutzer nicht viel im Weg. Nichts desto trotz ist es wichtig, auf einige Kernelemente zu achten, wenn man eine NFC-Anwendung entwickeln möchte.

Nutzer gewöhnen sich schnell an Technologien, wenn sie sie einmal in Benutzung haben. Dieser Faktor ist für NFC sowohl positiv wie negativ zu betrachten. Haben die Nutzer in einer Anwendung eine Alternative, die ihnen bekannt vorkommt, werden sie diese mit großer Wahrscheinlichkeit nutzen, auch wenn die neue Technik wohlmöglich schneller und einfacher zu bedienen ist. So haben viele der Nutzer im "-Prototypen zunächst versucht, die Anwendung durch einen QR-Code zu starten, weil sie diesen kannten. Vor allem in der aktuellen Phase der NFC-Verbreitung ist es wichtig, dem Nutzer möglichst viele Hinweise zu geben und ihn an möglichst vielen Stellen auf NFC aufmerksam zu machen.

Glücklicherweise ist NFC schnell erklärt und die Anwendung zudem sehr einfach. Es reicht, wenn der Nutzer sein Smartphone zwei Mal mit dem Rücken gegen ein NFC-Tag hält. Er lernt dann sehr schnell, wie sein Gerät funktioniert und bei welchen Zentimeter-Abständen es zu einer Verbindung kommt.

Es hat sich als sinnvoll herausgestellt, ein einheitliches Logo zu nutzen, damit Nutzer sofort eine NFC-Applikation erkennen können. Die Autoren raten das vom NFC-Forum

[1] SimpleNFC kann unter `https://github.com/Tarquinio/SimpleNFC` runtergeladen werden.

entwickelte N-Mark zu nutzen. Zudem kann es gerade in der Anfangsphase helfen, mit Piktogrammen das Verständnis von NFC zu fördern.

Damit auch sicherheitsrelevante Szenarien genutzt werden, müssen die Nutzer bereits Gefallen an weniger kritischen NFC-Anwendungen gefunden haben. Nur eine starke Bindung zu der Technik schafft das benötigte Vertrauen.

Das Grundproblem für die Verbreitung von NFC bleibt jedoch bestehen: NFC löst keine Probleme, die den Nutzern bekannt sind. Durch diese Begebenheit existiert noch keine Nachfrage nach NFC-Geräten und Anwendungen. Auch deswegen ist die Verbreitung so langsam vorangeschritten, sieht man das Alter der Technologie im Vergleich.

In vielen der bearbeiteten Studien ist eine Referenz auf eine Statistik von ABI Research [35] aus dem Jahr 2007 zu finden. Dort wurde damit gerechnet, dass 2012 bereits 292 Millionen mobile Geräte mit NFC ausgestattet sind. 2011 waren es tatsächlich gerade einmal 35 Millionen [14]. Auch wenn alle Zeichen darauf zeigen, dass NFC bald eine Technologie für die Massen sein wird, kann man das aktuell noch nicht sicher behaupten.

In den letzten Tagen vor Abgabe dieser Arbeit haben sich auch die Gerüchte um eine Integration von NFC in der nächsten Generation des iPhones verschärft [2,3]. Da Apple einen großen Teil der Smartphone-Branche beherrscht, würden sich mit einem solchen Schritt die Chancen vervielfachen, dass NFC schnell in unseren Alltag Einzug hält.

Durch das entwickelte Framework SimpleNFC hoffen die Autoren, dass sich mehr Entwickler davon überzeugen lassen, NFC in die eigenen Applikationen zu integrieren. Mit den Empfehlungen für die Gestaltung von NFC-Anwendungen sollen die Nutzer schneller angesprochen werden. Durch diese Maßnahmen kann die Verbreitung von NFC beschleunigt werden.

[2] http://computer.t-online.de/iphone-5-apple-smartphone-unterstuetzt-wahrscheinlich-nfc/id_57507356/index?news, abgerufen am 28.06.2012

[3] http://9to5mac.com/2012/06/25/new-iphone-prototypes-have-nfc-chips-and-antenna/, abgerufen am 28.06.2012

Kapitel 7

Zusammenfassung

Near Field Communication (zu deutsch -Kommunikation“, Abkürzung NFC), ist ein Übertragungsstandard der kontaktlosen Datenübertragung. Eine Datenübertragung findet auf Distanzen kleiner 4 cm statt. Die Übertragungsgeschwindigkeit ist mit 424kbit/s relativ gering. Daher eignet sich NFC dazu, kleine Nachrichten schnell zwischen zwei Teilnehmern auszutauschen - ohne zunächst eine Verbindung aufbauen zu müssen. Dies unterscheidet NFC von bestehenden Technologien: Bei Bluetooth etwa muss eine Verbindung zweier Geräte manuell aufgebaut und beidseitig bestätigt werden. Um als NFC Nutzer dennoch größere Datenmengen teilen zu können, kann automatisch eine schnellere Alternativverbindung über beispielsweise Bluetooth oder WLAN aufgebaut werden, ohne Eigenschaften dieser Verbindung manuell konfigurieren zu müssen.

Technisch baut NFC auf Radio Frequenzy Identification (Abkürzung RFID) auf und nutzt elektromagnetische Wellen zur Übertragung der Daten. Obwohl NFC bereits vor einem Jahrzehnt entwickelt wurde, ist diese Technologie im Jahr 2012 noch immer nicht im Massenmarkt angekommen. Die Entwicklungen der letzten Jahre in Verbindung mit mobilen Geräten, wie Smartphones, lassen jedoch vermehrt neue Nutzerszenarien entstehen.

In dieser Arbeit wurde untersucht, weshalb sich NFC bisher nicht durchsetzten konnte und welche Wege es gibt, die Verbreitung der Technologie zu beschleunigen. Zwei wesentliche Aspekte konnten herausgestellt werden: Auf der einen Seite müssen Software-Entwicklern Werkzeuge bereitgestellt werden, die eine unkomplizierte Integration von NFC-Funktionalitäten in Anwendungen ermöglichen. Auf der anderen Seite muss NFC für Nutzer so attraktiv gestaltet werden, dass sie die Technologie im Alltag einsetzen möchten. Um diesen Herausforderungen begegnen zu können, wurde untersucht, welche Szenarien für NFC-Nutzer im Alltag interessant sind und welche Hilfestellung sie benötigen, um mit der entsprechenden Anwendung umgehen zu können.

Seit 2011 werden verschiedene Smartphones mit NFC-Antenne ausgestattet. Das aktuell einzige Betriebssystem, dass NFC unterstützt ist Android, weshalb in dieser Arbeit alle technischen Entwicklungen auf Android begrenzt sind. Microsoft führt mit der neuen Version seines mobilen Betriebssystems, Windows Phone 8, die Unterstützung von NFC ebenfalls ein. Gerüchten zufolge wird dann auch Apple mit der nächsten iPhone-Generation Mitte 2012 dem Trend folgen.

Nach der Untersuchung bestehender Lösungen, die das Entwickeln für NFC vereinfachen sollen, wurde im Rahmen dieser Arbeit SimpleNFC als Framework entwickelt. Zu den in Betracht gezogenen Frameworks hat sich neben Open NFC, libnfc, AllJoyn, nfctools, nfcpy und CocoaTag vor allem das EasyNFC-Framework der Standford University als interessant herausgestellt. Doch auch bei EasyNFC wurde nicht in allen Aspekten darauf geachtet, es dem Entwickler so einfach wie möglich zu machen. Folgende Kritikpunkte sind besonders schwerwiegend: EasyNFC ermöglicht keine Kompatibilität zu älteren Android Versionen. Bereits bestehende Funktionen der NFC-Schnittstelle von Android wurden lediglich gekapselt, es wurden jedoch keine neuen Methoden geschrieben. Ebenso wenig wurde das Erstellen von NFC-Nachrichten vereinfacht. Der Entwickler muss auch zur Nutzung von EasyNFC eine hohe Kenntnis über die technischen Anforderungen von NFC unter Android kennen.

SimpleNFC hingegen soll dem Entwickler ermöglichen, die Nahfeld-Kommunikation in seine Applikationen zu integrieren ohne dabei über technisches NFC-Wissen verfügen zu müssen. Charakteristisch für SimpleNFC ist eine einfache Benutzung sowie eine abstraktere Sicht der Dinge. Spezifische Anwendungszenarien wurden in eigenen Methoden gekapselt. Das Beschreiben eines Tags beispielsweise ist somit mit nur einer Zeile Code möglich. Ebenso einfach ist das Erstellen von NFC-Nachrichten, die an die eigene Applikation gebunden werden und einer verantwortliche Instanz zugewiesen werden.

In einer Feldumfrage wurde das Wissen und die Meinung möglicher Anwender bezüglich NFC erhoben. Außerdem wurden zwei verschiedene Prototypen entwickelt, an denen exemplarisch untersucht wurde, welche Szenarien für den Anwender interessant sein können. Zusammenfassend kann festgehalten werden, dass der Großteil aller Befragten grundsätzlich Interesse an NFC hat. Zwar sind die Befragten bei Anwendungen mit sicherheitsrelevanten Aspekten deutlich zurückhaltender, man kann jedoch davon ausgehen, dass mit der Nutzung das Vertrauen steigt.

Für einige Befragte war diese Erhebung die erste Berührung mit der NFC-Technologie. Auch wenn keine Vorkenntnisse vorhanden waren, haben alle Befragten das Thema schnell und richtig verstanden. Auch in den Praxistests waren die Nutzer zwar zunächst unsicher, wie genau sie mit einem NFC-Tag umgehen müssen, konnten Gezeigtes aber sehr schnell umsetzen. NFC zeigte sich demnach als einfach vermittelbar und sollte so in relativ kurzer Zeit einen hohen Bekannt- und Beliebtheitsgrad erreichen können.

Die größte Hürde bei der Verbreitung von NFC besteht darin, dass die Nutzer keine Probleme haben, für die sie NFC als Lösung unbedingt benötigen. NFC ermöglicht zwar neue Nutzerszenarien, da sie nun praktikabel werden - technisch gesehen waren aber nahezu alle Anwendungen auch vorher umsetzbar. Durch NFC sind diese nun bequemer, schneller und sicherer geworden, so dass viele Aktivitäten im Alltag mit Hilfe von NFC optimiert werden könnten. Diese Vorteile lernen die Nutzer erst kennen, wenn sie das erste Mal aktiv mit NFC in Berührung kommen und die Möglichkeit haben, diese auch auszunutzen. Bis jetzt ist der Markt mobiler Endgeräten mit NFC-Chip relativ klein. Zwar wird sich das sicher dieses Jahr ändern, da bereits etliche Hersteller Smartphones mit NFC-Ausstattung angekündigt haben - wie groß der Prozentanteil der verkauften NFC-Smartphones am Ende sein wird, kann man aktuell noch nicht sicher voraussagen.

Damit befindet sich die Technologie in einem Teufelskreis (Abbildung 7.1). Die Nutzer haben kein Interesse an NFC, da ihnen die Möglichkeiten nicht bewusst sind, beziehungs-

weise sie die Technik noch nicht ausreichend kennen. Dadurch sehen die Hersteller von Smartphones keine Nachfrage für entsprechend ausgestattete Geräte. Sie entscheiden sich deshalb aus Kostengründen dafür, weiterhin Smartphones auf den Markt zu bringen, die keine NFC-Antennen besitzen. Daraus ergibt sich für die Entwickler kein Anreiz, sich mit der Technik auseinander zu setzen - zu mal die Programmierung ohne Framework kompliziert ist. Das wiederum mündet in der Tatsache, dass die Nutzer nicht in Berührung mit NFC kommen und so keine Nachfrage aufgebaut werden kann.

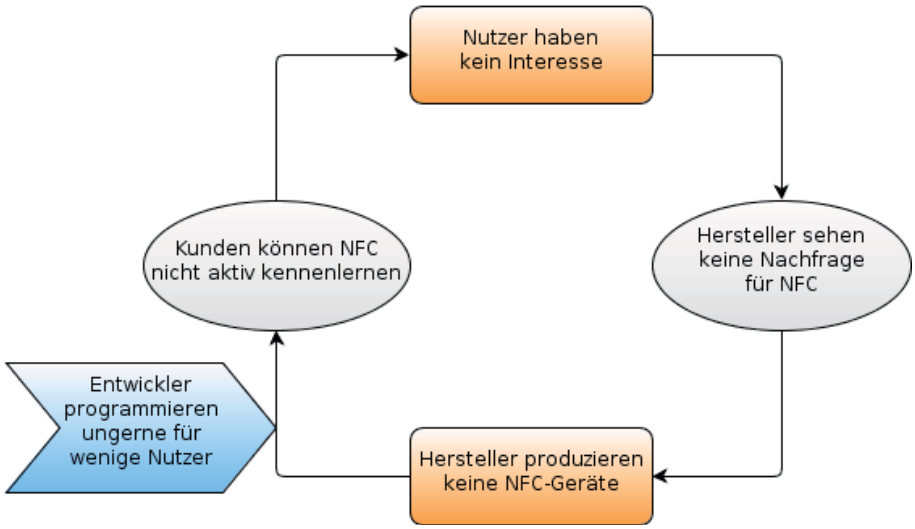

Abbildung 7.1: Der Teufelskreis von NFC

Man kann hoffen, dass durch die einflussreiche Bankenlobby genügend Druck auf die Hersteller ausgeübt wird NFC in den Geräten zu integrieren. Diese erhoffen sich mit NFC den Durchbruch der mobilen Zahlung zu erwirken, denn viele der laufenden und geplanten Großprojekte wurden von Banken oder Kreditinstituten initiiert.

Die Autoren dieser Arbeit hoffen, mit SimpleNFC die Verbreitung der Nahfeld-Kommunikation fördern zu können. Das Framework wird es einer breiten Masse an Entwicklern ermöglichen, bestimmte NFC-Funktionalitäten ohne großen Aufwand und Kenntnis der Datenstrukturen zu nutzen. Mit der ausführlichen Dokumentation und den einfachen Methoden sollen es Entwickler so einfach wie möglich haben. Hürden für einen Einstieg in NFC-Programmierung werden niedriger, so dass am Ende eine größere Menge und Bandbreite an Anwendungen mit NFC am Markt verfügbar sein werden und so die Verbreitung beschleunigt wird.

Desweiteren empfehlen die Autoren das vom NFC-Forum definierte N-Mark zu nutzen, um ebenso Applikationen mit NFC-Unterstützung zu kennzeichnen, wie auch Tags zu markieren. Dadurch können Nutzer NFC-Anwendungen schnell erkennen und nutzen. Zudem werden mehr Menschen darauf aufmerksam, wenn es global als Zeichen eingesetzt wird. Sobald der Nutzer NFC einmalig benutzt hat, sind ihm sowohl die praktische Anwendungen als auch die Möglichkeiten bekannt. Es ist also wichtig, neuen Nutzern, die selbst NFC noch nie genutzt haben, eine Erklärung zu geben und durch Piktrogramme das Verständnis zu

erleichtern. Diese Erläuterung kann sehr kurz sein, da NFC in wenigen Worten beschrieben werden kann.

Die Autoren hoffen, dass NFC schon bald Einzug in den deutschen Alltag hält, denn die Nahfeld-Kommunikation wird viele Dinge des alltäglichen Lebens weiter vereinfachen.

Anhang A

CD mit Quellcode und Dokumentation

Auf der beiliegenden CD befindet sich:

- SimpleNFC als Android-Library-Project.
- Ein Beispiel-Projekt, dass die Funktionalität von SimpleNFC zeigt (NFCLabs).
- Die Dokumentation von SimpleNFC.
- Die Rohdaten der Feldumfrage im .pdf-Format.
- Diese Arbeit im .pdf-Format.

Literaturverzeichnis

[1] Norm Standard ECMA-340: *Near Field Communication Interface and Protocol (NFCIP-1)*, 2004.

[2] NFC Forum: *NFC in Public Transport*, 2011.

[3] Haselsteiner, Ernst und Klemens Breitfuß: *Security in Near Field Communication (NFC) Strengths and Weaknesses*. In: *Workshop on RFID Security, RFIDSec 2006*, Graz, Austria, 2006.

[4] Statistisches Bundesamt Deutschland: *Statistisches Bundesamt Deutschland - GENESIS-Online*, 2010. `https://www-genesis.destatis.de/genesis/online`, besucht: 10.02.2012.

[5] Langer, J. und M. Roland: *Anwendungen und Technik von Near Field Communication (NFC)*. Springer-Verlag, Berlin Heidelberg, 2010.

[6] Kern, Christian: *Anwendung von RFID-Systemen*. Springer-Verlag, Berlin Heidelberg, 2006.

[7] Dillinger, O., G. Madlmayr, C. Schaffer und J. Langer: *An Approach to NFC's Mode Switch*. In: *FH Science Day: Proceedings, 25th October 2006 ; Workshops: How Emotions Influence Decisions and Behavior, Technical Applications in Medicine, Logistics and Supply Chain Management, Operations Management, Secure and Embedded Systems*. Dr. Jürgen Ecker, Shaker, 2006.

[8] Schwan, Ben: *NFC für Bezahlsysteme auf dem Vormarsch*, 2009. `http://heise.de/-752673`, besucht: 22.04.2012.

[9] Rogers, Everett: *Diffusion of innovations*. Free Press, New York, 1962.

[10] Rogers, Everett M: *Diffusion of Innovations*. Free Press, New York, 5. Auflage, 2003.

[11] Boileau, Roderic: *Low Level versus High Level APIs*, 2008.

[12] NFC Forum: *NFC Data Exchange Format (NDEF)*, 2006.

[13] NFC Forum: *Connection Handover*, 2010.

[14] Rogers, Wes: *IMS Research - Electronics market research & consultancy*, 2011. http://imsresearch.com/news-events/press-template.php?pr_id=2469, besucht: 22.02.2012.

[15] Burghardt, Benjamin, Markus Huber und Rainer Steffen: *Near Field Communication*. Technischer Bericht, BMW Forschung und Technik GmbH, DB Mobility Logistics AG, METRO AG, Nokia GmbH, Rhein-Main-Verkehrsverbund GmbH, Frankfurt, 2010.

[16] Sawall, Achim: *Bezahlen per Funkchip: Sparkassen kommen Google zuvor*, 2011. http://www.handelsblatt.com/technologie/it-tk/mobile-welt/bezahlen-per-funkchip-sparkassen-kommen-google-zuvor/4599778.html, besucht: 20.02.2012.

[17] Sawall, Achim: *Schneller Bezahlen: 45 Millionen EC-Karten der Sparkasse bekommen NFC-Chips*, 2011. http://www.golem.de/1106/84315.html, besucht: 20.01.2012.

[18] NFC News: *Samsung, Visa to showcase NFC payments with London 2012 athletes*, 2012. http://www.nfcnews.com/2012/05/10/samsung-visa-to-showcase-nfc-payments-with-london-2012-athletes, besucht: 13.06.2012.

[19] Gann, Eleanor Orebi: *Samsung and Visa Showcase Mobile Payments at the London 2012 Olympic and Paralympic Games*. Technischer Bericht, Visa Europe, 2012.

[20] Clark, Sarah: *Barcelona, Hamburg, Rio and Derry to get NFC information services*, 2012. http://www.nfcworld.com/2012/05/23/315886/barcelona-hamburg-rio-derry-get-nfc-information-services/, besucht: 14.06.2012.

[21] Ceipidor, U. Biader, C. M. Medaglia, A. Opromolla, V. Volpi, A. Moroni und S. Sposato: *A Survey about User Experience Improvement in Mobile Proximity Payment*. 2012 4th International Workshop on Near Field Communication, Seiten 51–56, März 2012.

[22] Kneissl, Fabian, Richard Rottger, Uwe Sandner, Jan Marco Leimeister und Helmut Krcmar: *All-I-Touch as Combination of NFC and Lifestyle*. In: *2009 First International Workshop on Near Field Communication*, Seiten 51–55. Ieee, Februar 2009.

[23] Steffen, Rainer, Jörg Preiß inger, Tobias Schöllermann, Armin Müller und Ingo Schnabel: *Near Field Communication (NFC) in an Automotive Environment*. In: *2010 Second International Workshop on Near Field Communication*, Nummer 01, Seiten 15–20. Ieee, 2010.

[24] Riekki, Jukka, Ivan Sanchez und Mikko Pyykkonen: *NFC-Based User Interfaces*. In: *2012 4th International Workshop on Near Field Communication*, Seiten 3–9. Ieee, März 2012.

[25] Sanchez, Ivan, Marta Cortes, Jukka Riekki und Mikko Pyykkonen: *NFC-Based Physical User Interfaces for Interactive Spaces.* In: *Near Field Communications Handbook*. Syed A. Ahson and Mohammad Ilyas, CRC Press, volume 13 Auflage, 2011.

[26] Pyykkönen, Mikko, Jukka Riekki, Ismo Alakärppä, Ivan Sanchez, Marta Cortes und Sonja Saukkonen: *Designing Tangible User Interfaces for NFC Phones*. Advances in Human-Computer Interaction, 2012:1–12, 2012, ISSN 1687-5893.

[27] Arnall, Timo: *A graphic language for touch-based interactions*. In: *Proceedings of Mobile Interaction with the Real World*, Seiten 18–22, 2006.

[28] Välkkynen, Pasi, Timo Tuomitsto und Ilkka Korhonen: *Suggestions for visualising physical hyperlinks*. In: *Pervasive 2006*, Dublin, Ireland, 2006.

[29] Hang, Alina, Gregor Broll und Alexander Wiethoff: *Visual Design of Physical User Interfaces for NFC-based Mobile Interaction*. In: *8th ACM Conference on Designing Interactive Systems*, DIS 2010, August 16-20, 2010, Aarhus Denmark, 2012.

[30] Paper, Draft und Rfid systems July: *AIM Frequency Forums AIM FF 2000 : 001*. Forum American Bar Association, Seiten Seiten 1–25, 2000.

[31] Rankl, Wolfgang und Wolfgang Effing: *Handbuch der Chipkarten*. Carl Hanser Verlag, München, Wien, 3. Auflage, 1999.

[32] Finkenzeller, Klaus: *RFID-Handbuch*. Hanser, 3., aktual Auflage, 2002.

[33] Choudhary, B. und J. Risikko: *Mobile Device Security Element*. Key Findings from Technical Analysis, 1:1–8, 2005.

[34] Nielsen, Jakob: *Why You Only Need to Test with 5 Users*, 2000. `http://www.useit.com/alertbox/20000319.html`, besucht: 31.05.2012.

[35] ABI Research: *Twenty Percent of Mobile Handsets Will Include Near Field Communication by 2012*, 2007. `http://www.abiresearch.com/press/838`.